O'Mega:
Optimal Monte-Carlo
Event Generation Amplitudes

Thorsten Ohl*

Institut für Theoretische Physik und Astrophysik
Julius-Maximilians-Universität Würzburg
Emil-Hilb-Weg 22, 97074 Würzburg, Germany

Jürgen Reuter†

DESY Theory Group, Notkestr. 85, 22603 Hamburg, Germany

Wolfgang Kilian[c,‡]

Theoretische Physik 1
Universität Siegen
Walter-Flex-Str. 3, 57068 Siegen, Germany

with contributions from Christian Speckner[d,§]
as well as Christian Schwinn et al.

unpublished draft, printed 02/06/2015, 18:51

Abstract

...

*ohl@physik.uni-wuerzburg.de, http://physik.uni-wuerzburg.de/ohl

†juergen.reuter@desy.de

‡kilian@physik.uni-siegen.de

§cnspeckn@googlemail.com

Revision Control

Contents

1 Introduction 1
- 1.1 *Complexity* 1
- 1.2 *Ancestors* 2
- 1.3 *Architecture* 2
  - 1.3.1 *General purpose libraries* 2
  - 1.3.2 *O'Mega* 2
  - 1.3.3 *Abstract interfaces* 3
  - 1.3.4 *Models* 3
  - 1.3.5 *Targets* 4
  - 1.3.6 *Applications* 4
- 1.4 *The Big To Do Lists* 4
  - 1.4.1 *Required* 4
  - 1.4.2 *Useful* 4
  - 1.4.3 *Future Features* 4
  - 1.4.4 *Science Fiction* 5

2 Tuples and Polytuples 6
- 2.1 *Interface of Tuple* 6
- 2.2 *Implementation of Tuple* 9
  - 2.2.1 *Typesafe Combinatorics* 10
  - 2.2.2 *Pairs* 10
  - 2.2.3 *Triples* 12
  - 2.2.4 *Pairs and Triples* 13
  - 2.2.5 *... and All The Rest* 15

3 Topologies 18
- 3.1 *Interface of Topology* 18
  - 3.1.1 *Diagnostics: Counting Diagrams and Factorizations for* $\sum_n \lambda_n \phi^n$ 19
  - 3.1.2 *Emulating HELAC* 20
- 3.2 *Implementation of Topology* 20
  - 3.2.1 *Factorizing Diagrams for* ϕ^3 21
  - 3.2.2 *Factorizing Diagrams for* $\sum_n \lambda_n \phi^n$ 24
  - 3.2.3 *Factorizing Diagrams for* ϕ^4 26
  - 3.2.4 *Factorizing Diagrams for* $\phi^3 + \phi^4$ 26
  - 3.2.5 *Diagnostics: Counting Diagrams and Factorizations for* $\sum_n \lambda_n \phi^n$ 27
  - 3.2.6 *Emulating HELAC* 33

4 DIRECTED ACYCLICAL GRAPHS 35
4.1 *Interface of DAG* . . . 35
4.1.1 *Forests* . . . 35
4.1.2 *DAGs* . . . 37
4.1.3 *Graded Sets, Forests & DAGs* . . . 39
4.2 *Implementation of DAG* . . . 40
4.2.1 *The Forest Functor* . . . 42
4.2.2 *Gradings* . . . 43
4.2.3 *The DAG Functor* . . . 45

5 MOMENTA 50
5.1 *Interface of Momentum* . . . 50
5.1.1 *Scattering Kinematics* . . . 52
5.1.2 *Decay Kinematics* . . . 53
5.2 *Implementation of Momentum* . . . 54
5.2.1 *Lists of Integers* . . . 55
5.2.2 *Bit Fiddlings* . . . 61
5.2.3 *Whizard* . . . 66
5.2.4 *Suggesting a Default Implementation* . . . 67

6 CASCADES 68
6.1 *Interface of Cascade_syntax* . . . 68
6.2 *Implementation of Cascade_syntax* . . . 68
6.3 *Lexer* . . . 70
6.4 *Parser* . . . 71
6.5 *Interface of Cascade* . . . 72
6.6 *Implementation of Cascade* . . . 73

7 COLOR 80
7.1 *Interface of Color* . . . 80
7.1.1 *Quantum Numbers* . . . 80
7.1.2 *Color Flows* . . . 80
7.2 *Implementation of Color* . . . 81
7.2.1 *Quantum Numbers* . . . 81
7.2.2 *Color Flows* . . . 83

8 FUSIONS 88
8.1 *Interface of Fusion* . . . 88
8.1.1 *Multiple Amplitudes* . . . 91
8.1.2 *Tags* . . . 93
8.2 *Implementation of Fusion* . . . 93
8.2.1 *Fermi Statistics* . . . 95
8.2.2 *Dirac Fermions* . . . 95
8.2.3 *Tags* . . . 97
8.2.4 *Tagged, the Fusion.Make Functor* . . . 98
8.2.5 *Fusions with Majorana Fermions* . . . 125
8.2.6 *Multiple Amplitudes* . . . 127

9 LORENTZ REPRESENTATIONS, COUPLINGS, MODELS AND TARGETS 135
9.1 Interface of Coupling 135
9.1.1 Propagators 135
9.1.2 Vertices 137
9.1.3 Gauge Couplings 140
9.1.4 $SU(2)$ Gauge Bosons 150
9.1.5 Quartic Couplings and Auxiliary Fields 152
9.1.6 Gravitinos and supersymmetric currents 154
9.1.7 Perturbative Quantum Gravity and Kaluza-Klein Interactions 154
9.1.8 Dependent Parameters 166
9.1.9 More Exotic Couplings 172
9.2 Interface of Model 172
9.2.1 General Quantum Field Theories 172
9.2.2 Mutable Quantum Field Theories 175
9.2.3 Gauge Field Theories 176
9.2.4 Gauge Field Theories with Broken Gauge Symmetries . 177
9.3 Interface of Target 178

10 CONSERVED QUANTUM NUMBERS 180
10.1 Interface of Charges 180
10.1.1 Abstract Type 180
10.1.2 Trivial Realisation 180
10.1.3 Nontrivial Realisations 180
10.2 Implementation of Charges 181

11 COLORIZATION 183
11.1 Interface of Colorize 183
11.1.1 183
11.2 Implementation of Colorize 183
11.2.1 Colorizing a Monochrome Model 183
11.2.2 Colorizing a Monochrome Gauge Model 207

12 PROCESSES 209
12.1 Interface of Process 209
12.2 Implementation of Process 210
12.2.1 Select Charge Conserving Processes 211
12.2.2 Parsing Process Descriptions 211
12.2.3 Remove Duplicate Final States 213

13 HARDCODED MODELS 217
13.1 Interface of Modeltools 217
13.1.1 Compilation 217
13.1.2 Mutable Models 217
13.2 Implementation of Modeltools 218
13.2.1 Compilation 218
13.2.2 Mutable Models 222
13.3 Interface of Modellib_SM 224
13.3.1 Hardcoded Models 224
13.4 Implementation of Modellib_SM 224

13.4.1 ϕ^3 . . . 225
13.4.2 $\lambda_3\phi^3 + \lambda_4\phi^4$ . . . 226
13.4.3 Quantum Electro Dynamics . . . 227
13.4.4 Quantum Chromo Dynamics . . . 230
13.4.5 Complete Minimal Standard Model (Unitarity Gauge) . . . 234
13.4.6 Complete Minimal Standard Model (including some extensions) . . . 236
13.4.7 Incomplete Standard Model in R_ξ Gauge . . . 265
13.4.8 Groves . . . 267
13.4.9 MSM With Cloned Families . . . 272
13.5 Interface of Modellib_BSM . . . 272
13.5.1 More Hardcoded BSM Models . . . 272
13.6 Implementation of Modellib_BSM . . . 273
13.6.1 Littlest Higgs Model . . . 273
13.6.2 Three-Site Higgsless Model . . . 395
13.6.3 2HDM with and without non-trivial flavor structure . . . 410
13.6.4 Complete Minimal Standard Model including additional Resonances . . . 428
13.7 Interface of Modellib_MSSM . . . 443
13.7.1 More Hardcoded Models . . . 443
13.8 Implementation of Modellib_MSSM . . . 444
13.8.1 Minimal Supersymmetric Standard Model . . . 444
13.9 Interface of Modellib_NMSSM . . . 495
13.9.1 Extended Supersymmetric Models . . . 495
13.10 Implementation of Modellib_NMSSM . . . 495
13.10.1 Next-to-Minimal Supersymmetric Standard Model . . . 495
13.11 Interface of Modellib_PSSSM . . . 525
13.11.1 Extended Supersymmetric Models . . . 525
13.12 Implementation of Modellib_PSSSM . . . 526
13.12.1 Extended Supersymmetric Standard Model(s) . . . 526

14 Comphep Models 564
14.1 Interface of Comphep_syntax . . . 564
14.2 Implementation of Comphep_syntax . . . 564
14.3 Lexer . . . 566
14.4 Parser . . . 567
14.5 Interface of Comphep . . . 568
14.6 Implementation of Comphep . . . 568

15 Hardcoded Targets 578
15.1 Interface of Targets . . . 578
15.1.1 Supported Targets . . . 578
15.1.2 Potential Targets . . . 578
15.2 Implementation of Targets . . . 578
15.2.1 O'Mega Virtual Machine with `Fortran 90/95` . . . 579
15.2.2 `Fortran 90/95` . . . 608
15.2.3 `C` . . . 678
15.2.4 O'Caml . . . 678
15.2.5 LaTeX . . . 678
15.3 Interface of Targets_Kmatrix . . . 678

15.4 Implementation of Targets_Kmatrix . . . 678

16 Phase Space 691
16.1 Interface of Phasespace . . . 691
16.2 Implementation of Phasespace . . . 692
16.2.1 Tools . . . 692
16.2.2 Phase Space Parameterization Trees . . . 692

17 Whizard 698
17.1 Interface of Whizard . . . 698
17.2 Implementation of Whizard . . . 698
17.2.1 Building Trees . . . 699
17.2.2 Merging Homomorphic Trees . . . 700
17.2.3 Printing Trees . . . 701
17.2.4 Process Dispatcher . . . 703
17.2.5 Makefile . . . 705

18 Applications 707
18.1 Sample . . . 707
18.2 Interface of Omega . . . 707
18.3 Implementation of Omega . . . 707
18.3.1 Main Program . . . 713
18.4 Implementation of Omega_QED . . . 718
18.5 Implementation of Omega_SM . . . 718
18.6 Implementation of Omega_SYM . . . 718

A Revision Control 726
A.1 Interface of RCS . . . 726
A.2 Implementation of RCS . . . 727

B Autotools 729
B.1 Interface of Config . . . 729
B.2 Implementation of Config . . . 729

C Textual Options 730
C.1 Interface of Options . . . 730
C.2 Implementation of Options . . . 730

D Progress Reports 732
D.1 Interface of Progress . . . 732
D.2 Implementation of Progress . . . 732

E More on Filenames 735
E.1 Interface of ThoFilename . . . 735
E.2 Implementation of ThoFilename . . . 735

F Cache Files 736
F.1 Interface of Cache . . . 736
F.2 Implementation of Cache . . . 737

G More On Lists 740
G.1 *Interface of ThoList* . . . 740
G.2 *Implementation of ThoList* . . . 742

H More On Arrays 748
H.1 *Interface of ThoArray* . . . 748
H.2 *Implementation of ThoArray* . . . 748

I More On Strings 751
I.1 *Interface of ThoString* . . . 751
I.2 *Implementation of ThoString* . . . 751

J Polymorphic Maps 754
J.1 *Interface of Pmap* . . . 754
J.2 *Implementation of Pmap* . . . 755
J.3 *Interface of Partial* . . . 764
J.4 *Implementation of Partial* . . . 765

K Tries 767
K.1 *Interface of Trie* . . . 767
K.1.1 *Monomorphically* . . . 767
K.1.2 *New in O'Caml 3.08* . . . 767
K.1.3 *O'Mega customization* . . . 768
K.1.4 *Polymorphically* . . . 768
K.1.5 *O'Mega customization* . . . 769
K.2 *Implementation of Trie* . . . 769
K.2.1 *Monomorphically* . . . 769
K.2.2 *O'Mega customization* . . . 772
K.2.3 *Polymorphically* . . . 772
K.2.4 *O'Mega customization* . . . 774

L Tensor Products 776
L.1 *Interface of Product* . . . 776
L.1.1 *Lists* . . . 776
L.1.2 *Sets* . . . 776
L.2 *Implementation of Product* . . . 777
L.2.1 *Lists* . . . 777
L.2.2 *Sets* . . . 778

M (Fiber) Bundles 779
M.1 *Interface of Bundle* . . . 779
M.2 *Implementation of Bundle* . . . 781

N Power Sets 783
N.1 *Interface of PowSet* . . . 783
N.2 *Implementation of PowSet* . . . 784

O Combinatorics 787
O.1 Interface of Combinatorics 787
O.1.1 Simple Combinatorial Functions 787
O.1.2 Partitions 787
O.1.3 Choices 789
O.1.4 Permutations 789
O.2 Implementation of Combinatorics 789
O.2.1 Simple Combinatorial Functions 789
O.2.2 Partitions 791
O.2.3 Choices 794
O.2.4 Permutations 795
O.3 Interface of Permutation 796
O.4 Implementation of Permutation 796

P Partitions 802
P.1 Interface of Partition 802
P.2 Implementation of Partition 802

Q Trees 804
Q.1 Interface of Tree 805
Q.1.1 Abstract Data Type 805
Q.1.2 Homomorphisms 806
Q.1.3 Output 806
Q.2 Implementation of Tree 808
Q.2.1 Abstract Data Type 808
Q.2.2 Homomorphisms 810
Q.2.3 Output 810
Q.2.4 Least Squares Layout 815

R Dependency Trees 820
R.1 Interface of Tree2 820
R.2 Implementation of Tree2 820

S Consistency Checks 821

T Complex Numbers 822

U Algebra 823
U.1 Interface of Algebra 823
U.1.1 Coefficients 823
U.1.2 Naive Rational Arithmetic 823
U.1.3 Expressions: Terms, Rings and Linear Combinations . . 824
U.2 Implementation of Algebra 826
U.2.1 Coefficients 826
U.2.2 Naive Rational Arithmetic 826
U.2.3 Expressions: Terms, Rings and Linear Combinations . . 827

V Simple Linear Algebra 833
V.1 Interface of Linalg 833
V.2 Implementation of Linalg 833
V.2.1 LU Decomposition 834

W Talk To The WHiZard . . . 838

X Fortran Libraries 839
X.1 Trivia . . . 839
X.1.1 Inner Product . . . 840
X.1.2 Spinor Vector Space . . . 840
X.1.3 Norm . . . 844
X.2 Spinors Revisited . . . 844
X.2.1 Spinor Vector Space . . . 845
X.2.2 Norm . . . 848
X.3 Vectorspinors . . . 848
X.3.1 Vectorspinor Vector Space . . . 849
X.3.2 Norm . . . 853
X.4 Vectors and Tensors . . . 853
X.4.1 Constructors . . . 854
X.4.2 Inner Products . . . 855
X.4.3 Not Entirely Inner Products . . . 856
X.4.4 Outer Products . . . 857
X.4.5 Vector Space . . . 858
X.4.6 Norm . . . 865
X.4.7 Conjugation . . . 865
X.4.8 ϵ-Tensors . . . 866
X.4.9 Utilities . . . 869
X.5 Polarization vectors . . . 869
X.6 Polarization vectors revisited . . . 872
X.7 Symmetric Tensors . . . 874
X.7.1 Vector Space . . . 875
X.8 Symmetric Polarization Tensors . . . 879
X.9 Couplings . . . 881
X.9.1 Triple Gauge Couplings . . . 885
X.9.2 Quadruple Gauge Couplings . . . 885
X.9.3 Scalar Current . . . 886
X.9.4 Transversal Scalar-Vector Coupling . . . 886
X.9.5 Triple Vector Couplings . . . 887
X.10 Graviton Couplings . . . 890
X.11 Tensor Couplings . . . 892
X.12 Scalar-Vector Dim-5 Couplings . . . 892
X.13 Spinor Couplings . . . 896
X.13.1 Fermionic Vector and Axial Couplings . . . 896
X.13.2 Fermionic Scalar and Pseudo Scalar Couplings . . . 910
X.13.3 On Shell Wave Functions . . . 914
X.13.4 Off Shell Wave Functions . . . 917
X.13.5 Propagators . . . 919
X.14 Spinor Couplings Revisited . . . 922
X.14.1 Fermionic Vector and Axial Couplings . . . 922
X.14.2 Fermionic Scalar and Pseudo Scalar Couplings . . . 928
X.14.3 Couplings for BRST Transformations . . . 930
X.14.4 Gravitino Couplings . . . 937
X.14.5 Gravitino 4-Couplings . . . 959
X.14.6 On Shell Wave Functions . . . 970

X.14.7 Off Shell Wave Functions 972
X.14.8 Propagators 973
X.15 Polarization vectorspinors 975
X.16 Color 979
X.16.1 Color Sum 979
X.17 Utilities 980
X.17.1 Helicity Selection Rule Heuristics 980
X.17.2 Diagnostics 981
X.17.3 Obsolete Summation 986
X.18 `omega95` 988
X.19 `omega95` Revisited 989
X.20 Testing 989
X.21 O'Mega Virtual Machine 1014
X.21.1 Memory Layout 1017
X.21.2 Controlling the VM 1020
X.21.3 Reading the bytecode 1026
X.21.4 Main Decode Function 1032
X.21.5 Helper functions 1043
X.21.6 O'Mega Interface 1050

Y Index 1054

—1—

Introduction

1.1 Complexity

There are

$$P(n) = \frac{2^n - 2}{2} - n = 2^{n-1} - n - 1 \tag{1.1}$$

independent internal momenta in a n-particle scattering amplitude [1]. This grows much slower than the number

$$F(n) = (2n-5)!! = (2n-5) \cdot (2n-7) \cdot \ldots \cdot 3 \cdot 1 \tag{1.2}$$

of tree Feynman diagrams in vanilla ϕ^3 (see table 1.1). There are no known corresponding expressions for theories with more than one particle type. However, empirical evidence from numerical studies [1, 2] as well as explicit counting results from O'Mega suggest

$$P^*(n) \propto 10^{n/2} \tag{1.3}$$

while he factorial growth of the number of Feynman diagrams remains unchecked, of course.

n	$P(n)$	$F(n)$
4	3	3
5	10	15
6	25	105
7	56	945
8	119	10395
9	246	135135
10	501	2027025
11	1012	34459425
12	2035	654729075
13	4082	13749310575
14	8177	316234143225
15	16368	7905853580625
16	32751	213458046676875

Table 1.1: The number of ϕ^3 Feynman diagrams $F(n)$ and independent poles $P(n)$.

The number of independent momenta in an amplitude is a better measure for the complexity of the amplitude than the number of Feynman diagrams, since there can be substantial cancellations among the latter. Therefore it should be possible to express the scattering amplitude more compactly than by a sum over Feynman diagrams.

1.2 Ancestors

Some of the ideas that O'Mega is based on can be traced back to HELAS [5]. HELAS builts Feynman amplitudes by recursively forming off-shell 'wave functions' from joining external lines with other external lines or off-shell 'wave functions'.

The program Madgraph [6] automatically generates Feynman diagrams and writes a Fortran program corresponding to their sum. The amplitudes are calculated by calls to HELAS [5]. Madgraph uses one straightforward optimization: no statement is written more than once. Since each statement corresponds to a collection of trees, this optimization is very effective for up to four particles in the final state. However, since the amplitudes are given as a sum of Feynman diagrams, this optimization can, by design, *not* remove the factorial growth and is substantially weaker than the algorithms of [1, 2] and the algorithm of O'Mega for more particles in the final state.

Then ALPHA [1] (see also the slightly modified variant [2]) provided a numerical algorithm for calculating scattering amplitudes and it could be shown empirically, that the calculational costs are rising with a power instead of factorially.

1.3 Architecture

1.3.1 General purpose libraries

Functions that are not specific to O'Mega and could be part of the O'Caml standard library

ThoList : (mostly) simple convenience functions for lists that are missing from the standard library module *List* (section G, p. 740)

Product : effcient tensor products for lists and sets (section L, p. 776)

Combinatorics : combinatorical formulae, sets of subsets, etc. (section O, p. 787)

1.3.2 O'Mega

The non-trivial algorithms that constitute O'Mega:

DAG : Directed Acyclical Graphs (section 4, p. 35)

Topology : unusual enumerations of unflavored tree diagrams (section 3, p. 18)

Momentum : finite sums of external momenta (section 5, p. 50)

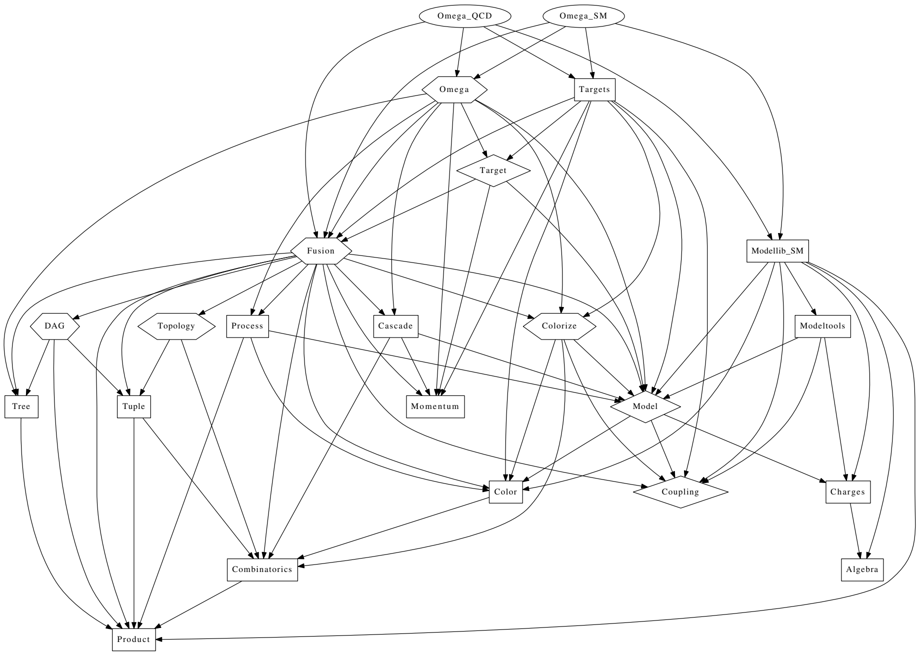


Figure 1.1: Module dependencies in O'Mega.

Fusion : off shell wave functions (section 8, p. 88)

Omega : functor constructing an application from a model and a target (section 18, p. 707)

1.3.3 Abstract interfaces

The domains and co-domains of functors (section 9, p. 135)

Coupling : all possible couplings (not comprensive yet)

Model : physical models

Target : target programming languages

1.3.4 Models

(section 13, p. 217)

Modellib$_S$*M.QED* : Quantum Electrodynamics

Modellib$_S$*M.QCD* : Quantum Chromodynamics (not complete yet)

Modellib$_S$*M.SM* : Minimal Standard Model (not complete yet)

etc.

1.3.5 Targets

Any programming language that supports arithmetic and a textual representation of programs can be targeted by O'Caml. The implementations translate the abstract expressions derived by *Fusion* to expressions in the target (section 15, p. 578).

Targets.Fortran : Fortran95 language implementation, calling subroutines

Other targets could come in the future: `C`, `C++`, O'Caml itself, symbolic manipulation languages, etc.

1.3.6 Applications

(section 18, p. 707)

1.4 The Big To Do Lists

1.4.1 Required

All features required for leading order physics applications are in place.

1.4.2 Useful

1. select allowed helicity combinations for massless fermions
2. Weyl-Van der Waerden spinors
3. speed up helicity sums by using discrete symmetries
4. general triple and quartic vector couplings
5. diagnostics: count corresponding Feynman diagrams more efficiently for more than ten external lines
6. recognize potential cascade decays (τ, b, etc.)
   - warn the user to add additional
   - kill fusions (at runtime), that contribute to a cascade
7. complete standard model in R_ξ-gauge
8. groves (the simple method of cloned generations works)

1.4.3 Future Features

1. investigate if unpolarized squared matrix elements can be calculated faster as traces of densitiy matrices. Unfortunately, the answer apears to be *no* for fermions and *up to a constant factor* for massive vectors. Since the number of fusions in the amplitude grows like $10^{n/2}$, the number of fusions in the squared matrix element grows like 10^n. On the other hand, there are $2^{\#\text{fermions}+\#\text{massless vectors}} \cdot 3^{\#\text{massive vectors}}$ terms in the helicity sum,

which grows *slower* than $10^{n/2}$. The constant factor is probably also not favorable. However, there will certainly be asymptotic gains for sums over gauge (and other) multiplets, like color sums.

2. compile Feynman rules from Lagrangians

3. evaluate amplitues in O'Caml by compiling it to three address code for a virtual machine

   type *mem* $=$ *scalar array* $\times$ *spinor array* $\times$ *spinor array* $\times$ *vector array*
   type *instr* $=$
   — *VSS* of *int* $\times$ *int* $\times$ *int*
   — *SVS* of *int* $\times$ *int* $\times$ *int*
   — *AVA* of *int* $\times$ *int* $\times$ *int*
   ...

   this could be as fast as [1] or [2].

4. a virtual machine will be useful for for other target as well, because native code appears to become to large for most compilers for more than ten external particles. Bytecode might even be faster due to improved cache locality.

5. use the virtual machine in O'Giga

1.4.4 Science Fiction

1. numerical and symbolical loop calculations with O'Tera: O'Mega Tool for Evaluating Renormalized Amplitudes

—2— Tuples and Polytuples

2.1 Interface of Tuple

The *Tuple.Poly* interface abstracts the notion of tuples with variable arity. Simple cases are binary polytuples, which are simply pairs and indefinite polytuples, which are nothing but lists. Another example is the union of pairs and triples. The interface is very similar to *List* from the O'Caml standard library, but the *Tuple.Poly* signature allows a more fine grained control of arities. The latter provides typesafe linking of models, targets and topologies.

```
module type Mono =
  sig
    type α t

    val arity : α t → int
    val max_arity : int

    val compare : (α → α → int) → α t → α t → int

    val for_all : (α → bool) → α t → bool

    val map : (α → β) → α t → β t
    val iter : (α → unit) → α t → unit
    val fold_left : (α → β → α) → α → β t → α
    val fold_right : (α → β → β) → α t → β → β
```

We have applications, where no sensible intial value can be defined:

```
    val fold_left_internal : (α → α → α) → α t → α
    val fold_right_internal : (α → α → α) → α t → α

    val map2 : (α → β → γ) → α t → β t → γ t

    val split : (α × β) t → α t × β t
```

The distributive tensor product expands a tuple of lists into list of tuples, e. g. for binary tuples:

$$\mathit{product}\,([x_1; x_2], [y_1; y_2]) = [(x_1, y_1); (x_1, y_2); (x_2, y_1); (x_2, y_2)] \tag{2.1}$$

NB: *product_fold* is usually much more memory efficient than the combination of *product* and *List.fold_right* for large sets.

```
    val product : α list t → α t list
```

```
val product_fold : (α t → β → β) → α list t → β → β
```

For homogeneous tuples the *power* function could trivially be built from *product*, e. g.:

$$power\,[x_1; x_2] = product\,([x_1; x_2], [x_1; x_2]) = [(x_1, x_1); (x_1, x_2); (x_2, x_1); (x_2, x_2)] \tag{2.2}$$

but it is also well defined for polytuples, e. g. for pairs and triples

$$power\,[x_1; x_2] = product\,([x_1; x_2], [x_1; x_2]) \cup product\,([x_1; x_2], [x_1; x_2], [x_1; x_2]) \tag{2.3}$$

For tuples and polytuples with bounded arity, the *power* and *power_fold* functions terminate. In polytuples with unbounded arity, the the *power* function always raises *No_termination*. *power_fold* also raises *No_termination*, but could be changed to run until the argument function raises an exception. However, if we need this behaviour, we should implemente *power_iter* instead.

```
val power : α list → α t list
val power_fold : (α t → β → β) → α list → β → β
```

We can also identify all (poly)tuples with permuted elements and return only one representative, e. g.:

$$sym\_power\,[x_1; x_2] = [(x_1, x_1); (x_1, x_2); (x_2, x_2)] \tag{2.4}$$

NB: this function has not yet been implemented, because O'Mega only needs the more efficient special case *graded_sym_power*.
If a set X is graded (i. e. there is a map $\phi : X \to \mathbf{N}$, called *rank* below), the results of *power* or *sym_power* can canonically be filtered by requiring that the sum of the ranks in each (poly)tuple has one chosen value. Implementing such a function directly is much more efficient than constructing and subsequently disregarding many (poly)tuples. The elements of rank n are at offset $(n-1)$ in the array. The array is assumed to be *immutable*, even if O'Caml doesn't support immutable arrays. NB: *graded_power* has not yet been implemented, because O'Mega only needs *graded_sym_power*.

```
type α graded = α list array
val graded_sym_power : int → α graded → α t list
val graded_sym_power_fold : int → (α t → β → β) → α graded →
  β → β
```

We hope to be able to avoid the next one in the long run, because it mildly breaks typesafety for arities. Unfortunately, we're still working on it ...

```
val to_list : α t → α list
```

The next one is only used for Fermi statistics below, but can not be implemented if there are no binary tuples. It must be retired as soon as possible.

```
val of2_kludge : α → α → α t

val rcs : RCS.t
end
```

```
module type Poly =
  sig
    include Mono
    exception Mismatched_arity
    exception No_termination
  end

module type Binary =
    sig
      include Poly (* should become Mono! *)
      val of2 : α → α → α t
    end
module Binary : Binary

module type Ternary =
    sig
      include Mono
      val of3 : α → α → α → α t
    end
module Ternary : Ternary

type α pair_or_triple = T2 of α × α | T3 of α × α × α

module type Mixed23 =
    sig
      include Poly
      val of2 : α → α → α t
      val of3 : α → α → α → α t
    end
module Mixed23 : Mixed23

module type Nary =
    sig
      include Poly
      val of2 : α → α → α t
      val of3 : α → α → α → α t
      val of_list : α list → α t
    end
module Unbounded_Nary : Nary

module type Bound = sig val max_arity : int end
module Nary (B : Bound) : Nary
```

For compleneteness sake, we could add most of the *List* signature

- `val length : α t → int`
- `val hd : α t → α`
- `val nth : α t → int → α`
- `val rev : α t → α t`
- `val rev_map : (α → β) → α t → β t`
- `val iter2 : (α → β → unit) → α t → β t → unit`

- val *rev_map2* : $(\alpha \to \beta \to \gamma) \to \alpha\ t \to \beta\ t \to \gamma\ t$
- val *fold_left2* : $(\alpha \to \beta \to \gamma \to \alpha) \to \alpha \to \beta\ t \to \gamma\ t \to \alpha$
- val *fold_right2* : $(\alpha \to \beta \to \gamma \to \gamma) \to \alpha\ t \to \beta\ t \to \gamma \to \gamma$
- val *exists* : $(\alpha \to bool) \to \alpha\ t \to bool$
- val *for_all2* : $(\alpha \to \beta \to bool) \to \alpha\ t \to \beta\ t \to bool$
- val *exists2* : $(\alpha \to \beta \to bool) \to \alpha\ t \to \beta\ t \to bool$
- val *mem* : $\alpha \to \alpha\ t \to bool$
- val *memq* : $\alpha \to \alpha\ t \to bool$
- val *find* : $(\alpha \to bool) \to \alpha\ t \to \alpha$
- val *find_all* : $(\alpha \to bool) \to \alpha\ t \to \alpha\ list$
- val *assoc* : $\alpha \to (\alpha \times \beta)\ t \to \beta$
- val *assq* : $\alpha \to (\alpha \times \beta)\ t \to \beta$
- val *mem_assoc* : $\alpha \to (\alpha \times \beta)\ t \to bool$
- val *mem_assq* : $\alpha \to (\alpha \times \beta)\ t \to bool$
- val *combine* : $\alpha\ t \to \beta\ t \to (\alpha \times \beta)\ t$
- val *sort* : $(\alpha \to \alpha \to int) \to \alpha\ t \to \alpha\ t$
- val *stable_sort* : $(\alpha \to \alpha \to int) \to \alpha\ t \to \alpha\ t$

but only if we ever have too much time on our hand ...

2.2 Implementation of *Tuple*

```
let rcs_file = RCS.parse "Tuple" ["Tuples␣of␣fixed␣and␣indefinite␣arity"]
    { RCS.revision = "$Revision:␣6465␣$";
      RCS.date = "$Date:␣2015-01-10␣16:22:31␣+0100␣(Sat,␣10␣Jan␣2015)␣$";
      RCS.author = "$Author:␣jr_reuter␣$";
      RCS.source
        = "$URL:␣svn+ssh://jr_reuter@login.hepforge.org/hepforge/svn/whizard/trunk/omega/sr

module type Mono =
  sig
    type α t
    val arity : α t → int
    val max_arity : int
    val compare : (α → α → int) → α t → α t → int
    val for_all : (α → bool) → α t → bool
    val map : (α → β) → α t → β t
    val iter : (α → unit) → α t → unit
    val fold_left : (α → β → α) → α → β t → α
    val fold_right : (α → β → β) → α t → β → β
    val fold_left_internal : (α → α → α) → α t → α
    val fold_right_internal : (α → α → α) → α t → α
    val map2 : (α → β → γ) → α t → β t → γ t
    val split : (α × β) t → α t × β t
```

```
    val product : α list t → α t list
    val product_fold : (α t → β → β) → α list t → β → β
    val power : α list → α t list
    val power_fold : (α t → β → β) → α list → β → β
    type α graded = α list array
    val graded_sym_power : int → α graded → α t list
    val graded_sym_power_fold : int → (α t → β → β) → α graded →
      β → β
    val to_list : α t → α list
    val of2_kludge : α → α → α t
    val rcs : RCS.t
  end

module type Poly =
  sig
    include Mono
    exception Mismatched_arity
    exception No_termination
  end
```

2.2.1 Typesafe Combinatorics

Wrap the combinatorical functions with varying arities into typesafe functions with fixed arities. We could provide specialized implementations, but since we *know* that *Impossible* is *never* raised, the present approach is just as good (except for a tiny inefficiency).

```
exception Impossible of string
let impossible name = raise (Impossible name)

let choose2 set =
  List.map (function [x; y] → (x, y) | _ → impossible "choose2")
      (Combinatorics.choose 2 set)

let choose3 set =
  List.map (function [x; y; z] → (x, y, z) | _ → impossible "choose3")
    (Combinatorics.choose 3 set)
```

2.2.2 Pairs

```
module type Binary =
    sig
      include Poly (* should become Mono! *)
      val of2 : α → α → α t
    end

module Binary =
  struct
    let rcs = RCS.rename rcs_file "Tuple.Binary" ["Pairs"]

    type α t = α × α
```

```
let arity _ = 2
let max_arity = 2

let of2 x y = (x, y)

let compare cmp (x1, y1) (x2, y2) =
  let cx = cmp x1 x2 in
  if cx ≠ 0 then
    cx
  else
    cmp y1 y2

let for_all p (x, y) = p x ∧ p y

let map f (x, y) = (f x, f y)
let iter f (x, y) = f x; f y
let fold_left f init (x, y) = f (f init x) y
let fold_right f (x, y) init = f x (f y init)
let fold_left_internal f (x, y) = f x y
let fold_right_internal f (x, y) = f x y

exception Mismatched_arity
let map2 f (x1, y1) (x2, y2) = (f x1 x2, f y1 y2)

let split ((x1, x2), (y1, y2)) = ((x1, y1), (x2, y2))

let product (lx, ly) =
  Product.list2 (fun x y → (x, y)) lx ly
let product_fold f (lx, ly) init =
  Product.fold2 (fun x y → f (x, y)) lx ly init

let power l = product (l, l)
let power_fold f l = product_fold f (l, l)
```

In the special case of binary fusions, the implementation is very concise.

```
type α graded = α list array

let fuse2 f set (i, j) acc =
  if i = j then
    List.fold_right (fun (x, y) → f x y) (choose2 set.(pred i)) acc
  else
    Product.fold2 f set.(pred i) set.(pred j) acc

let graded_sym_power_fold rank f set acc =
  let max_rank = Array.length set in
  List.fold_right (fuse2 (fun x y → f (of2 x y)) set)
    (Partition.pairs rank 1 max_rank) acc

let graded_sym_power rank set =
  graded_sym_power_fold rank (fun pair acc → pair :: acc) set []

let to_list (x, y) = [x; y]
let of2_kludge = of2

exception No_termination
end
```

2.2.3 Triples

```
module type Ternary =
  sig
    include Mono
    val of3 : α → α → α → α t
  end

module Ternary =
 struct
  let rcs = RCS.rename rcs_file "Tuple.Ternary" ["Triples"]

  type α t = α × α × α

  let arity _ = 3
  let max_arity = 3

  let of3 x y z = (x, y, z)

  let compare cmp (x1, y1, z1) (x2, y2, z2) =
    let cx = cmp x1 x2 in
    if cx ≠ 0 then
      cx
    else
      let cy = cmp y1 y2 in
      if cy ≠ 0 then
        cy
      else
        cmp z1 z2

  let for_all p (x, y, z) = p x ∧ p y ∧ p z

  let map f (x, y, z) = (f x, f y, f z)
  let iter f (x, y, z) = f x; f y; f z
  let fold_left f init (x, y, z) = f (f (f init x) y) z
  let fold_right f (x, y, z) init = f x (f y (f z init))
  let fold_left_internal f (x, y, z) = f (f x y) z
  let fold_right_internal f (x, y, z) = f x (f y z)

  exception Mismatched_arity
  let map2 f (x1, y1, z1) (x2, y2, z2) = (f x1 x2, f y1 y2, f z1 z2)

  let split ((x1, x2), (y1, y2), (z1, z2)) = ((x1, y1, z1), (x2, y2, z2))

  let product (lx, ly, lz) =
    Product.list3 (fun x y z → (x, y, z)) lx ly lz
  let product_fold f (lx, ly, lz) init =
    Product.fold3 (fun x y z → f (x, y, z)) lx ly lz init

  let power l = product (l, l, l)
  let power_fold f l = product_fold f (l, l, l)

  type α graded = α list array

  let fuse3 f set (i, j, k) acc =
    if i = j then begin
      if j = k then
```

```
        List.fold_right (fun (x, y, z) → f x y z) (choose3 set.(pred i)) acc
      else
        Product.fold2 (fun (x, y) z → f x y z)
          (choose2 set.(pred i)) set.(pred k) acc
    end else begin
      if j = k then
        Product.fold2 (fun x (y, z) → f x y z)
          set.(pred i) (choose2 set.(pred j)) acc
      else
        Product.fold3 (fun x y z → f x y z)
          set.(pred i) set.(pred j) set.(pred k) acc
    end

  let graded_sym_power_fold rank f set acc =
    let max_rank = Array.length set in
    List.fold_right (fuse3 (fun x y z → f (of3 x y z)) set)
      (Partition.triples rank 1 max_rank) acc

  let graded_sym_power rank set =
    graded_sym_power_fold rank (fun pair acc → pair :: acc) set []

  let of2_kludge _ = failwith "Tuple.Ternary.of2_kludge"

  let to_list (x, y, z) = [x; y; z]

end
```

2.2.4 Pairs and Triples

```
type α pair_or_triple = T2 of α × α | T3 of α × α × α

module type Mixed23 =
  sig
    include Poly
    val of2 : α → α → α t
    val of3 : α → α → α → α t
  end

module Mixed23 =
  struct
    let rcs = RCS.rename rcs_file "Tuple.Mixed23"
        ["Mixed␣pairs␣and␣triples"]

    type α t = α pair_or_triple

    let arity = function
      | T2 _ → 2
      | T3 _ → 3
    let max_arity = 3

    let of2 x y = T2 (x, y)
    let of3 x y z = T3 (x, y, z)

    let compare cmp m1 m2 =
      match m1, m2 with
```

```
  | T2 _, T3 _ → −1
  | T3 _, T2 _ → 1
  | T2 (x1, y1), T2 (x2, y2) →
      let cx = cmp x1 x2 in
      if cx ≠ 0 then
        cx
      else
        cmp y1 y2
  | T3 (x1, y1, z1), T3 (x2, y2, z2) →
      let cx = cmp x1 x2 in
      if cx ≠ 0 then
        cx
      else
        let cy = cmp y1 y2 in
        if cy ≠ 0 then
          cy
        else
          cmp z1 z2

let for_all p = function
  | T2 (x, y) → p x ∧ p y
  | T3 (x, y, z) → p x ∧ p y ∧ p z

let map f = function
  | T2 (x, y) → T2 (f x, f y)
  | T3 (x, y, z) → T3 (f x, f y, f z)

let iter f = function
  | T2 (x, y) → f x; f y
  | T3 (x, y, z) → f x; f y; f z

let fold_left f init = function
  | T2 (x, y) → f (f init x) y
  | T3 (x, y, z) → f (f (f init x) y) z

let fold_right f m init =
  match m with
  | T2 (x, y) → f x (f y init)
  | T3 (x, y, z) → f x (f y (f z init))

let fold_left_internal f m =
  match m with
  | T2 (x, y) → f x y
  | T3 (x, y, z) → f (f x y) z

let fold_right_internal f m =
  match m with
  | T2 (x, y) → f x y
  | T3 (x, y, z) → f x (f y z)

exception Mismatched_arity
let map2 f m1 m2 =
  match m1, m2 with
  | T2 (x1, y1), T2 (x2, y2) → T2 (f x1 x2, f y1 y2)
```

```
      | T3 (x1, y1, z1), T3 (x2, y2, z2) → T3 (f x1 x2, f y1 y2, f z1 z2)
      | T2 _, T3 _ | T3 _, T2 _ → raise Mismatched_arity

    let split = function
      | T2 ((x1, x2), (y1, y2)) → (T2 (x1, y1), T2 (x2, y2))
      | T3 ((x1, x2), (y1, y2), (z1, z2)) → (T3 (x1, y1, z1), T3 (x2, y2, z2))

    let product = function
      | T2 (lx, ly) → Product.list2 (fun x y → T2 (x, y)) lx ly
      | T3 (lx, ly, lz) → Product.list3 (fun x y z → T3 (x, y, z)) lx ly lz
    let product_fold f m init =
      match m with
      | T2 (lx, ly) → Product.fold2 (fun x y → f (T2 (x, y))) lx ly init
      | T3 (lx, ly, lz) →
          Product.fold3 (fun x y z → f (T3 (x, y, z))) lx ly lz init

    exception No_termination

    let power_fold f l init =
      product_fold f (T2 (l, l)) (product_fold f (T3 (l, l, l)) init)
    let power l =
      power_fold (fun m acc → m :: acc) l []

    type α graded = α list array

    let graded_sym_power_fold rank f set acc =
      let max_rank = Array.length set in
      List.fold_right (Binary.fuse2 (fun x y → f (of2 x y)) set)
        (Partition.pairs rank 1 max_rank)
        (List.fold_right (Ternary.fuse3 (fun x y z → f (of3 x y z)) set)
           (Partition.triples rank 1 max_rank) acc)

    let graded_sym_power rank set =
      graded_sym_power_fold rank (fun pair acc → pair :: acc) set []

    let to_list = function
      | T2 (x, y) → [x; y]
      | T3 (x, y, z) → [x; y; z]

    let of2_kludge = of2

  end
```

2.2.5 ... and All The Rest

```
module type Nary =
    sig
      include Poly
      val of2 : α → α → α t
      val of3 : α → α → α → α t
      val of_list : α list → α t
    end

module Nary (A : sig val max_arity : int end) =
  struct
```

```
let rcs = RCS.rename rcs_file "Tuple.Nary"
    ["Tupels of indefinite arity"]

type α t = α × α list

let arity (_, y) = succ (List.length y)
let max_arity = A.max_arity

let of2 x y = (x, [y])
let of3 x y z = (x, [y; z])

let of_list = function
  | x :: y → (x, y)
  | [] → invalid_arg "Tuple.Nary.of_list: empty"

let compare cmp (x1, y1) (x2, y2) =
  let c = cmp x1 x2 in
  if c ≠ 0 then
    c
  else
    ThoList.compare ~cmp y1 y2

let for_all p (x, y) = p x ∧ List.for_all p y

let map f (x, y) = (f x, List.map f y)
let iter f (x, y) = f x; List.iter f y
let fold_left f init (x, y) = List.fold_left f (f init x) y
let fold_right f (x, y) init = f x (List.fold_right f y init)
let fold_left_internal f (x, y) = List.fold_left f x y
let fold_right_internal f (x, y) =
  match List.rev y with
  | [] → x
  | y0 :: y_sans_y0 →
      f x (List.fold_right f (List.rev y_sans_y0) y0)

exception Mismatched_arity
let map2 f (x1, y1) (x2, y2) =
  try (f x1 x2, List.map2 f y1 y2) with
  | Invalid_argument _ → raise Mismatched_arity

let split ((x1, x2), y12) =
  let y1, y2 = List.split y12 in
  ((x1, y1), (x2, y2))

let product (xl, yl) =
  Product.list (function
    | x :: y → (x, y)
    | [] → failwith "Tuple.Nary.product") (xl :: yl)
let product_fold f (xl, yl) init =
  Product.fold (function
    | x :: y → f (x, y)
    | [] → failwith "Tuple.Nary.product_fold") (xl :: yl) init

let bounded_power_fold f l init =
  List.fold_right (fun n → product_fold f (l, ThoList.clone (pred n) l))
    (ThoList.range 2 A.max_arity) init
```

```
    let bounded_power l =
      bounded_power_fold (fun t acc → t :: acc) l []

    exception No_termination
    let unbounded_power_fold f l init = raise No_termination
    let unbounded_power l = raise No_termination

    let power_fold, power =
      if A.max_arity > 0 then
        (bounded_power_fold, bounded_power)
      else
        (unbounded_power_fold, unbounded_power)

    type α graded = α list array

    let fuse_n f set partition acc =
      let choose (n, r) =
        Printf.printf "chose: n=%d r=%d len=%d\n"
          n r (List.length set.(pred r));
        Combinatorics.choose n set.(pred r) in
      Product.fold (fun wfs → f (List.concat wfs))
        (List.map choose (ThoList.classify partition)) acc

    let fuse_n f set partition acc =
      let choose (n, r) = Combinatorics.choose n set.(pred r) in
      Product.fold (fun wfs → f (List.concat wfs))
        (List.map choose (ThoList.classify partition)) acc
```

graded_sym_power_fold is well defined for unbounded arities as well: derive a reasonable replacement from *set*. The length of the flattened *set* is an upper limit, of course, but too pessimistic in most cases.

```
    let graded_sym_power_fold rank f set acc =
      let max_rank = Array.length set in
      let degrees = ThoList.range 2 max_arity in
      let partitions =
        ThoList.flatmap
          (fun deg → Partition.tuples deg rank 1 max_rank) degrees in
      List.fold_right (fuse_n (fun wfs → f (of_list wfs)) set) partitions acc

    let graded_sym_power rank set =
      graded_sym_power_fold rank (fun pair acc → pair :: acc) set []

    let to_list (x, y) = x :: y
    let of2_kludge = of2

  end

module type Bound = sig val max_arity : int end
module Unbounded_Nary = Nary (struct let max_arity = − 1 end)
```

—3— Topologies

3.1 Interface of Topology

module type T =
  sig

partition is a collection of integers, with arity one larger than the arity of α *children* below. These arities can one fixed number corresponding to homogeneous tuples or a collection of tupes or lists.

type *partition*

partitions n returns the union of all $[n_1; n_2; \ldots; n_d]$ with $1 \leq n_1 \leq n_2 \leq \ldots \leq n_d \leq \lfloor n/2 \rfloor$ and

$$\sum_{i=1}^{d} n_i = n \tag{3.1}$$

for d from 3 to $d_{\max}$, where $d_{\max}$ is a fixed number for each module implementating T. In particular, if type $partition = int \times int \times int$, then *partitions* n returns all (n_1, n_2, n_3) with $n_1 \leq n_2 \leq n_3$ and $n_1 + n_2 + n_3 = n$.

val $partitions : int \to partition\ list$

A (poly)tuple as implemented by the modules in *Tuple*:

type α *children*

keystones externals returns all keystones for the amplitude with external states *externals* in the vanilla scalar theory with a

$$\sum_{3 \leq k \leq d_{\max}} \lambda_k \phi^k \tag{3.2}$$

interaction. One factor of the products is factorized. In particular, if

type $\alpha\ children = \alpha\ Tuple.Binary.t = \alpha \times \alpha$,

then *keystones externals* returns all keystones for the amplitude with external states *externals* in the vanilla scalar $\lambda\phi^3$-theory.

val $keystones : \alpha\ list \to (\alpha\ list \times \alpha\ list\ children\ list)\ list$

The maximal depth of subtrees for a given number of external lines.

```
val max_subtree : int → int
```

Only for diagnostics:

```
  val inspect_partition : partition → int list
  val rcs : RCS.t
end

module Binary : T with type α children = α Tuple.Binary.t
module Ternary : T with type α children = α Tuple.Ternary.t
module Mixed23 : T with type α children = α Tuple.Mixed23.t
module Nary : functor (B : Tuple.Bound) →
  (T with type α children = α Tuple.Nary(B).t)
```

3.1.1 Diagnostics: Counting Diagrams and Factorizations for $\sum_n \lambda_n \phi^n$

The number of diagrams for many particles can easily exceed the range of native integers. Even if we can not calculate the corresponding amplitudes, we want to check combinatorical factors. Therefore we code a functor that can use arbitray implementations of integers.

```
module type Integer =
  sig
    type t
    val zero : t
    val one : t
    val ( + ) : t → t → t
    val ( − ) : t → t → t
    val ( × ) : t → t → t
    val ( / ) : t → t → t
    val pred : t → t
    val succ : t → t
    val ( = ) : t → t → bool
    val ( ≠ ) : t → t → bool
    val ( < ) : t → t → bool
    val ( ≤ ) : t → t → bool
    val ( > ) : t → t → bool
    val ( ≥ ) : t → t → bool
    val of_int : int → t
    val to_int : t → int
    val to_string : t → string
    val compare : t → t → int
    val factorial : t → t
  end
```

Of course, native integers will provide the fastest implementation:

```
module Int : Integer

module type Count =
  sig
    type integer
```

diagrams f d n returns the number of tree diagrams contributing to the n-point amplitude in vanilla scalar theory with

$$\sum_{3 \leq k \leq d \wedge f(k)} \lambda_k \phi^k \tag{3.3}$$

interaction. The default value of f returns true for all arguments.

val *diagrams* : ?*f* : (*integer* → *bool*) → *integer* → *integer* → *integer*
val *diagrams_via_keystones* : *integer* → *integer* → *integer*

$$\frac{1}{S(n_k, n-n_k)} \frac{1}{S(n_1, n_2, \ldots, n_k)} \binom{n_1 + n_2 + \ldots + n_k}{n_1, n_2, \ldots, n_k} \tag{3.4}$$

val *keystones* : *integer list* → *integer*

diagrams_via_keystones d n must produce the same results as *diagrams d n*. This is shown explicitely in tables 3.2, 3.3 and 3.4 for small values of d and n. The test program in appendix S can be used to verify this relation for larger values.

val *diagrams_per_keystone* : *integer* → *integer list* → *integer*

end

module *Count* : functor (*I* : *Integer*) → *Count* with type *integer* = *I.t*

3.1.2 Emulating HELAC

We can also proceed á la [2].

module *Helac* : functor (*B* : *Tuple.Bound*) →
(*T* with type α *children* = α *Tuple.Nary*(*B*).*t*)

The following has never been tested, but it is no rocket science and should work anyway . . .

module *Helac_Binary* : *T* with type α *children* = α *Tuple.Binary.t*

3.2 Implementation of *Topology*

let *rcs_file* = *RCS.parse* "Topology" ["Topologies"]
{ *RCS.revision* = "$Revision:␣6465␣$";
RCS.date = "$Date:␣2015-01-10␣16:22:31␣+0100␣(Sat,␣10␣Jan␣2015)␣$";
RCS.author = "$Author:␣jr_reuter␣$";
RCS.source
= "$URL:␣svn+ssh://jr_reuter@login.hepforge.org/hepforge/svn/whizard/trunk/omega/s

module type *T* =
sig
type *partition*
val *partitions* : *int* → *partition list*

n	*partitions* n
4	(1,1,2)
5	(1,2,2)
6	(1,2,3), (2,2,2)
7	(1,3,3), (2,2,3)
8	(1,3,4), (2,2,4), (2,3,3)
9	(1,4,4), (2,3,4), (3,3,3)
10	(1,4,5), (2,3,5), (2,4,4), (3,3,4)
11	(1,5,5), (2,4,5), (3,3,5), (3,4,4)
12	(1,5,6), (2,4,6), (2,5,5), (3,3,6), (3,4,5), (4,4,4)
13	(1,6,6), (2,5,6), (3,4,6), (3,5,5), (4,4,5)
14	(1,6,7), (2,5,7), (2,6,6), (3,4,7), (3,5,6), (4,4,6), (4,5,5)
15	(1,7,7), (2,6,7), (3,5,7), (3,6,6), (4,4,7), (4,5,6), (5,5,5)
16	(1,7,8), (2,6,8), (2,7,7), (3,5,8), (3,6,7), (4,4,8), (4,5,7), (4,6,6), (5,5,6)

Table 3.1: *partitions* n for moderate values of n.

```
    type α children
    val keystones : α list → (α list × α list children list) list
    val max_subtree : int → int
    val inspect_partition : partition → int list
    val rcs : RCS.t
  end
```

3.2.1 Factorizing Diagrams for ϕ^3

```
module Binary =
  struct
    let rcs = RCS.rename rcs_file "Topology.Binary"
        ["phi**3 topology"]

    type partition = int × int × int
    let inspect_partition (n1, n2, n3) = [n1; n2; n3]
```

One way [1] to lift the degeneracy is to select the vertex that is closest to the center (see table 3.1):

$$partitions : n \to \left\{(n_1,n_2,n_3)\,|\,n_1+n_2+n_3 = n \wedge n_1 \leq n_2 \leq n_3 \leq \lfloor n/2 \rfloor\right\} \quad (3.5)$$

Other, less symmetric, approaches are possible. The simplest of these is: choose the vertex adjacent to a fixed external line [2]. They will be made available for comparison in the future.
An obvious consequence of $n_1 + n_2 + n_3 = n$ and $n_1 \leq n_2 \leq n_3$ is $n_1 \leq \lfloor n/3 \rfloor$:

```
    let rec partitions' n n1 =
      if n1 > n / 3 then
        []
      else
        List.map (fun (n2, n3) → (n1, n2, n3))
```

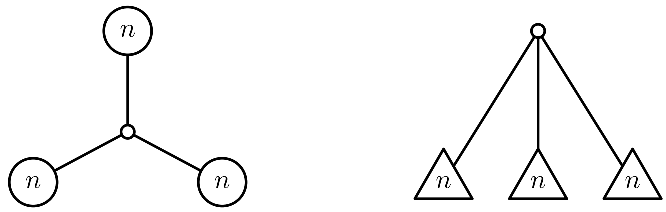


Figure 3.1: Topologies with a blatant three-fold permutation symmetry, if the number of external lines is a multiple of three

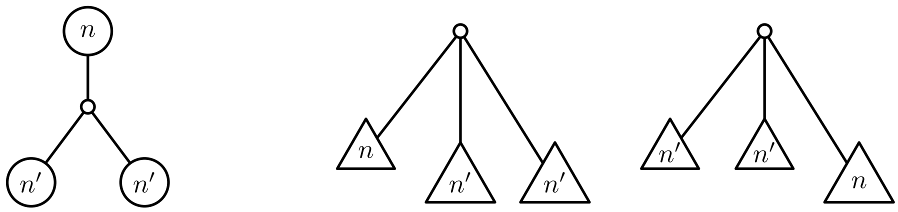


Figure 3.2: Topologies with a blatant two-fold symmetry.

$(Partition.pairs\ (n\ -\ n1)\ n1\ (n\ /\ 2))$ @ $partitions'\ n\ (succ\ n1)$

let $partitions\ n\ =\ partitions'\ n\ 1$

type $\alpha\ children\ =\ \alpha\ Tuple.Binary.t$

There remains one peculiar case, when the number of external lines is even and $n_3 = n_1 + n_2$ (cf. figure 3.3). Unfortunately, this reflection symmetry is not respected by the equivalence classes. E. g.

$$\{1\}\{2,3\}\{4,5,6\} \mapsto \big\{\{4\}\{5,6\}\{1,2,3\}; \{5\}\{4,6\}\{1,2,3\}; \{6\}\{4,5\}\{1,2,3\}\big\} \tag{3.6}$$

However, these reflections will always exchange the two halves and a representative can be chosen by requiring that one fixed momentum remains in one half. We choose to filter out the half of the partitions where the element p appears in the second half, i. e. the list of length $n3$.

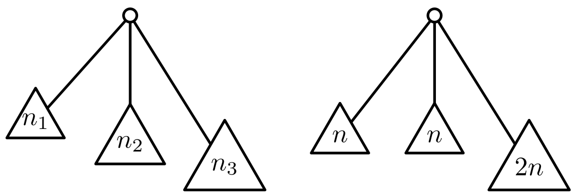


Figure 3.3: If $n_3 = n_1 + n_2$, the apparently asymmetric topologies on the left hand side have a non obvious two-fold symmetry, that exchanges the two halves. Therefore, the topologies on the right hand side have a four fold symmetry.

n	$(2n-5)!!$	$\sum N(n_1,n_2,n_3)$
4	3	$3\cdot(1,1,2)$
5	15	$15\cdot(1,2,2)$
6	105	$90\cdot(1,2,3)+15\cdot(2,2,2)$
7	945	$630\cdot(1,3,3)+315\cdot(2,2,3)$
8	10395	$6300\cdot(1,3,4)+1575\cdot(2,2,4)+2520\cdot(2,3,3)$
9	135135	$70875\cdot(1,4,4)+56700\cdot(2,3,4)+7560\cdot(3,3,3)$
10	2027025	$992250\cdot(1,4,5)+396900\cdot(2,3,5)$
		$+354375\cdot(2,4,4)+283500\cdot(3,3,4)$
11	34459425	$15280650\cdot(1,5,5)+10914750\cdot(2,4,5)$
		$+4365900\cdot(3,3,5)+3898125\cdot(3,4,4)$
12	654729075	$275051700\cdot(1,5,6)+98232750\cdot(2,4,6)$
		$+91683900\cdot(2,5,5)+39293100\cdot(3,3,6)$
		$+130977000\cdot(3,4,5)+19490625\cdot(4,4,4)$

Table 3.2: Equation (3.9) for small values of n.

Finally, a closed expression for the number of Feynman diagrams in the equivalence class (n_1,n_2,n_3) is

$$N(n_1,n_2,n_3)=\frac{(n_1+n_2+n_3)!}{S(n_1,n_2,n_3)}\prod_{i=1}^{3}\frac{(2n_i-3)!!}{n_i!} \tag{3.7}$$

where the symmetry factor from the above arguments is

$$S(n_1,n_2,n_3)=\begin{cases}3! & \text{for } n_1=n_2=n_3\\ 2\cdot 2 & \text{for } n_3=2n_1=2n_2\\ 2 & \text{for } n_1=n_2 \lor n_2=n_3\\ 2 & \text{for } n_1+n_2=n_3\end{cases} \tag{3.8}$$

Indeed, the sum of all Feynman diagrams

$$\sum_{\substack{n_1+n_2+n_3=n\\ 1\le n_1\le n_2\le n_3\le\lfloor n/2\rfloor}} N(n_1,n_2,n_3)=(2n-5)!! \tag{3.9}$$

can be checked numerically for large values of $n=n_1+n_2+n_3$, verifying the symmetry factor (see table 3.2).

P. M. claims to have seen similar formulae in the context of Young tableaux. That's a good occasion to read the new edition of Howard's book . . .

Return a list of all inequivalent partitions of the list l in three lists of length *n1*, *n2* and *n3*, respectively. Common first lists are factored. This is nothing more than a typedafe wrapper around *Combinatorics.factorized_keystones*.

```
exception Impossible of string
let tuple_of_list2 = function
  | [x1; x2] → Tuple.Binary.of2 x1 x2
  | _ → raise (Impossible "Topology.tuple_of_list")
```

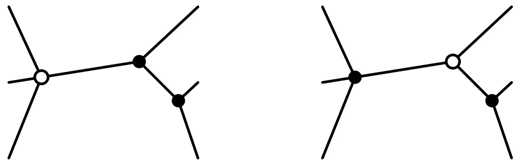


Figure 3.4: Degenerate (1, 1, 1, 3) and (1, 2, 3).

```
    let keystone (n1, n2, n3) l =
      List.map (fun (p1, p23) → (p1, List.rev_map tuple_of_list2 p23))
        (Combinatorics.factorized_keystones [n1; n2; n3] l)

    let keystones l =
      ThoList.flatmap (fun n123 → keystone n123 l) (partitions (List.length l))

    let max_subtree n = n / 2

  end
```

3.2.2 Factorizing Diagrams for $\sum_n \lambda_n \phi^n$

Mixed ϕ^n adds new degeneracies, as in figure 3.4. They appear if and only if one part takes exactly half of the external lines and can relate central vertices of different arity.

```
module Nary (B : Tuple.Bound) =
  struct
    let rcs = RCS.rename rcs_file "Topology.Nary"
        ["phi**n topology"]

    type partition = int list
    let inspect_partition p = p

    let partition d sum =
      Partition.tuples d sum 1 (sum / 2)

    let rec partitions' d sum =
      if d < 3 then
        []
      else
        partition d sum @ partitions' (pred d) sum

    let partitions sum = partitions' (succ B.max_arity) sum


    module Tuple = Tuple.Nary(B)
    type α children = α Tuple.t

    let keystones' l =
      let n = List.length l in
      ThoList.flatmap (fun p → Combinatorics.factorized_keystones p l)
```

n	$\sum$	$\sum$
4	4	$1 \cdot (1,1,1,1) + 3 \cdot (1,1,2)$
5	25	$10 \cdot (1,1,1,2) + 15 \cdot (1,2,2)$
6	220	$40 \cdot (1,1,1,3) + 45 \cdot (1,1,2,2) + 120 \cdot (1,2,3) + 15 \cdot (2,2,2)$
7	2485	$840 \cdot (1,1,2,3) + 105 \cdot (1,2,2,2) + 1120 \cdot (1,3,3) + 420 \cdot (2,2,3)$
8	34300	$5250 \cdot (1,1,2,4) + 4480 \cdot (1,1,3,3) + 3360 \cdot (1,2,2,3)$
		$+ 105 \cdot (2,2,2,2) + 14000 \cdot (1,3,4)$
		$+ 2625 \cdot (2,2,4) + 4480 \cdot (2,3,3)$
9	559405	$126000 \cdot (1,1,3,4) + 47250 \cdot (1,2,2,4) + 40320 \cdot (1,2,3,3)$
		$+ 5040 \cdot (2,2,2,3) + 196875 \cdot (1,4,4)$
		$+ 126000 \cdot (2,3,4) + 17920 \cdot (3,3,3)$
10	10525900	$1108800 \cdot (1,1,3,5) + 984375 \cdot (1,1,4,4) + 415800 \cdot (1,2,2,5)$
		$+ 1260000 \cdot (1,2,3,4) + 179200 \cdot (1,3,3,3) + 78750 \cdot (2,2,2,4)$
		$+ 100800 \cdot (2,2,3,3) + 3465000 \cdot (1,4,5) + 1108800 \cdot (2,3,5)$
		$+ 984375 \cdot (2,4,4) + 840000 \cdot (3,3,4)$

Table 3.3: $\mathcal{L} = \lambda_3 \phi^3 + \lambda_4 \phi^4$

n	$\sum$	$\sum$
4	4	$1 \cdot (1,1,1,1) + 3 \cdot (1,1,2)$
5	26	$1 \cdot (1,1,1,1,1) + 10 \cdot (1,1,1,2) + 15 \cdot (1,2,2)$
6	236	$1 \cdot (1,1,1,1,1,1) + 15 \cdot (1,1,1,1,2) + 40 \cdot (1,1,1,3)$
		$+ 45 \cdot (1,1,2,2) + 120 \cdot (1,2,3) + 15 \cdot (2,2,2)$
7	2751	$21 \cdot (1,1,1,1,1,2) + 140 \cdot (1,1,1,1,3) + 105 \cdot (1,1,1,2,2)$
		$+ 840 \cdot (1,1,2,3) + 105 \cdot (1,2,2,2) + 1120 \cdot (1,3,3) + 420 \cdot (2,2,3)$
8	39179	$224 \cdot (1,1,1,1,1,3) + 210 \cdot (1,1,1,1,2,2) + 910 \cdot (1,1,1,1,4)$
		$+ 2240 \cdot (1,1,1,2,3) + 420 \cdot (1,1,2,2,2) + 5460 \cdot (1,1,2,4)$
		$+ 4480 \cdot (1,1,3,3) + 3360 \cdot (1,2,2,3) + 105 \cdot (2,2,2,2)$
		$+ 14560 \cdot (1,3,4) + 2730 \cdot (2,2,4) + 4480 \cdot (2,3,3)$

Table 3.4: $\mathcal{L} = \lambda_3 \phi^3 + \lambda_4 \phi^4 + \lambda_5 \phi^5 + \lambda_6 \phi^6$

```
        (partitions n)

    let keystones l =
      List.map (fun (bra, kets) → (bra, List.map Tuple.of_list kets))
        (keystones' l)

    let max_subtree n = n / 2

  end

module Nary4 = Nary (struct let max_arity = 3 end)
```

3.2.3 Factorizing Diagrams for ϕ^4

```
module Ternary =
  struct
    let rcs = RCS.rename rcs_file "Topology.Ternary"
        ["phi**4␣topology"]
    let rcs = rcs_file
    type partition = int × int × int × int
    let inspect_partition (n1, n2, n3, n4) = [n1; n2; n3; n4]
    type α children = α Tuple.Ternary.t
    let collect4 acc = function
      | [x; y; z; u] → (x, y, z, u) :: acc
      | _ → acc
    let partitions n =
      List.fold_left collect4 [] (Nary4.partitions n)
    let collect3 acc = function
      | [x; y; z] → Tuple.Ternary.of3 x y z :: acc
      | _ → acc
    let keystones l =
      List.map (fun (bra, kets) → (bra, List.fold_left collect3 [] kets))
        (Nary4.keystones' l)
    let max_subtree = Nary4.max_subtree
  end
```

3.2.4 Factorizing Diagrams for $\phi^3 + \phi^4$

```
module Mixed23 =
  struct
    let rcs = RCS.rename rcs_file "Topology.Mixed23"
        ["phi**3␣+␣phi**4␣topology"]
    type partition =
      | P3 of int × int × int
      | P4 of int × int × int × int
    let inspect_partition = function
      | P3 (n1, n2, n3) → [n1; n2; n3]
      | P4 (n1, n2, n3, n4) → [n1; n2; n3; n4]
    type α children = α Tuple.Mixed23.t
    let collect34 acc = function
```

```
      | [x; y; z] → P3 (x, y, z) :: acc
      | [x; y; z; u] → P4 (x, y, z, u) :: acc
      | _ → acc
    let partitions n =
      List.fold_left collect34 [] (Nary4.partitions n)
    let collect23 acc = function
      | [x; y] → Tuple.Mixed23.of2 x y :: acc
      | [x; y; z] → Tuple.Mixed23.of3 x y z :: acc
      | _ → acc
    let keystones l =
      List.map (fun (bra, kets) → (bra, List.fold_left collect23 [] kets))
        (Nary4.keystones' l)
    let max_subtree = Nary4.max_subtree
  end
```

3.2.5 Diagnostics: Counting Diagrams and Factorizations for $\sum_n \lambda_n \phi^n$

```
module type Integer =
  sig
    type t
    val zero : t
    val one : t
    val ( + ) : t → t → t
    val ( − ) : t → t → t
    val ( × ) : t → t → t
    val ( / ) : t → t → t
    val pred : t → t
    val succ : t → t
    val ( = ) : t → t → bool
    val ( ≠ ) : t → t → bool
    val ( < ) : t → t → bool
    val ( ≤ ) : t → t → bool
    val ( > ) : t → t → bool
    val ( ≥ ) : t → t → bool
    val of_int : int → t
    val to_int : t → int
    val to_string : t → string
    val compare : t → t → int
    val factorial : t → t
  end
```

O'Caml's native integers suffice for all applications, but in appendix S, we want to use big integers for numeric checks in high orders:

```
module Int : Integer =
  struct
    type t = int
    let zero = 0
    let one = 1
```

```
    let ( + ) = ( + )
    let ( − ) = ( − )
    let ( × ) = ( × )
    let ( / ) = ( / )
    let pred = pred
    let succ = succ
    let ( = ) = ( = )
    let ( ≠ ) = ( ≠ )
    let ( < ) = ( < )
    let ( ≤ ) = ( ≤ )
    let ( > ) = ( > )
    let ( ≥ ) = ( ≥ )
    let of_int n = n
    let to_int n = n
    let to_string = string_of_int
    let compare = compare
    let factorial = Combinatorics.factorial
  end

module type Count =
  sig
    type integer
    val diagrams : ?f : (integer → bool) → integer → integer → integer
    val diagrams_via_keystones : integer → integer → integer
    val keystones : integer list → integer
    val diagrams_per_keystone : integer → integer list → integer
  end

module Count (I : Integer) =
  struct
    let rcs = rcs_file
    let description = ["(still␣inoperational)␣phi^n␣topology"]

    type integer = I.t
    open I
    let two = of_int 2
    let three = of_int 3
```

If $I.t$ is an abstract datatype, the polymorphic *Pervasives.min* can fail. Provide our own version using the specific comparison "$(\leq)$".

```
    let min x y =
      if x ≤ y then
        x
      else
        y
```

Counting Diagrams for $\sum_n \lambda_n \phi^n$

Classes of diagrams are defined by the number of vertices and their degrees. We could use fixed size arrays, but we will use a map instead. For efficiency, we also maintain the number of external lines and the total number of propagators.

```
module IMap = Map.Make (struct type t = integer let compare = compare end)

type diagram_class = { ext : integer; prop : integer; v : integer IMap.t }
```

The numbers of external lines, propagators and vertices are determined by the degrees and multiplicities of vertices:

$$E(\{n_3, n_4, \ldots\}) = 2 + \sum_{d=3}^{\infty}(d-2)n_d \tag{3.10a}$$

$$P(\{n_3, n_4, \ldots\}) = \sum_{d=3}^{\infty} n_d - 1 = V(\{n_3, n_4, \ldots\}) - 1 \tag{3.10b}$$

$$V(\{n_3, n_4, \ldots\}) = \sum_{d=3}^{\infty} n_d \tag{3.10c}$$

```
let num_ext v =
  List.fold_left (fun sum (d, n) → sum + (d − two) × n) two v

let num_prop v =
  List.fold_left (fun sum (_, n) → sum + n) (zero − one) v
```

The sum of all vertex degrees must be equal to the number of propagator end points. This can be verified easily:

$$2P(\{n_3, n_4, \ldots\}) + E(\{n_3, n_4, \ldots\}) = \sum_{d=3}^{\infty} dn_d \tag{3.11}$$

```
let add_degree map (d, n) =
  if d < three then
    invalid_arg "add_degree: d < 3"
  else if n < zero then
    invalid_arg "add_degree: n <= 0"
  else if n = zero then
    map
  else
    IMap.add d n map

let create_class v =
  { ext = num_ext v;
    prop = num_prop v;
    v = List.fold_left add_degree IMap.empty v }

let multiplicity cl d =
  if d ≥ three then
    try
      IMap.find d cl.v
    with
    | Not_found → zero
  else
    invalid_arg "multiplicity: d < 3"
```

Remove one vertex of degree d, maintaining the invariants. Raises *Zero* if all vertices of degree d are exhausted.

```
exception Zero

let remove cl d =
  let n = pred (multiplicity cl d) in
  if n < zero then
    raise Zero
  else
    { ext = cl.ext − (d − two);
      prop = pred cl.prop;
      v = if n = zero then
        IMap.remove d cl.v
      else
        IMap.add d n cl.v }
```

Add one vertex of degree d, maintaining the invariants.

```
let add cl d =
  { ext = cl.ext + (d − two);
    prop = succ cl.prop;
    v = IMap.add d (succ (multiplicity cl d)) cl.v }
```

Count the number of diagrams. Any diagram can be obtained recursively either from a diagram with one ternary vertex less by insertion if a ternary vertex in an internal or external propagator or from a diagram with a higher order vertex that has its degree reduced by one:

$$\begin{aligned} D(\{n_3,n_4,\ldots\}) = \\ &(P(\{n_3-1,n_4,\ldots\})+E(\{n_3-1,n_4,\ldots\}))\,D(\{n_3-1,n_4,\ldots\}) \\ &\qquad+\sum_{d=4}^{\infty}(n_{d-1}+1)D(\{n_3,n_4,\ldots,n_{d-1}+1,n_d-1,\ldots\}) \end{aligned} \tag{3.12}$$

```
let rec class_size cl =
  if cl.ext = two ∨ cl.prop = zero then
    one
  else
    IMap.fold (fun d _ s → class_size_n cl d + s) cl.v (class_size_3 cl)
```

Purely ternary vertices recurse among themselves:

```
and class_size_3 cl =
  try
    let d′ = remove cl three in
    (d′.ext + d′.prop) × class_size d′
  with
  | Zero → zero
```

Vertices of higher degree recurse one step towards lower degrees:

```
and class_size_n cl d =
  if d > three then begin
    try
      let d′ = pred d in
      let cl′ = add (remove cl d) d′ in
```

```
        multiplicity cl' d' × class_size cl'
      with
      | Zero → zero
    end else
      zero
```

Find all $\{n_3, n_4, \ldots, n_d\}$ with

$$E(\{n_3, n_4, \ldots, n_d\}) - 2 = \sum_{i=3}^{c} l(i-2)n_i = sum \tag{3.13}$$

The implementation is a variant of *tuples* above.

```
let rec distribute_degrees' d sum =
  if d < three then
    invalid_arg "distribute_degrees"
  else if d = three then
    [[(d, sum)]]
  else
    distribute_degrees'' d sum (sum / (d − two))

and distribute_degrees'' d sum n =
  if n < zero then
    []
  else
    List.fold_left (fun ll l → ((d, n) :: l) :: ll)
      (distribute_degrees'' d sum (pred n))
      (distribute_degrees' (pred d) (sum − (d − two) × n))
```

Actually, we need to find all $\{n_3, n_4, \ldots, n_d\}$ with

$$E(\{n_3, n_4, \ldots, n_d\}) = sum \tag{3.14}$$

```
let distribute_degrees d sum = distribute_degrees' d (sum − two)
```

Finally we can count all diagrams by adding all possible ways of splitting the degrees of vertices. We can also count diagrams where *all* degrees satisfy a predicate f:

```
let diagrams ?(f = fun _ → true) deg n =
  List.fold_left (fun s d →
    if List.for_all (fun (d', n') → f d' ∨ n' = zero) d then
      s + class_size (create_class d)
    else
      s)
    zero (distribute_degrees deg n)
```

The next two are duplicated from *ThoList* and *Combinatorics*, in order to use the specific comparison functions.

```
let classify l =
  let rec add_to_class a = function
    | [] → [of_int 1, a]
    | (n, a') :: rest →
```

```
        if a = a' then
          (succ n, a) :: rest
        else
          (n, a') :: add_to_class a rest
    in
    let rec classify' cl = function
      | [] → cl
      | a :: rest → classify' (add_to_class a cl) rest
    in
    classify' [] l

  let permutation_symmetry l =
    List.fold_left (fun s (n, _) → factorial n × s) one (classify l)

  let symmetry l =
    let sum = List.fold_left (+) zero l in
    if List.exists (fun x → two × x = sum) l then
      two × permutation_symmetry l
    else
      permutation_symmetry l
```

The number of Feynman diagrams built of vertices with maximum degree $d_{\max}$ in a partition $N_{d,n} = \{n_1, n_2, \ldots, n_d\}$ with $n = n_1 + n_2 + \cdots + n_d$ and

$$\tilde{F}(d_{\max}, N_{d,n}) = \frac{n!}{|\mathcal{S}(N_{d,n})|\sigma(n_d, n)} \prod_{i=1}^{d} \frac{F(d_{\max}, n_i + 1)}{n_i!} \tag{3.15}$$

with $|\mathcal{S}(N)|$ the size of the symmetric group of N, $\sigma(n, 2n) = 2$ and $\sigma(n, m) = 1$ otherwise.

```
  let keystones p =
    let sum = List.fold_left (+) zero p in
    List.fold_left (fun acc n → acc / (factorial n)) (factorial sum) p
      / symmetry p

  let diagrams_per_keystone deg p =
    List.fold_left (fun acc n → acc × diagrams deg (succ n)) one p
```

We must find

$$F(d_{\max}, n) = \sum_{d=3}^{d_{\max}} \sum_{\substack{N=\{n_1,n_2,\ldots,n_d\} \\ n_1+n_2+\cdots+n_d=n \\ 1\le n_1\le n_2\le\cdots\le n_d\le\lfloor n/2\rfloor}} \tilde{F}(d_{\max}, N) \tag{3.16}$$

```
  let diagrams_via_keystones deg n =
    let module N = Nary (struct let max_arity = to_int (pred deg) end) in
    List.fold_left
      (fun acc p → acc + diagrams_per_keystone deg p × keystones p)
      zero (List.map (List.map of_int) (N.partitions (to_int n)))

end
```

3.2.6 Emulating HELAC

In [2], one leg is singled out:

```
module Helac (B : Tuple.Bound) =
  struct
    let rcs = RCS.rename rcs_file "Topology.Helac"
        ["phi**n topology, Helac style"]
    module Tuple = Tuple.Nary(B)

    type partition = int list
    let inspect_partition p = p

    let partition d sum =
      Partition.tuples d sum 1 (sum - d + 1)

    let rec partitions' d sum =
      let d' = pred d in
      if d' < 2 then
        []
      else
        List.map (fun p -> 1 :: p) (partition d' (pred sum)) @ partitions' d' sum

    let partitions sum = partitions' (succ B.max_arity) sum

    type 'a children = 'a Tuple.t

    let keystones' l =
      match l with
      | [] -> []
      | head :: tail ->
          [([head],
            ThoList.flatmap (fun p -> Combinatorics.partitions (List.tl p) tail)
              (partitions (List.length l)))]

    let keystones l =
      List.map (fun (bra, kets) -> (bra, List.map Tuple.of_list kets))
        (keystones' l)

    let max_subtree n = pred n
  end
```

The following is not tested, but it is no rocket science either . . .

```
module Helac_Binary =
  struct
    let rcs = RCS.rename rcs_file "Topology.Helac_Binary"
        ["phi**3 topology, Helac style"]

    type partition = int * int * int
    let inspect_partition (n1, n2, n3) = [n1; n2; n3]

    let partitions sum =
      List.map (fun (n2, n3) -> (1, n2, n3))
        (Partition.pairs (sum - 1) 1 (sum - 2))
```

```
type α children = α Tuple.Binary.t

let keystones' l =
  match l with
  | [] → []
  | head :: tail →
      [([head],
        ThoList.flatmap (fun (_, p2, _) → Combinatorics.split p2 tail)
          (partitions (List.length l)))]

let keystones l =
  List.map (fun (bra, kets) →
    (bra, List.map (fun (x, y) → Tuple.Binary.of2 x y) kets))
    (keystones' l)

let max_subtree n = pred n

end
```

—4— Directed Acyclical Graphs

4.1 Interface of DAG

This datastructure describes large collections of trees with many shared nodes. The sharing of nodes is semantically irrelevant, but can turn a factorial complexity to exponential complexity. Note that DAG implements only a very specialized subset of Directed Acyclical Graphs (DAGs).
If $T(n,D)$ denotes the set of all binary trees with root n encoded in D, while

$$O(n,D) = \{(e_1,n_1,n'_1),\ldots,(e_k,n_k,n'_k)\} \tag{4.1}$$

denotes the set of all *offspring* of n in D, and $\mathrm{tree}(e,t,t')$ denotes the binary tree formed by joining the binary trees t and t' with the label e, then

$$\begin{split}T(n,D) = \bigl\{\mathrm{tree}(e_i,t_i,t'_i) \,\big|\, (e_i,t_i,t'_i) \in \{e_1\}\times T(n_1,D)\times T(n'_1,D)\cup\ldots \\ \ldots\cup\{e_k\}\times T(n_k,D)\times T(n'_k,D)\bigr\}\end{split} \tag{4.2}$$

is the recursive definition of the binary trees encoded in D. It is obvious how this definitions translates to n-ary trees (including trees with mixed arity).

4.1.1 Forests

We require edges and nodes to be members of ordered sets. The sematics of *compare* are compatible with *Pervasives.compare*:

$$compare(x,y) = \begin{cases} -1 & \text{for } x<y \\ 0 & \text{for } x=y \\ 1 & \text{for } x>y \end{cases} \tag{4.3}$$

Note that this requirement does *not* exclude any trees. Even if we consider only topological equivalence classes with anonymous nodes, we can always construct a canonical labeling and order from the children of the nodes. However, if practical applications, we will often have more efficient labelings and orders at our disposal.

```
module type Ord =
  sig
    type t
    val compare : t → t → int
```

```
end
```

A forest F over a set of nodes and a set of edges is a map from the set of nodes N, to the direct product of the set of edges E and the power set 2^N of N augmented by a special element $\perp$ ("bottom").

$$
\begin{aligned}
F : N &\to (E \times 2^N) \cup \{\perp\} \\
n &\mapsto \begin{cases} (e, \{n'_1, n'_2, \ldots\}) \\ \perp \end{cases}
\end{aligned}
\tag{4.4}
$$

The nodes are ordered so that cycles can be detected

$$\forall n \in N : F(n) = (e, x) \Rightarrow \forall n' \in x : n > n' \tag{4.5}$$

A suitable function that exists for *all* forests is the depth of the tree beneath a node.

Nodes that are mapped to $\perp$ are called *leaf* nodes and nodes that do not appear in any $F(n)$ are called *root* nodes. There are as many trees in the forest as there are root nodes.

```
module type Forest =
  sig
    module Nodes : Ord
    type node = Nodes.t
    type edge
```

A subset $X \subset 2^N$ of the powerset of the set of nodes. The members of X can be be characterized by a fixed number of members (e. g. two for binary trees, as in QED). We can also have mixed arities (e. g. two and three for QCD) or even arbitrary arities. However, in most cases, the members of X will have at least two members.

```
    type children
```

This type abbreviation and order allow to apply the *Set.Make* functor to $E \times X$.

```
    type t = edge × children
    val compare : t → t → int
```

Test a predicate for *all* children.

```
    val for_all : (node → bool) → t → bool
```

fold f (_, children) acc will calculate

$$f(x_1, f(x_2, \cdots f(x_n, acc))) \tag{4.6}$$

where the *children* are $\{x_1, x_2, \ldots, x_n\}$. There are slightly more efficient alternatives for fixed arity (in particular binary), but we want to be general.

```
    val fold : (node → α → α) → t → α → α
  end
```

```
module Forest : functor (PT : Tuple.Poly) →
  functor (N : Ord) → functor (E : Ord) →
      Forest with module Nodes = N and type edge = E.t
      and type node = N.t and type children = N.t PT.t
```

4.1.2 DAGs

module type T =
  sig

    type *node*
    type *edge*

In the description of the function we assume for definiteness DAGs of binary trees with type *children* = *node* × *node*. However, we will also have implementations with type *children* = *node list* below.
Other possibilities include type *children* = *V3* of *node* × *node* | *V4* of *node* × *node* × *node*. There's probable never a need to use sets with logarithmic access, but it is easy to add.

    type *children*
    type *t*

The empty DAG.

    val *empty* : *t*

add_node n dag returns the DAG *dag* with the node *n*. If the node *n* already exists in *dag*, it is returned unchanged. Otherwise *n* is added without offspring.

    val *add_node* : *node* → *t* → *t*

add_offspring n (*e*, (*n1*, *n2*)) *dag* returns the DAG *dag* with the node *n* and its offspring *n1* and *n2* with edge label *e*. Each node can have an arbitrary number of offspring, but identical offspring are added only once. In order to prevent cycles, *add_offspring* requires both $n > n1$ and $n > n2$ in the given ordering. The nodes *n1* and *n2* are added as by *add_node*. NB: Adding all nodes *n1* and *n2*, even if they are sterile, is not strictly necessary for our applications. It even slows down the code by a few percent. But it is desirable for consistency and allows much more efficient *iter_nodes* and *fold_nodes* below.

    val *add_offspring* : *node* → *edge* × *children* → *t* → *t*
    exception *Cycle*

Just like *add_offspring*, but does not check for potential cycles.

    val *add_offspring_unsafe* : *node* → *edge* × *children* → *t* → *t*

is_node n dag returns true iff *n* is a node in *dag*.

    val *is_node* : *node* → *t* → *bool*

is_sterile n dag returns true iff *n* is a node in *dag* and boasts no offspring.

    val *is_sterile* : *node* → *t* → *bool*

is_offspring n (*e*, (*n1*, *n2*)) *dag* returns true iff *n1* and *n2* are offspring of *n* with label *e* in *dag*.

    val *is_offspring* : *node* → *edge* × *children* → *t* → *bool*

Note that the following functions can run into infinite recursion if the DAG given as argument contains cycles.
The usual functionals for processing all nodes (including sterile) ...

val $\mathit{iter\_nodes} : (\mathit{node} \to \mathit{unit}) \to t \to \mathit{unit}$
val $\mathit{map\_nodes} : (\mathit{node} \to \mathit{node}) \to t \to t$
val $\mathit{fold\_nodes} : (\mathit{node} \to \alpha \to \alpha) \to t \to \alpha \to \alpha$

... and all parent/offspring relations. Note that *map* requires *two* functions: one for the nodes and one for the edges and children. This is so because a change in the definition of node is *not* propagated automatically to where it is used as a child.

val $\mathit{iter} : (\mathit{node} \to \mathit{edge} \times \mathit{children} \to \mathit{unit}) \to t \to \mathit{unit}$
val $\mathit{map} : (\mathit{node} \to \mathit{node}) \to$
  $(\mathit{node} \to \mathit{edge} \times \mathit{children} \to \mathit{edge} \times \mathit{children}) \to t \to t$
val $\mathit{fold} : (\mathit{node} \to \mathit{edge} \times \mathit{children} \to \alpha \to \alpha) \to t \to \alpha \to \alpha$

Note that in it's current incarnation, *fold add_offspring dag empty* copies *only* the fertile nodes, while *fold add_offspring dag* (*fold_nodes add_node dag empty*) includes sterile ones, as does *map* (fun $n \to n$) (fun $n\ ec \to ec$) *dag*.

Return the DAG as a list of lists.

val $\mathit{lists} : t \to (\mathit{node} \times (\mathit{edge} \times \mathit{children})\ \mathit{list})\ \mathit{list}$

dependencies dag node returns a canonically sorted *Tree2.t* of all nodes reachable from *node*.

val $\mathit{dependencies} : t \to \mathit{node} \to (\mathit{node}, \mathit{edge})\ \mathit{Tree2.t}$

harvest dag n roots returns the DAG *roots* enlarged by all nodes in *dag* reachable from *n*.

val $\mathit{harvest} : t \to \mathit{node} \to t \to t$

harvest_list dag nlist returns the part of the DAG *dag* that is reachable from the nodes in *nlist*.

val $\mathit{harvest\_list} : t \to \mathit{node\ list} \to t$

size dag returns the number of nodes in the DAG *dag*.

val $\mathit{size} : t \to \mathit{int}$

eval f mul_edge mul_nodes add null unit root dag interprets the part of *dag* beneath *root* as an algebraic expression:

- each node is evaluated by $f : \mathit{node} \to \alpha$
- each set of children is evaluated by iterating the binary $\mathit{mul\_nodes} : \alpha \to \gamma \to \gamma$ on the values of the nodes, starting from $\mathit{unit}: \gamma$
- each offspring relation $(\mathit{node}, (\mathit{edge}, \mathit{children}))$ is evaluated by applying $\mathit{mul\_edge} : \mathit{node} \to \mathit{edge} \to \gamma \to \delta$ to *node*, *edge* and the evaluation of *children*.
- all offspring relations of a *node* are combined by iterating the binary $\mathit{add} : \delta \to \alpha \to \alpha$ starting from $\mathit{null} : \alpha$

In our applications, we will always have $\alpha = \gamma = \delta$, but the more general type is useful for documenting the relationships. The memoizing variant *eval_memoized f mul_edge mul_nodes add null unit root dag* requires some overhead, but can be more efficient for complex operations.

$$\begin{array}{l} \mathsf{val}\ eval : (node \to \alpha) \to (node \to edge \to \gamma \to \delta) \to \\ \quad (\alpha \to \gamma \to \gamma) \to (\delta \to \alpha \to \alpha) \to \alpha \to \gamma \to node \to t \to \alpha \\ \mathsf{val}\ eval\_memoized : (node \to \alpha) \to (node \to edge \to \gamma \to \delta) \to \\ \quad (\alpha \to \gamma \to \gamma) \to (\delta \to \alpha \to \alpha) \to \alpha \to \gamma \to node \to t \to \alpha \end{array}$$

forest root dag expands the *dag* beneath *root* into the equivalent list of trees *Tree.t*. *children* are represented as list of nodes.

A sterile node n is represented as $Tree.Leaf\ ((n,\ None),\ n)$, cf. page 806. There might be a better way, but we need to change the interface and semantics of *Tree* for this.

$$\begin{array}{l} \mathsf{val}\ forest : node \to t \to (node \times edge\ option, node)\ Tree.t\ list \\ \mathsf{val}\ forest\_memoized : node \to t \to (node \times edge\ option, node)\ Tree.t\ list \end{array}$$

count_trees n dag returns the number of trees with root n encoded in the DAG *dag*, i. e. $|T(n,D)|$. NB: the current implementation is very naive and can take a *very* long time for moderately sized DAGs that encode a large set of trees.

$$\mathsf{val}\ count\_trees : node \to t \to int$$

$$\begin{array}{l} \mathsf{val}\ rcs : RCS.t \\ \mathsf{end} \end{array}$$

$$\begin{array}{l} \mathsf{module}\ Make\ (F : Forest) : \\ \quad T\ \mathsf{with\ type}\ node = F.node\ \mathsf{and\ type}\ edge = F.edge \\ \quad \mathsf{and\ type}\ children = F.children \end{array}$$

4.1.3 Graded Sets, Forests & DAGs

A graded ordered[1] set is an ordered set with a map into another ordered set (often the non-negative integers). The grading does not necessarily respect the ordering.

$$\begin{array}{l} \mathsf{module\ type}\ Graded\_Ord = \\ \quad \mathsf{sig} \\ \qquad \mathsf{include}\ Ord \\ \qquad \mathsf{module}\ G : Ord \\ \qquad \mathsf{val}\ rank : t \to G.t \\ \quad \mathsf{end} \end{array}$$

For all ordered sets, there are two canonical gradings: a *Chaotic* grading that assigns the same rank (e. g. *unit*) to all elements and the *Discrete* grading that uses the identity map as grading.

$$\begin{array}{l} \mathsf{module\ type}\ Grader = \mathsf{functor}\ (O : Ord) \to Graded\_Ord\ \mathsf{with\ type}\ t = O.t \\ \mathsf{module}\ Chaotic : Grader \\ \mathsf{module}\ Discrete : Grader \end{array}$$

A graded forest is just a forest in which the nodes form a graded ordered set.

[1] We don't appear to have use for graded unordered sets.

There doesn't appear to be a nice syntax for avoiding the repetition here. Fortunately, the signature is short ...

```
module type Graded_Forest =
  sig
    module Nodes : Graded_Ord
    type node = Nodes.t
    type edge
    type children
    type t = edge × children
    val compare : t → t → int
    val for_all : (node → bool) → t → bool
    val fold : (node → α → α) → t → α → α
  end

module type Forest_Grader = functor (G : Grader) → functor (F : Forest) →

  Graded_Forest with type Nodes.t = F.node
  and type node = F.node
  and type edge = F.edge
  and type children = F.children
  and type t = F.t

module Grade_Forest : Forest_Grader
```

Finally, a graded DAG is a DAG in which the nodes form a graded ordered set and the subsets with a given rank can be accessed cheaply.

```
module type Graded =
  sig
    include T
    type rank
    val rank : node → rank
    val ranks : t → rank list
    val min_max_rank : t → rank × rank
    val ranked : rank → t → node list
  end

module Graded (F : Graded_Forest) :
    Graded with type node = F.node and type edge = F.edge
    and type children = F.children and type rank = F.Nodes.G.t
```

4.2 Implementation of DAG

```
let rcs_file = RCS.parse "DAG" ["Directed␣Acyclical␣Graph"]
    { RCS.revision = "$Revision:␣6465␣$";
      RCS.date = "$Date:␣2015-01-10␣16:22:31␣+0100␣(Sat,␣10␣Jan␣2015)␣$";
      RCS.author = "$Author:␣jr_reuter␣$";
      RCS.source
        = "$URL:␣svn+ssh://jr_reuter@login.hepforge.org/hepforge/svn/whizard/trunk/omega/sr

module type Ord =
```

```
  sig
    type t
    val compare : t → t → int
  end

module type Forest =
  sig
    module Nodes : Ord
    type node = Nodes.t
    type edge
    type children
    type t = edge × children
    val compare : t → t → int
    val for_all : (node → bool) → t → bool
    val fold : (node → α → α) → t → α → α
  end

module type T =
  sig
    type node
    type edge
    type children
    type t
    val empty : t
    val add_node : node → t → t
    val add_offspring : node → edge × children → t → t
    exception Cycle
    val add_offspring_unsafe : node → edge × children → t → t
    val is_node : node → t → bool
    val is_sterile : node → t → bool
    val is_offspring : node → edge × children → t → bool
    val iter_nodes : (node → unit) → t → unit
    val map_nodes : (node → node) → t → t
    val fold_nodes : (node → α → α) → t → α → α
    val iter : (node → edge × children → unit) → t → unit
    val map : (node → node) →
      (node → edge × children → edge × children) → t → t
    val fold : (node → edge × children → α → α) → t → α → α
    val lists : t → (node × (edge × children) list) list
    val dependencies : t → node → (node, edge) Tree2.t
    val harvest : t → node → t → t
    val harvest_list : t → node list → t
    val size : t → int
    val eval : (node → α) → (node → edge → γ → δ) →
      (α → γ → γ) → (δ → α → α) → α → γ → node → t → α
    val eval_memoized : (node → α) → (node → edge → γ → δ) →
      (α → γ → γ) → (δ → α → α) → α → γ → node → t → α
    val forest : node → t → (node × edge option, node) Tree.t list
    val forest_memoized : node → t → (node × edge option, node) Tree.t list
    val count_trees : node → t → int
    val rcs : RCS.t
```

```
  end

module type Graded_Ord =
  sig
    include Ord
    module G : Ord
    val rank : t → G.t
  end

module type Grader = functor (O : Ord) → Graded_Ord with type t = O.t

module type Graded_Forest =
  sig
    module Nodes : Graded_Ord
    type node = Nodes.t
    type edge
    type children
    type t = edge × children
    val compare : t → t → int
    val for_all : (node → bool) → t → bool
    val fold : (node → α → α) → t → α → α
  end

module type Forest_Grader = functor (G : Grader) → functor (F : Forest) →

  Graded_Forest with type Nodes.t = F.node
  and type node = F.node
  and type edge = F.edge
  and type children = F.children
  and type t = F.t
```

4.2.1 The Forest Functor

```
module Forest (PT : Tuple.Poly) (N : Ord) (E : Ord) :
    Forest with module Nodes = N and type edge = E.t
    and type node = N.t and type children = N.t PT.t =
  struct
    module Nodes = N
    type edge = E.t
    type node = N.t
    type children = node PT.t
    type t = edge × children

    let compare (e1, n1) (e2, n2) =
      let c = PT.compare N.compare n1 n2 in
      if c ≠ 0 then
        c
      else
        E.compare e1 e2

    let for_all f (_, nodes) = PT.for_all f nodes
    let fold f (_, nodes) acc = PT.fold_right f nodes acc
```

```
end
```

4.2.2 Gradings

```
module Chaotic (O : Ord) =
  struct
    include O
    module G =
      struct
        type t = unit
        let compare _ _ = 0
      end
    let rank _ = ()
  end

module Discrete (O : Ord) =
  struct
    include O
    module G = O
    let rank x = x
  end

module Fake_Grading (O : Ord) =
  struct
    include O
    exception Impossible of string
    module G =
      struct
        type t = unit
        let compare _ _ = raise (Impossible "G.compare")
      end
    let rank _ = raise (Impossible "G.compare")
  end

module Grade_Forest (G : Grader) (F : Forest) =
  struct
    module Nodes = G(F.Nodes)
    type node = Nodes.t
    type edge = F.edge
    type children = F.children
    type t = F.t
    let compare = F.compare
    let for_all = F.for_all
    let fold = F.fold
  end
```

The following can easily be extended to *Map.S* in its full glory, if we ever need it.

```
module type Graded_Map =
```

```
  sig
    type key
    type rank
    type α t
    val empty : α t
    val add : key → α → α t → α t
    val find : key → α t → α
    val mem : key → α t → bool
    val iter : (key → α → unit) → α t → unit
    val fold : (key → α → β → β) → α t → β → β
    val ranks : α t → rank list
    val min_max_rank : α t → rank × rank
    val ranked : rank → α t → key list
  end

module type Graded_Map_Maker = functor (O : Graded_Ord) →
  Graded_Map with type key = O.t and type rank = O.G.t

module Graded_Map (O : Graded_Ord) :
    Graded_Map with type key = O.t and type rank = O.G.t =
  struct
    module M1 = Map.Make(O.G)
    module M2 = Map.Make(O)

    type key = O.t
    type rank = O.G.t

    type (+α) t = α M2.t M1.t

    let empty = M1.empty
    let add key data map1 =
      let rank = O.rank key in
      let map2 = try M1.find rank map1 with Not_found → M2.empty in
      M1.add rank (M2.add key data map2) map1
    let find key map = M2.find key (M1.find (O.rank key) map)
    let mem key map =
      M2.mem key (try M1.find (O.rank key) map with Not_found → M2.empty)
    let iter f map1 = M1.iter (fun rank → M2.iter f) map1
    let fold f map1 acc1 = M1.fold (fun rank → M2.fold f) map1 acc1
```

The set of ranks and its minimum and maximum should be maintained explicitely!

```
    module S1 = Set.Make(O.G)
    let ranks map = M1.fold (fun key data acc → key :: acc) map []
    let rank_set map = M1.fold (fun key data → S1.add key) map S1.empty
    let min_max_rank map =
      let s = rank_set map in
      (S1.min_elt s, S1.max_elt s)

    module S2 = Set.Make(O)
    let keys map = M2.fold (fun key data acc → key :: acc) map []
    let sorted_keys map =
```

```
      S2.elements (M2.fold (fun key data → S2.add key) map S2.empty)
    let ranked rank map =
      keys (try M1.find rank map with Not_found → M2.empty)
  end
```

4.2.3 The DAG Functor

```
module Maybe_Graded (GMM : Graded_Map_Maker) (F : Graded_Forest) =
  struct
    let rcs = RCS.rename rcs_file "DAG.Graded()"
        ["Graded directed Acyclical Graph ";
         "representing binary or n-ary trees"]

    module G = F.Nodes.G

    type node = F.node
    type rank = G.t
    type edge = F.edge
    type children = F.children
```

If we get tired of graded DAGs, we just have to replace *Graded_Map* by *Map* here and remove *ranked* below and gain a tiny amount of simplicity and efficiency.

```
    module Parents = GMM(F.Nodes)
    module Offspring = Set.Make(F)

    type t = Offspring.t Parents.t

    let rank = F.Nodes.rank
    let ranks = Parents.ranks
    let min_max_rank = Parents.min_max_rank
    let ranked = Parents.ranked

    let empty = Parents.empty

    let add_node node dag =
      if Parents.mem node dag then
        dag
      else
        Parents.add node Offspring.empty dag

    let add_offspring_unsafe node offspring dag =
      let offsprings =
        try Parents.find node dag with Not_found → Offspring.empty in
      Parents.add node (Offspring.add offspring offsprings)
        (F.fold add_node offspring dag)

    exception Cycle

    let add_offspring node offspring dag =
      if F.for_all (fun n → F.Nodes.compare n node < 0) offspring then
        add_offspring_unsafe node offspring dag
      else
        raise Cycle
```

```
let is_node node dag =
  Parents.mem node dag

let is_sterile node dag =
  try
    Offspring.is_empty (Parents.find node dag)
  with
  | Not_found → false

let is_offspring node offspring dag =
  try
    Offspring.mem offspring (Parents.find node dag)
  with
  | Not_found → false

let iter_nodes f dag =
  Parents.iter (fun n _ → f n) dag

let iter f dag =
  Parents.iter (fun node → Offspring.iter (f node)) dag

let map_nodes f dag =
  Parents.fold (fun n → Parents.add (f n)) dag Parents.empty

let map fn fo dag =
  Parents.fold (fun node offspring →
    Parents.add (fn node)
      (Offspring.fold (fun o → Offspring.add (fo node o))
         offspring Offspring.empty)) dag Parents.empty

let fold_nodes f dag acc =
  Parents.fold (fun n _ → f n) dag acc

let fold f dag acc =
  Parents.fold (fun node → Offspring.fold (f node)) dag acc
```

Note that in it's current incarnation, *fold add_offspring dag empty* copies *only* the fertile nodes, while *fold add_offspring dag* (*fold_nodes add_node dag empty*) includes sterile ones, as does *map* (fun *n* → *n*) (fun *n ec* → *ec*) *dag*.

```
let dependencies dag node =
  let rec dependencies' node' =
    let offspring = Parents.find node' dag in
    if Offspring.is_empty offspring then
      Tree2.leaf node'
    else
      Tree2.cons
        (Offspring.fold
           (fun o acc →
              (fst o,
               node',
               F.fold (fun wf acc' → dependencies' wf :: acc') o []) :: acc)
           offspring [])
  in
```

```
    dependencies' node

let lists dag =
  Sort.list (fun (n1, _) (n2, _) -> F.Nodes.compare n1 n2 <= 0)
    (Parents.fold (fun node offspring l ->
      (node, Offspring.elements offspring) :: l) dag [])

let size dag =
  Parents.fold (fun _ _ n -> succ n) dag 0

let rec harvest dag node roots =
  Offspring.fold
    (fun offspring roots' ->
      if is_offspring node offspring roots' then
        roots'
      else
        F.fold (harvest dag)
          offspring (add_offspring_unsafe node offspring roots'))
    (Parents.find node dag) (add_node node roots)

let harvest_list dag nodes =
  List.fold_left (fun roots node -> harvest dag node roots) empty nodes
```

Build a closure once, so that we can recurse faster:

```
let eval f mule muln add null unit node dag =
  let rec eval' n =
    if is_sterile n dag then
      f n
    else
      Offspring.fold
        (fun (e, _ as offspring) v0 ->
          add (mule n e (F.fold muln' offspring unit)) v0)
        (Parents.find n dag) null
  and muln' n = muln (eval' n) in
  eval' node

let count_trees node dag =
  eval (fun _ -> 1) (fun _ _ p -> p) ( * ) (+) 0 1 node dag

let build_forest evaluator node dag =
  evaluator (fun n -> [Tree.leaf (n, None) n])
    (fun n e p -> List.map (fun p' -> Tree.cons (n, Some e) p') p)
    (fun p1 p2 -> Product.fold2 (fun n nl pl -> (n :: nl) :: pl) p1 p2 [])
    (@) [] [[]] node dag

let forest = build_forest eval
```

At least for *count_trees*, the memoizing variant *eval_memoized* is considerably slower than direct recursive evaluation with *eval*.

```
let eval_offspring f mule muln add null unit dag values (node, offspring) =
  let muln' n = muln (Parents.find n values) in
  let v =
    if is_sterile node dag then
      f node
```

```
        else
          Offspring.fold
            (fun (e, _ as offspring) v0 →
              add (mule node e (F.fold muln′ offspring unit)) v0)
            offspring null
      in
      (v, Parents.add node v values)

    let eval_memoized′ f mule muln add null unit dag =
      let result, _ =
        List.fold_left
          (fun (v, values) → eval_offspring f mule muln add null unit dag values)
          (null, Parents.empty)
          (Sort.list (fun (n1, _) (n2, _) → F.Nodes.compare n1 n2 ≤ 0)
             (Parents.fold
                (fun node offspring l → (node, offspring) :: l) dag [])) in
      result

    let eval_memoized f mule muln add null unit node dag =
      eval_memoized′ f mule muln add null unit
        (harvest dag node empty)

    let forest_memoized = build_forest eval_memoized

  end

module type Graded =
  sig
    include T
    type rank
    val rank : node → rank
    val ranks : t → rank list
    val min_max_rank : t → rank × rank
    val ranked : rank → t → node list
  end

module Graded (F : Graded_Forest) = Maybe_Graded(Graded_Map)(F)
```

The following is not a graded map, obviously. But it can pass as one by the typechecker for constructing non-graded DAGs.

```
module Fake_Graded_Map (O : Graded_Ord) :
    Graded_Map with type key = O.t and type rank = O.G.t =
  struct
    module M = Map.Make(O)
    type key = O.t
    type (+α) t = α M.t
    let empty = M.empty
    let add = M.add
    let find = M.find
    let mem = M.mem
    let iter = M.iter
    let fold = M.fold
```

We make sure that the remaining three are never called inside *DAG* and are not visible outside.

```
    type rank = O.G.t
    exception Impossible of string
    let ranks _ = raise (Impossible "ranks")
    let min_max_rank _ = raise (Impossible "min_max_rank")
    let ranked _ _ = raise (Impossible "ranked")
  end
```

We could also have used signature projection with a chaotic or discrete grading, but the *Graded_Map* can cost some efficiency. This is probably not the case for the current simple implementation, but future embellishment can change this. Therefore, the ungraded DAG uses *Map* directly, without overhead.

```
module Make (F : Forest) =
  Maybe_Graded(Fake_Graded_Map)(Grade_Forest(Fake_Grading)(F))
```

If O'Caml had *polymorphic recursion*, we could think of even more elegant implementations unifying nodes and offspring (cf. the generalized tries in [4]).

—5— Momenta

5.1 Interface of Momentum

Model the finite combinations

$$p = \sum_{n=1}^{k} c_k \bar p_n, \qquad (\text{with } c_k \in \{0,1\}) \tag{5.1}$$

of n_{in} incoming and $k - n_{\text{in}}$ outgoing momenta p_n

$$\bar p_n = \begin{cases} -p_n & \text{for } 1 \le n \le n_{\text{in}} \\ p_n & \text{for } n_{\text{in}} + 1 \le n \le k \end{cases} \tag{5.2}$$

where momentum is conserved

$$\sum_{n=1}^{k} \bar p_n = 0 \tag{5.3}$$

below, we need the notion of 'rank' and 'dimension':

$$\mathit{dim}(p) = k \tag{5.4a}$$

$$\mathit{rank}(p) = \sum_{n=1}^{k} c_k \tag{5.4b}$$

where 'dimension' is *not* the dimension of the underlying space-time, of course.

```
module type T =
  sig
    type t
```

Constructor: $(k, N) \to p = \sum_{n\in N} \bar p_n$ and $k = \mathit{dim}(p)$ is the *overall* number of independent momenta, while $\mathit{rank}(p) = |N|$ is the number of momenta in p. It would be possible to fix *dim* as a functor argument instead. This might be slightly faster and allow a few more compile time checks, but would be much more tedious to use, since the number of particles will be chosen at runtime.

```
    val of_ints : int → int list → t
```

No two indices may be the same. Implementions of *of_ints* can either raise the exception *Duplicate* or ignore the duplicate, but implementations of *add* are required to raise *Duplicate*.

exception *Duplicate* of *int*

Raise *Range* iff $n > k$:

exception *Range* of *int*

Binary oparations require that both momenta have the same dimension. *Mismatch* is raised if this condition is violated.

exception *Mismatch* of $string \times t \times t$

Negative is raised if the result of *sub* is undefined.

exception *Negative*

The inverses of the constructor (we have $rank\ p = List.length\ (to\_ints\ p)$, but *rank* might be more efficient):

val $to\_ints : t \rightarrow int\ list$
val $dim : t \rightarrow int$
val $rank : t \rightarrow int$

Shortcuts: $singleton\ d\ p = of\_ints\ d\ [p]$ and $zero\ d = of\_ints\ d\ [\,]$:

val $singleton : int \rightarrow int \rightarrow t$
val $zero : int \rightarrow t$

An arbitrary total order, with the condition $rank(p_1) < rank(p_2) \Rightarrow p_1 < p_2$.

val $compare : t \rightarrow t \rightarrow int$

Use momentum conservation to construct the negative momentum with positive coefficients:

val $neg : t \rightarrow t$

Return the momentum or its negative, whichever has the lower rank. NB: the present implementation does *not* guarantee that

$$\text{abs}p = \text{abs}q \Longleftrightarrow p = p \vee p = -q \tag{5.5}$$

for momenta with rank = dim/2.

val $abs : t \rightarrow t$

Add and subtract momenta. This can fail, since the coefficients c_k must me either 0 or 1.

val $add : t \rightarrow t \rightarrow t$
val $sub : t \rightarrow t \rightarrow t$

Once more, but not raising exceptions this time:

val $try\_add : t \rightarrow t \rightarrow t\ option$
val $try\_sub : t \rightarrow t \rightarrow t\ option$

Not the total order provided by *compare*, but set inclusion of non-zero coefficients instead:

val $less : t \rightarrow t \rightarrow bool$
val $lesseq : t \rightarrow t \rightarrow bool$

$p_1 + (\pm p_2) + (\pm p_3) = 0$

```
val try_fusion : t → t → t → (bool × bool) option
```

A textual representation for debugging:

```
val to_string : t → string
```

split i n p splits $\bar{p}_i$ into n momenta $\bar{p}_i \to \bar{p}_i + \bar{p}_{i+1} + \ldots + \bar{p}_{i+n-1}$ and makes room via $\bar{p}_{j>i} \to \bar{p}_{j+n-1}$. This is used for implementating cascade decays, like combining

$$\mathrm{e}^+(p_1)\mathrm{e}^-(p_2) \to \mathrm{W}^-(p_3)\nu_{\mathrm{e}}(p_4)\mathrm{e}^+(p_5) \tag{5.6a}$$

$$\mathrm{W}^-(p_3) \to \mathrm{d}(p_3')\bar{\mathrm{u}}(p_4') \tag{5.6b}$$

to

$$\mathrm{e}^+(p_1)\mathrm{e}^-(p_2) \to \mathrm{d}(p_3)\bar{\mathrm{u}}(p_4)\nu_{\mathrm{e}}(p_5)\mathrm{e}^+(p_6) \tag{5.7}$$

in narrow width approximation for the W^-.

```
val split : int → int → t → t
```

5.1.1 Scattering Kinematics

From here on, we assume scattering kinematics $\{1,2\} \to \{3,4,\ldots\}$, i.e. $n_{\mathrm{in}} = 2$.

Since functions like *timelike* can be used for decays as well (in which case they must *always* return true, the representation—and consequently the constructors—should be extended by a flag discriminating between the two cases!

```
module Scattering :
  sig
```

Test if the momentum is an incoming one: $p = \bar{p}_1 \lor p = \bar{p}_2$

```
    val incoming : t → bool
```

$p = \bar{p}_3 \lor p = \bar{p}_4 \lor \ldots$

```
    val outgoing : t → bool
```

$p^2 \geq 0$. NB: *par abus de langange*, we report the incoming individual momenta as spacelike, instead as timelike. This will be useful for phasespace constructions below.

```
    val timelike : t → bool
```

$p^2 \leq 0$. NB: the simple algebraic criterion can be violated for heavy initial state particles.

```
    val spacelike : t → bool
```

$p = \bar{p}_1 + \bar{p}_2$

```
    val s_channel_in : t → bool
```

$p = \bar{p}_3 + \bar{p}_4 + \ldots + \bar{p}_n$

```
val s_channel_out : t -> bool
```

$p = \bar p_1 + \bar p_2 \lor p = \bar p_3 + \bar p_4 + \ldots + \bar p_n$

```
val s_channel : t -> bool
```

$\bar p_1 + \bar p_2 \to \bar p_3 + \bar p_4 + \ldots + \bar p_n$

```
  val flip_s_channel_in : t -> t
end
```

5.1.2 Decay Kinematics

```
module Decay :
  sig
```

Test if the momentum is an incoming one: $p = \bar p_1$

```
val incoming : t -> bool
```

$p = \bar p_2 \lor p = \bar p_3 \lor \ldots$

```
val outgoing : t -> bool
```

$p^2 \geq 0$. NB: here, we report the incoming individual momenta as timelike.

```
val timelike : t -> bool
```

$p^2 \leq 0$.

```
val spacelike : t -> bool
```

```
    end
  val rcs : RCS.t
end
```

```
module Lists : T
module Bits : T
module Default : T
```

Wolfgang's funny tree codes:

$$(2^n, 2^{n-1}) \to (1, 2, 4, \ldots, 2^{n-2}) \tag{5.8}$$

```
module type Whizard =
  sig
    type t
    val of_momentum : t -> int
    val to_momentum : int -> int -> t
  end
```

```
module ListsW : Whizard with type t = Lists.t
module BitsW : Whizard with type t = Bits.t
module DefaultW : Whizard with type t = Default.t
```

5.2 Implementation of Momentum

```
let rcs_file = RCS.parse "Momentum" ["Finite␣disjoint␣sums␣of␣momenta"]
    { RCS.revision = "$Revision:␣6465␣$";
      RCS.date = "$Date:␣2015-01-10␣16:22:31␣+0100␣(Sat,␣10␣Jan␣2015)␣$";
      RCS.author = "$Author:␣jr_reuter␣$";
      RCS.source
        = "$URL:␣svn+ssh://jr_reuter@login.hepforge.org/hepforge/svn/whizard/trunk/omega/s

module type T =
  sig
    type t
    val of_ints : int → int list → t
    exception Duplicate of int
    exception Range of int
    exception Mismatch of string × t × t
    exception Negative
    val to_ints : t → int list
    val dim : t → int
    val rank : t → int
    val singleton : int → int → t
    val zero : int → t
    val compare : t → t → int
    val neg : t → t
    val abs : t → t
    val add : t → t → t
    val sub : t → t → t
    val try_add : t → t → t option
    val try_sub : t → t → t option
    val less : t → t → bool
    val lesseq : t → t → bool
    val try_fusion : t → t → t → (bool × bool) option
    val to_string : t → string
    val split : int → int → t → t
    module Scattering :
        sig
          val incoming : t → bool
          val outgoing : t → bool
          val timelike : t → bool
          val spacelike : t → bool
          val s_channel_in : t → bool
          val s_channel_out : t → bool
          val s_channel : t → bool
          val flip_s_channel_in : t → t
        end
    module Decay :
        sig
          val incoming : t → bool
          val outgoing : t → bool
          val timelike : t → bool
```

```
        val spacelike : t → bool
      end
    val rcs : RCS.t
  end
```

5.2.1 Lists of Integers

The first implementation (as part of *Fusion*) was based on sorted lists, because I did not want to preclude the use of more general indices that integers. However, there's probably not much use for this generality (the indices are typically generated automatically and integer are the most natural choice) and it is no longer supported. by the current signature. Thus one can also use the more efficient implementation based on bitvectors below.

```
module Lists =
  struct
    let rcs = RCS.rename rcs_file "Momentum.Lists()"
        (RCS.description rcs_file @
         ["using lists as representation."])

    type t = { d : int; r : int; p : int list }

    exception Range of int
    exception Duplicate of int

    let rec check d = function
      | p1 :: p2 :: _ when p2 ≤ p1 → raise (Duplicate p1)
      | p1 :: (p2 :: _ as rest) → check d rest
      | [p] when p < 1 ∨ p > d → raise (Range p)
      | [p] → ()
      | [] → ()

    let of_ints d p =
      let p' = List.sort compare p in
      check d p';
      { d = d; r = List.length p; p = p' }

    let to_ints p = p.p
    let dim p = p.d
    let rank p = p.r
    let zero d = { d = d; r = 0; p = [] }
    let singleton d p = { d = d; r = 1; p = [p] }

    let to_string p =
      "[" ^ String.concat "," (List.map string_of_int p.p) ^
      "/" ^ string_of_int p.r ^ "/" ^ string_of_int p.d ^ "]"

    exception Mismatch of string × t × t
    let mismatch s p1 p2 = raise (Mismatch (s, p1, p2))

    let matching f s p1 p2 =
      if p1.d = p2.d then
        f p1 p2
      else
```

```
    mismatch s p1 p2

let compare p1 p2 =
  if p1.d = p2.d then begin
    let c = compare p1.r p2.r in
    if c ≠ 0 then
      c
    else
      compare p1.p p2.p
  end else
    mismatch "compare" p1 p2

let rec neg' d i = function
  | [] →
      if i ≤ d then
        i :: neg' d (succ i) []
      else
        []
  | i' :: rest as p →
      if i' > d then
        failwith "Integer_List.neg:␣internal␣error"
      else if i' = i then
        neg' d (succ i) rest
      else
        i :: neg' d (succ i) p

let neg p = { d = p.d; r = p.d − p.r; p = neg' p.d 1 p.p }

let abs p =
  if 2 × p.r > p.d then
    neg p
  else
    p

let rec add' p1 p2 =
  match p1, p2 with
  | [], p → p
  | p, [] → p
  | x1 :: p1', x2 :: p2' →
      if x1 < x2 then
        x1 :: add' p1' p2
      else if x2 < x1 then
        x2 :: add' p1 p2'
      else
        raise (Duplicate x1)

let add p1 p2 =
  if p1.d = p2.d then
    { d = p1.d; r = p1.r + p2.r; p = add' p1.p p2.p }
  else
    mismatch "add" p1 p2

let rec try_add' d r acc p1 p2 =
  match p1, p2 with
```

```
  | [], p → Some ({ d = d; r = r; p = List.rev_append acc p })
  | p, [] → Some ({ d = d; r = r; p = List.rev_append acc p })
  | x1 :: p1', x2 :: p2' →
      if x1 < x2 then
        try_add' d r (x1 :: acc) p1' p2
      else if x2 < x1 then
        try_add' d r (x2 :: acc) p1 p2'
      else
        None

let try_add p1 p2 =
  if p1.d = p2.d then
    try_add' p1.d (p1.r + p2.r) [] p1.p p2.p
  else
    mismatch "try_add" p1 p2

exception Negative

let rec sub' p1 p2 =
  match p1, p2 with
  | p, [] → p
  | [], _ → raise Negative
  | x1 :: p1', x2 :: p2' →
      if x1 < x2 then
        x1 :: sub' p1' p2
      else if x1 = x2 then
        sub' p1' p2'
      else
        raise Negative

let rec sub p1 p2 =
  if p1.d = p2.d then begin
    if p1.r ≥ p2.r then
      { d = p1.d; r = p1.r − p2.r; p = sub' p1.p p2.p }
    else
      neg (sub p2 p1)
   end else
    mismatch "sub" p1 p2

let rec try_sub' d r acc p1 p2 =
  match p1, p2 with
  | p, [] → Some ({ d = d; r = r; p = List.rev_append acc p })
  | [], _ → None
  | x1 :: p1', x2 :: p2' →
      if x1 < x2 then
        try_sub' d r (x1 :: acc) p1' p2
      else if x1 = x2 then
        try_sub' d r acc p1' p2'
      else
        None

let try_sub p1 p2 =
  if p1.d = p2.d then begin
```

```
    if p1.r ≥ p2.r then
      try_sub' p1.d (p1.r − p2.r) [] p1.p p2.p
    else
      match try_sub' p1.d (p2.r − p1.r) [] p2.p p1.p with
      | None → None
      | Some p → Some (neg p)
  end else
    mismatch "try_sub" p1 p2

let rec less' equal p1 p2 =
  match p1, p2 with
  | [], [] → ¬ equal
  | [], _ → true
  | x1 :: _ , [] → false
  | x1 :: p1', x2 :: p2' when x1 = x2 → less' equal p1' p2'
  | x1 :: p1', x2 :: p2' → less' false p1 p2'

let less p1 p2 =
  if p1.d = p2.d then
    less' true p1.p p2.p
  else
    mismatch "sub" p1 p2

let rec lesseq' p1 p2 =
  match p1, p2 with
  | [], _ → true
  | x1 :: _ , [] → false
  | x1 :: p1', x2 :: p2' when x1 = x2 → lesseq' p1' p2'
  | x1 :: p1', x2 :: p2' → lesseq' p1 p2'

let lesseq p1 p2 =
  if p1.d = p2.d then
    lesseq' p1.p p2.p
  else
    mismatch "lesseq" p1 p2

module Scattering =
  struct

    let incoming p =
      if p.r = 1 then
        match p.p with
        | [1] | [2] → true
        | _ → false
      else
        false

    let outgoing p =
      if p.r = 1 then
        match p.p with
        | [1] | [2] → false
        | _ → true
      else
        false
```

```
    let s_channel_in p =
      match p.p with
      | [1; 2] → true
      | _ → false

    let rec s_channel_out' d i = function
      | [] → i = succ d
      | i' :: p when i' = i → s_channel_out' d (succ i) p
      | _ → false

    let s_channel_out p =
      match p.p with
      | 3 :: p' → s_channel_out' p.d 4 p'
      | _ → false

    let s_channel p = s_channel_in p ∨ s_channel_out p

    let timelike p =
      match p.p with
      | p1 :: p2 :: _ → p1 > 2 ∨ (p1 = 1 ∧ p2 = 2)
      | p1 :: _ → p1 > 2
      | [] → false

    let spacelike p = ¬ (timelike p)

    let flip_s_channel_in p =
      if s_channel_in p then
        neg (of_ints p.d [1;2])
      else
        p

  end

module Decay =
  struct

    let incoming p =
      if p.r = 1 then
        match p.p with
        | [1] → true
        | _ → false
      else
        false

    let outgoing p =
      if p.r = 1 then
        match p.p with
        | [1] → false
        | _ → true
      else
        false

    let timelike p =
      match p.p with
      | [1] → true
      | p1 :: _ → p1 > 1
```

```
      | [] → false
    let spacelike p = ¬ (timelike p)
  end

let test_sum p inv1 p1 inv2 p2 =
  if p.d = p1.d then begin
    if p.d = p2.d then begin
      match (if inv1 then try_add else try_sub) p p1 with
      | None → false
      | Some p′ →
          begin match (if inv2 then try_add else try_sub) p′ p2 with
          | None → false
          | Some p″ → p″.r = 0 ∨ p″.r = p.d
          end
    end else
      mismatch "test_sum" p p2
  end else
    mismatch "test_sum" p p1

let try_fusion p p1 p2 =
  if test_sum p false p1 false p2 then
    Some (false, false)
  else if test_sum p true p1 false p2 then
    Some (true, false)
  else if test_sum p false p1 true p2 then
    Some (false, true)
  else if test_sum p true p1 true p2 then
    Some (true, true)
  else
    None

let split i n p =
  let n′ = n − 1 in
  let rec split′ head = function
    | [] → (p.r, List.rev head)
    | i1 :: ilist →
        if i1 < i then
          split′ (i1 :: head) ilist
        else if i1 > i then
          (p.r, List.rev_append head (List.map ((+) n′) (i1 :: ilist)))
        else
          (p.r + n′,
           List.rev_append head
             ((ThoList.range i1 (i1 + n′)) @ (List.map ((+) n′) ilist))) in
  let r′, p′ = split′ [] p.p in
  { d = p.d + n′; r = r′; p = p′ }
end
```

5.2.2 Bit Fiddlings

Bit vectors are popular in Fortran based implementations [1, 2, 11] and can be more efficient. In particular, when all infomation is packed into a single integer, much of the memory overhead is reduced.

```
module Bits =
  struct
    let rcs = RCS.rename rcs_file "Momentum.Bits()"
        (RCS.description rcs_file @
         [ "using bitfields as representation." ])

    type t = int
```

Bits $1 \ldots 21$ are used as a bitvector, indicating whether a particular momentum is included. Bits $22 \ldots 26$ represent the numbers of bits set in bits $1 \ldots 21$ and bits $27 \ldots 31$ denote the maximum number of momenta.

```
    let mask n = (1 lsl n) − 1
    let mask2 = mask 2
    let mask5 = mask 5
    let mask21 = mask 21

    let maskd = mask5 lsl 26
    let maskr = mask5 lsl 21
    let maskb = mask21

    let dim0 p = p land maskd
    let rank0 p = p land maskr
    let bits0 p = p land maskb

    let dim p = (dim0 p) lsr 26
    let rank p = (rank0 p) lsr 21
    let bits p = bits0 p

    let drb0 d r b = d lor r lor b
    let drb d r b = d lsl 26 lor r lsl 21 lor b
```

For a 64-bit architecture, the corresponding sizes could be increased to $1 \ldots 51$, $52 \ldots 57$, and $58 \ldots 63$. However, the combinatorical complexity will have killed us long before we can reach these values.

```
    exception Range of int
    exception Duplicate of int

    exception Mismatch of string × t × t
    let mismatch s p1 p2 = raise (Mismatch (s, p1, p2))

    let of_ints d p =
      let r = List.length p in
      if d ≤ 21 ∧ r ≤ 21 then begin
        List.fold_left (fun b p′ →
          if p′ ≤ d then
            b lor (1 lsl (pred p′))
          else
            raise (Range p′)) (drb d r 0) p
```

```
  end else
    raise (Range r)

let zero d = drb d 0 0

let singleton d p = drb d 1 (1 lsl (pred p))

let rec to_ints' acc p b =
  if b = 0 then
    List.rev acc
  else if (b land 1) = 1 then
    to_ints' (p :: acc) (succ p) (b lsr 1)
  else
    to_ints' acc (succ p) (b lsr 1)

let to_ints p = to_ints' [] 1 (bits p)

let to_string p =
  "[" ^ String.concat "," (List.map string_of_int (to_ints p)) ^
  "/" ^ string_of_int (rank p) ^ "/" ^ string_of_int (dim p) ^ "]"

let compare p1 p2 =
  if dim0 p1 = dim0 p2 then begin
    let c = compare (rank0 p1) (rank0 p2) in
    if c ≠ 0 then
      c
    else
      compare (bits p1) (bits p2)
  end else
    mismatch "compare" p1 p2

let neg p =
  let d = dim p and r = rank p in
  drb d (d − r) ((mask d) land (lnot p))

let abs p =
  if 2 × (rank p) > dim p then
    neg p
  else
    p

let add p1 p2 =
  let d1 = dim0 p1 and d2 = dim0 p2 in
  if d1 = d2 then begin
    let b1 = bits p1 and b2 = bits p2 in
    if b1 land b2 = 0 then
      drb0 d1 (rank0 p1 + rank0 p2) (b1 lor b2)
    else
      raise (Duplicate 0)
  end else
    mismatch "add" p1 p2

exception Negative

let rec sub p1 p2 =
  let d1 = dim0 p1 and d2 = dim0 p2 in
```

```
    if d1 = d2 then begin
      let r1 = rank0 p1 and r2 = rank0 p2 in
      if r1 ≥ r2 then begin
        let b1 = bits p1 and b2 = bits p2 in
        if b1 lor b2 = b1 then
          drb0 d1 (r1 − r2) (b1 lxor b2)
        else
          raise Negative
      end else
        neg (sub p2 p1)
    end else
      mismatch "sub" p1 p2
  let try_add p1 p2 =
    let d1 = dim0 p1 and d2 = dim0 p2 in
    if d1 = d2 then begin
      let b1 = bits p1 and b2 = bits p2 in
      if b1 land b2 = 0 then
        Some (drb0 d1 (rank0 p1 + rank0 p2) (b1 lor b2))
      else
        None
    end else
      mismatch "try_add" p1 p2
  let rec try_sub p1 p2 =
    let d1 = dim0 p1 and d2 = dim0 p2 in
    if d1 = d2 then begin
      let r1 = rank0 p1 and r2 = rank0 p2 in
      if r1 ≥ r2 then begin
        let b1 = bits p1 and b2 = bits p2 in
        if b1 lor b2 = b1 then
          Some (drb0 d1 (r1 − r2) (b1 lxor b2))
        else
          None
      end else
        begin match try_sub p2 p1 with
        | Some p → Some (neg p)
        | None → None
        end
    end else
      mismatch "sub" p1 p2
  let lesseq p1 p2 =
    let d1 = dim0 p1 and d2 = dim0 p2 in
    if d1 = d2 then begin
      let r1 = rank0 p1 and r2 = rank0 p2 in
      if r1 ≤ r2 then begin
        let b1 = bits p1 and b2 = bits p2 in
        b1 lor b2 = b2
      end else
        false
    end else
```

```
    mismatch "less" p1 p2

let less p1 p2 = p1 ≠ p2 ∧ lesseq p1 p2

let mask_in1 = 1
let mask_in2 = 2
let mask_in = mask_in1 lor mask_in2

module Scattering =
  struct

    let incoming p =
      rank p = 1 ∧ (mask_in land p ≠ 0)

    let outgoing p =
      rank p = 1 ∧ (mask_in land p = 0)

    let timelike p =
      (rank p > 0 ∧ (mask_in land p = 0)) ∨ (bits p = mask_in)

    let spacelike p =
      (rank p > 0) ∧ ¬ (timelike p)

    let s_channel_in p =
      bits p = mask_in

    let s_channel_out p =
      rank p > 0 ∧ (mask_in lxor p = 0)

    let s_channel p =
      s_channel_in p ∨ s_channel_out p

    let flip_s_channel_in p =
      if s_channel_in p then
        neg p
      else
        p

  end

module Decay =
  struct

    let incoming p =
      rank p = 1 ∧ (mask_in1 land p = mask_in1)

    let outgoing p =
      rank p = 1 ∧ (mask_in1 land p = 0)

    let timelike p =
      incoming p ∨ (rank p > 0 ∧ mask_in1 land p = 0)

    let spacelike p =
      ¬ (timelike p)

  end

let test_sum p inv1 p1 inv2 p2 =
  let d = dim p in
  if d = dim p1 then begin
```

```
        if d = dim p2 then begin
          match (if inv1 then try_add else try_sub) p p1 with
          | None → false
          | Some p′ →
              begin match (if inv2 then try_add else try_sub) p′ p2 with
              | None → false
              | Some p″ →
                  let r = rank p″ in
                  r = 0 ∨ r = d
              end
        end else
          mismatch "test_sum" p p2
      end else
        mismatch "test_sum" p p1

  let try_fusion p p1 p2 =
    if test_sum p false p1 false p2 then
      Some (false, false)
    else if test_sum p true p1 false p2 then
      Some (true, false)
    else if test_sum p false p1 true p2 then
      Some (false, true)
    else if test_sum p true p1 true p2 then
      Some (true, true)
    else
      None
```

First create a gap of size $n-1$ and subsequently fill it if and only if the bit i was set.

```
  let split i n p =
    let delta_d = n − 1
    and b = bits p in
    let mask_low = mask (pred i)
    and mask_i = 1 lsl (pred i)
    and mask_high = lnot (mask i) in
    let b_low = mask_low land b
    and b_med, delta_r =
      if mask_i land b ≠ 0 then
        ((mask n) lsl (pred i), delta_d)
      else
        (0, 0)
    and b_high =
      if delta_d > 0 then
        (mask_high land b) lsl delta_d
      else if delta_d = 0 then
        mask_high land b
      else
        (mask_high land b) lsr (−delta_d) in
    drb (dim p + delta_d) (rank p + delta_r) (b_low lor b_med lor b_high)
end
```

5.2.3 *Whizard*

```
module type Whizard =
  sig
    type t
    val of_momentum : t → int
    val to_momentum : int → int → t
  end

module BitsW =
  struct
    type t = Bits.t
    open Bits (* NB: this includes the internal functions not in T! *)

    let of_momentum p =
      let d = dim p in
      let bit_in1 = 1 land p
      and bit_in2 = 1 land (p lsr 1)
      and bits_out = ((mask d) land p) lsr 2 in
      bits_out lor (bit_in1 lsl (d - 1)) lor (bit_in2 lsl (d - 2))

    let rec count_non_zero' acc i last b =
      if i > last then
        acc
      else if (1 lsl (pred i)) land b = 0 then
        count_non_zero' acc (succ i) last b
      else
        count_non_zero' (succ acc) (succ i) last b

    let count_non_zero first last b =
      count_non_zero' 0 first last b

    let to_momentum d w =
      let bit_in1 = 1 land (w lsr (d - 1))
      and bit_in2 = 1 land (w lsr (d - 2))
      and bits_out = (mask (d - 2)) land w in
      let b = (bits_out lsl 2) lor bit_in1 lor (bit_in2 lsl 1) in
      drb d (count_non_zero 1 d b) b

  end
```

The following would be a tad more efficient, if coded directly, but there's no point in wasting effort on this.

```
module ListsW =
  struct
    type t = Lists.t
    let of_momentum p =
      BitsW.of_momentum (Bits.of_ints p.Lists.d p.Lists.p)
    let to_momentum d w =
      Lists.of_ints d (Bits.to_ints (BitsW.to_momentum d w))
  end
```

5.2.4 Suggesting a Default Implementation

Lists is better tested, but the more recent *Bits* appears to work as well and is *much* more efficient, resulting in a relative factor of better than 2. This performance ratio is larger than I had expected and we are not likely to reach its limit of 21 independent vectors anyway.

module $Default = Bits$
module $DefaultW = BitsW$

—6—
Cascades

6.1 Interface of Cascade_syntax

type $('flavor, 'p)\ t$ =
  | *True*
  | *False*
  | *On_shell* of $'flavor\ list \times 'p$
  | *On_shell_not* of $'flavor\ list \times 'p$
  | *Off_shell* of $'flavor\ list \times 'p$
  | *Off_shell_not* of $'flavor\ list \times 'p$
  | *Gauss* of $'flavor\ list \times 'p$
  | *Gauss_not* of $'flavor\ list \times 'p$
  | *Any_flavor* of $'p$
  | *Or* of $('flavor, 'p)\ t\ list$
  | *And* of $('flavor, 'p)\ t\ list$

val *mk_true* : $unit \rightarrow ('flavor, 'p)\ t$
val *mk_false* : $unit \rightarrow ('flavor, 'p)\ t$
val *mk_on_shell* : $'flavor\ list \rightarrow 'p \rightarrow ('flavor, 'p)\ t$
val *mk_on_shell_not* : $'flavor\ list \rightarrow 'p \rightarrow ('flavor, 'p)\ t$
val *mk_off_shell* : $'flavor\ list \rightarrow 'p \rightarrow ('flavor, 'p)\ t$
val *mk_off_shell_not* : $'flavor\ list \rightarrow 'p \rightarrow ('flavor, 'p)\ t$
val *mk_gauss* : $'flavor\ list \rightarrow 'p \rightarrow ('flavor, 'p)\ t$
val *mk_gauss_not* : $'flavor\ list \rightarrow 'p \rightarrow ('flavor, 'p)\ t$
val *mk_any_flavor* : $'p \rightarrow ('flavor, 'p)\ t$
val *mk_or* : $('flavor, 'p)\ t \rightarrow ('flavor, 'p)\ t \rightarrow ('flavor, 'p)\ t$
val *mk_and* : $('flavor, 'p)\ t \rightarrow ('flavor, 'p)\ t \rightarrow ('flavor, 'p)\ t$

val *to_string* : $('flavor \rightarrow string) \rightarrow ('p \rightarrow string) \rightarrow ('flavor, 'p)\ t \rightarrow string$

exception *Syntax_Error* of $string \times int \times int$

6.2 Implementation of Cascade_syntax

Concerning the Gaussian propagators, we admit the following: In principle, they would allow for flavor sums like the off-shell lines, but for all practical purposes they are used only for determining the significance of a specified intermediate state. So we select them in the same manner as on-shell states.

```
type ('flavor, 'p) t =
  | True
  | False
  | On_shell of 'flavor list × 'p
  | On_shell_not of 'flavor list × 'p
  | Off_shell of 'flavor list × 'p
  | Off_shell_not of 'flavor list × 'p
  | Gauss of 'flavor list × 'p
  | Gauss_not of 'flavor list × 'p
  | Any_flavor of 'p
  | Or of ('flavor, 'p) t list
  | And of ('flavor, 'p) t list

let mk_true () = True
let mk_false () = False
let mk_on_shell f p = On_shell (f, p)
let mk_on_shell_not f p = On_shell_not (f, p)
let mk_off_shell f p = Off_shell (f, p)
let mk_off_shell_not f p = Off_shell_not (f, p)
let mk_gauss f p = Gauss (f, p)
let mk_gauss_not f p = Gauss_not (f, p)
let mk_any_flavor p = Any_flavor p

let mk_or c1 c2 =
  match c1, c2 with
  | _, True | True, _ → True
  | c, False | False, c → c
  | Or cs, Or cs' → Or (cs @ cs')
  | Or cs, c | c, Or cs → Or (c :: cs)
  | c, c' → Or [c; c']

let mk_and c1 c2 =
  match c1, c2 with
  | c, True | True, c → c
  | c, False | False, c → False
  | And cs, And cs' → And (cs @ cs')
  | And cs, c | c, And cs → And (c :: cs)
  | c, c' → And [c; c']

let to_string flavor_to_string momentum_to_string cascades =
  let rec to_string' = function
    | True → "true"
    | False → "false"
    | On_shell (fs, p) →
        momentum_to_string p ^ " = " ^ (String.concat ":" (List.map flavor_to_string fs))
    | On_shell_not (fs, p) →
        momentum_to_string p ^ " = !" ^ (String.concat ":" (List.map flavor_to_string fs))
    | Off_shell (fs, p) →
        momentum_to_string p ^ " ~ " ^
        (String.concat ":" (List.map flavor_to_string fs))
```

```
    | Off_shell_not (fs, p) →
        momentum_to_string p ^ "␣~␣!" ^
        (String.concat ":" (List.map flavor_to_string fs))
    | Gauss (fs, p) →
        momentum_to_string p ^ "␣#␣" ^ (String.concat ":" (List.map flavor_to_string fs))
    | Gauss_not (fs, p) →
        momentum_to_string p ^ "␣#␣!" ^ (String.concat ":" (List.map flavor_to_string fs))
    | Any_flavor p →
        momentum_to_string p ^ "␣~␣?"
    | Or cs →
        String.concat "␣||␣" (List.map (fun c → "(" ^ to_string' c ^ ")") cs)
    | And cs →
        String.concat "␣&&␣" (List.map (fun c → "(" ^ to_string' c ^ ")") cs) in
  to_string' cascades

let int_list_to_string p =
  String.concat "+" (List.map string_of_int (Sort.list (<) p))

exception Syntax_Error of string × int × int
```

6.3 Lexer

```
{
open Cascade_parser
let unquote s =
  String.sub s 1 (String.length s − 2)
}

let digit = ['0'−'9']
let upper = ['A'−'Z']
let lower = ['a'−'z']
let char = upper | lower
let white = [' ' '\t' '\n']
```

We use a very liberal definition of strings for flavor names.

```
rule token = parse
    white { token lexbuf } (* skip blanks *)
  | '%' [^'\n']* '\n'
               { token lexbuf } (* skip comments *)
  | digit+ { INT (int_of_string (Lexing.lexeme lexbuf)) }
  | '+' { PLUS }
  | ':' { COLON }
  | '~' { OFFSHELL }
  | '=' { ONSHELL }
  | '#' { GAUSS }
  | '!' { NOT }
  | '&' '&'? { AND }
  | '|' '|'? { OR }
  | '(' { LPAREN }
  | ')' { RPAREN }
```

```
  | char [^ ' ' '\t' '\n' '|' '&' '(' ')' ':']*
                  { FLAVOR (Lexing.lexeme lexbuf) }
  | '"' [^ '"']* '"'
                  { FLAVOR (unquote (Lexing.lexeme lexbuf)) }
  | eof { END }
```

6.4 Parser

Header

```
open Cascade_syntax
let parse_error msg =
  raise (Syntax_Error (msg, symbol_start (), symbol_end ()))
```

Token declarations

```
%token < string > FLAVOR
%token < int > INT
%token LPAREN RPAREN
%token AND OR PLUS COLON NOT
%token ONSHELL OFFSHELL GAUSS
%token END
%left OR
%left AND
%left PLUS COLON
%left NOT

%start main
%type < (string, int list) Cascade_syntax.t > main
```

Grammar rules

```
main ::=
    END { mk_true () }
  | cascades END { $1 }

cascades ::=
    cascade { $1 }
  | LPAREN cascades RPAREN { $2 }
  | cascades AND cascades { mk_and $1 $3 }
  | cascades OR cascades { mk_or $1 $3 }
```

cascade ::=
    momentum_list { *mk_any_flavor* \$1 }
  | *momentum_list* *ONSHELL* *flavor_list*
      { *mk_on_shell* \$3 \$1 }
  | *momentum_list* *ONSHELL* *NOT* *flavor_list*
      { *mk_on_shell_not* \$4 \$1 }
  | *momentum_list* *OFFSHELL* *flavor_list*
      { *mk_off_shell* \$3 \$1 }
  | *momentum_list* *OFFSHELL* *NOT* *flavor_list*
      { *mk_off_shell_not* \$4 \$1 }
  | *momentum_list* *GAUSS* *flavor_list* { *mk_gauss* \$3 \$1 }
  | *momentum_list* *GAUSS* *NOT* *flavor_list*
      { *mk_gauss_not* \$4 \$1 }

momentum_list ::=
  | *momentum* { [\$1] }
  | *momentum_list* *PLUS* *momentum* { \$3 :: \$1 }

momentum ::=
    INT { \$1 }

flavor_list ::=
    FLAVOR { [\$1] }
  | *flavor_list* *COLON* *FLAVOR* { \$3 :: \$1 }

6.5 Interface of Cascade

```
module type T =
  sig

    type flavor
    type p

    type t
    val of_string_list : int → string list → t
    val to_string : t → string
```

An opaque type that describes the set of all constraints on an amplitude and how to construct it from a cascade description.

```
    type selectors
    val to_selectors : t → selectors
```

Don't throw anything away:

```
    val no_cascades : selectors
```

select_wf *s* *is_timelike* *f* *p* *ps* returns true iff either

- the flavor f and momentum p match the selection s or

- *all* combinations of the momenta in *ps* are compatible, i. e. $\pm \sum p_i \leq q$.

The latter test is only required in theories with quartic or higher vertices, where *ps* will be the list of all incoming momenta in a fusion. *is_timelike* is required to determine, whether particles and anti-particles should be distinct.

```
val select_wf : selectors → (p → bool) → flavor → p → p list → bool
```

select_p s p ps same as *select_wf s f p ps*, but ignores the flavor *f*

```
val select_p : selectors → p → p list → bool
```

on_shell s p

```
val on_shell : selectors → flavor → p → bool
```

is_gauss s p

```
val is_gauss : selectors → flavor → p → bool
```

partition s returns a partition of the external particles that can not be reordered without violating the cascade constraints.

```
val partition : selectors → int list list
```

Diagnostics:

```
  val description : selectors → string option
end
module Make (M : Model.T) (P : Momentum.T) :
  T with type flavor = M.flavor and type p = P.t
```

6.6 Implementation of Cascade

```
module type T =
  sig
    type flavor
    type p

    type t
    val of_string_list : int → string list → t
    val to_string : t → string

    type selectors
    val to_selectors : t → selectors
    val no_cascades : selectors

    val select_wf : selectors → (p → bool) → flavor → p → p list → bool
    val select_p : selectors → p → p list → bool
    val on_shell : selectors → flavor → p → bool
    val is_gauss : selectors → flavor → p → bool

    val partition : selectors → int list list
    val description : selectors → string option
  end
```

```
module Make (M : Model.T) (P : Momentum.T) :
    (T with type flavor = M.flavor and type p = P.t) =
  struct

    module CS = Cascade_syntax

    type flavor = M.flavor
    type p = P.t
```

Since we have

$$p \leq q \Longleftrightarrow (-q) \leq (-p) \tag{6.1}$$

also for $\leq$ as set inclusion *lesseq*, only four of the eight combinations are independent

$$\begin{aligned} p \leq q &\Longleftrightarrow (-q) \leq (-p) \\ q \leq p &\Longleftrightarrow (-p) \leq (-q) \\ p \leq (-q) &\Longleftrightarrow q \leq (-p) \\ (-q) \leq p &\Longleftrightarrow (-p) \leq q \end{aligned} \tag{6.2}$$

```
    let one_compatible p q =
      let neg_q = P.neg q in
      P.lesseq p q ∨
      P.lesseq q p ∨
      P.lesseq p neg_q ∨
      P.lesseq neg_q p
```

'tis wasteful ... (at least by a factor of two, because every momentum combination is generated, including the negative ones.

```
    let all_compatible p p_list q =
      let l = List.length p_list in
      if l ≤ 2 then
        one_compatible p q
      else
        let tuple_lengths = ThoList.range 2 (succ l / 2) in
        let tuples = ThoList.flatmap (fun n → Combinatorics.choose n p_list) tuple_lengths in
        let momenta = List.map (List.fold_left P.add (P.zero (P.dim q))) tuples in
        List.for_all (one_compatible q) momenta
```

The following assumes that the *flavor list* is always very short. Otherwise one should use an efficient set implementation.

```
    type t =
      | True
      | False
      | On_shell of flavor list × P.t
      | On_shell_not of flavor list × P.t
      | Off_shell of flavor list × P.t
      | Off_shell_not of flavor list × P.t
      | Gauss of flavor list × P.t
      | Gauss_not of flavor list × P.t
      | Any_flavor of P.t
      | And of t list
```

```
let of_string s =
  Cascade_parser.main Cascade_lexer.token (Lexing.from_string s)
```

If we knew that we're dealing with a scattering, we could apply *P.flip_s_channel_in* to all momenta, so that $1+2$ accepts the particle and not the antiparticle. Right now, we don't have this information.

```
let import dim cascades =
  let rec import' = function
    | CS.True ->
        True
    | CS.False ->
        False
    | CS.On_shell (f, p) ->
        On_shell (List.map M.flavor_of_string f, P.of_ints dim p)
    | CS.On_shell_not (f, p) ->
        On_shell_not (List.map M.flavor_of_string f, P.of_ints dim p)
    | CS.Off_shell (fs, p) ->
        Off_shell (List.map M.flavor_of_string fs, P.of_ints dim p)
    | CS.Off_shell_not (fs, p) ->
        Off_shell_not (List.map M.flavor_of_string fs, P.of_ints dim p)
    | CS.Gauss (f, p) ->
        Gauss (List.map M.flavor_of_string f, P.of_ints dim p)
    | CS.Gauss_not (f, p) ->
        Gauss (List.map M.flavor_of_string f, P.of_ints dim p)
    | CS.Any_flavor p ->
        Any_flavor (P.of_ints dim p)
    | CS.Or cs ->
        invalid_arg "Cascade: OR patterns (||) not supported in this version!"
    | CS.And cs -> And (List.map import' cs) in
  import' cascades

let of_string_list dim strings =
  match List.map of_string strings with
  | [] -> True
  | first :: next ->
      import dim (List.fold_right CS.mk_and next first)

let flavors_to_string fs =
  (String.concat ":" (List.map M.flavor_to_string fs))

let momentum_to_string p =
  String.concat "+" (List.map string_of_int (P.to_ints p))

let rec to_string = function
  | True ->
      "true"
  | False ->
      "false"
  | On_shell (fs, p) ->
      momentum_to_string p ^ " = " ^ flavors_to_string fs
  | On_shell_not (fs, p) ->
```

```
      momentum_to_string p ^ " = ! " ^ flavors_to_string fs
  | Off_shell (fs, p) →
      momentum_to_string p ^ " ~ " ^ flavors_to_string fs
  | Off_shell_not (fs, p) →
      momentum_to_string p ^ " ~ ! " ^ flavors_to_string fs
  | Gauss (fs, p) →
      momentum_to_string p ^ " # " ^ flavors_to_string fs
  | Gauss_not (fs, p) →
      momentum_to_string p ^ " # ! " ^ flavors_to_string fs
  | Any_flavor p →
      momentum_to_string p ^ " ~ ?"
  | And cs →
      String.concat " && " (List.map (fun c → "(" ^ to_string c ^ ")") cs)

type selectors =
    { select_p : p → p list → bool;
      select_wf : (p → bool) → flavor → p → p list → bool;
      on_shell : flavor → p → bool;
      is_gauss : flavor → p → bool;
      partition : int list list;
      description : string option }

let no_cascades =
  { select_p = (fun _ _ → true);
    select_wf = (fun _ _ _ _ → true);
    on_shell = (fun _ _ → false);
    is_gauss = (fun _ _ → false);
    partition = [];
    description = None }

let select_p s = s.select_p
let select_wf s = s.select_wf
let on_shell s = s.on_shell
let is_gauss s = s.is_gauss
let partition s = s.partition
let description s = s.description

let to_select_p cascades p p_in =
  let rec to_select_p' = function
    | True → true
    | False → false
    | On_shell (_, momentum) | On_shell_not (_, momentum)
    | Off_shell (_, momentum) | Off_shell_not (_, momentum)
    | Gauss (_, momentum) | Gauss_not (_, momentum)
    | Any_flavor momentum → all_compatible p p_in momentum
    | And [] → false
    | And cs → List.for_all to_select_p' cs in
  to_select_p' cascades

let to_select_wf cascades is_timelike f p p_in =
  let f' = M.conjugate f in
  let rec to_select_wf' = function
    | True → true
```

```
      | False → false
      | Off_shell (flavors, momentum) →
          if p = momentum then
            List.mem f' flavors ∨ (if is_timelike p then false else List.mem f flavors)
          else if p = P.neg momentum then
            List.mem f flavors ∨ (if is_timelike p then false else List.mem f' flavors)
          else
            one_compatible p momentum ∧ all_compatible p p_in momentum
      | On_shell (flavors, momentum) | Gauss (flavors, momentum) →
           if is_timelike p then begin
             if p = momentum then
               List.mem f' flavors
             else if p = P.neg momentum then
               List.mem f flavors
             else
               one_compatible p momentum ∧ all_compatible p p_in momentum
           end else
             false
      | Off_shell_not (flavors, momentum) →
          if p = momentum then
            ¬ (List.mem f' flavors ∨ (if is_timelike p then false else List.mem f flavors))
          else if p = P.neg momentum then
            ¬ (List.mem f flavors ∨ (if is_timelike p then false else List.mem f' flavors))
          else
            one_compatible p momentum ∧ all_compatible p p_in momentum
      | On_shell_not (flavors, momentum) | Gauss_not (flavors, momentum) →

          if is_timelike p then begin
            if p = momentum then
              ¬ (List.mem f' flavors)
            else if p = P.neg momentum then
              ¬ (List.mem f flavors)
            else
              one_compatible p momentum ∧ all_compatible p p_in momentum
          end else
            false
      | Any_flavor momentum →
          one_compatible p momentum ∧ all_compatible p p_in momentum
      | And [] → false
      | And cs → List.for_all to_select_wf' cs in
    to_select_wf' cascades
```

In case you're wondering: *to_on_shell f p* and *is_gauss f p* only search for on shell conditions and are to be used in a target, not in *Fusion*!

```
let to_on_shell cascades f p =
  let f' = M.conjugate f in
  let rec to_on_shell' = function
    | True | False | Any_flavor _
    | Off_shell (_, _) | Off_shell_not (_, _)
    | Gauss (_, _) | Gauss_not (_, _) → false
```

```
        | On_shell (flavors, momentum) →
            (p = momentum ∨ p = P.neg momentum) ∧ (List.mem f flavors ∨
List.mem f′ flavors)
        | On_shell_not (flavors, momentum) →
            (p = momentum ∨ p = P.neg momentum) ∧ ¬ (List.mem f flavors ∨
List.mem f′ flavors)
        | And [] → false
        | And cs → List.for_all to_on_shell′ cs in
      to_on_shell′ cascades

  let to_gauss cascades f p =
    let f′ = M.conjugate f in
    let rec to_gauss′ = function
        | True | False | Any_flavor _
        | Off_shell (_, _) | Off_shell_not (_, _)
        | On_shell (_, _) | On_shell_not (_, _) → false
        | Gauss (flavors, momentum) →
            (p = momentum ∨ p = P.neg momentum) ∧ (List.mem f flavors ∨
List.mem f′ flavors)
        | Gauss_not (flavors, momentum) →
            (p = momentum ∨ p = P.neg momentum) ∧ ¬ (List.mem f flavors ∨
List.mem f′ flavors)
        | And [] → false
        | And cs → List.for_all to_gauss′ cs in
      to_gauss′ cascades
```

Not a working implementation yet, but it isn't used either . . .

```
  module IPowSet =
    PowSet.Make (struct type t = int let compare = compare let to_string = string_of_int end)

  let rec coarsest_partition′ = function
      | True | False → IPowSet.empty
      | On_shell (_, momentum) | On_shell_not (_, momentum)
      | Off_shell (_, momentum) | Off_shell_not (_, momentum)
      | Gauss (_, momentum) | Gauss_not (_, momentum)
      | Any_flavor momentum → IPowSet.of_lists [P.to_ints momentum]
      | And [] → IPowSet.empty
      | And cs → IPowSet.basis (IPowSet.union (List.map coarsest_partition′ cs))

  let coarsest_partition cascades =
    let p = coarsest_partition′ cascades in
    if IPowSet.is_empty p then
      []
    else
      IPowSet.to_lists p

  let part_to_string part =
    "{" ^ String.concat "," (List.map string_of_int part) ^ "}"

  let partition_to_string = function
    | [] → ""
```

```
  | parts ->
      "  grouping {" ^ String.concat "," (List.map part_to_string parts) ^ "}"
let to_selectors = function
  | True -> no_cascades
  | c ->
      let partition = coarsest_partition c in
      { select_p = to_select_p c;
        select_wf = to_select_wf c;
        on_shell = to_on_shell c;
        is_gauss = to_gauss c;
        partition = partition;
        description = Some (to_string c ^ partition_to_string partition) }
end
```

—7—
Color

7.1 Interface of Color

7.1.1 Quantum Numbers

Color is not necessarily the SU(3) of QCD. Conceptually, it can be any *unbroken* symmetry (*broken* symmetries correspond to *Model.flavor*). In order to keep the group theory simple, we confine ourselves to the fundamental and adjoint representation of a single SU(N_C) for the moment. Therefore, particles are either color singlets or live in the defining representation of SU(N_C): $SUN(|N_C|)$, its conjugate $SUN(-|N_C|)$ or in the adjoint representation of SU(N_C): $AdjSUN(N_C)$.

```
type t = Singlet | SUN of int | AdjSUN of int

val conjugate : t → t
val compare : t → t → int
```

7.1.2 Color Flows

```
module type Flow =
  sig

    type color
    type t = color list × color list
    val rank : t → int

    val of_list : int list → color
    val ghost : unit → color
    val to_lists : t → int list list
    val in_to_lists : t → int list list
    val out_to_lists : t → int list list
    val ghost_flags : t → bool list
    val in_ghost_flags : t → bool list
    val out_ghost_flags : t → bool list
```

A factor is a list of powers

$$\sum_i \left(\frac{num_i}{den_i}\right)^{power_i} \tag{7.1}$$

```
    type power = { num : int; den : int; power : int }
    type factor = power list

    val factor : t -> t -> factor
    val zero : factor

  end

module Flow : Flow
```

7.2 Implementation of Color

7.2.1 Quantum Numbers

```
type t =
  | Singlet
  | SUN of int
  | AdjSUN of int

let conjugate = function
  | Singlet -> Singlet
  | SUN n -> SUN (-n)
  | AdjSUN n -> AdjSUN n

let compare c1 c2 =
  match c1, c2 with
  | Singlet, Singlet -> 0
  | Singlet, _ -> -1
  | _, Singlet -> 1
  | SUN n, SUN n' -> compare n n'
  | SUN _, AdjSUN _ -> -1
  | AdjSUN _, SUN _ -> 1
  | AdjSUN n, AdjSUN n' -> compare n n'

module type Line =
  sig
    type t
    val conj : t -> t
    val equal : t -> t -> bool
    val to_string : t -> string
  end

module type Cycles =
  sig

    type line
    type t = (line * line) list
```

Contract the graph by connecting lines and return the number of cycles together with the contracted graph.

The semantics of the contracted graph is not yet 100%ly fixed.

```
    val contract : t -> int * t
```

The same as *contract*, but returns only the number of cycles and raises *Open_line* when not all lines are closed.

```
val count : t → int
exception Open_line
```

Mainly for debugging ...

```
val to_string : t → string
```

```
end
```

```
module Cycles (L : Line) : Cycles with type line = L.t =
  struct

    type line = L.t
    type t = (line × line) list

    exception Open_line
```

NB: The following algorithm for counting the cycles is quadratic since it performs nested scans of the lists. If this was a serious problem one could replace the lists of pairs by a *Map* and replace one power by a logarithm.

```
    let rec find_fst c_final c1 disc seen = function
      | [] → ((L.conj c_final, c1) :: disc, List.rev seen)
      | (c1′, c2′) as c12′ :: rest →
          if L.equal c1 c1′ then
            find_snd c_final (L.conj c2′) disc [] (List.rev_append seen rest)
          else
            find_fst c_final c1 disc (c12′ :: seen) rest

    and find_snd c_final c2 disc seen = function
      | [] → ((L.conj c_final, L.conj c2) :: disc, List.rev seen)
      | (c1′, c2′) as c12′ :: rest →
          if L.equal c2′ c2 then begin
            if L.equal c1′ c_final then
              (disc, List.rev_append seen rest)
            else
              find_fst c_final (L.conj c1′) disc [] (List.rev_append seen rest)
          end else
            find_snd c_final c2 disc (c12′ :: seen) rest

    let consume = function
      | [] → ([], [])
      | (c1, c2) :: rest → find_snd (L.conj c1) (L.conj c2) [] [] rest

    let contract lines =
      let rec contract′ acc disc = function
        | [] → (acc, List.rev disc)
        | rest →
            begin match consume rest with
            | [], rest′ → contract′ (succ acc) disc rest′
            | disc′, rest′ → contract′ acc (List.rev_append disc′ disc) rest′
            end in
      contract′ 0 [] lines
```

```
    let count lines =
      match contract lines with
      | n, [] → n
      | n, _ → raise Open_line

    let to_string lines =
      String.concat ""
        (List.map
           (fun (c1, c2) → "[" ^ L.to_string c1 ^ "," ^ L.to_string c2 ^ "]")
           lines)

  end
```

7.2.2 Color Flows

```
module type Flow =
  sig
    type color
    type t = color list × color list
    val rank : t → int
    val of_list : int list → color
    val ghost : unit → color
    val to_lists : t → int list list
    val in_to_lists : t → int list list
    val out_to_lists : t → int list list
    val ghost_flags : t → bool list
    val in_ghost_flags : t → bool list
    val out_ghost_flags : t → bool list
    type power = { num : int; den : int; power : int }
    type factor = power list
    val factor : t → t → factor
    val zero : factor
  end

module Flow (* : Flow *) =
  struct

    type color =
      | Lines of int × int
      | Ghost

    type t = color list × color list

    let rank cflow =
      2
```

Constructors

```
    let ghost () =
      Ghost

    let of_list = function
```

```
  | [c1; c2] → Lines (c1, c2)
  | _ → invalid_arg "Color.Flow.of_list:␣num_lines␣!=␣2"

let to_list = function
  | Lines (c1, c2) → [c1; c2]
  | Ghost → [0; 0]

let to_lists (cfin, cfout) =
  (List.map to_list cfin) @ (List.map to_list cfout)

let in_to_lists (cfin, _) =
  List.map to_list cfin

let out_to_lists (_, cfout) =
  List.map to_list cfout

let ghost_flag = function
  | Lines _ → false
  | Ghost → true

let ghost_flags (cfin, cfout) =
  (List.map ghost_flag cfin) @ (List.map ghost_flag cfout)

let in_ghost_flags (cfin, _) =
  List.map ghost_flag cfin

let out_ghost_flags (_, cfout) =
  List.map ghost_flag cfout
```

Evaluation

```
type power = { num : int; den : int; power : int }
type factor = power list
let zero = []

let count_ghosts1 colors =
  List.fold_left
    (fun acc → function Ghost → succ acc | _ → acc)
    0 colors

let count_ghosts (fin, fout) =
  count_ghosts1 fin + count_ghosts1 fout

type α square =
  | Square of α
  | Mismatch

let conjugate = function
  | Lines (c1, c2) → Lines (−c2, − c1)
  | Ghost → Ghost

let cross_in (cin, cout) =
  cin @ (List.map conjugate cout)

let cross_out (cin, cout) =
  (List.map conjugate cin) @ cout
```

```
module C = Cycles (struct
  type t = int
  let conj = (−)
  let equal = (=)
  let to_string = string_of_int
end)

let square f1 f2 =
  let rec square′ acc f1′ f2′ =
    match f1′, f2′ with
    | [], [] → Square (List.rev acc)
    | _, [] | [], _ → Mismatch
    | Ghost :: rest1, Ghost :: rest2 →
        square′ acc rest1 rest2
    | Lines (0, 0) :: rest1, Lines (0, 0) :: rest2 →
        square′ acc rest1 rest2
    | Lines (0, c1′) :: rest1, Lines (0, c2′) :: rest2 →
        square′ ((c1′, c2′) :: acc) rest1 rest2
    | Lines (c1, 0) :: rest1, Lines (c2, 0) :: rest2 →
        square′ ((c1, c2) :: acc) rest1 rest2
    | Lines (0, _) :: _, _ | _ , Lines (0, _) :: _
    | Lines (_, 0) :: _, _ | _, Lines (_, 0) :: _ → Mismatch
    | Lines (_, _) :: _, Ghost :: _ | Ghost :: _, Lines (_, _) :: _ →
Mismatch
    | Lines (c1, c1′) :: rest1, Lines (c2, c2′) :: rest2 →
        square′ ((c1′, c2′) :: (c1, c2) :: acc) rest1 rest2 in
  square′ [] (cross_out f1) (cross_out f2)
```

In addition to counting closed color loops, we also need to count closed gluon loops. Fortunately, we can use the same algorithm on a different data type, provided it doesn't require all lines to be closed.

```
module C2 = Cycles (struct
  type t = int × int
  let conj (c1, c2) = (− c2, − c1)
  let equal (c1, c2) (c1′, c2′) = c1 = c1′ ∧ c2 = c2′
  let to_string (c1, c2) = "(" ^ string_of_int c1 ^ "," ^ string_of_int c2 ^ ")"
end)

let square2 f1 f2 =
  let rec square2′ acc f1′ f2′ =
    match f1′, f2′ with
    | [], [] → Square (List.rev acc)
    | _, [] | [], _ → Mismatch
    | Ghost :: rest1, Ghost :: rest2 →
        square2′ acc rest1 rest2
    | Lines (0, 0) :: rest1, Lines (0, 0) :: rest2 →
        square2′ acc rest1 rest2
    | Lines (0, _) :: rest1, Lines (0, _) :: rest2
    | Lines (_, 0) :: rest1, Lines (_, 0) :: rest2 →
        square2′ acc rest1 rest2
    | Lines (0, _) :: _, _ | _ , Lines (0, _) :: _
```

```
        | Lines (_, 0) :: _, _ | _, Lines (_, 0) :: _ → Mismatch
        | Lines (_, _) :: _, Ghost :: _ | Ghost :: _, Lines (_, _) :: _ →
Mismatch
        | Lines (c1, c1') :: rest1, Lines (c2, c2') :: rest2 →
            square2' (((c1, c1'), (c2, c2')) :: acc) rest1 rest2 in
      square2' [] (cross_out f1) (cross_out f2)
```

int_power : $n\, p \to n^p$ for integers is missing from *Pervasives*!

```
let int_power n p =
  let rec int_power' acc i =
    if i < 0 then
      invalid_arg "int_power"
    else if i = 0 then
      acc
    else
      int_power' (n × acc) (pred i) in
  int_power' 1 p
```

Instead of implementing a full fledged algebraic evaluator, let's simply expand the binomial by hand:

$$\left(\frac{N_C^2 - 2}{N_C^2}\right)^n = \sum_{i=0}^{n} \binom{n}{i} (-2)^i N_C^{-2i} \tag{7.2}$$

NB: Any result of *square* other than *Mismatch* guarantees *count_ghosts f1* = *count_ghosts f2*.

```
let factor f1 f2 =
  match square f1 f2, square2 f1 f2 with
  | Mismatch, _ | _, Mismatch → []
  | Square f12, Square f12' →
      let num_cycles = C.count f12
      and num_cycles2, disc = C2.contract f12'
      and num_ghosts = count_ghosts f1 in
      List.map
        (fun i →
          let parity = if num_ghosts mod 2 = 0 then 1 else -1
          and power = num_cycles − num_ghosts in
          let coeff = int_power (−2) i × Combinatorics.binomial num_cycles2 i
          and power2 = − 2 × i in
          { num = parity × coeff;
            den = 1;
            power = power + power2 })
        (ThoList.range 0 num_cycles2)

end
```

later:

```
module General_Flow =
  struct

    type color =
      | Lines of int list
```

```
    | Ghost of int

  type t = color list × color list

  let rank_default = 2 (* Standard model *)

  let rank cflow =
    try
      begin match List.hd cflow with
      | Lines lines → List.length lines
      | Ghost n_lines → n_lines
      end
    with
    | _ → rank_default
end
```

—8—
Fusions

8.1 Interface of Fusion

```
module type T =
  sig
    val options : Options.t
```

Wavefunctions are an abstract data type, containing a momentum p and additional quantum numbers, collected in *flavor*.

```
    type wf
    val conjugate : wf → wf
```

Obviously, *flavor* is not restricted to the physical notion of flavor, but can carry spin, color, etc.

```
    type flavor
    val flavor : wf → flavor
    type flavor_sans_color
    val flavor_sans_color : wf → flavor_sans_color
```

Momenta are represented by an abstract datatype (defined in *Momentum*) that is optimized for performance. They can be accessed either abstractly or as lists of indices of the external momenta. These indices are assigned sequentially by *amplitude* below.

```
    type p
    val momentum : wf → p
    val momentum_list : wf → int list
```

At tree level, the wave functions are uniquely specified by *flavor* and momentum. If loops are included, we need to distinguish among orders. Also, if we build a result from an incomplete sum of diagrams, we need to add a distinguishing mark. At the moment, we assume that a *string* that can be attached to the symbol suffices.

```
    val wf_tag : wf → string option
```

Coupling constants

```
    type constant
```

and right hand sides of assignments. The latter are formed from a sign from Fermi statistics, a coupling (constand and Lorentz structure) and wave functions.

```
type coupling
type rhs
type α children
val sign : rhs → int
val coupling : rhs → constant Coupling.t
```

```
val coupling_tag : rhs → string option
```

In renormalized perturbation theory, couplings come in different orders of the loop expansion. Be prepared: `val order : rhs → int`

This is here only for the benefit of *Target* and shall become `val children : rhs → wf children` later ...

```
val children : rhs → wf list
```

Fusions come in two types: fusions of wave functions to off-shell wave functions:

$$\phi(p+q) = \phi(p)\phi(q)$$

```
type fusion
val lhs : fusion → wf
val rhs : fusion → rhs list
```

and products at the keystones:

$$\phi(-p-q) \cdot \phi(p)\phi(q)$$

```
type braket
val bra : braket → wf
val ket : braket → rhs list
```

amplitude goldstones incoming outgoing calculates the amplitude for scattering of *incoming* to *outgoing*. If *goldstones* is true, also non-propagating off-shell Goldstone amplitudes are included to allow the checking of Slavnov-Taylor identities.

```
type amplitude
type selectors
val amplitudes : bool → selectors →
  flavor_sans_color list → flavor_sans_color list → amplitude list
```

```
val dependencies : amplitude → wf → (wf, coupling) Tree2.t
```

We should be precise regarding the semantics of the following functions, since modules implementating *Target* must not make any mistakes interpreting the return values. Instead of calculating the amplitude

$$\langle f_3, p_3, f_4, p_4, \ldots | T | f_1, p_1, f_2, p_2 \rangle \tag{8.1a}$$

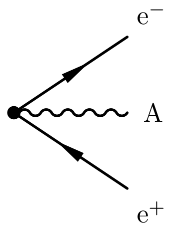
directly, O'Mega calculates the—equivalent, but more symmetrical—crossed amplitude

$$\left\langle \bar f_1, -p_1, \bar f_2, -p_2, f_3, p_3, f_4, p_4, \ldots \middle| T \middle| 0 \right\rangle \tag{8.1b}$$

Internally, all flavors are represented by their charge conjugates

$$A(f_1, -p_1, f_2, -p_2, \bar{f}_3, p_3, \bar{f}_4, p_4, \ldots) \tag{8.1c}$$

The correspondence of vertex and term in the lagrangian



$$: \bar\psi \not{A} \psi \tag{8.2}$$

suggests to denote the *outgoing* particle by the flavor of the *anti*particle and the *outgoing* *anti*particle by the flavor of the particle, since this choice allows to represent the vertex by a triple

$$\bar\psi \not{A} \psi : (\mathrm{e}^+, A, \mathrm{e}^-) \tag{8.3}$$

which is more intuitive than the alternative $(\mathrm{e}^-, A, \mathrm{e}^+)$. Also, when thinking in terms of building wavefunctions from the outside in, the outgoing *antiparticle* is represented by a *particle* propagator and vice versa[1]. *incoming* and *outgoing* are the physical flavors as in (8.1a)

```
val incoming : amplitude → flavor list
val outgoing : amplitude → flavor list
```

externals are flavors and momenta as in (8.1c)

```
val externals : amplitude → wf list

val variables : amplitude → wf list
val fusions : amplitude → fusion list
val brakets : amplitude → braket list
val on_shell : amplitude → (wf → bool)
val is_gauss : amplitude → (wf → bool)
val constraints : amplitude → string option
val symmetry : amplitude → int

val allowed : amplitude → bool
```

Performance Hacks

```
val initialize_cache : string → unit
val set_cache_name : string → unit
```

Diagnostics

```
val check_charges : unit → flavor_sans_color list list
val count_fusions : amplitude → int
```

[1]Even if this choice will appear slightly counter-intuitive on the *Target* side, one must keep in mind that much more people are expected to prepare *Models*.

```
    val count_propagators : amplitude → int
    val count_diagrams : amplitude → int

    val forest : wf → amplitude → ((wf × coupling option, wf) Tree.t) list
    val poles : amplitude → wf list list
    val s_channel : amplitude → wf list

    val tower_to_dot : out_channel → amplitude → unit
    val amplitude_to_dot : out_channel → amplitude → unit

    val rcs_list : RCS.t list
  end
```

There is more than one way to make fusions.

```
module type Maker =
    functor (P : Momentum.T) → functor (M : Model.T) →
      T with type p = P.t
      and type flavor = Colorize.It(M).flavor
      and type flavor_sans_color = M.flavor
      and type constant = M.constant
      and type selectors = Cascade.Make(M)(P).selectors
```

Straightforward Dirac fermions vs. slightly more complicated Majorana fermions:

```
module Binary : Maker
module Binary_Majorana : Maker

module Mixed23 : Maker
module Mixed23_Majorana : Maker

module Nary : functor (B : Tuple.Bound) → Maker
module Nary_Majorana : functor (B : Tuple.Bound) → Maker
```

We can also proceed á la [2]. Empirically, this will use slightly ($O(10\%)$) fewer fusions than the symmetric factorization. Our implementation uses significantly ($O(50\%)$) fewer fusions than reported by [2]. Our pruning of the DAG might be responsible for this.

```
module Helac : functor (B : Tuple.Bound) → Maker
module Helac_Majorana : functor (B : Tuple.Bound) → Maker
```

8.1.1 Multiple Amplitudes

```
module type Multi =
  sig
    exception Mismatch
    val options : Options.t

    type flavor
    type process = flavor list × flavor list
    type amplitude
    type fusion
    type wf
    type selectors
```

type *amplitudes*

Construct all possible color flow amplitudes for a given process.

val *amplitudes* : *bool* → *int option* → *selectors* → *process list* → *amplitudes*
val *empty* : *amplitudes*

Precompute the vertex table cache.

val *initialize_cache* : *string* → *unit*
val *set_cache_name* : *string* → *unit*

The list of all combinations of incoming and outgoing particles with a nonvanishing scattering amplitude.

val *flavors* : *amplitudes* → *process list*

The list of all combinations of incoming and outgoing particles that don't lead to any color flow with non vanishing scattering amplitude.

val *vanishing_flavors* : *amplitudes* → *process list*

The list of all color flows with a nonvanishing scattering amplitude.

val *color_flows* : *amplitudes* → *Color.Flow.t list*

The list of all valid helicity combinations.

val *helicities* : *amplitudes* → (*int list* × *int list*) *list*

The list of all amplitudes.

val *processes* : *amplitudes* → *amplitude list*

(*process_table a*).(f).(c) returns the amplitude for the fth allowed flavor combination and the cth allowed color flow as an *amplitude option*.

val *process_table* : *amplitudes* → *amplitude option array array*

The list of all non redundant fusions together with the amplitudes they came from.

val *fusions* : *amplitudes* → (*fusion* × *amplitude*) *list*

If there's more than external flavor state, the wavefunctions are *not* uniquely specified by *flavor* and *Momentum.t*. This function can be used to determine how many variables must be allocated.

val *multiplicity* : *amplitudes* → *wf* → *int*

This function can be used to disambiguate wavefunctions with the same combination of *flavor* and *Momentum.t*.

val *dictionary* : *amplitudes* → *amplitude* → *wf* → *int*

(*color_factors a*).(*c1*).(*c2*) power of N_C for the given product of color flows.

val *color_factors* : *amplitudes* → *Color.Flow.factor array array*

A description of optional diagram selectors.

val *constraints* : *amplitudes* → *string option*

```
  end

module type Multi_Maker = functor (Fusion_Maker : Maker) →
  functor (P : Momentum.T) →
    functor (M : Model.T) →
      Multi with type flavor = M.flavor
      and type amplitude = Fusion_Maker(P)(M).amplitude
      and type fusion = Fusion_Maker(P)(M).fusion
      and type wf = Fusion_Maker(P)(M).wf
      and type selectors = Fusion_Maker(P)(M).selectors

module Multi : Multi_Maker
```

8.1.2 Tags

It appears that there are useful applications for tagging couplings and wave functions, e. g. skeleton expansion and diagram selections. We can abstract this in a *Tags* signature:

```
module type Tags =
  sig
    type wf
    type coupling
    type α children
    val null_wf : wf
    val null_coupling : coupling
    val fuse : coupling → wf children → wf
    val wf_to_string : wf → string option
    val coupling_to_string : coupling → string option
  end

module type Tagger =
    functor (PT : Tuple.Poly) → Tags with type α children = α PT.t

module type Tagged_Maker =
    functor (Tagger : Tagger) →
      functor (P : Momentum.T) → functor (M : Model.T) →
        T with type p = P.t
        and type flavor = Colorize.It(M).flavor
        and type flavor_sans_color = M.flavor
        and type constant = M.constant

module Tagged_Binary : Tagged_Maker
```

8.2 Implementation of *Fusion*

```
let rcs_file = RCS.parse "Fusion" ["General␣Fusions"]
    { RCS.revision = "$Revision:␣6465␣$";
      RCS.date = "$Date:␣2015-01-10␣16:22:31␣+0100␣(Sat,␣10␣Jan␣2015)␣$";
      RCS.author = "$Author:␣jr_reuter␣$";
      RCS.source
```

```
    = "$URL: svn+ssh://jr_reuter@login.hepforge.org/hepforge/svn/whizard/trunk/omega/sr
module type T =
  sig
    val options : Options.t
    type wf
    val conjugate : wf → wf
    type flavor
    type flavor_sans_color
    val flavor : wf → flavor
    val flavor_sans_color : wf → flavor_sans_color
    type p
    val momentum : wf → p
    val momentum_list : wf → int list
    val wf_tag : wf → string option
    type constant
    type coupling
    type rhs
    type α children
    val sign : rhs → int
    val coupling : rhs → constant Coupling.t
    val coupling_tag : rhs → string option
    val children : rhs → wf list
    type fusion
    val lhs : fusion → wf
    val rhs : fusion → rhs list
    type braket
    val bra : braket → wf
    val ket : braket → rhs list
    type amplitude
    type selectors
    val amplitudes : bool → selectors →
      flavor_sans_color list → flavor_sans_color list → amplitude list
    val dependencies : amplitude → wf → (wf, coupling) Tree2.t
    val incoming : amplitude → flavor list
    val outgoing : amplitude → flavor list
    val externals : amplitude → wf list
    val variables : amplitude → wf list
    val fusions : amplitude → fusion list
    val brakets : amplitude → braket list
    val on_shell : amplitude → (wf → bool)
    val is_gauss : amplitude → (wf → bool)
    val constraints : amplitude → string option
    val symmetry : amplitude → int
    val allowed : amplitude → bool
    val initialize_cache : string → unit
    val set_cache_name : string → unit
    val check_charges : unit → flavor_sans_color list list
    val count_fusions : amplitude → int
    val count_propagators : amplitude → int
```

```
    val count_diagrams : amplitude → int
    val forest : wf → amplitude → ((wf × coupling option, wf) Tree.t) list
    val poles : amplitude → wf list list
    val s_channel : amplitude → wf list
    val tower_to_dot : out_channel → amplitude → unit
    val amplitude_to_dot : out_channel → amplitude → unit
    val rcs_list : RCS.t list
  end

module type Maker =
    functor (P : Momentum.T) → functor (M : Model.T) →
      T with type p = P.t
      and type flavor = Colorize.It(M).flavor
      and type flavor_sans_color = M.flavor
      and type constant = M.constant
      and type selectors = Cascade.Make(M)(P).selectors
```

8.2.1 Fermi Statistics

```
module type Stat =
  sig
    type flavor
    type stat
    exception Impossible
    val stat : flavor → int → stat
    val stat_fuse : stat → stat → flavor → stat
    val stat_sign : stat → int
    val rcs : RCS.t
  end

module type Stat_Maker = functor (M : Model.T) →
  Stat with type flavor = M.flavor
```

8.2.2 Dirac Fermions

```
module Stat_Dirac (M : Model.T) : (Stat with type flavor = M.flavor) =
  struct
    let rcs = RCS.rename rcs_file "Fusion.Stat_Dirac()"
        [ "Fermi␣statistics␣for␣Dirac␣fermions"]

    type flavor = M.flavor
```

$$\gamma_\mu\psi(1)\,G^{\mu\nu}\,\bar\psi(2)\gamma_\nu\psi(3) - \gamma_\mu\psi(3)\,G^{\mu\nu}\,\bar\psi(2)\gamma_\nu\psi(1) \tag{8.4}$$

```
    type stat =
      | Fermion of int × (int option × int option) list
      | AntiFermion of int × (int option × int option) list
      | Boson of (int option × int option) list

    let stat f p =
```

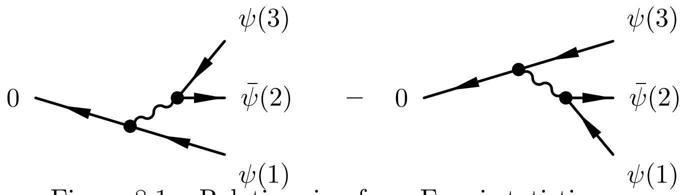


Figure 8.1: Relative sign from Fermi statistics.

```
    let s = M.fermion f in
    if s = 0 then
      Boson []
    else if s < 0 then
      AntiFermion (p, [])
    else (* if s > 0 then *)
      Fermion (p, [])

  exception Impossible

  let stat_fuse s1 s2 f =
    match s1, s2 with
    | Boson l1, Boson l2 → Boson (l1 @ l2)
    | Boson l1, Fermion (p, l2) → Fermion (p, l1 @ l2)
    | Boson l1, AntiFermion (p, l2) → AntiFermion (p, l1 @ l2)
    | Fermion (p, l1), Boson l2 → Fermion (p, l1 @ l2)
    | AntiFermion (p, l1), Boson l2 → AntiFermion (p, l1 @ l2)
    | AntiFermion (pbar, l1), Fermion (p, l2) →
        Boson ((Some pbar, Some p) :: l1 @ l2)
    | Fermion (p, l1), AntiFermion (pbar, l2) →
        Boson ((Some pbar, Some p) :: l1 @ l2)
    | Fermion _, Fermion _ | AntiFermion _, AntiFermion _ →
        raise Impossible
```

$$\epsilon\left(\{(0,1),(2,3)\}\right) = -\epsilon\left(\{(0,3),(2,1)\}\right) \tag{8.5}$$

```
  let permutation lines =
    let fout, fin = List.split lines in
    let eps_in, _ = Combinatorics.sort_signed compare fin
    and eps_out, _ = Combinatorics.sort_signed compare fout in
    (eps_in × eps_out)
```

This comparing of permutations of fermion lines is a bit tedious and takes a macroscopic fraction of time. However, it's less than 20 %, so we don't focus on improving on it yet.

```
  let stat_sign = function
    | Boson lines → permutation lines
    | Fermion (p, lines) → permutation ((None, Some p) :: lines)
    | AntiFermion (pbar, lines) → permutation ((Some pbar, None) :: lines)

end
```

8.2.3 Tags

```
module type Tags =
  sig
    type wf
    type coupling
    type α children
    val null_wf : wf
    val null_coupling : coupling
    val fuse : coupling → wf children → wf
    val wf_to_string : wf → string option
    val coupling_to_string : coupling → string option
  end
```

```
module type Tagger =
    functor (PT : Tuple.Poly) → Tags with type α children = α PT.t
```

```
module type Tagged_Maker =
    functor (Tagger : Tagger) →
      functor (P : Momentum.T) → functor (M : Model.T) →
        T with type p = P.t
        and type flavor = Colorize.It(M).flavor
        and type flavor_sans_color = M.flavor
        and type constant = M.constant
```

No tags is one option for good tags . . .

```
module No_Tags (PT : Tuple.Poly) =
  struct
    type wf = unit
    type coupling = unit
    type α children = α PT.t
    let null_wf = ()
    let null_coupling = ()
    let fuse () _ = ()
    let wf_to_string () = None
    let coupling_to_string () = None
  end
```

Here's a simple additive tag that can grow into something useful for loop calculations.

```
module Loop_Tags (PT : Tuple.Poly) =
  struct
    type wf = int
    type coupling = int
    type α children = α PT.t
    let null_wf = 0
    let null_coupling = 0
    let fuse c wfs = PT.fold_left (+) c wfs
    let wf_to_string n = Some (string_of_int n)
    let coupling_to_string n = Some (string_of_int n)
```

```
  end

module Order_Tags (PT : Tuple.Poly) =
  struct
    type wf = int
    type coupling = int
    type α children = α PT.t
    let null_wf = 0
    let null_coupling = 0
    let fuse c wfs = PT.fold_left (+) c wfs
    let wf_to_string n = Some (string_of_int n)
    let coupling_to_string n = Some (string_of_int n)
  end
```

8.2.4 Tagged, the Fusion.Make Functor

```
module Tagged (Tagger : Tagger) (PT : Tuple.Poly)
    (Stat : Stat_Maker) (T : Topology.T with type α children = α PT.t)
    (P : Momentum.T) (M : Model.T) =
  struct
    let rcs = RCS.rename rcs_file "Fusion.Make()"
        [ "Fusions␣for␣arbitrary␣topologies" ]

    type cache_mode = Cache_Use | Cache_Ignore | Cache_Overwrite
    let cache_option = ref Cache_Use
    type qcd_order =
      | QCD_order of int
    type ew_order =
      | EW_order of int
    let qcd_order = ref (QCD_order 99)
    let ew_order = ref (EW_order 99)

    let options = Options.create
        [ "ignore-cache", Arg.Unit (fun () → cache_option := Cache_Ignore),
          "ignore␣cached␣model␣tables";
          "overwrite-cache", Arg.Unit (fun () → cache_option := Cache_Overwrite),
          "overwrite␣cached␣model␣tables";
          "qcd", Arg.Int (fun n → qcd_order := QCD_order n),
          "set␣QCD␣order␣n␣[>=␣0,␣default␣=␣99]";
          "ew", Arg.Int (fun n → ew_order := EW_order n),
          "set␣QCD␣order␣n␣[>=0,␣default␣=␣99]"]

    exception Negative_QCD_order
    exception Negative_EW_order
    exception Vanishing_couplings
    exception Negative_QCD_EW_orders

    let int_orders =
      match !qcd_order, !ew_order with
        | QCD_order n, EW_order n' when n < 0 ∧ n' ≥ 0 →
            raise Negative_QCD_order
        | QCD_order n, EW_order n' when n ≥ 0 ∧ n' < 0 →
```

```
      raise Negative_EW_order
  | QCD_order n, EW_order n' when n < 0 ∧ n' < 0 →
      raise Negative_QCD_EW_orders
  | QCD_order n, EW_order n' → (n, n')
```

```
open Coupling
```

```
module S = Stat(M)
```

```
type stat = S.stat
let stat = S.stat
let stat_sign = S.stat_sign
```

This will do *something* for 4-, 6-, ... fermion vertices, but not necessarily the right thing ...

```
let stat_fuse s f =
  PT.fold_right_internal (fun s' acc → S.stat_fuse s' acc f) s
```

```
type constant = M.constant
```

Wave Functions

The code below is not yet functional. Too often, we assign to *Tags.null_wf* instead of calling *Tags.fuse*.

We will need two types of amplitudes: with color and without color. Since we can build them using the same types with only *flavor* replaced, it pays to use a functor to set up the scaffolding.

```
module Tags = Tagger(PT)
```

In the future, we might want to have *Coupling* among the functor arguments. However, for the moment, *Coupling* is assumed to be comprehensive.

```
module type Tagged_Coupling =
  sig
    type sign = int
    type t =
        { sign : sign;
          coupling : constant Coupling.t;
          coupling_tag : Tags.coupling }
    val sign : t → sign
    val coupling : t → constant Coupling.t
    val coupling_tag : t → string option
  end
```

```
module Tagged_Coupling : Tagged_Coupling =
  struct
    type sign = int
    type t =
        { sign : sign;
          coupling : constant Coupling.t;
```

```
        coupling_tag : Tags.coupling }
    let sign c = c.sign
    let coupling c = c.coupling
    let coupling_tag_raw c = c.coupling_tag
    let coupling_tag rhs = Tags.coupling_to_string (coupling_tag_raw rhs)
  end
```

Amplitudes: Monochrome and Colored

```
module type Amplitude =
  sig

    module Tags : Tags

    type flavor
    type p

    type wf =
        { flavor : flavor;
          momentum : p;
          wf_tag : Tags.wf }

    val flavor : wf → flavor
    val conjugate : wf → wf
    val momentum : wf → p
    val momentum_list : wf → int list
    val wf_tag : wf → string option
    val wf_tag_raw : wf → Tags.wf
    val order_wf : wf → wf → int
    val external_wfs : int → (flavor × int) list → wf list

    type α children
    type coupling = Tagged_Coupling.t
    type rhs = coupling × wf children
    val sign : rhs → int
    val coupling : rhs → constant Coupling.t
    val coupling_tag : rhs → string option
    val children : rhs → wf list

    type fusion = wf × rhs list
    val lhs : fusion → wf
    val rhs : fusion → rhs list

    type braket = wf × rhs list
    val bra : braket → wf
    val ket : braket → rhs list

    module D :
        DAG.T with type node = wf and type edge = coupling and type children = wf children

    val wavefunctions : braket list → wf list

    type amplitude =
        { fusions : fusion list;
```

```
        brakets : braket list;
        on_shell : (wf → bool);
        is_gauss : (wf → bool);
        constraints : string option;
        incoming : flavor list;
        outgoing : flavor list;
        externals : wf list;
        symmetry : int;
        dependencies : (wf → (wf, coupling) Tree2.t);
        fusion_tower : D.t;
        fusion_dag : D.t }

    val incoming : amplitude → flavor list
    val outgoing : amplitude → flavor list
    val externals : amplitude → wf list
    val variables : amplitude → wf list
    val fusions : amplitude → fusion list
    val brakets : amplitude → braket list
    val on_shell : amplitude → (wf → bool)
    val is_gauss : amplitude → (wf → bool)
    val constraints : amplitude → string option
    val symmetry : amplitude → int
    val dependencies : amplitude → wf → (wf, coupling) Tree2.t
    val fusion_dag : amplitude → D.t

  end

module Amplitude (PT : Tuple.Poly) (P : Momentum.T) (M : Model.T) :
    Amplitude
    with type p = P.t
    and type flavor = M.flavor
    and type α children = α PT.t
    and module Tags = Tags =
  struct

    type flavor = M.flavor
    type p = P.t

    module Tags = Tags

    type wf =
        { flavor : flavor;
          momentum : p;
          wf_tag : Tags.wf }

    let flavor wf = wf.flavor
    let conjugate wf = { wf with flavor = M.conjugate wf.flavor }
    let momentum wf = wf.momentum
    let momentum_list wf = P.to_ints wf.momentum
    let wf_tag wf = Tags.wf_to_string wf.wf_tag
    let wf_tag_raw wf = wf.wf_tag

    let external_wfs rank particles =
      List.map
```

```
(fun (f, p) →
  { flavor = f;
    momentum = P.singleton rank p;
    wf_tag = Tags.null_wf })
particles
```

Order wavefunctions so that the external come first, then the pairs, etc. Also put possible Goldstone bosons *before* their gauge bosons.

```
let lorentz_ordering f =
  match M.lorentz f with
  | Coupling.Scalar → 0
  | Coupling.Spinor → 1
  | Coupling.ConjSpinor → 2
  | Coupling.Majorana → 3
  | Coupling.Vector → 4
  | Coupling.Massive_Vector → 5
  | Coupling.Tensor_2 → 6
  | Coupling.Tensor_1 → 7
  | Coupling.Vectorspinor → 8
  | Coupling.BRS Coupling.Scalar → 9
  | Coupling.BRS Coupling.Spinor → 10
  | Coupling.BRS Coupling.ConjSpinor → 11
  | Coupling.BRS Coupling.Majorana → 12
  | Coupling.BRS Coupling.Vector → 13
  | Coupling.BRS Coupling.Massive_Vector → 14
  | Coupling.BRS Coupling.Tensor_2 → 15
  | Coupling.BRS Coupling.Tensor_1 → 16
  | Coupling.BRS Coupling.Vectorspinor → 17
  | Coupling.BRS _ → invalid_arg "Fusion.lorentz_ordering: not needed"
  | Coupling.Maj_Ghost → 18

let order_flavor f1 f2 =
  let c = compare (lorentz_ordering f1) (lorentz_ordering f2) in
  if c ≠ 0 then
    c
  else
    compare f1 f2
```

Note that *Momentum*().*compare* guarantees that wavefunctions will be ordered according to *increasing* *Momentum*().*rank* of their momenta.

```
let order_wf wf1 wf2 =
  let c = P.compare wf1.momentum wf2.momentum in
  if c ≠ 0 then
    c
  else
    let c = order_flavor wf1.flavor wf2.flavor in
    if c ≠ 0 then
      c
    else
      compare wf1.wf_tag wf2.wf_tag
```

This *must* be a pair matching the *edge* × *node children* pairs of *DAG.Forest*!

```
type coupling = Tagged_Coupling.t
type α children = α PT.t
type rhs = coupling × wf children
let sign (c, _) = Tagged_Coupling.sign c
let coupling (c, _) = Tagged_Coupling.coupling c
let coupling_tag (c, _) = Tagged_Coupling.coupling_tag c
let children (_, wfs) = PT.to_list wfs

type fusion = wf × rhs list
let lhs (l, _) = l
let rhs (_, r) = r

type braket = wf × rhs list
let bra (b, _) = b
let ket (_, k) = k

module D = DAG.Make
    (DAG.Forest(PT)
        (struct type t = wf let compare = order_wf end)
        (struct type t = coupling let compare = compare end))

module WFSet =
  Set.Make (struct type t = wf let compare = order_wf end)

let wavefunctions brakets =
  WFSet.elements (List.fold_left (fun set (wf1, wf23) →
    WFSet.add wf1 (List.fold_left (fun set′ (_, wfs) →
      PT.fold_right WFSet.add wfs set′) set wf23)) WFSet.empty brakets)

type amplitude =
    { fusions : fusion list;
      brakets : braket list;
      on_shell : (wf → bool);
      is_gauss : (wf → bool);
      constraints : string option;
      incoming : flavor list;
      outgoing : flavor list;
      externals : wf list;
      symmetry : int;
      dependencies : (wf → (wf, coupling) Tree2.t);
      fusion_tower : D.t;
      fusion_dag : D.t }

let incoming a = a.incoming
let outgoing a = a.outgoing
let externals a = a.externals
let fusions a = a.fusions
let brakets a = a.brakets
let symmetry a = a.symmetry
let on_shell a = a.on_shell
let is_gauss a = a.is_gauss
let constraints a = a.constraints
```

```
    let variables a = List.map lhs a.fusions
    let dependencies a = a.dependencies
    let fusion_dag a = a.fusion_dag

  end

module A = Amplitude(PT)(P)(M)
```

Operator insertions can be fused only if they are external.

```
let is_source wf =
  match M.propagator wf.A.flavor with
  | Only_Insertion → P.rank wf.A.momentum = 1
  | _ → true
```

is_goldstone_of g v is true if and only if g is the Goldstone boson corresponding to the gauge particle v.

```
let is_goldstone_of g v =
  match M.goldstone v with
  | None → false
  | Some (g′, _) → g = g′
```

In the end, *PT.to_list* should become redudant!

```
let fuse_rhs rhs = M.fuse (PT.to_list rhs)
```

Vertices

Compute the set of all vertices in the model from the allowed fusions and the set of all flavors:

One could think of using *M.vertices* instead of *M.fuse2*, *M.fuse3* and *M.fuse* ...

```
module VSet = Map.Make(struct type t = A.flavor let compare = compare end)

let add_vertices f rhs m =
  VSet.add f (try rhs :: VSet.find f m with Not_found → [rhs]) m

let collect_vertices rhs =
  List.fold_right (fun (f1, c) → add_vertices (M.conjugate f1) (c, rhs))
    (fuse_rhs rhs)
```

The set of all vertices with common left fields factored.
I used to think that constant initializers are a good idea to allow compile time optimizations. The down side turned out to be that the constant initializers will be evaluated *every time* the functor is applied. *Relying on the fact that the functor will be called only once is not a good idea!*

```
type vertices = (A.flavor × (constant Coupling.t × A.flavor PT.t) list) list

let vertices_nocache max_degree flavors : vertices =
  VSet.fold (fun f rhs v → (f, rhs) :: v)
    (PT.power_fold collect_vertices flavors VSet.empty) []
```

Performance hack:

```
type vertex_table =
    ((A.flavor × A.flavor × A.flavor) × constant Coupling.vertex3 × constant) list
  × ((A.flavor × A.flavor × A.flavor × A.flavor)
       × constant Coupling.vertex4 × constant) list
  × (A.flavor list × constant Coupling.vertexn × constant) list

module VCache =
  Cache.Make (struct type t = vertex_table end) (struct type t = RCS.t × vertices end)

let vertices_cache = ref None
let hash = VCache.hash (M.vertices ())
```

Can we do better than the executable name provided by *Config.cache_prefix*??? We need a better way to avoid collisions among the caches for different models in the same program.

```
let cache_name =
  ref (Config.cache_prefix ^ "." ^ Config.cache_suffix)

let set_cache_name name =
  cache_name := name

let initialize_cache dir =
  Printf.eprintf
    " >>> Initializing vertex table for model %s.  This may take some time ... "
    (RCS.name M.rcs);
  flush stderr;
  VCache.write_dir hash dir !cache_name
    (M.rcs, vertices_nocache (M.max_degree ()) (M.flavors()));
  Printf.eprintf "done. <<< \n"

let vertices max_degree flavors : vertices =
  match !vertices_cache with
  | None →
      begin match !cache_option with
      | Cache_Use →
          begin match VCache.maybe_read hash !cache_name with
          | VCache.Hit (rcs, result) →
              result
          | VCache.Miss →
              Printf.eprintf
                " >>> Initializing vertex table for model %s.  This may take some time
                (RCS.name M.rcs);
              flush stderr;
              let result = vertices_nocache max_degree flavors in
              VCache.write hash !cache_name (M.rcs, result);
              vertices_cache := Some result;
              Printf.eprintf "done. <<< \n";
              flush stderr;
```

```
          result
      | VCache.Stale file →
          Printf.eprintf
            " >>> Re-initializing stale vertex table for model %s in file %s.  "
            (RCS.name M.rcs) file;
          Printf.eprintf "This may take some time ... ";
          flush stderr;
          let result = vertices_nocache max_degree flavors in
          VCache.write hash !cache_name (M.rcs, result);
          vertices_cache := Some result;
          Printf.eprintf "done. <<< \n";
          flush stderr;
          result
      end
    | Cache_Overwrite →
        Printf.eprintf
          " >>> Overwriting vertex table for model %s.  This may take some time ...
          (RCS.name M.rcs);
        flush stderr;
        let result = vertices_nocache max_degree flavors in
        VCache.write hash !cache_name (M.rcs, result);
        vertices_cache := Some result;
        Printf.eprintf "done. <<< \n";
        flush stderr;
        result
    | Cache_Ignore →
        Printf.eprintf
          " >>> Ignoring vertex table for model %s.  This may take some time ... "
          (RCS.name M.rcs);
        flush stderr;
        let result = vertices_nocache max_degree flavors in
        vertices_cache := Some result;
        Printf.eprintf "done. <<< \n";
        flush stderr;
        result
    end
  | Some result → result
```

Partitions

Vertices that are not crossing invariant need special treatment so that they're only generated for the correct combinations of momenta.

NB: the *crossing* checks here are a bit redundant, because *CM.fuse* below will bring the killed vertices back to life and will have to filter once more. Nevertheless, we keep them here, for the unlikely case that anybody ever wants to use uncolored amplitudes directly.

NB: the analogous problem does not occur for *select_wf*, because this applies to momenta instead of vertices.

This approach worked before the colorize, but has become *futile*, because *CM.fuse* will bring the killed vertices back to life. We need to implement the same checks there again!!!

Using *PT.Mismatched_arity* is not really good style ...

Tho's approach doesn't work since he does not catch charge conjugated processes or crossed processes. Another very strange thing is that O'Mega seems always to run in the q2 q3 timelike case, but not in the other two. (Property of how the DAG is built?). For the $ZZZZ$ vertex I add the same vertex again, but interchange 1 and 3 in the *crossing* vertex

```
let kmatrix_cuts c momenta =
  match c with
  | V4 (Vector4_K_Matrix_tho (disc, _), fusion, _)
  | V4 (Vector4_K_Matrix_jr (disc, _), fusion, _) →
      let s12, s23, s13 =
        begin match PT.to_list momenta with
        | [q1; q2; q3] → (P.Scattering.timelike (P.add q1 q2),
                           P.Scattering.timelike (P.add q2 q3),
                           P.Scattering.timelike (P.add q1 q3))
        | _ → raise PT.Mismatched_arity
        end in
      begin match disc, s12, s23, s13, fusion with
      | 0, true, false, false, (F341 | F431 | F342 | F432 | F123 | F213 |
F124 | F214)
      | 0, false, true, false, (F134 | F143 | F234 | F243 | F312 | F321 |
F412 | F421)
      | 0, false, false, true, (F314 | F413 | F324 | F423 | F132 | F231 |
F142 | F241) →
          true
      | 1, true, false, false, (F341 | F431 | F342 | F432)
      | 1, false, true, false, (F134 | F143 | F234 | F243)
      | 1, false, false, true, (F314 | F413 | F324 | F423) →
          true
      | 2, true, false, false, (F123 | F213 | F124 | F214)
      | 2, false, true, false, (F312 | F321 | F412 | F421)
      | 2, false, false, true, (F132 | F231 | F142 | F241) →
          true
      | 3, true, false, false, (F143 | F413 | F142 | F412 | F321 | F231 |
F324 | F234)
      | 3, false, true, false, (F314 | F341 | F214 | F241 | F132 | F123 |
F432 | F423)
      | 3, false, false, true, (F134 | F431 | F124 | F421 | F312 | F213 |
F342 | F243) →
          true
      | _ → false
      end
  | _ → true
```

Counting QCD and EW orders.

```
let qcd_ew_check orders =
  if fst (orders) ≤ fst (int_orders) ∧
      snd (orders) ≤ snd (int_orders) then
    true
  else
    false
```

Match a set of flavors to a set of momenta. Form the direct product for the lists of momenta two and three with the list of couplings and flavors two and three.

```
let flavor_keystone select_p dim (f1, f23) (p1, p23) =
  ({ A.flavor = f1;
      A.momentum = P.of_ints dim p1;
      A.wf_tag = A.Tags.null_wf },
   Product.fold2 (fun (c, f) p acc →
     try
       let p′ = PT.map (P.of_ints dim) p in
       if select_p (P.of_ints dim p1) (PT.to_list p′) ∧ kmatrix_cuts c p′ then
         (c, PT.map2 (fun f″ p″ → { A.flavor = f″;
                                     A.momentum = p″;
                                     A.wf_tag = A.Tags.null_wf }) f p′) :: acc
       else
         acc
     with
     | PT.Mismatched_arity → acc) f23 p23 [])
```

Produce all possible combinations of vertices (flavor keystones) and momenta by forming the direct product. The semantically equivalent *Product.list2* (*flavor_keystone select_wf n*) *vertices k* with *subsequent* filtering would be a *very bad* idea, because a potentially huge intermediate list is built for large models. E. g. for the MSSM this would lead to non-termination by thrashing for $2 \to 4$ processes on most PCs.

```
let flavor_keystones filter select_p dim vertices keystones =
  Product.fold2 (fun v k acc →
    filter (flavor_keystone select_p dim v k) acc) vertices keystones []
```

Flatten the nested lists of vertices into a list of attached lines.

```
let flatten_keystones t =
  ThoList.flatmap (fun (p1, p23) →
    p1 :: (ThoList.flatmap (fun (_, rhs) → PT.to_list rhs) p23)) t
```

Subtrees

Fuse a tuple of wavefunctions, keeping track of Fermi statistics. Record only the the sign *relative* to the children. (The type annotation is only for documentation.)

```
let fuse select_wf wfss : (A.wf × stat × A.rhs) list =
  if PT.for_all (fun (wf, _) → is_source wf) wfss then
    try
      let wfs, ss = PT.split wfss in
      let flavors = PT.map A.flavor wfs
```

```
          and momenta = PT.map A.momentum wfs
          and wf_tags = PT.map A.wf_tag_raw wfs in
          let p = PT.fold_left_internal P.add momenta in
          List.fold_left
            (fun acc (f, c) →
              if select_wf f p (PT.to_list momenta) ∧ kmatrix_cuts c momenta then
                let s = stat_fuse ss f in
                let flip =
                  PT.fold_left (fun acc s' → acc × stat_sign s') (stat_sign s) ss in
                ({ A.flavor = f;
                   A.momentum = p;
                   A.wf_tag = A.Tags.null_wf }, s,
                 ({ Tagged_Coupling.sign = flip;
                    Tagged_Coupling.coupling = c;
                    Tagged_Coupling.coupling_tag = A.Tags.null_coupling }, wfs)) :: acc
              else
                acc)
            [] (fuse_rhs flavors)
        with
        | P.Duplicate _ | S.Impossible → []
      else
        []
```

Eventually, the pairs of *tower* and *dag* in *fusion_tower'* below could and should be replaced by a graded *DAG*. This will look like, but currently *tower* containts statistics information that is missing from *dag*:

```
Type node = flavor * p is not compatible with type wf * stat
```

This should be easy to fix. However, replacing type $t = wf$ with type $t = wf \times stat$ is *not* a good idea because the variable *stat* makes it impossible to test for the existance of a particular *wf* in a *DAG*.

In summary, it seems that $(wf \times stat)$ *list array* $\times A.D.t$ should be replaced by $(wf \rightarrow stat) \times A.D.t$.

```
  module GF =
    struct
      module Nodes =
        struct
          type t = A.wf
          module G = struct type t = int let compare = compare end
          let compare = A.order_wf
          let rank wf = P.rank wf.A.momentum
        end
      module Edges = struct type t = A.coupling let compare = compare end
      module F = DAG.Forest(PT)(Nodes)(Edges)
      type node = Nodes.t
      type edge = F.edge
      type children = F.children
```

```
    type t = F.t
    let compare = F.compare
    let for_all = F.for_all
    let fold = F.fold
  end

module D' = DAG.Graded(GF)

let tower_of_dag dag =
  let _, max_rank = D'.min_max_rank dag in
  Array.init max_rank (fun n → D'.ranked n dag)
```

The function *fusion_tower'* recursively builds the tower of all fusions from bottom up to a chosen level. The argument *tower* is an array of lists, where the i-th sublist (counting from 0) represents all off shell wave functions depending on $i+1$ momenta and their Fermistatistics.

$$
\begin{aligned}
\Big[&\{\phi_1(p_1),\phi_2(p_2),\phi_3(p_3),\ldots\},\\
&\{\phi_{12}(p_1+p_2),\phi'_{12}(p_1+p_2),\ldots,\phi_{13}(p_1+p_3),\ldots,\phi_{23}(p_2+p_3),\ldots\},\\
&\ldots\\
&\{\phi_{1\cdots n}(p_1+\cdots+p_n),\phi'_{1\cdots n}(p_1+\cdots+p_n),\ldots\}\Big]
\end{aligned}
\tag{8.6}
$$

The argument *dag* is a DAG representing all the fusions calculated so far. NB: The outer array in *tower* is always very short, so we could also have accessed a list with *List.nth*. Appending of new members at the end brings no loss of performance. NB: the array is supposed to be immutable.
The towers must be sorted so that the combinatorical functions can make consistent selections.

Intuitively, this seems to be correct. However, one could have expected that no element appears twice and that this ordering is not necessary ...

```
let grow select_wf tower =
  let rank = succ (Array.length tower) in
  List.sort Pervasives.compare
    (PT.graded_sym_power_fold rank
       (fun wfs acc → fuse select_wf wfs @ acc) tower [])

let add_offspring dag (wf, _, rhs) =
  A.D.add_offspring wf rhs dag

let filter_offspring fusions =
  List.map (fun (wf, s, _) → (wf, s)) fusions

let rec fusion_tower' n_max select_wf tower dag : (A.wf × stat) list array × A.D.t =
  if Array.length tower ≥ n_max then
    (tower, dag)
  else
    let tower' = grow select_wf tower in
    fusion_tower' n_max select_wf
      (Array.append tower [|filter_offspring tower'|])
```

```
        (List.fold_left add_offspring dag tower')
```

Discard the tower and return a map from wave functions to Fermistatistics together with the DAG.

```
let make_external_dag wfs =
  List.fold_left (fun m (wf, _) → A.D.add_node wf m) A.D.empty wfs
```

```
let mixed_fold_left f acc lists =
  Array.fold_left (List.fold_left f) acc lists
```

```
module Stat_Map =
  Map.Make (struct type t = A.wf let compare = A.order_wf end)
```

```
let fusion_tower height select_wf wfs : (A.wf → stat) × A.D.t =
  let tower, dag =
    fusion_tower' height select_wf [|wfs|] (make_external_dag wfs) in
  let stats = mixed_fold_left
      (fun m (wf, s) → Stat_Map.add wf s m) Stat_Map.empty tower in
  ((fun wf → Stat_Map.find wf stats), dag)
```

Calculate the minimal tower of fusions that suffices for calculating the amplitude.

```
let minimal_fusion_tower n select_wf wfs : (A.wf → stat) × A.D.t =
  fusion_tower (T.max_subtree n) select_wf wfs
```

Calculate the complete tower of fusions. It is much larger than required, but it allows a complete set of gauge checks.

```
let complete_fusion_tower select_wf wfs : (A.wf → stat) × A.D.t =
  fusion_tower (List.length wfs − 1) select_wf wfs
```

There is a natural product of two DAGs using *fuse*. Can this be used in a replacement for *fusion_tower*? The hard part is to avoid double counting, of course. A straight forward solution could do a diagonal sum (in order to reject flipped offspring representing the same fusion) and rely on the uniqueness in *DAG* otherwise. However, this will (probably) slow down the procedure significanty, because most fusions (including Fermi signs!) will be calculated before being rejected by *DAG().add_offspring*.

Add to *dag* all Goldstone bosons defined in *tower* that correspond to gauge bosons in *dag*. This is only required for checking Slavnov-Taylor identities in unitarity gauge. Currently, it is not used, because we use the complete tower for gauge checking.

```
let harvest_goldstones tower dag =
  A.D.fold_nodes (fun wf dag' →
    match M.goldstone wf.A.flavor with
    | Some (g, _) →
        let wf' = { wf with A.flavor = g } in
        if A.D.is_node wf' tower then begin
          A.D.harvest tower wf' dag'
        end else begin
          dag'
```

```
        end
    | None → dag′) dag dag
```

Calculate the sign from Fermi statistics that is not already included in the children.

The use of *PT.of2_kludge* is the largest skeleton on the cupboard of unified fusions. Currently, it is just another name for *PT.of2*, but the existence of the latter requires binary fusions. Of course, this is just a symptom for not fully supporting four fermion vertices …

```
let stat_keystone stats wf1 wfs =
  let wf1′ = stats wf1
  and wfs′ = PT.map stats wfs in
  stat_sign
    (stat_fuse
       (PT.of2_kludge wf1′ (stat_fuse wfs′ (M.conjugate (A.flavor wf1))))
       (A.flavor wf1))
    × PT.fold_left (fun acc wf → acc × stat_sign wf) (stat_sign wf1′) wfs′
```

Test all members of a list of wave functions are defined by the DAG simultaneously:

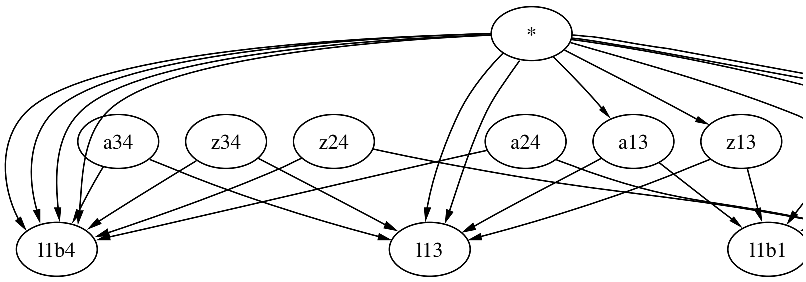
```
let test_rhs dag (_, wfs) =
  PT.for_all (fun wf → is_source wf ∧ A.D.is_node wf dag) wfs
```

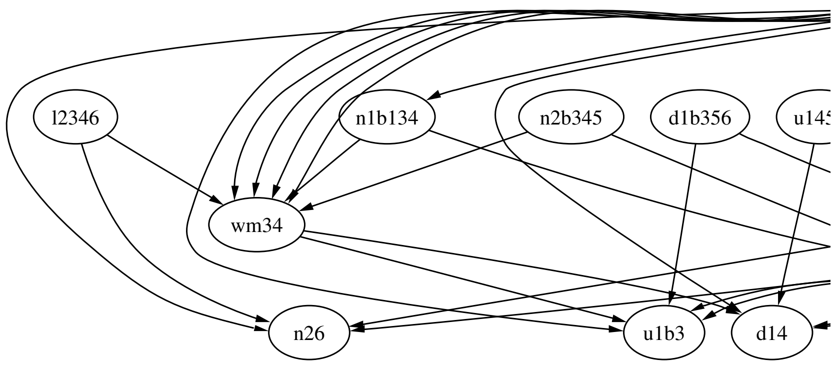
Add the keystone (*wf1*, *pairs*) to *acc* only if it is present in *dag* and calculate the statistical factor depending on *stats en passant*:

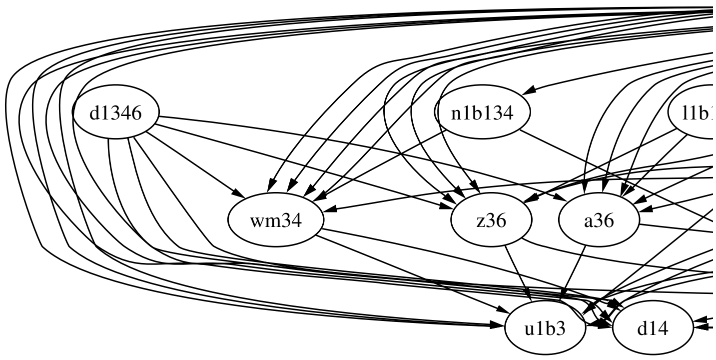
```
let filter_keystone stats dag (wf1, pairs) acc =
  if is_source wf1 ∧ A.D.is_node wf1 dag then
    match List.filter (test_rhs dag) pairs with
    | [] → acc
    | pairs′ → (wf1, List.map (fun (c, wfs) →
        ({ Tagged_Coupling.sign = stat_keystone stats wf1 wfs;
           Tagged_Coupling.coupling = c;
           Tagged_Coupling.coupling_tag = A.Tags.null_coupling },
         wfs)) pairs′) :: acc
  else
    acc
```

Amplitudes

```
module C = Cascade.Make(M)(P)
type selectors = C.selectors

let external_wfs n particles =
  List.map (fun (f, p) →
    ({ A.flavor = f;
       A.momentum = P.singleton n p;
       A.wf_tag = A.Tags.null_wf },
     stat f p)) particles
```


*
a34
z34
z24
a24
a13
z13
l1b4
l13
l1b1


l2346
n1b134
n2b345
d1b356
u145
wm34
n26
u1b3
d14


d1346
n1b134
l1b
wm34
z36
a36
u1b3
d14

Main Function

```
module WFMap = Map.Make (struct type t = A.wf let compare = compare end)
```

map_amplitude_wfs f a applies the function $f : wf \to wf$ to all wavefunctions appearing in the amplitude a.

```
let map_amplitude_wfs f a =
  let map_rhs (c, wfs) = (c, PT.map f wfs) in
  let map_braket (wf, rhs) = (f wf, List.map map_rhs rhs)
  and map_fusion (lhs, rhs) = (f lhs, List.map map_rhs rhs) in
  let map_dag = A.D.map f (fun node rhs → map_rhs rhs) in
  let tower = map_dag a.A.fusion_tower
  and dag = map_dag a.A.fusion_dag in
  let dependencies_map =
    A.D.fold (fun wf _ → WFMap.add wf (A.D.dependencies dag wf)) dag WFMap.empty in
  { A.fusions = List.map map_fusion a.A.fusions;
    A.brakets = List.map map_braket a.A.brakets;
    A.on_shell = a.A.on_shell;
    A.is_gauss = a.A.is_gauss;
    A.constraints = a.A.constraints;
    A.incoming = a.A.incoming;
    A.outgoing = a.A.outgoing;
    A.externals = List.map f a.A.externals;
    A.symmetry = a.A.symmetry;
    A.dependencies = (fun wf → WFMap.find wf dependencies_map);
    A.fusion_tower = tower;
    A.fusion_dag = dag }
```

This is the main function that constructs the amplitude for sets of incoming and outgoing particles and returns the results in conveniently packaged pieces.

```
let amplitude goldstones selectors fin fout =
```

Set up external lines and match flavors with numbered momenta.

```
  let f = fin @ List.map M.conjugate fout in
  let nin, nout = List.length fin, List.length fout in
  let n = nin + nout in
  let externals = List.combine f (ThoList.range 1 n) in
  let wfs = external_wfs n externals in
  let select_p = C.select_p selectors in
  let select_wf =
    match fin with
    | [_] → C.select_wf selectors P.Decay.timelike
    | _ → C.select_wf selectors P.Scattering.timelike in
```

Build the full fusion tower (including nodes that are never needed in the amplitude).

```
  let stats, tower =

    if goldstones then
      complete_fusion_tower select_wf wfs
```

```
      else
        minimal_fusion_tower n select_wf wfs in
```

Find all vertices for which *all* off shell wavefunctions are defined by the tower.

```
    let brakets =
      flavor_keystones (filter_keystone stats tower) select_p n
        (vertices (M.max_degree ()) (M.flavors ()))
        (T.keystones (ThoList.range 1 n)) in
```

Remove the part of the DAG that is never needed in the amplitude.

```
    let dag =
      if goldstones then
        tower
      else
        A.D.harvest_list tower (A.wavefunctions brakets) in
```

Remove the leaf nodes of the DAG, corresponding to external lines.

```
    let fusions =
      List.filter (function (_, []) → false | _ → true) (A.D.lists dag) in
```

Calculate the symmetry factor for identical particles in the final state.

```
    let symmetry =
      Combinatorics.symmetry fout in

    let dependencies_map =
      A.D.fold (fun wf _ → WFMap.add wf (A.D.dependencies dag wf)) dag WFMap.empty in
```

Finally: package the results:

```
    { A.fusions = fusions;
      A.brakets = brakets;
      A.on_shell = (fun wf → C.on_shell selectors (A.flavor wf) wf.A.momentum);
      A.is_gauss = (fun wf → C.is_gauss selectors (A.flavor wf) wf.A.momentum);
      A.constraints = C.description selectors;
      A.incoming = fin;
      A.outgoing = fout;
      A.externals = List.map fst wfs;
      A.symmetry = symmetry;
      A.dependencies = (fun wf → WFMap.find wf dependencies_map);
      A.fusion_tower = tower;
      A.fusion_dag = dag }
```

Color

```
  module CM = Colorize.It(M)
  module CA = Amplitude(PT)(P)(CM)

  let colorize_wf flavor wf =
    { CA.flavor = flavor;
      CA.momentum = wf.A.momentum;
      CA.wf_tag = wf.A.wf_tag }
```

```
let uncolorize_wf wf =
  { A.flavor = CM.flavor_sans_color wf.CA.flavor;
    A.momentum = wf.CA.momentum;
    A.wf_tag = wf.CA.wf_tag }
```

At the end of the day, I shall want to have some sort of *fibered DAG* as abstract data type, with a projection of colored nodes to their uncolored counterparts.

```
module CWFBundle = Bundle.Make
    (struct
      type elt = CA.wf
      let compare_elt = compare
      type base = A.wf
      let compare_base = compare
      let pi wf =
        { A.flavor = CM.flavor_sans_color wf.CA.flavor;
          A.momentum = wf.CA.momentum;
          A.wf_tag = wf.CA.wf_tag }
    end)
```

For now, we can live with simple aggregation:

```
type fibered_dag = { dag : CA.D.t; bundle : CWFBundle.t }
```

Not yet(?) needed: `module CS = Stat (CM)`

```
let colorize_sterile_nodes dag f wf fibered_dag =
  if A.D.is_sterile wf dag then
    let wf', wf_bundle' = f wf fibered_dag in
    { dag = CA.D.add_node wf' fibered_dag.dag;
      bundle = wf_bundle' }
  else
    fibered_dag
```

```
let colorize_nodes f wf rhs fibered_dag =
  let wf_rhs_list', wf_bundle' = f wf rhs fibered_dag in
  let dag' =
    List.fold_right
      (fun (wf', rhs') → CA.D.add_offspring wf' rhs')
      wf_rhs_list' fibered_dag.dag in
  { dag = dag';
    bundle = wf_bundle' }
```

O'Caml (correctly) infers the type `val` $colorize\_dag$: $(D.node \rightarrow D.edge \times D.children \rightarrow fibered\_dag \rightarrow (CA.D.node \times (CA.D.edge \times CA.D.children))\ list \times CWFBundle.t) \rightarrow (D.node \rightarrow fibered\_dag \rightarrow CA.D.node \times CWFBundle.t) \rightarrow D.t \rightarrow CWFBundle.t \rightarrow fibered\_dag$.

```
let colorize_dag f_node f_ext dag wf_bundle =
  A.D.fold (colorize_nodes f_node) dag
    (A.D.fold_nodes (colorize_sterile_nodes dag f_ext) dag
```

```
        { dag = CA.D.empty; bundle = wf_bundle })

let colorize_external wf fibered_dag =
  match CWFBundle.inv_pi wf fibered_dag.bundle with
  | [c_wf] → (c_wf, fibered_dag.bundle)
  | [] → failwith "colorize_external: not found"
  | _ → failwith "colorize_external: not unique"

let fuse_c_wf rhs =
  let momenta = PT.map (fun wf → wf.CA.momentum) rhs in
  List.filter
    (fun (_, c) → kmatrix_cuts c momenta)
    (CM.fuse (List.map (fun wf → wf.CA.flavor) (PT.to_list rhs)))

let colorize_coupling c coupling =
    { coupling with Tagged_Coupling.coupling = c }

let colorize_fusion wf (coupling, children) fibered_dag =
  let match_flavor (f, _) = (CM.flavor_sans_color f = A.flavor wf)
  and find_colored wf' = CWFBundle.inv_pi wf' fibered_dag.bundle in
  let fusions =
    ThoList.flatmap
      (fun c_children →
        List.map
          (fun (f, c) →
            (colorize_wf f wf, (colorize_coupling c coupling, c_children)))
          (List.filter match_flavor (fuse_c_wf c_children)))
      (PT.product (PT.map find_colored children)) in
  let bundle =
    List.fold_right
      (fun (c_wf, _) → CWFBundle.add c_wf)
      fusions fibered_dag.bundle in
  (fusions, bundle)

let colorize_braket1 (wf, (coupling, children)) fibered_dag =
  let find_colored wf' = CWFBundle.inv_pi wf' fibered_dag.bundle in
  Product.fold2
    (fun bra ket acc →
      List.fold_left
        (fun brakets (f, c) →
          if CM.conjugate bra.CA.flavor = f then
            (bra, (colorize_coupling c coupling, ket)) :: brakets
          else
            brakets)
        acc (fuse_c_wf ket))
    (find_colored wf) (PT.product (PT.map find_colored children)) []

module CWFMap =
  Map.Make (struct type t = CA.wf let compare = CA.order_wf end)

module CKetSet =
  Set.Make (struct type t = CA.rhs let compare = compare end)
```

Find a set of kets in *map* that belong to *bra*. Return the empty set, if nothing is found.

```
let lookup_ketset bra map =
  try CWFMap.find bra map with Not_found → CKetSet.empty
```

Return the set of kets belonging to *bra* in *map*, augmented by *ket*.

```
let addto_ketset bra ket map =
  CKetSet.add ket (lookup_ketset bra map)
```

Augment or update *map* with a new (*bra*, *ket*) relation.

```
let addto_ketset_map map (bra, ket) =
  CWFMap.add bra (addto_ketset bra ket map) map
```

Take a list of (*bra*, *ket*) pairs and group the *kets* according to *bra*. This is very similar to *ThoList.factorize* on page 741, but the latter keeps duplicate copies, while we keep only one, with equality determined by *CA.order_wf*.

Isn't *Bundle* M.1 the correct framework for this?

```
let factorize_brakets brakets =
  CWFMap.fold
    (fun bra ket acc → (bra, CKetSet.elements ket) :: acc)
    (List.fold_left addto_ketset_map CWFMap.empty brakets)
    []

let colorize_braket (wf, rhs_list) fibered_dag =
  factorize_brakets
    (ThoList.flatmap
       (fun rhs → (colorize_braket1 (wf, rhs) fibered_dag))
       rhs_list)

let colorize_amplitude a fin fout =
  let f = fin @ List.map CM.conjugate fout in
  let nin, nout = List.length fin, List.length fout in
  let n = nin + nout in
  let externals = List.combine f (ThoList.range 1 n) in
  let external_wfs = CA.external_wfs n externals in
  let wf_bundle = CWFBundle.of_list external_wfs in

  let fibered_dag =
    colorize_dag
      colorize_fusion colorize_external a.A.fusion_dag wf_bundle in

  let brakets =
    ThoList.flatmap
      (fun braket → colorize_braket braket fibered_dag)
      a.A.brakets in

  let dag = CA.D.harvest_list fibered_dag.dag (CA.wavefunctions brakets) in

  let fusions =
    List.filter (function (_, []) → false | _ → true) (CA.D.lists dag) in

  let dependencies_map =
    CA.D.fold
      (fun wf _ → CWFMap.add wf (CA.D.dependencies dag wf))
```

```
        dag CWFMap.empty in
    { CA.fusions = fusions;
      CA.brakets = brakets;
      CA.constraints = a.A.constraints;
      CA.incoming = fin;
      CA.outgoing = fout;
      CA.externals = external_wfs;
      CA.fusion_dag = dag;
      CA.fusion_tower = dag;
      CA.symmetry = a.A.symmetry;
      CA.on_shell = (fun wf → a.A.on_shell (uncolorize_wf wf));
      CA.is_gauss = (fun wf → a.A.is_gauss (uncolorize_wf wf));
      CA.dependencies = (fun wf → CWFMap.find wf dependencies_map) }

let allowed amplitude =
  match amplitude.CA.brakets with
  | [] → false
  | _ → true

let colorize_amplitudes a =
  List.fold_left
    (fun amps (fin, fout) →
      let amp = colorize_amplitude a fin fout in
      if allowed amp then
        amp :: amps
      else
        amps)
    [] (CM.amplitude a.A.incoming a.A.outgoing)

let amplitudes goldstones selectors fin fout =
  colorize_amplitudes (amplitude goldstones selectors fin fout)

type flavor = CA.flavor
type flavor_sans_color = A.flavor
type p = A.p
type wf = CA.wf
let conjugate = CA.conjugate
let flavor = CA.flavor
let flavor_sans_color wf = CM.flavor_sans_color (CA.flavor wf)
let momentum = CA.momentum
let momentum_list = CA.momentum_list
let wf_tag = CA.wf_tag

type coupling = CA.coupling

let sign = CA.sign
let coupling = CA.coupling
let coupling_tag = CA.coupling_tag

type α children = α CA.children
type rhs = CA.rhs
let children = CA.children

type fusion = CA.fusion
```

```
let lhs = CA.lhs
let rhs = CA.rhs

type braket = CA.braket
let bra = CA.bra
let ket = CA.ket

type amplitude = CA.amplitude
let incoming = CA.incoming
let outgoing = CA.outgoing
let externals = CA.externals
let fusions = CA.fusions
let brakets = CA.brakets
let symmetry = CA.symmetry
let on_shell = CA.on_shell
let is_gauss = CA.is_gauss
let constraints = CA.constraints
let variables a = List.map lhs (fusions a)
let dependencies = CA.dependencies
```

Checking Conservation Laws

```
let check_charges () =
  let vlist3, vlist4, vlistn = M.vertices () in
  List.filter
    (fun flist → ¬ (M.Ch.is_null (M.Ch.sum (List.map M.charges flist))))
    (List.map (fun ((f1, f2, f3), _, _) → [f1; f2; f3]) vlist3
     @ List.map (fun ((f1, f2, f3, f4), _, _) → [f1; f2; f3; f4]) vlist4
     @ List.map (fun (flist, _, _) → flist) vlistn)
```

Diagnostics

```
let count_propagators a =
  List.length a.CA.fusions

let count_fusions a =
  List.fold_left (fun n (_, a) → n + List.length a) 0 a.CA.fusions
    + List.fold_left (fun n (_, t) → n + List.length t) 0 a.CA.brakets
    + List.length a.CA.brakets
```

This brute force approach blows up for more than ten particles. Find a smarter algorithm.

```
let count_diagrams a =
  List.fold_left (fun n (wf1, wf23) →
    n + CA.D.count_trees wf1 a.CA.fusion_dag ×
      (List.fold_left (fun n′ (_, wfs) →
        n′ + PT.fold_left (fun n″ wf →
          n″ × CA.D.count_trees wf a.CA.fusion_dag) 1 wfs) 0 wf23))
    0 a.CA.brakets
```

```
exception Impossible

let forest' a =
  let below wf = CA.D.forest_memoized wf a.CA.fusion_dag in
  ThoList.flatmap
    (fun (bra, ket) ->
      (Product.list2 (fun bra' ket' -> bra' :: ket')
         (below bra)
         (ThoList.flatmap
            (fun (_, wfs) ->
              Product.list (fun w -> w) (PT.to_list (PT.map below wfs)))
            ket)))
    a.CA.brakets

let cross wf =
  { CA.flavor = CM.conjugate wf.CA.flavor;
    CA.momentum = P.neg wf.CA.momentum;
    CA.wf_tag = wf.CA.wf_tag }

let fuse_trees wf ts =
  Tree.fuse (fun (wf', e) -> (cross wf', e))
    wf (fun t -> List.mem wf (Tree.leafs t)) ts

let forest wf a =
  List.map (fuse_trees wf) (forest' a)

let poles_beneath wf dag =
  CA.D.eval_memoized (fun wf' -> [[]])
    (fun wf' _ p -> List.map (fun p' -> wf' :: p') p)
    (fun wf1 wf2 ->
      Product.fold2 (fun wf' wfs' wfs'' -> (wf' @ wfs') :: wfs'') wf1 wf2 [])
    (@) [[]] [[]] wf dag

let poles a =
  ThoList.flatmap (fun (wf1, wf23) ->
    let poles_wf1 = poles_beneath wf1 a.CA.fusion_dag in
    (ThoList.flatmap (fun (_, wfs) ->
      Product.list List.flatten
        (PT.to_list (PT.map (fun wf ->
          poles_wf1 @ poles_beneath wf a.CA.fusion_dag) wfs)))
       wf23))
    a.CA.brakets

module WFSet =
  Set.Make (struct type t = CA.wf let compare = CA.order_wf end)

let s_channel a =
  WFSet.elements
    (ThoList.fold_right2
       (fun wf wfs ->
         if P.Scattering.timelike wf.CA.momentum then
           WFSet.add wf wfs
         else
           wfs) (poles a) WFSet.empty)
```

This should be much faster! Is it correct? Is it faster indeed?

```
let poles' a =
  List.map CA.lhs a.CA.fusions

let s_channel a =
  WFSet.elements
    (List.fold_right
       (fun wf wfs ->
         if P.Scattering.timelike wf.CA.momentum then
           WFSet.add wf wfs
         else
           wfs) (poles' a) WFSet.empty)
```

Pictures

Export the DAG in the `dot(1)` file format so that we can draw pretty pictures to impress audiences ...

```
let p2s p =
  if p >= 0 && p <= 9 then
    string_of_int p
  else if p <= 36 then
    String.make 1 (Char.chr (Char.code 'A' + p - 10))
  else
    "_"

let variable wf =
  CM.flavor_symbol wf.CA.flavor ^
  String.concat "" (List.map p2s (P.to_ints wf.CA.momentum))

module Int = Map.Make (struct type t = int let compare = compare end)

let add_to_list i n m =
  Int.add i (n :: try Int.find i m with Not_found -> []) m

let classify_nodes dag =
  Int.fold (fun i n acc -> (i, n) :: acc)
    (CA.D.fold_nodes (fun wf -> add_to_list (P.rank wf.CA.momentum) wf)
       dag Int.empty) []

let dag_to_dot ch brakets dag =
  Printf.fprintf ch "digraph OMEGA {\n";
  CA.D.iter_nodes (fun wf ->
    Printf.fprintf ch "  \"%s\" [ label = \"%s\" ];\n"
      (variable wf) (variable wf)) dag;
  List.iter (fun (_, wfs) ->
    Printf.fprintf ch "  { rank = same;";
    List.iter (fun n ->
      Printf.fprintf ch " \"%s\";" (variable n)) wfs;
    Printf.fprintf ch " };\n") (classify_nodes dag);
  List.iter (fun n ->
    Printf.fprintf ch " \"*\" -> \"%s\";\n" (variable n))
```

```
        (flatten_keystones brakets);
      CA.D.iter (fun n (_, ns) →
        let p = variable n in
        PT.iter (fun n' →
          Printf.fprintf ch "␣␣\"%s\"␣->␣\"%s\";\n" p (variable n')) ns) dag;
      Printf.fprintf ch "}\n"

    let tower_to_dot ch a =
      dag_to_dot ch a.CA.brakets a.CA.fusion_tower

    let amplitude_to_dot ch a =
      dag_to_dot ch a.CA.brakets a.CA.fusion_dag

    let rcs_list = [CA.D.rcs; T.rcs; P.rcs; CM.rcs; rcs]

  end

module Make = Tagged(Order_Tags)

module Binary = Make(Tuple.Binary)(Stat_Dirac)(Topology.Binary)
module Tagged_Binary (T : Tagger) =
  Tagged(T)(Tuple.Binary)(Stat_Dirac)(Topology.Binary)
```

8.2.5 Fusions with Majorana Fermions

```
module Stat_Majorana (M : Model.T) : (Stat with type flavor = M.flavor) =
  struct
    let rcs = RCS.rename rcs_file "Fusion.Stat_Majorana()"
        [ "Fermi␣statistics␣with␣fermion␣number␣violation"]

    type flavor = M.flavor

    type stat =
      | Fermion of int × int list
      | AntiFermion of int × int list
      | Boson of int list
      | Majorana of int × int list

    let stat f p =
      let s = M.fermion f in
      if s = 0 then
        Boson []
      else if s < 0 then
        AntiFermion (p, [])
      else if s = 1 then (* if s = 1 then *)
        Fermion (p, [])
      else (* if s > 1 then *)
        Majorana (p, [])
```

JR sez' (regarding the Majorana Feynman rules): In the formalism of [7], it does not matter to distinguish spinors and conjugate spinors, it is only important to know in which direction a fermion line is calculated. So the sign is made by the calculation together with an aditional one due to the permuation of the pairs of endpoints of fermion lines in the direction they are calculated. We propose a "canonical" direction from the right to the left child at a fusion point so we only have to keep in mind which external particle hangs at each side. Therefore we need not to have a list of pairs of conjugate spinors and spinors but just a list in which the pairs are right-left-right-left and so on. Unfortunately it is unavoidable to have couplings with clashing arrows in supersymmetric theories so we need transmutations from fermions in antifermions and vice versa as well. *(JR's probably right, but I need to check myself ...)*

```
exception Impossible

let stat_fuse s1 s2 f =
  match s1, s2, M.lorentz f with
  | Boson l1, Fermion (p, l2), Coupling.Majorana
  | Boson l1, AntiFermion (p, l2), Coupling.Majorana
  | Fermion (p, l1), Boson l2, Coupling.Majorana
  | AntiFermion (p, l1), Boson l2, Coupling.Majorana
  | Majorana (p, l1), Boson l2, Coupling.Majorana
  | Boson l1, Majorana (p, l2), Coupling.Majorana →
      Majorana (p, l1 @ l2)
  | Boson l1, Fermion (p, l2), Coupling.Spinor
  | Boson l1, AntiFermion (p, l2), Coupling.Spinor
  | Fermion (p, l1), Boson l2, Coupling.Spinor
  | AntiFermion (p, l1), Boson l2, Coupling.Spinor
  | Majorana (p, l1), Boson l2, Coupling.Spinor
  | Boson l1, Majorana (p, l2), Coupling.Spinor →
      Fermion (p, l1 @ l2)
  | Boson l1, Fermion (p, l2), Coupling.ConjSpinor
  | Boson l1, AntiFermion (p, l2), Coupling.ConjSpinor
  | Fermion (p, l1), Boson l2, Coupling.ConjSpinor
  | AntiFermion (p, l1), Boson l2, Coupling.ConjSpinor
  | Majorana (p, l1), Boson l2, Coupling.ConjSpinor
  | Boson l1, Majorana (p, l2), Coupling.ConjSpinor →
      AntiFermion (p, l1 @ l2)
  | Boson l1, Fermion (p, l2), Coupling.Vectorspinor
  | Boson l1, AntiFermion (p, l2), Coupling.Vectorspinor
  | Fermion (p, l1), Boson l2, Coupling.Vectorspinor
  | AntiFermion (p, l1), Boson l2, Coupling.Vectorspinor
  | Majorana (p, l1), Boson l2, Coupling.Vectorspinor
  | Boson l1, Majorana (p, l2), Coupling.Vectorspinor →
      Majorana (p, l1 @ l2)
  | Boson l1, Boson l2, _ → Boson (l1 @ l2)
  | AntiFermion (p1, l1), Fermion (p2, l2), _
  | Fermion (p1, l1), AntiFermion (p2, l2), _
  | Fermion (p1, l1), Fermion (p2, l2), _
  | AntiFermion (p1, l1), AntiFermion (p2, l2), _
  | Fermion (p1, l1), Majorana (p2, l2), _
  | Majorana (p1, l1), Fermion (p2, l2), _
  | AntiFermion (p1, l1), Majorana (p2, l2), _
  | Majorana (p1, l1), AntiFermion (p2, l2), _
```

```
      | Majorana (p1, l1), Majorana (p2, l2), _ →
          Boson ([p2; p1] @ l1 @ l2)
      | Boson l1, Majorana (p, l2), _ → Majorana (p, l1 @ l2)
      | Boson l1, Fermion (p, l2), _ → Fermion (p, l1 @ l2)
      | Boson l1, AntiFermion (p, l2), _ → AntiFermion (p, l1 @ l2)
      | Fermion (p, l1), Boson l2, _ → Fermion (p, l1 @ l2)
      | AntiFermion (p, l1), Boson l2, _ → AntiFermion (p, l1 @ l2)
      | Majorana (p, l1), Boson l2, _ → Majorana (p, l1 @ l2)

    let permutation lines = fst(Combinatorics.sort_signed compare lines)

    let stat_sign = function
      | Boson lines → permutation lines
      | Fermion (p, lines) → permutation (p :: lines)
      | AntiFermion (pbar, lines) → permutation (pbar :: lines)
      | Majorana (pm, lines) → permutation (pm :: lines)

  end

module Binary_Majorana =
  Make(Tuple.Binary)(Stat_Majorana)(Topology.Binary)

module Nary (B : Tuple.Bound) =
  Make(Tuple.Nary(B))(Stat_Dirac)(Topology.Nary(B))
module Nary_Majorana (B : Tuple.Bound) =
  Make(Tuple.Nary(B))(Stat_Majorana)(Topology.Nary(B))

module Mixed23 =
  Make(Tuple.Mixed23)(Stat_Dirac)(Topology.Mixed23)
module Mixed23_Majorana =
  Make(Tuple.Mixed23)(Stat_Majorana)(Topology.Mixed23)

module Helac (B : Tuple.Bound) =
  Make(Tuple.Nary(B))(Stat_Dirac)(Topology.Helac(B))
module Helac_Majorana (B : Tuple.Bound) =
  Make(Tuple.Nary(B))(Stat_Majorana)(Topology.Helac(B))
```

8.2.6 Multiple Amplitudes

```
module type Multi =
  sig
    exception Mismatch
    val options : Options.t
    type flavor
    type process = flavor list × flavor list
    type amplitude
    type fusion
    type wf
    type selectors
    type amplitudes
    val amplitudes : bool → int option → selectors → process list →
amplitudes
    val empty : amplitudes
```

```
    val initialize_cache : string → unit
    val set_cache_name : string → unit
    val flavors : amplitudes → process list
    val vanishing_flavors : amplitudes → process list
    val color_flows : amplitudes → Color.Flow.t list
    val helicities : amplitudes → (int list × int list) list
    val processes : amplitudes → amplitude list
    val process_table : amplitudes → amplitude option array array
    val fusions : amplitudes → (fusion × amplitude) list
    val multiplicity : amplitudes → wf → int
    val dictionary : amplitudes → amplitude → wf → int
    val color_factors : amplitudes → Color.Flow.factor array array
    val constraints : amplitudes → string option
  end

module type Multi_Maker = functor (Fusion_Maker : Maker) →
  functor (P : Momentum.T) →
    functor (M : Model.T) →
      Multi with type flavor = M.flavor
      and type amplitude = Fusion_Maker(P)(M).amplitude
      and type fusion = Fusion_Maker(P)(M).fusion
      and type wf = Fusion_Maker(P)(M).wf
      and type selectors = Fusion_Maker(P)(M).selectors

module Multi (Fusion_Maker : Maker) (P : Momentum.T) (M : Model.T) =
  struct

    exception Mismatch

    type progress_mode =
      | Quiet
      | Channel of out_channel
      | File of string

    let progress_option = ref Quiet

    module CM = Colorize.It(M)
    module F = Fusion_Maker(P)(M)
    module C = Cascade.Make(M)(P)
```

A kludge, at best ...

```
    let options = Options.extend F.options
        [ "progress", Arg.Unit (fun () → progress_option := Channel stderr),
          "report␣progress␣to␣the␣standard␣error␣stream";
          "progress_file", Arg.String (fun s → progress_option := File s),
          "report␣progress␣to␣a␣file" ]

    type flavor = M.flavor
    type p = F.p
    type process = flavor list × flavor list
    type amplitude = F.amplitude
    type fusion = F.fusion
```

```
type wf = F.wf
type selectors = F.selectors

type flavors = flavor list array
type helicities = int list array
type colors = Color.Flow.t array

type amplitudes' = amplitude array array array

type amplitudes =
    { flavors : process list;
      vanishing_flavors : process list;
      color_flows : Color.Flow.t list;
      helicities : (int list × int list) list;
      processes : amplitude list;
      process_table : amplitude option array array;
      fusions : (fusion × amplitude) list;
      multiplicity : (wf → int);
      dictionary : (amplitude → wf → int);
      color_factors : Color.Flow.factor array array;
      constraints : string option }

let flavors a = a.flavors
let vanishing_flavors a = a.vanishing_flavors
let color_flows a = a.color_flows
let helicities a = a.helicities
let processes a = a.processes
let process_table a = a.process_table
let fusions a = a.fusions
let multiplicity a = a.multiplicity
let dictionary a = a.dictionary
let color_factors a = a.color_factors
let constraints a = a.constraints

let sans_colors f =
  List.map CM.flavor_sans_color f

let colors (fin, fout) =
  List.map M.color (fin @ fout)

let process_sans_color a =
  (sans_colors (F.incoming a), sans_colors (F.outgoing a))

let color_flow a =
  CM.flow (F.incoming a) (F.outgoing a)

let process_to_string fin fout =
  String.concat "␣" (List.map M.flavor_to_string fin)
  ^ "␣->␣" ^ String.concat "␣" (List.map M.flavor_to_string fout)

let count_processes colored_processes =
  List.length colored_processes

module FMap =
  Map.Make (struct type t = process let compare = compare end)
```

```
module CMap =
  Map.Make (struct type t = Color.Flow.t let compare = compare end)
```

Recently *Product.list* began to guarantee lexicographic order for sorted arguments. Anyway, we still force a lexicographic order.

```
let rec order_spin_table1 s1 s2 =
  match s1, s2 with
  | h1 :: t1, h2 :: t2 →
      let c = compare h1 h2 in
      if c ≠ 0 then
        c
      else
        order_spin_table1 t1 t2
  | [], [] → 0
  | _ → invalid_arg "order_spin_table: inconsistent lengths"

let order_spin_table (s1_in, s1_out) (s2_in, s2_out) =
  let c = compare s1_in s2_in in
  if c ≠ 0 then
    c
  else
    order_spin_table1 s1_out s2_out

let sort_spin_table table =
  List.sort order_spin_table table

let id x = x

let pair x y = (x, y)
```

Improve support for on shell Ward identities: *Coupling.Vector* → [4] for one and only one external vector.

```
let rec hs_of_lorentz = function
  | Coupling.Scalar → [0]
  | Coupling.Spinor | Coupling.ConjSpinor
  | Coupling.Majorana | Coupling.Maj_Ghost → [−1; 1]
  | Coupling.Vector → [−1; 1]
  | Coupling.Massive_Vector → [−1; 0; 1]
  | Coupling.Tensor_1 → [−1; 0; 1]
  | Coupling.Vectorspinor → [−2; − 1; 1; 2]
  | Coupling.Tensor_2 → [−2; − 1; 0; 1; 2]
  | Coupling.BRS f → hs_of_lorentz f

let hs_of_flavor f =
  hs_of_lorentz (M.lorentz f)

let hs_of_flavors (fin, fout) =
  (List.map hs_of_flavor fin, List.map hs_of_flavor fout)

let rec unphysical_of_lorentz = function
  | Coupling.Vector → [4]
  | Coupling.Massive_Vector → [4]
```

```
  | _ → invalid_arg "unphysical_of_lorentz: not a vector particle"

let unphysical_of_flavor f =
  unphysical_of_lorentz (M.lorentz f)

let unphysical_of_flavors1 n f_list =
  ThoList.mapi
    (fun i f → if i = n then unphysical_of_flavor f else hs_of_flavor f)
    1 f_list

let unphysical_of_flavors n (fin, fout) =
  (unphysical_of_flavors1 n fin, unphysical_of_flavors1 (n − List.length fin) fout)

let helicity_table unphysical flavors =
  let hs =
    begin match unphysical with
    | None → List.map hs_of_flavors flavors
    | Some n → List.map (unphysical_of_flavors n) flavors
    end in
  if ¬ (ThoList.homogeneous hs) then
    invalid_arg "Fusion.helicity_table: not all flavors have the same helicity states!"
  else
    match hs with
    | [] → []
    | (hs_in, hs_out) :: _ →
        sort_spin_table (Product.list2 pair (Product.list id hs_in) (Product.list id hs_out))

module Proc = Process.Make(M)

module WFMap = Map.Make (struct type t = F.wf let compare = compare end)
module WFSet2 =
  Set.Make (struct type t = F.wf × (F.wf, F.coupling) Tree2.t let compare = compare end)
module WFMap2 =
  Map.Make (struct type t = F.wf × (F.wf, F.coupling) Tree2.t let compare = compare end)
module WFTSet =
  Set.Make (struct type t = (F.wf, F.coupling) Tree2.t let compare = compare end)
```

All wavefunctions are unique per amplitude. So we can use per-amplitude dependency trees without additional *internal* tags to identify identical wave functions.

NB: we miss potential optimizations, because we assume all coupling to be different, while in fact we have horizontal/family symmetries and non abelian gauge couplings are universal anyway.

```
let disambiguate_fusions amplitudes =
  let fusions =
    ThoList.flatmap (fun amplitude →
      List.map
        (fun fusion → (fusion, F.dependencies amplitude (F.lhs fusion)))
        (F.fusions amplitude))
      amplitudes in
  let duplicates =
    List.fold_left
      (fun map (fusion, dependencies) →
```

```
          let wf = F.lhs fusion in
          let set = try WFMap.find wf map with Not_found → WFTSet.empty in
          WFMap.add wf (WFTSet.add dependencies set) map)
        WFMap.empty fusions in
    let multiplicity_map =
      WFMap.fold (fun wf dependencies acc →
        let cardinal = WFTSet.cardinal dependencies in
        if cardinal ≤ 1 then
          acc
        else
          WFMap.add wf cardinal acc)
        duplicates WFMap.empty
    and dictionary_map =
      WFMap.fold (fun wf dependencies acc →
        let cardinal = WFTSet.cardinal dependencies in
        if cardinal ≤ 1 then
          acc
        else
          snd (WFTSet.fold
                 (fun dependency (i', acc') →
                   (succ i', WFMap2.add (wf, dependency) i' acc'))
                 dependencies (1, acc)))
        duplicates WFMap2.empty in
    let multiplicity wf =
      WFMap.find wf multiplicity_map
    and dictionary amplitude wf =
      WFMap2.find (wf, F.dependencies amplitude wf) dictionary_map in
    (multiplicity, dictionary)

  let eliminate_common_fusions1 seen_wfs amplitude =
    List.fold_left
      (fun (seen, acc) f →
        let wf = F.lhs f in
        let dependencies = F.dependencies amplitude wf in
        if WFSet2.mem (wf, dependencies) seen then
          (seen, acc)
        else
          (WFSet2.add (wf, dependencies) seen, (f, amplitude) :: acc))
      seen_wfs (F.fusions amplitude)

  let eliminate_common_fusions processes =
    let _, rev_fusions =
      List.fold_left
        eliminate_common_fusions1
        (WFSet2.empty, []) processes in
    List.rev rev_fusions
```

Calculate All The Amplitudes

```
  let amplitudes goldstones unphysical select_wf processes =
```

Eventually, we might want to support inhomogeneous helicities. However, this makes little physics sense for external particles on the mass shell, unless we have a model with degenerate massive fermions and bosons.

```
if ¬ (ThoList.homogeneous (List.map hs_of_flavors processes)) then
  invalid_arg "Fusion.Multi.amplitudes:␣incompatible␣helicities";

let unique_uncolored_processes =
  Proc.remove_duplicate_final_states (C.partition select_wf) processes in

let progress =
  match !progress_option with
  | Quiet → Progress.dummy
  | Channel oc → Progress.channel oc (count_processes unique_uncolored_processes)
  | File name → Progress.file name (count_processes unique_uncolored_processes) in

let allowed =
  ThoList.flatmap
    (fun (fi, fo) →
      Progress.begin_step progress (process_to_string fi fo);
      let amps = F.amplitudes goldstones select_wf fi fo in
      begin match amps with
      | [] → Progress.end_step progress "forbidden"
      | _ → Progress.end_step progress "allowed"
      end;
      amps) unique_uncolored_processes in

Progress.summary progress "all␣processes␣done";

let color_flows =
  ThoList.uniq (List.sort compare (List.map color_flow allowed))
and flavors =
  ThoList.uniq (List.sort compare (List.map process_sans_color allowed)) in

let vanishing_flavors =
  Proc.diff processes flavors in

let helicities =
  helicity_table unphysical flavors in

let f_index =
  fst (List.fold_left
         (fun (m, i) f → (FMap.add f i m, succ i))
         (FMap.empty, 0) flavors)
and c_index =
  fst (List.fold_left
         (fun (m, i) c → (CMap.add c i m, succ i))
         (CMap.empty, 0) color_flows) in

let table =
  Array.make_matrix (List.length flavors) (List.length color_flows) None in
List.iter
  (fun a →
    let f = FMap.find (process_sans_color a) f_index
```

```
        and c = CMap.find (color_flow a) c_index in
        table.(f).(c) ← Some (a))
      allowed;

    let cf_array = Array.of_list color_flows in
    let ncf = Array.length cf_array in
    let color_factor_table = Array.make_matrix ncf ncf Color.Flow.zero in

    for i = 0 to pred ncf do
      for j = 0 to i do
        color_factor_table.(i).(j) ←
          Color.Flow.factor cf_array.(i) cf_array.(j);
        color_factor_table.(j).(i) ←
          color_factor_table.(i).(j)
      done
    done;

    let fusions = eliminate_common_fusions allowed
    and multiplicity, dictionary = disambiguate_fusions allowed in

    { flavors = flavors;
      vanishing_flavors = vanishing_flavors;
      color_flows = color_flows;
      helicities = helicities;
      processes = allowed;
      process_table = table;
      fusions = fusions;
      multiplicity = multiplicity;
      dictionary = dictionary;
      color_factors = color_factor_table;
      constraints = C.description select_wf }

  let initialize_cache = F.initialize_cache
  let set_cache_name = F.set_cache_name

  let empty =
    { flavors = [];
      vanishing_flavors = [];
      color_flows = [];
      helicities = [];
      processes = [];
      process_table = Array.make_matrix 0 0 None;
      fusions = [];
      multiplicity = (fun _ → 1);
      dictionary = (fun _ _ → 1);
      color_factors = Array.make_matrix 0 0 Color.Flow.zero;
      constraints = None }

end
```

—9—
Lorentz Representations, Couplings, Models and Targets

9.1 Interface of Coupling

The enumeration types used for communication from *Models* to *Targets*. On the physics side, the modules in *Models* must implement the Feynman rules according to the conventions set up here. On the numerics side, the modules in *Targets* must handle all cases according to the same conventions.

9.1.1 Propagators

The Lorentz representation of the particle. NB: O'Mega treats all lines as *outgoing* and particles are therefore transforming as *ConjSpinor* and antiparticles as *Spinor*.

```
type lorentz =
  | Scalar
  | Spinor (* ψ *)
  | ConjSpinor (* ψ̄ *)
  | Majorana (* χ *)
  | Maj_Ghost (* SUSY ghosts *)
  | Vector
  | Massive_Vector
  | Vectorspinor (* supersymmetric currents and gravitinos *)
  | Tensor_1
  | Tensor_2 (* massive gravitons (large extra dimensions) *)
  | BRS of lorentz
```

If there were no vectors or auxiliary fields, we could deduce the propagator from the Lorentz representation. While we're at it, we can introduce "propagators" for the contact interactions of auxiliary fields as well. *Prop_Gauge* and *Prop_Feynman* are redundant as special cases of *Prop_Rxi*.

The special case *Only_Insertion* corresponds to operator insertions that do not correspond to a propagating field all. These are used for checking Slavnov-Taylor identities

$$\partial_\mu \langle \text{out}|W^\mu(x)|\text{in}\rangle = m_W \langle \text{out}|\phi(x)|\text{in}\rangle \tag{9.1}$$

	only Dirac fermions	incl. Majorana fermions
Prop_Scalar	$\phi(p) \leftarrow \frac{\mathrm{i}}{p^2 - m^2 + \mathrm{i}m\Gamma}\phi(p)$	
Prop_Spinor	$\psi(p) \leftarrow \frac{\mathrm{i}(-\not p + m)}{p^2 - m^2 + \mathrm{i}m\Gamma}\psi(p)$	$\psi(p) \leftarrow \frac{\mathrm{i}(-\not p + m)}{p^2 - m^2 + \mathrm{i}m\Gamma}\psi(p)$
Prop_ConjSpinor	$\bar\psi(p) \leftarrow \bar\psi(p)\frac{\mathrm{i}(\not p + m)}{p^2 - m^2 + \mathrm{i}m\Gamma}$	$\psi(p) \leftarrow \frac{\mathrm{i}(-\not p + m)}{p^2 - m^2 + \mathrm{i}m\Gamma}\psi(p)$
Prop_Majorana	N/A	$\chi(p) \leftarrow \frac{\mathrm{i}(-\not p + m)}{p^2 - m^2 + \mathrm{i}m\Gamma}\chi(p)$
Prop_Unitarity	$\epsilon_\mu(p) \leftarrow \frac{\mathrm{i}}{p^2 - m^2 + \mathrm{i}m\Gamma}\left(-g_{\mu\nu} + \frac{p_\mu p_\nu}{m^2}\right)\epsilon^\nu(p)$	
Prop_Feynman	$\epsilon^\nu(p) \leftarrow \frac{-\mathrm{i}}{p^2 - m^2 + \mathrm{i}m\Gamma}\epsilon^\nu(p)$	
Prop_Gauge	$\epsilon_\mu(p) \leftarrow \frac{\mathrm{i}}{p^2}\left(-g_{\mu\nu} + (1-\xi)\frac{p_\mu p_\nu}{p^2}\right)\epsilon^\nu(p)$	
Prop_Rxi	$\epsilon_\mu(p) \leftarrow \frac{\mathrm{i}}{p^2 - m^2 + \mathrm{i}m\Gamma}\left(-g_{\mu\nu} + (1-\xi)\frac{p_\mu p_\nu}{p^2 - \xi m^2}\right)\epsilon^\nu(p)$	

Table 9.1: Propagators. NB: The sign of the momenta in the spinor propagators comes about because O'Mega treats all momenta as *outgoing* and the charge flow for *Spinor* is therefore opposite to the momentum, while the charge flow for *ConjSpinor* is parallel to the momentum.

Aux_Scalar	$\phi(p) \leftarrow \mathrm{i}\phi(p)$
Aux_Spinor	$\psi(p) \leftarrow \mathrm{i}\psi(p)$
Aux_ConjSpinor	$\bar\psi(p) \leftarrow \mathrm{i}\bar\psi(p)$
Aux_Vector	$\epsilon^\mu(p) \leftarrow \mathrm{i}\epsilon^\mu(p)$
Aux_Tensor_1	$T^{\mu\nu}(p) \leftarrow \mathrm{i}T^{\mu\nu}(p)$
Only_Insertion	N/A

Table 9.2: Auxiliary and non propagating fields

of gauge theories in unitarity gauge where the Goldstone bosons are not propagating. Numerically, it would suffice to use a vanishing propagator, but then superflous fusions would be calculated in production code in which the Slavnov-Taylor identities are not tested.

```
type α propagator =
  | Prop_Scalar | Prop_Ghost
  | Prop_Spinor | Prop_ConjSpinor | Prop_Majorana
  | Prop_Unitarity | Prop_Feynman | Prop_Gauge of α | Prop_Rxi of α
  | Prop_Tensor_2 | Prop_Vectorspinor
  | Prop_Col_Scalar | Prop_Col_Feynman | Prop_Col_Majorana
  | Prop_Col_Unitarity
  | Aux_Scalar | Aux_Vector | Aux_Tensor_1
  | Aux_Col_Scalar | Aux_Col_Vector | Aux_Col_Tensor_1
  | Aux_Spinor | Aux_ConjSpinor | Aux_Majorana
  | Only_Insertion
```

JR sez' (regarding the Majorana Feynman rules): We don't need different fermionic propagators as supposed by the variable names *Prop_Spinor*, *Prop_ConjSpinor* or *Prop_Majorana*. The propagator in all cases has to be multiplied on the left hand side of the spinor out of which a new one should be built. All momenta are treated as *outgoing*, so for the propagation of the different fermions the following table arises, in which the momentum direction is always downwards and the arrows show whether the momentum and the fermion line, respectively are parallel or antiparallel to the direction of calculation:

Fermion type	fermion arrow	mom.	calc.	sign
Dirac fermion	$\uparrow$	$\uparrow$ $\downarrow$	$\uparrow$ $\uparrow$	negative
Dirac antifermion	$\downarrow$	$\downarrow$ $\downarrow$	$\uparrow$ $\downarrow$	negative
Majorana fermion	-	$\uparrow$ $\downarrow$	-	negative

So the sign of the momentum is always negative and no further distinction is needed. *(JR's probably right, but I need to check myself . . .)*

```
type width =
  | Vanishing
  | Constant
  | Timelike
  | Running
  | Fudged
  | Custom of string
```

9.1.2 Vertices

The combined $S - P$ and $V - A$ couplings (see tables 9.5, 9.6, 9.8 and 9.12) are redundant, of course, but they allow some targets to create more efficient numerical code.[1] Choosing VA2 over VA will cause the FORTRAN backend to pass the coupling as a whole array

[1] An additional benefit is that the counting of Feynman diagrams is not upset by a splitting of the vectorial and axial pieces of gauge bosons.

type *fermion* = *Psi* | *Chi* | *Grav*
type *fermionbar* = *Psibar* | *Chibar* | *Gravbar*
type *boson* =
  | *SP* | *SPM* | *S* | *P* | *SL* | *SR* | *SLR* | *VA* | *V* | *A* | *VL* | *VR* | *VLR* | *VLRM* | *VAM*
  | *TVA* | *TLR* | *TRL* | *TVAM* | *TLRM* | *TRLM*
  | *POT* | *MOM* | *MOM5* | *MOML* | *MOMR* | *LMOM* | *RMOM* | *VMOM* | *VA2* | *VA3* | *VA3M*
type *boson2* = *S2* | *P2* | *S2P* | *S2L* | *S2R* | *S2LR*
  | *SV* | *PV* | *SLV* | *SRV* | *SLRV* | *V2* | *V2LR*

The integer is an additional coefficient that multiplies the respective coupling constant. This allows to reduce the number of required coupling constants in manifestly symmetrc cases. Most of times it will be equal unity, though.
The two vertex types *PBP* and *BBB* for the couplings of two fermions or two antifermions ("clashing arrows") is unavoidable in supersymmetric theories.

... tho doesn't like the names and has promised to find a better mnemonics!

type α *vertex3* =
  | *FBF* of *int* × *fermionbar* × *boson* × *fermion*
  | *PBP* of *int* × *fermion* × *boson* × *fermion*
  | *BBB* of *int* × *fermionbar* × *boson* × *fermionbar*
  | *GBG* of *int* × *fermionbar* × *boson* × *fermion* (∗ gravitino-boson-fermion ∗)
  | *Gauge_Gauge_Gauge* of *int* | *Aux_Gauge_Gauge* of *int*
  | *Scalar_Vector_Vector* of *int*
  | *Aux_Vector_Vector* of *int* | *Aux_Scalar_Vector* of *int*
  | *Scalar_Scalar_Scalar* of *int* | *Aux_Scalar_Scalar* of *int*
  | *Vector_Scalar_Scalar* of *int*
  | *Graviton_Scalar_Scalar* of *int*
  | *Graviton_Vector_Vector* of *int*
  | *Graviton_Spinor_Spinor* of *int*
  | *Dim4_Vector_Vector_Vector_T* of *int*
  | *Dim4_Vector_Vector_Vector_L* of *int*
  | *Dim4_Vector_Vector_Vector_T5* of *int*
  | *Dim4_Vector_Vector_Vector_L5* of *int*
  | *Dim6_Gauge_Gauge_Gauge* of *int*
  | *Dim6_Gauge_Gauge_Gauge_5* of *int*
  | *Aux_DScalar_DScalar* of *int* | *Aux_Vector_DScalar* of *int*
  | *Dim5_Scalar_Gauge2* of *int* (∗ $\frac{1}{2}\phi F_{1,\mu\nu}F_2^{\mu\nu} = -\frac{1}{2}\phi(\mathrm{i}\partial_{[\mu,}V_{1,\nu]})(\mathrm{i}\partial^{[\mu,}V_2^{\nu]})$ ∗)
  | *Dim5_Scalar_Gauge2_Skew* of *int*
    (∗ $\frac{1}{2}\phi F_{1,\mu\nu}\tilde{F}_2^{\mu\nu} = -\phi(\mathrm{i}\partial_\mu V_{1,\nu})(\mathrm{i}\partial_\rho V_{2,\sigma})\epsilon^{\mu\nu\rho\sigma}$ ∗)
  | *Dim5_Scalar_Vector_Vector_T* of *int* (∗ $\phi(\mathrm{i}\partial_\mu V_1^\nu)(\mathrm{i}\partial_\nu V_2^\mu)$ ∗)
  | *Dim5_Scalar_Vector_Vector_TU* of *int* (∗ $(\mathrm{i}\partial_\nu\phi)(\mathrm{i}\partial_\mu V_1^\nu)V_2^\mu$ ∗)
  | *Dim5_Scalar_Vector_Vector_U* of *int* (∗ $(\mathrm{i}\partial_\nu\phi)(\mathrm{i}\partial_\mu V^\nu)V^\mu$ ∗)
  | *Scalar_Vector_Vector_t* of *int* (∗ $(\partial_\mu V_\nu - \partial_\nu V_\mu)^2$ ∗)
  | *Dim6_Vector_Vector_Vector_T* of *int* (∗ $V_1^\mu((\mathrm{i}\partial_\nu V_2^\rho)\mathrm{i}\overleftrightarrow{\partial_\mu}(\mathrm{i}\partial_\rho V_3^\nu))$ ∗)
  | *Tensor_2_Vector_Vector* of *int* (∗ $T^{\mu\nu}(V_{1,\mu}V_{2,\nu} + V_{1,\nu}V_{2,\mu})$ ∗)
  | *Tensor_2_Vector_Vector_1* of *int* (∗ $T^{\mu\nu}(V_{1,\mu}V_{2,\nu} + V_{1,\nu}V_{2,\mu} - g_{\mu,\nu}V_1^\rho V_{2,\rho})$ ∗)

| *Tensor_2_Vector_Vector_t* of *int* (* $T^{\mu\nu}(V_{1,\mu}V_{2,\nu} + V_{1,\nu}V_{2,\mu} - g_{\mu,\nu}V_1^{\rho}V_{2,\rho})$ *)
| *Dim5_Tensor_2_Vector_Vector_1* of *int* (* $T^{\alpha\beta}(V_1^{\mu}\mathrm{i}\overleftrightarrow{\partial}_{\alpha}\mathrm{i}\overleftrightarrow{\partial}_{\beta}V_{2,\mu}$ *)
| *Dim5_Tensor_2_Vector_Vector_2* of *int*
(* $T^{\alpha\beta}(V_1^{\mu}\mathrm{i}\overleftrightarrow{\partial}_{\beta}(\mathrm{i}\partial_{\mu}V_{2,\alpha}) + V_1^{\mu}\mathrm{i}\overleftrightarrow{\partial}_{\alpha}(\mathrm{i}\partial_{\mu}V_{2,\beta}))$ *)
| *Dim7_Tensor_2_Vector_Vector_T* of *int* (* $T^{\alpha\beta}((\mathrm{i}\partial^{\mu}V_1^{\nu})\mathrm{i}\overleftrightarrow{\partial}_{\alpha}\mathrm{i}\overleftrightarrow{\partial}_{\beta}(\mathrm{i}\partial_{\nu}V_{2,\mu}))$ *)

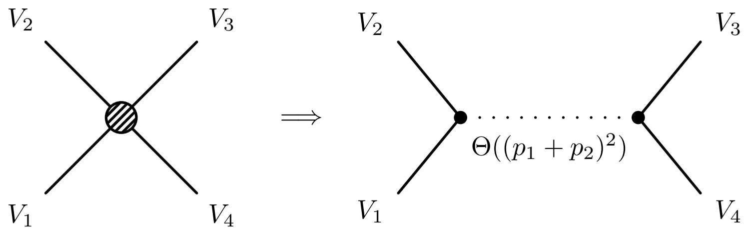
As long as we stick to renormalizable couplings, there are only three types of quartic couplings: *Scalar4*, *Scalar2_Vector2* and *Vector4*. However, there are three inequivalent contractions for the latter and the general vertex will be a linear combination with integer coefficients:

$$\begin{aligned}
\textit{Scalar4}\ 1: &\quad \phi_1\phi_2\phi_3\phi_4 && \text{(9.2a)}\\
\textit{Scalar2_Vector2}\ 1: &\quad \phi_1\phi_2 V_3^{\mu}V_{4,\mu} && \text{(9.2b)}\\
\textit{Vector4}\ [1, C_12_34]: &\quad V_1^{\mu}V_{2,\mu}V_3^{\nu}V_{4,\nu} && \text{(9.2c)}\\
\textit{Vector4}\ [1, C_13_42]: &\quad V_1^{\mu}V_2^{\nu}V_{3,\mu}V_{4,\nu} && \text{(9.2d)}\\
\textit{Vector4}\ [1, C_14_23]: &\quad V_1^{\mu}V_2^{\nu}V_{3,\nu}V_{4,\mu} && \text{(9.2e)}
\end{aligned}$$

type *contract4* = *C_12_34* | *C_13_42* | *C_14_23*

type α *vertex4* =
| *Scalar4* of *int*
| *Scalar2_Vector2* of *int*
| *Vector4* of (*int* × *contract4*) *list*
| *DScalar4* of (*int* × *contract4*) *list*
| *DScalar2_Vector2* of (*int* × *contract4*) *list*
| *GBBG* of *int* × *fermionbar* × *boson2* × *fermion*

In some applications, we have to allow for contributions outside of perturbation theory. The most prominent example is heavy gauge boson scattering at very high energies, where the perturbative expression violates unitarity.
One solution is the 'K-matrix' ansatz. Such unitarizations typically introduce effective propagators and/or vertices that violate crossing symmetry and vanish in the t-channel. This can be taken care of in *Fusion* by filtering out vertices that have the wrong momenta.
In this case the ordering of the fields in a vertex of the Feynman rules becomes significant. In particular, we assume that (V_1, V_2, V_3, V_4) implies



(9.3)

The list of pairs of parameters denotes the location and strengths of the poles in the K-matrix ansatz:

$$(c_1, a_1, c_2, a_2, \ldots, c_n, a_n) \Longrightarrow f(s) = \sum_{i=1}^{n} \frac{c_i}{s - a_i} \tag{9.4}$$

| $Vector4\_K\_Matrix\_tho$ of $int \times (\alpha \times \alpha)$ $list$
| $Vector4\_K\_Matrix\_jr$ of $int \times (int \times contract4)$ $list$

type α $vertexn$ = $unit$

An obvious candidate for addition to *boson* is T, of course.

This list is sufficient for the minimal standard model, but not comprehensive enough for most of its extensions, supersymmetric or otherwise. In particular, we need a *general* parameterization for all trilinear vertices. One straightforward possibility are polynomials in the momenta for each combination of fields.

JR sez' (regarding the Majorana Feynman rules): Here we use the rules which can be found in [7] and are more properly described in *Targets* where the performing of the fusion rules in analytical expressions is encoded. *(JR's probably right, but I need to check myself ...)*

Signify which two of three fields are fused:

type $fuse2$ = $F23$ | $F32$ | $F31$ | $F13$ | $F12$ | $F21$

Signify which three of four fields are fused:

type $fuse3$ =
| $F123$ | $F231$ | $F312$ | $F132$ | $F321$ | $F213$
| $F124$ | $F241$ | $F412$ | $F142$ | $F421$ | $F214$
| $F134$ | $F341$ | $F413$ | $F143$ | $F431$ | $F314$
| $F234$ | $F342$ | $F423$ | $F243$ | $F432$ | $F324$

Explicit enumeration types make no sense for higher degrees.

type $fusen$ = int $list$

The third member of the triplet will contain the coupling constant:

type α t =
| $V3$ of α $vertex3 \times fuse2 \times \alpha$
| $V4$ of α $vertex4 \times fuse3 \times \alpha$
| Vn of α $vertexn \times fusen \times \alpha$

9.1.3 Gauge Couplings

Dimension-4 trilinear vector boson couplings

$$\begin{aligned} f_{abc}\partial^\mu A^{a,\nu} A^b_\mu A^c_\nu &\to \mathrm{i} f_{abc} k_1^\mu A^{a,\nu}(k_1) A^b_\mu(k_2) A^c_\nu(k_3) \\ &= -\frac{\mathrm{i}}{3!} f_{a_1 a_2 a_3} C^{\mu_1\mu_2\mu_3}(k_1,k_2,k_3) A^{a_1}_{\mu_1}(k_1) A^{a_2}_{\mu_2}(k_2) A^{a_3}_{\mu_3}(k_3) \end{aligned} \tag{9.5a}$$

with the totally antisymmetric tensor (under simultaneous permutations of all quantum numbers μ_i and k_i) and all momenta *outgoing*

$$C^{\mu_1\mu_2\mu_3}(k_1,k_2,k_3) = (g^{\mu_1\mu_2}(k_1^{\mu_3} - k_2^{\mu_3}) + g^{\mu_2\mu_3}(k_2^{\mu_1} - k_3^{\mu_1}) + g^{\mu_3\mu_1}(k_3^{\mu_2} - k_1^{\mu_2})) \tag{9.5b}$$

	only Dirac fermions	incl. Majorana fermions
FBF (*Psibar*, *S*, *Psi*): $\mathcal{L}_I = g_S\bar\psi_1 S\psi_2$		
F12	$\bar\psi_2 \leftarrow \mathrm{i}\cdot g_S\bar\psi_1 S$	$\psi_2 \leftarrow \mathrm{i}\cdot g_S\psi_1 S$
F21	$\bar\psi_2 \leftarrow \mathrm{i}\cdot g_S S\bar\psi_1$	$\psi_2 \leftarrow \mathrm{i}\cdot g_S S\psi_1$
F13	$S \leftarrow \mathrm{i}\cdot g_S\bar\psi_1\psi_2$	$S \leftarrow \mathrm{i}\cdot g_S\psi_1^T \mathrm{C}\psi_2$
F31	$S \leftarrow \mathrm{i}\cdot g_S\psi_{2,\alpha}\bar\psi_{1,\alpha}$	$S \leftarrow \mathrm{i}\cdot g_S\psi_2^T \mathrm{C}\psi_1$
F23	$\psi_1 \leftarrow \mathrm{i}\cdot g_S S\psi_2$	$\psi_1 \leftarrow \mathrm{i}\cdot g_S S\psi_2$
F32	$\psi_1 \leftarrow \mathrm{i}\cdot g_S\psi_2 S$	$\psi_1 \leftarrow \mathrm{i}\cdot g_S\psi_2 S$
FBF (*Psibar*, *P*, *Psi*): $\mathcal{L}_I = g_P\bar\psi_1 P\gamma_5\psi_2$		
F12	$\bar\psi_2 \leftarrow \mathrm{i}\cdot g_P\bar\psi_1\gamma_5 P$	$\psi_2 \leftarrow \mathrm{i}\cdot g_P\gamma_5\psi_1 P$
F21	$\bar\psi_2 \leftarrow \mathrm{i}\cdot g_P P\bar\psi_1\gamma_5$	$\psi_2 \leftarrow \mathrm{i}\cdot g_P P\gamma_5\psi_1$
F13	$P \leftarrow \mathrm{i}\cdot g_P\bar\psi_1\gamma_5\psi_2$	$P \leftarrow \mathrm{i}\cdot g_P\psi_1^T \mathrm{C}\gamma_5\psi_2$
F31	$P \leftarrow \mathrm{i}\cdot g_P[\gamma_5\psi_2]_\alpha\bar\psi_{1,\alpha}$	$P \leftarrow \mathrm{i}\cdot g_P\psi_2^T \mathrm{C}\gamma_5\psi_1$
F23	$\psi_1 \leftarrow \mathrm{i}\cdot g_P P\gamma_5\psi_2$	$\psi_1 \leftarrow \mathrm{i}\cdot g_P P\gamma_5\psi_2$
F32	$\psi_1 \leftarrow \mathrm{i}\cdot g_P\gamma_5\psi_2 P$	$\psi_1 \leftarrow \mathrm{i}\cdot g_P\gamma_5\psi_2 P$
FBF (*Psibar*, *V*, *Psi*): $\mathcal{L}_I = g_V\bar\psi_1 \not{V}\psi_2$		
F12	$\bar\psi_2 \leftarrow \mathrm{i}\cdot g_V\bar\psi_1\not{V}$	$\psi_{2,\alpha} \leftarrow \mathrm{i}\cdot(-g_V)\psi_{1,\beta}\not{V}_{\alpha\beta}$
F21	$\bar\psi_{2,\beta} \leftarrow \mathrm{i}\cdot g_V\not{V}_{\alpha\beta}\bar\psi_{1,\alpha}$	$\psi_2 \leftarrow \mathrm{i}\cdot(-g_V)\not{V}\psi_1$
F13	$V_\mu \leftarrow \mathrm{i}\cdot g_V\bar\psi_1\gamma_\mu\psi_2$	$V_\mu \leftarrow \mathrm{i}\cdot g_V(\psi_1)^T \mathrm{C}\gamma_\mu\psi_2$
F31	$V_\mu \leftarrow \mathrm{i}\cdot g_V[\gamma_\mu\psi_2]_\alpha\bar\psi_{1,\alpha}$	$V_\mu \leftarrow \mathrm{i}\cdot(-g_V)(\psi_2)^T \mathrm{C}\gamma_\mu\psi_1$
F23	$\psi_1 \leftarrow \mathrm{i}\cdot g_V\not{V}\psi_2$	$\psi_1 \leftarrow \mathrm{i}\cdot g_V\not{V}\psi_2$
F32	$\psi_{1,\alpha} \leftarrow \mathrm{i}\cdot g_V\psi_{2,\beta}\not{V}_{\alpha\beta}$	$\psi_{1,\alpha} \leftarrow \mathrm{i}\cdot g_V\psi_{2,\beta}\not{V}_{\alpha\beta}$
FBF (*Psibar*, *A*, *Psi*): $\mathcal{L}_I = g_A\bar\psi_1\gamma_5\not{A}\psi_2$		
F12	$\bar\psi_2 \leftarrow \mathrm{i}\cdot g_A\bar\psi_1\gamma_5\not{A}$	$\psi_{2,\alpha} \leftarrow \mathrm{i}\cdot g_A\psi_\beta[\gamma_5\not{A}]_{\alpha\beta}$
F21	$\bar\psi_{2,\beta} \leftarrow \mathrm{i}\cdot g_A[\gamma_5\not{A}]_{\alpha\beta}\bar\psi_{1,\alpha}$	$\psi_2 \leftarrow \mathrm{i}\cdot g_A\gamma_5\not{A}\psi$
F13	$A_\mu \leftarrow \mathrm{i}\cdot g_A\bar\psi_1\gamma_5\gamma_\mu\psi_2$	$A_\mu \leftarrow \mathrm{i}\cdot g_A\psi_1^T \mathrm{C}\gamma_5\gamma_\mu\psi_2$
F31	$A_\mu \leftarrow \mathrm{i}\cdot g_A[\gamma_5\gamma_\mu\psi_2]_\alpha\bar\psi_{1,\alpha}$	$A_\mu \leftarrow \mathrm{i}\cdot g_A\psi_2^T \mathrm{C}\gamma_5\gamma_\mu\psi_1$
F23	$\psi_1 \leftarrow \mathrm{i}\cdot g_A\gamma_5\not{A}\psi_2$	$\psi_1 \leftarrow \mathrm{i}\cdot g_A\gamma_5\not{A}\psi_2$
F32	$\psi_{1,\alpha} \leftarrow \mathrm{i}\cdot g_A\psi_{2,\beta}[\gamma_5\not{A}]_{\alpha\beta}$	$\psi_{1,\alpha} \leftarrow \mathrm{i}\cdot g_A\psi_{2,\beta}[\gamma_5\not{A}]_{\alpha\beta}$

Table 9.3: Dimension-4 trilinear fermionic couplings. The momenta are unambiguous, because there are no derivative couplings and all participating fields are different.

	only Dirac fermions	incl. Majorana fermions
FBF (*Psibar*, *T*, *Psi*): $\mathcal{L}_I = g_T T_{\mu\nu} \bar\psi_1 [\gamma^\mu, \gamma^\nu]_- \psi_2$		
F12	$\bar\psi_2 \leftarrow \mathrm{i} \cdot g_T \bar\psi_1 [\gamma^\mu, \gamma^\nu]_- T_{\mu\nu}$	$\bar\psi_2 \leftarrow \mathrm{i} \cdot g_T \cdots$
F21	$\bar\psi_2 \leftarrow \mathrm{i} \cdot g_T T_{\mu\nu} \bar\psi_1 [\gamma^\mu, \gamma^\nu]_-$	$\bar\psi_2 \leftarrow \mathrm{i} \cdot g_T \cdots$
F13	$T_{\mu\nu} \leftarrow \mathrm{i} \cdot g_T \bar\psi_1 [\gamma_\mu, \gamma_\nu]_- \psi_2$	$T_{\mu\nu} \leftarrow \mathrm{i} \cdot g_T \cdots$
F31	$T_{\mu\nu} \leftarrow \mathrm{i} \cdot g_T [[\gamma_\mu, \gamma_\nu]_- \psi_2]_\alpha \bar\psi_{1,\alpha}$	$T_{\mu\nu} \leftarrow \mathrm{i} \cdot g_T \cdots$
F23	$\psi_1 \leftarrow \mathrm{i} \cdot g_T T_{\mu\nu} [\gamma^\mu, \gamma^\nu]_- \psi_2$	$\psi_1 \leftarrow \mathrm{i} \cdot g_T \cdots$
F32	$\psi_1 \leftarrow \mathrm{i} \cdot g_T [\gamma^\mu, \gamma^\nu]_- \psi_2 T_{\mu\nu}$	$\psi_1 \leftarrow \mathrm{i} \cdot g_T \cdots$

Table 9.4: Dimension-5 trilinear fermionic couplings (NB: the coefficients and signs are not fixed yet). The momenta are unambiguous, because there are no derivative couplings and all participating fields are different.

Since $f_{a_1a_2a_3} C^{\mu_1\mu_2\mu_3}(k_1, k_2, k_3)$ is totally symmetric (under simultaneous permutations of all quantum numbers a_i, μ_i and k_i), it is easy to take the partial derivative

$$A^{a,\mu}(k_2 + k_3) = -\frac{\mathrm{i}}{2!} f_{abc} C^{\mu\rho\sigma}(-k_2 - k_3, k_2, k_3) A^b_\rho(k_2) A^c_\sigma(k_3) \tag{9.6a}$$

with

$$C^{\mu\rho\sigma}(-k_2 - k_3, k_2, k_3) = (g^{\rho\sigma}(k_2^\mu - k_3^\mu) + g^{\mu\sigma}(2k_3^\rho + k_2^\rho) - g^{\mu\rho}(2k_2^\sigma + k_3^\sigma)) \tag{9.6b}$$

i. e.

$$\begin{aligned} A^{a,\mu}(k_2 + k_3) = -\frac{\mathrm{i}}{2!} f_{abc} \big(& (k_2^\mu - k_3^\mu) A^b(k_2) \cdot A^c(k_3) \\ & + (2k_3 + k_2) \cdot A^b(k_2) A^{c,\mu}(k_3) - A^{b,\mu}(k_2) A^c(k_3) \cdot (2k_2 + k_3) \big) \end{aligned} \tag{9.6c}$$

Investigate the rearrangements proposed in [5] for improved numerical stability.

Non-Gauge Vector Couplings

As a basis for the dimension-4 couplings of three vector bosons, we choose “transversal” and “longitudinal” (with respect to the first vector field) tensors that are odd and even under permutation of the second and third argument

$$\mathcal{L}_T(V_1, V_2, V_3) = V_1^\mu (V_{2,\nu} \mathrm{i} \overleftrightarrow{\partial_\mu} V_3^\nu) = -\mathcal{L}_T(V_1, V_3, V_2) \tag{9.7a}$$

$$\mathcal{L}_L(V_1, V_2, V_3) = (\mathrm{i}\partial_\mu V_1^\mu) V_{2,\nu} V_3^\nu = \mathcal{L}_L(V_1, V_3, V_2) \tag{9.7b}$$

Using partial integration in $\mathcal{L}_L$, we find the convenient combinations

$$\mathcal{L}_T(V_1, V_2, V_3) + \mathcal{L}_L(V_1, V_2, V_3) = -2V_1^\mu \mathrm{i}\partial_\mu V_{2,\nu} V_3^\nu \tag{9.8a}$$

$$\mathcal{L}_T(V_1, V_2, V_3) - \mathcal{L}_L(V_1, V_2, V_3) = 2V_1^\mu V_{2,\nu} \mathrm{i}\partial_\mu V_3^\nu \tag{9.8b}$$

	only Dirac fermions	incl. Majorana fermions
FBF $(Psibar,\ SP,\ Psi)$: $\mathcal{L}_I = \bar\psi_1\phi(g_S + g_P\gamma_5)\psi_2$		
F12	$\bar\psi_2 \leftarrow \mathrm{i}\cdot\bar\psi_1(g_S + g_P\gamma_5)\phi$	$\psi_2 \leftarrow \mathrm{i}\cdot\cdots$
F21	$\bar\psi_2 \leftarrow \mathrm{i}\cdot\phi\bar\psi_1(g_S + g_P\gamma_5)$	$\psi_2 \leftarrow \mathrm{i}\cdot\cdots$
F13	$\phi \leftarrow \mathrm{i}\cdot\bar\psi_1(g_S + g_P\gamma_5)\psi_2$	$\phi \leftarrow \mathrm{i}\cdot\cdots$
F31	$\phi \leftarrow \mathrm{i}\cdot[(g_S + g_P\gamma_5)\psi_2]_\alpha\bar\psi_{1,\alpha}$	$\phi \leftarrow \mathrm{i}\cdot\cdots$
F23	$\psi_1 \leftarrow \mathrm{i}\cdot\phi(g_S + g_P\gamma_5)\psi_2$	$\psi_1 \leftarrow \mathrm{i}\cdot\cdots$
F32	$\psi_1 \leftarrow \mathrm{i}\cdot(g_S + g_P\gamma_5)\psi_2\phi$	$\psi_1 \leftarrow \mathrm{i}\cdot\cdots$
FBF $(Psibar,\ SL,\ Psi)$: $\mathcal{L}_I = g_L\bar\psi_1\phi(1-\gamma_5)\psi_2$		
F12	$\bar\psi_2 \leftarrow \mathrm{i}\cdot g_L\bar\psi_1(1-\gamma_5)\phi$	$\psi_2 \leftarrow \mathrm{i}\cdot\cdots$
F21	$\bar\psi_2 \leftarrow \mathrm{i}\cdot g_L\phi\bar\psi_1(1-\gamma_5)$	$\psi_2 \leftarrow \mathrm{i}\cdot\cdots$
F13	$\phi \leftarrow \mathrm{i}\cdot g_L\bar\psi_1(1-\gamma_5)\psi_2$	$\phi \leftarrow \mathrm{i}\cdot\cdots$
F31	$\phi \leftarrow \mathrm{i}\cdot g_L[(1-\gamma_5)\psi_2]_\alpha\bar\psi_{1,\alpha}$	$\phi \leftarrow \mathrm{i}\cdot\cdots$
F23	$\psi_1 \leftarrow \mathrm{i}\cdot g_L\phi(1-\gamma_5)\psi_2$	$\psi_1 \leftarrow \mathrm{i}\cdot\cdots$
F32	$\psi_1 \leftarrow \mathrm{i}\cdot g_L(1-\gamma_5)\psi_2\phi$	$\psi_1 \leftarrow \mathrm{i}\cdot\cdots$
FBF $(Psibar,\ SR,\ Psi)$: $\mathcal{L}_I = g_R\bar\psi_1\phi(1+\gamma_5)\psi_2$		
F12	$\bar\psi_2 \leftarrow \mathrm{i}\cdot g_R\bar\psi_1(1+\gamma_5)\phi$	$\psi_2 \leftarrow \mathrm{i}\cdot\cdots$
F21	$\bar\psi_2 \leftarrow \mathrm{i}\cdot g_R\phi\bar\psi_1(1+\gamma_5)$	$\psi_2 \leftarrow \mathrm{i}\cdot\cdots$
F13	$\phi \leftarrow \mathrm{i}\cdot g_R\bar\psi_1(1+\gamma_5)\psi_2$	$\phi \leftarrow \mathrm{i}\cdot\cdots$
F31	$\phi \leftarrow \mathrm{i}\cdot g_R[(1+\gamma_5)\psi_2]_\alpha\bar\psi_{1,\alpha}$	$\phi \leftarrow \mathrm{i}\cdot\cdots$
F23	$\psi_1 \leftarrow \mathrm{i}\cdot g_R\phi(1+\gamma_5)\psi_2$	$\psi_1 \leftarrow \mathrm{i}\cdot\cdots$
F32	$\psi_1 \leftarrow \mathrm{i}\cdot g_R(1+\gamma_5)\psi_2\phi$	$\psi_1 \leftarrow \mathrm{i}\cdot\cdots$
FBF $(Psibar,\ SLR,\ Psi)$: $\mathcal{L}_I = g_L\bar\psi_1\phi(1-\gamma_5)\psi_2 + g_R\bar\psi_1\phi(1+\gamma_5)\psi_2$		

Table 9.5: Combined dimension-4 trilinear fermionic couplings.

	only Dirac fermions	incl. Majorana fermions
FBF (*Psibar*, *VA*, *Psi*): $\mathcal{L}_I = \bar\psi_1 \not Z (g_V - g_A\gamma_5)\psi_2$		
F12	$\bar\psi_2 \leftarrow \mathrm{i}\cdot\bar\psi_1 \not Z (g_V - g_A\gamma_5)$	$\psi_2 \leftarrow \mathrm{i}\cdot\cdots$
F21	$\bar\psi_{2,\beta} \leftarrow \mathrm{i}\cdot[\not Z (g_V - g_A\gamma_5)]_{\alpha\beta}\bar\psi_{1,\alpha}$	$\psi_2 \leftarrow \mathrm{i}\cdot\cdots$
F13	$Z_\mu \leftarrow \mathrm{i}\cdot\bar\psi_1\gamma_\mu(g_V - g_A\gamma_5)\psi_2$	$Z_\mu \leftarrow \mathrm{i}\cdot\cdots$
F31	$Z_\mu \leftarrow \mathrm{i}\cdot[\gamma_\mu(g_V - g_A\gamma_5)\psi_2]_\alpha\bar\psi_{1,\alpha}$	$Z_\mu \leftarrow \mathrm{i}\cdot\cdots$
F23	$\psi_1 \leftarrow \mathrm{i}\cdot\not Z (g_V - g_A\gamma_5)\psi_2$	$\psi_1 \leftarrow \mathrm{i}\cdot\cdots$
F32	$\psi_{1,\alpha} \leftarrow \mathrm{i}\cdot\psi_{2,\beta}[\not Z (g_V - g_A\gamma_5)]_{\alpha\beta}$	$\psi_1 \leftarrow \mathrm{i}\cdot\cdots$
FBF (*Psibar*, *VL*, *Psi*): $\mathcal{L}_I = g_L\bar\psi_1 \not Z (1-\gamma_5)\psi_2$		
F12	$\bar\psi_2 \leftarrow \mathrm{i}\cdot g_L\bar\psi_1 \not Z (1-\gamma_5)$	$\psi_2 \leftarrow \mathrm{i}\cdot\cdots$
F21	$\bar\psi_{2,\beta} \leftarrow \mathrm{i}\cdot g_L[\not Z (1-\gamma_5)]_{\alpha\beta}\bar\psi_{1,\alpha}$	$\psi_2 \leftarrow \mathrm{i}\cdot\cdots$
F13	$Z_\mu \leftarrow \mathrm{i}\cdot g_L\bar\psi_1\gamma_\mu(1-\gamma_5)\psi_2$	$Z_\mu \leftarrow \mathrm{i}\cdot\cdots$
F31	$Z_\mu \leftarrow \mathrm{i}\cdot g_L[\gamma_\mu(1-\gamma_5)\psi_2]_\alpha\bar\psi_{1,\alpha}$	$Z_\mu \leftarrow \mathrm{i}\cdot\cdots$
F23	$\psi_1 \leftarrow \mathrm{i}\cdot g_L\not Z (1-\gamma_5)\psi_2$	$\psi_1 \leftarrow \mathrm{i}\cdot\cdots$
F32	$\psi_{1,\alpha} \leftarrow \mathrm{i}\cdot g_L\psi_{2,\beta}[\not Z (1-\gamma_5)]_{\alpha\beta}$	$\psi_1 \leftarrow \mathrm{i}\cdot\cdots$
FBF (*Psibar*, *VR*, *Psi*): $\mathcal{L}_I = g_R\bar\psi_1 \not Z (1+\gamma_5)\psi_2$		
F12	$\bar\psi_2 \leftarrow \mathrm{i}\cdot g_R\bar\psi_1 \not Z (1+\gamma_5)$	$\psi_2 \leftarrow \mathrm{i}\cdot\cdots$
F21	$\bar\psi_{2,\beta} \leftarrow \mathrm{i}\cdot g_R[\not Z (1+\gamma_5)]_{\alpha\beta}\bar\psi_{1,\alpha}$	$\psi_2 \leftarrow \mathrm{i}\cdot\cdots$
F13	$Z_\mu \leftarrow \mathrm{i}\cdot g_R\bar\psi_1\gamma_\mu(1+\gamma_5)\psi_2$	$Z_\mu \leftarrow \mathrm{i}\cdot\cdots$
F31	$Z_\mu \leftarrow \mathrm{i}\cdot g_R[\gamma_\mu(1+\gamma_5)\psi_2]_\alpha\bar\psi_{1,\alpha}$	$Z_\mu \leftarrow \mathrm{i}\cdot\cdots$
F23	$\psi_1 \leftarrow \mathrm{i}\cdot g_R\not Z (1+\gamma_5)\psi_2$	$\psi_1 \leftarrow \mathrm{i}\cdot\cdots$
F32	$\psi_{1,\alpha} \leftarrow \mathrm{i}\cdot g_R\psi_{2,\beta}[\not Z (1+\gamma_5)]_{\alpha\beta}$	$\psi_1 \leftarrow \mathrm{i}\cdot\cdots$
FBF (*Psibar*, *VLR*, *Psi*): $\mathcal{L}_I = g_L\bar\psi_1 \not Z (1-\gamma_5)\psi_2 + g_R\bar\psi_1 \not Z (1+\gamma_5)\psi_2$		

Table 9.6: Combined dimension-4 trilinear fermionic couplings continued.

$FBF\ (Psibar,\ S,\ Chi)$: $\bar\psi S\chi$	
$F12$: $\chi \leftarrow \psi S$	$F21$: $\chi \leftarrow S\psi$
$F13$: $S \leftarrow \psi^T \mathrm{C}\chi$	$F31$: $S \leftarrow \chi^T \mathrm{C}\psi$
$F23$: $\psi \leftarrow S\chi$	$F32$: $\psi \leftarrow \chi S$
$FBF\ (Psibar,\ P,\ Chi)$: $\bar\psi P\gamma_5\chi$	
$F12$: $\chi \leftarrow \gamma_5\psi P$	$F21$: $\chi \leftarrow P\gamma_5\psi$
$F13$: $P \leftarrow \psi^T \mathrm{C}\gamma_5\chi$	$F31$: $P \leftarrow \chi^T \mathrm{C}\gamma_5\psi$
$F23$: $\psi \leftarrow P\gamma_5\chi$	$F32$: $\psi \leftarrow \gamma_5\chi P$
$FBF\ (Psibar,\ V,\ Chi)$: $\bar\psi \not{V}\chi$	
$F12$: $\chi_\alpha \leftarrow -\psi_\beta \not{V}_{\alpha\beta}$	$F21$: $\chi \leftarrow -\not{V}\psi$
$F13$: $V_\mu \leftarrow \psi^T \mathrm{C}\gamma_\mu\chi$	$F31$: $V_\mu \leftarrow \chi^T \mathrm{C}(-\gamma_\mu\psi)$
$F23$: $\psi \leftarrow \not{V}\chi$	$F32$: $\psi_\alpha \leftarrow \chi_\beta \not{V}_{\alpha\beta}$
$FBF\ (Psibar,\ A,\ Chi)$: $\bar\psi\gamma^5 \not{A}\chi$	
$F12$: $\chi_\alpha \leftarrow \psi_\beta[\gamma^5\not{A}]_{\alpha\beta}$	$F21$: $\chi \leftarrow \gamma^5\not{A}\psi$
$F13$: $A_\mu \leftarrow \psi^T \mathrm{C}\gamma^5\gamma_\mu\chi$	$F31$: $A_\mu \leftarrow \chi^T \mathrm{C}(\gamma^5\gamma_\mu\psi)$
$F23$: $\psi \leftarrow \gamma^5\not{A}\chi$	$F32$: $\psi_\alpha \leftarrow \chi_\beta[\gamma^5\not{A}]_{\alpha\beta}$

Table 9.7: Dimension-4 trilinear couplings including one Dirac and one Majorana fermion

$FBF\ (Psibar,\ SP,\ Chi)$: $\bar\psi\phi(g_S + g_P\gamma_5)\chi$	
$F12$: $\chi \leftarrow (g_S + g_P\gamma_5)\psi\phi$	$F21$: $\chi \leftarrow \phi(g_S + g_P\gamma_5)\psi$
$F13$: $\phi \leftarrow \psi^T \mathrm{C}(g_S + g_P\gamma_5)\chi$	$F31$: $\phi \leftarrow \chi^T \mathrm{C}(g_S + g_P\gamma_5)\chi$
$F23$: $\psi \leftarrow \phi(g_S + g_P\gamma_5)\chi$	$F32$: $\psi \leftarrow (g_S + g_P\gamma_5)\chi\phi$
$FBF\ (Psibar,\ VA,\ Chi)$: $\bar\psi \not{Z}(g_V - g_A\gamma_5)\chi$	
$F12$: $\chi_\alpha \leftarrow \psi_\beta[\not{Z}(-g_V - g_A\gamma_5)]_{\alpha\beta}$	$F21$: $\chi \leftarrow \not{Z}(-g_V - g_A\gamma_5)]\psi$
$F13$: $Z_\mu \leftarrow \psi^T \mathrm{C}\gamma_\mu(g_V - g_A\gamma_5)\chi$	$F31$: $Z_\mu \leftarrow \chi^T \mathrm{C}\gamma_\mu(-g_V - g_A\gamma_5)\psi$
$F23$: $\psi \leftarrow \not{Z}(g_V - g_A\gamma_5)\chi$	$F32$: $\psi_\alpha \leftarrow \chi_\beta[\not{Z}(g_V - g_A\gamma_5)]_{\alpha\beta}$

Table 9.8: Combined dimension-4 trilinear fermionic couplings including one Dirac and one Majorana fermion.

FBF (*Chibar*, *S*, *Psi*): $\bar{\chi}S\psi$	
F12: $\psi \leftarrow \chi S$	*F21*: $\psi \leftarrow S\chi$
F13: $S \leftarrow \chi^T \mathrm{C}\psi$	*F31*: $S \leftarrow \psi^T \mathrm{C}\chi$
F23: $\chi \leftarrow S\psi$	*F32*: $\chi \leftarrow \psi S$
FBF (*Chibar*, *P*, *Psi*): $\bar{\chi}P\gamma_5\psi$	
F12: $\psi \leftarrow \gamma_5\chi P$	*F21*: $\psi \leftarrow P\gamma_5\chi$
F13: $P \leftarrow \chi^T \mathrm{C}\gamma_5\psi$	*F31*: $P \leftarrow \psi^T \mathrm{C}\gamma_5\chi$
F23: $\chi \leftarrow P\gamma_5\psi$	*F32*: $\chi \leftarrow \gamma_5\psi P$
FBF (*Chibar*, *V*, *Psi*): $\bar{\chi}\not{V}\psi$	
F12: $\psi_\alpha \leftarrow -\chi_\beta \not{V}_{\alpha\beta}$	*F21*: $\psi \leftarrow -\not{V}\chi$
F13: $V_\mu \leftarrow \chi^T \mathrm{C}\gamma_\mu\psi$	*F31*: $V_\mu \leftarrow \psi^T \mathrm{C}(-\gamma_\mu\chi)$
F23: $\chi \leftarrow \not{V}\psi$	*F32*: $\chi_\alpha \leftarrow \psi_\beta \not{V}_{\alpha\beta}$
FBF (*Chibar*, *A*, *Psi*): $\bar{\chi}\gamma^5\not{A}\psi$	
F12: $\psi_\alpha \leftarrow \chi_\beta[\gamma^5\not{A}]_{\alpha\beta}$	*F21*: $\psi \leftarrow \gamma^5\not{A}\chi$
F13: $A_\mu \leftarrow \chi^T \mathrm{C}(\gamma^5\gamma_\mu\psi)$	*F31*: $A_\mu \leftarrow \psi^T \mathrm{C}\gamma^5\gamma_\mu\chi$
F23: $\chi \leftarrow \gamma^5\not{A}\psi$	*F32*: $\chi_\alpha \leftarrow \psi_\beta[\gamma^5\not{A}]_{\alpha\beta}$

Table 9.9: Dimension-4 trilinear couplings including one Dirac and one Majorana fermion

FBF (*Chibar*, *SP*, *Psi*): $\bar{\chi}\phi(g_S + g_P\gamma_5)\psi$	
F12: $\psi \leftarrow (g_S + g_P\gamma_5)\chi\phi$	*F21*: $\psi \leftarrow \phi(g_S + g_P\gamma_5)\chi$
F13: $\phi \leftarrow \chi^T \mathrm{C}(g_S + g_P\gamma_5)\psi$	*F31*: $\phi \leftarrow \psi^T \mathrm{C}(g_S + g_P\gamma_5)\chi$
F23: $\chi \leftarrow \phi(g_S + g_P\gamma_5)\psi$	*F32*: $\chi \leftarrow (g_S + g_P\gamma_5)\psi\phi$
FBF (*Chibar*, *VA*, *Psi*): $\bar{\chi}\not{Z}(g_V - g_A\gamma_5)\psi$	
F12: $\psi_\alpha \leftarrow \chi_\beta[\not{Z}(-g_V - g_A\gamma_5)]_{\alpha\beta}$	*F21*: $\psi \leftarrow \not{Z}(-g_V - g_A\gamma_5)\chi$
F13: $Z_\mu \leftarrow \chi^T \mathrm{C}\gamma_\mu(g_V - g_A\gamma_5)\psi$	*F31*: $Z_\mu \leftarrow \psi^T \mathrm{C}\gamma_\mu(-g_V - g_A\gamma_5)\chi$
F23: $\chi \leftarrow \not{Z}(g_V - g_A\gamma_5)]\psi$	*F32*: $\chi_\alpha \leftarrow \psi_\beta[\not{Z}(g_V - g_A\gamma_5)]_{\alpha\beta}$

Table 9.10: Combined dimension-4 trilinear fermionic couplings including one Dirac and one Majorana fermion.

FBF $(Chibar,\ S,\ Chi)$: $\bar{\chi}_a S \chi_b$	
F12: $\chi_b \leftarrow \chi_a S$	*F21*: $\chi_b \leftarrow S\chi_a$
F13: $S \leftarrow \chi_a^T \mathrm{C}\chi_b$	*F31*: $S \leftarrow \chi_b^T \mathrm{C}\chi_a$
F23: $\chi_a \leftarrow S\chi_b$	*F32*: $\chi_a \leftarrow \chi S_b$
FBF $(Chibar,\ P,\ Chi)$: $\bar{\chi}_a P\gamma_5\psi_b$	
F12: $\chi_b \leftarrow \gamma_5\chi_a P$	*F21*: $\chi_b \leftarrow P\gamma_5\chi_a$
F13: $P \leftarrow \chi_a^T \mathrm{C}\gamma_5\chi_b$	*F31*: $P \leftarrow \chi_b^T \mathrm{C}\gamma_5\chi_a$
F23: $\chi_a \leftarrow P\gamma_5\chi_b$	*F32*: $\chi_a \leftarrow \gamma_5\chi_b P$
FBF $(Chibar,\ V,\ Chi)$: $\bar{\chi}_a \not{V}\chi_b$	
F12: $\chi_{b,\alpha} \leftarrow -\chi_{a,\beta}\not{V}_{\alpha\beta}$	*F21*: $\chi_b \leftarrow -\not{V}\chi_a$
F13: $V_\mu \leftarrow \chi_a^T \mathrm{C}\gamma_\mu\chi_b$	*F31*: $V_\mu \leftarrow -\chi_b^T \mathrm{C}\gamma_\mu\chi_a$
F23: $\chi_a \leftarrow \not{V}\chi_b$	*F32*: $\chi_{a,\alpha} \leftarrow \chi_{b,\beta}\not{V}_{\alpha\beta}$
FBF $(Chibar,\ A,\ Chi)$: $\bar{\chi}_a\gamma^5\not{A}\chi_b$	
F12: $\chi_{b,\alpha} \leftarrow \chi_{a,\beta}[\gamma^5\not{A}]_{\alpha\beta}$	*F21*: $\chi_b \leftarrow \gamma^5\not{A}\chi_a$
F13: $A_\mu \leftarrow \chi_a^T \mathrm{C}\gamma^5\gamma_\mu\chi_b$	*F31*: $A_\mu \leftarrow \chi_b^T \mathrm{C}(\gamma^5\gamma_\mu\chi_a)$
F23: $\chi_a \leftarrow \gamma^5\not{A}\chi_b$	*F32*: $\chi_{a,\alpha} \leftarrow \chi_{b,\beta}[\gamma^5\not{A}]_{\alpha\beta}$

Table 9.11: Dimension-4 trilinear couplings of two Majorana fermions

FBF $(Chibar,\ SP,\ Chi)$: $\bar{\chi}\phi_a(g_S + g_P\gamma_5)\chi_b$	
F12: $\chi_b \leftarrow (g_S + g_P\gamma_5)\chi_a\phi$	*F21*: $\chi_b \leftarrow \phi(g_S + g_P\gamma_5)\chi_a$
F13: $\phi \leftarrow \chi_a^T \mathrm{C}(g_S + g_P\gamma_5)\chi_b$	*F31*: $\phi \leftarrow \chi_b^T \mathrm{C}(g_S + g_P\gamma_5)\chi_a$
F23: $\chi_a \leftarrow \phi(g_S + g_P\gamma_5)\chi_b$	*F32*: $\chi_a \leftarrow (g_S + g_P\gamma_5)\chi_b\phi$
FBF $(Chibar,\ VA,\ Chi)$: $\bar{\chi}_a\not{Z}(g_V - g_A\gamma_5)\chi_b$	
F12: $\chi_{b,\alpha} \leftarrow \chi_{a,\beta}[\not{Z}(-g_V - g_A\gamma_5)]_{\alpha\beta}$	*F21*: $\chi_b \leftarrow \not{Z}(-g_V - g_A\gamma_5)]\chi_a$
F13: $Z_\mu \leftarrow \chi_a^T \mathrm{C}\gamma_\mu(g_V - g_A\gamma_5)\chi_b$	*F31*: $Z_\mu \leftarrow \chi_b^T \mathrm{C}\gamma_\mu(-g_V - g_A\gamma_5)\chi_a$
F23: $\chi_a \leftarrow \not{Z}(g_V - g_A\gamma_5)\chi_b$	*F32*: $\chi_{a,\alpha} \leftarrow \chi_{b,\beta}[\not{Z}(g_V - g_A\gamma_5)]_{\alpha\beta}$

Table 9.12: Combined dimension-4 trilinear fermionic couplings of two Majorana fermions.

Gauge_Gauge_Gauge: $\mathcal{L}_I = gf_{abc}A_a^\mu A_b^\nu \partial_\mu A_{c,\nu}$
_: $A_a^\mu \leftarrow \mathrm{i}\cdot(-\mathrm{i}g/2)\cdot C_{abc}^{\mu\rho\sigma}(-k_2-k_3,k_2,k_3)A_\rho^b A_\sigma^c$
Aux_Gauge_Gauge: $\mathcal{L}_I = gf_{abc}X_{a,\mu\nu}(k_1)(A_b^\mu(k_2)A_c^\nu(k_3) - A_b^\nu(k_2)A_c^\mu(k_3))$
F23∨*F32*: $X_a^{\mu\nu}(k_2+k_3) \leftarrow \mathrm{i}\cdot gf_{abc}(A_b^\mu(k_2)A_c^\nu(k_3) - A_b^\nu(k_2)A_c^\mu(k_3))$
F12∨*F13*: $A_{a,\mu}(k_1+k_{2/3}) \leftarrow \mathrm{i}\cdot gf_{abc}X_{b,\nu\mu}(k_1)A_c^\nu(k_{2/3})$
F21∨*F31*: $A_{a,\mu}(k_{2/3}+k_1) \leftarrow \mathrm{i}\cdot gf_{abc}A_b^\nu(k_{2/3})X_{c,\mu\nu}(k_1)$

Table 9.13: Dimension-4 Vector Boson couplings with *outgoing* momenta. See (11.1b) and (9.6b) for the definition of the antisymmetric tensor $C^{\mu_1\mu_2\mu_3}(k_1,k_2,k_3)$.

Scalar_Vector_Vector: $\mathcal{L}_I = g\phi V_1^\mu V_{2,\mu}$	
F13: $\leftarrow \mathrm{i}\cdot g\cdots$	*F31*: $\leftarrow \mathrm{i}\cdot g\cdots$
F12: $\leftarrow \mathrm{i}\cdot g\cdots$	*F21*: $\leftarrow \mathrm{i}\cdot g\cdots$
F23: $\phi \leftarrow \mathrm{i}\cdot gV_1^\mu V_{2,\mu}$	*F32*: $\phi \leftarrow \mathrm{i}\cdot gV_{2,\mu}V_1^\mu$
Aux_Vector_Vector: $\mathcal{L}_I = gXV_1^\mu V_{2,\mu}$	
F13: $\leftarrow \mathrm{i}\cdot g\cdots$	*F31*: $\leftarrow \mathrm{i}\cdot g\cdots$
F12: $\leftarrow \mathrm{i}\cdot g\cdots$	*F21*: $\leftarrow \mathrm{i}\cdot g\cdots$
F23: $X \leftarrow \mathrm{i}\cdot gV_1^\mu V_{2,\mu}$	*F32*: $X \leftarrow \mathrm{i}\cdot gV_{2,\mu}V_1^\mu$
Aux_Scalar_Vector: $\mathcal{L}_I = gX^\mu\phi V_\mu$	
F13: $\leftarrow \mathrm{i}\cdot g\cdots$	*F31*: $\leftarrow \mathrm{i}\cdot g\cdots$
F12: $\leftarrow \mathrm{i}\cdot g\cdots$	*F21*: $\leftarrow \mathrm{i}\cdot g\cdots$
F23: $\leftarrow \mathrm{i}\cdot g\cdots$	*F32*: $\leftarrow \mathrm{i}\cdot g\cdots$

Table 9.14: ...

Scalar_Scalar_Scalar: $\mathcal{L}_I = g\phi_1\phi_2\phi_3$	
F13: $\phi_2 \leftarrow \mathrm{i}\cdot g\phi_1\phi_3$	*F31*: $\phi_2 \leftarrow \mathrm{i}\cdot g\phi_3\phi_1$
F12: $\phi_3 \leftarrow \mathrm{i}\cdot g\phi_1\phi_2$	*F21*: $\phi_3 \leftarrow \mathrm{i}\cdot g\phi_2\phi_1$
F23: $\phi_1 \leftarrow \mathrm{i}\cdot g\phi_2\phi_3$	*F32*: $\phi_1 \leftarrow \mathrm{i}\cdot g\phi_3\phi_2$
Aux_Scalar_Scalar: $\mathcal{L}_I = gX\phi_1\phi_2$	
F13: $\leftarrow \mathrm{i}\cdot g\cdots$	*F31*: $\leftarrow \mathrm{i}\cdot g\cdots$
F12: $\leftarrow \mathrm{i}\cdot g\cdots$	*F21*: $\leftarrow \mathrm{i}\cdot g\cdots$
F23: $X \leftarrow \mathrm{i}\cdot g\phi_1\phi_2$	*F32*: $X \leftarrow \mathrm{i}\cdot g\phi_2\phi_1$

Table 9.15: ...

Vector_Scalar_Scalar: $\mathcal{L}_I = gV^\mu\phi_1 \mathrm{i}\overleftrightarrow{\partial_\mu}\phi_2$	
F23:	$V^\mu(k_2+k_3) \leftarrow \mathrm{i}\cdot g(k_2^\mu - k_3^\mu)\phi_1(k_2)\phi_2(k_3)$
F32:	$V^\mu(k_2+k_3) \leftarrow \mathrm{i}\cdot g(k_2^\mu - k_3^\mu)\phi_2(k_3)\phi_1(k_2)$
F12:	$\phi_2(k_1+k_2) \leftarrow \mathrm{i}\cdot g(k_1^\mu + 2k_2^\mu)V_\mu(k_1)\phi_1(k_2)$
F21:	$\phi_2(k_1+k_2) \leftarrow \mathrm{i}\cdot g(k_1^\mu + 2k_2^\mu)\phi_1(k_2)V_\mu(k_1)$
F13:	$\phi_1(k_1+k_3) \leftarrow \mathrm{i}\cdot g(-k_1^\mu - 2k_3^\mu)V_\mu(k_1)\phi_2(k_3)$
F31:	$\phi_1(k_1+k_3) \leftarrow \mathrm{i}\cdot g(-k_1^\mu - 2k_3^\mu)\phi_2(k_3)V_\mu(k_1)$

Table 9.16: ...

Aux_DScalar_DScalar: $\mathcal{L}_I = g\chi(\mathrm{i}\partial_\mu\phi_1)(\mathrm{i}\partial^\mu\phi_2)$	
F23:	$\chi(k_2+k_3) \leftarrow \mathrm{i}\cdot g(k_2\cdot k_3)\phi_1(k_2)\phi_2(k_3)$
F32:	$\chi(k_2+k_3) \leftarrow \mathrm{i}\cdot g(k_3\cdot k_2)\phi_2(k_3)\phi_1(k_2)$
F12:	$\phi_2(k_1+k_2) \leftarrow \mathrm{i}\cdot g((-k_1-k_2)\cdot k_2)\chi(k_1)\phi_1(k_2)$
F21:	$\phi_2(k_1+k_2) \leftarrow \mathrm{i}\cdot g(k_2\cdot(-k_1-k_2))\phi_1(k_2)\chi(k_1)$
F13:	$\phi_1(k_1+k_3) \leftarrow \mathrm{i}\cdot g((-k_1-k_3)\cdot k_3)\chi(k_1)\phi_2(k_3)$
F31:	$\phi_1(k_1+k_3) \leftarrow \mathrm{i}\cdot g(k_3\cdot(-k_1-k_3))\phi_2(k_3)\chi(k_1)$

Table 9.17: ...

Aux_Vector_DScalar: $\mathcal{L}_I = g\chi V_\mu(\mathrm{i}\partial^\mu\phi)$	
F23:	$\chi(k_2+k_3) \leftarrow \mathrm{i}\cdot gk_3^\mu V_\mu(k_2)\phi(k_3)$
F32:	$\chi(k_2+k_3) \leftarrow \mathrm{i}\cdot g\phi(k_3)k_3^\mu V_\mu(k_2)$
F12:	$\phi(k_1+k_2) \leftarrow \mathrm{i}\cdot g\chi(k_1)(-k_1-k_2)^\mu V_\mu(k_2)$
F21:	$\phi(k_1+k_2) \leftarrow \mathrm{i}\cdot g(-k_1-k_2)^\mu V_\mu(k_2)\chi(k_1)$
F13:	$V_\mu(k_1+k_3) \leftarrow \mathrm{i}\cdot g(-k_1-k_3)_\mu\chi(k_1)\phi(k_3)$
F31:	$V_\mu(k_1+k_3) \leftarrow \mathrm{i}\cdot g(-k_1-k_3)_\mu\phi(k_3)\chi(k_1)$

Table 9.18: ...

As an important example, we can rewrite the dimension-4 “anomalous” triple gauge couplings

$$\begin{aligned}\mathrm{i}\mathcal{L}_{\mathrm{TGC}}(g_1,\kappa,g_4)/g_{VWW} = {} & g_1 V^\mu (W^-_{\mu\nu}W^{+,\nu} - W^+_{\mu\nu}W^{-,\nu}) \\ & + \kappa W^+_\mu W^-_\nu V^{\mu\nu} + g_4 W^+_\mu W^-_\nu (\partial^\mu V^\nu + \partial^\nu V^\mu)\end{aligned} \tag{9.9}$$

as

$$\begin{aligned}\mathcal{L}_{\mathrm{TGC}}(g_1,\kappa,g_4) = {} & g_1 \mathcal{L}_T(V,W^-,W^+) \\ & - \frac{\kappa+g_1-g_4}{2}\mathcal{L}_T(W^-,V,W^+) + \frac{\kappa+g_1+g_4}{2}\mathcal{L}_T(W^+,V,W^-) \\ & - \frac{\kappa-g_1-g_4}{2}\mathcal{L}_L(W^-,V,W^+) + \frac{\kappa-g_1+g_4}{2}\mathcal{L}_L(W^+,V,W^-)\end{aligned} \tag{9.10}$$

CP Violation

$$\mathcal{L}_{\tilde{T}}(V_1,V_2,V_3) = V_{1,\mu}(V_{2,\rho}\mathrm{i}\overleftrightarrow{\partial_\nu}V_{3,\sigma})\epsilon^{\mu\nu\rho\sigma} = +\mathcal{L}_T(V_1,V_3,V_2) \tag{9.11a}$$
$$\mathcal{L}_{\tilde{L}}(V_1,V_2,V_3) = (\mathrm{i}\partial_\mu V_{1,\nu})V_{2,\rho}V_{3,\sigma}\epsilon^{\mu\nu\rho\sigma} = -\mathcal{L}_L(V_1,V_3,V_2) \tag{9.11b}$$

Here the notations $\tilde{T}$ and $\tilde{L}$ are clearly *abuse de langage*, because $\mathcal{L}_{\tilde{L}}(V_1,V_2,V_3)$ is actually the transversal combination, due to the antisymmetry of ϵ. Using partial integration in $\mathcal{L}_{\tilde{L}}$, we could again find combinations

$$\mathcal{L}_{\tilde{T}}(V_1,V_2,V_3) + \mathcal{L}_{\tilde{L}}(V_1,V_2,V_3) = -2V_{1,\mu}V_{2,\nu}\mathrm{i}\partial_\rho V_{3,\sigma}\epsilon^{\mu\nu\rho\sigma} \tag{9.12a}$$
$$\mathcal{L}_{\tilde{T}}(V_1,V_2,V_3) - \mathcal{L}_{\tilde{L}}(V_1,V_2,V_3) = -2V_{1,\mu}\mathrm{i}\partial_\nu V_{2,\rho}V_{3,\sigma}\epsilon^{\mu\nu\rho\sigma} \tag{9.12b}$$

but we don’t need them, since

$$\begin{aligned}\mathrm{i}\mathcal{L}_{\mathrm{TGC}}(g_5,\tilde{\kappa})/g_{VWW} = {} & g_5\epsilon_{\mu\nu\rho\sigma}(W^{+,\mu}\mathrm{i}\overleftrightarrow{\partial^\rho}W^{-,\nu})V^\sigma \\ & - \frac{\tilde{\kappa}_V}{2}W^-_\mu W^+_\nu \epsilon^{\mu\nu\rho\sigma}V_{\rho\sigma}\end{aligned} \tag{9.13}$$

is immediately recognizable as

$$\mathcal{L}_{\mathrm{TGC}}(g_5,\tilde{\kappa})/g_{VWW} = -\mathrm{i}g_5\mathcal{L}_{\tilde{L}}(V,W^-,W^+) + \tilde{\kappa}\mathcal{L}_{\tilde{T}}(V,W^-,W^+) \tag{9.14}$$

9.1.4 SU(2) Gauge Bosons

An important special case for table 9.13 are the two usual coordinates of SU(2)

$$W_\pm = \frac{1}{\sqrt{2}}(W_1 \mp \mathrm{i}W_2) \tag{9.15}$$

i. e.

$$W_1 = \frac{1}{\sqrt{2}}(W_+ + W_-) \tag{9.16a}$$
$$W_2 = \frac{\mathrm{i}}{\sqrt{2}}(W_+ - W_-) \tag{9.16b}$$

Dim4_Vector_Vector_Vector_T: $\mathcal{L}_I = gV_1^\mu V_{2,\nu}\mathrm{i}\overleftrightarrow{\partial_\mu}V_3^\nu$	
F23:	$V_1^\mu(k_2+k_3) \leftarrow \mathrm{i}\cdot g(k_2^\mu - k_3^\mu)V_{2,\nu}(k_2)V_3^\nu(k_3)$
F32:	$V_1^\mu(k_2+k_3) \leftarrow \mathrm{i}\cdot g(k_2^\mu - k_3^\mu)V_3^\nu(k_3)V_{2,\nu}(k_2)$
F12:	$V_3^\mu(k_1+k_2) \leftarrow \mathrm{i}\cdot g(2k_2^\nu + k_1^\nu)V_{1,\nu}(k_1)V_2^\mu(k_2)$
F21:	$V_3^\mu(k_1+k_2) \leftarrow \mathrm{i}\cdot g(2k_2^\nu + k_1^\nu)V_2^\mu(k_2)V_{1,\nu}(k_1)$
F13:	$V_2^\mu(k_1+k_3) \leftarrow \mathrm{i}\cdot g(-k_1^\nu - 2k_3^\nu)V_1^\nu(k_1)V_3^\mu(k_3)$
F31:	$V_2^\mu(k_1+k_3) \leftarrow \mathrm{i}\cdot g(-k_1^\nu - 2k_3^\nu)V_3^\mu(k_3)V_1^\nu(k_1)$
Dim4_Vector_Vector_Vector_L: $\mathcal{L}_I = g\mathrm{i}\partial_\mu V_1^\mu V_{2,\nu}V_3^\nu$	
F23:	$V_1^\mu(k_2+k_3) \leftarrow \mathrm{i}\cdot g(k_2^\mu + k_3^\mu)V_{2,\nu}(k_2)V_3^\nu(k_3)$
F32:	$V_1^\mu(k_2+k_3) \leftarrow \mathrm{i}\cdot g(k_2^\mu + k_3^\mu)V_3^\nu(k_3)V_{2,\nu}(k_2)$
F12:	$V_3^\mu(k_1+k_2) \leftarrow \mathrm{i}\cdot g(-k_1^\nu)V_{1,\nu}(k_1)V_2^\mu(k_2)$
F21:	$V_3^\mu(k_1+k_2) \leftarrow \mathrm{i}\cdot g(-k_1^\nu)V_2^\mu(k_2)V_{1,\nu}(k_1)$
F13:	$V_2^\mu(k_1+k_3) \leftarrow \mathrm{i}\cdot g(-k_1^\nu)V_1^\nu(k_1)V_3^\mu(k_3)$
F31:	$V_2^\mu(k_1+k_3) \leftarrow \mathrm{i}\cdot g(-k_1^\nu)V_3^\mu(k_3)V_1^\nu(k_1)$

Table 9.19: ...

Dim4_Vector_Vector_Vector_T5: $\mathcal{L}_I = gV_{1,\mu}V_{2,\rho}\mathrm{i}\overleftrightarrow{\partial_\nu}V_{3,\sigma}\epsilon^{\mu\nu\rho\sigma}$	
F23:	$V_1^\mu(k_2+k_3) \leftarrow \mathrm{i}\cdot g\epsilon^{\mu\nu\rho\sigma}(k_{2,\nu} - k_{3,\nu})V_{2,\rho}(k_2)V_{3,\sigma}(k_3)$
F32:	$V_1^\mu(k_2+k_3) \leftarrow \mathrm{i}\cdot g\epsilon^{\mu\nu\rho\sigma}(k_{2,\nu} - k_{3,\nu})V_{3,\sigma}(k_3)V_{2,\rho}(k_2)$
F12:	$V_3^\mu(k_1+k_2) \leftarrow \mathrm{i}\cdot g\epsilon^{\mu\nu\rho\sigma}(2k_{2,\nu} + k_{1,\nu})V_{1,\rho}(k_1)V_{2,\sigma}(k_2)$
F21:	$V_3^\mu(k_1+k_2) \leftarrow \mathrm{i}\cdot g\epsilon^{\mu\nu\rho\sigma}(2k_{2,\nu} + k_{1,\nu})V_{2,\sigma}(k_2)V_{1,\rho}(k_1)$
F13:	$V_2^\mu(k_1+k_3) \leftarrow \mathrm{i}\cdot g\epsilon^{\mu\nu\rho\sigma}(-k_{1,\nu} - 2k_{3,\nu})V_{1,\rho}(k_1)V_{3,\sigma}(k_3)$
F31:	$V_2^\mu(k_1+k_3) \leftarrow \mathrm{i}\cdot g\epsilon^{\mu\nu\rho\sigma}(-k_{1,\nu} - 2k_{3,\nu})V_{3,\sigma}(k_3)V_{1,\rho}(k_1)$
Dim4_Vector_Vector_Vector_L5: $\mathcal{L}_I = g\mathrm{i}\partial_\mu V_{1,\nu}V_{2,\nu}V_{3,\sigma}\epsilon^{\mu\nu\rho\sigma}$	
F23:	$V_1^\mu(k_2+k_3) \leftarrow \mathrm{i}\cdot g\epsilon^{\mu\nu\rho\sigma}(k_{2,\nu} + k_{3,\nu})V_{2,\rho}(k_2)V_{3,\sigma}(k_3)$
F32:	$V_1^\mu(k_2+k_3) \leftarrow \mathrm{i}\cdot g\epsilon^{\mu\nu\rho\sigma}(k_{2,\nu} + k_{3,\nu})V_{2,\rho}(k_2)V_{3,\sigma}(k_3)$
F12:	$V_3^\mu(k_1+k_2) \leftarrow \mathrm{i}\cdot g\epsilon^{\mu\nu\rho\sigma}(-k_{1,\nu})V_{1,\rho}(k_1)V_{2,\sigma}(k_2)$
F21:	$V_3^\mu(k_1+k_2) \leftarrow \mathrm{i}\cdot g\epsilon^{\mu\nu\rho\sigma}(-k_{1,\nu})V_{2,\sigma}(k_2)V_{1,\rho}(k_1)$
F13:	$V_2^\mu(k_1+k_3) \leftarrow \mathrm{i}\cdot g\epsilon^{\mu\nu\rho\sigma}(-k_{1,\nu})V_{1,\rho}(k_1)V_{3,\sigma}(k_3)$
F31:	$V_2^\mu(k_1+k_3) \leftarrow \mathrm{i}\cdot g\epsilon^{\mu\nu\rho\sigma}(-k_{1,\nu})V_{3,\sigma}(k_3)V_{1,\rho}(k_1)$

Table 9.20: ...

Dim6_Gauge_Gauge_Gauge: $\mathcal{L}_I = gF_1^{\mu\nu}F_{2,\nu\rho}F_{3,\ \mu}^{\ \rho}$
_: $A_1^\mu(k_2+k_3) \leftarrow -\mathrm{i}\cdot\Lambda^{\mu\rho\sigma}(-k_2-k_3,k_2,k_3)A_{2,\rho}A_{c,\sigma}$

Table 9.21: ...

Dim6_Gauge_Gauge_Gauge_5: $\mathcal{L}_I = g/2\cdot\epsilon^{\mu\nu\lambda\tau}F_{1,\mu\nu}F_{2,\tau\rho}F_{3,\ \lambda}^{\ \rho}$
F23: $A_1^\mu(k_2+k_3) \leftarrow -\mathrm{i}\cdot\Lambda_5^{\mu\rho\sigma}(-k_2-k_3,k_2,k_3)A_{2,\rho}A_{3,\sigma}$
F32: $A_1^\mu(k_2+k_3) \leftarrow -\mathrm{i}\cdot\Lambda_5^{\mu\rho\sigma}(-k_2-k_3,k_2,k_3)A_{3,\sigma}A_{2,\rho}$
F12: $A_3^\mu(k_1+k_2) \leftarrow -\mathrm{i}\cdot$
F21: $A_3^\mu(k_1+k_2) \leftarrow -\mathrm{i}\cdot$
F13: $A_2^\mu(k_1+k_3) \leftarrow -\mathrm{i}\cdot$
F31: $A_2^\mu(k_1+k_3) \leftarrow -\mathrm{i}\cdot$

Table 9.22: ...

and

$$W_1^\mu W_2^\nu - W_2^\mu W_1^\nu = \mathrm{i}\left(W_-^\mu W_+^\nu - W_+^\mu W_-^\nu\right) \tag{9.17}$$

Thus the symmtry remains after the change of basis:

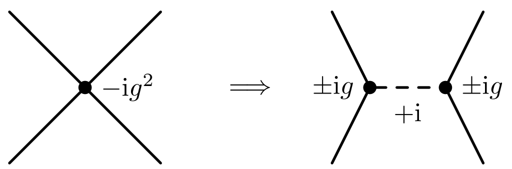
$$\begin{aligned}\epsilon^{abc}W_a^{\mu_1}W_b^{\mu_2}W_c^{\mu_3} &= \mathrm{i}W_-^{\mu_1}(W_+^{\mu_2}W_3^{\mu_3} - W_3^{\mu_2}W_+^{\mu_3}) \\ &\quad + \mathrm{i}W_+^{\mu_1}(W_3^{\mu_2}W_-^{\mu_3} - W_-^{\mu_2}W_3^{\mu_3}) + \mathrm{i}W_3^{\mu_1}(W_-^{\mu_2}W_+^{\mu_3} - W_+^{\mu_2}W_-^{\mu_3})\end{aligned} \tag{9.18}$$

9.1.5 Quartic Couplings and Auxiliary Fields

Quartic couplings can be replaced by cubic couplings to a non-propagating auxiliary field. The quartic term should get a negative sign so that it the energy is bounded from below for identical fields. In the language of functional integrals

$$\begin{aligned}\mathcal{L}_{\phi^4} &= -g^2\phi_1\phi_2\phi_3\phi_4 \Longrightarrow \\ \mathcal{L}_{X\phi^2} &= X^*X \pm gX\phi_1\phi_2 \pm gX^*\phi_3\phi_4 = (X^*\pm g\phi_1\phi_2)(X\pm g\phi_3\phi_4) - g^2\phi_1\phi_2\phi_3\phi_4\end{aligned} \tag{9.19a}$$

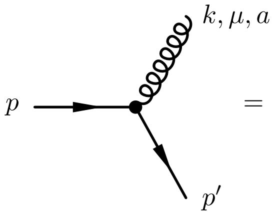
and in the language of Feynman diagrams



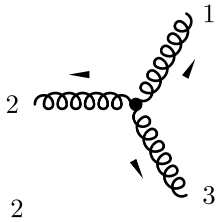
(9.19b)

The other choice of signs

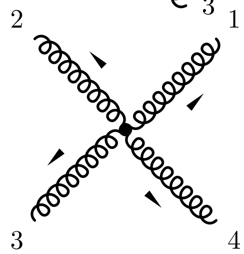
$$\mathcal{L}'_{X\phi^2} = -X^*X \pm gX\phi_1\phi_2 \mp gX^*\phi_3\phi_4 = -(X^*\pm g\phi_1\phi_2)(X\mp g\phi_3\phi_4) - g^2\phi_1\phi_2\phi_3\phi_4 \tag{9.20}$$



$$= \qquad +\mathrm{i}g\gamma_\mu T_a \tag{9.22a}$$



$$= \qquad g f_{a_1 a_2 a_3} C^{\mu_1\mu_2\mu_3}(k_1, k_2, k_3) \tag{9.22b}$$



$$= \begin{array}{l} -\mathrm{i}g^2 f_{a_1 a_2 b} f_{a_3 a_4 b}(g_{\mu_1\mu_3} g_{\mu_4\mu_2} - g_{\mu_1\mu_4} g_{\mu_2\mu_3}) \\ -\mathrm{i}g^2 f_{a_1 a_3 b} f_{a_4 a_2 b}(g_{\mu_1\mu_4} g_{\mu_2\mu_3} - g_{\mu_1\mu_2} g_{\mu_3\mu_4}) \\ -\mathrm{i}g^2 f_{a_1 a_4 b} f_{a_2 a_3 b}(g_{\mu_1\mu_2} g_{\mu_3\mu_4} - g_{\mu_1\mu_3} g_{\mu_4\mu_2}) \end{array} \tag{9.22c}$$

Figure 9.1: Gauge couplings. See (11.1b) for the definition of the antisymmetric tensor $C^{\mu_1\mu_2\mu_3}(k_1, k_2, k_3)$.

can not be extended easily to identical particles and is therefore not used. For identical particles we have

$$\mathcal{L}_{\phi^4} = -\frac{g^2}{4!}\phi^4 \Longrightarrow$$
$$\mathcal{L}_{X\phi^2} = \frac{1}{2}X^2 \pm \frac{g}{2}X\phi^2 \pm \frac{g}{2}X\phi^2 = \frac{1}{2}\left(X \pm \frac{g}{2}\phi^2\right)\left(X \pm \frac{g}{2}\phi^2\right) - \frac{g^2}{4!}\phi^4 \tag{9.21}$$

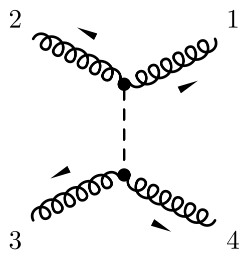
Explain the factor 1/3 in the functional setting and its relation to the three diagrams in the graphical setting?

Quartic Gauge Couplings

The three crossed versions of figure 9.2 reproduces the quartic coupling in figure 9.1, because

$$\begin{aligned} &-\mathrm{i}g^2 f_{a_1 a_2 b} f_{a_3 a_4 b}(g_{\mu_1\mu_3} g_{\mu_4\mu_2} - g_{\mu_1\mu_4} g_{\mu_2\mu_3}) \\ &\qquad = (\mathrm{i}g f_{a_1 a_2 b} T_{\mu_1\mu_2,\nu_1\nu_2})\left(\frac{\mathrm{i}g^{\nu_1\nu_3} g^{\nu_2\nu_4}}{2}\right)(\mathrm{i}g f_{a_3 a_4 b} T_{\mu_3\mu_4,\nu_3\nu_4}) \end{aligned} \tag{9.24}$$

with $T_{\mu_1\mu_2,\mu_3\mu_4} = g_{\mu_1\mu_3} g_{\mu_4\mu_2} - g_{\mu_1\mu_4} g_{\mu_2\mu_3}$.



$$= -\mathrm{i}g^2 f_{a_1a_2b}f_{a_3a_4b}(g_{\mu_1\mu_3}g_{\mu_4\mu_2} - g_{\mu_1\mu_4}g_{\mu_2\mu_3}) \tag{9.23}$$

Figure 9.2: Gauge couplings.

9.1.6 Gravitinos and supersymmetric currents

In supergravity theories there is a fermionic partner of the graviton, the gravitino. Therefore we have introduced the Lorentz type *Vectorspinor*.

9.1.7 Perturbative Quantum Gravity and Kaluza-Klein Interactions

The gravitational coupling constant and the relative strength of the dilaton coupling are abbreviated as

$$\kappa = \sqrt{16\pi G_N} \tag{9.25a}$$

$$\omega = \sqrt{\frac{2}{3(n+2)}} = \sqrt{\frac{2}{3(d-2)}}\,, \tag{9.25b}$$

where $n = d - 4$ is the number of extra space dimensions.
In (9.27-9.34), we use the notation of [13]:

$$C_{\mu\nu,\rho\sigma} = g_{\mu\rho}g_{\nu\sigma} + g_{\mu\sigma}g_{\nu\rho} - g_{\mu\nu}g_{\rho\sigma} \tag{9.26a}$$

$$\begin{aligned} D_{\mu\nu,\rho\sigma}(k_1,k_2) &= g_{\mu\nu}k_{1,\sigma}k_{2,\rho} \\ &\quad - (g_{\mu\sigma}k_{1,\nu}k_{2,\rho} + g_{\mu\rho}k_{1,\sigma}k_{2,\nu} - g_{\rho\sigma}k_{1,\mu}k_{2,\nu} + (\mu \leftrightarrow \nu)) \end{aligned} \tag{9.26b}$$

$$\begin{aligned} E_{\mu\nu,\rho\sigma}(k_1,k_2) &= g_{\mu\nu}(k_{1,\rho}k_{1,\sigma} + k_{2,\rho}k_{2,\sigma} + k_{1,\rho}k_{2,\sigma}) \\ &\quad - (g_{\nu\sigma}k_{1,\mu}k_{1,\rho} + g_{\nu\rho}k_{2,\mu}k_{2,\sigma} + (\mu \leftrightarrow \nu)) \end{aligned} \tag{9.26c}$$

$$\begin{aligned} &F_{\mu\nu,\rho\sigma\lambda}(k_1,k_2,k_3) = \\ &g_{\mu\rho}g_{\sigma\lambda}(k_2-k_3)_\nu + g_{\mu\sigma}g_{\lambda\rho}(k_3-k_1)_\nu + g_{\mu\lambda}g_{\rho\sigma}(k_1-k_2)_\nu + (\mu \leftrightarrow \nu) \end{aligned} \tag{9.26d}$$

$$\begin{aligned} G_{\mu\nu,\rho\sigma\lambda\delta} &= g_{\mu\nu}(g_{\rho\sigma}g_{\lambda\delta} - g_{\rho\delta}g_{\lambda\sigma}) \\ &\quad + (g_{\mu\rho}g_{\nu\delta}g_{\lambda\sigma} + g_{\mu\lambda}g_{\nu\sigma}g_{\rho\delta} - g_{\mu\rho}g_{\nu\sigma}g_{\lambda\delta} - g_{\mu\lambda}g_{\nu\delta}g_{\rho\sigma} + (\mu \leftrightarrow \nu)) \end{aligned} \tag{9.26e}$$

Derivation of (9.27a)

GBG (*Fermbar*, *MOM*, *Ferm*): $\bar\psi_1(\mathrm{i}\partial\!\!\!/ \pm m)\phi\psi_2$	
F12: $\psi_2 \leftarrow -(k\!\!\!/ \mp m)\psi_1 S$	*F21*: $\psi_2 \leftarrow -S(k\!\!\!/ \mp m)\psi_1$
F13: $S \leftarrow \psi_1^T \mathrm{C}(k\!\!\!/ \pm m)\psi_2$	*F31*: $S \leftarrow \psi_2^T \mathrm{C}(-(k\!\!\!/ \mp m)\psi_1)$
F23: $\psi_1 \leftarrow S(k\!\!\!/ \pm m)\psi_2$	*F32*: $\psi_1 \leftarrow (k\!\!\!/ \pm m)\psi_2 S$
GBG (*Fermbar*, *MOM5*, *Ferm*): $\bar\psi_1(\mathrm{i}\partial\!\!\!/ \pm m)\phi\gamma^5\psi_2$	
F12: $\psi_2 \leftarrow (k\!\!\!/ \pm m)\gamma^5\psi_1 P$	*F21*: $\psi_2 \leftarrow P(k\!\!\!/ \pm m)\gamma^5\psi_1$
F13: $P \leftarrow \psi_1^T \mathrm{C}(k\!\!\!/ \pm m)\gamma^5\psi_2$	*F31*: $P \leftarrow \psi_2^T \mathrm{C}(k\!\!\!/ \pm m)\gamma^5\psi_1$
F23: $\psi_1 \leftarrow P(k\!\!\!/ \pm m)\gamma^5\psi_2$	*F32*: $\psi_1 \leftarrow (k\!\!\!/ \pm m)\gamma^5\psi_2 P$
GBG (*Fermbar*, *MOML*, *Ferm*): $\bar\psi_1(\mathrm{i}\partial\!\!\!/ \pm m)\phi(1-\gamma^5)\psi_2$	
F12: $\psi_2 \leftarrow -(1-\gamma^5)(k\!\!\!/ \mp m)\psi_1\phi$	*F21*: $\psi_2 \leftarrow -\phi(1-\gamma^5)(k\!\!\!/ \mp m)\psi_1$
F13: $\phi \leftarrow \psi_1^T \mathrm{C}(k\!\!\!/ \pm m)(1-\gamma^5)\psi_2$	*F31*: $\phi \leftarrow \psi_2^T \mathrm{C}(1-\gamma^5)(-(k\!\!\!/ \mp m)\psi_1)$
F23: $\psi_1 \leftarrow \phi(k\!\!\!/ \pm m)(1-\gamma^5)\psi_2$	*F32*: $\psi_1 \leftarrow (k\!\!\!/ \pm m)(1-\gamma^5)\psi_2\phi$
GBG (*Fermbar*, *LMOM*, *Ferm*): $\bar\psi_1\phi(1-\gamma^5)(\mathrm{i}\partial\!\!\!/ \pm m)\psi_2$	
F12: $\psi_2 \leftarrow -(k\!\!\!/ \mp m)\psi_1(1-\gamma^5)\phi$	*F21*: $\psi_2 \leftarrow -\phi(k\!\!\!/ \mp m)(1-\gamma^5)\psi_1$
F13: $\phi \leftarrow \psi_1^T \mathrm{C}(1-\gamma^5)(k\!\!\!/ \pm m)\psi_2$	*F31*: $\phi \leftarrow \psi_2^T \mathrm{C}(-(k\!\!\!/ \mp m)(1-\gamma^5)\psi_1)$
F23: $\psi_1 \leftarrow \phi(1-\gamma^5)(k\!\!\!/ \pm m)\psi_2$	*F32*: $\psi_1 \leftarrow (1-\gamma^5)(k\!\!\!/ \pm m)\psi_2\phi$
GBG (*Fermbar*, *VMOM*, *Ferm*): $\bar\psi_1\mathrm{i}\partial\!\!\!/_\alpha V_\beta[\gamma^\alpha,\gamma^\beta]\psi_2$	
F12: $\psi_2 \leftarrow -[k\!\!\!/,\gamma^\alpha]\psi_1 V_\alpha$	*F21*: $\psi_2 \leftarrow -[k\!\!\!/,V\!\!\!/]\psi_1$
F13: $V_\alpha \leftarrow \psi_1^T \mathrm{C}[k\!\!\!/,\gamma_\alpha]\psi_2$	*F31*: $V_\alpha \leftarrow \psi_2^T \mathrm{C}(-[k\!\!\!/,\gamma_\alpha]\psi_1)$
F23: $\psi_1 \leftarrow]k\!\!\!/,V\!\!\!/]\psi_2$	*F32*: $\psi_1 \leftarrow [k\!\!\!/,\gamma^\alpha]\psi_2 V_\alpha$

Table 9.23: Combined dimension-4 trilinear fermionic couplings including a momentum. $Ferm$ stands for Psi and Chi. The case of $MOMR$ is identical to $MOML$ if one substitutes $1+\gamma^5$ for $1-\gamma^5$, as well as for $LMOM$ and $RMOM$. The mass term forces us to keep the chiral projector always on the left after "inverting the line" for $MOML$ while on the right for $LMOM$.

GBBG (*Fermbar*, *S2LR*, *Ferm*): $\bar\psi_1 S_1 S_2 (g_L P_L + g_R P_R)\psi_2$	
F123 F213 F132 F231 F312 F321:	$\psi_2 \leftarrow S_1 S_2 (g_R P_L + g_L P_R)\psi_1$
F423 F243 F432 F234 F342 F324:	$\psi_1 \leftarrow S_1 S_2 (g_L P_L + g_R P_R)\psi_2$
F134 F143 F314:	$S_1 \leftarrow \psi_1^T C S_2 (g_L P_L + g_R P_R)\psi_2$
F124 F142 F214:	$S_2 \leftarrow \psi_1^T C S_1 (g_L P_L + g_R P_R)\psi_2$
F413 F431 F341:	$S_1 \leftarrow \psi_2^T C S_2 (g_R P_L + g_L P_R)\psi_1$
F412 F421 F241:	$S_2 \leftarrow \psi_2^T C S_1 (g_R P_L + g_L P_R)\psi_1$
GBBG (*Fermbar*, *S2*, *Ferm*): $\bar\psi_1 S_1 S_2 \gamma^5 \psi_2$	
F123 F213 F132 F231 F312 F321:	$\psi_2 \leftarrow S_1 S_2 \gamma^5 \psi_1$
F423 F243 F432 F234 F342 F324:	$\psi_1 \leftarrow S_1 S_2 \gamma^5 \psi_2$
F134 F143 F314:	$S_1 \leftarrow \psi_1^T C S_2 \gamma^5 \psi_2$
F124 F142 F214:	$S_2 \leftarrow \psi_1^T C S_1 \gamma^5 \psi_2$
F413 F431 F341:	$S_1 \leftarrow \psi_2^T C S_2 \gamma^5 \psi_1$
F412 F421 F241:	$S_2 \leftarrow \psi_2^T C S_1 \gamma^5 \psi_1$
GBBG (*Fermbar*, *V2*, *Ferm*): $\bar\psi_1 [\not{V}_1, \not{V}_2] \psi_2$	
F123 F213 F132 F231 F312 F321:	$\psi_2 \leftarrow -[\not{V}_1, \not{V}_2] \psi_1$
F423 F243 F432 F234 F342 F324:	$\psi_1 \leftarrow [\not{V}_1, \not{V}_2] \psi_2$
F134 F143 F314:	$V_{1\,\alpha} \leftarrow \psi_1^T C [\gamma_\alpha, \not{V}_2] \psi_2$
F124 F142 F214:	$V_{2\,\alpha} \leftarrow \psi_1^T C (-[\gamma_\alpha, \not{V}_1]) \psi_2$
F413 F431 F341:	$V_{1\,\alpha} \leftarrow \psi_2^T C (-[\gamma_\alpha, \not{V}_2]) \psi_1$
F412 F421 F241:	$V_{2\,\alpha} \leftarrow \psi_2^T C [\gamma_\alpha, \not{V}_1] \psi_1$

Table 9.24: Vertices with two fermions (*Ferm* stands for *Psi* and *Chi*, but not for *Grav*) and two bosons (two scalars, scalar/vector, two vectors) for the BRST transformations. Part I

GBBG (*Fermbar*, *SV*, *Ferm*): $\bar\psi_1 \not{V} S \psi_2$	
F123 F213 F132 F231 F312 F321:	$\psi_2 \leftarrow -\not{V} S \psi_1$
F423 F243 F432 F234 F342 F324:	$\psi_1 \leftarrow \not{V} S \psi_2$
F134 F143 F314:	$V_\alpha \leftarrow \psi_1^T C \gamma_\alpha S \psi_2$
F124 F142 F214:	$S \leftarrow \psi_1^T C \not{V} \psi_2$
F413 F431 F341:	$V_\alpha \leftarrow \psi_2^T C(-\gamma_\alpha S \psi_1)$
F412 F421 F241:	$S \leftarrow \psi_2^T C(-\not{V} \psi_1)$
GBBG (*Fermbar*, *PV*, *Ferm*): $\bar\psi_1 \not{V} \gamma^5 P \psi_2$	
F123 F213 F132 F231 F312 F321:	$\psi_2 \leftarrow \not{V} \gamma^5 P \psi_1$
F423 F243 F432 F234 F342 F324:	$\psi_1 \leftarrow \not{V} \gamma^5 P \psi_2$
F134 F143 F314:	$V_\alpha \leftarrow \psi_1^T C \gamma_\alpha \gamma^5 P \psi_2$
F124 F142 F214:	$P \leftarrow \psi_1^T C \not{V} \gamma^5 \psi_2$
F413 F431 F341:	$V_\alpha \leftarrow \psi_2^T C \gamma_\alpha \gamma^5 P \psi_1$
F412 F421 F241:	$P \leftarrow \psi_2^T C \not{V} \gamma^5 \psi_1$
GBBG (*Fermbar*, *S(L/R)V*, *Ferm*): $\bar\psi_1 \not{V} (1 \mp \gamma^5) \phi \psi_2$	
F123 F213 F132 F231 F312 F321:	$\psi_2 \leftarrow -\not{V} (1 \pm \gamma^5) \phi \psi_1$
F423 F243 F432 F234 F342 F324:	$\psi_1 \leftarrow \not{V} (1 \mp \gamma^5) \phi \psi_2$
F134 F143 F314:	$V_\alpha \leftarrow \psi_1^T C \gamma_\alpha (1 \mp \gamma^5) \phi \psi_2$
F124 F142 F214:	$\phi \leftarrow \psi_1^T C \not{V} (1 \mp \gamma^5) \psi_2$
F413 F431 F341:	$V_\alpha \leftarrow \psi_2^T C \gamma_\alpha (-(1 \pm \gamma^5) \phi \psi_1)$
F412 F421 F241:	$\phi \leftarrow \psi_2^T C \not{V} (-(1 \pm \gamma^5) \psi_1)$

Table 9.25: Vertices with two fermions (*Ferm* stands for *Psi* and *Chi*, but not for *Grav*) and two bosons (two scalars, scalar/vector, two vectors) for the BRST transformations. Part II

$GBG\ (Gravbar,\ POT,\ Psi)$: $\bar{\psi}_\mu S\gamma^\mu\psi$	
$F12$: $\psi \leftarrow -\gamma^\mu\psi_\mu S$	$F21$: $\psi \leftarrow -S\gamma^\mu\psi_\mu$
$F13$: $S \leftarrow \psi_\mu^T \mathrm{C}\gamma^\mu\psi$	$F31$: $S \leftarrow \psi^T \mathrm{C}(-\gamma^\mu)\psi_\mu$
$F23$: $\psi_\mu \leftarrow S\gamma_\mu\psi$	$F32$: $\psi_\mu \leftarrow \gamma_\mu\psi S$
$GBG\ (Gravbar,\ S,\ Psi)$: $\bar{\psi}_\mu \not{k}_S S\gamma^\mu\psi$	
$F12$: $\psi \leftarrow \gamma^\mu \not{k}_S \psi_\mu S$	$F21$: $\psi \leftarrow S\gamma^\mu \not{k}_S \psi_\mu$
$F13$: $S \leftarrow \psi_\mu^T \mathrm{C}\not{k}_S \gamma^\mu\psi$	$F31$: $S \leftarrow \psi^T \mathrm{C}\gamma^\mu \not{k}_S \psi_\mu$
$F23$: $\psi_\mu \leftarrow S\not{k}_S \gamma_\mu\psi$	$F32$: $\psi_\mu \leftarrow \not{k}_S \gamma_\mu\psi S$
$GBG\ (Gravbar,\ P,\ Psi)$: $\bar{\psi}_\mu \not{k}_P P\gamma^\mu\gamma_5\psi$	
$F12$: $\psi \leftarrow \gamma^\mu \not{k}_P \gamma_5\psi_\mu P$	$F21$: $\psi \leftarrow P\gamma^\mu \not{k}_P \gamma_5\psi_\mu$
$F13$: $P \leftarrow \psi_\mu^T \mathrm{C}\not{k}_P \gamma^\mu\gamma_5\psi$	$F31$: $P \leftarrow \psi^T \mathrm{C}\gamma^\mu \not{k}_P \gamma_5\psi_\mu$
$F23$: $\psi_\mu \leftarrow P\not{k}_P \gamma_\mu\gamma_5\psi$	$F32$: $\psi_\mu \leftarrow \not{k}_P \gamma_\mu\gamma_5\psi P$
$GBG\ (Gravbar,\ V,\ Psi)$: $\bar{\psi}_\mu[\not{k}_V, \not{V}]\gamma^\mu\gamma^5\psi$	
$F12$: $\psi \leftarrow \gamma^5\gamma^\mu[\not{k}_V, \gamma^\alpha]\psi_\mu V_\alpha$	$F21$: $\psi \leftarrow \gamma^5\gamma^\mu[\not{k}_V, \not{V}]\psi_\mu$
$F13$: $V_\mu \leftarrow \psi_\rho^T \mathrm{C}[\not{k}_V, \gamma_\mu]\gamma^\rho\gamma^5\psi$	$F31$: $V_\mu \leftarrow \psi^T \mathrm{C}\gamma^5\gamma^\rho[\not{k}_V, \gamma_\mu]\psi_\rho$
$F23$: $\psi_\mu \leftarrow [\not{k}_V, \not{V}]\gamma_\mu\gamma^5\psi$	$F32$: $\psi_\mu \leftarrow [\not{k}_V, \gamma^\alpha]\gamma_\mu\gamma^5\psi V_\alpha$

Table 9.26: Dimension-5 trilinear couplings including one Dirac, one Gravitino fermion and one additional particle.The option POT is for the coupling of the supersymmetric current to the derivative of the quadratic terms in the superpotential.

GBG (*Psibar*, *POT*, *Grav*): $\bar{\psi}\gamma^\mu S\psi_\mu$	
F12: $\psi_\mu \leftarrow -\gamma_\mu\psi S$	*F21*: $\psi_\mu \leftarrow -S\gamma_\mu\psi$
F13: $S \leftarrow \psi^T \mathrm{C}\gamma^\mu\psi_\mu$	*F31*: $S \leftarrow \psi_\mu^T \mathrm{C}(-\gamma^\mu)\psi$
F23: $\psi \leftarrow S\gamma^\mu\psi_\mu$	*F32*: $\psi \leftarrow \gamma^\mu\psi_\mu S$
GBG (*Psibar*, *S*, *Grav*): $\bar{\psi}\gamma^\mu \not{k}_S S\psi_\mu$	
F12: $\psi_\mu \leftarrow \not{k}_S\gamma_\mu\psi S$	*F21*: $\psi_\mu \leftarrow S\not{k}_S\gamma_\mu\psi$
F13: $S \leftarrow \psi^T \mathrm{C}\gamma^\mu\not{k}_S\psi_\mu$	*F31*: $S \leftarrow \psi_\mu^T \mathrm{C}\not{k}_S\gamma^\mu\psi$
F23: $\psi \leftarrow S\gamma^\mu\not{k}_S\psi_\mu$	*F32*: $\psi \leftarrow \gamma^\mu\not{k}_S\psi_\mu S$
GBG (*Psibar*, *P*, *Grav*): $\bar{\psi}\gamma^\mu\gamma^5 P\not{k}_P\psi_\mu$	
F12: $\psi_\mu \leftarrow -\not{k}_P\gamma_\mu\gamma^5\psi P$	*F21*: $\psi_\mu \leftarrow -P\not{k}_P\gamma_\mu\gamma^5\psi$
F13: $P \leftarrow \psi^T \mathrm{C}\gamma^\mu\gamma^5\not{k}_P\psi_\mu$	*F31*: $P \leftarrow -\psi_\mu^T \mathrm{C}\not{k}_P\gamma^\mu\gamma_5\psi$
F23: $\psi \leftarrow P\gamma^\mu\gamma^5\not{k}_P\psi_\mu$	*F32*: $\psi \leftarrow \gamma^\mu\gamma^5\not{k}_P\psi_\mu P$
GBG (*Psibar*, *V*, *Grav*): $\bar{\psi}\gamma^5\gamma^\mu[\not{k}_V, \not{V}]\psi_\mu$	
F12: $\psi_\mu \leftarrow [\not{k}_V, \gamma^\alpha]\gamma_\mu\gamma^5\psi V_\alpha$	*F21*: $\psi_\mu \leftarrow [\not{k}_V, \not{V}]\gamma_\mu\gamma^5\psi$
F13: $V_\mu \leftarrow \psi^T \mathrm{C}\gamma^5\gamma^\rho[\not{k}_V, \gamma_\mu]\psi_\rho$	*F31*: $V_\mu \leftarrow \psi_\rho^T \mathrm{C}[\not{k}_V, \gamma_\mu]\gamma^\rho\gamma^5\psi$
F23: $\psi \leftarrow \gamma^5\gamma^\mu[\not{k}_V, \not{V}]\psi_\mu$	*F32*: $\psi \leftarrow \gamma^5\gamma^\mu[\not{k}_V, \gamma^\alpha]\psi_\mu V_\alpha$

Table 9.27: Dimension-5 trilinear couplings including one conjugated Dirac, one Gravitino fermion and one additional particle.

GBG $(Gravbar,\ POT,\ Chi)$: $\bar\psi_\mu S\gamma^\mu\chi$	
$F12$: $\chi \leftarrow -\gamma^\mu\psi_\mu S$	$F21$: $\chi \leftarrow -S\gamma^\mu\psi_\mu$
$F13$: $S \leftarrow \psi_\mu^T \mathrm{C}\gamma^\mu\chi$	$F31$: $S \leftarrow \chi^T \mathrm{C}(-\gamma^\mu)\psi_\mu$
$F23$: $\psi_\mu \leftarrow S\gamma_\mu\chi$	$F32$: $\psi_\mu \leftarrow \gamma_\mu\chi S$
GBG $(Gravbar,\ S,\ Chi)$: $\bar\psi_\mu \not{k}_S S\gamma^\mu\chi$	
$F12$: $\chi \leftarrow \gamma^\mu\not{k}_S\psi_\mu S$	$F21$: $\chi \leftarrow S\gamma^\mu\not{k}_S\psi_\mu$
$F13$: $S \leftarrow \psi_\mu^T \mathrm{C}\not{k}_S\gamma^\mu\chi$	$F31$: $S \leftarrow \chi^T \mathrm{C}\gamma^\mu\not{k}_S\psi_\mu$
$F23$: $\psi_\mu \leftarrow S\not{k}_S\gamma_\mu\chi$	$F32$: $\psi_\mu \leftarrow \not{k}_S\gamma_\mu\chi S$
GBG $(Gravbar,\ P,\ Chi)$: $\bar\psi_\mu \not{k}_P P\gamma^\mu\gamma_5\chi$	
$F12$: $\chi \leftarrow \gamma^\mu\not{k}_P\gamma_5\psi_\mu P$	$F21$: $\chi \leftarrow P\gamma^\mu\not{k}_P\gamma_5\psi_\mu$
$F13$: $P \leftarrow \psi_\mu^T \mathrm{C}\not{k}_P\gamma^\mu\gamma_5\chi$	$F31$: $P \leftarrow \chi^T \mathrm{C}\gamma^\mu\not{k}_P\gamma_5\psi_\mu$
$F23$: $\psi_\mu \leftarrow P\not{k}_P\gamma_\mu\gamma_5\chi$	$F32$: $\psi_\mu \leftarrow \not{k}_P\gamma_\mu\gamma_5\chi P$
GBG $(Gravbar,\ V,\ Chi)$: $\bar\psi_\mu[\not{k}_V, \not{V}]\gamma^\mu\gamma^5\chi$	
$F12$: $\chi \leftarrow \gamma^5\gamma^\mu[\not{k}_V, \gamma^\alpha]\psi_\mu V_\alpha$	$F21$: $\chi \leftarrow \gamma^5\gamma^\mu[\not{k}_V, \not{V}]\psi_\mu$
$F13$: $V_\mu \leftarrow \psi_\rho^T \mathrm{C}[\not{k}_V, \gamma_\mu]\gamma^\rho\gamma^5\chi$	$F31$: $V_\mu \leftarrow \chi^T \mathrm{C}\gamma^5\gamma^\rho[\not{k}_V, \gamma_\mu]\psi_\rho$
$F23$: $\psi_\mu \leftarrow [\not{k}_V, \not{V}]\gamma_\mu\gamma^5\chi$	$F32$: $\psi_\mu \leftarrow [\not{k}_V, \gamma^\alpha]\gamma_\mu\gamma^5\chi V_\alpha$

Table 9.28: Dimension-5 trilinear couplings including one Majorana, one Gravitino fermion and one additional particle. The table is essentially the same as the one with the Dirac fermion and only written for the sake of completeness.

GBG (*Chibar*, *POT*, *Grav*): $\bar{\chi}\gamma^\mu S\psi_\mu$		
F12: $\psi_\mu \leftarrow -\gamma_\mu\chi S$	*F21*: $\psi_\mu \leftarrow -S\gamma_\mu\chi$
F13: $S \leftarrow \chi^T \mathrm{C}\gamma^\mu\psi_\mu$	*F31*: $S \leftarrow \psi_\mu^T \mathrm{C}(-\gamma^\mu)\chi$
F23: $\chi \leftarrow S\gamma^\mu\psi_\mu$	*F32*: $\chi \leftarrow \gamma^\mu\psi_\mu S$
GBG (*Chibar*, *S*, *Grav*): $\bar{\chi}\gamma^\mu \not{k}_S S\psi_\mu$	
F12: $\psi_\mu \leftarrow \not{k}_S\gamma_\mu\chi S$	*F21*: $\psi_\mu \leftarrow S\not{k}_S\gamma_\mu\chi$
F13: $S \leftarrow \chi^T \mathrm{C}\gamma^\mu\not{k}_S\psi_\mu$	*F31*: $S \leftarrow \psi_\mu^T \mathrm{C}\not{k}_S\gamma^\mu\chi$
F23: $\chi \leftarrow S\gamma^\mu\not{k}_S\psi_\mu$	*F32*: $\chi \leftarrow \gamma^\mu\not{k}_S\psi_\mu S$
GBG (*Chibar*, *P*, *Grav*): $\bar{\chi}\gamma^\mu\gamma^5 P\not{k}_P\psi_\mu$	
F12: $\psi_\mu \leftarrow -\not{k}_P\gamma_\mu\gamma^5\chi P$	*F21*: $\psi_\mu \leftarrow -P\not{k}_P\gamma_\mu\gamma^5\chi$
F13: $P \leftarrow \chi^T \mathrm{C}\gamma^\mu\gamma^5\not{k}_P\psi_\mu$	*F31*: $P \leftarrow -\psi_\mu^T \mathrm{C}\not{k}_P\gamma^\mu\gamma_5\chi$
F23: $\chi \leftarrow P\gamma^\mu\gamma^5\not{k}_P\psi_\mu$	*F32*: $\chi \leftarrow \gamma^\mu\gamma^5\not{k}_P\psi_\mu P$
GBG (*Chibar*, *V*, *Grav*): $\bar{\chi}\gamma^5\gamma^\mu[\not{k}_V, \not{V}]\psi_\mu$	
F12: $\psi_\mu \leftarrow [\not{k}_V, \gamma^\alpha]\gamma_\mu\gamma^5\chi V_\alpha$	*F21*: $\psi_\mu \leftarrow [\not{k}_V, \not{V}]\gamma_\mu\gamma^5\chi$
F13: $V_\mu \leftarrow \chi^T \mathrm{C}\gamma^5\gamma^\rho[\not{k}_V, \gamma_\mu]\psi_\rho$	*F31*: $V_\mu \leftarrow \psi_\rho^T \mathrm{C}[\not{k}_V, \gamma_\mu]\gamma^\rho\gamma^5\chi$
F23: $\chi \leftarrow \gamma^5\gamma^\mu[\not{k}_V, \not{V}]\psi_\mu$	*F32*: $\chi \leftarrow \gamma^5\gamma^\mu[\not{k}_V, \gamma^\alpha]\psi_\mu V_\alpha$

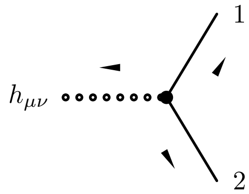
Table 9.29: Dimension-5 trilinear couplings including one conjugated Majorana, one Gravitino fermion and one additional particle. This table is not only the same as the one with the conjugated Dirac fermion but also the same part of the Lagrangian density as the one with the Majorana particle on the right of the gravitino.

GBBG (*Gravbar*, *S2*, *Psi*): $\bar{\psi}_\mu S_1 S_2 \gamma^\mu \psi$	
F123 F213 F132 F231 F312 F321:	$\psi \leftarrow -\gamma^\mu S_1 S_2 \psi_\mu$
F423 F243 F432 F234 F342 F324:	$\psi_\mu \leftarrow \gamma_\mu S_1 S_2 \psi$
F134 F143 F314:	$S_1 \leftarrow \psi_\mu^T C S_2 \gamma^\mu \psi$
F124 F142 F214:	$S_2 \leftarrow \psi_\mu^T C S_1 \gamma^\mu \psi$
F413 F431 F341:	$S_1 \leftarrow -\psi^T C S_2 \gamma^\mu \psi_\mu$
F412 F421 F241:	$S_2 \leftarrow -\psi^T C S_1 \gamma^\mu \psi_\mu$
GBBG (*Gravbar*, *SV*, *Psi*): $\bar{\psi}_\mu S \slashed{V} \gamma^\mu \gamma^5 \psi$	
F123 F213 F132 F231 F312 F321:	$\psi \leftarrow \gamma^5 \gamma^\mu S \slashed{V} \psi_\mu$
F423 F243 F432 F234 F342 F324:	$\psi_\mu \leftarrow \slashed{V} S \gamma_\mu \gamma^5 \psi$
F134 F143 F314:	$S \leftarrow \psi_\mu^T C \slashed{V} \gamma^\mu \gamma^5 \psi$
F124 F142 F214:	$V_\mu \leftarrow \psi_\rho^T C S \gamma_\mu \gamma^\rho \gamma^5 \psi$
F413 F431 F341:	$S \leftarrow \psi^T C \gamma^5 \gamma^\mu \slashed{V} \psi_\mu$
F412 F421 F241:	$V_\mu \leftarrow \psi^T C S \gamma^5 \gamma^\rho \gamma_\mu \psi_\rho$
GBBG (*Gravbar*, *PV*, *Psi*): $\bar{\psi}_\mu P \slashed{V} \gamma^\mu \psi$	
F123 F213 F132 F231 F312 F321:	$\psi \leftarrow \gamma^\mu P \slashed{V} \psi_\mu$
F423 F243 F432 F234 F342 F324:	$\psi_\mu \leftarrow \slashed{V} P \gamma_\mu \psi$
F134 F143 F314:	$P \leftarrow \psi_\mu^T C \slashed{V} \gamma^\mu \psi$
F124 F142 F214:	$V_\mu \leftarrow \psi_\rho^T C P \gamma_\mu \gamma^\rho \psi$
F413 F431 F341:	$P \leftarrow \psi^T C \gamma^\mu \slashed{V} \psi_\mu$
F412 F421 F241:	$V_\mu \leftarrow \psi^T C P \gamma^\rho \gamma_\mu \psi_\rho$
GBBG (*Gravbar*, *V2*, *Psi*): $\bar{\psi}_\mu f_{abc} [\slashed{V}^a, \slashed{V}^b] \gamma^\mu \gamma^5 \psi$	
F123 F213 F132 F231 F312 F321:	$\psi \leftarrow f_{abc} \gamma^5 \gamma^\mu [\slashed{V}^a, \slashed{V}^b] \psi_\mu$
F423 F243 F432 F234 F342 F324:	$\psi_\mu \leftarrow f_{abc} [\slashed{V}^a, \slashed{V}^b] \gamma_\mu \gamma^5 \psi$
F134 F143 F314 F124 F142 F214:	$V_\mu^a \leftarrow \psi_\rho^T C f_{abc} [\gamma_\mu, \slashed{V}^b] \gamma^\rho \gamma^5 \psi$
F413 F431 F341 F412 F421 F241:	$V_\mu^a \leftarrow \psi^T C f_{abc} \gamma^5 \gamma^\rho [\gamma_\mu, \slashed{V}^b] \psi_\rho$

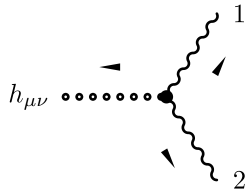
Table 9.30: Dimension-5 trilinear couplings including one Dirac, one Gravitino fermion and two additional bosons. In each lines we list the fusion possibilities with the same order of the fermions, but the order of the bosons is arbitrary (of course, one has to take care of this order in the mapping of the wave functions in *fusion*).

$GBBG\ (Psibar,\ S2,\ Grav)$: $\bar\psi S_1 S_2 \gamma^\mu \psi_\mu$	
$F123\ F213\ F132\ F231\ F312\ F321$:	$\psi_\mu \leftarrow -\gamma_\mu S_1 S_2 \psi$
$F423\ F243\ F432\ F234\ F342\ F324$:	$\psi \leftarrow \gamma^\mu S_1 S_2 \psi_\mu$
$F134\ F143\ F314$:	$S_1 \leftarrow \psi^T C S_2 \gamma^\mu \psi_\mu$
$F124\ F142\ F214$:	$S_2 \leftarrow \psi^T C S_1 \gamma^\mu \psi_\mu$
$F413\ F431\ F341$:	$S_1 \leftarrow -\psi_\mu^T C S_2 \gamma^\mu \psi$
$F412\ F421\ F241$:	$S_2 \leftarrow -\psi_\mu^T C S_1 \gamma^\mu \psi$
$GBBG\ (Psibar,\ SV,\ Grav)$: $\bar\psi S \gamma^\mu \gamma^5 \not{V} \psi_\mu$	
$F123\ F213\ F132\ F231\ F312\ F321$:	$\psi_\mu \leftarrow \not{V} S \gamma^5 \gamma^\mu \psi$
$F423\ F243\ F432\ F234\ F342\ F324$:	$\psi \leftarrow \gamma^\mu \gamma^5 S \not{V} \psi_\mu$
$F134\ F143\ F314$:	$S \leftarrow \psi^T C \gamma^\mu \gamma^5 \not{V} \psi$
$F124\ F142\ F214$:	$V_\mu \leftarrow \psi^T C \gamma^\rho \gamma^5 S \gamma_\mu \psi_\rho$
$F413\ F431\ F341$:	$S \leftarrow \psi_\mu^T C \not{V} \gamma^5 \gamma^\mu \psi$
$F412\ F421\ F241$:	$V_\mu \leftarrow \psi_\rho^T C S \gamma_\mu \gamma^5 \gamma^\rho \psi$
$GBBG\ (Psibar,\ PV,\ Grav)$: $\bar\psi P \gamma^\mu \not{V} \psi_\mu$	
$F123\ F213\ F132\ F231\ F312\ F321$:	$\psi_\mu \leftarrow \not{V} \gamma_\mu P \psi$
$F423\ F243\ F432\ F234\ F342\ F324$:	$\psi \leftarrow \gamma^\mu \not{V} P \psi_\mu$
$F134\ F143\ F314$:	$P \leftarrow \psi^T C \gamma^\mu \not{V} \psi_\mu$
$F124\ F142\ F214$:	$V_\mu \leftarrow \psi^T C P \gamma^\rho \gamma_\mu \psi_\rho$
$F413\ F431\ F341$:	$P \leftarrow \psi_\mu^T C \not{V} \gamma^\mu \psi$
$F412\ F421\ F241$:	$V_\mu \leftarrow \psi_\rho^T C P \gamma_\mu \gamma^\rho \psi$
$GBBG\ (Psibar,\ V2,\ Grav)$: $\bar\psi f_{abc} \gamma^5 \gamma^\mu [\not{V}^a, \not{V}^b] \psi_\mu$	
$F123\ F213\ F132\ F231\ F312\ F321$:	$\psi_\mu \leftarrow f_{abc} [\not{V}^a, \not{V}^b] \gamma_\mu \gamma^5 \psi$
$F423\ F243\ F432\ F234\ F342\ F324$:	$\psi \leftarrow f_{abc} \gamma^5 \gamma^\mu [\not{V}^a, \not{V}^b] \psi_\mu$
$F134\ F143\ F314\ F124\ F142\ F214$:	$V_\mu^a \leftarrow \psi^T C f_{abc} \gamma^5 \gamma^\rho [\gamma_\mu, \not{V}^b] \psi_\rho$
$F413\ F431\ F341\ F412\ F421\ F241$:	$V_\mu^a \leftarrow \psi_\rho^T C f_{abc} [\gamma_\mu, \not{V}^b] \gamma^\rho \gamma^5 \psi$

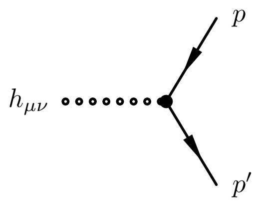
Table 9.31: Dimension-5 trilinear couplings including one conjugated Dirac, one Gravitino fermion and two additional bosons. The couplings of Majorana fermions to the gravitino and two bosons are essentially the same as for Dirac fermions and they are omitted here.



$$= -\mathrm{i}\frac{\kappa}{2}g_{\mu\nu}m^2 + \mathrm{i}\frac{\kappa}{2}C_{\mu\nu,\mu_1\mu_2}k_1^{\mu_1}k_2^{\mu_2} \tag{9.27a}$$



$$\begin{aligned} = -\mathrm{i}\frac{\kappa}{2}m^2 C_{\mu\nu,\mu_1\mu_2} - \mathrm{i}\frac{\kappa}{2}(k_1 k_2 C_{\mu\nu,\mu_1\mu_2} & \\ + D_{\mu\nu,\mu_1\mu_2}(k_1,k_2) & \\ + \xi^{-1}E_{\mu\nu,\mu_1\mu_2}(k_1,k_2)) & \end{aligned} \tag{9.27b}$$

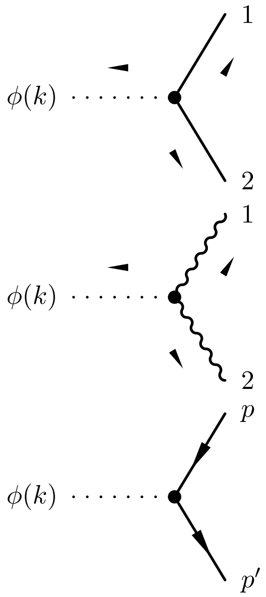


$$\begin{aligned} = -\mathrm{i}\frac{\kappa}{2}m g_{\mu\nu} - \mathrm{i}\frac{\kappa}{8}(\gamma_\mu(p+p')_\nu + \gamma_\nu(p+p')_\mu & \\ - 2g_{\mu\nu}(\not{p} + \not{p}')) & \end{aligned} \tag{9.27c}$$

Figure 9.3: Three-point graviton couplings.

Graviton_Scalar_Scalar: $h_{\mu\nu}C_0^{\mu\nu}(k_1,k_2)\phi_1\phi_2$
F12 | *F21*: $\phi_2 \leftarrow \mathrm{i}\cdot h_{\mu\nu}C_0^{\mu\nu}(k_1,-k-k_1)\phi_1$
F13 | *F31*: $\phi_1 \leftarrow \mathrm{i}\cdot h_{\mu\nu}C_0^{\mu\nu}(-k-k_2,k_2)\phi_2$
F23 | *F32*: $h^{\mu\nu} \leftarrow \mathrm{i}\cdot C_0^{\mu\nu}(k_1,k_2)\phi_1\phi_2$
Graviton_Vector_Vector: $h_{\mu\nu}C_1^{\mu\nu,\mu_1\mu_2}(k_1,k_2,\xi)V_{\mu_1}V_{\mu_2}$
F12 | *F21*: $V_2^\mu \leftarrow \mathrm{i}\cdot h_{\kappa\lambda}C_1^{\kappa\lambda,\mu\nu}(-k-k_1,k_1\xi)V_{1,\nu}$
F13 | *F31*: $V_1^\mu \leftarrow \mathrm{i}\cdot h_{\kappa\lambda}C_1^{\kappa\lambda,\mu\nu}(-k-k_2,k_2,\xi)V_{2,\nu}$
F23 | *F32*: $h^{\mu\nu} \leftarrow \mathrm{i}\cdot C_1^{\mu\nu,\mu_1\mu_2}(k_1,k_2,\xi)V_{1,\mu_1}V_{2,\mu_2}$
Graviton_Spinor_Spinor: $h_{\mu\nu}\bar\psi_1 C_{\frac{1}{2}}^{\mu\nu}(k_1,k_2)\psi_2$
F12: $\bar\psi_2 \leftarrow \mathrm{i}\cdot h_{\mu\nu}\bar\psi_1 C_{\frac{1}{2}}^{\mu\nu}(k_1,-k-k_1)$
F21: $\bar\psi_2 \leftarrow \mathrm{i}\cdot\ldots$
F13: $\psi_1 \leftarrow \mathrm{i}\cdot h_{\mu\nu}C_{\frac{1}{2}}^{\mu\nu}(-k-k_2,k_2)\psi_2$
F31: $\psi_1 \leftarrow \mathrm{i}\cdot\ldots$
F23: $h^{\mu\nu} \leftarrow \mathrm{i}\cdot\bar\psi_1 C_{\frac{1}{2}}^{\mu\nu}(k_1,k_2)\psi_2$
F32: $h^{\mu\nu} \leftarrow \mathrm{i}\cdot\ldots$

Table 9.32: ...



$$= -\mathrm{i}\omega\kappa 2m^2 - \mathrm{i}\omega\kappa k_1 k_2 \tag{9.30a}$$

$$= -\mathrm{i}\omega\kappa g_{\mu_1\mu_2} m^2 - \mathrm{i}\omega\kappa\xi^{-1}(k_{1,\mu_1}k_{\mu_2} + k_{2,\mu_2}k_{\mu_1}) \tag{9.30b}$$

$$= -\mathrm{i}\omega\kappa 2m + \mathrm{i}\omega\kappa\frac{3}{4}(\not p + \not p') \tag{9.30c}$$

Figure 9.4: Three-point dilaton couplings.

$$L = \frac{1}{2}(\partial_\mu\phi)(\partial^\mu\phi) - \frac{m^2}{2}\phi^2 \tag{9.28a}$$

$$(\partial_\mu\phi)\frac{\partial L}{\partial(\partial^\nu\phi)} = (\partial_\mu\phi)(\partial_\nu\phi) \tag{9.28b}$$

$$T_{\mu\nu} = -g_{\mu\nu}L + (\partial_\mu\phi)\frac{\partial L}{\partial(\partial^\nu\phi)} + \tag{9.28c}$$

$$C_0^{\mu\nu}(k_1,k_2) = C^{\mu\nu,\mu_1\mu_2}k_{1,\mu_1}k_{2,\mu_2} \tag{9.29a}$$

$$C_1^{\mu\nu,\mu_1\mu_2}(k_1,k_2,\xi) = k_1k_2C^{\mu\nu,\mu_1\mu_2} + D^{\mu\nu,\mu_1\mu_2}(k_1,k_2) + \xi^{-1}E^{\mu\nu,\mu_1\mu_2}(k_1,k_2) \tag{9.29b}$$

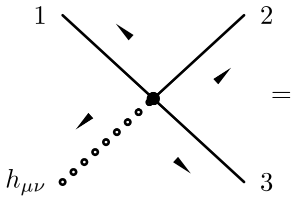
$$C_{\frac{1}{2},\alpha\beta}^{\mu\nu}(p,p') = \gamma_{\alpha\beta}^{\mu}(p+p')^\nu + \gamma_{\alpha\beta}^{\nu}(p+p')^\mu - 2g^{\mu\nu}(\not p + \not p')_{\alpha\beta} \tag{9.29c}$$

9.1.8 Dependent Parameters

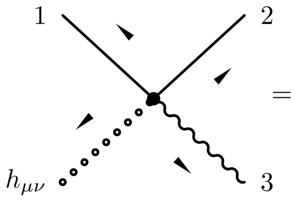
This is a simple abstract syntax for parameter dependencies. Later, there will be a parser for a convenient concrete syntax as a part of a concrete syntax for models. There is no intention to do *any* symbolic manipulation with this. The expressions will be translated directly by *Targets* to the target language.

Dilaton_Scalar_Scalar: $\phi \ldots k_1 k_2 \phi_1 \phi_2$	
F12 | *F21*:	$\phi_2 \leftarrow \mathrm{i} \cdot k_1(-k-k_1)\phi\phi_1$
F13 | *F31*:	$\phi_1 \leftarrow \mathrm{i} \cdot (-k-k_2)k_2\phi\phi_2$
F23 | *F32*:	$\phi \leftarrow \mathrm{i} \cdot k_1 k_2 \phi_1 \phi_2$
Dilaton_Vector_Vector: $\phi \ldots$	
F12:	$V_{2,\mu} \leftarrow \mathrm{i} \cdot \ldots$
F21:	$V_{2,\mu} \leftarrow \mathrm{i} \cdot \ldots$
F13:	$V_{1,\mu} \leftarrow \mathrm{i} \cdot \ldots$
F31:	$V_{1,\mu} \leftarrow \mathrm{i} \cdot \ldots$
F23:	$\phi \leftarrow \mathrm{i} \cdot \ldots$
F32:	$\phi \leftarrow \mathrm{i} \cdot \ldots$
Dilaton_Spinor_Spinor: $\phi \ldots$	
F12:	$\bar\psi_2 \leftarrow \mathrm{i} \cdot \ldots$
F21:	$\bar\psi_2 \leftarrow \mathrm{i} \cdot \ldots$
F13:	$\psi_1 \leftarrow \mathrm{i} \cdot \ldots$
F31:	$\psi_1 \leftarrow \mathrm{i} \cdot \ldots$
F23:	$\phi \leftarrow \mathrm{i} \cdot \ldots$
F32:	$\phi \leftarrow \mathrm{i} \cdot \ldots$

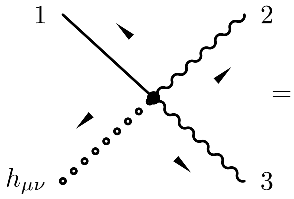
Table 9.33: ...



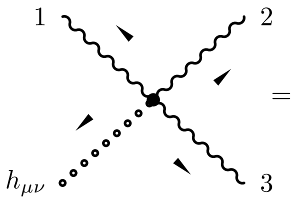
$$= \quad ??? \tag{9.31a}$$



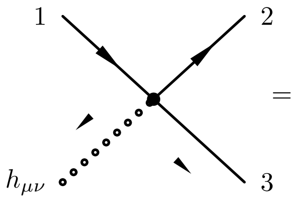
$$= -\mathrm{i}g\frac{\kappa}{2}C_{\mu\nu,\mu_3\rho}(k_1-k_2)^\rho T^{a_3}_{n_2n_1} \tag{9.31b}$$



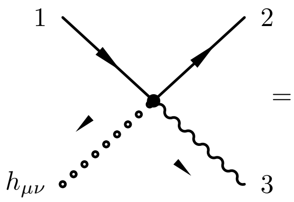
$$= \quad ??? \tag{9.31c}$$



$$\begin{aligned} = -g\frac{\kappa}{2}f^{a_1a_2a_3}(&C_{\mu\nu,\mu_1\mu_2}(k_1-k_2)_{\mu_3} \\ &+C_{\mu\nu,\mu_2\mu_3}(k_2-k_3)_{\mu_1} \\ &+C_{\mu\nu,\mu_3\mu_1}(k_3-k_1)_{\mu_2} \\ &+F_{\mu\nu,\mu_1\mu_2\mu_3}(k_1,k_2,k_3)) \end{aligned} \tag{9.31d}$$

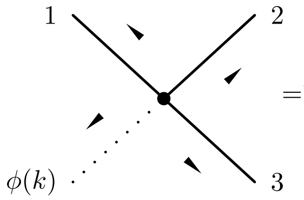


$$= \quad ??? \tag{9.31e}$$

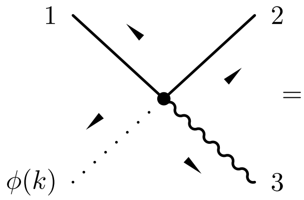


$$= \mathrm{i}g\frac{\kappa}{4}(C_{\mu\nu,\mu_3\rho}-g_{\mu\nu}g_{\mu_3\rho})\gamma^\rho T^{a_3}_{n_2n_1} \tag{9.31f}$$

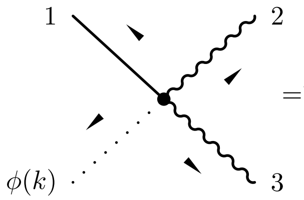
Figure 9.5: Four-point graviton couplings. (9.31a), (9.31c), and (**??** are missing in [13], but should be generated by standard model Higgs selfcouplings, Higgs-gaugeboson couplings, and Yukawa couplings.



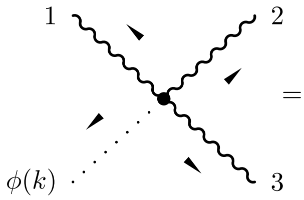
$$=???\tag{9.32a}$$



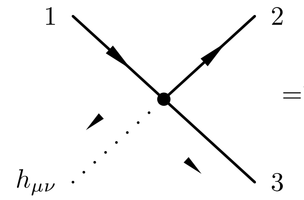
$$= -\mathrm{i}\omega\kappa(k_1+k_2)_{\mu_3}T^{a_3}_{n_1,n_2}\tag{9.32b}$$



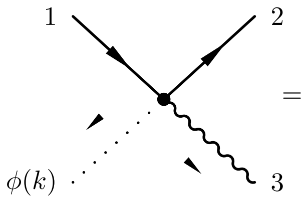
$$=???\tag{9.32c}$$



$$= 0\tag{9.32d}$$

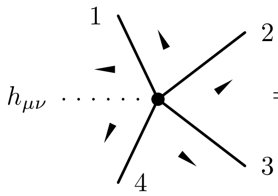


$$=???\tag{9.32e}$$

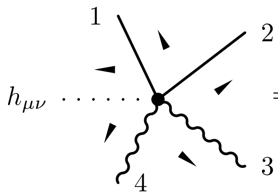


$$= -\mathrm{i}\frac{3}{2}\omega g\kappa\gamma_{\mu_3}T^{a_3}_{n_1n_2}\tag{9.32f}$$

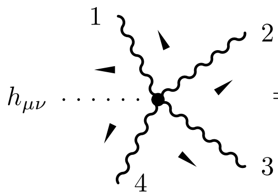
Figure 9.6: Four-point dilaton couplings. (9.32a), (9.32c) and (9.32e) are missing in [13], but could be generated by standard model Higgs selfcouplings, Higgs-gaugeboson couplings, and Yukawa couplings.



$$= \qquad ??? \tag{9.33a}$$

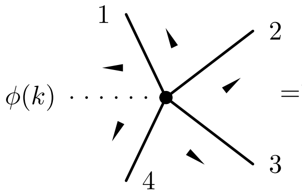


$$= \quad -\mathrm{i}g^2\frac{\kappa}{2}C_{\mu\nu,\mu_3\mu_4}(T^{a_3}T^{a_4}+T^{a_4}T^{a_3})_{n_2n_1} \tag{9.33b}$$

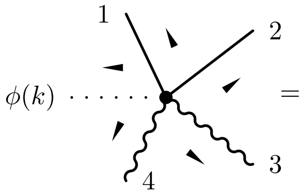


$$\begin{aligned} = \quad -\mathrm{i}g^2\frac{\kappa}{2}(&f^{ba_1a_3}f^{ba_2a_4}G_{\mu\nu,\mu_1\mu_2\mu_3\mu_4} \\ &+f^{ba_1a_2}f^{ba_3a_4}G_{\mu\nu,\mu_1\mu_3\mu_2\mu_4} \\ &+f^{ba_1a_4}f^{ba_2a_3}G_{\mu\nu,\mu_1\mu_2\mu_4\mu_3}) \end{aligned} \tag{9.33c}$$

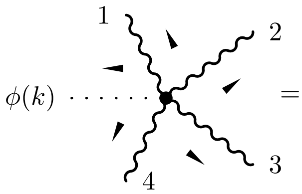
Figure 9.7: Five-point graviton couplings. (9.33a) is missing in [13], but should be generated by standard model Higgs selfcouplings.



$$=??? \tag{9.34a}$$



$$= \mathrm{i}\omega g^2 \kappa g_{\mu_3\mu_4} (T^{a_3}T^{a_4} + T^{a_4}T^{a_3})_{n_2n_1} \tag{9.34b}$$



$$= 0 \tag{9.34c}$$

Figure 9.8: Five-point dilaton couplings. (9.34a) is missing in [13], but could be generated by standard model Higgs selfcouplings.

```
type α expr =
  | I | Const of int
  | Atom of α
  | Sum of α expr list
  | Diff of α expr × α expr
  | Neg of α expr
  | Prod of α expr list
  | Quot of α expr × α expr
  | Rec of α expr
  | Pow of α expr × int
  | Sqrt of α expr
  | Sin of α expr
  | Cos of α expr
  | Tan of α expr
  | Cot of α expr
  | Atan2 of α expr × α expr
  | Conj of α expr

type α variable = Real of α | Complex of α
type α variable_array = Real_Array of α | Complex_Array of α

type α parameters =
    { input : (α × float) list;
      derived : (α variable × α expr) list;
      derived_arrays : (α variable_array × α expr list) list }
```

Dim5_Scalar_Vector_Vector_T: $\mathcal{L}_I = g\phi(\mathrm{i}\partial_\mu V_1^\nu)(\mathrm{i}\partial_\nu V_2^\mu)$	
F23:	$\phi(k_2+k_3) \leftarrow \mathrm{i}\cdot gk_3^\mu V_{1,\mu}(k_2)k_2^\nu V_{2,\nu}(k_3)$
F32:	$\phi(k_2+k_3) \leftarrow \mathrm{i}\cdot gk_2^\mu V_{2,\mu}(k_3)k_3^\nu V_{1,\nu}(k_2)$
F12:	$V_2^\mu(k_1+k_2) \leftarrow \mathrm{i}\cdot gk_2^\mu \phi(k_1)(-k_1^\nu - k_2^\nu)V_{1,\nu}(k_2)$
F21:	$V_2^\mu(k_1+k_2) \leftarrow \mathrm{i}\cdot gk_2^\mu (-k_1^\nu - k_2^\nu)V_{1,\nu}(k_2)\phi(k_1)$
F13:	$V_1^\mu(k_1+k_3) \leftarrow \mathrm{i}\cdot gk_3^\mu \phi(k_1)(-k_1^\nu - k_3^\nu)V_{2,\nu}(k_3)$
F31:	$V_1^\mu(k_1+k_3) \leftarrow \mathrm{i}\cdot gk_3^\mu (-k_1^\nu - k_3^\nu)V_{2,\nu}(k_3)\phi(k_1)$

Table 9.34: ...

Dim6_Vector_Vector_Vector_T: $\mathcal{L}_I = gV_1^\mu((\mathrm{i}\partial_\nu V_2^\rho)\mathrm{i}\overleftrightarrow{\partial_\mu}(\mathrm{i}\partial_\rho V_3^\nu))$	
F23:	$V_1^\mu(k_2+k_3) \leftarrow \mathrm{i}\cdot g(k_2^\mu - k_3^\mu)k_3^\nu V_{2,\nu}(k_2)k_2^\rho V_{3,\rho}(k_3)$
F32:	$V_1^\mu(k_2+k_3) \leftarrow \mathrm{i}\cdot g(k_2^\mu - k_3^\mu)k_2^\nu V_{3,\nu}(k_3)k_3^\rho V_{2,\rho}(k_2)$
F12:	$V_3^\mu(k_1+k_2) \leftarrow \mathrm{i}\cdot gk_2^\mu(k_1^\nu + 2k_2^\nu)V_{1,\nu}(k_1)(-k_1^\rho - k_2^\rho)V_{2,\rho}(k_2)$
F21:	$V_3^\mu(k_1+k_2) \leftarrow \mathrm{i}\cdot gk_2^\mu(-k_1^\rho - k_2^\rho)V_{2,\rho}(k_2)(k_1^\nu + 2k_2^\nu)V_{1,\nu}(k_1)$
F13:	$V_2^\mu(k_1+k_3) \leftarrow \mathrm{i}\cdot gk_3^\mu(k_1^\nu + 2k_3^\nu)V_{1,\nu}(k_1)(-k_1^\rho - k_3^\rho)V_{3,\rho}(k_3)$
F31:	$V_2^\mu(k_1+k_3) \leftarrow \mathrm{i}\cdot gk_3^\mu(-k_1^\rho - k_3^\rho)V_{3,\rho}(k_3)(k_1^\nu + 2k_3^\nu)V_{1,\nu}(k_1)$

Table 9.35: ...

9.1.9 More Exotic Couplings

9.2 Interface of Model

9.2.1 General Quantum Field Theories

module type T =
  sig

flavor abstractly encodes all quantum numbers.

    type flavor

Color.t encodes the (SU(N)) color representation.

    val color : flavor → Color.t

The set of conserved charges.

    module Ch : Charges.T
    val charges : flavor → Ch.t

The PDG particle code for interfacing with Monte Carlos.

    val pdg : flavor → int

The Lorentz representation of the particle.

$Tensor\_2\_Vector\_Vector$: $\mathcal{L}_I = gT^{\mu\nu}(V_{1,\mu}V_{2,\nu} + V_{1,\nu}V_{2,\mu})$
$F23$: $T^{\mu\nu}(k_2+k_3) \leftarrow \mathrm{i}\cdot g(V_{1,\mu}(k_2)V_{2,\nu}(k_3) + V_{1,\nu}(k_2)V_{2,\mu}(k_3))$
$F32$: $T^{\mu\nu}(k_2+k_3) \leftarrow \mathrm{i}\cdot g(V_{2,\nu}(k_3)V_{1,\mu}(k_2) + V_{2,\mu}(k_3)V_{1,\nu}(k_2))$
$F12$: $V_2^\mu(k_1+k_2) \leftarrow \mathrm{i}\cdot g(T^{\mu\nu}(k_1) + T^{\nu\mu}(k_1))V_{1,\nu}(k_2)$
$F21$: $V_2^\mu(k_1+k_2) \leftarrow \mathrm{i}\cdot gV_{1,\nu}(k_2)(T^{\mu\nu}(k_1) + T^{\nu\mu}(k_1))$
$F13$: $V_1^\mu(k_1+k_3) \leftarrow \mathrm{i}\cdot g(T^{\mu\nu}(k_1) + T^{\nu\mu}(k_1))V_{2,\nu}(k_3)$
$F31$: $V_1^\mu(k_1+k_3) \leftarrow \mathrm{i}\cdot gV_{2,\nu}(k_3)(T^{\mu\nu}(k_1) + T^{\nu\mu}(k_1))$

Table 9.36: ...

$Dim5\_Tensor\_2\_Vector\_Vector\_1$: $\mathcal{L}_I = gT^{\alpha\beta}(V_1^\mu \mathrm{i}\overleftrightarrow{\partial}_\alpha \mathrm{i}\overleftrightarrow{\partial}_\beta V_{2,\mu})$
$F23$: $T^{\alpha\beta}(k_2+k_3) \leftarrow \mathrm{i}\cdot g(k_2^\alpha - k_3^\alpha)(k_2^\beta - k_3^\beta)V_1^\mu(k_2)V_{2,\mu}(k_3)$
$F32$: $T^{\alpha\beta}(k_2+k_3) \leftarrow \mathrm{i}\cdot g(k_2^\alpha - k_3^\alpha)(k_2^\beta - k_3^\beta)V_{2,\mu}(k_3)V_1^\mu(k_2)$
$F12$: $V_2^\mu(k_1+k_2) \leftarrow \mathrm{i}\cdot g(k_1^\alpha + 2k_2^\alpha)(k_1^\beta + 2k_2^\beta)T_{\alpha\beta}(k_1)V_1^\mu(k_2)$
$F21$: $V_2^\mu(k_1+k_2) \leftarrow \mathrm{i}\cdot g(k_1^\alpha + 2k_2^\alpha)(k_1^\beta + 2k_2^\beta)V_1^\mu(k_2)T_{\alpha\beta}(k_1)$
$F13$: $V_1^\mu(k_1+k_3) \leftarrow \mathrm{i}\cdot g(k_1^\alpha + 2k_3^\alpha)(k_1^\beta + 2k_3^\beta)T_{\alpha\beta}(k_1)V_2^\mu(k_3)$
$F31$: $V_1^\mu(k_1+k_3) \leftarrow \mathrm{i}\cdot g(k_1^\alpha + 2k_3^\alpha)(k_1^\beta + 2k_3^\beta)V_2^\mu(k_3)T_{\alpha\beta}(k_1)$

Table 9.37: ...

$Dim5\_Tensor\_2\_Vector\_Vector\_2$: $\mathcal{L}_I = gT^{\alpha\beta}(V_1^\mu \mathrm{i}\overleftrightarrow{\partial}_\beta(\mathrm{i}\partial_\mu V_{2,\alpha}) + V_1^\mu \mathrm{i}\overleftrightarrow{\partial}_\alpha(\mathrm{i}\partial_\mu V_{2,\beta}))$
$F23$: $T^{\alpha\beta}(k_2+k_3) \leftarrow \mathrm{i}\cdot g(k_3^\beta - k_2^\beta)k_3^\mu V_{1,\mu}(k_2)V_2^\alpha(k_3) + (\alpha\leftrightarrow\beta)$
$F32$: $T^{\alpha\beta}(k_2+k_3) \leftarrow \mathrm{i}\cdot g(k_3^\beta - k_2^\beta)V_2^\alpha(k_3)k_3^\mu V_{1,\mu}(k_2) + (\alpha\leftrightarrow\beta)$
$F12$: $V_2^\alpha(k_1+k_2) \leftarrow \mathrm{i}\cdot g(k_1^\beta + 2k_2^\beta)(T^{\alpha\beta}(k_1) + T^{\beta\alpha}(k_1))(k_1^\mu + k_2^\mu)V_{1,\mu}(k_2)$
$F21$: $V_2^\alpha(k_1+k_2) \leftarrow \mathrm{i}\cdot g(k_1^\mu + k_2^\mu)V_{1,\mu}(k_2)(k_1^\beta + 2k_2^\beta)(T^{\alpha\beta}(k_1) + T^{\beta\alpha}(k_1))$
$F13$: $V_1^\alpha(k_1+k_3) \leftarrow \mathrm{i}\cdot g(k_1^\beta + 2k_3^\beta)(T^{\alpha\beta}(k_1) + T^{\beta\alpha}(k_1))(k_1^\mu + k_3^\mu)V_{2,\mu}(k_3)$
$F31$: $V_1^\alpha(k_1+k_3) \leftarrow \mathrm{i}\cdot g(k_1^\mu + k_3^\mu)V_{2,\mu}(k_3)(k_1^\beta + 2k_3^\beta)(T^{\alpha\beta}(k_1) + T^{\beta\alpha}(k_1))$

Table 9.38: ...

Dim7_Tensor_2_Vector_Vector_T: $\mathcal{L}_I = gT^{\alpha\beta}((\mathrm{i}\partial^\mu V_1^\nu)\mathrm{i}\overleftrightarrow{\partial}_\alpha \mathrm{i}\overleftrightarrow{\partial}_\beta(\mathrm{i}\partial_\nu V_{2,\mu}))$	
F23:	$T^{\alpha\beta}(k_2+k_3) \leftarrow \mathrm{i}\cdot g(k_2^\alpha - k_3^\alpha)(k_2^\beta - k_3^\beta)k_3^\mu V_{1,\mu}(k_2)k_2^\nu V_{2,\nu}(k_3)$
F32:	$T^{\alpha\beta}(k_2+k_3) \leftarrow \mathrm{i}\cdot g(k_2^\alpha - k_3^\alpha)(k_2^\beta - k_3^\beta)k_2^\nu V_{2,\nu}(k_3)k_3^\mu V_{1,\mu}(k_2)$
F12:	$V_2^\mu(k_1+k_2) \leftarrow \mathrm{i}\cdot gk_2^\mu(k_1^\alpha + 2k_2^\alpha)(k_1^\beta + 2k_2^\beta)T_{\alpha\beta}(k_1)(-k_1^\nu - k_2^\nu)V_{1,\nu}(k_2)$
F21:	$V_2^\mu(k_1+k_2) \leftarrow \mathrm{i}\cdot gk_2^\mu(-k_1^\nu - k_2^\nu)V_{1,\nu}(k_2)(k_1^\alpha + 2k_2^\alpha)(k_1^\beta + 2k_2^\beta)T_{\alpha\beta}(k_1)$
F13:	$V_1^\mu(k_1+k_3) \leftarrow \mathrm{i}\cdot gk_3^\mu(k_1^\alpha + 2k_3^\alpha)(k_1^\beta + 2k_3^\beta)T_{\alpha\beta}(k_1)(-k_1^\nu - k_3^\nu)V_{2,\nu}(k_3)$
F31:	$V_1^\mu(k_1+k_3) \leftarrow \mathrm{i}\cdot gk_3^\mu(-k_1^\nu - k_3^\nu)V_{2,\nu}(k_3)(k_1^\alpha + 2k_3^\alpha)(k_1^\beta + 2k_3^\beta)T_{\alpha\beta}(k_1)$

Table 9.39: ...

val *lorentz* : *flavor* → *Coupling.lorentz*

The propagator for the particle, which *can* depend on a gauge parameter.

type *gauge*
val *propagator* : *flavor* → *gauge Coupling.propagator*

Not the symbol for the numerical value, but the scheme or strategy.

val *width* : *flavor* → *Coupling.width*

Charge conjugation, with and without color.

val *conjugate* : *flavor* → *flavor*

Returns 1 for fermions, −1 for anti-fermions, 2 for Majoranas and 0 otherwise.

val *fermion* : *flavor* → *int*

The Feynman rules. *vertices* and (*fuse2*, *fuse3*, *fusen*) are redundant, of course. However, *vertices* is required for building functors for models and *vertices* can be recovered from (*fuse2*, *fuse3*, *fusen*) only at great cost.

Nevertheless: *vertices* is a candidate for removal, b/c we can build a smarter *Colorize* functor acting on (*fuse2*, *fuse3*, *fusen*). It can support an arbitrary numer of color lines. But we have to test whether it is efficient enough.

type *constant*

Later: type *orders* to count orders of couplings

val *max_degree* : *unit* → *int*
val *vertices* : *unit* →
((((*flavor* × *flavor* × *flavor*) × *constant Coupling.vertex3* × *constant*) *list*)
× (((*flavor* × *flavor* × *flavor* × *flavor*) × *constant Coupling.vertex4* × *constant*) *list*)
× (((*flavor list*) × *constant Coupling.vertexn* × *constant*) *list*))
val *fuse2* : *flavor* → *flavor* → (*flavor* × *constant Coupling.t*) *list*
val *fuse3* : *flavor* → *flavor* → *flavor* → (*flavor* × *constant Coupling.t*) *list*
val *fuse* : *flavor list* → (*flavor* × *constant Coupling.t*) *list*

Later: val *orders* : *constant* → *orders* counting orders of couplings

The list of all known flavors.

```
val flavors : unit → flavor list
```

The flavors that can appear in incoming or outgoing states, grouped in a way that is useful for user interfaces.

```
val external_flavors : unit → (string × flavor list) list
```

The Goldstone bosons corresponding to a gauge field, if any.

```
val goldstone : flavor → (flavor × constant Coupling.expr) option
```

The dependent parameters.

```
val parameters : unit → constant Coupling.parameters
```

Translate from and to convenient textual representations of flavors.

```
val flavor_of_string : string → flavor
val flavor_to_string : flavor → string
```

TeX and LaTeX

```
val flavor_to_TeX : flavor → string
```

The following must return unique symbols that are acceptable as symbols in all programming languages under consideration as targets. Strings of alphanumeric characters (starting with a letter) should be safe. Underscores are also usable, but would violate strict Fortran77.

```
val flavor_symbol : flavor → string
val gauge_symbol : gauge → string
val mass_symbol : flavor → string
val width_symbol : flavor → string
val constant_symbol : constant → string
```

Model specific options.

```
val options : Options.t
```

Revision control information.

```
val rcs : RCS.t
```

```
end
```

In addition to hardcoded models, we can have models that are initialized at run time.

9.2.2 Mutable Quantum Field Theories

```
module type Mutable =
  sig
    include T
```

Export only one big initialization function to discourage partial initializations. Labels make this usable.

```
val setup :
```

```
        color : (flavor → Color.t) →
        pdg : (flavor → int) →
        lorentz : (flavor → Coupling.lorentz) →
        propagator : (flavor → gauge Coupling.propagator) →
        width : (flavor → Coupling.width) →
        goldstone : (flavor → (flavor × constant Coupling.expr) option) →
        conjugate : (flavor → flavor) →
        fermion : (flavor → int) →
        max_degree :int →
        vertices : (unit →
          ((((flavor × flavor × flavor) × constant Coupling.vertex3 ×
constant) list)
              × (((flavor × flavor × flavor × flavor) × constant Coupling.vertex4 ×
constant) list)
              × (((flavor list) × constant Coupling.vertexn × constant) list))) →

        fuse : ((flavor → flavor → (flavor × constant Coupling.t) list)
                 × (flavor → flavor → flavor →
                   (flavor × constant Coupling.t) list)
                 × (flavor list → (flavor × constant Coupling.t) list)) →
        flavors : ((string × flavor list) list) →
        parameters : (unit → constant Coupling.parameters) →
        flavor_of_string : (string → flavor) →
        flavor_to_string : (flavor → string) →
        flavor_to_TeX : (flavor → string) →
        flavor_symbol : (flavor → string) →
        gauge_symbol : (gauge → string) →
        mass_symbol : (flavor → string) →
        width_symbol : (flavor → string) →
        constant_symbol : (constant → string) →
        unit
  end
```

9.2.3 Gauge Field Theories

The following signatures are used only for model building. The diagrammatics and numerics is supposed to be completely ignorant about the detail of the models and expected to rely on the interface T exclusively.

In the end, we might have functors $(M : T) \rightarrow Gauge$, but we will need to add the quantum numbers to T.

```
module type Gauge =
  sig
    include T
```

Matter field carry conserved quantum numbers and can be replicated in generations without changing the gauge sector.

```
    type matter_field
```

Gauge bosons proper.

```
type gauge_boson
```

Higgses, Goldstones and all the rest:

```
type other
```

We can query the kind of field

```
type field =
  | Matter of matter_field
  | Gauge of gauge_boson
  | Other of other
val field : flavor → field
```

and we can build new fields of a given kind:

```
  val matter_field : matter_field → flavor
  val gauge_boson : gauge_boson → flavor
  val other : other → flavor
end
```

9.2.4 Gauge Field Theories with Broken Gauge Symmetries

Both are carefully crafted as subtypes of *Gauge* so that they can be used in place of *Gauge* and *T* everywhere:

```
module type Broken_Gauge =
  sig
    include Gauge

    type massless
    type massive
    type goldstone

    type kind =
      | Massless of massless
      | Massive of massive
      | Goldstone of goldstone
    val kind : gauge_boson → kind

    val massless : massive → gauge_boson
    val massive : massive → gauge_boson
    val goldstone : goldstone → gauge_boson

  end

module type Unitarity_Gauge =
  sig
    include Gauge

    type massless
    type massive

    type kind =
      | Massless of massless
      | Massive of massive
    val kind : gauge_boson → kind
```

```
    val massless : massive → gauge_boson
    val massive : massive → gauge_boson
  end
module type Colorized =
  sig
    include T
    type flavor_sans_color
    val flavor_sans_color : flavor → flavor_sans_color
    val conjugate_sans_color : flavor_sans_color → flavor_sans_color
    val nc : unit → int
    val amplitude : flavor_sans_color list → flavor_sans_color list →
      (flavor list × flavor list) list
    val flow : flavor list → flavor list → Color.Flow.t
  end
module type Colorized_Gauge =
  sig
    include Gauge
    type flavor_sans_color
    val flavor_sans_color : flavor → flavor_sans_color
    val conjugate_sans_color : flavor_sans_color → flavor_sans_color
    val nc : unit → int
    val amplitude : flavor_sans_color list → flavor_sans_color list →
      (flavor list × flavor list) list
    val flow : flavor list → flavor list → Color.Flow.t
  end
```

Interface `vertex.mli` *unavailable!*

Implementation `vertex.ml` *unavailable!*

9.3 Interface of Target

```
module type T =
  sig
    type amplitudes
    val options : Options.t
    type diagnostic = All | Arguments | Momenta | Gauge
```

Format the amplitudes as a sequence of strings.

```
    val amplitudes_to_channel : string → out_channel →
      (diagnostic × bool) list → amplitudes → unit
    val parameters_to_channel : out_channel → unit
```

```
    val rcs_list : RCS.t list
  end

module type Maker =
    functor (F : Fusion.Maker) →
      functor (P : Momentum.T) → functor (M : Model.T) →
        T with type amplitudes = Fusion.Multi(F)(P)(M).amplitudes
```

—10— Conserved Quantum Numbers

10.1 Interface of Charges

10.1.1 Abstract Type

module type T =
  sig

The abstract type of the set of conserved charges or additive quantum numbers.

    type t

Add the quantum numbers of a pair or a list of particles.

    val add : $t \rightarrow t \rightarrow t$
    val sum : t $list \rightarrow t$

Test the charge conservation.

    val is_null : $t \rightarrow bool$

  end

10.1.2 Trivial Realisation

module $Null$: T with type t = $unit$

10.1.3 Nontrivial Realisations

Z

module Z : T with type t = int

$\mathbf{Z} \times \mathbf{Z} \times \cdots \times \mathbf{Z}$

module ZZ : T with type t = int $list$

Q

module Q : T with type $t = Algebra.Small\_Rational.t$

$$\mathbf{Q} \times \mathbf{Q} \times \cdots \times \mathbf{Q}$$

module QQ : T with type $t = Algebra.Small\_Rational.t\ list$

10.2 Implementation of Charges

```
module type T =
  sig
    type t
    val add : t → t → t
    val sum : t list → t
    val is_null : t → bool
  end

module Null : T with type t = unit =
  struct
    type t = unit
    let add () () = ()
    let sum _ = ()
    let is_null _ = true
  end

module Z : T with type t = int =
  struct
    type t = int
    let add = ( + )
    let sum = List.fold_left add 0
    let is_null n = (n = 0)
  end

module ZZ : T with type t = int list =
  struct
    type t = int list
    let add = List.map2 ( + )
    let sum = function
      | [] → []
      | [charges] → charges
      | charges :: rest → List.fold_left add charges rest
    let is_null = List.for_all (fun n → n = 0)
  end

module Rat = Algebra.Small_Rational

module Q : T with type t = Rat.t =
  struct
    type t = Rat.t
    let add = Rat.add
```

```
    let sum = List.fold_left Rat.add Rat.null
    let is_null = Rat.is_null
  end

module QQ : T with type t = Rat.t list =
  struct
    type t = Rat.t list
    let add = List.map2 Rat.add
    let sum = function
      | [] → []
      | [charges] → charges
      | charges :: rest → List.fold_left add charges rest
    let is_null = List.for_all Rat.is_null
  end
```

—11—
Colorization

11.1 Interface of Colorize

11.1.1 ...

```
module It (M : Model.T) :
  Model.Colorized with type flavor_sans_color = M.flavor
  and type constant = M.constant

module Gauge (M : Model.Gauge) :
  Model.Colorized_Gauge with type flavor_sans_color = M.flavor
  and type constant = M.constant
```

11.2 Implementation of Colorize

```
let rcs_file = RCS.parse "Colorize" ["Colorizing Monochrome Models"]
  { RCS.revision = "$Revision: 6465 $";
    RCS.date = "$Date: 2015-01-10 16:22:31 +0100 (Sat, 10 Jan 2015) $";
    RCS.author = "$Author: jr_reuter $";
    RCS.source
      = "$URL: svn+ssh://jr_reuter@login.hepforge.org/hepforge/svn/whizard/trunk/omega/sr
```

11.2.1 Colorizing a Monochrome Model

```
module It (M : Model.T) =
 struct

  let rcs = RCS.rename rcs_file "Colorize.It()"
      [ "Colorizing Generic Monochrome Models"]

  open Coupling

  module C = Color

  let incomplete s =
    failwith ("Colorize.It()." ^ s ^ " not done yet!")

  let invalid s =
    invalid_arg ("Colorize.It()." ^ s ^ " must not be evaluated!")
```

```
let impossible s =
  invalid_arg ("Colorize.It()." ^ s ^ " can't happen! (but just did ...)")

let su0 s =
  invalid_arg ("Colorize.It()." ^ s ^ ": found SU(0)!")

let colored_vertex s =
  invalid_arg ("Colorize.It()." ^ s ^ ": colored vertex!")

let baryonic_vertex s =
  invalid_arg ("Colorize.It()." ^ s ^
               ": baryonic (i.e. eps_ijk) vertices not supported yet!")

let color_flow_ambiguous s =
  invalid_arg ("Colorize.It()." ^ s ^ ": ambiguous color flow!")

let color_flow_of_string s =
  let c = int_of_string s in
  if c < 1 then
    invalid_arg ("Colorize.It()." ^ s ^ ": color flow # < 1!")
  else
    c

type cf_in = int
type cf_out = int

type flavor =
  | White of M.flavor
  | CF_in of M.flavor × cf_in
  | CF_out of M.flavor × cf_out
  | CF_io of M.flavor × cf_in × cf_out
  | CF_aux of M.flavor

type flavor_sans_color = M.flavor

let flavor_sans_color = function
  | White f → f
  | CF_in (f, _) → f
  | CF_out (f, _) → f
  | CF_io (f, _, _) → f
  | CF_aux f → f

let pullback f arg1 =
  f (flavor_sans_color arg1)

type gauge = M.gauge
type constant = M.constant
let options = M.options

let color = pullback M.color
let pdg = pullback M.pdg
let lorentz = pullback M.lorentz

module Ch = M.Ch
let charges = pullback M.charges
```

For the propagator we cannot use pullback because we have to add the case of the color singlet propagator by hand.

```
let cf_aux_propagator = function
  | Prop_Scalar → Prop_Col_Scalar (* Spin 0 octets. *)
  | Prop_Majorana → Prop_Col_Majorana (* Spin 1/2 octets. *)
  | Prop_Feynman → Prop_Col_Feynman (* Spin 1 states, massless. *)
  | Prop_Unitarity → Prop_Col_Unitarity (* Spin 1 states, massive. *)
  | Aux_Scalar → Aux_Col_Scalar (* constant colored scalar propagator *)
  | Aux_Vector → Aux_Col_Vector (* constant colored vector propagator *)
  | Aux_Tensor_1 → Aux_Col_Tensor_1 (* constant colored tensor propagator *)
  | Prop_Col_Scalar | Prop_Col_Feynman
  | Prop_Col_Majorana | Prop_Col_Unitarity
  | Aux_Col_Scalar | Aux_Col_Vector | Aux_Col_Tensor_1
    → failwith ("Colorize.It().colorize_propagator: already colored particle!")
  | _ → failwith ("Colorize.It().colorize_propagator: impossible!")

let propagator = function
  | CF_aux f → cf_aux_propagator (M.propagator f)
  | White f → M.propagator f
  | CF_in (f, _) → M.propagator f
  | CF_out (f, _) → M.propagator f
  | CF_io (f, _, _) → M.propagator f

let width = pullback M.width

let goldstone = function
  | White f →
      begin match M.goldstone f with
      | None → None
      | Some (f', g) → Some (White f', g)
      end
  | CF_in (f, c) →
      begin match M.goldstone f with
      | None → None
      | Some (f', g) → Some (CF_in (f', c), g)
      end
  | CF_out (f, c) →
      begin match M.goldstone f with
      | None → None
      | Some (f', g) → Some (CF_out (f', c), g)
      end
  | CF_io (f, c1, c2) →
      begin match M.goldstone f with
      | None → None
      | Some (f', g) → Some (CF_io (f', c1, c2), g)
      end
  | CF_aux f →
      begin match M.goldstone f with
      | None → None
      | Some (f', g) → Some (CF_aux f', g)
      end
```

```
let conjugate = function
  | White f → White (M.conjugate f)
  | CF_in (f, c) → CF_out (M.conjugate f, c)
  | CF_out (f, c) → CF_in (M.conjugate f, c)
  | CF_io (f, c1, c2) → CF_io (M.conjugate f, c2, c1)
  | CF_aux f → CF_aux (M.conjugate f)

let conjugate_sans_color = M.conjugate

let fermion = pullback M.fermion

let max_degree = M.max_degree

let flavors () =
  invalid "flavors"

let external_flavors () =
  invalid "external_flavors"

let parameters = M.parameters

module ISet = Set.Make (struct type t = int let compare = compare end)

let nc_value =
  let nc_set =
    List.fold_left
      (fun nc_set f →
        match M.color f with
        | C.Singlet → nc_set
        | C.SUN nc → ISet.add (abs nc) nc_set
        | C.AdjSUN nc → ISet.add (abs nc) nc_set)
      ISet.empty (M.flavors ()) in
  match ISet.elements nc_set with
  | [] → 0
  | [n] → n
  | nc_list →
      invalid_arg
        ("Colorize.It(): more than one value of N_C: " ^
         String.concat ", " (List.map string_of_int nc_list))

let nc () =
  nc_value

let split_color_string s =
  try
    let i1 = String.index s '/' in
    let i2 = String.index_from s (succ i1) '/' in
    let sf = String.sub s 0 i1
    and sc1 = String.sub s (succ i1) (i2 - i1 - 1)
    and sc2 = String.sub s (succ i2) (String.length s - i2 - 1) in
    (sf, sc1, sc2)
  with
  | Not_found → (s, "", "")

let flavor_of_string s =
  try
```

```
      let sf, sc1, sc2 = split_color_string s in
      let f = M.flavor_of_string sf in
      match M.color f with
      | C.Singlet → White f
      | C.SUN nc →
          if nc > 0 then
            CF_in (f, color_flow_of_string sc1)
          else
            CF_out (f, color_flow_of_string sc2)
      | C.AdjSUN _ →
          begin match sc1, sc2 with
          | "", "" → CF_aux f
          | _, _ → CF_io (f, color_flow_of_string sc1, color_flow_of_string sc2)
          end
    with
    | Failure "int_of_string" →
        invalid_arg "Colorize().flavor_of_string: expecting integer"

  let flavor_to_string = function
    | White f →
        M.flavor_to_string f
    | CF_in (f, c) →
        M.flavor_to_string f ^ "/" ^ string_of_int c ^ "/"
    | CF_out (f, c) →
        M.flavor_to_string f ^ "//" ^ string_of_int c
    | CF_io (f, c1, c2) →
        M.flavor_to_string f ^ "/" ^ string_of_int c1 ^ "/" ^ string_of_int c2
    | CF_aux f →
        M.flavor_to_string f ^ "//"

  let flavor_to_TeX = function
    | White f →
        M.flavor_to_TeX f
    | CF_in (f, c) →
        "{" ^ M.flavor_to_TeX f ^ "}_{\\mathstrut " ^ string_of_int c ^ "}"
    | CF_out (f, c) →
        "{" ^ M.flavor_to_TeX f ^ "}_{\\mathstrut\\overline{" ^
        string_of_int c ^ "}}"
    | CF_io (f, c1, c2) →
        "{" ^ M.flavor_to_TeX f ^ "}_{\\mathstrut " ^
        string_of_int c1 ^ "\\overline{" ^ string_of_int c2 ^ "}}"
    | CF_aux f →
        "{" ^ M.flavor_to_TeX f ^ "}_{\\mathstrut 0}"

  let flavor_symbol = function
    | White f →
        M.flavor_symbol f
    | CF_in (f, c) →
        M.flavor_symbol f ^ "_" ^ string_of_int c ^ "_"
    | CF_out (f, c) →
        M.flavor_symbol f ^ "__" ^ string_of_int c
    | CF_io (f, c1, c2) →
```

```
          M.flavor_symbol f ^ "_" ^ string_of_int c1 ^ "_" ^ string_of_int c2
      | CF_aux f →
          M.flavor_symbol f ^ "__"

  let gauge_symbol = M.gauge_symbol
```

Masses and widths must not depend on the colors anyway!

```
  let mass_symbol = pullback M.mass_symbol
  let width_symbol = pullback M.width_symbol

  let constant_symbol = M.constant_symbol
```

Vertices

Auxiliary functions

```
  let mult_vertex3 x = function
    | FBF (c, fb, coup, f) →
        FBF ((x × c), fb, coup, f)
    | PBP (c, fb, coup, f) →
        PBP ((x × c), fb, coup, f)
    | BBB (c, fb, coup, f) →
        BBB ((x × c), fb, coup, f)
    | GBG (c, fb, coup, f) →
        GBG ((x × c), fb, coup, f)
    | Gauge_Gauge_Gauge c →
        Gauge_Gauge_Gauge (x × c)
    | Aux_Gauge_Gauge c →
        Aux_Gauge_Gauge (x × c)
    | Scalar_Vector_Vector c →
        Scalar_Vector_Vector (x × c)
    | Aux_Vector_Vector c →
        Aux_Vector_Vector (x × c)
    | Aux_Scalar_Vector c →
        Aux_Scalar_Vector (x × c)
    | Scalar_Scalar_Scalar c →
        Scalar_Scalar_Scalar (x × c)
    | Aux_Scalar_Scalar c →
        Aux_Scalar_Scalar (x × c)
    | Vector_Scalar_Scalar c →
        Vector_Scalar_Scalar (x × c)
    | Graviton_Scalar_Scalar c →
        Graviton_Scalar_Scalar (x × c)
    | Graviton_Vector_Vector c →
        Graviton_Vector_Vector (x × c)
    | Graviton_Spinor_Spinor c →
        Graviton_Spinor_Spinor (x × c)
    | Dim4_Vector_Vector_Vector_T c →
        Dim4_Vector_Vector_Vector_T (x × c)
```

```
  | Dim4_Vector_Vector_Vector_L c →
      Dim4_Vector_Vector_Vector_L (x × c)
  | Dim4_Vector_Vector_Vector_T5 c →
      Dim4_Vector_Vector_Vector_T5 (x × c)
  | Dim4_Vector_Vector_Vector_L5 c →
      Dim4_Vector_Vector_Vector_L5 (x × c)
  | Dim6_Gauge_Gauge_Gauge c →
      Dim6_Gauge_Gauge_Gauge (x × c)
  | Dim6_Gauge_Gauge_Gauge_5 c →
      Dim6_Gauge_Gauge_Gauge_5 (x × c)
  | Aux_DScalar_DScalar c →
      Aux_DScalar_DScalar (x × c)
  | Aux_Vector_DScalar c →
      Aux_Vector_DScalar (x × c)
  | Dim5_Scalar_Gauge2 c →
      Dim5_Scalar_Gauge2 (x × c)
  | Dim5_Scalar_Gauge2_Skew c →
      Dim5_Scalar_Gauge2_Skew (x × c)
  | Dim5_Scalar_Vector_Vector_T c →
      Dim5_Scalar_Vector_Vector_T (x × c)
  | Dim5_Scalar_Vector_Vector_U c →
      Dim5_Scalar_Vector_Vector_U (x × c)
  | Dim5_Scalar_Vector_Vector_TU c →
      Dim5_Scalar_Vector_Vector_TU (x × c)
  | Scalar_Vector_Vector_t c →
      Scalar_Vector_Vector_t (x × c)
  | Dim6_Vector_Vector_Vector_T c →
      Dim6_Vector_Vector_Vector_T (x × c)
  | Tensor_2_Vector_Vector c →
      Tensor_2_Vector_Vector (x × c)
  | Tensor_2_Vector_Vector_1 c →
      Tensor_2_Vector_Vector_1 (x × c)
  | Tensor_2_Vector_Vector_t c →
      Tensor_2_Vector_Vector_t (x × c)
  | Dim5_Tensor_2_Vector_Vector_1 c →
      Dim5_Tensor_2_Vector_Vector_1 (x × c)
  | Dim5_Tensor_2_Vector_Vector_2 c →
      Dim5_Tensor_2_Vector_Vector_2 (x × c)
  | Dim7_Tensor_2_Vector_Vector_T c →
      Dim7_Tensor_2_Vector_Vector_T (x × c)

let mult_vertex4 x = function
  | Scalar4 c →
      Scalar4 (x × c)
  | Scalar2_Vector2 c →
      Scalar2_Vector2 (x × c)
  | Vector4 ic4_list →
      Vector4 (List.map (fun (c, icl) → (x × c, icl)) ic4_list)
  | DScalar4 ic4_list →
      DScalar4 (List.map (fun (c, icl) → (x × c, icl)) ic4_list)
```

```
  | DScalar2_Vector2 ic4_list →
      DScalar2_Vector2 (List.map (fun (c, icl) → (x × c, icl)) ic4_list)
  | GBBG (c, fb, b2, f) →
      GBBG ((x × c), fb, b2, f)
  | Vector4_K_Matrix_tho (c, ic4_list) →
      Vector4_K_Matrix_tho ((x × c), ic4_list)
  | Vector4_K_Matrix_jr (c, ch2_list) →
      Vector4_K_Matrix_jr ((x × c), ch2_list)

let mult_vertexn x = function
  | foo → ignore (incomplete "mult_vertexn"); foo

let mult_vertex x = function
  | V3 (v, fuse, c) → V3 (mult_vertex3 x v, fuse, c)
  | V4 (v, fuse, c) → V4 (mult_vertex4 x v, fuse, c)
  | Vn (v, fuse, c) → Vn (mult_vertexn x v, fuse, c)
```

Below, we will need to permute Lorentz structures. The following permutes the three possible contractions of four vectors. We permute the first three indices, as they correspond to the particles entering the fusion.

```
type permutation4 =
  | P123 | P231 | P312
  | P213 | P321 | P132

let permute_contract4 = function
  | P123 →
      begin function
        | C_12_34 → C_12_34
        | C_13_42 → C_13_42
        | C_14_23 → C_14_23
      end
  | P231 →
      begin function
        | C_12_34 → C_14_23
        | C_13_42 → C_12_34
        | C_14_23 → C_13_42
      end
  | P312 →
      begin function
        | C_12_34 → C_13_42
        | C_13_42 → C_14_23
        | C_14_23 → C_12_34
      end
  | P213 →
      begin function
        | C_12_34 → C_12_34
        | C_13_42 → C_14_23
        | C_14_23 → C_13_42
      end
  | P321 →
      begin function
        | C_12_34 → C_14_23
```

```
          | C_13_42 → C_13_42
          | C_14_23 → C_12_34
        end
    | P132 →
        begin function
          | C_12_34 → C_13_42
          | C_13_42 → C_12_34
          | C_14_23 → C_14_23
        end
```

```
let permute_contract4_list perm ic4_list =
  List.map (fun (i, c4) → (i, permute_contract4 perm c4)) ic4_list
```

```
let permute_vertex4' perm = function
  | Scalar4 c →
      Scalar4 c
  | Vector4 ic4_list →
      Vector4 (permute_contract4_list perm ic4_list)
  | Vector4_K_Matrix_jr (c, ic4_list) →
      Vector4_K_Matrix_jr (c, permute_contract4_list perm ic4_list)
  | Scalar2_Vector2 c →
      incomplete "permute_vertex4'␣Scalar2_Vector2"
  | DScalar4 ic4_list →
      incomplete "permute_vertex4'␣DScalar4"
  | DScalar2_Vector2 ic4_list →
      incomplete "permute_vertex4'␣DScalar2_Vector2"
  | GBBG (c, fb, b2, f) →
      incomplete "permute_vertex4'␣GBBG"
  | Vector4_K_Matrix_tho (c, ch2_list) →
      incomplete "permute_vertex4'␣Vector4_K_Matrix_tho"
```

```
let permute_vertex4 perm = function
  | V3 (v, fuse, c) → V3 (v, fuse, c)
  | V4 (v, fuse, c) → V4 (permute_vertex4' perm v, fuse, c)
  | Vn (v, fuse, c) → Vn (v, fuse, c)
```

vertices are *only* used by functor applications and for indexing a cache of pre-computed fusion rules, which is not used for colorized models.

```
let vertices () =
  invalid "vertices"
```

Cubic Vertices

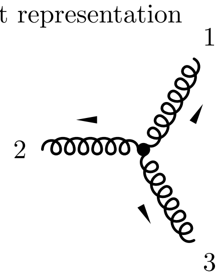
The following pattern matches could eventually become quite long. The O'Caml compiler will (hopefully) optimize them aggressively (http://pauillac.inria.fr/~maranget/papers/opat/).

```
let colorize_fusion2 f1 f2 (f, v) =
  match M.color f with

  | C.Singlet →
```

```
begin match f1, f2 with
| White _, White _ →
    [White f, v]
| CF_in (_, c1), CF_out (_, c2′)
| CF_out (_, c1), CF_in (_, c2′) →
    if c1 = c2′ then
      [White f, v]
    else
      []
| CF_io (f1, c1, c1′), CF_io (f2, c2, c2′) →
    if c1 = c2′ ∧ c2 = c1′ then
      [White f, v]
    else
      []
| CF_aux f1, CF_aux f2 →
    [White f, mult_vertex (− (nc ())) v]
| CF_aux _, CF_io _ | CF_io _, CF_aux _ →
    []
| (CF_in _ | CF_out _ | CF_io _ | CF_aux _), White _
| White _, (CF_in _ | CF_out _ | CF_io _ | CF_aux _)
| (CF_io _ | CF_aux _), (CF_in _ | CF_out _)
| (CF_in _ | CF_out _), (CF_io _ | CF_aux _)
| CF_in _, CF_in _ | CF_out _, CF_out _ →
    colored_vertex "colorize_fusion2"
end
| C.SUN nc1 →
  begin match f1, f2 with
  | CF_in (_, c1), (White _ | CF_aux _)
  | (White _ | CF_aux _), CF_in (_, c1) →
      if nc1 > 0 then
        [CF_in (f, c1), v]
      else
        colored_vertex "colorize_fusion2"
  | CF_out (_, c1′), (White _ | CF_aux _)
  | (White _ | CF_aux _), CF_out (_, c1′) →
      if nc1 < 0 then
        [CF_out (f, c1′), v]
      else
        colored_vertex "colorize_fusion2"
  | CF_in (_, c1), CF_io (_, c2, c2′)
  | CF_io (_, c2, c2′), CF_in (_, c1) →
      if nc1 > 0 then begin
        if c1 = c2′ then
          [CF_in (f, c2), v]
        else
```

```
              []
          end else
            colored_vertex "colorize_fusion2"
      | CF_out (_, c1'), CF_io (_, c2, c2')
      | CF_io (_, c2, c2'), CF_out (_, c1') →
          if nc1 < 0 then begin
            if c1' = c2 then
              [CF_out (f, c2'), v]
            else
              []
          end else
            colored_vertex "colorize_fusion2"
      | CF_in _, CF_in _ →
          if nc1 > 0 then
            baryonic_vertex "colorize_fusion2"
          else
            colored_vertex "colorize_fusion2"
      | CF_out _, CF_out _ →
          if nc1 < 0 then
            baryonic_vertex "colorize_fusion2"
          else
            colored_vertex "colorize_fusion2"
      | CF_in _, CF_out _ | CF_out _, CF_in _
      | (White _ | CF_io _ | CF_aux _),
            (White _ | CF_io _ | CF_aux _) →
          colored_vertex "colorize_fusion2"
      end
  | C.AdjSUN _ →
      begin match f1, f2 with
      | White _, CF_io (_, c1, c2') | CF_io (_, c1, c2'), White _ →
          [CF_io (f, c1, c2'), v]
      | White _, CF_aux _ | CF_aux _, White _ →
          [CF_aux f, mult_vertex (− (nc ())) v]
      | CF_in (_, c1), CF_out (_, c2')
      | CF_out (_, c2'), CF_in (_, c1) →
          if c1 ≠ c2' then
            [CF_io (f, c1, c2'), v]
          else
            [CF_aux f, v]
```

In the adjoint representation



$$= g f_{a_1 a_2 a_3} C^{\mu_1 \mu_2 \mu_3}(k_1, k_2, k_3) \tag{11.1a}$$

with

$$C^{\mu_1\mu_2\mu_3}(k_1,k_2,k_3) = \\ (g^{\mu_1\mu_2}(k_1^{\mu_3}-k_2^{\mu_3}) + g^{\mu_2\mu_3}(k_2^{\mu_1}-k_3^{\mu_1}) + g^{\mu_3\mu_1}(k_3^{\mu_2}-k_1^{\mu_2})) \tag{11.1b}$$

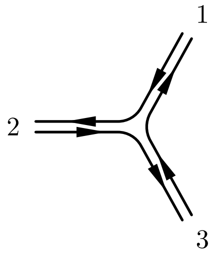
while in the color flow basis find from

$$\mathrm{i}f_{a_1a_2a_3} = \mathrm{tr}\left(T_{a_1}\left[T_{a_2},T_{a_3}\right]\right) = \mathrm{tr}\left(T_{a_1}T_{a_2}T_{a_3}\right) - \mathrm{tr}\left(T_{a_1}T_{a_3}T_{a_2}\right) \tag{11.2}$$

the decomposition

$$\mathrm{i}f_{a_1a_2a_3}T_{a_1}^{i_1j_1}T_{a_2}^{i_2j_2}T_{a_3}^{i_3j_3} = \delta^{i_1j_2}\delta^{i_2j_3}\delta^{i_3j_1} - \delta^{i_1j_3}\delta^{i_3j_2}\delta^{i_2j_1}\,. \tag{11.3}$$

The resulting Feynman rule is



$$= \mathrm{i}g\left(\delta^{i_1j_3}\delta^{i_2j_1}\delta^{i_3j_2} - \delta^{i_1j_2}\delta^{i_2j_3}\delta^{i_3j_1}\right)C^{\mu_1\mu_2\mu_3}(k_1,k_2,k_3) \tag{11.4}$$

We have to generalize this for cases of three particles in the adjoint that are not all gluons (gluinos, scalar octets):

- scalar-scalar-scalar
- scalar-scalar-vector
- scalar-vector-vector
- scalar-fermion-fermion
- vector-fermion-fermion

We could use a better understanding of the signs for the gaugino-gaugino-gaugeboson couplings!!!

```
| CF_io (f1, c1, c1'), CF_io (f2, c2, c2') ->
    let sign =
      begin match v with
      | V3 (Gauge_Gauge_Gauge _, _, _)
      | V3 (Aux_Gauge_Gauge _, _, _) -> 1
      | V3 (FBF (_, _, _, _), fuse2, _) ->
          begin match fuse2 with
          | F12 -> 1 (* works, but needs theoretical underpinning
*)
          | F21 -> -1 (* dto. *)
          | F31 -> 1 (* dto. *)
          | F32 -> -1 (* transposition of F12 (no testcase) *)
          | F23 -> 1 (* transposition of F21 (no testcase) *)
          | F13 -> -1 (* transposition of F12 (no testcase) *)
          end
```

```
          | V3 _ → incomplete "colorize_fusion2␣(V3␣_)"
          | V4 _ → impossible "colorize_fusion2␣(V4␣_)"
          | Vn _ → impossible "colorize_fusion2␣(Vn␣_)"
          end in
        if c1′ = c2 then
          [CF_io (f, c1, c2′), mult_vertex (−sign) v]
        else if c2′ = c1 then
          [CF_io (f, c2, c1′), mult_vertex ( sign) v]
        else
          []

    | CF_aux _ , CF_io _
    | CF_io _ , CF_aux _
    | CF_aux _ , CF_aux _ →
        []

    | White _, White _
    | (White _ | CF_io _ | CF_aux _), (CF_in _ | CF_out _)
    | (CF_in _ | CF_out _), (White _ | CF_io _ | CF_aux _)
    | CF_in _, CF_in _ | CF_out _, CF_out _ →
        colored_vertex "colorize_fusion2"

    end
```

Quartic Vertices

```
let colorize_fusion3 f1 f2 f3 (f, v) =
  match M.color f with

  | C.Singlet →
      begin match f1, f2, f3 with

      | White _, White _, White _ →
          [White f, v]

      | (White _ | CF_aux _), CF_in (_, c1), CF_out (_, c2′)
      | (White _ | CF_aux _), CF_out (_, c1), CF_in (_, c2′)
      | CF_in (_, c1), (White _ | CF_aux _), CF_out (_, c2′)
      | CF_out (_, c1), (White _ | CF_aux _), CF_in (_, c2′)
      | CF_in (_, c1), CF_out (_, c2′), (White _ | CF_aux _)
      | CF_out (_, c1), CF_in (_, c2′), (White _ | CF_aux _) →
          if c1 = c2′ then
            [White f, v]
          else
            []

      | White _, CF_io (_, c1, c1′), CF_io (_, c2, c2′)
      | CF_io (_, c1, c1′), White _, CF_io (_, c2, c2′)
      | CF_io (_, c1, c1′), CF_io (_, c2, c2′), White _ →
          if c1 = c2′ ∧ c2 = c1′ then
            [White f, v]
          else
            []
```

```
| White _, CF_aux _, CF_aux _
| CF_aux _, White _, CF_aux _
| CF_aux _, CF_aux _, White _ ->
    [White f, mult_vertex (- (nc ())) v]
| White _, CF_io _, CF_aux _
| White _, CF_aux _, CF_io _
| CF_io _, White _, CF_aux _
| CF_aux _, White _, CF_io _
| CF_io _, CF_aux _, White _
| CF_aux _, CF_io _, White _ ->
    []
| CF_io (_, c1, c1'), CF_in (_, c2), CF_out (_, c3')
| CF_io (_, c1, c1'), CF_out (_, c3'), CF_in (_, c2)
| CF_in (_, c2), CF_io (_, c1, c1'), CF_out (_, c3')
| CF_out (_, c3'), CF_io (_, c1, c1'), CF_in (_, c2)
| CF_in (_, c2), CF_out (_, c3'), CF_io (_, c1, c1')
| CF_out (_, c3'), CF_in (_, c2), CF_io (_, c1, c1') ->
    if c1 = c3' ∧ c1' = c2 then
      [White f, v]
    else
      []
| CF_io (_, c1, c1'), CF_io (_, c2, c2'), CF_io (_, c3, c3') ->
    if c1' = c2 ∧ c2' = c3 ∧ c3' = c1 then
      [White f, mult_vertex (-1) v]
    else if c1' = c3 ∧ c2' = c1 ∧ c3' = c2 then
      [White f, mult_vertex ( 1) v]
    else
      []
| CF_io _, CF_io _, CF_aux _
| CF_io _, CF_aux _, CF_io _
| CF_aux _, CF_io _, CF_io _
| CF_io _, CF_aux _, CF_aux _
| CF_aux _, CF_io _, CF_aux _
| CF_aux _, CF_aux _, CF_io _
| CF_aux _, CF_aux _, CF_aux _ ->
    []
| CF_in _, CF_in _, CF_in _
| CF_out _, CF_out _, CF_out _ ->
    baryonic_vertex "colorize_fusion3"
| CF_in _, CF_in _, CF_out _
| CF_in _, CF_out _, CF_in _
| CF_out _, CF_in _, CF_in _
| CF_in _, CF_out _, CF_out _
| CF_out _, CF_in _, CF_out _
| CF_out _, CF_out _, CF_in _
| White _, White _, (CF_io _ | CF_aux _)
| White _, (CF_io _ | CF_aux _), White _
```

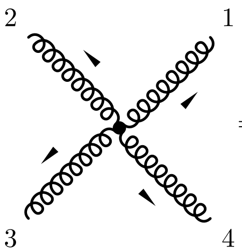
```
| (CF_io _ | CF_aux _), White _, White _

| (White _ | CF_io _ | CF_aux _), CF_in _, CF_in _
| CF_in _, (White _ | CF_io _ | CF_aux _), CF_in _
| CF_in _, CF_in _, (White _ | CF_io _ | CF_aux _)

| (White _ | CF_io _ | CF_aux _), CF_out _, CF_out _
| CF_out _, (White _ | CF_io _ | CF_aux _), CF_out _
| CF_out _, CF_out _, (White _ | CF_io _ | CF_aux _)

| (CF_in _ | CF_out _),
    (White _ | CF_io _ | CF_aux _),
    (White _ | CF_io _ | CF_aux _)
| (White _ | CF_io _ | CF_aux _),
    (CF_in _ | CF_out _),
    (White _ | CF_io _ | CF_aux _)
| (White _ | CF_io _ | CF_aux _),
    (White _ | CF_io _ | CF_aux _),
    (CF_in _ | CF_out _) →
      colored_vertex "colorize_fusion3"

end

| C.SUN nc1 →
    begin match f1, f2, f3 with

    | CF_in (_, c1), CF_io (_, c2, c2'), CF_io (_, c3, c3')
    | CF_io (_, c2, c2'), CF_in (_, c1), CF_io (_, c3, c3')
    | CF_io (_, c2, c2'), CF_io (_, c3, c3'), CF_in (_, c1) →
        if nc1 > 0 then
          if c1 = c2' ∧ c2 = c3' then
            [CF_in (f, c3), v]
          else if c1 = c3' ∧ c3 = c2' then
            [CF_in (f, c2), v]
          else
            []
        else
          colored_vertex "colorize_fusion3"

    | CF_out (_, c1'), CF_io (_, c2, c2'), CF_io (_, c3, c3')
    | CF_io (_, c2, c2'), CF_out (_, c1'), CF_io (_, c3, c3')
    | CF_io (_, c2, c2'), CF_io (_, c3, c3'), CF_out (_, c1') →
        if nc1 < 0 then
          if c1' = c2 ∧ c2' = c3 then
            [CF_out (f, c3'), v]
          else if c1' = c3 ∧ c3' = c2 then
            [CF_out (f, c2'), v]
          else
            []
        else
          colored_vertex "colorize_fusion3"

    | CF_aux _, CF_in (_, c1), CF_io (_, c2, c2')
    | CF_aux _, CF_io (_, c2, c2'), CF_in (_, c1)
```

```
| CF_in (_, c1), CF_aux _, CF_io (_, c2, c2')
| CF_io (_, c2, c2'), CF_aux _, CF_in (_, c1)
| CF_in (_, c1), CF_io (_, c2, c2'), CF_aux _
| CF_io (_, c2, c2'), CF_in (_, c1), CF_aux _ →
    if nc1 > 0 then
      if c1 = c2' then
        [CF_in (f, c2), mult_vertex ( 2) v]
      else
        []
    else
      colored_vertex "colorize_fusion3"
| CF_aux _, CF_out (_, c1'), CF_io (_, c2, c2')
| CF_aux _, CF_io (_, c2, c2'), CF_out (_, c1')
| CF_out (_, c1'), CF_aux _, CF_io (_, c2, c2')
| CF_io (_, c2, c2'), CF_aux _, CF_out (_, c1')
| CF_out (_, c1'), CF_io (_, c2, c2'), CF_aux _
| CF_io (_, c2, c2'), CF_out (_, c1'), CF_aux _ →
    if nc1 < 0 then
      if c1' = c2 then
        [CF_out (f, c2'), mult_vertex ( 2) v]
      else
        []
    else
      colored_vertex "colorize_fusion3"
| White _, CF_in (_, c1), CF_io (_, c2, c2')
| White _, CF_io (_, c2, c2'), CF_in (_, c1)
| CF_in (_, c1), White _, CF_io (_, c2, c2')
| CF_io (_, c2, c2'), White _, CF_in (_, c1)
| CF_in (_, c1), CF_io (_, c2, c2'), White _
| CF_io (_, c2, c2'), CF_in (_, c1), White _ →
    if nc1 > 0 then
      if c1 = c2' then
        [CF_in (f, c2), v]
      else
        []
    else
      colored_vertex "colorize_fusion3"
| White _, CF_out (_, c1'), CF_io (_, c2, c2')
| White _, CF_io (_, c2, c2'), CF_out (_, c1')
| CF_out (_, c1'), White _, CF_io (_, c2, c2')
| CF_io (_, c2, c2'), White _, CF_out (_, c1')
| CF_out (_, c1'), CF_io (_, c2, c2'), White _
| CF_io (_, c2, c2'), CF_out (_, c1'), White _ →
    if nc1 < 0 then
      if c2 = c1' then
        [CF_out (f, c2'), v]
      else
        []
    else
```

```
      colored_vertex "colorize_fusion3"
  | CF_in (_, c1), CF_aux _, CF_aux _
  | CF_aux _, CF_in (_, c1), CF_aux _
  | CF_aux _, CF_aux _, CF_in (_, c1) →
    if nc1 > 0 then
      [CF_in (f, c1), mult_vertex ( 2) v]
    else
      colored_vertex "colorize_fusion3"
  | CF_in (_, c1), CF_aux _, White _
  | CF_in (_, c1), White _, CF_aux _
  | CF_in (_, c1), White _, White _
  | CF_aux _, CF_in (_, c1), White _
  | White _, CF_in (_, c1), CF_aux _
  | White _, CF_in (_, c1), White _
  | CF_aux _, White _, CF_in (_, c1)
  | White _, CF_aux _, CF_in (_, c1)
  | White _, White _, CF_in (_, c1) →
    if nc1 > 0 then
      [CF_in (f, c1), v]
    else
      colored_vertex "colorize_fusion3"
  | CF_out (_, c1'), CF_aux _, CF_aux _
  | CF_aux _, CF_out (_, c1'), CF_aux _
  | CF_aux _, CF_aux _, CF_out (_, c1') →
    if nc1 < 0 then
      [CF_out (f, c1'), mult_vertex ( 2) v]
    else
      colored_vertex "colorize_fusion3"
  | CF_out (_, c1'), CF_aux _, White _
  | CF_out (_, c1'), White _, CF_aux _
  | CF_out (_, c1'), White _, White _
  | CF_aux _, CF_out (_, c1'), White _
  | White _, CF_out (_, c1'), CF_aux _
  | White _, CF_out (_, c1'), White _
  | CF_aux _, White _, CF_out (_, c1')
  | White _, CF_aux _, CF_out (_, c1')
  | White _, White _, CF_out (_, c1') →
    if nc1 < 0 then
      [CF_out (f, c1'), v]
    else
      colored_vertex "colorize_fusion3"
  | CF_in _, CF_in _, CF_out _
  | CF_in _, CF_out _, CF_in _
  | CF_out _, CF_in _, CF_in _ →
    if nc1 > 0 then
      color_flow_ambiguous "colorize_fusion3"
    else
      colored_vertex "colorize_fusion3"
```

```
    | CF_in _, CF_out _, CF_out _
    | CF_out _, CF_in _, CF_out _
    | CF_out _, CF_out _, CF_in _ ->
        if nc1 < 0 then
          color_flow_ambiguous "colorize_fusion3"
        else
          colored_vertex "colorize_fusion3"

    | CF_in _, CF_in _, CF_in _
    | CF_out _, CF_out _, CF_out _

    | (White _ | CF_io _ | CF_aux _),
      (White _ | CF_io _ | CF_aux _),
      (White _ | CF_io _ | CF_aux _)

    | (CF_in _ | CF_out _),
        (CF_in _ | CF_out _),
        (White _ | CF_io _ | CF_aux _)
    | (CF_in _ | CF_out _),
        (White _ | CF_io _ | CF_aux _),
        (CF_in _ | CF_out _)
    | (White _ | CF_io _ | CF_aux _),
        (CF_in _ | CF_out _),
        (CF_in _ | CF_out _) ->
        colored_vertex "colorize_fusion3"

    end

| C.AdjSUN nc ->
    begin match f1, f2, f3 with

    | CF_in (_, c1), CF_out (_, c1'), White _
    | CF_out (_, c1'), CF_in (_, c1), White _
    | CF_in (_, c1), White _, CF_out (_, c1')
    | CF_out (_, c1'), White _, CF_in (_, c1)
    | White _, CF_in (_, c1), CF_out (_, c1')
    | White _, CF_out (_, c1'), CF_in (_, c1) ->
        if c1 ≠ c1' then
          [CF_io (f, c1, c1'), v]
        else
          [CF_aux f, v]

    | CF_in (_, c1), CF_out (_, c1'), CF_aux _
    | CF_out (_, c1'), CF_in (_, c1), CF_aux _
    | CF_in (_, c1), CF_aux _, CF_out (_, c1')
    | CF_out (_, c1'), CF_aux _, CF_in (_, c1)
    | CF_aux _, CF_in (_, c1), CF_out (_, c1')
    | CF_aux _, CF_out (_, c1'), CF_in (_, c1) ->
        if c1 ≠ c1' then
          [CF_io (f, c1, c1'), mult_vertex ( 2) v]
        else
          [CF_aux f, mult_vertex ( 2) v]

    | CF_in (_, c1), CF_out (_, c1'), CF_io (_, c2, c2')
```

```
| CF_out (_, c1'), CF_in (_, c1), CF_io (_, c2, c2')
| CF_in (_, c1), CF_io (_, c2, c2'), CF_out (_, c1')
| CF_out (_, c1'), CF_io (_, c2, c2'), CF_in (_, c1)
| CF_io (_, c2, c2'), CF_in (_, c1), CF_out (_, c1')
| CF_io (_, c2, c2'), CF_out (_, c1'), CF_in (_, c1) →
    if c1 = c2' ∧ c2 = c1' then
      [CF_aux f, mult_vertex ( 2) v]
    else if c1 = c2' then
      [CF_io (f, c2, c1'), v]
    else if c2 = c1' then
      [CF_io (f, c1, c2'), v]
    else
      []
```

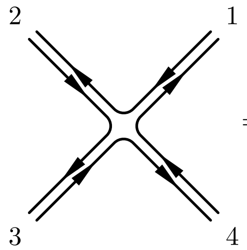


$$
= \begin{aligned}
&-\mathrm{i}g^2 f_{a_1 a_2 b} f_{a_3 a_4 b} (g_{\mu_1\mu_3} g_{\mu_4\mu_2} - g_{\mu_1\mu_4} g_{\mu_2\mu_3}) \\
&-\mathrm{i}g^2 f_{a_1 a_3 b} f_{a_4 a_2 b} (g_{\mu_1\mu_4} g_{\mu_2\mu_3} - g_{\mu_1\mu_2} g_{\mu_3\mu_4}) \\
&-\mathrm{i}g^2 f_{a_1 a_4 b} f_{a_2 a_3 b} (g_{\mu_1\mu_2} g_{\mu_3\mu_4} - g_{\mu_1\mu_3} g_{\mu_4\mu_2})
\end{aligned} \tag{11.5}
$$

Using

$$
\mathcal{P}_4 = \{\{1,2,3,4\},\{1,3,4,2\},\{1,4,2,3\},\{1,2,4,3\},\{1,4,3,2\},\{1,3,2,4\}\} \tag{11.6}
$$

as the set of permutations of $\{1,2,3,4\}$ with the cyclic permutations factored out, we have:



$$
= \mathrm{i}g^2 \sum_{\{\alpha_k\}_{k=1,2,3,4}\in\mathcal{P}_4} \delta^{i_{\alpha_1} j_{\alpha_2}} \delta^{i_{\alpha_2} j_{\alpha_3}} \delta^{i_{\alpha_3} j_{\alpha_4}} \delta^{i_{\alpha_4} j_{\alpha_1}} \\
\left(2 g_{\mu_{\alpha_1}\mu_{\alpha_3}} g_{\mu_{\alpha_4}\mu_{\alpha_2}} - g_{\mu_{\alpha_1}\mu_{\alpha_4}} g_{\mu_{\alpha_2}\mu_{\alpha_3}} - g_{\mu_{\alpha_1}\mu_{\alpha_2}} g_{\mu_{\alpha_3}\mu_{\alpha_4}}\right) \tag{11.7}
$$

The different color connections correspond to permutations of the particles entering the fusion and have to be matched by a corresponding permutation of the Lorentz structure:

We have to generalize this for cases of four particles in the adjoint that are not all gluons:

- scalar-scalar-scalar-scalar
- scalar-scalar-vector-vector

and even ones including fermions (gluinos) if higher dimensional operators are involved.

```
| CF_io (_, c1, c1'), CF_io (_, c2, c2'), CF_io (_, c3, c3') →
```

```
      if c1′ = c2 ∧ c2′ = c3 then
        [CF_io (f, c1, c3′), permute_vertex4 P123 v]
      else if c1′ = c3 ∧ c3′ = c2 then
        [CF_io (f, c1, c2′), permute_vertex4 P132 v]
      else if c2′ = c3 ∧ c3′ = c1 then
        [CF_io (f, c2, c1′), permute_vertex4 P231 v]
      else if c2′ = c1 ∧ c1′ = c3 then
        [CF_io (f, c2, c3′), permute_vertex4 P213 v]
      else if c3′ = c1 ∧ c1′ = c2 then
        [CF_io (f, c3, c2′), permute_vertex4 P312 v]
      else if c3′ = c2 ∧ c2′ = c1 then
        [CF_io (f, c3, c1′), permute_vertex4 P321 v]
      else
        []
  | CF_io _, CF_io _, CF_aux _
  | CF_io _, CF_aux _, CF_io _
  | CF_aux _, CF_io _, CF_io _
  | CF_io _, CF_aux _, CF_aux _
  | CF_aux _, CF_aux _, CF_io _
  | CF_aux _, CF_io _, CF_aux _
  | CF_aux _, CF_aux _, CF_aux _ →
      []
  | CF_io (_, c1, c1′), CF_io (_, c2, c2′), White _
  | CF_io (_, c1, c1′), White _, CF_io (_, c2, c2′)
  | White _, CF_io (_, c1, c1′), CF_io (_, c2, c2′) →
      if c1′ = c2 then
        [CF_io (f, c1, c2′), mult_vertex (−1) v]
      else if c2′ = c1 then
        [CF_io (f, c2, c1′), mult_vertex ( 1) v]
      else
        []
  | CF_io (_, c1, c1′), CF_aux _, White _
  | CF_aux _, CF_io (_, c1, c1′), White _
  | CF_io (_, c1, c1′), White _, CF_aux _
  | CF_aux _, White _, CF_io (_, c1, c1′)
  | White _, CF_io (_, c1, c1′), CF_aux _
  | White _, CF_aux _, CF_io (_, c1, c1′) →
      []
  | CF_aux _, CF_aux _, White _
  | CF_aux _, White _, CF_aux _
  | White _, CF_aux _, CF_aux _ →
      []
  | White _, White _, CF_io (_, c1, c1′)
  | White _, CF_io (_, c1, c1′), White _
  | CF_io (_, c1, c1′), White _, White _ →
      [CF_io (f, c1, c1′), v]
  | White _, White _, CF_aux _
  | White _, CF_aux _, White _
```

```
  | CF_aux _, White _, White _ ->
      []
  | White _, White _, White _
  | (White _ | CF_io _ | CF_aux _),
      (White _ | CF_io _ | CF_aux _),
      (CF_in _ | CF_out _)
  | (White _ | CF_io _ | CF_aux _),
      (CF_in _ | CF_out _),
      (White _ | CF_io _ | CF_aux _)
  | (CF_in _ | CF_out _),
      (White _ | CF_io _ | CF_aux _),
      (White _ | CF_io _ | CF_aux _)
  | CF_in _, CF_in _, (White _ | CF_io _ | CF_aux _)
  | CF_in _, (White _ | CF_io _ | CF_aux _), CF_in _
  | (White _ | CF_io _ | CF_aux _), CF_in _, CF_in _
  | CF_out _, CF_out _, (White _ | CF_io _ | CF_aux _)
  | CF_out _, (White _ | CF_io _ | CF_aux _), CF_out _
  | (White _ | CF_io _ | CF_aux _), CF_out _, CF_out _
  | (CF_in _ | CF_out _),
      (CF_in _ | CF_out _),
      (CF_in _ | CF_out _) ->
      colored_vertex "colorize_fusion3"
  end
```

Quintic and Higher Vertices

```
let is_white = function
  | White _ -> true
  | _ -> false

let colorize_fusionn flist (f, v) =
  let incomplete_match () =
    incomplete
      ("colorize_fusionn { " ^
       String.concat ", " (List.map flavor_to_string flist) ^
       " } -> " ^ M.flavor_to_string f) in
  match M.color f with
  | C.Singlet ->
      if List.for_all is_white flist then
        [White f, v]
      else
        incomplete_match ()
  | C.SUN _ ->
      if List.for_all is_white flist then
        colored_vertex "colorize_fusionn"
      else
        incomplete_match ()
```

```
  | C.AdjSUN _ →
      if List.for_all is_white flist then
        colored_vertex "colorize_fusionn"
      else
        incomplete_match ()

let fuse2 f1 f2 =
  ThoList.flatmap
    (colorize_fusion2 f1 f2)
    (M.fuse2 (flavor_sans_color f1) (flavor_sans_color f2))

let fuse3 f1 f2 f3 =
  ThoList.flatmap
    (colorize_fusion3 f1 f2 f3)
    (M.fuse3 (flavor_sans_color f1) (flavor_sans_color f2) (flavor_sans_color f3))

let fuse_list flist =
  ThoList.flatmap
    (colorize_fusionn flist)
    (M.fuse (List.map flavor_sans_color flist))

let fuse = function
  | [] | [_] → invalid_arg "Colorize.It().fuse"
  | [f1; f2] → fuse2 f1 f2
  | [f1; f2; f3] → fuse3 f1 f2 f3
  | flist → fuse_list flist

let max_degree = M.max_degree
```

Adding Color to External Particles

```
let count_color_strings f_list =
  let rec count_color_strings' n_in n_out n_glue = function
    | f :: rest →
        begin match M.color f with
        | C.Singlet → count_color_strings' n_in n_out n_glue rest
        | C.SUN nc →
            if nc > 0 then
              count_color_strings' (succ n_in) n_out n_glue rest
            else if nc < 0 then
              count_color_strings' n_in (succ n_out) n_glue rest
            else
              su0 "count_color_strings"
        | C.AdjSUN _ →
            count_color_strings' (succ n_in) (succ n_out) (succ n_glue) rest
        end
    | [] → (n_in, n_out, n_glue)
  in
  count_color_strings' 0 0 0 f_list

let external_color_flows f_list =
  let n_in, n_out, n_glue = count_color_strings f_list in
```

```
if n_in ≠ n_out then
  []
else
  let color_strings = ThoList.range 1 n_in in
  List.rev_map
    (fun permutation → (color_strings, permutation))
    (Combinatorics.permute color_strings)
```

If there are only adjoints *and* there are no couplings of adjoints to singlets, we can ignore the U(1)-ghosts.

```
let pure_adjoints f_list =
  List.for_all (fun f → match M.color f with C.AdjSUN _ → true |
_ → false) f_list
```

```
let two_adjoints_couple_to_singlets () =
  let vertices3, vertices4, verticesn = M.vertices () in
  List.exists (fun ((f1, f2, f3), _, _) →
    match M.color f1, M.color f2, M.color f3 with
    | C.AdjSUN _, C.AdjSUN _, C.Singlet
    | C.AdjSUN _, C.Singlet, C.AdjSUN _
    | C.Singlet, C.AdjSUN _, C.AdjSUN _ → true
    | _ → false) vertices3 ∨
  List.exists (fun ((f1, f2, f3, f4), _, _) →
    match M.color f1, M.color f2, M.color f3, M.color f4 with
    | C.AdjSUN _, C.AdjSUN _, C.Singlet, C.Singlet
    | C.AdjSUN _, C.Singlet, C.AdjSUN _, C.Singlet
    | C.Singlet, C.AdjSUN _, C.AdjSUN _, C.Singlet
    | C.AdjSUN _, C.Singlet, C.Singlet, C.AdjSUN _
    | C.Singlet, C.AdjSUN _, C.Singlet, C.AdjSUN _
    | C.Singlet, C.Singlet, C.AdjSUN _, C.AdjSUN _ → true
    | _ → false) vertices4 ∨
  List.exists (fun (flist, _, g) → true) verticesn
```

```
let external_ghosts f_list =
  if pure_adjoints f_list then
    two_adjoints_couple_to_singlets ()
  else
    true
```

We use *List.hd* and *List.tl* instead of pattern matching, because we consume *ecf_in* and *ecf_out* at a different pace.

```
let tail_opt = function
  | [] → []
  | _ :: tail → tail
```

```
let head_req = function
  | [] →
      invalid_arg "Colorize.It().colorize_crossed_amplitude1: insufficient flows"
  | x :: _ → x
```

```
let rec colorize_crossed_amplitude1 ghosts acc f_list (ecf_in, ecf_out) =
  match f_list, ecf_in, ecf_out with
```

```
  | [], [], [] → [List.rev acc]
  | [], _, _ →
      invalid_arg "Colorize.It().colorize_crossed_amplitude1:␣leftover␣flows"
  | f :: rest, _, _ →
      begin match M.color f with
      | C.Singlet →
          colorize_crossed_amplitude1 ghosts
            (White f :: acc)
            rest (ecf_in, ecf_out)
      | C.SUN nc →
          if nc > 0 then
            colorize_crossed_amplitude1 ghosts
              (CF_in (f, head_req ecf_in) :: acc)
              rest (tail_opt ecf_in, ecf_out)
          else if nc < 0 then
            colorize_crossed_amplitude1 ghosts
              (CF_out (f, head_req ecf_out) :: acc)
              rest (ecf_in, tail_opt ecf_out)
          else
            su0 "colorize_flavor"
      | C.AdjSUN _ →
          let ecf_in' = head_req ecf_in
          and ecf_out' = head_req ecf_out in
          if ecf_in' = ecf_out' then begin
            if ghosts then
              colorize_crossed_amplitude1 ghosts
                (CF_aux f :: acc)
                rest (tail_opt ecf_in, tail_opt ecf_out)
            else
              []
          end else
            colorize_crossed_amplitude1 ghosts
              (CF_io (f, ecf_in', ecf_out') :: acc)
              rest (tail_opt ecf_in, tail_opt ecf_out)
      end

let colorize_crossed_amplitude1 ghosts f_list (ecf_in, ecf_out) =
  colorize_crossed_amplitude1 ghosts [] f_list (ecf_in, ecf_out)

let colorize_crossed_amplitude f_list =
  ThoList.rev_flatmap
    (colorize_crossed_amplitude1 (external_ghosts f_list) f_list)
    (external_color_flows f_list)

let cross_uncolored p_in p_out =
  (List.map M.conjugate p_in) @ p_out

let uncross_colored n_in p_lists_colorized =
  let p_in_out_colorized = List.map (ThoList.splitn n_in) p_lists_colorized in
  List.map
    (fun (p_in_colored, p_out_colored) →
      (List.map conjugate p_in_colored, p_out_colored))
```

```
      p_in_out_colorized

  let amplitude p_in p_out =
    uncross_colored
      (List.length p_in)
      (colorize_crossed_amplitude (cross_uncolored p_in p_out))
```

The −-sign in the second component is redundant, but a Whizard convention.

```
  let indices = function
    | White _ → Color.Flow.of_list [0; 0]
    | CF_in (_, c) → Color.Flow.of_list [c; 0]
    | CF_out (_, c) → Color.Flow.of_list [0; − c]
    | CF_io (_, c1, c2) → Color.Flow.of_list [c1; − c2]
    | CF_aux f → Color.Flow.ghost ()

  let flow p_in p_out =
    (List.map indices p_in, List.map indices p_out)

end
```

11.2.2 *Colorizing a Monochrome Gauge Model*

```
module Gauge (M : Model.Gauge) =
  struct

  let rcs = RCS.rename rcs_file "Colorize.Gauge()"
      [ "Colorizing␣Monochrome␣Gauge␣Models"]

  module CM = It(M)

  type flavor = CM.flavor
  type flavor_sans_color = CM.flavor_sans_color
  type gauge = CM.gauge
  type constant = CM.constant
  module Ch = CM.Ch
  let charges = CM.charges
  let flavor_sans_color = CM.flavor_sans_color
  let color = CM.color
  let pdg = CM.pdg
  let lorentz = CM.lorentz
  let propagator = CM.propagator
  let width = CM.width
  let conjugate = CM.conjugate
  let conjugate_sans_color = CM.conjugate_sans_color
  let fermion = CM.fermion
  let max_degree = CM.max_degree
  let vertices = CM.vertices
  let fuse2 = CM.fuse2
  let fuse3 = CM.fuse3
  let fuse = CM.fuse
  let flavors = CM.flavors
  let nc = CM.nc
```

```
let external_flavors = CM.external_flavors
let goldstone = CM.goldstone
let parameters = CM.parameters
let flavor_of_string = CM.flavor_of_string
let flavor_to_string = CM.flavor_to_string
let flavor_to_TeX = CM.flavor_to_TeX
let flavor_symbol = CM.flavor_symbol
let gauge_symbol = CM.gauge_symbol
let mass_symbol = CM.mass_symbol
let width_symbol = CM.width_symbol
let constant_symbol = CM.constant_symbol
let options = CM.options

let incomplete s =
  failwith ("Colorize.Gauge()." ^ s ^ " not done yet!")

type matter_field = M.matter_field
type gauge_boson = M.gauge_boson
type other = M.other

type field =
  | Matter of matter_field
  | Gauge of gauge_boson
  | Other of other

let field f =
  incomplete "field"

let matter_field f =
  incomplete "matter_field"

let gauge_boson f =
  incomplete "gauge_boson"

let other f =
  incomplete "other"

let amplitude = CM.amplitude

let flow = CM.flow

end
```

—12— Processes

12.1 Interface of Process

module type T =
  sig

type *flavor*

Eventually this should become an abstract type:

type t = *flavor list* $\times$ *flavor list*

val *incoming* : t $\rightarrow$ *flavor list*
val *outgoing* : t $\rightarrow$ *flavor list*

parse_decay s decodes a decay description `"a␣->␣b␣c␣..."`, where each word is split into a bag of flavors separated by `':'`s.

type *decay*
val *parse_decay* : *string* $\rightarrow$ *decay*
val *expand_decays* : *decay list* $\rightarrow$ *t list*

parse_scattering s decodes a scattering description `"a␣b␣->␣c␣d␣..."`, where each word is split into a bag of flavors separated by `':'`s.

type *scattering*
val *parse_scattering* : *string* $\rightarrow$ *scattering*
val *expand_scatterings* : *scattering list* $\rightarrow$ *t list*

parse_process s decodes process descriptions

$$\texttt{"a b c d"} \Rightarrow \mathit{Any}\ [a;\ b;\ c;\ d] \tag{12.1a}$$
$$\texttt{"a -> b c d"} \Rightarrow \mathit{Decay}\ (a,\ [b;\ c;\ d]) \tag{12.1b}$$
$$\texttt{"a b -> c d"} \Rightarrow \mathit{Scattering}\ (a,\ b,\ [c;\ d]) \tag{12.1c}$$

where each word is split into a bag of flavors separated by `':'`s.

type *any*
type *process* = *Any* of *any* | *Decay* of *decay* | *Scattering* of *scattering*
val *parse_process* : *string* $\rightarrow$ *process*

remove_duplicate_final_states partition processes removes duplicates from *processes*, which differ only by a permutation of final state particles. The permutation must respect the partitioning given by the offset 1 integers in *partition*.

```
val remove_duplicate_final_states : int list list → t list → t list
```

diff set1 set2 returns the processes in *set1* with the processes in *set2* removed. *set2* does not need to be a subset of *set1*.

```
val diff : t list → t list → t list
```

Not functional yet. Interface subject to change. Should be moved to *Fusion.Multi*, because we will want to cross *colored* matrix elements.

Factor amplitudes that are related by crossing symmetry.

```
  val crossing : t list → (flavor list × int list × t) list
end
```

```
module Make (M : Model.T) : T with type flavor = M.flavor
```

12.2 Implementation of Process

```
module type T =
  sig
    type flavor
    type t = flavor list × flavor list
    val incoming : t → flavor list
    val outgoing : t → flavor list
    type decay
    val parse_decay : string → decay
    val expand_decays : decay list → t list
    type scattering
    val parse_scattering : string → scattering
    val expand_scatterings : scattering list → t list
    type any
    type process = Any of any | Decay of decay | Scattering of scattering
    val parse_process : string → process
    val remove_duplicate_final_states : int list list → t list → t list
    val diff : t list → t list → t list
    val crossing : t list → (flavor list × int list × t) list
  end
```

```
module Make (M : Model.T) =
  struct

    type flavor = M.flavor

    type t = flavor list × flavor list

    let incoming (fin, _ ) = fin
    let outgoing (_, fout) = fout
```

12.2.1 Select Charge Conserving Processes

```
let allowed (fin, fout) =
  M.Ch.is_null (M.Ch.sum (List.map M.charges (List.map M.conjugate fin @ fout)))
```

12.2.2 Parsing Process Descriptions

```
type α bag = α list

type any = flavor bag list
type decay = flavor bag × flavor bag list
type scattering = flavor bag × flavor bag × flavor bag list

type process =
  | Any of any
  | Decay of decay
  | Scattering of scattering

let unique_flavors f_bags =
  List.for_all (function [f] → true | _ → false) f_bags

let unique_final_state = function
  | Any fs → unique_flavors fs
  | Decay (_, fs) → unique_flavors fs
  | Scattering (_, _, fs) → unique_flavors fs

let parse_process process =
  let last = String.length process − 1
  and flavor off len = M.flavor_of_string (String.sub process off len) in

  let add_flavors flavors = function
    | Any l → Any (List.rev flavors :: l)
    | Decay (i, f) → Decay (i, List.rev flavors :: f)
    | Scattering (i1, i2, f) → Scattering (i1, i2, List.rev flavors :: f) in

  let rec scan_list so_far n =
    if n > last then
      so_far
    else
      let n' = succ n in
      match process.[n] with
      | ' ' | '\n' → scan_list so_far n'
      | '-' → scan_gtr so_far n'
      | c → scan_flavors so_far [] n n'

  and scan_flavors so_far flavors w n =
    if n > last then
      add_flavors (flavor w (last − w + 1) :: flavors) so_far
    else
      let n' = succ n in
      match process.[n] with
      | ' ' | '\n' →
          scan_list (add_flavors (flavor w (n − w) :: flavors) so_far) n'
```

```
        | ':' → scan_flavors so_far (flavor w (n − w) :: flavors) n' n'
        | _ → scan_flavors so_far flavors w n'

  and scan_gtr so_far n =
    if n > last then
      invalid_arg "expecting␣‘>’"
    else
      let n' = succ n in
      match process.[n] with
      | '>' →
          begin match so_far with
          | Any [i] → scan_list (Decay (i, [])) n'
          | Any [i2; i1] → scan_list (Scattering (i1, i2, [])) n'
          | Any _ → invalid_arg "only␣1␣or␣2␣particles␣in␣|in>"
          | _ → invalid_arg "too␣many␣‘->’s"
          end
      | _ → invalid_arg "expecting␣‘>’" in

  match scan_list (Any []) 0 with
  | Any l → Any (List.rev l)
  | Decay (i, f) → Decay (i, List.rev f)
  | Scattering (i1, i2, f) → Scattering (i1, i2, List.rev f)

let parse_decay process =
  match parse_process process with
  | Any (i :: f) →
      prerr_endline "missing␣‘->’␣in␣process␣description,␣assuming␣decay.";
      (i, f)
  | Decay (i, f) → (i, f)
  | _ → invalid_arg "expecting␣decay␣description:␣got␣scattering"

let parse_scattering process =
  match parse_process process with
  | Any (i1 :: i2 :: f) →
      prerr_endline "missing␣‘->’␣in␣process␣description,␣assuming␣scattering.";
      (i1, i2, f)
  | Scattering (i1, i2, f) → (i1, i2, f)
  | _ → invalid_arg "expecting␣scattering␣description:␣got␣decay"

let expand_scatterings scatterings =
  ThoList.flatmap
    (function (fin1, fin2, fout) →
      Product.fold
        (fun flist acc →
          match flist with
          | fin1' :: fin2' :: fout' →
              let fin_fout' = ([fin1'; fin2'], fout') in
              if allowed fin_fout' then
                fin_fout' :: acc
              else
                acc
          | [_] | [] → failwith "Omega.expand_scatterings:␣can’t␣happen")
        (fin1 :: fin2 :: fout) []) scatterings
```

```
let expand_decays decays =
  ThoList.flatmap
    (function (fin, fout) →
      Product.fold
        (fun flist acc →
          match flist with
          | fin' :: fout' →
              let fin_fout' = ([fin'], fout') in
              if allowed fin_fout' then
                fin_fout' :: acc
              else
                acc
          | [] → failwith "Omega.expand_decays: can't happen")
        (fin :: fout) []) decays
```

12.2.3 Remove Duplicate Final States

Test if all final states are the same. Identical to *ThoList.homogeneous* ∘ (*List.map snd*).

```
let rec homogeneous_final_state = function
  | [] | [_] → true
  | (_, fs1) :: ((_, fs2) :: _ as rest) →
      if fs1 ≠ fs2 then
        false
      else
        homogeneous_final_state rest

let by_color f1 f2 =
  let c = Color.compare (M.color f1) (M.color f2) in
  if c ≠ 0 then
    c
  else
    compare f1 f2

module Pre_Bundle =
  struct

    type elt = t
    type base = elt

    let compare_elt (fin1, fout1) (fin2, fout2) =
      let c = ThoList.compare ~cmp:by_color fin1 fin2 in
      if c ≠ 0 then
        c
      else
        ThoList.compare ~cmp:by_color fout1 fout2

    let compare_base b1 b2 = compare_elt b2 b1

  end

module Process_Bundle = Bundle.Dyn (Pre_Bundle)

let to_string (fin, fout) =
```

```
    String.concat " " (List.map M.flavor_to_string fin)
    ^ " -> " ^ String.concat " " (List.map M.flavor_to_string fout)

let fiber_to_string (base, fiber) =
  (to_string base) ^ " -> [" ^
  (String.concat ", " (List.map to_string fiber)) ^ "]"

let bundle_to_strings list =
  List.map fiber_to_string list
```

Subtract $n+1$ from each element in *index_set* and drop all negative numbers from the result.

```
let shift_left_pred' n index_set =
  List.fold_right
    (fun i acc → let i' = i − n − 1 in if i' < 0 then acc else i' :: acc)
    index_set []
```

Convert 1-based indices for initial and final state to 0-based indices for the final state only. (NB: *ThoList.partitioned_sort* expects 0-based indices.)

```
let shift_left_pred fin index_sets =
  let n = match fin with [_] → 1 | [_; _] → 2 | _ → 0 in
  List.fold_right
    (fun iset acc →
      match shift_left_pred' n iset with
      | [] → acc
      | iset' → iset' :: acc)
    index_sets []

module FSet = Set.Make (struct type t = flavor let compare = compare end)
```

Take a list of final states and return a list of sets of flavors appearing in each slot.

```
let flavors = function
  | [] → []
  | fs :: fs_list →
      List.fold_right (List.map2 FSet.add) fs_list (List.map FSet.singleton fs)

let flavor_sums flavor_sets =
  let _, result =
    List.fold_left
      (fun (n, acc) flavors →
        if FSet.cardinal flavors = 1 then
          (succ n, acc)
        else
          (succ n, (n, flavors) :: acc))
      (0, []) flavor_sets in
  List.rev result

let overlapping s1 s2 =
  ¬ (FSet.is_empty (FSet.inter s1 s2))

let rec merge_overlapping (n, flavors) = function
  | [] → [([n], flavors)]
```

```
  | (n_list, flavor_set) :: rest →
      if overlapping flavors flavor_set then
        (n :: n_list, FSet.union flavors flavor_set) :: rest
      else
        (n_list, flavor_set) :: merge_overlapping (n, flavors) rest

let overlapping_flavor_sums flavor_sums =
  List.rev_map
    (fun (n_list, flavor_set) → (n_list, FSet.elements flavor_set))
    (List.fold_right merge_overlapping flavor_sums [])

module ISet = Set.Make (struct type t = int let compare = compare end)

let integer_range n1 n2 =
  let rec integer_range' acc n' =
    if n' < n1 then
      acc
    else
      integer_range' (ISet.add n' acc) (pred n') in
  integer_range' ISet.empty n2

let coarsest_partition = function
  | [] → invalid_arg "coarsest_partition: empty process list"
  | ((_, fs) :: _) as proc_list →
      let fs_list = List.map snd proc_list in
      let overlaps =
        List.map fst (overlapping_flavor_sums (flavor_sums (flavors fs_list))) in
      let singletons =
        ISet.elements
          (List.fold_right ISet.remove
             (List.concat overlaps) (integer_range 0 (pred (List.length fs)))) in
      List.map (fun n → [n]) singletons @ overlaps

module IPowSet =
  PowSet.Make (struct type t = int let compare = compare let to_string = string_of_int end)

let merge_partitions p_list =
  IPowSet.to_lists (IPowSet.basis (IPowSet.union (List.map IPowSet.of_lists p_list)))

let remove_duplicate_final_states cascade_partition = function
  | [] → []
  | [process] → [process]
  | list →
      if homogeneous_final_state list then
        list
      else
        let partition = coarsest_partition list in
        let pi (fin, fout) =
          let partition' =
            merge_partitions [partition; shift_left_pred fin cascade_partition] in
          (fin, ThoList.partitioned_sort by_color partition' fout) in
        Process_Bundle.base (Process_Bundle.of_list pi list)

type t' = t
```

```
module PSet = Set.Make (struct type t = t' let compare = compare end)

let set list =
  List.fold_right PSet.add list PSet.empty

let diff list1 list2 =
  PSet.elements (PSet.diff (set list1) (set list2))
```

Not functional yet.

```
module Crossing_Projection =
  struct

    type elt = t
    type base = flavor list × int list × t

    let compare_elt (fin1, fout1) (fin2, fout2) =
      let c = ThoList.compare ~cmp : by_color fin1 fin2 in
      if c ≠ 0 then
        c
      else
        ThoList.compare ~cmp : by_color fout1 fout2

    let compare_base (f1, _, _) (f2, _, _) =
      ThoList.compare ~cmp : by_color f1 f2

    let pi (fin, fout as process) =
      let flist, indices =
        ThoList.ariadne_sort ~cmp : by_color (List.map M.conjugate fin @ fout) in
      (flist, indices, process)

  end

module Crossing_Bundle = Bundle.Make (Crossing_Projection)

let crossing processes =
  List.map
    (fun (fin, fout as process) →
      (List.map M.conjugate fin @ fout, [], process))
    processes

end
```

—13—
Hardcoded Models

13.1 Interface of Modeltools

13.1.1 Compilation

module type $Flavor$ =
  sig
    type f
    type c
    val $compare$: $f \to f \to int$
    val $conjugate$: $f \to f$
  end

module type $Fusions$ =
  sig
    type t
    type f
    type c
    val $fuse2$: $t \to f \to f \to (f \times c\ Coupling.t)\ list$
    val $fuse3$: $t \to f \to f \to f \to (f \times c\ Coupling.t)\ list$
    val $fuse$: $t \to f\ list \to (f \times c\ Coupling.t)\ list$
    val $of\_vertices$:
        $(((f \times f \times f) \times c\ Coupling.vertex3 \times c)\ list$
          $\times ((f \times f \times f \times f) \times c\ Coupling.vertex4 \times c)\ list$
          $\times (f\ list \times c\ Coupling.vertexn \times c)\ list) \to t$
  end

module $Fusions$: functor $(F : Flavor) \to$
  $Fusions$ with type $f = F.f$ and type $c = F.c$

13.1.2 Mutable Models

module $Mutable$: functor (FGC : sig type f and g and c end) $\to$
  $Model.Mutable$ with type $flavor = FGC.f$ and type $gauge = FGC.g$
  and type $constant = FGC.c$

13.2 Implementation of Modeltools

```
let rcs_file = RCS.parse "Modeltools" ["Lagragians"]
    { RCS.revision = "$Revision: 6465 $";
      RCS.date = "$Date: 2015-01-10 16:22:31 +0100 (Sat, 10 Jan 2015) $";
      RCS.author = "$Author: jr_reuter $";
      RCS.source
        = "$URL: svn+ssh://jr_reuter@login.hepforge.org/hepforge/svn/whizard/trunk/omega/sr
```

13.2.1 Compilation

Flavors and coupling constants: flavors can be tested for equality and charge conjugation is defined.

```
module type Flavor =
  sig
    type f
    type c
    val compare : f → f → int
    val conjugate : f → f
  end
```

Compiling fusions from a list of vertices:

```
module type Fusions =
  sig
    type t
    type f
    type c
    val fuse2 : t → f → f → (f × c Coupling.t) list
    val fuse3 : t → f → f → f → (f × c Coupling.t) list
    val fuse : t → f list → (f × c Coupling.t) list
    val of_vertices :
        (((f × f × f) × c Coupling.vertex3 × c) list
            × ((f × f × f × f) × c Coupling.vertex4 × c) list
            × (f list × c Coupling.vertexn × c) list) → t
  end

module Fusions (F : Flavor) : Fusions with type f = F.f and type c = F.c =
  struct

    type f = F.f
    type c = F.c

    module F2 =
      struct
        type t = f × f
        let hash = Hashtbl.hash
        let compare (f1, f2) (f1′, f2′) =
          let c1 = F.compare f1 f1′ in
          if c1 ≠ 0 then
            c1
```

```
        else
          F.compare f2 f2'
      let equal f f' = compare f f' = 0
    end

  module F3 =
    struct
      type t = f × f × f
      let hash = Hashtbl.hash
      let compare (f1, f2, f3) (f1', f2', f3') =
        let c1 = F.compare f1 f1' in
        if c1 ≠ 0 then
          c1
        else
          let c2 = F.compare f2 f2' in
          if c2 ≠ 0 then
            c2
          else
            F.compare f3 f3'
      let equal f f' = compare f f' = 0
    end

  module Fn =
    struct
      type t = f list
      let hash = Hashtbl.hash
      let compare f f' = ThoList.compare ~cmp : F.compare f f'
      let equal f f' = compare f f' = 0
    end

  module H2 = Hashtbl.Make (F2)
  module H3 = Hashtbl.Make (F3)
  module Hn = Hashtbl.Make (Fn)

  type t =
      { v3 : (f × c Coupling.t) list H2.t;
        v4 : (f × c Coupling.t) list H3.t;
        vn : (f × c Coupling.t) list Hn.t }

  let fuse2 table f1 f2 =
    try
      H2.find table.v3 (f1, f2)
    with
    | Not_found → []

  let fuse3 table f1 f2 f3 =
    try
      H3.find table.v4 (f1, f2, f3)
    with
    | Not_found → []

  let fusen table f =
    try
      Hn.find table.vn f
```

```
  with
  | Not_found → []

let fuse table = function
  | [] | [_] → invalid_arg "Fusions().fuse"
  | [f1; f2] → fuse2 table f1 f2
  | [f1; f2; f3] → fuse3 table f1 f2 f3
  | f → fusen table f
```

Note that a pair or a triplet can appear more than once (e. g. $e^+e^- \to \gamma$ and $e^+e^- \to Z$). Therefore don't replace the entry, but augment it instead.

```
let add_fusion2 table f1 f2 fusions =
  H2.add table.v3 (f1, f2) (fusions :: fuse2 table f1 f2)

let add_fusion3 table f1 f2 f3 fusions =
  H3.add table.v4 (f1, f2, f3) (fusions :: fuse3 table f1 f2 f3)

let add_fusionn table f fusions =
  Hn.add table.vn f (fusions :: fusen table f)
```

Do we need to take into account the charge conjugation of the coupling constants here?

If some flavors are identical, we must not introduce the same vertex more than once:

```
open Coupling

let permute3 (f1, f2, f3) =
  [ (f1, f2), F.conjugate f3, F12;
    (f2, f1), F.conjugate f3, F21;
    (f2, f3), F.conjugate f1, F23;
    (f3, f2), F.conjugate f1, F32;
    (f3, f1), F.conjugate f2, F31;
    (f1, f3), F.conjugate f2, F13 ]
```

Here we add identical permutations of pairs only once:

```
module F2' = Set.Make (F2)

let add_permute3 table v c set ((f1, f2 as f12), f, p) =
  if F2'.mem f12 set then
    set
  else begin
    add_fusion2 table f1 f2 (f, V3 (v, p, c));
    F2'.add f12 set
  end

let add_vertex3 table (f123, v, c) =
  ignore (List.fold_left (fun set f → add_permute3 table v c set f)
            F2'.empty (permute3 f123))
```

Handling all the cases explicitely is OK for cubic vertices, but starts to become questionable already for quartic couplings. The advantage remains that we can check completeness in *Targets*.

```
let permute4 (f1, f2, f3, f4) =
  [ (f1, f2, f3), F.conjugate f4, F123;
    (f2, f3, f1), F.conjugate f4, F231;
    (f3, f1, f2), F.conjugate f4, F312;
    (f2, f1, f3), F.conjugate f4, F213;
    (f3, f2, f1), F.conjugate f4, F321;
    (f1, f3, f2), F.conjugate f4, F132;
    (f1, f2, f4), F.conjugate f3, F124;
    (f2, f4, f1), F.conjugate f3, F241;
    (f4, f1, f2), F.conjugate f3, F412;
    (f2, f1, f4), F.conjugate f3, F214;
    (f4, f2, f1), F.conjugate f3, F421;
    (f1, f4, f2), F.conjugate f3, F142;
    (f1, f3, f4), F.conjugate f2, F134;
    (f3, f4, f1), F.conjugate f2, F341;
    (f4, f1, f3), F.conjugate f2, F413;
    (f3, f1, f4), F.conjugate f2, F314;
    (f4, f3, f1), F.conjugate f2, F431;
    (f1, f4, f3), F.conjugate f2, F143;
    (f2, f3, f4), F.conjugate f1, F234;
    (f3, f4, f2), F.conjugate f1, F342;
    (f4, f2, f3), F.conjugate f1, F423;
    (f3, f2, f4), F.conjugate f1, F324;
    (f4, f3, f2), F.conjugate f1, F432;
    (f2, f4, f3), F.conjugate f1, F243 ]
```

Add identical permutations of triplets only once:

```
module F3' = Set.Make (F3)

let add_permute4 table v c set ((f1, f2, f3 as f123), f, p) =
  if F3'.mem f123 set then
    set
  else begin
    add_fusion3 table f1 f2 f3 (f, V4 (v, p, c));
    F3'.add f123 set
  end

let add_vertex4 table (f1234, v, c) =
  ignore (List.fold_left (fun set f -> add_permute4 table v c set f)
            F3'.empty (permute4 f1234))

let of_vertices (vlist3, vlist4, vlistn) =
  match vlistn with
  | [] ->
      let table =
        { v3 = H2.create 37; v4 = H3.create 37; vn = Hn.create 37 } in
      List.iter (add_vertex3 table) vlist3;
      List.iter (add_vertex4 table) vlist4;
      table
  | _ -> failwith "Models.Fusions.of_vertices: incomplete"

end
```

13.2.2 Mutable Models

```
module Mutable (FGC : sig type f and g and c end) =
  struct
    type flavor = FGC.f
    type gauge = FGC.g
    type constant = FGC.c

    let options = Options.empty

    module Ch = Charges.Null
    let charges _ = ()

    exception Uninitialized of string
    let unitialized name =
      raise (Uninitialized name)
```

Note that *lookup* works, by the magic of currying, for any arity. But we need to supply one argument to delay evaluation.
Also note that the references are *not* shared among results of functor applications. Simple module renaming causes sharing.

```
    let declare template =
      let reference = ref template in
      let update fct = reference := fct
      and lookup arg = !reference arg in
      (update, lookup)

    let set_color, color =
      declare (fun f → unitialized "color")
    let set_pdg, pdg =
      declare (fun f → unitialized "pdg")
    let set_lorentz, lorentz =
      declare (fun f → unitialized "lorentz")
    let set_propagator, propagator =
      declare (fun f → unitialized "propagator")
    let set_width, width =
      declare (fun f → unitialized "width")
    let set_goldstone, goldstone =
      declare (fun f → unitialized "goldstone")
    let set_conjugate, conjugate =
      declare (fun f → unitialized "conjugate")
    let set_fermion, fermion =
      declare (fun f → unitialized "fermion")
    let set_max_degree, max_degree =
      declare (fun () → unitialized "max_degree")
    let set_vertices, vertices =
      declare (fun () → unitialized "vertices")
    let set_fuse2, fuse2 =
      declare (fun f1 f2 → unitialized "fuse2")
    let set_fuse3, fuse3 =
      declare (fun f1 f2 f3 → unitialized "fuse3")
    let set_fuse, fuse =
```

```
    declare (fun f → unitialized "fuse")
  let set_flavors, flavors =
    declare (fun () → [])
  let set_external_flavors, external_flavors =
    declare (fun () → [("unitialized", [])])
  let set_parameters, parameters =
    declare (fun f → unitialized "parameters")
  let set_flavor_of_string, flavor_of_string =
    declare (fun f → unitialized "flavor_of_string")
  let set_flavor_to_string, flavor_to_string =
    declare (fun f → unitialized "flavor_to_string")
  let set_flavor_to_TeX, flavor_to_TeX =
    declare (fun f → unitialized "flavor_to_TeX")
  let set_flavor_symbol, flavor_symbol =
    declare (fun f → unitialized "flavor_symbol")
  let set_gauge_symbol, gauge_symbol =
    declare (fun f → unitialized "gauge_symbol")
  let set_mass_symbol, mass_symbol =
    declare (fun f → unitialized "mass_symbol")
  let set_width_symbol, width_symbol =
    declare (fun f → unitialized "width_symbol")
  let set_constant_symbol, constant_symbol =
    declare (fun f → unitialized "constant_symbol")

  let setup ~color ~pdg ~lorentz ~propagator ~width ~goldstone
      ~conjugate ~fermion ~max_degree ~vertices
      ~fuse : (fuse2, fuse3, fusen)
      ~flavors ~parameters ~flavor_of_string ~flavor_to_string
      ~flavor_to_TeX ~flavor_symbol
      ~gauge_symbol ~mass_symbol ~width_symbol ~constant_symbol =
    set_color color;
    set_pdg pdg;
    set_lorentz lorentz;
    set_propagator propagator;
    set_width width;
    set_goldstone goldstone;
    set_conjugate conjugate;
    set_fermion fermion;
    set_max_degree (fun () → max_degree);
    set_vertices vertices;
    set_fuse2 fuse2;
    set_fuse3 fuse3;
    set_fuse fusen;
    set_external_flavors (fun f → flavors);
    let flavors = ThoList.flatmap snd flavors in
    set_flavors (fun f → flavors);
    set_parameters parameters;
    set_flavor_of_string flavor_of_string;
    set_flavor_to_string flavor_to_string;
    set_flavor_to_TeX flavor_to_TeX;
```

```
      set_flavor_symbol flavor_symbol;
      set_gauge_symbol gauge_symbol;
      set_mass_symbol mass_symbol;
      set_width_symbol width_symbol;
      set_constant_symbol constant_symbol

    let rcs = RCS.rename rcs_file "Models.Mutable" ["Mutable␣Model"]
  end
```

13.3 Interface of Modellib_SM

13.3.1 Hardcoded Models

```
module Phi3 : Model.T with module Ch = Charges.Null
module Phi4 : Model.T with module Ch = Charges.Null
module QED : Model.T with module Ch = Charges.ZZ
module QCD : Model.T with module Ch = Charges.ZZ

module type SM_flags =
  sig
    val higgs_triangle : bool (* Hγγ, Hgγ and Hggcouplings*)
    val higgs_hmm : bool
    val triple_anom : bool
    val quartic_anom : bool
    val higgs_anom : bool
    val k_matrix : bool
    val ckm_present : bool
    val top_anom : bool
    val top_anom_4f : bool
    val tt_threshold : bool
  end

module SM_no_anomalous : SM_flags
module SM_anomalous : SM_flags
module SM_k_matrix : SM_flags
module SM_no_anomalous_ckm : SM_flags
module SM_anomalous_ckm : SM_flags
module SM_Higgs : SM_flags
module SM_anomalous_top : SM_flags
module SM_tt_threshold : SM_flags

module SM : functor (F : SM_flags) → Model.Gauge with module Ch = Charges.QQ

module SM_Rxi : Model.T with module Ch = Charges.QQ

module Groves : functor (M : Model.Gauge) → Model.Gauge with module Ch = M.Ch
module SM_clones : Model.Gauge with module Ch = Charges.QQ
```

13.4 Implementation of Modellib_SM

```
let rcs_file = RCS.parse "Modellib_SM" ["Lagragians"]
```

```
{ RCS.revision = "$Revision: 6465 $";
  RCS.date = "$Date: 2015-01-10 16:22:31 +0100 (Sat, 10 Jan 2015) $";
  RCS.author = "$Author: jr_reuter $";
  RCS.source
    = "$URL: svn+ssh://jr_reuter@login.hepforge.org/hepforge/svn/whizard/trunk/omega/sr
```

13.4.1 ϕ^3

```
module Phi3 =
  struct
    let rcs = RCS.rename rcs_file "Modellib.Phi3"
        ["phi**3 with a single flavor"]

    open Coupling

    let options = Options.empty

    type flavor = Phi
    let external_flavors () = [ "", [Phi]]
    let flavors () = ThoList.flatmap snd (external_flavors ())

    type gauge = unit
    type constant = G

    type orders = unit
    let orders = function
      | _ → ()

    let lorentz _ = Scalar
    let color _ = Color.Singlet
    let propagator _ = Prop_Scalar
    let width _ = Timelike
    let goldstone _ = None
    let conjugate f = f
    let fermion _ = 0

    module Ch = Charges.Null
    let charges _ = ()

    module F = Modeltools.Fusions (struct
      type f = flavor
      type c = constant
      let compare = compare
      let conjugate = conjugate
    end)

    let vertices () =
      ([(Phi, Phi, Phi), Scalar_Scalar_Scalar 1, G], [], [])

    let table = F.of_vertices (vertices ())
    let fuse2 = F.fuse2 table
    let fuse3 = F.fuse3 table
    let fuse = F.fuse table
    let max_degree () = 3
```

```
let parameters () = { input = [G, 1.0]; derived = []; derived_arrays = [] }

let flavor_of_string = function
  | "p" → Phi
  | _ → invalid_arg "Modellib.Phi3.flavor_of_string"

let flavor_to_string Phi = "phi"
let flavor_to_TeX Phi = "\\phi"
let flavor_symbol Phi = "phi"

let gauge_symbol () =
  failwith "Modellib.Phi3.gauge_symbol:␣internal␣error"

let pdg _ = 1
let mass_symbol _ = "m"
let width_symbol _ = "w"
let constant_symbol G = "g"

end
```

13.4.2 $\lambda_3\phi^3 + \lambda_4\phi^4$

```
module Phi4 =
  struct
    let rcs = RCS.rename rcs_file "Modellib.Phi4"
        ["phi**4␣with␣a␣single␣flavor"]

    open Coupling

    let options = Options.empty

    type flavor = Phi
    let external_flavors () = [ "", [Phi]]
    let flavors () = ThoList.flatmap snd (external_flavors ())

    type gauge = unit
    type constant = G3 | G4

    type orders = unit
    let orders = function
      | _ → ()

    let lorentz _ = Scalar
    let color _ = Color.Singlet
    let propagator _ = Prop_Scalar
    let width _ = Timelike
    let goldstone _ = None
    let conjugate f = f
    let fermion _ = 0

    module Ch = Charges.Null
    let charges _ = ()

    module F = Modeltools.Fusions (struct
      type f = flavor
      type c = constant
```

```
      let compare = compare
      let conjugate = conjugate
    end)

    let vertices () =
      ([(Phi, Phi, Phi), Scalar_Scalar_Scalar 1, G3],
       [(Phi, Phi, Phi, Phi), Scalar4 1, G4], [])

    let fuse2 _ = failwith "Modellib.Phi4.fuse2"
    let fuse3 _ = failwith "Modellib.Phi4.fuse3"
    let fuse = function
      | [] | [_] → invalid_arg "Modellib.Phi4.fuse"
      | [_; _] → [Phi, V3 (Scalar_Scalar_Scalar 1, F23, G3)]
      | [_; _; _] → [Phi, V4 (Scalar4 1, F234, G4)]
      | _ → []
    let max_degree () = 4
    let parameters () =
      { input = [G3, 1.0; G4, 1.0]; derived = []; derived_arrays = [] }

    let flavor_of_string = function
      | "p" → Phi
      | _ → invalid_arg "Modellib.Phi4.flavor_of_string"

    let flavor_to_string Phi = "phi"
    let flavor_to_TeX Phi = "\\phi"
    let flavor_symbol Phi = "phi"

    let gauge_symbol () =
      failwith "Modellib.Phi4.gauge_symbol: internal error"

    let pdg _ = 1
    let mass_symbol _ = "m"
    let width_symbol _ = "w"
    let constant_symbol = function
      | G3 → "g3"
      | G4 → "g4"

  end
```

13.4.3 Quantum Electro Dynamics

```
module QED =
  struct
    let rcs = RCS.rename rcs_file "Modellib.QED"
        ["QED with two leptonic flavors"]

    open Coupling

    let options = Options.empty

    type flavor =
      | Electron | Positron
      | Muon | AntiMuon
      | Tau | AntiTau
```

```
  | Photon

let external_flavors () =
  [ "Leptons", [Electron; Positron; Muon; AntiMuon; Tau; AntiTau];
    "Gauge␣Bosons", [Photon] ]
let flavors () = ThoList.flatmap snd (external_flavors ())

type gauge = unit
type constant = Q

type orders = unit
let orders = function
  | _ → ()

let lorentz = function
  | Electron | Muon | Tau → Spinor
  | Positron | AntiMuon | AntiTau → ConjSpinor
  | Photon → Vector

let color _ = Color.Singlet

let propagator = function
  | Electron | Muon | Tau → Prop_Spinor
  | Positron | AntiMuon | AntiTau → Prop_ConjSpinor
  | Photon → Prop_Feynman

let width _ = Timelike

let goldstone _ =
  None

let conjugate = function
  | Electron → Positron | Positron → Electron
  | Muon → AntiMuon | AntiMuon → Muon
  | Tau → AntiTau | AntiTau → Tau
  | Photon → Photon

let fermion = function
  | Electron | Muon | Tau → 1
  | Positron | AntiMuon | AntiTau → − 1
  | Photon → 0
```

Taking generation numbers makes electric charge redundant.

```
module Ch = Charges.ZZ
let charges = function
  | Electron → [1; 0; 0]
  | Muon → [0; 1; 0]
  | Tau → [0; 0; 1]
  | Positron → [−1; 0; 0]
  | AntiMuon → [0; −1; 0]
  | AntiTau → [0; 0; −1]
  | Photon → [0; 0; 0]

module F = Modeltools.Fusions (struct
  type f = flavor
  type c = constant
```

```
  let compare = compare
  let conjugate = conjugate
end)

let vertices () =
  ([(Positron, Photon, Electron), FBF (1, Psibar, V, Psi), Q;
    (AntiMuon, Photon, Muon), FBF (1, Psibar, V, Psi), Q;
    (AntiTau, Photon, Tau), FBF (1, Psibar, V, Psi), Q], [], [])

let table = F.of_vertices (vertices ())
let fuse2 = F.fuse2 table
let fuse3 = F.fuse3 table
let fuse = F.fuse table
let max_degree () = 3

let parameters () = { input = [Q, 1.0]; derived = []; derived_arrays = [] }

let flavor_of_string = function
  | "e-" → Electron | "e+" → Positron
  | "m-" → Muon | "m+" → AntiMuon
  | "t-" → Tau | "t+" → AntiTau
  | "A" → Photon
  | _ → invalid_arg "Modellib.QED.flavor_of_string"

let flavor_to_string = function
  | Electron → "e-" | Positron → "e+"
  | Muon → "m-" | AntiMuon → "m+"
  | Tau → "t-" | AntiTau → "t+"
  | Photon → "A"

let flavor_to_TeX = function
  | Electron → "e^-" | Positron → "e^+"
  | Muon → "\\mu^-" | AntiMuon → "\\mu^+"
  | Tau → "^\\tau^-" | AntiTau → "\\tau+^"
  | Photon → "\\gamma"

let flavor_symbol = function
  | Electron → "ele" | Positron → "pos"
  | Muon → "muo" | AntiMuon → "amu"
  | Tau → "tau" | AntiTau → "ata"
  | Photon → "gam"

let gauge_symbol () =
  failwith "Modellib.QED.gauge_symbol:␣internal␣error"

let pdg = function
  | Electron → 11 | Positron → −11
  | Muon → 13 | AntiMuon → −13
  | Tau → 15 | AntiTau → −15
  | Photon → 22

let mass_symbol f =
  "mass(" ^ string_of_int (abs (pdg f)) ^ ")"

let width_symbol f =
  "width(" ^ string_of_int (abs (pdg f)) ^ ")"
```

```
    let constant_symbol = function
      | Q → "qlep"
  end
```

13.4.4 Quantum Chromo Dynamics

```
module QCD =
  struct
    let rcs = RCS.rename rcs_file "Modellib.QCD"
        ["QCD"]

    open Coupling

    let options = Options.empty

    type flavor =
      | U | Ubar | D | Dbar
      | C | Cbar | S | Sbar
      | T | Tbar | B | Bbar
      | Gl

    let external_flavors () =
      [ "Quarks", [U; D; C; S; T; B; Ubar; Dbar; Cbar; Sbar; Tbar; Bbar];
        "Gauge Bosons", [Gl]]
    let flavors () = ThoList.flatmap snd (external_flavors ())

    type gauge = unit
    type constant = Gs | G2 | I_Gs

    type orders = unit
    let orders = function
      | _ → ()

    let lorentz = function
      | U | D | C | S | T | B → Spinor
      | Ubar | Dbar | Cbar | Sbar | Tbar | Bbar → ConjSpinor
      | Gl → Vector

    let color = function
      | U | D | C | S | T | B → Color.SUN 3
      | Ubar | Dbar | Cbar | Sbar | Tbar | Bbar → Color.SUN (−3)
      | Gl → Color.AdjSUN 3

    let propagator = function
      | U | D | C | S | T | B → Prop_Spinor
      | Ubar | Dbar | Cbar | Sbar | Tbar | Bbar → Prop_ConjSpinor
      | Gl → Prop_Feynman

    let width _ = Timelike

    let goldstone _ =
      None

    let conjugate = function
      | U → Ubar
```

```
    | D → Dbar
    | C → Cbar
    | S → Sbar
    | T → Tbar
    | B → Bbar
    | Ubar → U
    | Dbar → D
    | Cbar → C
    | Sbar → S
    | Tbar → T
    | Bbar → B
    | Gl → Gl

  let fermion = function
    | U | D | C | S | T | B → 1
    | Ubar | Dbar | Cbar | Sbar | Tbar | Bbar → − 1
    | Gl → 0

  module Ch = Charges.ZZ
  let charges = function
    | D → [1; 0; 0; 0; 0; 0]
    | U → [0; 1; 0; 0; 0; 0]
    | S → [0; 0; 1; 0; 0; 0]
    | C → [0; 0; 0; 1; 0; 0]
    | B → [0; 0; 0; 0; 1; 0]
    | T → [0; 0; 0; 0; 0; 1]
    | Dbar → [−1; 0; 0; 0; 0; 0]
    | Ubar → [0; − 1; 0; 0; 0; 0]
    | Sbar → [0; 0; − 1; 0; 0; 0]
    | Cbar → [0; 0; 0; − 1; 0; 0]
    | Bbar → [0; 0; 0; 0; − 1; 0]
    | Tbar → [0; 0; 0; 0; 0; − 1]
    | Gl → [0; 0; 0; 0; 0; 0]

  module F = Modeltools.Fusions (struct
    type f = flavor
    type c = constant
    let compare = compare
    let conjugate = conjugate
  end)
```

This is compatible with CD+.

```
  let color_current =
    [ ((Dbar, Gl, D), FBF ((−1), Psibar, V, Psi), Gs);
      ((Ubar, Gl, U), FBF ((−1), Psibar, V, Psi), Gs);
      ((Cbar, Gl, C), FBF ((−1), Psibar, V, Psi), Gs);
      ((Sbar, Gl, S), FBF ((−1), Psibar, V, Psi), Gs);
      ((Tbar, Gl, T), FBF ((−1), Psibar, V, Psi), Gs);
      ((Bbar, Gl, B), FBF ((−1), Psibar, V, Psi), Gs)]

 let three_gluon =
    [ ((Gl, Gl, Gl), Gauge_Gauge_Gauge 1, I_Gs)]
```

```
let gauge4 = Vector4 [(2, C_13_42); (−1, C_12_34); (−1, C_14_23)]
let four_gluon =
  [ ((Gl, Gl, Gl, Gl), gauge4, G2)]
let vertices3 =
  (color_current @ three_gluon)
let vertices4 = four_gluon
let vertices () =
  (vertices3, vertices4, [])
let table = F.of_vertices (vertices ())
let fuse2 = F.fuse2 table
let fuse3 = F.fuse3 table
let fuse = F.fuse table
let max_degree () = 4
let parameters () = { input = [Gs, 1.0]; derived = []; derived_arrays = [] }
let flavor_of_string = function
  | "u" → U
  | "d" → D
  | "c" → C
  | "s" → S
  | "t" → T
  | "b" → B
  | "ubar" → Ubar
  | "dbar" → Dbar
  | "cbar" → Cbar
  | "sbar" → Sbar
  | "tbar" → Tbar
  | "bbar" → Bbar
  | "gl" → Gl
  | _ → invalid_arg "Modellib.QCD.flavor_of_string"
let flavor_to_string = function
  | U → "u"
  | Ubar → "ubar"
  | D → "d"
  | Dbar → "dbar"
  | C → "c"
  | Cbar → "cbar"
  | S → "s"
  | Sbar → "sbar"
  | T → "t"
  | Tbar → "tbar"
  | B → "b"
  | Bbar → "bbar"
  | Gl → "gl"
let flavor_to_TeX = function
  | U → "u"
  | Ubar → "\\bar{u}"
```

```
    | D → "d"
    | Dbar → "\\bar{d}"
    | C → "c"
    | Cbar → "\\bar{c}"
    | S → "s"
    | Sbar → "\\bar{s}"
    | T → "t"
    | Tbar → "\\bar{t}"
    | B → "b"
    | Bbar → "\\bar{b}"
    | Gl → "g"

  let flavor_symbol = function
    | U → "u"
    | Ubar → "ubar"
    | D → "d"
    | Dbar → "dbar"
    | C → "c"
    | Cbar → "cbar"
    | S → "s"
    | Sbar → "sbar"
    | T → "t"
    | Tbar → "tbar"
    | B → "b"
    | Bbar → "bbar"
    | Gl → "gl"

  let gauge_symbol () =
    failwith "Modellib.QCD.gauge_symbol:␣internal␣error"

  let pdg = function
    | D → 1 | Dbar → −1
    | U → 2 | Ubar → −2
    | S → 3 | Sbar → −3
    | C → 4 | Cbar → −4
    | B → 5 | Bbar → −5
    | T → 6 | Tbar → −6
    | Gl → 21

  let mass_symbol f =
    "mass(" ^ string_of_int (abs (pdg f)) ^ ")"

  let width_symbol f =
    "width(" ^ string_of_int (abs (pdg f)) ^ ")"

  let constant_symbol = function
    | I_Gs → "(0,1)*gs"
    | Gs → "gs"
    | G2 → "gs**2"

end
```

13.4.5 Complete Minimal Standard Model (Unitarity Gauge)

```
module type SM_flags =
  sig
    val higgs_triangle : bool (* Hγγ, Hgγ and Hgg couplings *)
    val higgs_hmm : bool
    val triple_anom : bool
    val quartic_anom : bool
    val higgs_anom : bool
    val k_matrix : bool
    val ckm_present : bool
    val top_anom : bool
    val top_anom_4f : bool
    val tt_threshold : bool
  end

module SM_no_anomalous : SM_flags =
  struct
    let higgs_triangle = false
    let higgs_hmm = false
    let triple_anom = false
    let quartic_anom = false
    let higgs_anom = false
    let k_matrix = false
    let ckm_present = false
    let top_anom = false
    let top_anom_4f = false
    let tt_threshold = false
  end

module SM_no_anomalous_ckm : SM_flags =
  struct
    let higgs_triangle = false
    let higgs_hmm = false
    let triple_anom = false
    let quartic_anom = false
    let higgs_anom = false
    let k_matrix = false
    let ckm_present = true
    let top_anom = false
    let top_anom_4f = false
    let tt_threshold = false
  end

module SM_anomalous : SM_flags =
  struct
    let higgs_triangle = false
    let higgs_hmm = false
    let triple_anom = true
    let quartic_anom = true
    let higgs_anom = true
```

```
    let k_matrix = false
    let ckm_present = false
    let top_anom = false
    let top_anom_4f = false
    let tt_threshold = false
  end

module SM_anomalous_ckm : SM_flags =
  struct
    let higgs_triangle = false
    let higgs_hmm = false
    let triple_anom = true
    let quartic_anom = true
    let higgs_anom = true
    let k_matrix = false
    let ckm_present = true
    let top_anom = false
    let top_anom_4f = false
    let tt_threshold = false
  end

module SM_k_matrix : SM_flags =
  struct
    let higgs_triangle = false
    let higgs_hmm = false
    let triple_anom = false
    let quartic_anom = true
    let higgs_anom = false
    let k_matrix = true
    let ckm_present = false
    let top_anom = false
    let top_anom_4f = false
    let tt_threshold = false
  end

module SM_Higgs : SM_flags =
  struct
    let higgs_triangle = true
    let higgs_hmm = true
    let triple_anom = false
    let quartic_anom = false
    let higgs_anom = false
    let k_matrix = false
    let ckm_present = false
    let top_anom = false
    let top_anom_4f = false
    let tt_threshold = false
  end

module SM_anomalous_top : SM_flags =
  struct
    let higgs_triangle = false
```

```
    let higgs_hmm = false
    let triple_anom = false
    let quartic_anom = false
    let higgs_anom = false
    let k_matrix = false
    let ckm_present = false
    let top_anom = true
    let top_anom_4f = true
    let tt_threshold = false
  end

module SM_tt_threshold : SM_flags =
  struct
    let higgs_triangle = false
    let higgs_hmm = false
    let triple_anom = false
    let quartic_anom = false
    let higgs_anom = false
    let k_matrix = false
    let ckm_present = false
    let top_anom = false
    let top_anom_4f = false
    let tt_threshold = true
  end
```

13.4.6 Complete Minimal Standard Model (including some extensions)

```
module SM (Flags : SM_flags) =
  struct
    let rcs = RCS.rename rcs_file "Modellib.SM"
        [ "minimal electroweak standard model in unitarity gauge"]

    open Coupling

    let default_width = ref Timelike
    let use_fudged_width = ref false

    let options = Options.create
      [ "constant_width", Arg.Unit (fun () → default_width := Constant),
        "use constant width (also in t-channel)";
        "fudged_width", Arg.Set use_fudged_width,
        "use fudge factor for charge particle width";
        "custom_width", Arg.String (fun f → default_width := Custom f),
        "use custom width";
        "cancel_widths", Arg.Unit (fun () → default_width := Vanishing),
        "use vanishing width"]

    type f_aux_top = TTGG | TBWA | TBWZ | TTWW | BBWW |
 QGUG | QBUB | QW | DL | DR |
                      QUQD1L | QUQD1R | QUQD8L | QUQD8R
```

```
type matter_field = L of int | N of int | U of int | D of int
type gauge_boson = Ga | Wp | Wm | Z | Gl
type other = Phip | Phim | Phi0 | H
           | Aux_top of int × int × int × bool × f_aux_top type flavor = M of matter_field |
G of gauge_boson | O of other

let matter_field f = M f
let gauge_boson f = G f
let other f = O f

type field =
  | Matter of matter_field
  | Gauge of gauge_boson
  | Other of other

let field = function
  | M f → Matter f
  | G f → Gauge f
  | O f → Other f

type gauge = unit

let gauge_symbol () =
  failwith "Modellib.SM.gauge_symbol: internal error"

let family n = List.map matter_field [ L n; N n; U n; D n ]

let rec aux_top_flavors (f, l, co, ch) = List.append
  ( List.map other [ Aux_top(l, co, ch/2,true,f); Aux_top(l, co, ch/2,false,f) ] )
  ( if ch > 1 then List.append
      ( List.map other [ Aux_top(l, co, −ch/2,true,f); Aux_top(l, co, −ch/2,false,f) ] )
      ( aux_top_flavors (f, l, co, (ch − 2)) )
    else [ ] )

let external_flavors () =
  [ "1st Generation", ThoList.flatmap family [1; − 1];
    "2nd Generation", ThoList.flatmap family [2; − 2];
    "3rd Generation", ThoList.flatmap family [3; − 3];
    "Gauge Bosons", List.map gauge_boson [Ga; Z; Wp; Wm; Gl];
    "Higgs", List.map other [H];
    "Goldstone Bosons", List.map other [Phip; Phim; Phi0] ]

let flavors () = List.append
  ( ThoList.flatmap snd (external_flavors ()) )
  ( ThoList.flatmap aux_top_flavors
      [ (TTGG, 2, 1, 1); (TBWA, 2, 0, 2); (TBWZ, 2, 0, 2); (TTWW, 2, 0, 1); (BBWW, 2, 0, 1);
        (QGUG, 1, 1, 1); (QBUB, 1, 0, 1); (QW, 1, 0, 3); (DL, 0, 0, 3); (DR, 0, 0, 3);
        (QUQD1L, 0, 0, 3); (QUQD1R, 0, 0, 3); (QUQD8L, 0, 1, 3); (QUQD8R, 0, 1, 3) ] )

let spinor n =
  if n ≥ 0 then
    Spinor
  else
    ConjSpinor
```

```
let lorentz_aux = function
  | 2 → Tensor_1
  | 1 → Vector
  | 0 → Scalar
  | _ → invalid_arg ("SM.lorentz_aux: wrong value")

let lorentz = function
  | M f →
      begin match f with
      | L n → spinor n | N n → spinor n
      | U n → spinor n | D n → spinor n
      end
  | G f →
      begin match f with
      | Ga | Gl → Vector
      | Wp | Wm | Z → Massive_Vector
      end
  | O f →
      begin match f with
      | Aux_top (l, _, _, _, _) → lorentz_aux l
      | _ → Scalar
      end

let color = function
  | M (U n) → Color.SUN (if n > 0 then 3 else -3)
  | M (D n) → Color.SUN (if n > 0 then 3 else -3)
  | G Gl → Color.AdjSUN 3
  | O (Aux_top (_, co, _, _, _)) → if co ≡ 0 then Color.Singlet else Color.AdjSUN 3
  | _ → Color.Singlet

let prop_spinor n =
  if n ≥ 0 then
    Prop_Spinor
  else
    Prop_ConjSpinor

let prop_aux = function
  | 2 → Aux_Tensor_1
  | 1 → Aux_Vector
  | 0 → Aux_Scalar
  | _ → invalid_arg ("SM.prop_aux: wrong value")

let propagator = function
  | M f →
      begin match f with
      | L n → prop_spinor n | N n → prop_spinor n
      | U n → prop_spinor n | D n → prop_spinor n
      end
  | G f →
      begin match f with
      | Ga | Gl → Prop_Feynman
      | Wp | Wm | Z → Prop_Unitarity
      end
```

```
  | O f ->
      begin match f with
      | Phip | Phim | Phi0 -> Only_Insertion
      | H -> Prop_Scalar
      | Aux_top (l, _, _, _, _) -> prop_aux l
      end
```

Optionally, ask for the fudge factor treatment for the widths of charged particles. Currently, this only applies to $W^{\pm}$ and top.

```
let width f =
  if !use_fudged_width then
    match f with
    | G Wp | G Wm | M (U 3) | M (U (-3)) -> Fudged
    | _ -> !default_width
  else
    !default_width

let goldstone = function
  | G f ->
      begin match f with
      | Wp -> Some (O Phip, Coupling.Const 1)
      | Wm -> Some (O Phim, Coupling.Const 1)
      | Z -> Some (O Phi0, Coupling.Const 1)
      | _ -> None
      end
  | _ -> None

let conjugate = function
  | M f ->
      M (begin match f with
      | L n -> L (-n) | N n -> N (-n)
      | U n -> U (-n) | D n -> D (-n)
      end)
  | G f ->
      G (begin match f with
      | Gl -> Gl | Ga -> Ga | Z -> Z
      | Wp -> Wm | Wm -> Wp
      end)
  | O f ->
      O (begin match f with
      | Phip -> Phim | Phim -> Phip | Phi0 -> Phi0
      | H -> H
      | Aux_top (l, co, ch, n, f) -> Aux_top (l, co, (-ch), (¬ n), f)
      end)

let fermion = function
  | M f ->
      begin match f with
      | L n -> if n > 0 then 1 else -1
      | N n -> if n > 0 then 1 else -1
      | U n -> if n > 0 then 1 else -1
      | D n -> if n > 0 then 1 else -1
```

```
        end
  | G f →
        begin match f with
        | Gl | Ga | Z | Wp | Wm → 0
        end
  | O _ → 0
```

Electrical charge, lepton number, baryon number. We could avoid the rationals altogether by multiplying the first and last by 3 ...

```
module Ch = Charges.QQ
let ( // ) = Algebra.Small_Rational.make

let generation' = function
  | 1 → [ 1//1; 0//1; 0//1]
  | 2 → [ 0//1; 1//1; 0//1]
  | 3 → [ 0//1; 0//1; 1//1]
  | −1 → [−1//1; 0//1; 0//1]
  | −2 → [ 0//1; −1//1; 0//1]
  | −3 → [ 0//1; 0//1; −1//1]
  | n → invalid_arg ("SM.generation': " ^ string_of_int n)

let generation f =
  if Flags.ckm_present then
    []
  else
    match f with
    | M (L n | N n | U n | D n) → generation' n
    | G _ | O _ → [0//1; 0//1; 0//1]

let charge = function
  | M f →
        begin match f with
        | L n → if n > 0 then -1//1 else 1//1
        | N n → 0//1
        | U n → if n > 0 then 2//3 else -2//3
        | D n → if n > 0 then -1//3 else 1//3
        end
  | G f →
        begin match f with
        | Gl | Ga | Z → 0//1
        | Wp → 1//1
        | Wm → −1//1
        end
  | O f →
        begin match f with
        | H | Phi0 → 0//1
        | Phip → 1//1
        | Phim → −1//1
        | Aux_top (_, _, ch, _, _) → ch//1
        end

let lepton = function
```

```
  | M f →
      begin match f with
      | L n | N n → if n > 0 then 1//1 else -1//1
      | U _ | D _ → 0//1
      end
  | G _ | O _ → 0//1

let baryon = function
  | M f →
      begin match f with
      | L _ | N _ → 0//1
      | U n | D n → if n > 0 then 1//1 else -1//1
      end
  | G _ | O _ → 0//1

let charges f =
  [ charge f; lepton f; baryon f] @ generation f

type constant =
  | Unit | Half | Pi | Alpha_QED | Sin2thw
  | Sinthw | Costhw | E | G_weak | I_G_weak | Vev
  | Q_lepton | Q_up | Q_down | G_CC | G_CCQ of int×int
  | G_NC_neutrino | G_NC_lepton | G_NC_up | G_NC_down
  | G_TVA_ttA | G_TVA_bbA
  | G_VLR_ttZ | G_TVA_ttZ | G_TVA_bbZ
  | VA_ILC_ttA | VA_ILC_ttZ
  | G_VLR_btW | G_VLR_tbW
  | G_TLR_btW | G_TRL_tbW
  | G_TLR_btWZ | G_TRL_tbWZ
  | G_TLR_btWA | G_TRL_tbWA
  | G_TVA_ttWW | G_TVA_bbWW
  | G_TVA_ttG | G_TVA_ttGG
  | G_SP_ttH
  | G_VLR_qGuG | G_VLR_qBuB
  | G_VLR_qBuB_u | G_VLR_qBuB_d | G_VLR_qBuB_e | G_VL_qBuB_n
  | G_VL_qW | G_VL_qW_u | G_VL_qW_d
  | G_SL_DttR | G_SR_DttR | G_SL_DttL | G_SLR_DbtR |
G_SL_DbtL
  | C_quqd1R_bt | C_quqd1R_tb | C_quqd1L_bt | C_quqd1L_tb
  | C_quqd8R_bt | C_quqd8R_tb | C_quqd8L_bt | C_quqd8L_tb
  | I_Q_W | I_G_ZWW
  | G_WWWW | G_ZZWW | G_AZWW | G_AAWW
  | I_G1_AWW | I_G1_ZWW
  | I_G1_plus_kappa_plus_G4_AWW
  | I_G1_plus_kappa_plus_G4_ZWW
  | I_G1_plus_kappa_minus_G4_AWW
  | I_G1_plus_kappa_minus_G4_ZWW
  | I_G1_minus_kappa_plus_G4_AWW
  | I_G1_minus_kappa_plus_G4_ZWW
  | I_G1_minus_kappa_minus_G4_AWW
  | I_G1_minus_kappa_minus_G4_ZWW
  | I_lambda_AWW | I_lambda_ZWW
```

```
    | G5_AWW | G5_ZWW
    | I_kappa5_AWW | I_kappa5_ZWW
    | I_lambda5_AWW | I_lambda5_ZWW
    | Alpha_WWWW0 | Alpha_ZZWW1 | Alpha_WWWW2
    | Alpha_ZZWW0 | Alpha_ZZZZ
    | D_Alpha_ZZWW0_S | D_Alpha_ZZWW0_T | D_Alpha_ZZWW1_S
    | D_Alpha_ZZWW1_T | D_Alpha_ZZWW1_U | D_Alpha_WWWW0_S
    | D_Alpha_WWWW0_T | D_Alpha_WWWW0_U | D_Alpha_WWWW2_S
    | D_Alpha_WWWW2_T | D_Alpha_ZZZZ_S | D_Alpha_ZZZZ_T
    | G_HWW | G_HHWW | G_HZZ | G_HHZZ
    | G_Htt | G_Hbb | G_Hcc | G_Hmm | G_Htautau | G_H3 |
G_H4
    | G_HGaZ | G_HGaGa | G_Hgg
    | G_HGaZ_anom | G_HGaGa_anom | G_HZZ_anom | G_HWW_anom
    | G_HGaZ_u | G_HZZ_u | G_HWW_u
    | Gs | I_Gs | G2
    | Mass of flavor | Width of flavor
    | K_Matrix_Coeff of int | K_Matrix_Pole of int
```

Two integer counters for the QCD and EW order of the couplings.

```
type orders = int × int

let orders = function
    | Q_lepton | Q_up | Q_down | G_NC_lepton | G_NC_neutrino
    | G_NC_up | G_NC_down | G_CC | G_CCQ _ | G_Htt | G_H3
    | G_Hbb | G_Hcc | G_Htautau | G_Hmm | I_Q_W
    | I_G_ZWW | I_G1_AWW | I_G1_ZWW | I_G_weak
    | G_HWW | G_HZZ | G_HWW_u | G_HZZ_u | G_HGaZ_u
    | G_HWW_anom | G_HZZ_anom | G_HGaZ | G_HGaGa |
G_HGaZ_anom
    | G_HGaGa_anom | Half | Unit
    | I_G1_plus_kappa_plus_G4_AWW
    | I_G1_plus_kappa_plus_G4_ZWW
    | I_G1_minus_kappa_plus_G4_AWW
    | I_G1_minus_kappa_plus_G4_ZWW
    | I_G1_plus_kappa_minus_G4_AWW
    | I_G1_plus_kappa_minus_G4_ZWW
    | I_G1_minus_kappa_minus_G4_AWW
    | I_G1_minus_kappa_minus_G4_ZWW | I_kappa5_AWW
    | I_kappa5_ZWW | G5_AWW | G5_ZWW
    | I_lambda_AWW | I_lambda_ZWW | I_lambda5_AWW
    | I_lambda5_ZWW | G_TVA_ttA | G_TVA_bbA
    | G_VLR_ttZ | G_TVA_ttZ | G_TVA_bbZ
    | VA_ILC_ttA | VA_ILC_ttZ
    | G_VLR_btW | G_VLR_tbW | G_TLR_btW | G_TRL_tbW
    | G_TLR_btWA | G_TRL_tbWA | G_TLR_btWZ | G_TRL_tbWZ
    | G_VLR_qBuB | G_VLR_qBuB_u | G_VLR_qBuB_d
    | G_VLR_qBuB_e | G_VL_qBuB_n | G_VL_qW | G_VL_qW_u |
G_VL_qW_d
    | G_SL_DttR | G_SR_DttR | G_SL_DttL | G_SLR_DbtR |
G_SL_DbtL
```

| G_TVA_ttWW | G_TVA_bbWW | G_SP_ttH → (0, 1)
| G_HHWW | G_HHZZ | G_H4
| G_WWWW | G_ZZWW | G_AZWW | G_AAWW
| Alpha_WWWW0 | Alpha_WWWW2 | Alpha_ZZWW0
| Alpha_ZZWW1 | Alpha_ZZZZ
| D_Alpha_WWWW0_S | D_Alpha_WWWW0_T | D_Alpha_WWWW0_U
| D_Alpha_WWWW2_S | D_Alpha_WWWW2_T | D_Alpha_ZZWW0_S
| D_Alpha_ZZWW0_T | D_Alpha_ZZWW1_S | D_Alpha_ZZWW1_T
| D_Alpha_ZZWW1_U | D_Alpha_ZZZZ_S | D_Alpha_ZZZZ_T →
(0, 2)
| Gs | I_Gs | G_TVA_ttG | G_TVA_ttGG | G_VLR_qGuG
| C_quqd1R_bt | C_quqd1R_tb | C_quqd1L_bt | C_quqd1L_tb
| C_quqd8R_bt | C_quqd8R_tb | C_quqd8L_bt | C_quqd8L_tb →
(1, 0)
| G2 | G_Hgg → (2, 0)
(∗ These constants are not used, hence initialized to zero. ∗)
| Sinthw | Sin2thw | Costhw | Pi
| Alpha_QED | G_weak | K_Matrix_Coeff _
| K_Matrix_Pole _ | Mass _ | Width _ | Vev | E → (0, 0)

The current abstract syntax for parameter dependencies is admittedly tedious. Later, there will be a parser for a convenient concrete syntax as a part of a concrete syntax for models. But as these examples show, it should include simple functions.

$$\alpha_{\text{QED}} = \frac{1}{137.0359895} \tag{13.1a}$$
$$\sin^2\theta_w = 0.23124 \tag{13.1b}$$

let *input_parameters* =
[*Alpha_QED*, 1. /. 137.0359895;
Sin2thw, 0.23124;
Mass (*G Z*), 91.187;
Mass (*M* (*N* 1)), 0.0; *Mass* (*M* (*L* 1)), $0.51099907 \cdot 10^{-3}$;
Mass (*M* (*N* 2)), 0.0; *Mass* (*M* (*L* 2)), 0.105658389;
Mass (*M* (*N* 3)), 0.0; *Mass* (*M* (*L* 3)), 1.77705;
Mass (*M* (*U* 1)), $5.0 \cdot 10^{-3}$; *Mass* (*M* (*D* 1)), $3.0 \cdot 10^{-3}$;
Mass (*M* (*U* 2)), 1.2; *Mass* (*M* (*D* 2)), 0.1;
Mass (*M* (*U* 3)), 174.0; *Mass* (*M* (*D* 3)), 4.2]

$$e = \sqrt{4\pi\alpha} \tag{13.2a}$$
$$\sin\theta_w = \sqrt{\sin^2\theta_w} \tag{13.2b}$$
$$\cos\theta_w = \sqrt{1-\sin^2\theta_w} \tag{13.2c}$$
$$g = \frac{e}{\sin\theta_w} \tag{13.2d}$$

$$m_W = \cos\theta_w m_Z \tag{13.2e}$$

$$v = \frac{2m_W}{g} \tag{13.2f}$$

$$g_{CC} = -\frac{g}{2\sqrt{2}} = -\frac{e}{2\sqrt{2}\sin\theta_w} \tag{13.2g}$$

$$Q_{\text{lepton}} = -q_{\text{lepton}}e = e \tag{13.2h}$$

$$Q_{\text{up}} = -q_{\text{up}}e = -\frac{2}{3}e \tag{13.2i}$$

$$Q_{\text{down}} = -q_{\text{down}}e = \frac{1}{3}e \tag{13.2j}$$

$$\mathrm{i}q_W e = \mathrm{i}g_{\gamma WW} = \mathrm{i}e \tag{13.2k}$$

$$\mathrm{i}g_{ZWW} = \mathrm{i}g\cos\theta_w \tag{13.2l}$$

$$\mathrm{i}g_{WWW} = \mathrm{i}g \tag{13.2m}$$

... to be continued ... The quartic couplings can't be correct, because the dimensions are wrong!

$$g_{HWW} = gm_W = 2\frac{m_W^2}{v} \tag{13.3a}$$

$$g_{HHWW} = 2\frac{m_W^2}{v^2} = \frac{g^2}{2} \tag{13.3b}$$

$$g_{HZZ} = \frac{g}{\cos\theta_w}m_Z \tag{13.3c}$$

$$g_{HHZZ} = 2\frac{m_Z^2}{v^2} = \frac{g^2}{2\cos\theta_w} \tag{13.3d}$$

$$g_{Htt} = \lambda_t \tag{13.3e}$$

$$g_{Hbb} = \lambda_b = \frac{m_b}{m_t}\lambda_t \tag{13.3f}$$

$$g_{H^3} = -\frac{3g}{2}\frac{m_H^2}{m_W} = -3\frac{m_H^2}{v} \quad g_{H^4} = -\frac{3g^2}{4}\frac{m_W^2}{v^2} = -3\frac{m_H^2}{v^2} \tag{13.3g}$$

```
let derived_parameters =
  [ Real E, Sqrt (Prod [Const 4; Atom Pi; Atom Alpha_QED]);
    Real Sinthw, Sqrt (Atom Sin2thw);
    Real Costhw, Sqrt (Diff (Const 1, Atom Sin2thw));
    Real G_weak, Quot (Atom E, Atom Sinthw);
    Real (Mass (G Wp)), Prod [Atom Costhw; Atom (Mass (G Z))];
    Real Vev, Quot (Prod [Const 2; Atom (Mass (G Wp))], Atom G_weak);
    Real Q_lepton, Atom E;
    Real Q_up, Prod [Quot (Const (−2), Const 3); Atom E];
    Real Q_down, Prod [Quot (Const 1, Const 3); Atom E];
    Real G_CC, Neg (Quot (Atom G_weak, Prod [Const 2; Sqrt (Const 2)]));
    Complex I_Q_W, Prod [I; Atom E];
    Complex I_G_weak, Prod [I; Atom G_weak];
    Complex I_G_ZWW, Prod [I; Atom G_weak; Atom Costhw] ]
```

$$-\frac{g}{2\cos\theta_w} \tag{13.4}$$

```
let g_over_2_costh =
  Quot (Neg (Atom G_weak), Prod [Const 2; Atom Costhw])
```

$$-\frac{g}{2\cos\theta_w}g_V = -\frac{g}{2\cos\theta_w}(T_3 - 2q\sin^2\theta_w) \tag{13.5a}$$

$$-\frac{g}{2\cos\theta_w}g_A = -\frac{g}{2\cos\theta_w}T_3 \tag{13.5b}$$

```
let nc_coupling c t3 q =
  (Real_Array c,
   [Prod [g_over_2_costh; Diff (t3, Prod [Const 2; q; Atom Sin2thw])];
    Prod [g_over_2_costh; t3]])

let half = Quot (Const 1, Const 2)

let derived_parameter_arrays =
  [ nc_coupling G_NC_neutrino half (Const 0);
    nc_coupling G_NC_lepton (Neg half) (Const (-1));
    nc_coupling G_NC_up half (Quot (Const 2, Const 3));
    nc_coupling G_NC_down (Neg half) (Quot (Const (-1), Const 3)) ]

let parameters () =
  { input = input_parameters;
    derived = derived_parameters;
    derived_arrays = derived_parameter_arrays }

module F = Modeltools.Fusions (struct
  type f = flavor
  type c = constant
  let compare = compare
  let conjugate = conjugate
end)
```

$$\mathcal{L}_{\mathrm{EM}} = -e\sum_i q_i\bar\psi_i \not{A}\psi_i \tag{13.6}$$

```
let mgm ((m1, g, m2), fbf, c) = ((M m1, G g, M m2), fbf, c)
let mom ((m1, o, m2), fbf, c) = ((M m1, O o, M m2), fbf, c)

let electromagnetic_currents n =
  List.map mgm
    [ ((L (-n), Ga, L n), FBF (1, Psibar, V, Psi), Q_lepton);
      ((U (-n), Ga, U n), FBF (1, Psibar, V, Psi), Q_up);
      ((D (-n), Ga, D n), FBF (1, Psibar, V, Psi), Q_down) ]

let color_currents n =
  List.map mgm
    [ ((U (-n), Gl, U n), FBF ((-1), Psibar, V, Psi), Gs);
      ((D (-n), Gl, D n), FBF ((-1), Psibar, V, Psi), Gs) ]
```

$$\mathcal{L}_{\mathrm{NC}} = -\frac{g}{2\cos\theta_W}\sum_i \bar\psi_i \not{Z}(g_V^i - g_A^i\gamma_5)\psi_i \tag{13.7}$$

```
let neutral_currents n =
  List.map mgm
    [ ((L (-n), Z, L n), FBF (1, Psibar, VA, Psi), G_NC_lepton);
      ((N (-n), Z, N n), FBF (1, Psibar, VA, Psi), G_NC_neutrino);
      ((U (-n), Z, U n), FBF (1, Psibar, VA, Psi), G_NC_up);
      ((D (-n), Z, D n), FBF (1, Psibar, VA, Psi), G_NC_down) ]
```

$$\mathcal{L}_{\mathrm{CC}} = -\frac{g}{2\sqrt{2}} \sum_i \bar\psi_i (T^+ \not W^+ + T^- \not W^-)(1-\gamma_5)\psi_i \tag{13.8}$$

```
let charged_currents' n =
  List.map mgm
    [ ((L (-n), Wm, N n), FBF (1, Psibar, VL, Psi), G_CC);
      ((N (-n), Wp, L n), FBF (1, Psibar, VL, Psi), G_CC) ]

let charged_currents'' n =
  List.map mgm
    [ ((D (-n), Wm, U n), FBF (1, Psibar, VL, Psi), G_CC);
      ((U (-n), Wp, D n), FBF (1, Psibar, VL, Psi), G_CC) ]

let charged_currents_triv =
  ThoList.flatmap charged_currents' [1; 2; 3] @
  ThoList.flatmap charged_currents'' [1; 2; 3]

let charged_currents_ckm =
  let charged_currents_2 n1 n2 =
    List.map mgm
      [ ((D (-n1), Wm, U n2), FBF (1, Psibar, VL, Psi), G_CCQ (n2, n1));
        ((U (-n1), Wp, D n2), FBF (1, Psibar, VL, Psi), G_CCQ (n1, n2)) ] in
  ThoList.flatmap charged_currents' [1; 2; 3] @
  List.flatten (Product.list2 charged_currents_2 [1; 2; 3] [1; 2; 3])

let yukawa =
  [ ((M (U (-3)), O H, M (U 3)), FBF (1, Psibar, S, Psi), G_Htt);
    ((M (D (-3)), O H, M (D 3)), FBF (1, Psibar, S, Psi), G_Hbb);
    ((M (U (-2)), O H, M (U 2)), FBF (1, Psibar, S, Psi), G_Hcc);
    ((M (L (-3)), O H, M (L 3)), FBF (1, Psibar, S, Psi), G_Htautau) ] @
  if Flags.higgs_hmm then
  [ ((M (L (-2)), O H, M (L 2)), FBF (1, Psibar, S, Psi), G_Hmm)]
      else
  []
```

$$\mathcal{L}_{\mathrm{TGC}} = -e\partial_\mu A_\nu W_+^\mu W_-^\nu + \ldots - e\cot\theta_w \partial_\mu Z_\nu W_+^\mu W_-^\nu + \ldots \tag{13.9}$$

```
let tgc ((g1, g2, g3), t, c) = ((G g1, G g2, G g3), t, c)

let standard_triple_gauge =
  List.map tgc
    [ ((Ga, Wm, Wp), Gauge_Gauge_Gauge 1, I_Q_W);
      ((Z, Wm, Wp), Gauge_Gauge_Gauge 1, I_G_ZWW);
      ((Gl, Gl, Gl), Gauge_Gauge_Gauge 1, I_Gs)]
```

$$\begin{aligned}\mathcal{L}_{\mathrm{TGC}}(g_1,\kappa) = g_1\mathcal{L}_T(V,W^+,W^-) \\ + \frac{\kappa+g_1}{2}\Big(\mathcal{L}_T(W^-,V,W^+) - \mathcal{L}_T(W^+,V,W^-)\Big) \\ + \frac{\kappa-g_1}{2}\Big(\mathcal{L}_L(W^-,V,W^+) - \mathcal{L}_T(W^+,V,W^-)\Big)\end{aligned} \quad (13.10)$$

The whole thing in the LEP2 workshop notation:

$$\begin{aligned}&\mathrm{i}\mathcal{L}_{\mathrm{TGC},V}/g_{WWV} = \\ &g_1^V V^\mu (W^-_{\mu\nu}W^{+,\nu} - W^+_{\mu\nu}W^{-,\nu}) + \kappa_V W^+_\mu W^-_\nu V^{\mu\nu} + \frac{\lambda_V}{m_W^2} V_{\mu\nu} W^-_{\rho\mu} W^{+,}{}_\nu{}^\rho \\ &\quad + \mathrm{i}g_5^V \epsilon_{\mu\nu\rho\sigma}\left((\partial^\rho W^{-,\mu})W^{+,\nu} - W^{-,\mu}(\partial^\rho W^{+,\nu})\right)V^\sigma \\ &+\mathrm{i}g_4^V W^-_\mu W^+_\nu(\partial^\mu V^\nu + \partial^\nu V^\mu) - \frac{\tilde\kappa_V}{2} W^-_\mu W^+_\nu \epsilon^{\mu\nu\rho\sigma} V_{\rho\sigma} - \frac{\tilde\lambda_V}{2m_W^2} W^-_{\rho\mu} W^{+,\mu}{}_\nu \epsilon^{\nu\rho\alpha\beta} V_{\alpha\beta}\end{aligned} \quad (13.11)$$

using the conventions of Itzykson and Zuber with $\epsilon^{0123} = +1$.

This is equivalent to the notation of Hagiwara et al. [**?**], if we remember that they have opposite signs for g_{WWV}:

$$\begin{aligned}&\mathcal{L}_{WWV}/(-g_{WWV}) = \\ &\mathrm{i}g_1^V\left(W^\dagger_{\mu\nu}W^\mu - W^\dagger_\mu W^\mu{}_\nu\right)V^\nu + \mathrm{i}\kappa_V W^\dagger_\mu W_\nu V^{\mu\nu} + \mathrm{i}\frac{\lambda_V}{m_W^2} W^\dagger_{\lambda\mu} W^\mu{}_\nu V^{\nu\lambda} \\ &\quad - g_4^V W^\dagger_\mu W_\nu (\partial^\mu V^\nu + \partial^\nu V^\mu) + g_5^V \epsilon^{\mu\nu\lambda\sigma}\left(W^\dagger_\mu \overleftrightarrow{\partial_\lambda} W_\nu\right)V_\sigma \\ &\qquad + \mathrm{i}\tilde\kappa_V W^\dagger_\mu W_\nu \tilde V^{\mu\nu} + \mathrm{i}\frac{\tilde\lambda_V}{m_W^2} W^\dagger_{\lambda\mu} W^\mu{}_\nu \tilde V^{\nu\lambda}\end{aligned} \quad (13.12)$$

Here V^μ stands for either the photon or the Z field, W^μ is the W^- field, $W_{\mu\nu} = \partial_\mu W_\nu - \partial_\nu W_\mu$, $V_{\mu\nu} = \partial_\mu V_\nu - \partial_\nu V_\mu$, and $\tilde V_{\mu\nu} = \frac{1}{2}\epsilon_{\mu\nu\lambda\sigma}V^{\lambda\sigma}$.

```
let anomalous_triple_gauge =
  List.map tgc
    [ ((Ga, Wm, Wp), Dim4_Vector_Vector_Vector_T (−1),
        I_G1_AWW);
      ((Z, Wm, Wp), Dim4_Vector_Vector_Vector_T (−1),
       I_G1_ZWW);
      ((Wm, Ga, Wp), Dim4_Vector_Vector_Vector_T 1,
       I_G1_plus_kappa_minus_G4_AWW);
      ((Wm, Z, Wp), Dim4_Vector_Vector_Vector_T 1,
       I_G1_plus_kappa_minus_G4_ZWW);
      ((Wp, Ga, Wm), Dim4_Vector_Vector_Vector_T (−1),
       I_G1_plus_kappa_plus_G4_AWW);
      ((Wp, Z, Wm), Dim4_Vector_Vector_Vector_T (−1),
       I_G1_plus_kappa_plus_G4_ZWW);
      ((Wm, Ga, Wp), Dim4_Vector_Vector_Vector_L (−1),
```

```
      I_G1_minus_kappa_plus_G4_AWW);
     ((Wm, Z, Wp), Dim4_Vector_Vector_Vector_L (-1),
      I_G1_minus_kappa_plus_G4_ZWW);
     ((Wp, Ga, Wm), Dim4_Vector_Vector_Vector_L 1,
      I_G1_minus_kappa_minus_G4_AWW);
     ((Wp, Z, Wm), Dim4_Vector_Vector_Vector_L 1,
      I_G1_minus_kappa_minus_G4_ZWW);
     ((Ga, Wm, Wp), Dim4_Vector_Vector_Vector_L5 (-1),
      I_kappa5_AWW);
     ((Z, Wm, Wp), Dim4_Vector_Vector_Vector_L5 (-1),
      I_kappa5_ZWW);
     ((Ga, Wm, Wp), Dim4_Vector_Vector_Vector_T5 (-1),
      G5_AWW);
     ((Z, Wm, Wp), Dim4_Vector_Vector_Vector_T5 (-1),
      G5_ZWW);
     ((Ga, Wp, Wm), Dim6_Gauge_Gauge_Gauge (-1),
      I_lambda_AWW);
     ((Z, Wp, Wm), Dim6_Gauge_Gauge_Gauge (-1),
      I_lambda_ZWW);
     ((Ga, Wp, Wm), Dim6_Gauge_Gauge_Gauge_5 (-1),
      I_lambda5_AWW);
     ((Z, Wp, Wm), Dim6_Gauge_Gauge_Gauge_5 (-1),
      I_lambda5_ZWW) ]

let triple_gauge =
  if Flags.triple_anom then
    anomalous_triple_gauge
  else
    standard_triple_gauge
```

$$\mathcal{L}_{\text{QGC}} = -g^2 W_{+,\mu} W_{-,\nu} W_+^\mu W_-^\nu + \ldots \tag{13.13}$$

Actually, quartic gauge couplings are a little bit more straightforward using auxiliary fields. Here we have to impose the antisymmetry manually:

$$\begin{aligned}(W_1^{+,\mu} W_2^{-,\nu} - W_1^{+,\nu} W_2^{-,\mu})(W_{3,\mu}^+ W_{4,\nu}^- - W_{3,\nu}^+ W_{4,\mu}^-)\\ = 2(W_1^+ W_3^+)(W_2^- W_4^-) - 2(W_1^+ W_4^-)(W_2^- W_3^+)\end{aligned} \tag{13.14a}$$

also (V can be A or Z)

$$\begin{aligned}(W_1^{+,\mu} V_2^{\nu} - W_1^{+,\nu} V_2^{\mu})(W_{3,\mu}^- V_{4,\nu} - W_{3,\nu}^- V_{4,\mu})\\ = 2(W_1^+ W_3^-)(V_2 V_4) - 2(W_1^+ V_4)(V_2 W_3^-)\end{aligned} \tag{13.14b}$$

$$W^{+,\mu} W^{-,\nu} W_\mu^+ W_\nu^- \tag{13.15a}$$

```
let qgc ((g1, g2, g3, g4), t, c) = ((G g1, G g2, G g3, G g4), t, c)
let gauge4 = Vector4 [(2, C_13_42); (-1, C_12_34); (-1, C_14_23)]
```

```
let minus_gauge4 = Vector4 [(−2, C_13_42); (1, C_12_34); (1, C_14_23)]
let standard_quartic_gauge =
  List.map qgc
    [ (Wm, Wp, Wm, Wp), gauge4, G_WWWW;
      (Wm, Z, Wp, Z), minus_gauge4, G_ZZWW;
      (Wm, Z, Wp, Ga), minus_gauge4, G_AZWW;
      (Wm, Ga, Wp, Ga), minus_gauge4, G_AAWW;
      (Gl, Gl, Gl, Gl), gauge4, G2 ]
```

$$\begin{aligned}\mathcal{L}_4 = \alpha_4 \Bigg(\frac{g^4}{2} \left((W^+_\mu W^{-,\mu})^2 + W^+_\mu W^{+,\mu} W^-_\mu W^{-,\mu} \right) \\ + \frac{g^4}{\cos^2\theta_w} W^+_\mu Z^\mu W^-_\nu Z^\nu + \frac{g^4}{4\cos^4\theta_w} (Z_\mu Z^\mu)^2 \Bigg) \end{aligned} \tag{13.16a}$$

$$\mathcal{L}_5 = \alpha_5 \left(g^4 (W^+_\mu W^{-,\mu})^2 + \frac{g^4}{\cos^2\theta_w} W^+_\mu W^{-,\mu} Z_\nu Z^\nu + \frac{g^4}{4\cos^4\theta_w} (Z_\mu Z^\mu)^2 \right) \tag{13.16b}$$

or

$$\begin{aligned}\mathcal{L}_4 + \mathcal{L}_5 = (\alpha_4 + 2\alpha_5) g^4 \frac{1}{2} (W^+_\mu W^{-,\mu})^2 \\ + 2\alpha_4 g^4 \frac{1}{4} W^+_\mu W^{+,\mu} W^-_\mu W^{-,\mu} + \alpha_4 \frac{g^4}{\cos^2\theta_w} W^+_\mu Z^\mu W^-_\nu Z^\nu \\ + 2\alpha_5 \frac{g^4}{\cos^2\theta_w} \frac{1}{2} W^+_\mu W^{-,\mu} Z_\nu Z^\nu + (2\alpha_4 + 2\alpha_5) \frac{g^4}{\cos^4\theta_w} \frac{1}{8} (Z_\mu Z^\mu)^2 \end{aligned} \tag{13.17}$$

and therefore

$$\alpha_{(WW)_0} = (\alpha_4 + 2\alpha_5) g^4 \tag{13.18a}$$

$$\alpha_{(WW)_2} = 2\alpha_4 g^4 \tag{13.18b}$$

$$\alpha_{(WZ)_0} = 2\alpha_5 \frac{g^4}{\cos^2\theta_w} \tag{13.18c}$$

$$\alpha_{(WZ)_1} = \alpha_4 \frac{g^4}{\cos^2\theta_w} \tag{13.18d}$$

$$\alpha_{ZZ} = (2\alpha_4 + 2\alpha_5) \frac{g^4}{\cos^4\theta_w} \tag{13.18e}$$

```
let anomalous_quartic_gauge =
  if Flags.quartic_anom then
    List.map qgc
      [ ((Wm, Wm, Wp, Wp),
           Vector4 [(1, C_13_42); (1, C_14_23)], Alpha_WWWW0);
        ((Wm, Wm, Wp, Wp),
          Vector4 [1, C_12_34], Alpha_WWWW2);
        ((Wm, Wp, Z, Z),
          Vector4 [1, C_12_34], Alpha_ZZWW0);
        ((Wm, Wp, Z, Z),
```

```
        Vector4 [(1, C_13_42); (1, C_14_23)], Alpha_ZZWW1);
      ((Z, Z, Z, Z),
        Vector4 [(1, C_12_34); (1, C_13_42); (1, C_14_23)], Alpha_ZZZZ) ]
  else
    []
```

In any diagonal channel χ, the scattering amplitude $a_\chi(s)$ is unitary iff[1]

$$\mathrm{Im}\left(\frac{1}{a_\chi(s)}\right) = -1 \tag{13.20}$$

For a real perturbative scattering amplitude $r_\chi(s)$ this can be enforced easily–and arbitrarily–by

$$\frac{1}{a_\chi(s)} = \frac{1}{r_\chi(s)} - \mathrm{i} \tag{13.21}$$

```
let k_matrix_quartic_gauge =
  if Flags.k_matrix then
    List.map qgc
      [ ((Wm, Wp, Wm, Wp), Vector4_K_Matrix_jr (0,
                [(1, C_12_34)]), D_Alpha_WWWW0_S);
        ((Wm, Wp, Wm, Wp), Vector4_K_Matrix_jr (0,
                [(1, C_14_23)]), D_Alpha_WWWW0_T);
        ((Wm, Wp, Wm, Wp), Vector4_K_Matrix_jr (0,
                [(1, C_13_42)]), D_Alpha_WWWW0_U);
        ((Wp, Wm, Wp, Wm), Vector4_K_Matrix_jr (0,
                [(1, C_12_34)]), D_Alpha_WWWW0_S);
        ((Wp, Wm, Wp, Wm), Vector4_K_Matrix_jr (0,
                [(1, C_14_23)]), D_Alpha_WWWW0_T);
        ((Wp, Wm, Wp, Wm), Vector4_K_Matrix_jr (0,
                [(1, C_13_42)]), D_Alpha_WWWW0_U);
        ((Wm, Wm, Wp, Wp), Vector4_K_Matrix_jr (0,
                [(1, C_12_34)]), D_Alpha_WWWW2_S);
        ((Wm, Wm, Wp, Wp), Vector4_K_Matrix_jr (0,
                [(1, C_13_42); (1, C_14_23)]), D_Alpha_WWWW2_T);
        ((Wm, Wp, Z, Z), Vector4_K_Matrix_jr (0,
                [(1, C_12_34)]), D_Alpha_ZZWW0_S);
        ((Wm, Wp, Z, Z), Vector4_K_Matrix_jr (0,
                [(1, C_13_42); (1, C_14_23)]), D_Alpha_ZZWW0_T);
        ((Wm, Z, Wp, Z), Vector4_K_Matrix_jr (0,
                [(1, C_12_34)]), D_Alpha_ZZWW1_S);
        ((Wm, Z, Wp, Z), Vector4_K_Matrix_jr (0,
                [(1, C_13_42)]), D_Alpha_ZZWW1_T);
        ((Wm, Z, Wp, Z), Vector4_K_Matrix_jr (0,
                [(1, C_14_23)]), D_Alpha_ZZWW1_U);
        ((Wp, Z, Z, Wm), Vector4_K_Matrix_jr (1,
```

[1]Trivial proof:

$$-1 = \mathrm{Im}\left(\frac{1}{a_\chi(s)}\right) = \frac{\mathrm{Im}(a^*_\chi(s))}{|a_\chi(s)|^2} = -\frac{\mathrm{Im}(a_\chi(s))}{|a_\chi(s)|^2} \tag{13.19}$$

i. e. $\mathrm{Im}(a_\chi(s)) = |a_\chi(s)|^2$.

$$
\begin{aligned}
&\qquad [(1,\ C\_12\_34)]),\ D\_Alpha\_ZZWW1\_S);\\
&((Wp,\ Z,\ Z,\ Wm),\ Vector4\_K\_Matrix\_jr\ (1,\\
&\qquad [(1,\ C\_13\_42)]),\ D\_Alpha\_ZZWW1\_U);\\
&((Wp,\ Z,\ Z,\ Wm),\ Vector4\_K\_Matrix\_jr\ (1,\\
&\qquad [(1,\ C\_14\_23)]),\ D\_Alpha\_ZZWW1\_T);\\
&((Z,\ Wp,\ Wm,\ Z),\ Vector4\_K\_Matrix\_jr\ (2,\\
&\qquad [(1,\ C\_12\_34)]),\ D\_Alpha\_ZZWW1\_S);\\
&((Z,\ Wp,\ Wm,\ Z),\ Vector4\_K\_Matrix\_jr\ (2,\\
&\qquad [(1,\ C\_13\_42)]),\ D\_Alpha\_ZZWW1\_U);\\
&((Z,\ Wp,\ Wm,\ Z),\ Vector4\_K\_Matrix\_jr\ (2,\\
&\qquad [(1,\ C\_14\_23)]),\ D\_Alpha\_ZZWW1\_T);\\
&((Z,\ Z,\ Z,\ Z),\ Vector4\_K\_Matrix\_jr\ (0,\\
&\qquad [(1,\ C\_12\_34)]),\ D\_Alpha\_ZZZZ\_S);\\
&((Z,\ Z,\ Z,\ Z),\ Vector4\_K\_Matrix\_jr\ (0,\\
&\qquad [(1,\ C\_13\_42);\ (1,\ C\_14\_23)]),\ D\_Alpha\_ZZZZ\_T);\\
&((Z,\ Z,\ Z,\ Z),\ Vector4\_K\_Matrix\_jr\ (3,\\
&\qquad [(1,\ C\_14\_23)]),\ D\_Alpha\_ZZZZ\_S);\\
&((Z,\ Z,\ Z,\ Z),\ Vector4\_K\_Matrix\_jr\ (3,\\
&\qquad [(1,\ C\_13\_42);\ (1,\ C\_12\_34)]),\ D\_Alpha\_ZZZZ\_T)]
\end{aligned}
$$

```
    else
      []

  let quartic_gauge =
    standard_quartic_gauge @ anomalous_quartic_gauge @ k_matrix_quartic_gauge

  let standard_gauge_higgs =
    [ ((O H, G Wp, G Wm), Scalar_Vector_Vector 1, G_HWW);
      ((O H, G Z, G Z), Scalar_Vector_Vector 1, G_HZZ) ]

  let standard_gauge_higgs4 =
    [ (O H, O H, G Wp, G Wm), Scalar2_Vector2 1, G_HHWW;
      (O H, O H, G Z, G Z), Scalar2_Vector2 1, G_HHZZ ]

  let standard_higgs =
    [ (O H, O H, O H), Scalar_Scalar_Scalar 1, G_H3 ]

  let standard_higgs4 =
    [ (O H, O H, O H, O H), Scalar4 1, G_H4 ]
```

WK's couplings (apparently, he still intends to divide by $\Lambda^2_{\text{EWSB}} = 16\pi^2 v_{\text{F}}^2$):

$$\mathcal{L}_4^\tau = \left[(\partial_\mu H)(\partial^\mu H) + \frac{g^2 v_{\text{F}}^2}{4} V_\mu V^\mu\right]^2 \tag{13.22a}$$

$$\mathcal{L}_5^\tau = \left[(\partial_\mu H)(\partial_\nu H) + \frac{g^2 v_{\text{F}}^2}{4} V_\mu V_\nu\right]^2 \tag{13.22b}$$

with

$$V_\mu V_\nu = \frac{1}{2}\left(W_\mu^+ W_\nu^- + W_\nu^+ W_\mu^-\right) + \frac{1}{2\cos^2\theta_w} Z_\mu Z_\nu \tag{13.23}$$

(note the symmetrization!), i. e.

$$\mathcal{L}_4 = \alpha_4 \frac{g^4 v_{\text{F}}^4}{16} (V_\mu V_\nu)^2 \tag{13.24a}$$

$$\mathcal{L}_5 = \alpha_5 \frac{g^4 v_F^4}{16} (V_\mu V^\mu)^2 \tag{13.24b}$$

Breaking thinks up

$$\mathcal{L}_4^{\tau,H^4} = [(\partial_\mu H)(\partial^\mu H)]^2 \tag{13.25a}$$
$$\mathcal{L}_5^{\tau,H^4} = [(\partial_\mu H)(\partial^\mu H)]^2 \tag{13.25b}$$

and

$$\mathcal{L}_4^{\tau,H^2V^2} = \frac{g^2 v_F^2}{2} (\partial_\mu H)(\partial^\mu H) V_\mu V^\mu \tag{13.26a}$$
$$\mathcal{L}_5^{\tau,H^2V^2} = \frac{g^2 v_F^2}{2} (\partial_\mu H)(\partial_\nu H) V_\mu V_\nu \tag{13.26b}$$

i. e.

$$\mathcal{L}_4^{\tau,H^2V^2} = \frac{g^2 v_F^2}{2} \left[(\partial_\mu H)(\partial^\mu H) W_\nu^+ W^{-,\nu} + \frac{1}{2\cos^2\theta_w} (\partial_\mu H)(\partial^\mu H) Z_\nu Z^\nu \right] \tag{13.27a}$$
$$\mathcal{L}_5^{\tau,H^2V^2} = \frac{g^2 v_F^2}{2} \left[(W^{+,\mu}\partial_\mu H)(W^{-,\nu}\partial_\nu H) + \frac{1}{2\cos^2\theta_w} (Z^\mu \partial_\mu H)(Z^\nu \partial_\nu H) \right] \tag{13.27b}$$

$$\begin{aligned} &\tau_8^4 \mathcal{L}_4^{\tau,H^2V^2} + \tau_8^5 \mathcal{L}_5^{\tau,H^2V^2} = \\ &\quad - \frac{g^2 v_F^2}{2} \Bigg[2\tau_8^4 \frac{1}{2} (\mathrm{i}\partial_\mu H)(\mathrm{i}\partial^\mu H) W_\nu^+ W^{-,\nu} + \tau_8^5 (W^{+,\mu}\mathrm{i}\partial_\mu H)(W^{-,\nu}\mathrm{i}\partial_\nu H) \\ &\quad + \frac{2\tau_8^4}{\cos^2\theta_w} \frac{1}{4} (\mathrm{i}\partial_\mu H)(\mathrm{i}\partial^\mu H) Z_\nu Z^\nu + \frac{\tau_8^5}{\cos^2\theta_w} \frac{1}{2} (Z^\mu \mathrm{i}\partial_\mu H)(Z^\nu \mathrm{i}\partial_\nu H) \Bigg] \end{aligned} \tag{13.28}$$

where the two powers of i make the sign conveniently negative, i. e.

$$\alpha^2_{(\partial H)^2 W^2} = \tau_8^4 g^2 v_F^2 \tag{13.29a}$$
$$\alpha^2_{(\partial H W)^2} = \frac{\tau_8^5 g^2 v_F^2}{2} \tag{13.29b}$$
$$\alpha^2_{(\partial H)^2 Z^2} = \frac{\tau_8^4 g^2 v_F^2}{\cos^2\theta_w} \tag{13.29c}$$
$$\alpha^2_{(\partial H Z)^2} = \frac{\tau_8^5 g^2 v_F^2}{2\cos^2\theta_w} \tag{13.29d}$$

```
let anomalous_gauge_higgs =
  [ (O H, G Ga, G Ga), Dim5_Scalar_Gauge2 1, G_HGaGa_anom;
    (O H, G Ga, G Z), Dim5_Scalar_Gauge2 1, G_HGaZ_anom;
    (O H, G Z, G Z), Dim5_Scalar_Gauge2 1, G_HZZ_anom;
    (O H, G Wp, G Wm), Dim5_Scalar_Gauge2 1, G_HWW_anom;
    (O H, G Ga, G Z), Dim5_Scalar_Vector_Vector_TU 1, G_HGaZ_u;
    (O H, G Z, G Z), Dim5_Scalar_Vector_Vector_U 1, G_HZZ_u;
```

```
    (O H, G Wp, G Wm), Dim5_Scalar_Vector_Vector_U 1, G_HWW_u
  ]

let anomalous_gauge_higgs4 =
  []

let anomalous_higgs =
  []

let higgs_triangle_vertices =
  if Flags.higgs_triangle then
    [ (O H, G Ga, G Ga), Dim5_Scalar_Gauge2 1, G_HGaGa;
      (O H, G Ga, G Z), Dim5_Scalar_Gauge2 1, G_HGaZ;
      (O H, G Gl, G Gl), Dim5_Scalar_Gauge2 1, G_Hgg ]
  else
    []

let anomalous_higgs4 =
  []

let gauge_higgs =
  if Flags.higgs_anom then
    standard_gauge_higgs @ anomalous_gauge_higgs
  else
    standard_gauge_higgs

let gauge_higgs4 =
  if Flags.higgs_anom then
    standard_gauge_higgs4 @ anomalous_gauge_higgs4
  else
    standard_gauge_higgs4

let higgs =
  if Flags.higgs_anom then
    standard_higgs @ anomalous_higgs
  else
    standard_higgs

let higgs4 =
  if Flags.higgs_anom then
    standard_higgs4 @ anomalous_higgs4
  else
    standard_higgs4

let goldstone_vertices =
  [ ((O Phi0, G Wm, G Wp), Scalar_Vector_Vector 1, I_G_ZWW);
    ((O Phip, G Ga, G Wm), Scalar_Vector_Vector 1, I_Q_W);
    ((O Phip, G Z, G Wm), Scalar_Vector_Vector 1, I_G_ZWW);
    ((O Phim, G Wp, G Ga), Scalar_Vector_Vector 1, I_Q_W);
    ((O Phim, G Wp, G Z), Scalar_Vector_Vector 1, I_G_ZWW) ]
```

Anomalous trilinear interactions $f_i f_j V$ and ttH:

$$\Delta\mathcal{L}_{tt\gamma} = -e\frac{v}{\Lambda^2}\bar{t}i\sigma^{\mu\nu}k_\nu(d_V(k^2) + id_A(k^2)\gamma_5)tA_\mu \tag{13.30}$$

let *anomalous_ttA* =
  if *Flags.top_anom* then
    [((*M* (*U* (−3)), *G Ga*, *M* (*U* 3)), *FBF* (1, *Psibar*, *TVAM*, *Psi*), *G_TVA_ttA*)]
  else
    []

let *tt_threshold_ttA* =
  if *Flags.tt_threshold* then
    [((*M* (*U* (−3)), *G Ga*, *M* (*U* 3)), *FBF* (1, *Psibar*, *VAM*, *Psi*), *VA_ILC_ttA*)]
  else
    []

$$\Delta\mathcal{L}_{bb\gamma} = -e\frac{v}{\Lambda^2}\bar{b}i\sigma^{\mu\nu}k_\nu(d_V(k^2) + id_A(k^2)\gamma_5)bA_\mu \tag{13.31}$$

let *anomalous_bbA* =
  if *Flags.top_anom* then
    [((*M* (*D* (−3)), *G Ga*, *M* (*D* 3)), *FBF* (1, *Psibar*, *TVAM*, *Psi*), *G_TVA_bbA*)]
  else
    []

$$\Delta\mathcal{L}_{ttg} = -g_s\frac{v}{\Lambda^2}\bar{t}\lambda^a i\sigma^{\mu\nu}k_\nu(d_V(k^2) + id_A(k^2)\gamma_5)tG^a_\mu \tag{13.32}$$

let *anomalous_ttG* =
  if *Flags.top_anom* then
    [((*M* (*U* (−3)), *G Gl*, *M* (*U* 3)), *FBF* (1, *Psibar*, *TVAM*, *Psi*), *G_TVA_ttG*)]
  else
    []

$$\Delta\mathcal{L}_{ttZ} = -\frac{g}{2c_W}\frac{v^2}{\Lambda^2}\left[\bar{t}\not{Z}(X_L(k^2)P_L + X_R(k^2)P_R)t + \bar{t}\frac{i\sigma^{\mu\nu}k_\nu}{m_Z}(d_V(k^2) + id_A(k^2)\gamma_5)tZ_\mu\right] \tag{13.33}$$

let *anomalous_ttZ* =
  if *Flags.top_anom* then
    [((*M* (*U* (−3)), *G Z*, *M* (*U* 3)), *FBF* (1, *Psibar*, *VLRM*, *Psi*), *G_VLR_ttZ*);
      ((*M* (*U* (−3)), *G Z*, *M* (*U* 3)), *FBF* (1, *Psibar*, *TVAM*, *Psi*), *G_TVA_ttZ*)]
  else
    []

let *tt_threshold_ttZ* =
  if *Flags.tt_threshold* then
    [((*M* (*U* (−3)), *G Z*, *M* (*U* 3)), *FBF* (1, *Psibar*, *VAM*, *Psi*), *VA_ILC_ttZ*)]
  else
    []

$$\Delta\mathcal{L}_{bbZ} = -\frac{g}{2c_W}\frac{v^2}{\Lambda^2}\bar{b}\frac{i\sigma^{\mu\nu}k_\nu}{m_Z}(d_V(k^2) + id_A(k^2)\gamma_5)bZ_\mu \tag{13.34}$$

let *anomalous_bbZ* =

```
if Flags.top_anom then
  [ ((M (D (−3)), G Z, M (D 3)), FBF (1, Psibar, TVAM, Psi), G_TVA_bbZ) ]
else
  []
```

$$\Delta\mathcal{L}_{tbW} = -\frac{g}{\sqrt{2}}\frac{\upsilon^2}{\Lambda^2}\left[\bar{b}\not{W}^-(V_L(k^2)P_L + V_R(k^2)P_R)t + \bar{b}\frac{i\sigma^{\mu\nu}k_\nu}{m_W}(g_L(k^2)P_L + g_R(k^2)P_R)tW_\mu^-\right] + \text{H.c.} \tag{13.35}$$

```
let anomalous_tbW =
  if Flags.top_anom then
    [ ((M (D (−3)), G Wm, M (U 3)), FBF (1, Psibar, VLRM, Psi), G_VLR_btW);
      ((M (U (−3)), G Wp, M (D 3)), FBF (1, Psibar, VLRM, Psi), G_VLR_tbW);
      ((M (D (−3)), G Wm, M (U 3)), FBF (1, Psibar, TLRM, Psi), G_TLR_btW);
      ((M (U (−3)), G Wp, M (D 3)), FBF (1, Psibar, TRLM, Psi), G_TRL_tbW) ]
  else
    []
```

$$\Delta\mathcal{L}_{ttH} = -\frac{1}{\sqrt{2}}\bar{t}(Y_V(k^2) + iY_A(k^2)\gamma_5)tH \tag{13.36}$$

```
let anomalous_ttH =
  if Flags.top_anom then
    [ ((M (U (−3)), O H, M (U 3)), FBF (1, Psibar, SPM, Psi), G_SP_ttH) ]
  else
    []
```

quartic fermion-gauge interactions $f_i f_j V_1 V_2$ emerging from gauge-invariant effective operators:

$$\Delta\mathcal{L}_{ttgg} = -\frac{g_s^2}{2}f_{abc}\frac{\upsilon}{\Lambda^2}\bar{t}\lambda^a\sigma^{\mu\nu}(d_V(k^2) + id_A(k^2)\gamma_5)tG_\mu^b G_\nu^c \tag{13.37}$$

```
let anomalous_ttGG =
  if Flags.top_anom then
    [ ((M (U (−3)), O (Aux_top (2, 1, 0, true, TTGG)), M (U 3)), FBF (1, Psibar, TVA, Psi), G_TVA
      ((O (Aux_top (2, 1, 0, false, TTGG)), G Gl, G Gl), Aux_Gauge_Gauge 1, I_Gs) ]
  else
    []
```

$$\Delta\mathcal{L}_{tbWA} = -i\sin\theta_w\frac{g^2}{2\sqrt{2}}\frac{\upsilon^2}{\Lambda^2}\left[\bar{b}\frac{\sigma^{\mu\nu}}{m_W}(g_L(k^2)P_L + g_R(k^2)P_R)tA_\mu W_\nu^-\right] + \text{H.c.} \tag{13.38}$$

```
let anomalous_tbWA =
  if Flags.top_anom then
    [ ((M (D (−3)), O (Aux_top (2, 0, −1, true, TBWA)), M (U 3)), FBF (1, Psibar, TLR, Psi), G_TL
      ((O (Aux_top (2, 0, 1, false, TBWA)), G Ga, G Wm), Aux_Gauge_Gauge 1, I_G_weak);
      ((M (U (−3)), O (Aux_top (2, 0, 1, true, TBWA)), M (D 3)), FBF (1, Psibar, TRL, Psi), G_TRI
```

```
      ((O (Aux_top (2, 0, −1,false, TBWA)), G Wp, G Ga), Aux_Gauge_Gauge 1, I_G_weak) ]
  else
    [ ]
```

$$\Delta\mathcal{L}_{tbWZ} = -i\cos\theta_w \frac{g^2}{2\sqrt{2}}\frac{v^2}{\Lambda^2}\left[\bar{b}\frac{\sigma^{\mu\nu}}{m_W}(g_L(k^2)P_L + g_R(k^2)P_R)tZ_\mu W^-_\nu\right] + \text{H.c.} \tag{13.39}$$

```
let anomalous_tbWZ =
  if Flags.top_anom then
    [ ((M (D (−3)), O (Aux_top (2, 0, −1,true, TBWZ)), M (U 3)), FBF (1, Psibar, TLR, Psi), G_TL
      ((O (Aux_top (2, 0, 1,false, TBWZ)), G Z, G Wm), Aux_Gauge_Gauge 1, I_G_weak);
      ((M (U (−3)), O (Aux_top (2, 0, 1,true, TBWZ)), M (D 3)), FBF (1, Psibar, TRL, Psi), G_TR
      ((O (Aux_top (2, 0, −1,false, TBWZ)), G Wp, G Z), Aux_Gauge_Gauge 1, I_G_weak) ]
  else
    [ ]
```

$$\Delta\mathcal{L}_{ttWW} = -i\frac{g^2}{2}\frac{v^2}{\Lambda^2}\bar{t}\frac{\sigma^{\mu\nu}}{m_W}(d_V(k^2) + id_A(k^2)\gamma_5)tW^-_\mu W^+_\nu \tag{13.40}$$

```
let anomalous_ttWW =
  if Flags.top_anom then
    [ ((M (U (−3)), O (Aux_top (2, 0, 0,true, TTWW)), M (U 3)), FBF (1, Psibar, TVA, Psi), G_TV
      ((O (Aux_top (2, 0, 0,false, TTWW)), G Wm, G Wp), Aux_Gauge_Gauge 1, I_G_weak) ]
  else
    [ ]
```

$$\Delta\mathcal{L}_{bbWW} = -i\frac{g^2}{2}\frac{v^2}{\Lambda^2}\bar{b}\frac{\sigma^{\mu\nu}}{m_W}(d_V(k^2) + id_A(k^2)\gamma_5)bW^-_\mu W^+_\nu \tag{13.41}$$

```
let anomalous_bbWW =
  if Flags.top_anom then
    [ ((M (D (−3)), O (Aux_top (2, 0, 0,true, BBWW)), M (D 3)), FBF (1, Psibar, TVA, Psi), G_TV
      ((O (Aux_top (2, 0, 0,false, BBWW)), G Wm, G Wp), Aux_Gauge_Gauge 1, I_G_weak) ]
  else
    [ ]
```

4-fermion contact terms emerging from operator rewriting:

```
let anomalous_top_qGuG_tt =
  [ ((M (U (−3)), O (Aux_top (1, 1, 0,true, QGUG)), M (U 3)), FBF (1, Psibar, VLR, Psi), G_VLR_q

let anomalous_top_qGuG_ff n =
  List.map mom
    [ ((U (−n), Aux_top (1, 1, 0,false, QGUG), U n), FBF (1, Psibar, V, Psi), Unit);
      ((D (−n), Aux_top (1, 1, 0,false, QGUG), D n), FBF (1, Psibar, V, Psi), Unit) ]

let anomalous_top_qGuG =
  if Flags.top_anom_4f then
    anomalous_top_qGuG_tt @ ThoList.flatmap anomalous_top_qGuG_ff [1; 2; 3]
  else
```

```
    []

let anomalous_top_qBuB_tt =
  [ ((M (U (−3)), O (Aux_top (1, 0, 0,true,QBUB)), M (U 3)), FBF (1, Psibar, VLR, Psi), G_VLR_q

let anomalous_top_qBuB_ff n =
  List.map mom
    [ ((U (−n), Aux_top (1, 0, 0,false,QBUB), U n), FBF (1, Psibar, VLR, Psi), G_VLR_qBuB_u);
      ((D (−n), Aux_top (1, 0, 0,false,QBUB), D n), FBF (1, Psibar, VLR, Psi), G_VLR_qBuB_d);
      ((L (−n), Aux_top (1, 0, 0,false,QBUB), L n), FBF (1, Psibar, VLR, Psi), G_VLR_qBuB_e);
      ((N (−n), Aux_top (1, 0, 0,false,QBUB), N n), FBF (1, Psibar, VL, Psi), G_VL_qBuB_n) ]

let anomalous_top_qBuB =
  if Flags.top_anom_4f then
    anomalous_top_qBuB_tt @ ThoList.flatmap anomalous_top_qBuB_ff [1; 2; 3]
  else
    []

let anomalous_top_qW_tq =
  [ ((M (U (−3)), O (Aux_top (1, 0, 0,true,QW)), M (U 3)), FBF (1, Psibar, VL, Psi), G_VL_qW);
    ((M (D (−3)), O (Aux_top (1, 0, −1,true,QW)), M (U 3)), FBF (1, Psibar, VL, Psi), G_VL_qW)
    ((M (U (−3)), O (Aux_top (1, 0, 1,true,QW)), M (D 3)), FBF (1, Psibar, VL, Psi), G_VL_qW) ]

let anomalous_top_qW_ff n =
  List.map mom
    [ ((U (−n), Aux_top (1, 0, 0,false,QW), U n), FBF (1, Psibar, VL, Psi), G_VL_qW_u);
      ((D (−n), Aux_top (1, 0, 0,false,QW), D n), FBF (1, Psibar, VL, Psi), G_VL_qW_d);
      ((N (−n), Aux_top (1, 0, 0,false,QW), N n), FBF (1, Psibar, VL, Psi), G_VL_qW_u);
      ((L (−n), Aux_top (1, 0, 0,false,QW), L n), FBF (1, Psibar, VL, Psi), G_VL_qW_d);
      ((D (−n), Aux_top (1, 0, −1,false,QW), U n), FBF (1, Psibar, VL, Psi), Half);
      ((U (−n), Aux_top (1, 0, 1,false,QW), D n), FBF (1, Psibar, VL, Psi), Half);
      ((L (−n), Aux_top (1, 0, −1,false,QW), N n), FBF (1, Psibar, VL, Psi), Half);
      ((N (−n), Aux_top (1, 0, 1,false,QW), L n), FBF (1, Psibar, VL, Psi), Half) ]

let anomalous_top_qW =
  if Flags.top_anom_4f then
    anomalous_top_qW_tq @ ThoList.flatmap anomalous_top_qW_ff [1; 2; 3]
  else
    []

let anomalous_top_DuDd =
  if Flags.top_anom_4f then
    [ ((M (U (−3)), O (Aux_top (0, 0, 0,true,DR)), M (U 3)), FBF (1, Psibar, SR, Psi), Half);
      ((M (U (−3)), O (Aux_top (0, 0, 0,false,DR)), M (U 3)), FBF (1, Psibar, SL, Psi), G_SL_DttR
      ((M (D (−3)), O (Aux_top (0, 0, 0,false,DR)), M (D 3)), FBF (1, Psibar, SR, Psi), G_SR_DttR
      ((M (U (−3)), O (Aux_top (0, 0, 0,true,DL)), M (U 3)), FBF (1, Psibar, SL, Psi), Half);
      ((M (D (−3)), O (Aux_top (0, 0, 0,false,DL)), M (D 3)), FBF (1, Psibar, SL, Psi), G_SL_DttL)
      ((M (D (−3)), O (Aux_top (0, 0, −1,true,DR)), M (U 3)), FBF (1, Psibar, SR, Psi), Half);
      ((M (U (−3)), O (Aux_top (0, 0, 1,false,DR)), M (D 3)), FBF (1, Psibar, SLR, Psi), G_SLR_D
      ((M (D (−3)), O (Aux_top (0, 0, −1,true,DL)), M (U 3)), FBF (1, Psibar, SL, Psi), Half);
      ((M (U (−3)), O (Aux_top (0, 0, 1,false,DL)), M (D 3)), FBF (1, Psibar, SL, Psi), G_SL_DbtL)
  else
    []
```

```
let anomalous_top_quqd1_tq =
  [ ((M (D (−3)), O (Aux_top (0, 0, −1, true, QUQD1R)), M (U 3)), FBF (1, Psibar, SR, Psi), C_quqd
    ((M (U (−3)), O (Aux_top (0, 0, 1, true, QUQD1R)), M (D 3)), FBF (1, Psibar, SL, Psi), C_quqd
    ((M (D (−3)), O (Aux_top (0, 0, −1, true, QUQD1L)), M (U 3)), FBF (1, Psibar, SL, Psi), C_quqd
    ((M (U (−3)), O (Aux_top (0, 0, 1, true, QUQD1L)), M (D 3)), FBF (1, Psibar, SR, Psi), C_quqd

let anomalous_top_quqd1_ff n =
  List.map mom
    [ ((U (−n), Aux_top (0, 0, 1, false, QUQD1R), D n), FBF (1, Psibar, SR, Psi), Half);
      ((D (−n), Aux_top (0, 0, −1, false, QUQD1R), U n), FBF (1, Psibar, SL, Psi), Half);
      ((U (−n), Aux_top (0, 0, 1, false, QUQD1L), D n), FBF (1, Psibar, SL, Psi), Half);
      ((D (−n), Aux_top (0, 0, −1, false, QUQD1L), U n), FBF (1, Psibar, SR, Psi), Half) ]

let anomalous_top_quqd1 =
  if Flags.top_anom_4f then
    anomalous_top_quqd1_tq @ ThoList.flatmap anomalous_top_quqd1_ff [1; 2; 3]
  else
    []

let anomalous_top_quqd8_tq =
  [ ((M (D (−3)), O (Aux_top (0, 1, −1, true, QUQD8R)), M (U 3)), FBF (1, Psibar, SR, Psi), C_quqd
    ((M (U (−3)), O (Aux_top (0, 1, 1, true, QUQD8R)), M (D 3)), FBF (1, Psibar, SL, Psi), C_quqd
    ((M (D (−3)), O (Aux_top (0, 1, −1, true, QUQD8L)), M (U 3)), FBF (1, Psibar, SL, Psi), C_quqd
    ((M (U (−3)), O (Aux_top (0, 1, 1, true, QUQD8L)), M (D 3)), FBF (1, Psibar, SR, Psi), C_quqd

let anomalous_top_quqd8_ff n =
  List.map mom
    [ ((U (−n), Aux_top (0, 1, 1, false, QUQD8R), D n), FBF (1, Psibar, SR, Psi), Half);
      ((D (−n), Aux_top (0, 1, −1, false, QUQD8R), U n), FBF (1, Psibar, SL, Psi), Half);
      ((U (−n), Aux_top (0, 1, 1, false, QUQD8L), D n), FBF (1, Psibar, SL, Psi), Half);
      ((D (−n), Aux_top (0, 1, −1, false, QUQD8L), U n), FBF (1, Psibar, SR, Psi), Half) ]

let anomalous_top_quqd8 =
  if Flags.top_anom_4f then
    anomalous_top_quqd8_tq @ ThoList.flatmap anomalous_top_quqd8_ff [1; 2; 3]
  else
    []

let vertices3 =
  (ThoList.flatmap electromagnetic_currents [1; 2; 3] @
   ThoList.flatmap color_currents [1; 2; 3] @
   ThoList.flatmap neutral_currents [1; 2; 3] @
   (if Flags.ckm_present then
      charged_currents_ckm
    else
      charged_currents_triv) @
   yukawa @ triple_gauge @
   gauge_higgs @ higgs @ higgs_triangle_vertices
   @ goldstone_vertices @
   tt_threshold_ttA @ tt_threshold_ttZ @
   anomalous_ttA @ anomalous_bbA @
   anomalous_ttZ @ anomalous_bbZ @
   anomalous_tbW @ anomalous_tbWA @ anomalous_tbWZ @
```

```
      anomalous_ttWW @ anomalous_bbWW @
      anomalous_ttG @ anomalous_ttGG @
      anomalous_ttH @
      anomalous_top_qGuG @ anomalous_top_qBuB @
      anomalous_top_qW @ anomalous_top_DuDd @
      anomalous_top_quqd1 @ anomalous_top_quqd8)

  let vertices4 =
    quartic_gauge @ gauge_higgs4 @ higgs4

  let vertices () = (vertices3, vertices4, [])
```

For efficiency, make sure that $F.of\_vertices\ vertices$ is evaluated only once.

```
  let table = F.of_vertices (vertices ())
  let fuse2 = F.fuse2 table
  let fuse3 = F.fuse3 table
  let fuse = F.fuse table
  let max_degree () = 4

  let flavor_of_string = function
    | "e-" → M (L 1) | "e+" → M (L (−1))
    | "mu-" → M (L 2) | "mu+" → M (L (−2))
    | "tau-" → M (L 3) | "tau+" → M (L (−3))
    | "nue" → M (N 1) | "nuebar" → M (N (−1))
    | "numu" → M (N 2) | "numubar" → M (N (−2))
    | "nutau" → M (N 3) | "nutaubar" → M (N (−3))
    | "u" → M (U 1) | "ubar" → M (U (−1))
    | "c" → M (U 2) | "cbar" → M (U (−2))
    | "t" → M (U 3) | "tbar" → M (U (−3))
    | "d" → M (D 1) | "dbar" → M (D (−1))
    | "s" → M (D 2) | "sbar" → M (D (−2))
    | "b" → M (D 3) | "bbar" → M (D (−3))
    | "g" | "gl" → G Gl
    | "A" → G Ga | "Z" | "Z0" → G Z
    | "W+" → G Wp | "W-" → G Wm
    | "H" → O H
    | "Aux_t_ttGG0" → O (Aux_top (2,1, 0,true,TTGG)) | "Aux_ttGG0" →
 O (Aux_top (2,1, 0,false,TTGG))
    | "Aux_t_tbWA+" → O (Aux_top (2,0, 1,true,TBWA)) | "Aux_tbWA+" →
 O (Aux_top (2,0, 1,false,TBWA))
    | "Aux_t_tbWA-" → O (Aux_top (2,0,−1,true,TBWA)) | "Aux_tbWA-" →
 O (Aux_top (2,0,−1,false,TBWA))
    | "Aux_t_tbWZ+" → O (Aux_top (2,0, 1,true,TBWZ)) | "Aux_tbWZ+" →
 O (Aux_top (2,0, 1,false,TBWZ))
    | "Aux_t_tbWZ-" → O (Aux_top (2,0,−1,true,TBWZ)) | "Aux_tbWZ-" →
 O (Aux_top (2,0,−1,false,TBWZ))
    | "Aux_t_ttWW0" → O (Aux_top (2,0, 0,true,TTWW)) | "Aux_ttWW0" →
 O (Aux_top (2,0, 0,false,TTWW))
    | "Aux_t_bbWW0" → O (Aux_top (2,0, 0,true,BBWW)) | "Aux_bbWW0" →
 O (Aux_top (2,0, 0,false,BBWW))
    | "Aux_t_qGuG0" → O (Aux_top (1,1, 0,true,QGUG)) | "Aux_qGuG0" →
 O (Aux_top (1,1, 0,false,QGUG))
```

```
  | "Aux_t_qBuB0" → O (Aux_top (1,0, 0,true,QBUB)) | "Aux_qBuB0" →
O (Aux_top (1,0, 0,false,QBUB))
  | "Aux_t_qW0" → O (Aux_top (1,0, 0,true,QW)) | "Aux_qW0" →
O (Aux_top (1,0, 0,false,QW))
  | "Aux_t_qW+" → O (Aux_top (1,0, 1,true,QW)) | "Aux_qW+" →
O (Aux_top (1,0, 1,false,QW))
  | "Aux_t_qW-" → O (Aux_top (1,0,−1,true,QW)) | "Aux_qW-" →
O (Aux_top (1,0,−1,false,QW))
  | "Aux_t_dL0" → O (Aux_top (0,0, 0,true,DL)) | "Aux_dL0" →
O (Aux_top (0,0, 0,false,DL))
  | "Aux_t_dL+" → O (Aux_top (0,0, 1,true,DL)) | "Aux_dL+" →
O (Aux_top (0,0, 1,false,DL))
  | "Aux_t_dL-" → O (Aux_top (0,0,−1,true,DL)) | "Aux_dL-" →
O (Aux_top (0,0,−1,false,DL))
  | "Aux_t_dR0" → O (Aux_top (0,0, 0,true,DR)) | "Aux_dR0" →
O (Aux_top (0,0, 0,false,DR))
  | "Aux_t_dR+" → O (Aux_top (0,0, 1,true,DR)) | "Aux_dR+" →
O (Aux_top (0,0, 1,false,DR))
  | "Aux_t_dR-" → O (Aux_top (0,0,−1,true,DR)) | "Aux_dR-" →
O (Aux_top (0,0,−1,false,DR))
  | "Aux_t_quqd1L+" → O (Aux_top (0,0, 1,true,QUQD1L)) | "Aux_quqd1L+" →
O (Aux_top (0,0, 1,false,QUQD1L))
  | "Aux_t_quqd1L-" → O (Aux_top (0,0,−1,true,QUQD1L)) | "Aux_quqd1L-" →
O (Aux_top (0,0,−1,false,QUQD1L))
  | "Aux_t_quqd1R+" → O (Aux_top (0,0, 1,true,QUQD1R)) | "Aux_quqd1R+" →
O (Aux_top (0,0, 1,false,QUQD1R))
  | "Aux_t_quqd1R-" → O (Aux_top (0,0,−1,true,QUQD1R)) | "Aux_quqd1R-" →
O (Aux_top (0,0,−1,false,QUQD1R))
  | "Aux_t_quqd8L+" → O (Aux_top (0,1, 1,true,QUQD8L)) | "Aux_quqd8L+" →
O (Aux_top (0,1, 1,false,QUQD8L))
  | "Aux_t_quqd8L-" → O (Aux_top (0,1,−1,true,QUQD8L)) | "Aux_quqd8L-" →
O (Aux_top (0,1,−1,false,QUQD8L))
  | "Aux_t_quqd8R+" → O (Aux_top (0,1, 1,true,QUQD8R)) | "Aux_quqd8R+" →
O (Aux_top (0,1, 1,false,QUQD8R))
  | "Aux_t_quqd8R-" → O (Aux_top (0,1,−1,true,QUQD8R)) | "Aux_quqd8R-" →
O (Aux_top (0,1,−1,false,QUQD8R))
  | _ → invalid_arg "Modellib.SM.flavor_of_string"

let flavor_to_string = function
  | M f →
      begin match f with
      | L 1 → "e-" | L (−1) → "e+"
      | L 2 → "mu-" | L (−2) → "mu+"
      | L 3 → "tau-" | L (−3) → "tau+"
      | L _ → invalid_arg
            "Modellib.SM.flavor_to_string:␣invalid␣lepton"
      | N 1 → "nue" | N (−1) → "nuebar"
      | N 2 → "numu" | N (−2) → "numubar"
      | N 3 → "nutau" | N (−3) → "nutaubar"
      | N _ → invalid_arg
```

```
            "Modellib.SM.flavor_to_string:␣invalid␣neutrino"
        | U 1 → "u" | U (−1) → "ubar"
        | U 2 → "c" | U (−2) → "cbar"
        | U 3 → "t" | U (−3) → "tbar"
        | U _ → invalid_arg
            "Modellib.SM.flavor_to_string:␣invalid␣up␣type␣quark"
        | D 1 → "d" | D (−1) → "dbar"
        | D 2 → "s" | D (−2) → "sbar"
        | D 3 → "b" | D (−3) → "bbar"
        | D _ → invalid_arg
            "Modellib.SM.flavor_to_string:␣invalid␣down␣type␣quark"
        end
    | G f →
        begin match f with
        | Gl → "gl"
        | Ga → "A" | Z → "Z"
        | Wp → "W+" | Wm → "W-"
        end
    | O f →
        begin match f with
        | Phip → "phi+" | Phim → "phi-" | Phi0 → "phi0"
        | H → "H"
        | Aux_top (_, _, ch, n, v) → "Aux_" ^ (if n then "t_" else "") ^ (
            begin match v with
            | TTGG → "ttGG" | TBWA → "tbWA" | TBWZ →
"tbWZ"
            | TTWW → "ttWW" | BBWW → "bbWW"
            | QGUG → "qGuG" | QBUB → "qBuB"
            | QW → "qW" | DL → "dL" | DR → "dR"
            | QUQD1L → "quqd1L" | QUQD1R → "quqd1R"
            | QUQD8L → "quqd8L" | QUQD8R → "quqd8R"
            end ) ^ ( if ch > 0 then "+" else if ch < 0 then "-" else "0" )
        end

  let flavor_to_TeX = function
    | M f →
        begin match f with
        | L 1 → "e^-" | L (−1) → "e^+"
        | L 2 → "\\mu^-" | L (−2) → "\\mu^+"
        | L 3 → "\\tau^-" | L (−3) → "\\tau^+"
        | L _ → invalid_arg
            "Modellib.SM.flavor_to_TeX:␣invalid␣lepton"
        | N 1 → "\\nu_e" | N (−1) → "\\bar{\\nu}_e"
        | N 2 → "\\nu_\\mu" | N (−2) → "\\bar{\\nu}_\\mu"
        | N 3 → "\\nu_\\tau" | N (−3) → "\\bar{\\nu}_\\tau"
        | N _ → invalid_arg
            "Modellib.SM.flavor_to_TeX:␣invalid␣neutrino"
        | U 1 → "u" | U (−1) → "\\bar{u}"
        | U 2 → "c" | U (−2) → "\\bar{c}"
        | U 3 → "t" | U (−3) → "\\bar{t}"
```

```
        | U _ → invalid_arg
              "Modellib.SM.flavor_to_TeX:␣invalid␣up␣type␣quark"
        | D 1 → "d" | D (−1) → "\\bar{d}"
        | D 2 → "s" | D (−2) → "\\bar{s}"
        | D 3 → "b" | D (−3) → "\\bar{b}"
        | D _ → invalid_arg
              "Modellib.SM.flavor_to_TeX:␣invalid␣down␣type␣quark"
        end
    | G f →
        begin match f with
        | Gl → "g"
        | Ga → "\\gamma" | Z → "Z"
        | Wp → "W^+" | Wm → "W^-"
        end
    | O f →
        begin match f with
        | Phip → "\\phi^+" | Phim → "\\phi^-" | Phi0 →
"\\phi^0"
        | H → "H"
        | Aux_top (_, _, ch, n, v) → "\\textnormal{Aux_" ^ (if n then "t_" else "") ^ (
            begin match v with
            | TTGG → "ttGG" | TBWA → "tbWA" | TBWZ →
"tbWZ"
            | TTWW → "ttWW" | BBWW → "bbWW"
            | QGUG → "qGuG" | QBUB → "qBuB"
            | QW → "qW" | DL → "dL" | DR → "dR"
            | QUQD1L → "quqd1L" | QUQD1R → "quqd1R"
            | QUQD8L → "quqd8L" | QUQD8R → "quqd8R"
            end ) ^ ( if ch > 0 then "^+" else if ch < 0 then "^-" else "^0" ) ^ "}"
        end

  let flavor_symbol = function
    | M f →
        begin match f with
        | L n when n > 0 → "l" ^ string_of_int n
        | L n → "l" ^ string_of_int (abs n) ^ "b"
        | N n when n > 0 → "n" ^ string_of_int n
        | N n → "n" ^ string_of_int (abs n) ^ "b"
        | U n when n > 0 → "u" ^ string_of_int n
        | U n → "u" ^ string_of_int (abs n) ^ "b"
        | D n when n > 0 → "d" ^ string_of_int n
        | D n → "d" ^ string_of_int (abs n) ^ "b"
        end
    | G f →
        begin match f with
        | Gl → "gl"
        | Ga → "a" | Z → "z"
        | Wp → "wp" | Wm → "wm"
        end
    | O f →
```

```
        begin match f with
        | Phip → "pp" | Phim → "pm" | Phi0 → "p0"
        | H → "h"
        | Aux_top (_, _, ch, n, v) → "aux_" ^ (if n then "t_" else "") ^ (
            begin match v with
            | TTGG → "ttgg" | TBWA → "tbwa" | TBWZ →
"tbwz"
            | TTWW → "ttww" | BBWW → "bbww"
            | QGUG → "qgug" | QBUB → "qbub"
            | QW → "qw" | DL → "dl" | DR → "dr"
            | QUQD1L → "quqd1l" | QUQD1R → "quqd1r"
            | QUQD8L → "quqd8l" | QUQD8R → "quqd8r"
            end ) ^ "_" ^ ( if ch > 0 then "p" else if ch < 0 then "m" else "0" )
        end

  let pdg = function
    | M f →
        begin match f with
        | L n when n > 0 → 9 + 2 × n
        | L n → − 9 + 2 × n
        | N n when n > 0 → 10 + 2 × n
        | N n → − 10 + 2 × n
        | U n when n > 0 → 2 × n
        | U n → 2 × n
        | D n when n > 0 → − 1 + 2 × n
        | D n → 1 + 2 × n
        end
    | G f →
        begin match f with
        | Gl → 21
        | Ga → 22 | Z → 23
        | Wp → 24 | Wm → (−24)
        end
    | O f →
        begin match f with
        | Phip | Phim → 27 | Phi0 → 26
        | H → 25
        | Aux_top (_, _, ch, t, f) → let n =
          begin match f with
          | QW → 0
          | QUQD1R → 1 | QUQD1L → 2
          | QUQD8R → 3 | QUQD8L → 4
          | _ → 5
          end
          in (602 + 3 × n − ch) × ( if t then (1) else (−1) )
        end

  let mass_symbol f =
    if ( Flags.tt_threshold ∧ (abs (pdg f)) ≡ 6 ) then
      "ttv_mtpole(p12*p12)"
    else
```

```
    "mass(" ^ string_of_int (abs (pdg f)) ^ ")"

  let width_symbol f =
    "width(" ^ string_of_int (abs (pdg f)) ^ ")"

  let constant_symbol = function
    | Unit → "unit" | Half → "half" | Pi → "PI"
    | Alpha_QED → "alpha" | E → "e" | G_weak → "g" | Vev →
"vev"
    | I_G_weak → "ig"
    | Sin2thw → "sin2thw" | Sinthw → "sinthw" | Costhw →
"costhw"
    | Q_lepton → "qlep" | Q_up → "qup" | Q_down → "qdwn"
    | G_NC_lepton → "gnclep" | G_NC_neutrino → "gncneu"
    | G_NC_up → "gncup" | G_NC_down → "gncdwn"
    | G_TVA_ttA → "gtva_tta" | G_TVA_bbA → "gtva_bba"
    | G_VLR_ttZ → "gvlr_ttz" | G_TVA_ttZ → "gtva_ttz" |
G_TVA_bbZ → "gtva_bbz"
    | VA_ILC_ttA → "va_ilc_tta" | VA_ILC_ttZ → "va_ilc_ttz"
    | G_VLR_btW → "gvlr_btw" | G_VLR_tbW → "gvlr_tbw"
    | G_TLR_btW → "gtlr_btw" | G_TRL_tbW → "gtrl_tbw"
    | G_TLR_btWA → "gtlr_btwa" | G_TRL_tbWA → "gtrl_tbwa"
    | G_TLR_btWZ → "gtlr_btwz" | G_TRL_tbWZ → "gtrl_tbwz"
    | G_TVA_ttWW → "gtva_ttww" | G_TVA_bbWW → "gtva_bbww"
    | G_TVA_ttG → "gtva_ttg" | G_TVA_ttGG → "gtva_ttgg"
    | G_SP_ttH → "gsp_tth"
    | G_VLR_qGuG → "gvlr_qgug"
    | G_VLR_qBuB → "gvlr_qbub"
    | G_VLR_qBuB_u → "gvlr_qbub_u" | G_VLR_qBuB_d → "gvlr_qbub_d"
    | G_VLR_qBuB_e → "gvlr_qbub_e" | G_VL_qBuB_n → "gvl_qbub_n"
    | G_VL_qW → "gvl_qw"
    | G_VL_qW_u → "gvl_qw_u" | G_VL_qW_d → "gvl_qw_d"
    | G_SL_DttR → "gsl_dttr" | G_SR_DttR → "gsr_dttr" |
G_SL_DttL → "gsl_dttl"
    | G_SLR_DbtR → "gslr_dbtr" | G_SL_DbtL → "gsl_dbtl"
    | C_quqd1R_bt → "c_quqd1_1" | C_quqd1R_tb → "conjg(c_quqd1_1)"
    | C_quqd1L_bt → "conjg(c_quqd1_2)" | C_quqd1L_tb → "c_quqd1_2"
    | C_quqd8R_bt → "c_quqd8_1" | C_quqd8R_tb → "conjg(c_quqd8_1)"
    | C_quqd8L_bt → "conjg(c_quqd8_2)" | C_quqd8L_tb → "c_quqd8_2"
    | G_CC → "gcc"
    | G_CCQ (n1, n2) → "gccq" ^ string_of_int n1 ^ string_of_int n2
    | I_Q_W → "iqw" | I_G_ZWW → "igzww"
    | G_WWWW → "gw4" | G_ZZWW → "gzzww"
    | G_AZWW → "gazww" | G_AAWW → "gaaww"
    | I_G1_AWW → "ig1a" | I_G1_ZWW → "ig1z"
    | I_G1_plus_kappa_plus_G4_AWW → "ig1pkpg4a"
    | I_G1_plus_kappa_plus_G4_ZWW → "ig1pkpg4z"
    | I_G1_plus_kappa_minus_G4_AWW → "ig1pkmg4a"
    | I_G1_plus_kappa_minus_G4_ZWW → "ig1pkmg4z"
    | I_G1_minus_kappa_plus_G4_AWW → "ig1mkpg4a"
    | I_G1_minus_kappa_plus_G4_ZWW → "ig1mkpg4z"
```

```
    | I_G1_minus_kappa_minus_G4_AWW → "ig1mkmg4a"
    | I_G1_minus_kappa_minus_G4_ZWW → "ig1mkmg4z"
    | I_lambda_AWW → "ila"
    | I_lambda_ZWW → "ilz"
    | G5_AWW → "rg5a"
    | G5_ZWW → "rg5z"
    | I_kappa5_AWW → "ik5a"
    | I_kappa5_ZWW → "ik5z"
    | I_lambda5_AWW → "il5a" | I_lambda5_ZWW → "il5z"
    | Alpha_WWWW0 → "alww0" | Alpha_WWWW2 → "alww2"
    | Alpha_ZZWW0 → "alzw0" | Alpha_ZZWW1 → "alzw1"
    | Alpha_ZZZZ → "alzz"
    | D_Alpha_ZZWW0_S → "dalzz0_s(gkm,mkm,"
    | D_Alpha_ZZWW0_T → "dalzz0_t(gkm,mkm,"
    | D_Alpha_ZZWW1_S → "dalzz1_s(gkm,mkm,"
    | D_Alpha_ZZWW1_T → "dalzz1_t(gkm,mkm,"
    | D_Alpha_ZZWW1_U → "dalzz1_u(gkm,mkm,"
    | D_Alpha_WWWW0_S → "dalww0_s(gkm,mkm,"
    | D_Alpha_WWWW0_T → "dalww0_t(gkm,mkm,"
    | D_Alpha_WWWW0_U → "dalww0_u(gkm,mkm,"
    | D_Alpha_WWWW2_S → "dalww2_s(gkm,mkm,"
    | D_Alpha_WWWW2_T → "dalww2_t(gkm,mkm,"
    | D_Alpha_ZZZZ_S → "dalz4_s(gkm,mkm,"
    | D_Alpha_ZZZZ_T → "dalz4_t(gkm,mkm,"
    | G_HWW → "ghww" | G_HZZ → "ghzz"
    | G_HHWW → "ghhww" | G_HHZZ → "ghhzz"
    | G_Htt → "ghtt" | G_Hbb → "ghbb"
    | G_Htautau → "ghtautau" | G_Hcc → "ghcc" | G_Hmm →
"ghmm"
    | G_HGaZ → "ghgaz" | G_HGaGa → "ghgaga" | G_Hgg →
"ghgg"
    | G_HGaGa_anom → "ghgaga_ac" | G_HGaZ_anom → "ghgaz_ac"
    | G_HZZ_anom → "ghzz_ac" | G_HWW_anom → "ghww_ac"
    | G_HGaZ_u → "ghgaz_u" | G_HZZ_u → "ghzz_u"
    | G_HWW_u → "ghww_u"
    | G_H3 → "gh3" | G_H4 → "gh4"
    | Gs → "gs" | I_Gs → "igs" | G2 → "gs**2"
    | Mass f → "mass" ^ flavor_symbol f
    | Width f → "width" ^ flavor_symbol f
    | K_Matrix_Coeff i → "kc" ^ string_of_int i
    | K_Matrix_Pole i → "kp" ^ string_of_int i

end
```

13.4.7 Incomplete Standard Model in R_ξ Gauge

At the end of the day, we want a functor mapping from gauge models in unitarity gauge to R_ξ gauge and vice versa. For this, we will need a more abstract implementation of (spontaneously broken) gauge theories.

```
module SM_Rxi =
  struct
    let rcs = RCS.rename rcs_file "Modellib.SM_Rxi"
        [ "minimal␣electroweak␣standard␣model␣in␣R-xi␣gauge";
          "NB:␣very␣incomplete␣still!,␣no␣CKM␣matrix" ]

    open Coupling

    module SM = SM(SM_no_anomalous)
    let options = SM.options
    type flavor = SM.flavor
    let flavors = SM.flavors
    let external_flavors = SM.external_flavors
    (* Later: type orders = SM.orders *)
    type constant = SM.constant
    (* Later: let orders = SM.orders *)
    let lorentz = SM.lorentz
    let color = SM.color
    let goldstone = SM.goldstone
    let conjugate = SM.conjugate
    let fermion = SM.fermion
```

Check if it makes sense to have separate gauge fixing parameters for each vector boson. There's probably only one independent parameter for each group factor.

```
    type gauge =
      | XiA | XiZ | XiW

    let gauge_symbol = function
      | XiA → "xia" | XiZ → "xi0" | XiW → "xipm"
```

Change the gauge boson propagators and make the Goldstone bosons propagating.

```
    let propagator = function
      | SM.G SM.Ga → Prop_Gauge XiA
      | SM.G SM.Z → Prop_Rxi XiZ
      | SM.G SM.Wp | SM.G SM.Wm → Prop_Rxi XiW
      | SM.O SM.Phip | SM.O SM.Phim | SM.O SM.Phi0 → Prop_Scalar
      | f → SM.propagator f

    let width = SM.width

    module Ch = Charges.QQ
    let charges = SM.charges

    module F = Modeltools.Fusions (struct
      type f = flavor
      type c = constant
      let compare = compare
      let conjugate = conjugate
    end)
```

```
    let vertices = SM.vertices

    let table = F.of_vertices (vertices ())
    let fuse2 = F.fuse2 table
    let fuse3 = F.fuse3 table
    let fuse = F.fuse table
    let max_degree () = 3

    let parameters = SM.parameters
    let flavor_of_string = SM.flavor_of_string
    let flavor_to_string = SM.flavor_to_string
    let flavor_to_TeX = SM.flavor_to_TeX
    let flavor_symbol = SM.flavor_symbol
    let pdg = SM.pdg
    let mass_symbol = SM.mass_symbol
    let width_symbol = SM.width_symbol
    let constant_symbol = SM.constant_symbol

  end
```

13.4.8 Groves

```
module Groves (M : Model.Gauge) : Model.Gauge with module Ch = M.Ch =
  struct
    let max_generations = 5
    let rcs = RCS.rename M.rcs
        ("Modellib.Groves(" ^ (RCS.name M.rcs) ^ ")")
        ([ "experimental Groves functor";
           Printf.sprintf "for maximally %d flavored legs"
             (2 × max_generations) ] @
         RCS.description M.rcs)

    let options = M.options

    type matter_field = M.matter_field × int
    type gauge_boson = M.gauge_boson
    type other = M.other
    type field =
      | Matter of matter_field
      | Gauge of gauge_boson
      | Other of other
    type flavor = M of matter_field | G of gauge_boson | O of other
    let matter_field (f, g) = M (f, g)
    let gauge_boson f = G f
    let other f = O f
    let field = function
      | M f → Matter f
      | G f → Gauge f
      | O f → Other f
    let project = function
      | M (f, _) → M.matter_field f
      | G f → M.gauge_boson f
```

```
  | O f  →  M.other f
let inject g f  =
  match M.field f with
  | M.Matter f  →  M (f, g)
  | M.Gauge f  →  G f
  | M.Other f  →  O f
type gauge  =  M.gauge
let gauge_symbol  =  M.gauge_symbol
let color f  =  M.color (project f)
let pdg f  =  M.pdg (project f)
let lorentz f  =  M.lorentz (project f)
let propagator f  =  M.propagator (project f)
let fermion f  =  M.fermion (project f)
let width f  =  M.width (project f)
let mass_symbol f  =  M.mass_symbol (project f)
let width_symbol f  =  M.width_symbol (project f)
let flavor_symbol f  =  M.flavor_symbol (project f)

type constant  =  M.constant
(* Later: type orders  =  M.orders *)
let constant_symbol  =  M.constant_symbol
let max_degree  =  M.max_degree
let parameters  =  M.parameters
(* Later: let orders  =  M.orders *)

let conjugate  =  function
  | M (_, g) as f  →  inject g (M.conjugate (project f))
  | f  →  inject 0 (M.conjugate (project f))

let read_generation s  =
  try
    let offset  =  String.index s '/' in
    (int_of_string
       (String.sub s (succ offset) (String.length s  −  offset  −  1)),
     String.sub s 0 offset)
  with
  | Not_found  →  (1, s)

let format_generation c s  =
  s ^ "/" ^ string_of_int c

let flavor_of_string s  =
  let g, s  =  read_generation s in
  inject g (M.flavor_of_string s)

let flavor_to_string  =  function
  | M (_, g) as f  →  format_generation g (M.flavor_to_string (project f))
  | f  →  M.flavor_to_string (project f)

let flavor_to_TeX  =  function
  | M (_, g) as f  →  format_generation g (M.flavor_to_TeX (project f))
  | f  →  M.flavor_to_TeX (project f)

let goldstone  =  function
```

```
| G _ as f →
    begin match M.goldstone (project f) with
    | None → None
    | Some (f, c) → Some (inject 0 f, c)
    end
| M _ | O _ → None

let clone generations flavor =
  match M.field flavor with
  | M.Matter f → List.map (fun g → M (f, g)) generations
  | M.Gauge f → [G f]
  | M.Other f → [O f]

let generations = ThoList.range 1 max_generations

let flavors () =
  ThoList.flatmap (clone generations) (M.flavors ())

let external_flavors () =
  List.map (fun (s, fl) → (s, ThoList.flatmap (clone generations) fl))
    (M.external_flavors ())

module Ch = M.Ch
let charges f = M.charges (project f)

module F = Modeltools.Fusions (struct
  type f = flavor
  type c = constant
  let compare = compare
  let conjugate = conjugate
end)
```

In the following functions, we might replace _ by $(M.Gauge\ \_\ \mid\ M.Other\ \_)$, in order to allow the compiler to check completeness. However, this makes the code much less readable.

```
let clone3 ((f1, f2, f3), v, c) =
  match M.field f1, M.field f2, M.field f3 with
  | M.Matter _, M.Matter _, M.Matter _ →
      invalid_arg "Modellib.Groves().vertices:␣three␣matter␣fields!"
  | M.Matter f1', M.Matter f2', _ →
      List.map (fun g → ((M (f1', g), M (f2', g), inject 0 f3), v, c))
        generations
  | M.Matter f1', _, M.Matter f3' →
      List.map (fun g → ((M (f1', g), inject 0 f2, M (f3', g)), v, c))
        generations
  | _, M.Matter f2', M.Matter f3' →
      List.map (fun g → ((inject 0 f1, M (f2', g), M (f3', g)), v, c))
        generations
  | M.Matter _, _, _ | _, M.Matter _, _ | _, _, M.Matter _ →
      invalid_arg "Modellib.Groves().vertices:␣lone␣matter␣field!"
  | _, _, _ →
      [(inject 0 f1, inject 0 f2, inject 0 f3), v, c]
```

```
let clone4 ((f1, f2, f3, f4), v, c) =
  match M.field f1, M.field f2, M.field f3, M.field f4 with
  | M.Matter _, M.Matter _, M.Matter _, M.Matter _ →
      invalid_arg "Modellib.Groves().vertices: four matter fields!"
  | M.Matter _, M.Matter _, M.Matter _, _
  | M.Matter _, M.Matter _, _, M.Matter _
  | M.Matter _, _, M.Matter _, M.Matter _
  | _, M.Matter _, M.Matter _, M.Matter _ →
      invalid_arg "Modellib.Groves().vertices: three matter fields!"
  | M.Matter f1', M.Matter f2', _, _ →
      List.map (fun g →
        ((M (f1', g), M (f2', g), inject 0 f3, inject 0 f4), v, c))
        generations
  | M.Matter f1', _, M.Matter f3', _ →
      List.map (fun g →
        ((M (f1', g), inject 0 f2, M (f3', g), inject 0 f4), v, c))
        generations
  | M.Matter f1', _, _, M.Matter f4' →
      List.map (fun g →
        ((M (f1', g), inject 0 f2, inject 0 f3, M (f4', g)), v, c))
        generations
  | _, M.Matter f2', M.Matter f3', _ →
      List.map (fun g →
        ((inject 0 f1, M (f2', g), M (f3', g), inject 0 f4), v, c))
        generations
  | _, M.Matter f2', _, M.Matter f4' →
      List.map (fun g →
        ((inject 0 f1, M (f2', g), inject 0 f3, M (f4', g)), v, c))
        generations
  | _, _, M.Matter f3', M.Matter f4' →
      List.map (fun g →
        ((inject 0 f1, inject 0 f2, M (f3', g), M (f4', g)), v, c))
        generations
  | M.Matter _, _, _, _ | _, M.Matter _, _, _
  | _, _, M.Matter _, _ | _, _, _, M.Matter _ →
      invalid_arg "Modellib.Groves().vertices: lone matter field!"
  | _, _, _, _ →
      [(inject 0 f1, inject 0 f2, inject 0 f3, inject 0 f4), v, c]

let clonen (fl, v, c) =
  match List.map M.field fl with
  | _ → failwith "Modellib.Groves().vertices: incomplete"

let vertices () =
  let vertices3, vertices4, verticesn = M.vertices () in
  (ThoList.flatmap clone3 vertices3,
   ThoList.flatmap clone4 vertices4,
   ThoList.flatmap clonen verticesn)

let table = F.of_vertices (vertices ())
let fuse2 = F.fuse2 table
let fuse3 = F.fuse3 table
```

```
let fuse = F.fuse table
```

The following (incomplete) alternative implementations are included for illustrative purposes only:

```
let injectl g fcl =
  List.map (fun (f, c) → (inject g f, c)) fcl

let alt_fuse2 f1 f2 =
  match f1, f2 with
  | M (f1′, g1′), M (f2′, g2′) →
      if g1′ = g2′ then
        injectl 0 (M.fuse2 (M.matter_field f1′) (M.matter_field f2′))
      else
        []
  | M (f1′, g′), _ → injectl g′ (M.fuse2 (M.matter_field f1′) (project f2))
  | _, M (f2′, g′) → injectl g′ (M.fuse2 (project f1) (M.matter_field f2′))
  | _, _ → injectl 0 (M.fuse2 (project f1) (project f2))

let alt_fuse3 f1 f2 f3 =
  match f1, f2, f3 with
  | M (f1′, g1′), M (f2′, g2′), M (f3′, g3′) →
      invalid_arg "Modellib.Groves().fuse3: three matter fields!"
  | M (f1′, g1′), M (f2′, g2′), _ →
      if g1′ = g2′ then
        injectl 0
          (M.fuse3 (M.matter_field f1′) (M.matter_field f2′) (project f3))
      else
        []
  | M (f1′, g1′), _, M (f3′, g3′) →
      if g1′ = g3′ then
        injectl 0
          (M.fuse3 (M.matter_field f1′) (project f2) (M.matter_field f3′))
      else
        []
  | _, M (f2′, g2′), M (f3′, g3′) →
      if g2′ = g3′ then
        injectl 0
          (M.fuse3 (project f1) (M.matter_field f2′) (M.matter_field f3′))
      else
        []
  | M (f1′, g′), _, _ →
      injectl g′ (M.fuse3 (M.matter_field f1′) (project f2) (project f3))
  | _, M (f2′, g′), _ →
      injectl g′ (M.fuse3 (project f1) (M.matter_field f2′) (project f3))
  | _, _, M (f3′, g′) →
      injectl g′ (M.fuse3 (project f1) (project f2) (M.matter_field f3′))
  | _, _, _ → injectl 0 (M.fuse3 (project f1) (project f2) (project f3))

end
```

13.4.9 MSM With Cloned Families

module $SM\_clones = Groves(SM(SM\_no\_anomalous))$

13.5 Interface of Modellib_BSM

13.5.1 More Hardcoded BSM Models

```
module type BSM_flags =
  sig
    val u1_gauged : bool
    val anom_ferm_ass : bool
  end

module type THDM_flags =
  sig
    val ckm_present : bool
  end

module BSM_bsm : BSM_flags
module BSM_ungauged : BSM_flags
module BSM_anom : BSM_flags
module Littlest : functor (F : BSM_flags) → Model.Gauge with module Ch = Charges.QQ
module Littlest_Tpar : functor (F : BSM_flags) → Model.T with module Ch = Charges.QQ
module Simplest : functor (F : BSM_flags) → Model.T with module Ch = Charges.QQ
module Xdim : functor (F : BSM_flags) → Model.Gauge with module Ch = Charges.QQ
module UED : functor (F : BSM_flags) → Model.Gauge with module Ch = Charges.QQ
module GravTest : functor (F : BSM_flags) → Model.Gauge with module Ch = Charges.QQ
module THDM : THDM_flags
module THDM_CKM : THDM_flags
module TwoHiggsDoublet : functor (F : THDM_flags) → Model.Gauge with module Ch = Charges.QQ
module Template : functor (F : BSM_flags) → Model.Gauge with module Ch = Charges.QQ
module HSExt : functor (F : BSM_flags) → Model.Gauge with module Ch = Charges.QQ

module type Threeshl_options =
  sig
    val include_ckm : bool
    val include_hf : bool
    val diet : bool
  end

module Threeshl_no_ckm : Threeshl_options
module Threeshl_ckm : Threeshl_options
```

```
module Threeshl_no_ckm_no_hf : Threeshl_options
module Threeshl_ckm_no_hf : Threeshl_options
module Threeshl_diet_no_hf : Threeshl_options
module Threeshl_diet : Threeshl_options
module Threeshl : functor (Module_options : Threeshl_options) →
  Model.T with module Ch = Charges.QQ
```

```
module type SSC_flags =
  sig
    val higgs_triangle : bool (* Hγγ, Hgγ and Hgg couplings *)
    val higgs_hmm : bool
    val triple_anom : bool
    val quartic_anom : bool
    val higgs_anom : bool
    val k_matrix : bool
    val ckm_present : bool
    val top_anom : bool
    val top_anom_4f : bool
  end
```

```
module SSC_kmatrix : SSC_flags
```

```
module SSC : functor (F : SSC_flags) → Model.Gauge with module Ch = Charges.QQ
```

13.6 Implementation of Modellib_BSM

```
let rcs_file = RCS.parse "Modellib_BSM" ["BSM␣Models"]
    { RCS.revision = "$Revision:␣6465␣$";
      RCS.date = "$Date:␣2015-01-10␣16:22:31␣+0100␣(Sat,␣10␣Jan␣2015)␣$";
      RCS.author = "$Author:␣jr_reuter␣$";
      RCS.source
        = "$URL:␣svn+ssh://jr_reuter@login.hepforge.org/hepforge/svn/whizard/trunk/omega/s
```

13.6.1 Littlest Higgs Model

```
module type BSM_flags =
  sig
    val u1_gauged : bool
    val anom_ferm_ass : bool
  end
```

```
module BSM_bsm : BSM_flags =
  struct
    let u1_gauged = true
    let anom_ferm_ass = false
  end
```

```
module BSM_ungauged : BSM_flags =
  struct
    let u1_gauged = false
```

```
    let anom_ferm_ass = false
  end

module BSM_anom : BSM_flags =
  struct
    let u1_gauged = false
    let anom_ferm_ass = true
  end

module Littlest (Flags : BSM_flags) =
  struct
    let rcs = rcs_file

    open Coupling

    let default_width = ref Timelike
    let use_fudged_width = ref false

    let options = Options.create
      [ "constant_width", Arg.Unit (fun () → default_width := Constant),
        "use␣constant␣width␣(also␣in␣t-channel)";
        "fudged_width", Arg.Set use_fudged_width,
        "use␣fudge␣factor␣for␣charge␣particle␣width";
        "custom_width", Arg.String (fun f → default_width := Custom f),
        "use␣custom␣width";
        "cancel_widths", Arg.Unit (fun () → default_width := Vanishing),
        "use␣vanishing␣width" ]

    let gauge_symbol () =
      failwith "Modellib_BSM.Littlest.gauge_symbol:␣internal␣error"

    type matter_field = L of int | N of int | U of int | D of int
        | TopH | TopHb
    type gauge_boson = Ga | Wp | Wm | Z | Gl | WHp | WHm
        | ZH | AH
    type other = Phip | Phim | Phi0 | H | Eta | Psi0
        | Psi1 | Psip | Psim | Psipp | Psimm

    type flavor = M of matter_field | G of gauge_boson | O of other

    let matter_field f = M f
    let gauge_boson f = G f
    let other f = O f

    type field =
      | Matter of matter_field
      | Gauge of gauge_boson
      | Other of other

    let field = function
      | M f → Matter f
      | G f → Gauge f
      | O f → Other f

    type gauge = unit
```

```
let gauge_symbol () =
  failwith "Modellib_BSM.Littlest.gauge_symbol: internal error"

let family n = List.map matter_field [ L n; N n; U n; D n ]
```

Since *Phi* already belongs to the EW Goldstone bosons we use *Psi* for the TeV scale complex triplet.

```
let external_flavors () =
  [ "1st Generation", ThoList.flatmap family [1; -1];
    "2nd Generation", ThoList.flatmap family [2; -2];
    "3rd Generation", ThoList.flatmap family [3; -3];
    "Heavy Quarks", List.map matter_field [TopH; TopHb];
    "Heavy Scalars", List.map other
      [Psi0; Psi1; Psip; Psim; Psipp; Psimm];
    "Gauge Bosons", List.map gauge_boson
      (if Flags.u1_gauged then
       [Ga; Z; Wp; Wm; Gl; WHp; WHm; ZH; AH]
           else
       [Ga; Z; Wp; Wm; Gl; WHp; WHm; ZH]);
    "Higgs", List.map other
      (if Flags.u1_gauged then [H]
           else [H; Eta]);
    "Goldstone Bosons", List.map other [Phip; Phim; Phi0] ]

let flavors () = ThoList.flatmap snd (external_flavors ())

let spinor n =
  if n ≥ 0 then
    Spinor
  else
    ConjSpinor

let lorentz = function
  | M f →
      begin match f with
      | L n → spinor n | N n → spinor n
      | U n → spinor n | D n → spinor n
      | TopH → Spinor | TopHb → ConjSpinor
      end
  | G f →
      begin match f with
      | Ga | Gl → Vector
      | Wp | Wm | Z | WHp | WHm | ZH | AH →
Massive_Vector
      end
  | O f →
      begin match f with
      | Phip | Phim | Phi0 | H | Eta | Psi0
      | Psi1 | Psip | Psim | Psipp | Psimm → Scalar
      end

let color = function
  | M (U n) → Color.SUN (if n > 0 then 3 else -3)
```

```
  | M (D n) → Color.SUN (if n > 0 then 3 else -3)
  | M TopH → Color.SUN 3 | M TopHb → Color.SUN (−3)
  | G Gl → Color.AdjSUN 3
  | _ → Color.Singlet

let prop_spinor n =
  if n ≥ 0 then
    Prop_Spinor
  else
    Prop_ConjSpinor

let propagator = function
  | M f →
      begin match f with
      | L n → prop_spinor n | N n → prop_spinor n
      | U n → prop_spinor n | D n → prop_spinor n
      | TopH → Prop_Spinor | TopHb → Prop_ConjSpinor
      end
  | G f →
      begin match f with
      | Ga | Gl → Prop_Feynman
      | Wp | Wm | Z | WHp | WHm | ZH | AH →
Prop_Unitarity
      end
  | O f →
      begin match f with
      | Phip | Phim | Phi0 → Only_Insertion
      | H | Eta | Psi0 | Psi1 | Psip | Psim
      | Psipp | Psimm → Prop_Scalar
      end
```

Optionally, ask for the fudge factor treatment for the widths of charged particles. Currently, this only applies to $W^{\pm}$ and top.

```
let width f =
  if !use_fudged_width then
    match f with
    | G Wp | G Wm | M (U 3) | M (U (−3))
    | G WHp | G WHm | G ZH | G AH
    | M TopH | M TopHb → Fudged
    | _ → !default_width
  else
    !default_width

let goldstone = function
  | G f →
      begin match f with
      | Wp → Some (O Phip, Coupling.Const 1)
      | Wm → Some (O Phim, Coupling.Const 1)
      | Z → Some (O Phi0, Coupling.Const 1)
      | _ → None
      end
  | _ → None
```

```
let conjugate = function
  | M f →
      M (begin match f with
      | L n → L (−n) | N n → N (−n)
      | U n → U (−n) | D n → D (−n)
      | TopH → TopHb | TopHb → TopH
      end)
  | G f →
      G (begin match f with
      | Gl → Gl | Ga → Ga | Z → Z
      | Wp → Wm | Wm → Wp | WHm → WHp
      | WHp → WHm | ZH → ZH | AH → AH
      end)
  | O f →
      O (begin match f with
      | Psi0 → Psi0 | Psi1 → Psi1 | Psip → Psim
      | Psim → Psip | Psipp → Psimm | Psimm → Psipp
      | Phip → Phim | Phim → Phip | Phi0 → Phi0
      | H → H | Eta → Eta
      end)

let fermion = function
  | M f →
      begin match f with
      | L n → if n > 0 then 1 else -1
      | N n → if n > 0 then 1 else -1
      | U n → if n > 0 then 1 else -1
      | D n → if n > 0 then 1 else -1
      | TopH → 1 | TopHb → −1
      end
  | G f →
      begin match f with
      | Gl | Ga | Z | Wp | Wm | WHp
      | WHm | AH | ZH → 0
      end
  | O f →
      begin match f with
      | Psi0 | Psi1 | Psip | Psim | Psipp | Psimm
      | Phip | Phim | Phi0 | H | Eta → 0
      end
```

This model does NOT have a conserved generation charge even in absence of CKM mixing because of the heavy top admixture.

```
module Ch = Charges.QQ
let ( // ) = Algebra.Small_Rational.make

let charge = function
  | M f →
      begin match f with
      | L n → if n > 0 then -1//1 else 1//1
      | N n → 0//1
```

```
        | U n → if n > 0 then 2//3 else -2//3
        | D n → if n > 0 then -1//3 else 1//3
        | TopH → 2//3
        | TopHb → − 2//3
        end
    | G f →
        begin match f with
        | Gl | Ga | Z | AH | ZH → 0//1
        | Wp | WHp → 1//1
        | Wm | WHm → − 1//1
        end
    | O f →
        begin match f with
        | H | Phi0 | Eta | Psi1 | Psi0 → 0//1
        | Phip | Psip → 1//1
        | Phim | Psim → − 1//1
        | Psipp → 2//1
        | Psimm → − 2//1
        end

let lepton = function
    | M f →
        begin match f with
        | L n | N n → if n > 0 then 1//1 else -1//1
        | U _ | D _ | _ → 0//1
        end
    | G _ | O _ → 0//1

let baryon = function
    | M f →
        begin match f with
        | L _ | N _ → 0//1
        | U n | D n → if n > 0 then 1//1 else -1//1
        | TopH → 1//1
        | TopHb → − 1//1
        end
    | G _ | O _ → 0//1

let charges f =
    [ charge f; lepton f; baryon f]

type constant =
    | Unit | Pi | Alpha_QED | Sin2thw
    | Sinthw | Costhw | E | G_weak | Vev | VHeavy
    | Supp | Supp2
    | Sinpsi | Cospsi | Atpsi | Sccs (* Mixing angles of SU(2) *)
    | Q_lepton | Q_up | Q_down | Q_Z_up | G_CC | G_CCtop
    | G_NC_neutrino | G_NC_lepton | G_NC_up | G_NC_down |
G_NC_heavy
    | G_NC_h_neutrino | G_NC_h_lepton | G_NC_h_up | G_NC_h_down
    | G_CC_heavy | G_ZHTHT | G_ZTHT | G_AHTHTH |
G_AHTHT | G_AHTT
```

| *G_CC_WH* | *G_CC_W*
| *I_Q_W* | *I_G_ZWW* | *I_G_WWW*
| *I_G_AHWW* | *I_G_ZHWW* | *I_G_ZWHW* | *I_G_AHWHWH* | *I_G_ZHWHWH*
| *I_G_AHWHW* | *I_Q_H*
| *G_WWWW* | *G_ZZWW* | *G_AZWW* | *G_AAWW*
| *G_WH4* | *G_WHWHWW* | *G_WHWWW* | *G_WH3W*
| *G_WWAAH* | *G_WWAZH* | *G_WWZZH* | *G_WWZAH* | *G_WHWHAAH*
| *G_WHWHAZH* | *G_WHWHZZH* | *G_WHWHZAH* | *G_WWZHAH*
| *G_WHWHZHAH* | *G_WHWZZ* | *G_WHWAZ* | *G_WHWAAH* | *G_WHWZAH*
| *G_WHWZHZH* | *G_WHWZHAH* | *G_WHWAZH* | *G_WHWZZH*
| *G_HWW* | *G_HHWW* | *G_HZZ* | *G_HHZZ*
| *G_PsiWW* | *G_PsiWHW* | *G_PsiZZ* | *G_PsiZHZH*
| *G_PsiZHZ* | *G_PsiZAH* | *G_PsiZHAH* | *G_PsiAHAH*
| *G_PsiZW* | *G_PsiZWH* | *G_PsiAHW* | *G_PsiAHWH*
| *G_PsiZHW* | *G_PsiZHWH*
| *G_PsippWW* | *G_PsippWHW* | *G_PsippWHWH*
| *G_PsiHW* | *G_PsiHWH* | *G_Psi0W* | *G_Psi0WH*
| *G_Psi1W* | *G_Psi1WH* | *G_PsiPPW* | *G_PsiPPWH*
| *G_Psi1HAH* | *G_Psi01AH* | *G_AHPsip* | *G_Psi1HZ*
| *G_Psi1HZH* | *G_Psi01Z* | *G_Psi01ZH* | *G_ZPsip* | *G_ZPsipp* | *G_ZHPsipp*
| *G_HHAA* | *G_HHWHW* | *G_HHZHZ* | *G_HHAHZ* | *G_HHZHAH*
| *G_HPsi0WW* | *G_HPsi0WHW* | *G_HPsi0ZZ*
| *G_HPsi0ZHZH* | *G_HPsi0ZHZ* | *G_HPsi0AHAH* | *G_HPsi0ZAH* | *G_HPsi0ZHAH*
| *G_HPsipWA* | *G_HPsipWHA* | *G_HPsipWZ* | *G_HPsipWHZ* | *G_HPsipWAH*
| *G_HPsipWHAH* | *G_HPsipWZH* | *G_HPsipWHZH* | *G_HPsippWW* | *G_HPsippWHWH*
| *G_HPsippWHW* | *G_Psi00ZH* | *G_Psi00AH* | *G_Psi00ZHAH*
| *G_Psi0pWA* | *G_Psi0pWHA* | *G_Psi0pWZ* | *G_Psi0pWHZ* | *G_Psi0pWAH*
| *G_Psi0pWHAH* | *G_Psi0pWZH* | *G_Psi0pWHZH* | *G_Psi0ppWW* | *G_Psi0ppWHWH*
| *G_Psi0ppWHW* | *I_G_Psi0pWA* | *I_G_Psi0pWHA* | *I_G_Psi0pWZ* | *I_G_Psi0pWHZ*
| *I_G_Psi0pWAH* | *I_G_Psi0pWHAH* | *I_G_Psi0pWZH* | *I_G_Psi0pWHZH*
| *I_G_Psi0ppWW* | *I_G_Psi0ppWHWH* | *I_G_Psi0ppWHW*
| *G_PsippZZ* | *G_PsippZHZH* | *G_PsippAZ* | *G_PsippAAH* | *G_PsippZAH*
| *G_PsippWA* | *G_PsippWHA* | *G_PsippWZ* | *G_PsippWHZ* | *G_PsippWAH*
| *G_PsippWHAH* | *G_PsippWZH* | *G_PsippWHZH*
| *G_PsiccZZ* | *G_PsiccAZ* | *G_PsiccAAH* | *G_PsiccZZH* | *G_PsiccAZH*
| *G_PsiccZAH*
| *G_Htt* | *G_Hbb* | *G_Hcc* | *G_Htautau* | *G_H3* | *G_H4*

```
| G_Hthth | G_Htht | G_Ethth | G_Etht | G_Ett
| G_HHtt | G_HHthth | G_HHtht
| G_Psi0tt | G_Psi0bb | G_Psi0cc | G_Psi0tautau
| G_Psi1tt | G_Psi1bb | G_Psi1cc | G_Psi1tautau
| G_Psipq3 | G_Psipq2 | G_Psipl3 | G_Psi0tth | G_Psi1tth
| G_Psipbth | G_Ebb
| G_HGaGa | G_HGaZ | G_EGaGa | G_EGaZ | G_EGlGl
| Gs | I_Gs | G2
| G_HWHW | G_HWHWH | G_HAHAH | G_HZHZ | G_HZHAH |
G_HAHZ
| Mass of flavor | Width of flavor
```

Two integer counters for the QCD and EW order of the couplings.

```
type orders = int × int

let orders = function
  | _ → (0, 0)

let input_parameters =
  []

let derived_parameters =
  []

let g_over_2_costh =
  Quot (Neg (Atom G_weak), Prod [Const 2; Atom Costhw])

let nc_coupling c t3 q =
  (Real_Array c,
   [Prod [g_over_2_costh; Diff (t3, Prod [Const 2; q; Atom Sin2thw])];
    Prod [g_over_2_costh; t3]])

let half = Quot (Const 1, Const 2)

let derived_parameter_arrays =
  [ nc_coupling G_NC_neutrino half (Const 0);
    nc_coupling G_NC_lepton (Neg half) (Const (−1));
    nc_coupling G_NC_up half (Quot (Const 2, Const 3));
    nc_coupling G_NC_down (Neg half) (Quot (Const (−1), Const 3));
    nc_coupling G_NC_h_neutrino half (Const 0);
    nc_coupling G_NC_h_lepton (Neg half) (Const (−1));
    nc_coupling G_NC_h_up half (Quot (Const 2, Const 3));
    nc_coupling G_NC_h_down (Neg half) (Quot (Const (−1), Const 3)) ]

let parameters () =
  { input = input_parameters;
    derived = derived_parameters;
    derived_arrays = derived_parameter_arrays }

module F = Modeltools.Fusions (struct
  type f = flavor
  type c = constant
  let compare = compare
  let conjugate = conjugate
end)
```

```
let mgm ((m1, g, m2), fbf, c) = ((M m1, G g, M m2), fbf, c)
let mhm ((m1, h, m2), fbf, c) = ((M m1, O h, M m2), fbf, c)
let tgc ((g1, g2, g3), t, c) = ((G g1, G g2, G g3), t, c)
let qgc ((g1, g2, g3, g4), t, c) = ((G g1, G g2, G g3, G g4), t, c)
let hgg ((h, g1, g2), coup, c) = ((O h, G g1, G g2), coup, c)
let ghh ((g, h1, h2), coup, c) = ((G g, O h1, O h2), coup, c)
let hhgg ((h1, h2, g1, g2), coup, c) = ((O h1, O h2, G g1, G g2), coup, c)

let electromagnetic_currents n =
  List.map mgm
    [ ((L (-n), Ga, L n), FBF (1, Psibar, V, Psi), Q_lepton);
      ((U (-n), Ga, U n), FBF (1, Psibar, V, Psi), Q_up);
      ((D (-n), Ga, D n), FBF (1, Psibar, V, Psi), Q_down) ]

let neutral_currents n =
  List.map mgm
    [ ((L (-n), Z, L n), FBF (1, Psibar, VA, Psi), G_NC_lepton);
      ((N (-n), Z, N n), FBF (1, Psibar, VA, Psi), G_NC_neutrino);
      ((U (-n), Z, U n), FBF (1, Psibar, VA, Psi), G_NC_up);
      ((D (-n), Z, D n), FBF (1, Psibar, VA, Psi), G_NC_down) ]
```

The sign of this coupling is just the one of the T3, being -(1/2) for leptons and down quarks, and +(1/2) for neutrinos and up quarks.

```
let neutral_heavy_currents n =
  List.map mgm
   ([ ((L (-n), ZH, L n), FBF ((-1), Psibar, VL, Psi), G_NC_heavy);
      ((N (-n), ZH, N n), FBF (1, Psibar, VL, Psi), G_NC_heavy);
      ((U (-n), ZH, U n), FBF (1, Psibar, VL, Psi), G_NC_heavy);
      ((D (-n), ZH, D n), FBF ((-1), Psibar, VL, Psi), G_NC_heavy)]
    @
      (if Flags.u1_gauged then
    [ ((L (-n), AH, L n), FBF (1, Psibar, VA, Psi), G_NC_h_lepton);
      ((N (-n), AH, N n), FBF (1, Psibar, VA, Psi), G_NC_h_neutrino);
      ((D (-n), AH, D n), FBF (1, Psibar, VA, Psi), G_NC_h_down)]
      else
        []))

let color_currents n =
  List.map mgm
  [ ((D (-n), Gl, D n), FBF ((-1), Psibar, V, Psi), Gs);
    ((U (-n), Gl, U n), FBF ((-1), Psibar, V, Psi), Gs)]

let heavy_top_currents =
  List.map mgm
   ([ ((TopHb, Ga, TopH), FBF (1, Psibar, V, Psi), Q_up);
      ((TopHb, Z, TopH), FBF (1, Psibar, V, Psi), Q_Z_up);
      ((TopHb, Z, U 3), FBF (1, Psibar, VL, Psi), G_ZTHT);
      ((U (-3), Z, TopH), FBF (1, Psibar, VL, Psi), G_ZTHT);
      ((TopHb, ZH, U 3), FBF (1, Psibar, VL, Psi), G_ZHTHT);
      ((U (-3), ZH, TopH), FBF (1, Psibar, VL, Psi), G_ZHTHT);
      ((U (-3), Wp, D 3), FBF (1, Psibar, VL, Psi), G_CCtop);
      ((D (-3), Wm, U 3), FBF (1, Psibar, VL, Psi), G_CCtop);
```

```
        ((TopHb, WHp, D 3), FBF (1, Psibar, VL, Psi), G_CC_WH);
        ((D (-3), WHm, TopH), FBF (1, Psibar, VL, Psi), G_CC_WH);
        ((TopHb, Wp, D 3), FBF (1, Psibar, VL, Psi), G_CC_W);
        ((D (-3), Wm, TopH), FBF (1, Psibar, VL, Psi), G_CC_W)]
        @
          (if Flags.u1_gauged then
      [ ((U (-3), AH, U 3), FBF (1, Psibar, VA, Psi), G_AHTT);
        ((TopHb, AH, TopH), FBF (1, Psibar, VA, Psi), G_AHTHTH);
        ((TopHb, AH, U 3), FBF (1, Psibar, VR, Psi), G_AHTHT);
        ((U (-3), AH, TopH), FBF (1, Psibar, VR, Psi), G_AHTHT)]
          else
            []))
```

$$\mathcal{L}_{\mathrm{CC}} = -\frac{g}{2\sqrt{2}}\sum_i \bar\psi_i(T^+\not{W}^+ + T^-\not{W}^-)(1-\gamma_5)\psi_i \tag{13.42}$$

```
  let charged_currents n =
    List.map mgm
      [ ((L (-n), Wm, N n), FBF (1, Psibar, VL, Psi), G_CC);
        ((N (-n), Wp, L n), FBF (1, Psibar, VL, Psi), G_CC);
        ((D (-n), Wm, U n), FBF (1, Psibar, VL, Psi), G_CC);
        ((U (-n), Wp, D n), FBF (1, Psibar, VL, Psi), G_CC) ]

  let charged_heavy_currents n =
    List.map mgm
     ([ ((L (-n), WHm, N n), FBF (1, Psibar, VL, Psi), G_CC_heavy);
        ((N (-n), WHp, L n), FBF (1, Psibar, VL, Psi), G_CC_heavy);
        ((D (-n), WHm, U n), FBF (1, Psibar, VL, Psi), G_CC_heavy);
        ((U (-n), WHp, D n), FBF (1, Psibar, VL, Psi), G_CC_heavy)]
        @
          (if Flags.u1_gauged then
      [ ((U (-n), AH, U n), FBF (1, Psibar, VA, Psi), G_NC_h_up)]
          else
            []))
```

We specialize the third generation since there is an additional shift coming from the admixture of the heavy top quark. The universal shift, coming from the mixing in the non-Abelian gauge boson sector is unobservable. (Redefinition of coupling constants by measured ones.

```
  let yukawa =
    List.map mhm
      [ ((U (-3), H, U 3), FBF (1, Psibar, S, Psi), G_Htt);
        ((D (-3), H, D 3), FBF (1, Psibar, S, Psi), G_Hbb);
        ((U (-2), H, U 2), FBF (1, Psibar, S, Psi), G_Hcc);
        ((L (-3), H, L 3), FBF (1, Psibar, S, Psi), G_Htautau)]

  let yukawa_add' =
    List.map mhm
      [ ((TopHb, H, TopH), FBF (1, Psibar, S, Psi), G_Hthth);
        ((TopHb, H, U 3), FBF (1, Psibar, SLR, Psi), G_Htht);
        ((U (-3), H, TopH), FBF (1, Psibar, SLR, Psi), G_Htht);
```

```
((U (−3), Psi0, U 3), FBF (1, Psibar, S, Psi), G_Psi0tt);
((D (−3), Psi0, D 3), FBF (1, Psibar, S, Psi), G_Psi0bb);
((U (−2), Psi0, U 2), FBF (1, Psibar, S, Psi), G_Psi0cc);
((L (−3), Psi0, L 3), FBF (1, Psibar, S, Psi), G_Psi0tautau);
((U (−3), Psi1, U 3), FBF (1, Psibar, P, Psi), G_Psi1tt);
((D (−3), Psi1, D 3), FBF (1, Psibar, P, Psi), G_Psi1bb);
((U (−2), Psi1, U 2), FBF (1, Psibar, P, Psi), G_Psi1cc);
((L (−3), Psi1, L 3), FBF (1, Psibar, P, Psi), G_Psi1tautau);
((U (−3), Psip, D 3), FBF (1, Psibar, SLR, Psi), G_Psipq3);
((U (−2), Psip, D 2), FBF (1, Psibar, SLR, Psi), G_Psipq2);
((N (−3), Psip, L 3), FBF (1, Psibar, SR, Psi), G_Psipl3);
((D (−3), Psim, U 3), FBF (1, Psibar, SLR, Psi), G_Psipq3);
((D (−2), Psim, U 2), FBF (1, Psibar, SLR, Psi), G_Psipq2);
((L (−3), Psim, N 3), FBF (1, Psibar, SL, Psi), G_Psipl3);
((TopHb, Psi0, U 3), FBF (1, Psibar, SL, Psi), G_Psi0tth);
((U (−3), Psi0, TopH), FBF (1, Psibar, SR, Psi), G_Psi0tth);
((TopHb, Psi1, U 3), FBF (1, Psibar, SL, Psi), G_Psi1tth);
((U (−3), Psi1, TopH), FBF (1, Psibar, SR, Psi), G_Psi1tth);
((TopHb, Psip, D 3), FBF (1, Psibar, SL, Psi), G_Psipbth);
((D (−3), Psim, TopH), FBF (1, Psibar, SR, Psi), G_Psipbth)]

let yukawa_add =
  if Flags.u1_gauged then
    yukawa_add'
  else
    yukawa_add' @
    List.map mhm
    [ ((U (−3), Eta, U 3), FBF (1, Psibar, P, Psi), G_Ett);
      ((TopHb, Eta, U 3), FBF (1, Psibar, SLR, Psi), G_Etht);
      ((D (−3), Eta, D 3), FBF (1, Psibar, P, Psi), G_Ebb);
      ((U (−3), Eta, TopH), FBF (1, Psibar, SLR, Psi), G_Etht)]
```

$$\mathcal{L}_{\mathrm{TGC}} = -e\partial_\mu A_\nu W_+^\mu W_-^\nu + \ldots - e\cot\theta_w \partial_\mu Z_\nu W_+^\mu W_-^\nu + \ldots \tag{13.43}$$

```
let standard_triple_gauge =
  List.map tgc
   [ ((Ga, Wm, Wp), Gauge_Gauge_Gauge 1, I_Q_W);
     ((Z, Wm, Wp), Gauge_Gauge_Gauge 1, I_G_ZWW);
     ((Gl, Gl, Gl), Gauge_Gauge_Gauge 1, I_Gs) ]

let heavy_triple_gauge =
  List.map tgc
  ([ ((Ga, WHm, WHp), Gauge_Gauge_Gauge 1, I_Q_W);
     ((Z, WHm, WHp), Gauge_Gauge_Gauge 1, I_G_ZWW);
     ((ZH, Wm, Wp), Gauge_Gauge_Gauge 1, I_G_ZHWW);
     ((Z, WHm, Wp), Gauge_Gauge_Gauge 1, I_G_ZWHW);
     ((Z, Wm, WHp), Gauge_Gauge_Gauge (−1), I_G_ZWHW);
     ((ZH, WHm, Wp), Gauge_Gauge_Gauge 1, I_G_WWW);
     ((ZH, Wm, WHp), Gauge_Gauge_Gauge (−1), I_G_WWW);
     ((ZH, WHm, WHp), Gauge_Gauge_Gauge (−1), I_G_ZHWHWH)]
```

```
      @
        (if Flags.u1_gauged then
     [ ((AH, Wm, Wp), Gauge_Gauge_Gauge 1, I_G_AHWW);
       ((AH, WHm, Wp), Gauge_Gauge_Gauge 1, I_G_AHWHW);
       ((AH, Wm, WHp), Gauge_Gauge_Gauge (-1), I_G_AHWHW);
       ((AH, WHm, WHp), Gauge_Gauge_Gauge 1, I_G_AHWHWH)]
        else
          []))
let triple_gauge =
     standard_triple_gauge @ heavy_triple_gauge
let gauge4 = Vector4 [(2, C_13_42); (-1, C_12_34); (-1, C_14_23)]
let minus_gauge4 = Vector4 [(-2, C_13_42); (1, C_12_34); (1, C_14_23)]
let standard_quartic_gauge =
  List.map qgc
    [ (Wm, Wp, Wm, Wp), gauge4, G_WWWW;
      (Wm, Z, Wp, Z), minus_gauge4, G_ZZWW;
      (Wm, Z, Wp, Ga), minus_gauge4, G_AZWW;
      (Wm, Ga, Wp, Ga), minus_gauge4, G_AAWW;
      (Gl, Gl, Gl, Gl), gauge4, G2 ]
let heavy_quartic_gauge =
  List.map qgc
   ([ (WHm, Wp, WHm, Wp), gauge4, G_WWWW;
      (Wm, WHp, Wm, WHp), gauge4, G_WWWW;
      (WHm, WHp, WHm, WHp), gauge4, G_WH4;
      (Wm, Wp, WHm, WHp), gauge4, G_WHWHWW;
      (Wm, Wp, Wm, WHp), gauge4, G_WHWWW;
      (Wm, Wp, WHm, Wp), gauge4, G_WHWWW;
      (WHm, WHp, Wm, WHp), gauge4, G_WH3W;
      (WHm, WHp, WHm, Wp), gauge4, G_WH3W;
      (WHm, Z, WHp, Z), minus_gauge4, G_ZZWW;
      (WHm, Z, WHp, Ga), minus_gauge4, G_AZWW;
      (WHm, Ga, WHp, ZH), minus_gauge4, G_AAWW;
      (WHm, Z, WHp, ZH), minus_gauge4, G_ZZWW;
      (Wm, ZH, Wp, ZH), minus_gauge4, G_WWWW;
      (Wm, Ga, Wp, ZH), minus_gauge4, G_WWAZH;
      (Wm, Z, Wp, ZH), minus_gauge4, G_WWZZH;
      (WHm, Ga, WHp, ZH), minus_gauge4, G_WHWHAZH;
      (WHm, Z, WHp, ZH), minus_gauge4, G_WHWHZZH;
      (WHm, ZH, WHp, ZH), minus_gauge4, G_WH4;
      (WHm, Z, Wp, Z), minus_gauge4, G_WHWZZ;
      (Wm, Z, WHp, Z), minus_gauge4, G_WHWZZ;
      (WHm, Ga, Wp, Z), minus_gauge4, G_WHWAZ;
      (Wm, Ga, WHp, Z), minus_gauge4, G_WHWAZ;
      (WHm, ZH, Wp, ZH), minus_gauge4, G_WHWZHZH;
      (Wm, ZH, WHp, ZH), minus_gauge4, G_WHWZHZH;
      (WHm, Ga, Wp, ZH), minus_gauge4, G_WHWAZH;
      (Wm, Ga, WHp, ZH), minus_gauge4, G_WHWAZH;
      (WHm, Z, Wp, ZH), minus_gauge4, G_WHWZZH;
      (Wm, Z, WHp, ZH), minus_gauge4, G_WHWZZH]
```

```
@
  (if Flags.u1_gauged then
    [ (Wm, Ga, Wp, AH), minus_gauge4, G_WWAAH;
      (Wm, Z, Wp, AH), minus_gauge4, G_WWZAH;
      (WHm, Ga, WHp, AH), minus_gauge4, G_WHWHAAH;
      (WHm, Z, WHp, AH), minus_gauge4, G_WHWHZAH;
      (Wm, ZH, Wp, AH), minus_gauge4, G_WWZHAH;
      (WHm, ZH, WHp, AH), minus_gauge4, G_WHWHZHAH;
      (WHm, Ga, Wp, AH), minus_gauge4, G_WHWAAH;
      (Wm, Ga, WHp, AH), minus_gauge4, G_WHWAAH;
      (WHm, Z, Wp, AH), minus_gauge4, G_WHWZAH;
      (Wm, Z, WHp, AH), minus_gauge4, G_WHWZAH;
      (WHm, ZH, Wp, AH), minus_gauge4, G_WHWZHAH;
      (Wm, ZH, WHp, AH), minus_gauge4, G_WHWZHAH]
  else
    []))

let quartic_gauge =
  standard_quartic_gauge @ heavy_quartic_gauge

let standard_gauge_higgs' =
  List.map hgg
    [ ((H, Wp, Wm), Scalar_Vector_Vector 1, G_HWW);
      ((H, Z, Z), Scalar_Vector_Vector 1, G_HZZ) ]

let heavy_gauge_higgs =
  List.map hgg
   ([ ((H, Wp, WHm), Scalar_Vector_Vector 1, G_HWHW);
      ((H, WHp, Wm), Scalar_Vector_Vector 1, G_HWHW);
      ((H, WHp, WHm), Scalar_Vector_Vector 1, G_HWHWH);
      ((H, ZH, ZH), Scalar_Vector_Vector 1, G_HWHWH);
      ((H, ZH, Z), Scalar_Vector_Vector 1, G_HZHZ);
      ((H, Wp, Wm), Scalar_Vector_Vector 1, G_HZHAH)]
  @
    (if Flags.u1_gauged then
      [((H, AH, AH), Scalar_Vector_Vector 1, G_HAHAH);
       ((H, Z, AH), Scalar_Vector_Vector 1, G_HAHZ)]
    else
      []))

let triplet_gauge_higgs =
  List.map hgg
   ([ ((Psi0, Wp, Wm), Scalar_Vector_Vector 1, G_PsiWW);
      ((Psi0, WHp, WHm), Scalar_Vector_Vector (-1), G_PsiWW);
      ((Psi0, WHp, Wm), Scalar_Vector_Vector 1, G_PsiWHW);
      ((Psi0, WHm, Wp), Scalar_Vector_Vector 1, G_PsiWHW);
      ((Psi0, Z, Z), Scalar_Vector_Vector 1, G_PsiZZ);
      ((Psi0, ZH, ZH), Scalar_Vector_Vector 1, G_PsiZHZH);
      ((Psi0, ZH, Z), Scalar_Vector_Vector 1, G_PsiZHZ);
      ((Psim, Wp, Z), Scalar_Vector_Vector 1, G_PsiZW);
      ((Psip, Wm, Z), Scalar_Vector_Vector 1, G_PsiZW);
      ((Psim, WHp, Z), Scalar_Vector_Vector 1, G_PsiZWH);
```

```
        ((Psip, WHm, Z), Scalar_Vector_Vector 1, G_PsiZWH);
        ((Psim, Wp, ZH), Scalar_Vector_Vector 1, G_PsiZHW);
        ((Psip, Wm, ZH), Scalar_Vector_Vector 1, G_PsiZHW);
        ((Psim, WHp, ZH), Scalar_Vector_Vector 1, G_PsiZHWH);
        ((Psip, WHm, ZH), Scalar_Vector_Vector 1, G_PsiZHWH);
        ((Psimm, Wp, Wp), Scalar_Vector_Vector 1, G_PsippWW);
        ((Psipp, Wm, Wm), Scalar_Vector_Vector 1, G_PsippWW);
        ((Psimm, WHp, Wp), Scalar_Vector_Vector 1, G_PsippWHW);
        ((Psipp, WHm, Wm), Scalar_Vector_Vector 1, G_PsippWHW);
        ((Psimm, WHp, WHp), Scalar_Vector_Vector 1, G_PsippWHWH);
        ((Psipp, WHm, WHm), Scalar_Vector_Vector 1, G_PsippWHWH)]
  @
    (if Flags.u1_gauged then
       [((Psi0, AH, Z), Scalar_Vector_Vector 1, G_PsiZAH);
        ((Psi0, AH, ZH), Scalar_Vector_Vector 1, G_PsiZHAH);
        ((Psi0, AH, AH), Scalar_Vector_Vector 1, G_PsiAHAH);
        ((Psim, Wp, AH), Scalar_Vector_Vector 1, G_PsiAHW);
        ((Psip, Wm, AH), Scalar_Vector_Vector 1, G_PsiAHW);
        ((Psim, WHp, AH), Scalar_Vector_Vector 1, G_PsiAHWH);
        ((Psip, WHm, AH), Scalar_Vector_Vector 1, G_PsiAHWH)]
    else
       []))

let triplet_gauge2_higgs =
  List.map ghh
   ([ ((Wp, H, Psim), Vector_Scalar_Scalar 1, G_PsiHW);
      ((Wm, H, Psip), Vector_Scalar_Scalar 1, G_PsiHW);
      ((WHp, H, Psim), Vector_Scalar_Scalar 1, G_PsiHWH);
      ((WHm, H, Psip), Vector_Scalar_Scalar 1, G_PsiHWH);
      ((Wp, Psi0, Psim), Vector_Scalar_Scalar 1, G_Psi0W);
      ((Wm, Psi0, Psip), Vector_Scalar_Scalar 1, G_Psi0W);
      ((WHp, Psi0, Psim), Vector_Scalar_Scalar 1, G_Psi0WH);
      ((WHm, Psi0, Psip), Vector_Scalar_Scalar 1, G_Psi0WH);
      ((Wp, Psi1, Psim), Vector_Scalar_Scalar 1, G_Psi1W);
      ((Wm, Psi1, Psip), Vector_Scalar_Scalar (-1), G_Psi1W);
      ((WHp, Psi1, Psim), Vector_Scalar_Scalar 1, G_Psi1WH);
      ((WHm, Psi1, Psip), Vector_Scalar_Scalar (-1), G_Psi1WH);
      ((Wp, Psip, Psimm), Vector_Scalar_Scalar 1, G_PsiPPW);
      ((Wm, Psim, Psipp), Vector_Scalar_Scalar 1, G_PsiPPW);
      ((WHp, Psip, Psimm), Vector_Scalar_Scalar 1, G_PsiPPWH);
      ((WHm, Psim, Psipp), Vector_Scalar_Scalar 1, G_PsiPPWH);
      ((Ga, Psip, Psim), Vector_Scalar_Scalar 1, Q_lepton);
      ((Ga, Psipp, Psimm), Vector_Scalar_Scalar 2, Q_lepton);
      ((Z, H, Psi1), Vector_Scalar_Scalar 1, G_Psi1HZ);
      ((ZH, H, Psi1), Vector_Scalar_Scalar 1, G_Psi1HZH);
      ((Z, Psi0, Psi1), Vector_Scalar_Scalar 1, G_Psi01Z);
      ((ZH, Psi0, Psi1), Vector_Scalar_Scalar 1, G_Psi01ZH);
      ((Z, Psip, Psim), Vector_Scalar_Scalar 1, G_ZPsip);
      ((Z, Psipp, Psimm), Vector_Scalar_Scalar 2, G_ZPsipp);
      ((ZH, Psipp, Psimm), Vector_Scalar_Scalar 2, G_ZHPsipp)]
```

```
    @
      (if Flags.u1_gauged then
        [((AH, H, Psi1), Vector_Scalar_Scalar 1, G_Psi1HAH);
         ((AH, Psi0, Psi1), Vector_Scalar_Scalar 1, G_Psi01AH);
         ((AH, Psip, Psim), Vector_Scalar_Scalar 1, G_AHPsip);
         ((AH, Psipp, Psimm), Vector_Scalar_Scalar 2, G_AHPsip)]
      else []))

let standard_gauge_higgs =
  standard_gauge_higgs' @ heavy_gauge_higgs @ triplet_gauge_higgs @
  triplet_gauge2_higgs

let standard_gauge_higgs4 =
  List.map hhgg
  [ (H, H, Wp, Wm), Scalar2_Vector2 1, G_HHWW;
    (H, H, Z, Z), Scalar2_Vector2 1, G_HHZZ ]

let littlest_gauge_higgs4 =
  List.map hhgg
    ([ (H, H, WHp, WHm), Scalar2_Vector2 (-1), G_HHWW;
       (H, H, ZH, ZH), Scalar2_Vector2 (-1), G_HHWW;
       (H, H, Wp, WHm), Scalar2_Vector2 1, G_HHWHW;
       (H, H, WHp, Wm), Scalar2_Vector2 1, G_HHWHW;
       (H, H, ZH, Z), Scalar2_Vector2 (-1), G_HHZHZ;
       (H, Psi0, Wp, Wm), Scalar2_Vector2 1, G_HPsi0WW;
       (H, Psi0, WHp, WHm), Scalar2_Vector2 (-1), G_HPsi0WW;
       (H, Psi0, WHp, Wm), Scalar2_Vector2 1, G_HPsi0WHW;
       (H, Psi0, Wp, WHm), Scalar2_Vector2 1, G_HPsi0WHW;
       (H, Psi0, Z, Z), Scalar2_Vector2 1, G_HPsi0ZZ;
       (H, Psi0, ZH, ZH), Scalar2_Vector2 1, G_HPsi0ZHZH;
       (H, Psi0, ZH, Z), Scalar2_Vector2 1, G_HPsi0ZHZ;
       (H, Psim, Wp, Ga), Scalar2_Vector2 1, G_HPsipWA;
       (H, Psip, Wm, Ga), Scalar2_Vector2 1, G_HPsipWA;
       (H, Psim, WHp, Ga), Scalar2_Vector2 1, G_HPsipWHA;
       (H, Psip, WHm, Ga), Scalar2_Vector2 1, G_HPsipWHA;
       (H, Psim, Wp, Z), Scalar2_Vector2 1, G_HPsipWZ;
       (H, Psip, Wm, Z), Scalar2_Vector2 1, G_HPsipWZ;
       (H, Psim, WHp, Z), Scalar2_Vector2 1, G_HPsipWHZ;
       (H, Psip, WHm, Z), Scalar2_Vector2 1, G_HPsipWHZ;
       (H, Psim, Wp, ZH), Scalar2_Vector2 1, G_HPsipWZH;
       (H, Psip, Wm, ZH), Scalar2_Vector2 1, G_HPsipWZH;
       (H, Psim, WHp, ZH), Scalar2_Vector2 1, G_HPsipWHZH;
       (H, Psip, WHm, ZH), Scalar2_Vector2 1, G_HPsipWHZH;
       (H, Psimm, Wp, Wp), Scalar2_Vector2 1, G_HPsippWW;
       (H, Psipp, Wm, Wm), Scalar2_Vector2 1, G_HPsippWW;
       (H, Psimm, WHp, WHp), Scalar2_Vector2 1, G_HPsippWHWH;
       (H, Psipp, WHm, WHm), Scalar2_Vector2 1, G_HPsippWHWH;
       (H, Psimm, WHp, Wp), Scalar2_Vector2 1, G_HPsippWHW;
       (H, Psipp, WHm, Wm), Scalar2_Vector2 1, G_HPsippWHW;
       (Psi0, Psi0, Wp, Wm), Scalar2_Vector2 2, G_HHWW;
       (Psi0, Psi0, WHp, WHm), Scalar2_Vector2 (-2), G_HHWW;
       (Psi0, Psi0, Z, Z), Scalar2_Vector2 4, G_HHZZ;
```

$(Psi0,\ Psi0,\ ZH,\ ZH),\ Scalar2\_Vector2\ 1,\ G\_Psi00ZH;$
$(Psi0,\ Psi0,\ WHp,\ Wm),\ Scalar2\_Vector2\ 2,\ G\_HHWHW;$
$(Psi0,\ Psi0,\ Wp,\ WHm),\ Scalar2\_Vector2\ 2,\ G\_HHWHW;$
$(Psi0,\ Psi0,\ Z,\ ZH),\ Scalar2\_Vector2\ 4,\ G\_HHZHZ;$
$(Psi0,\ Psim,\ Wp,\ Ga),\ Scalar2\_Vector2\ 1,\ G\_Psi0pWA;$
$(Psi0,\ Psip,\ Wm,\ Ga),\ Scalar2\_Vector2\ 1,\ G\_Psi0pWA;$
$(Psi0,\ Psim,\ WHp,\ Ga),\ Scalar2\_Vector2\ 1,\ G\_Psi0pWHA;$
$(Psi0,\ Psip,\ WHm,\ Ga),\ Scalar2\_Vector2\ 1,\ G\_Psi0pWHA;$
$(Psi0,\ Psim,\ Wp,\ Z),\ Scalar2\_Vector2\ 1,\ G\_Psi0pWZ;$
$(Psi0,\ Psip,\ Wm,\ Z),\ Scalar2\_Vector2\ 1,\ G\_Psi0pWZ;$
$(Psi0,\ Psim,\ WHp,\ Z),\ Scalar2\_Vector2\ 1,\ G\_Psi0pWHZ;$
$(Psi0,\ Psip,\ WHm,\ Z),\ Scalar2\_Vector2\ 1,\ G\_Psi0pWHZ;$
$(Psi0,\ Psim,\ Wp,\ ZH),\ Scalar2\_Vector2\ 1,\ G\_Psi0pWZH;$
$(Psi0,\ Psip,\ Wm,\ ZH),\ Scalar2\_Vector2\ 1,\ G\_Psi0pWZH;$
$(Psi0,\ Psim,\ WHp,\ ZH),\ Scalar2\_Vector2\ 1,\ G\_Psi0pWHZH;$
$(Psi0,\ Psip,\ WHm,\ ZH),\ Scalar2\_Vector2\ 1,\ G\_Psi0pWHZH;$
$(Psi0,\ Psimm,\ Wp,\ Wp),\ Scalar2\_Vector2\ 1,\ G\_Psi0ppWW;$
$(Psi0,\ Psipp,\ Wm,\ Wm),\ Scalar2\_Vector2\ 1,\ G\_Psi0ppWW;$
$(Psi0,\ Psimm,\ WHp,\ WHp),\ Scalar2\_Vector2\ 1,\ G\_Psi0ppWHWH;$
$(Psi0,\ Psipp,\ WHm,\ WHm),\ Scalar2\_Vector2\ 1,\ G\_Psi0ppWHWH;$
$(Psi0,\ Psimm,\ WHp,\ Wp),\ Scalar2\_Vector2\ 1,\ G\_Psi0ppWHW;$
$(Psi0,\ Psipp,\ WHm,\ Wm),\ Scalar2\_Vector2\ 1,\ G\_Psi0ppWHW;$
$(Psi1,\ Psi1,\ Wp,\ Wm),\ Scalar2\_Vector2\ 2,\ G\_HHWW;$
$(Psi1,\ Psi1,\ WHp,\ WHm),\ Scalar2\_Vector2\ (-2),\ G\_HHWW;$
$(Psi1,\ Psi1,\ Z,\ Z),\ Scalar2\_Vector2\ 4,\ G\_HHZZ;$
$(Psi1,\ Psi1,\ ZH,\ ZH),\ Scalar2\_Vector2\ 1,\ G\_Psi00ZH;$
$(Psi1,\ Psi1,\ WHp,\ Wm),\ Scalar2\_Vector2\ 2,\ G\_HHWHW;$
$(Psi1,\ Psi1,\ Wp,\ WHm),\ Scalar2\_Vector2\ 2,\ G\_HHWHW;$
$(Psi1,\ Psi1,\ Z,\ ZH),\ Scalar2\_Vector2\ 4,\ G\_HHZHZ;$
$(Psi1,\ Psim,\ Wp,\ Ga),\ Scalar2\_Vector2\ 1,\ I\_G\_Psi0pWA;$
$(Psi1,\ Psip,\ Wm,\ Ga),\ Scalar2\_Vector2\ (-1),\ I\_G\_Psi0pWA;$
$(Psi1,\ Psim,\ WHp,\ Ga),\ Scalar2\_Vector2\ 1,\ I\_G\_Psi0pWHA;$
$(Psi1,\ Psip,\ WHm,\ Ga),\ Scalar2\_Vector2\ (-1),\ I\_G\_Psi0pWHA;$
$(Psi1,\ Psim,\ Wp,\ Z),\ Scalar2\_Vector2\ 1,\ I\_G\_Psi0pWZ;$
$(Psi1,\ Psip,\ Wm,\ Z),\ Scalar2\_Vector2\ (-1),\ I\_G\_Psi0pWZ;$
$(Psi1,\ Psim,\ WHp,\ Z),\ Scalar2\_Vector2\ 1,\ I\_G\_Psi0pWHZ;$
$(Psi1,\ Psip,\ WHm,\ Z),\ Scalar2\_Vector2\ (-1),\ I\_G\_Psi0pWHZ;$
$(Psi1,\ Psim,\ Wp,\ ZH),\ Scalar2\_Vector2\ 1,\ I\_G\_Psi0pWZH;$
$(Psi1,\ Psip,\ Wm,\ ZH),\ Scalar2\_Vector2\ (-1),\ I\_G\_Psi0pWZH;$
$(Psi1,\ Psim,\ WHp,\ ZH),\ Scalar2\_Vector2\ 1,\ I\_G\_Psi0pWHZH;$
$(Psi1,\ Psip,\ WHm,\ ZH),\ Scalar2\_Vector2\ (-1),\ I\_G\_Psi0pWHZH;$
$(Psi1,\ Psimm,\ Wp,\ Wp),\ Scalar2\_Vector2\ 1,\ I\_G\_Psi0ppWW;$
$(Psi1,\ Psipp,\ Wm,\ Wm),\ Scalar2\_Vector2\ (-1),\ I\_G\_Psi0ppWW;$
$(Psi1,\ Psimm,\ WHp,\ WHp),\ Scalar2\_Vector2\ 1,\ I\_G\_Psi0ppWHWH;$
$(Psi1,\ Psipp,\ WHm,\ WHm),\ Scalar2\_Vector2\ (-1),\ I\_G\_Psi0ppWHWH;$
$(Psi1,\ Psimm,\ WHp,\ Wp),\ Scalar2\_Vector2\ 1,\ I\_G\_Psi0ppWHW;$
$(Psi1,\ Psipp,\ WHm,\ Wm),\ Scalar2\_Vector2\ (-1),\ I\_G\_Psi0ppWHW;$
$(Psip,\ Psim,\ Wp,\ Wm),\ Scalar2\_Vector2\ 4,\ G\_HHWW;$
$(Psip,\ Psim,\ WHp,\ WHm),\ Scalar2\_Vector2\ 1,\ G\_Psi00ZH;$
$(Psip,\ Psim,\ WHp,\ Wm),\ Scalar2\_Vector2\ 4,\ G\_HHWHW;$

```
    (Psip, Psim, Wp, WHm), Scalar2_Vector2 4, G_HHWHW;
    (Psip, Psim, Z, Z), Scalar2_Vector2 1, G_PsippZZ;
    (Psip, Psim, Ga, Ga), Scalar2_Vector2 2, G_AAWW;
    (Psip, Psim, ZH, ZH), Scalar2_Vector2 1, G_PsippZHZH;
    (Psip, Psim, Ga, Z), Scalar2_Vector2 4, G_PsippAZ;
    (Psip, Psimm, Wp, Ga), Scalar2_Vector2 1, G_PsippWA;
    (Psim, Psipp, Wm, Ga), Scalar2_Vector2 1, G_PsippWA;
    (Psip, Psimm, WHp, Ga), Scalar2_Vector2 1, G_PsippWHA;
    (Psim, Psipp, WHm, Ga), Scalar2_Vector2 1, G_PsippWHA;
    (Psip, Psimm, Wp, Z), Scalar2_Vector2 1, G_PsippWZ;
    (Psim, Psipp, Wm, Z), Scalar2_Vector2 1, G_PsippWZ;
    (Psip, Psimm, WHp, Z), Scalar2_Vector2 1, G_PsippWHZ;
    (Psim, Psipp, WHm, Z), Scalar2_Vector2 1, G_PsippWHZ;
    (Psip, Psimm, Wp, ZH), Scalar2_Vector2 1, G_PsippWZH;
    (Psim, Psipp, Wm, ZH), Scalar2_Vector2 1, G_PsippWZH;
    (Psip, Psimm, WHp, ZH), Scalar2_Vector2 1, G_PsippWHZH;
    (Psim, Psipp, WHm, ZH), Scalar2_Vector2 1, G_PsippWHZH;
    (Psipp, Psimm, Wp, Wm), Scalar2_Vector2 2, G_HHWW;
    (Psipp, Psimm, WHp, WHm), Scalar2_Vector2 (-2), G_HHWW;
    (Psipp, Psimm, WHp, Wm), Scalar2_Vector2 2, G_HHWHW;
    (Psipp, Psimm, Wp, WHm), Scalar2_Vector2 2, G_HHWHW;
    (Psipp, Psimm, Z, Z), Scalar2_Vector2 1, G_PsiccZZ;
    (Psipp, Psimm, Ga, Ga), Scalar2_Vector2 8, G_AAWW;
    (Psipp, Psimm, ZH, ZH), Scalar2_Vector2 1, G_Psi00ZH;
    (Psipp, Psimm, Ga, Z), Scalar2_Vector2 1, G_PsiccAZ;
    (Psipp, Psimm, Z, ZH), Scalar2_Vector2 4, G_PsiccZZH;
    (Psipp, Psimm, Ga, ZH), Scalar2_Vector2 4, G_PsiccAZH]
  @
    (if Flags.u1_gauged then
      [(H, H, AH, AH), Scalar2_Vector2 1, G_HHAA;
       (H, H, AH, Z), Scalar2_Vector2 (-1), G_HHAHZ;
       (H, H, ZH, AH), Scalar2_Vector2 (-1), G_HHZHAH;
       (H, Psi0, AH, AH), Scalar2_Vector2 1, G_HPsi0AHAH;
       (H, Psi0, Z, AH), Scalar2_Vector2 1, G_HPsi0ZAH;
       (H, Psi0, ZH, AH), Scalar2_Vector2 1, G_HPsi0ZHAH;
       (H, Psim, Wp, AH), Scalar2_Vector2 1, G_HPsipWAH;
       (H, Psip, Wm, AH), Scalar2_Vector2 1, G_HPsipWAH;
       (H, Psim, WHp, AH), Scalar2_Vector2 1, G_HPsipWHAH;
       (H, Psip, WHm, AH), Scalar2_Vector2 1, G_HPsipWHAH;
       (Psi0, Psi0, AH, AH), Scalar2_Vector2 1, G_Psi00AH;
       (Psi0, Psi0, Z, AH), Scalar2_Vector2 4, G_HHAHZ;
       (Psi0, Psi0, AH, ZH), Scalar2_Vector2 1, G_Psi00ZHAH;
       (Psi0, Psim, Wp, AH), Scalar2_Vector2 1, G_Psi0pWAH;
       (Psi0, Psip, Wm, AH), Scalar2_Vector2 1, G_Psi0pWAH;
       (Psi0, Psim, WHp, AH), Scalar2_Vector2 1, G_Psi0pWHAH;
       (Psi0, Psip, WHm, AH), Scalar2_Vector2 1, G_Psi0pWHAH;
       (Psi1, Psi1, AH, AH), Scalar2_Vector2 1, G_Psi00AH;
       (Psi1, Psi1, Z, AH), Scalar2_Vector2 4, G_HHAHZ;
       (Psi1, Psi1, AH, ZH), Scalar2_Vector2 1, G_Psi00ZHAH;
       (Psi1, Psim, Wp, AH), Scalar2_Vector2 1, I_G_Psi0pWAH;
```

```
          (Psi1, Psip, Wm, AH), Scalar2_Vector2 (-1), I_G_Psi0pWAH;
          (Psi1, Psim, WHp, AH), Scalar2_Vector2 1, I_G_Psi0pWHAH;
          (Psi1, Psip, WHm, AH), Scalar2_Vector2 (-1), I_G_Psi0pWHAH;
          (Psip, Psim, AH, AH), Scalar2_Vector2 1, G_Psi00AH;
          (Psip, Psim, Ga, AH), Scalar2_Vector2 4, G_PsippAAH;
          (Psip, Psim, Z, AH), Scalar2_Vector2 4, G_PsippZAH;
          (Psip, Psimm, Wp, AH), Scalar2_Vector2 1, G_PsippWAH;
          (Psim, Psipp, Wm, AH), Scalar2_Vector2 1, G_PsippWAH;
          (Psip, Psimm, WHp, AH), Scalar2_Vector2 1, G_PsippWHAH;
          (Psim, Psipp, WHm, AH), Scalar2_Vector2 1, G_PsippWHAH;
          (Psipp, Psimm, AH, AH), Scalar2_Vector2 1, G_Psi00AH;
          (Psipp, Psimm, AH, ZH), Scalar2_Vector2 (-1), G_Psi00ZHAH;
          (Psipp, Psimm, Ga, AH), Scalar2_Vector2 4, G_PsiccAAH;
          (Psipp, Psimm, Z, AH), Scalar2_Vector2 4, G_PsiccZAH]
        else []))

  let standard_higgs =
    [ (O H, O H, O H), Scalar_Scalar_Scalar 1, G_H3 ]

  let anomaly_higgs =
    List.map hgg
     [ (Eta, Gl, Gl), Dim5_Scalar_Gauge2_Skew 1, G_EGlGl;
       (Eta, Ga, Ga), Dim5_Scalar_Gauge2_Skew 1, G_EGaGa;
       (Eta, Ga, Z), Dim5_Scalar_Gauge2_Skew 1, G_EGaZ]
(* @ (H, Ga, Ga), Dim5_Scalar_Gauge2 1, G_HGaGa; (H, Ga, Z), Dim5_Scalar_Gauge2 1, G_HGaZ
*)

  let standard_higgs4 =
    [ (O H, O H, O H, O H), Scalar4 1, G_H4 ]

  let gauge_higgs =
      standard_gauge_higgs

  let gauge_higgs4 =
      standard_gauge_higgs4

  let higgs =
      standard_higgs

  let higgs4 =
      standard_higgs4

  let top_quartic =
    [ ((M (U (-3)), O H, O H, M (U 3)), GBBG (1, Psibar, S2, Psi), G_HHtt);
 ((M (TopHb), O H, O H, M TopH), GBBG (1, Psibar, S2, Psi), G_HHthth);
 ((M (U (-3)), O H, O H, M TopH), GBBG (1, Psibar, S2LR, Psi), G_HHtht);
 ((M (TopHb), O H, O H, M (U 3)), GBBG (1, Psibar, S2LR, Psi), G_HHtht)]

  let goldstone_vertices =
    List.map hgg
    [ ((Phi0, Wm, Wp), Scalar_Vector_Vector 1, I_G_ZWW);
      ((Phip, Ga, Wm), Scalar_Vector_Vector 1, I_Q_W);
      ((Phip, Z, Wm), Scalar_Vector_Vector 1, I_G_ZWW);
      ((Phim, Wp, Ga), Scalar_Vector_Vector 1, I_Q_W);
      ((Phim, Wp, Z), Scalar_Vector_Vector 1, I_G_ZWW) ]
```

```
let vertices3 =
  (ThoList.flatmap electromagnetic_currents [1;2;3] @
   ThoList.flatmap color_currents [1;2;3] @
   ThoList.flatmap neutral_currents [1;2;3] @
   ThoList.flatmap neutral_heavy_currents [1;2;3] @
   ThoList.flatmap charged_currents [1;2;3] @
   ThoList.flatmap charged_heavy_currents [1;2;3] @
   heavy_top_currents @
   (if Flags.u1_gauged then []
       else anomaly_higgs) @
   yukawa @ yukawa_add @ triple_gauge @
   gauge_higgs @ higgs @ goldstone_vertices)

let vertices4 =
  quartic_gauge @ gauge_higgs4 @ higgs4 @ top_quartic

let vertices () = (vertices3, vertices4, [])
```

For efficiency, make sure that *F.of_vertices vertices* is evaluated only once.

```
let table = F.of_vertices (vertices ())
let fuse2 = F.fuse2 table
let fuse3 = F.fuse3 table
let fuse = F.fuse table
let max_degree () = 4

let flavor_of_string = function
  | "e-" → M (L 1) | "e+" → M (L (−1))
  | "mu-" → M (L 2) | "mu+" → M (L (−2))
  | "tau-" → M (L 3) | "tau+" → M (L (−3))
  | "nue" → M (N 1) | "nuebar" → M (N (−1))
  | "numu" → M (N 2) | "numubar" → M (N (−2))
  | "nutau" → M (N 3) | "nutaubar" → M (N (−3))
  | "u" → M (U 1) | "ubar" → M (U (−1))
  | "c" → M (U 2) | "cbar" → M (U (−2))
  | "t" → M (U 3) | "tbar" → M (U (−3))
  | "d" → M (D 1) | "dbar" → M (D (−1))
  | "s" → M (D 2) | "sbar" → M (D (−2))
  | "b" → M (D 3) | "bbar" → M (D (−3))
  | "th" → M TopH | "thbar" → M TopHb
  | "g" | "gl" → G Gl
  | "A" → G Ga | "Z" | "Z0" → G Z
  | "AH" | "AH0" | "Ah" | "Ah0" → G AH
  | "ZH" | "ZH0" | "Zh" | "Zh0" → G ZH
  | "W+" → G Wp | "W-" → G Wm
  | "WH+" → G WHp | "WH-" → G WHm
  | "H" | "h" → O H | "eta" | "Eta" → O Eta
  | "Psi" | "Psi0" | "psi" | "psi0" → O Psi0
  | "Psi1" | "psi1" → O Psi1
  | "Psi+" | "psi+" | "Psip" | "psip" → O Psip
  | "Psi-" | "psi-" | "Psim" | "psim" → O Psim
  | "Psi++" | "psi++" | "Psipp" | "psipp" → O Psipp
  | "Psi--" | "psi--" | "Psimm" | "psimm" → O Psimm
```

```
  | _ → invalid_arg "Modellib_BSM.Littlest.flavor_of_string"

let flavor_to_string = function
  | M f →
      begin match f with
      | L 1 → "e-" | L (−1) → "e+"
      | L 2 → "mu-" | L (−2) → "mu+"
      | L 3 → "tau-" | L (−3) → "tau+"
      | L _ → invalid_arg "Modellib_BSM.Littlest.flavor_to_string"
      | N 1 → "nue" | N (−1) → "nuebar"
      | N 2 → "numu" | N (−2) → "numubar"
      | N 3 → "nutau" | N (−3) → "nutaubar"
      | N _ → invalid_arg "Modellib_BSM.Littlest.flavor_to_string"
      | U 1 → "u" | U (−1) → "ubar"
      | U 2 → "c" | U (−2) → "cbar"
      | U 3 → "t" | U (−3) → "tbar"
      | U _ → invalid_arg "Modellib_BSM.Littlest.flavor_to_string"
      | D 1 → "d" | D (−1) → "dbar"
      | D 2 → "s" | D (−2) → "sbar"
      | D 3 → "b" | D (−3) → "bbar"
      | D _ → invalid_arg "Modellib_BSM.Littlest.flavor_to_string"
      | TopH → "th" | TopHb → "thbar"
      end
  | G f →
      begin match f with
      | Gl → "g"
      | Ga → "A" | Z → "Z"
      | Wp → "W+" | Wm → "W-"
      | ZH → "ZH" | AH → "AH" | WHp → "WHp" | WHm → "WHm"
      end
  | O f →
      begin match f with
      | Phip → "phi+" | Phim → "phi-" | Phi0 → "phi0"
      | H → "H" | Eta → "Eta"
      | Psi0 → "Psi0" | Psi1 → "Psi1" | Psip → "Psi+"
      | Psim → "Psi-" | Psipp → "Psi++" | Psimm → "Psi--"
      end

let flavor_to_TeX = function
  | M f →
      begin match f with
      | L 1 → "e^-" | L (−1) → "e^+"
      | L 2 → "\\mu^-" | L (−2) → "\\mu^+"
      | L 3 → "\\tau^-" | L (−3) → "\\tau^+"
      | L _ → invalid_arg "Modellib_BSM.Littlest.flavor_to_TeX"
      | N 1 → "\\nu_e" | N (−1) → "\\bar{\\nu}_e"
      | N 2 → "\\nu_\\mu" | N (−2) → "\\bar{\\nu}_\\mu"
      | N 3 → "\\nu_\\tau" | N (−3) → "\\bar{\\nu}_\\tau"
      | N _ → invalid_arg "Modellib_BSM.Littlest.flavor_to_TeX"
      | U 1 → "u" | U (−1) → "\\bar{u}"
      | U 2 → "c" | U (−2) → "\\bar{c}"
```

```
          | U 3 → "t" | U (−3) → "\\bar{t}"
          | U _ → invalid_arg "Modellib_BSM.Littlest.flavor_to_TeX"
          | D 1 → "d" | D (−1) → "\\bar{d}"
          | D 2 → "s" | D (−2) → "\\bar{s}"
          | D 3 → "b" | D (−3) → "\\bar{b}"
          | D _ → invalid_arg "Modellib_BSM.Littlest.flavor_to_TeX"
          | TopH → "T" | TopHb → "\\bar{T}"
          end
      | G f →
          begin match f with
          | Gl → "g"
          | Ga → "\\gamma" | Z → "Z"
          | Wp → "W^+" | Wm → "W^-"
          | ZH → "Z_H" | AH → "\\gamma_H" | WHp → "W_H^+" |
WHm → "W_H^-"
          end
      | O f →
          begin match f with
          | Phip → "\\Phi^+" | Phim → "\\Phi^-" | Phi0 →
"\\Phi^0"
          | H → "H" | Eta → "\\eta"
          | Psi0 → "\\Psi_S" | Psi1 → "\\Psi_P" | Psip → "\\Psi^+"
          | Psim → "\\Psi^-" | Psipp → "\\Psi^{++}" | Psimm →
"\\Psi^{--}"
          end

  let flavor_symbol = function
      | M f →
          begin match f with
          | L n when n > 0 → "l" ^ string_of_int n
          | L n → "l" ^ string_of_int (abs n) ^ "b"
          | N n when n > 0 → "n" ^ string_of_int n
          | N n → "n" ^ string_of_int (abs n) ^ "b"
          | U n when n > 0 → "u" ^ string_of_int n
          | U n → "u" ^ string_of_int (abs n) ^ "b"
          | D n when n > 0 → "d" ^ string_of_int n
          | D n → "d" ^ string_of_int (abs n) ^ "b"
          | TopH → "th" | TopHb → "thb"
          end
      | G f →
          begin match f with
          | Gl → "gl"
          | Ga → "a" | Z → "z"
          | Wp → "wp" | Wm → "wm"
          | ZH → "zh" | AH → "ah" | WHp → "whp" | WHm → "whm"
          end
      | O f →
          begin match f with
          | Phip → "pp" | Phim → "pm" | Phi0 → "p0"
          | H → "h" | Eta → "eta"
```

```
      | Psi0 → "psi0" | Psi1 → "psi1" | Psip → "psip"
      | Psim → "psim" | Psipp → "psipp" | Psimm → "psimm"
      end
```

There are PDG numbers for Z', Z", W', 32-34, respectively. We just introduce a number 38 for Y0 as a Z"'. As well, there is the number 8 for a t'. But we cheat a little bit and take the number 35 which is reserved for a heavy scalar Higgs for the Eta scalar. For the heavy Higgs states we take 35 and 36 for the neutral ones, 37 for the charged and 38 for the doubly-charged. The pseudoscalar gets the 39.

```
let pdg = function
  | M f →
      begin match f with
      | L n when n > 0 → 9 + 2 × n
      | L n → − 9 + 2 × n
      | N n when n > 0 → 10 + 2 × n
      | N n → − 10 + 2 × n
      | U n when n > 0 → 2 × n
      | U n → 2 × n
      | D n when n > 0 → − 1 + 2 × n
      | D n → 1 + 2 × n
      | TopH → 8 | TopHb → (−8)
      end
  | G f →
      begin match f with
      | Gl → 21
      | Ga → 22 | Z → 23
      | Wp → 24 | Wm → (−24)
      | AH → 32 | ZH → 33 | WHp → 34 | WHm → (−34)
      end
  | O f →
      begin match f with
      | Phip | Phim → 27 | Phi0 → 26
      | Psi0 → 35 | Psi1 → 36 | Psip → 37 | Psim → (−37)
      | Psipp → 38 | Psimm → (−38)
      | H → 25 | Eta → 39
      end

let mass_symbol f =
  "mass(" ^ string_of_int (abs (pdg f)) ^ ")"

let width_symbol f =
  "width(" ^ string_of_int (abs (pdg f)) ^ ")"

let constant_symbol = function
  | Unit → "unit" | Pi → "PI" | VHeavy → "vheavy"
  | Alpha_QED → "alpha" | E → "e" | G_weak → "g" | Vev →
"vev"
  | Sin2thw → "sin2thw" | Sinthw → "sinthw" | Costhw →
"costhw"
  | Sinpsi → "sinpsi" | Cospsi → "cospsi"
  | Atpsi → "atpsi" | Sccs → "sccs"
```

| *Supp* → "vF" | *Supp2* → "v2F2"
| *Q_lepton* → "qlep" | *Q_up* → "qup" | *Q_down* → "qdwn"
| *Q_Z_up* → "qzup"
| *G_ZHTHT* → "gzhtht" | *G_ZTHT* → "gzhtht"
| *G_AHTHTH* → "gahthth" | *G_AHTHT* → "gahtht" | *G_AHTT* → "gahtt"
| *G_NC_lepton* → "gnclep" | *G_NC_neutrino* → "gncneu"
| *G_NC_up* → "gncup" | *G_NC_down* → "gncdwn"
| *G_CC* → "gcc" | *G_CCtop* → "gcctop" | *G_CC_heavy* → "gcch"
| *G_CC_WH* → "gccwh" | *G_CC_W* → "gccw"
| *G_NC_h_lepton* → "gnchlep" | *G_NC_h_neutrino* → "gnchneu"
| *G_NC_h_up* → "gnchup" | *G_NC_h_down* → "gnchdwn"
| *G_NC_heavy* → "gnch"
| *I_Q_W* → "iqw" | *I_G_ZWW* → "igzww" | *I_G_WWW* → "igwww"
| *I_G_AHWW* → "igahww" | *I_G_ZHWW* → "igzhww" | *I_G_ZWHW* → "igzwhw"
| *I_G_AHWHWH* → "igahwhwh" | *I_G_ZHWHWH* → "igzhwhwh"
| *I_G_AHWHW* → "igahwhw"
| *I_Q_H* → "iqh"
| *G_WWWW* → "gw4" | *G_ZZWW* → "gzzww"
| *G_AZWW* → "gazww" | *G_AAWW* → "gaaww"
| *G_WH4* → "gwh4" | *G_WHWHWW* → "gwhwhww" | *G_WHWWW* → "gwhwww"
| *G_WH3W* → "gwh3w"
| *G_WWAAH* → "gwwaah" | *G_WWAZH* → "gwwazh" | *G_WWZZH* → "gwwzzh"
| *G_WWZAH* → "gwwzah" | *G_WHWHAAH* → "gwhwhaah"
| *G_WHWHAZH* → "gwhwhazh" | *G_WHWHZZH* → "gwhwhzzh"
| *G_WHWHZAH* → "gwhwhzah"
| *G_WWZHAH* → "gwwzhah" | *G_WHWHZHAH* → "gwhwhzhah"
| *G_WHWZZ* → "gwhwzz" | *G_WHWAZ* → "gwhwaz"
| *G_WHWAAH* → "gwhwaah" | *G_WHWZAH* → "gwhwzah"
| *G_WHWZHZH* → "gwhwzhzh" | *G_WHWZHAH* → "gwhwzhah"
| *G_WHWAZH* → "gwhwazh" | *G_WHWZZH* → "gwhwzzh"
| *G_HWW* → "ghww" | *G_HZZ* → "ghzz"
| *G_HHWW* → "ghhww" | *G_HHZZ* → "ghhzz"
| *G_HWHW* → "ghwhw" | *G_HWHWH* → "ghwhwh" | *G_HAHAH* → "ghahah"
| *G_HZHZ* → "ghzhz" | *G_HZHAH* → "ghzhah"
| *G_HAHZ* → "ghahz"
| *G_Htt* → "ghtt" | *G_Hbb* → "ghbb"
| *G_Htautau* → "ghtautau" | *G_Hcc* → "ghcc"
| *G_Hthth* → "ghthth" | *G_Htht* → "ghtht"
| *G_HHtt* → "ghhtt" | *G_HHthth* → "ghhthth" | *G_HHtht* → "ghhtht"
| *G_Psi0tt* → "gpsi0tt" | *G_Psi0bb* → "gpsi0bb"
| *G_Psi0cc* → "gpsi0cc" | *G_Psi0tautau* → "gpsi0tautau"
| *G_Psi1tt* → "gpsi1tt" | *G_Psi1bb* → "gpsi1bb"

```
| G_Psi1cc → "gpsi1cc" | G_Psi1tautau → "gpsi1tautau"
| G_Psipq3 → "gpsipq3" | G_Psipq2 → "gpsipq2" | G_Psipl3 →
"gpsil3"
| G_Psi0tth → "gpsi0tth" | G_Psi1tth → "gpsi1tth"
| G_Psipbth → "gpsipbth"
| G_Ethth → "gethth" | G_Etht → "getht"
| G_Ett → "gett" | G_Ebb → "gebb"
| G_HGaGa → "ghgaga" | G_HGaZ → "ghgaz"
| G_EGaGa → "geaa" | G_EGaZ → "geaz" | G_EGlGl → "gegg"
| G_H3 → "gh3" | G_H4 → "gh4"
| G_PsiWW → "gpsiww" | G_PsiWHW → "gpsiwhw"
| G_PsiZZ → "gpsizz" | G_PsiZHZH → "gpsizhzh"
| G_PsiZHZ → "gpsizhz" | G_PsiZAH → "gpsizah"
| G_PsiZHAH → "gpsizhah" | G_PsiAHAH → "gpsiahah"
| G_PsiZW → "gpsizw" | G_PsiZWH → "gpsizwh" | G_PsiAHW →
"gpsiahw"
| G_PsiAHWH → "gpsiahwh" | G_PsiZHW → "gpsizhw"
| G_PsiZHWH → "gpsizhwh"
| G_PsippWW → "gpsippww" | G_PsippWHW → "gpsippwhw"
| G_PsippWHWH → "gpsippwhwh"
| Gs → "gs" | G2 → "gs**2" | I_Gs → "igs"
| G_PsiHW → "gpsihw" | G_PsiHWH → "gpsihwh"
| G_Psi0W → "gpsi0w" | G_Psi0WH → "gpsi0wh"
| G_Psi1W → "gpsi1w" | G_Psi1WH → "gpsi1wh"
| G_PsiPPW → "gpsippw" | G_PsiPPWH → "gpsippwh"
| G_Psi1HAH → "gpsihah" | G_Psi01AH → "gpsi0ah"
| G_AHPsip → "gahpsip" | G_Psi1HZ → "gpsi1hz"
| G_Psi1HZH → "gpsi1hzh" | G_Psi01Z → "gpsi01z"
| G_Psi01ZH → "gpsi01zh" | G_ZPsip → "gzpsip"
| G_ZPsipp → "gzpsipp" | G_ZHPsipp → "gzhpsipp"
| G_HHAA → "ghhaa" | G_HHWHW → "ghhwhw" | G_HHZHZ →
"ghhzhz"
| G_HHAHZ → "ghhahz" | G_HHZHAH → "ghhzhah"
| G_HPsi0WW → "ghpsi0ww" | G_HPsi0WHW → "ghpsi0whw"
| G_HPsi0ZZ → "ghpsi0zz" | G_HPsi0ZHZH → "ghpsi0zhzh"
| G_HPsi0ZHZ → "ghpsi0zhz" | G_HPsi0AHAH → "ghpsi0ahah"
| G_HPsi0ZAH → "ghpsi0zah" | G_HPsi0ZHAH → "ghpsi0zhah"
| G_HPsipWA → "ghpsipwa" | G_HPsipWHA → "ghpsipwha"
| G_HPsipWZ → "ghpsipwz" | G_HPsipWHZ → "ghpsiwhz"
| G_HPsipWAH → "ghpsipwah" | G_HPsipWHAH → "ghpsipwhah"
| G_HPsipWZH → "ghpsipwzh" | G_HPsipWHZH → "ghpsipwhzh"
| G_HPsippWW → "ghpsippww" | G_HPsippWHWH → "ghpsippwhwh"
| G_HPsippWHW → "ghpsippwhw" | G_Psi00ZH → "gpsi00zh"
| G_Psi00AH → "gpsi00ah" | G_Psi00ZHAH → "gpsi00zhah"
| G_Psi0pWA → "gpsi0pwa" | G_Psi0pWHA → "gpsi0pwha"
| G_Psi0pWZ → "gpsi0pwz" | G_Psi0pWHZ → "gpsi0pwhz"
| G_Psi0pWAH → "gpsi0pwah" | G_Psi0pWHAH → "gpsi0pwhah"
| G_Psi0pWZH → "gpsi0pwzh" | G_Psi0pWHZH → "gpsi0pwhzh"
| G_Psi0ppWW → "gpsi0ppww" | G_Psi0ppWHWH → "gpsi0ppwhwh"
| G_Psi0ppWHW → "gpsi0ppwhw"
```

```
    | I_G_Psi0pWA → "i_gpsi0pwa" | I_G_Psi0pWHA → "i_gpsi0pwha"
    | I_G_Psi0pWZ → "i_gpsi0pwz" | I_G_Psi0pWHZ → "i_gpsi0pwhz"
    | I_G_Psi0pWAH → "i_gpsi0pwah" | I_G_Psi0pWHAH → "i_gpsi0pwhah"
    | I_G_Psi0pWZH → "i_gpsi0pwzh" | I_G_Psi0pWHZH → "i_gpsi0pwhzh"
    | I_G_Psi0ppWW → "i_gpsi0ppww" | I_G_Psi0ppWHWH →
"i_gpsi0ppwhwh"
    | I_G_Psi0ppWHW → "i_gpsi0ppwhw"
    | G_PsippZZ → "gpsippzz" | G_PsippZHZH → "gpsippzhzh"
    | G_PsippAZ → "gpsippaz" | G_PsippAAH → "gpsippaah"
    | G_PsippZAH → "gpsippzah"
    | G_PsippWA → "gpsippwa" | G_PsippWHA → "gpsippwha"
    | G_PsippWZ → "gpsippwz" | G_PsippWHZ → "gpsippwhz"
    | G_PsippWAH → "gpsippwah" | G_PsippWHAH → "gpsippwhah"
    | G_PsippWZH → "gpsippwzh" | G_PsippWHZH → "gpsippwhzh"
    | G_PsiccZZ → "gpsicczz" | G_PsiccAZ → "gpsiccaz"
    | G_PsiccAAH → "gpsiccaah" | G_PsiccZZH → "gpsicczzh"
    | G_PsiccAZH → "gpsiccazh" | G_PsiccZAH → "gpsicczah"
    | Mass f → "mass" ^ flavor_symbol f
    | Width f → "width" ^ flavor_symbol f
  end

module Littlest_Tpar (Flags : BSM_flags) =
  struct
    let rcs = rcs_file

    open Coupling

    let default_width = ref Timelike
    let use_fudged_width = ref false

    let options = Options.create
      [ "constant_width", Arg.Unit (fun () → default_width := Constant),
        "use␣constant␣width␣(also␣in␣t-channel)";
        "fudged_width", Arg.Set use_fudged_width,
        "use␣fudge␣factor␣for␣charge␣particle␣width";
        "custom_width", Arg.String (fun f → default_width := Custom f),
        "use␣custom␣width";
        "cancel_widths", Arg.Unit (fun () → default_width := Vanishing),
        "use␣vanishing␣width" ]

    type flavor = L of int | N of int | U of int | D of int
        | Topp | Toppb
        | Ga | Wp | Wm | Z | Gl | Lodd of int | Nodd of int
        | Uodd of int | Dodd of int
        | WHp | WHm | ZH | AH | Phip | Phim | Phi0 | H | Eta |
Psi0
        | Psi1 | Psip | Psim | Psipp | Psimm

    type gauge = unit

    let gauge_symbol () =
      failwith "Modellib_BSM.Littlest_Tpar.gauge_symbol:␣internal␣error"
```

```
let family n = [ L n; N n; U n; D n; Dodd n; Nodd n; Lodd n; Uodd n ]
```

Since *Phi* already belongs to the EW Goldstone bosons we use *Psi* for the TeV scale complex triplet.

We use the notation Todd1 = Uodd 3, Todd2 = Uodd 4.

```
let external_flavors () =
  [ "1st Generation", ThoList.flatmap family [1; -1];
    "2nd Generation", ThoList.flatmap family [2; -2];
    "3rd Generation", ThoList.flatmap family [3; -3];
    "Heavy Quarks", [Topp; Toppb; Uodd 4; Uodd (-4)];
    "Heavy Scalars", [Psi0; Psi1; Psip; Psim; Psipp; Psimm];
    "Gauge Bosons", if Flags.u1_gauged then
      [Ga; Z; Wp; Wm; Gl; WHp; WHm; ZH; AH]
        else
      [Ga; Z; Wp; Wm; Gl; WHp; WHm; ZH];
    "Higgs", if Flags.u1_gauged then [H]
    else [H; Eta];
    "Goldstone Bosons", [Phip; Phim; Phi0] ]

let flavors () = ThoList.flatmap snd (external_flavors ())

let spinor n =
  if n ≥ 0 then
    Spinor
  else
    ConjSpinor

let lorentz = function
  | L n → spinor n | N n → spinor n
  | U n → spinor n | D n → spinor n
  | Topp → Spinor | Toppb → ConjSpinor
  | Ga | Gl → Vector
  | Wp | Wm | Z | WHp | WHm | ZH | AH → Massive_Vector
  | _ → Scalar

let color = function
  | U n → Color.SUN (if n > 0 then 3 else -3)
  | Uodd n → Color.SUN (if n > 0 then 3 else -3)
  | D n → Color.SUN (if n > 0 then 3 else -3)
  | Dodd n → Color.SUN (if n > 0 then 3 else -3)
  | Topp → Color.SUN 3 | Toppb → Color.SUN (-3)
  | Gl → Color.AdjSUN 3
  | _ → Color.Singlet

let prop_spinor n =
  if n ≥ 0 then
    Prop_Spinor
  else
    Prop_ConjSpinor

let propagator = function
  | L n → prop_spinor n | N n → prop_spinor n
  | Lodd n → prop_spinor n | Nodd n → prop_spinor n
```

```
  | U n → prop_spinor n | D n → prop_spinor n
  | Uodd n → prop_spinor n | Dodd n → prop_spinor n
  | Topp → Prop_Spinor | Toppb → Prop_ConjSpinor
  | Ga | Gl → Prop_Feynman
  | Wp | Wm | Z | WHp | WHm | ZH | AH → Prop_Unitarity
  | Phip | Phim | Phi0 → Only_Insertion
  | H | Eta | Psi0 | Psi1 | Psip | Psim | Psipp | Psimm →
Prop_Scalar
```

Optionally, ask for the fudge factor treatment for the widths of charged particles. Currently, this only applies to $W^{\pm}$ and top.

```
let width f =
  if !use_fudged_width then
    match f with
    | Wp | Wm | U 3 | U (−3)
    | WHp | WHm | ZH | AH
    | Uodd _ | Dodd _ | Nodd _ | Lodd _
    | Topp | Toppb → Fudged
    | _ → !default_width
  else
    !default_width

let goldstone = function
  | Wp → Some (Phip, Coupling.Const 1)
  | Wm → Some (Phim, Coupling.Const 1)
  | Z → Some (Phi0, Coupling.Const 1)
  | _ → None

let conjugate = function
  | L n → L (−n) | N n → N (−n)
  | Lodd n → L (−n) | Nodd n → N (−n)
  | U n → U (−n) | D n → D (−n)
  | Uodd n → U (−n) | Dodd n → D (−n)
  | Topp → Toppb | Toppb → Topp
  | Gl → Gl | Ga → Ga | Z → Z
  | Wp → Wm | Wm → Wp | WHm → WHp
  | WHp → WHm | ZH → ZH | AH → AH
  | Psi0 → Psi0 | Psi1 → Psi1 | Psip → Psim
  | Psim → Psip | Psipp → Psimm | Psimm → Psipp
  | Phip → Phim | Phim → Phip | Phi0 → Phi0
  | H → H | Eta → Eta

let fermion = function
  | L n → if n > 0 then 1 else -1
  | N n → if n > 0 then 1 else -1
  | U n → if n > 0 then 1 else -1
  | D n → if n > 0 then 1 else -1
  | Lodd n → if n > 0 then 1 else -1
  | Nodd n → if n > 0 then 1 else -1
  | Uodd n → if n > 0 then 1 else -1
  | Dodd n → if n > 0 then 1 else -1
  | Topp → 1 | Toppb → − 1
```

```
    | Gl | Ga | Z | Wp | Wm | WHp | WHm | AH | ZH → 0
    | _ → 0
  module Ch = Charges.QQ
  let ( // ) = Algebra.Small_Rational.make
  let charge = function
     | L n | Lodd n → if n > 0 then -1//1 else 1//1
     | N n | Nodd n → 0//1
     | U n | Uodd n → if n > 0 then 2//3 else -2//3
     | D n | Dodd n → if n > 0 then -1//3 else 1//3
     | Topp → 2//3
     | Toppb → −2//3
     | Gl | Ga | Z | AH | ZH → 0//1
     | Wp | WHp → 1//1
     | Wm | WHm → −1//1
     | H | Phi0 | Eta | Psi1 | Psi0 → 0//1
     | Phip | Psip → 1//1
     | Phim | Psim → −1//1
     | Psipp → 2//1
     | Psimm → −2//1
  let lepton = function
     | L n | N n | Lodd n | Nodd n
         → if n > 0 then 1//1 else -1//1
     | U _ | D _ | _ → 0//1
  let baryon = function
     | L _ | N _ → 0//1
     | U n | D n | Uodd n | Dodd n
         → if n > 0 then 1//1 else -1//1
     | Topp → 1//1
     | Toppb → −1//1
     | _ → 0//1
  let charges f =
    [ charge f; lepton f; baryon f]
  type constant =
    | Unit | Pi | Alpha_QED | Sin2thw
    | Sinthw | Costhw | E | G_weak | Vev | VHeavy
    | Supp | Supp2
    | Sinpsi | Cospsi | Atpsi | Sccs (* Mixing angles of SU(2) *)
    | Q_lepton | Q_up | Q_down | Q_Z_up | G_CC | G_CCtop
    | G_NC_neutrino | G_NC_lepton | G_NC_up | G_NC_down |
G_NC_heavy
    | G_NC_h_neutrino | G_NC_h_lepton | G_NC_h_up | G_NC_h_down
    | G_CC_heavy | G_ZHTHT | G_ZTHT | G_AHTHTH |
G_AHTHT | G_AHTT
    | G_CC_WH | G_CC_W
    | Gs | I_Gs | G2
    | I_Q_W | I_G_ZWW | I_G_WWW
    | I_G_AHWW | I_G_ZHWW | I_G_ZWHW | I_G_AHWHWH |
I_G_ZHWHWH
```

```
    | I_G_AHWHW | I_Q_H
    | G_WWWW | G_ZZWW | G_AZWW | G_AAWW
    | G_WH4 | G_WHWHWW | G_WHWWW | G_WH3W
    | G_WWAAH | G_WWAZH | G_WWZZH | G_WWZAH |
G_WHWHAAH
    | G_WHWHAZH | G_WHWHZZH | G_WHWHZAH | G_WWZHAH
    | G_WHWHZHAH | G_WHWZZ | G_WHWAZ | G_WHWAAH |
G_WHWZAH
    | G_WHWZHZH | G_WHWZHAH | G_WHWAZH | G_WHWZZH
    | G_HWW | G_HHWW | G_HZZ | G_HHZZ
    | G_PsiWW | G_PsiWHW | G_PsiZZ | G_PsiZHZH
    | G_PsiZHZ | G_PsiZAH | G_PsiZHAH | G_PsiAHAH
    | G_PsiZW | G_PsiZWH | G_PsiAHW | G_PsiAHWH
    | G_PsiZHW | G_PsiZHWH
    | G_PsippWW | G_PsippWHW | G_PsippWHWH
    | G_PsiHW | G_PsiHWH | G_Psi0W | G_Psi0WH
    | G_Psi1W | G_Psi1WH | G_PsiPPW | G_PsiPPWH
    | G_Psi1HAH | G_Psi01AH | G_AHPsip | G_Psi1HZ
    | G_Psi1HZH | G_Psi01Z | G_Psi01ZH | G_ZPsip | G_ZPsipp |
G_ZHPsipp
    | G_HHAA | G_HHWHW | G_HHZHZ | G_HHAHZ | G_HHZHAH
    | G_HPsi0WW | G_HPsi0WHW | G_HPsi0ZZ
    | G_HPsi0ZHZH | G_HPsi0ZHZ | G_HPsi0AHAH | G_HPsi0ZAH |
G_HPsi0ZHAH
    | G_HPsipWA | G_HPsipWHA | G_HPsipWZ | G_HPsipWHZ |
G_HPsipWAH
    | G_HPsipWHAH | G_HPsipWZH | G_HPsipWHZH | G_HPsippWW |
G_HPsippWHWH
    | G_HPsippWHW | G_Psi00ZH | G_Psi00AH | G_Psi00ZHAH
    | G_Psi0pWA | G_Psi0pWHA | G_Psi0pWZ | G_Psi0pWHZ |
G_Psi0pWAH
    | G_Psi0pWHAH | G_Psi0pWZH | G_Psi0pWHZH | G_Psi0ppWW |
G_Psi0ppWHWH
    | G_Psi0ppWHW | I_G_Psi0pWA | I_G_Psi0pWHA | I_G_Psi0pWZ |
I_G_Psi0pWHZ
    | I_G_Psi0pWAH | I_G_Psi0pWHAH | I_G_Psi0pWZH | I_G_Psi0pWHZH
    | I_G_Psi0ppWW | I_G_Psi0ppWHWH | I_G_Psi0ppWHW
    | G_PsippZZ | G_PsippZHZH | G_PsippAZ | G_PsippAAH |
G_PsippZAH
    | G_PsippWA | G_PsippWHA | G_PsippWZ | G_PsippWHZ |
G_PsippWAH
    | G_PsippWHAH | G_PsippWZH | G_PsippWHZH
    | G_PsiccZZ | G_PsiccAZ | G_PsiccAAH | G_PsiccZZH |
G_PsiccAZH
    | G_PsiccZAH
    | G_Htt | G_Hbb | G_Hcc | G_Htautau | G_H3 | G_H4
    | G_Hthth | G_Htht | G_Ethth | G_Etht | G_Ett
    | G_HHtt | G_HHthth | G_HHtht
    | G_Psi0tt | G_Psi0bb | G_Psi0cc | G_Psi0tautau
    | G_Psi1tt | G_Psi1bb | G_Psi1cc | G_Psi1tautau
```

```
    | G_Psipq3 | G_Psipq2 | G_Psipl3 | G_Psi0tth | G_Psi1tth
    | G_Psipbth | G_Ebb
    | G_HGaGa | G_HGaZ | G_EGaGa | G_EGaZ | G_EGlGl
    | G_HWHW | G_HWHWH | G_HAHAH | G_HZHZ | G_HZHAH |
G_HAHZ
    | Mass of flavor | Width of flavor
```

Two integer counters for the QCD and EW order of the couplings.

```
type orders = int × int

let orders = function
  | _ → (0,0)

let input_parameters =
  []

let derived_parameters =
  []

let g_over_2_costh =
  Quot (Neg (Atom G_weak), Prod [Const 2; Atom Costhw])

let nc_coupling c t3 q =
  (Real_Array c,
   [Prod [g_over_2_costh; Diff (t3, Prod [Const 2; q; Atom Sin2thw])];
    Prod [g_over_2_costh; t3]])

let half = Quot (Const 1, Const 2)

let derived_parameter_arrays =
  [ nc_coupling G_NC_neutrino half (Const 0);
    nc_coupling G_NC_lepton (Neg half) (Const (−1));
    nc_coupling G_NC_up half (Quot (Const 2, Const 3));
    nc_coupling G_NC_down (Neg half) (Quot (Const (−1), Const 3));
    nc_coupling G_NC_h_neutrino half (Const 0);
    nc_coupling G_NC_h_lepton (Neg half) (Const (−1));
    nc_coupling G_NC_h_up half (Quot (Const 2, Const 3));
    nc_coupling G_NC_h_down (Neg half) (Quot (Const (−1), Const 3)) ]

let parameters () =
  { input = input_parameters;
    derived = derived_parameters;
    derived_arrays = derived_parameter_arrays }

module F = Modeltools.Fusions (struct
  type f = flavor
  type c = constant
  let compare = compare
  let conjugate = conjugate
end)

let electromagnetic_currents n =
    [ ((L (−n), Ga, L n), FBF (1, Psibar, V, Psi), Q_lepton);
      ((U (−n), Ga, U n), FBF (1, Psibar, V, Psi), Q_up);
      ((D (−n), Ga, D n), FBF (1, Psibar, V, Psi), Q_down) ]
```

```
let color_currents n =
  [ ((U (-n), Gl, U n), FBF ((-1), Psibar, V, Psi), Gs);
    ((D (-n), Gl, D n), FBF ((-1), Psibar, V, Psi), Gs) ]

let neutral_currents n =
  [ ((L (-n), Z, L n), FBF (1, Psibar, VA, Psi), G_NC_lepton);
    ((N (-n), Z, N n), FBF (1, Psibar, VA, Psi), G_NC_neutrino);
    ((U (-n), Z, U n), FBF (1, Psibar, VA, Psi), G_NC_up);
    ((D (-n), Z, D n), FBF (1, Psibar, VA, Psi), G_NC_down) ]
```

The sign of this coupling is just the one of the T3, being -(1/2) for leptons and down quarks, and +(1/2) for neutrinos and up quarks.

```
let neutral_heavy_currents n =
  ([ ((L (-n), ZH, L n), FBF ((-1), Psibar, VL, Psi), G_NC_heavy);
     ((N (-n), ZH, N n), FBF (1, Psibar, VL, Psi), G_NC_heavy);
     ((U (-n), ZH, U n), FBF (1, Psibar, VL, Psi), G_NC_heavy);
     ((D (-n), ZH, D n), FBF ((-1), Psibar, VL, Psi), G_NC_heavy)]
   @
     (if Flags.u1_gauged then
   [ ((L (-n), AH, L n), FBF (1, Psibar, VA, Psi), G_NC_h_lepton);
     ((N (-n), AH, N n), FBF (1, Psibar, VA, Psi), G_NC_h_neutrino);
     ((D (-n), AH, D n), FBF (1, Psibar, VA, Psi), G_NC_h_down)]
     else
       []))

let heavy_top_currents =
  ([ ((Toppb, Ga, Topp), FBF (1, Psibar, V, Psi), Q_up);
     ((Toppb, Z, Topp), FBF (1, Psibar, V, Psi), Q_Z_up);
     ((Toppb, Z, U 3), FBF (1, Psibar, VL, Psi), G_ZTHT);
     ((U (-3), Z, Topp), FBF (1, Psibar, VL, Psi), G_ZTHT);
     ((Toppb, ZH, U 3), FBF (1, Psibar, VL, Psi), G_ZHTHT);
     ((U (-3), ZH, Topp), FBF (1, Psibar, VL, Psi), G_ZHTHT);
     ((U (-3), Wp, D 3), FBF (1, Psibar, VL, Psi), G_CCtop);
     ((D (-3), Wm, U 3), FBF (1, Psibar, VL, Psi), G_CCtop);
     ((Toppb, WHp, D 3), FBF (1, Psibar, VL, Psi), G_CC_WH);
     ((D (-3), WHm, Topp), FBF (1, Psibar, VL, Psi), G_CC_WH);
     ((Toppb, Wp, D 3), FBF (1, Psibar, VL, Psi), G_CC_W);
     ((D (-3), Wm, Topp), FBF (1, Psibar, VL, Psi), G_CC_W)]
     @
       (if Flags.u1_gauged then
   [ ((U (-3), AH, U 3), FBF (1, Psibar, VA, Psi), G_AHTT);
     ((Toppb, AH, Topp), FBF (1, Psibar, VA, Psi), G_AHTHTH);
     ((Toppb, AH, U 3), FBF (1, Psibar, VR, Psi), G_AHTHT);
     ((U (-3), AH, Topp), FBF (1, Psibar, VR, Psi), G_AHTHT)]
       else
         []))
```

$$\mathcal{L}_{\text{CC}} = -\frac{g}{2\sqrt{2}}\sum_i \bar\psi_i (T^+ \not W^+ + T^- \not W^-)(1-\gamma_5)\psi_i \tag{13.44}$$

```
let charged_currents n =
```

```
    [ ((L (-n), Wm, N n), FBF (1, Psibar, VL, Psi), G_CC);
      ((N (-n), Wp, L n), FBF (1, Psibar, VL, Psi), G_CC);
      ((L (-n), WHm, N n), FBF (1, Psibar, VL, Psi), G_CC_heavy);
      ((N (-n), WHp, L n), FBF (1, Psibar, VL, Psi), G_CC_heavy);
      ((D (-n), WHm, U n), FBF (1, Psibar, VL, Psi), G_CC_heavy);
      ((U (-n), WHp, D n), FBF (1, Psibar, VL, Psi), G_CC_heavy)]

let quark_currents n =
   ([ ((D (-n), Wm, U n), FBF (1, Psibar, VL, Psi), G_CC);
      ((U (-n), Wp, D n), FBF (1, Psibar, VL, Psi), G_CC)]
      @
        (if Flags.u1_gauged then
    [ ((U (-n), AH, U n), FBF (1, Psibar, VA, Psi), G_NC_h_up)]
        else
          []))
```

We specialize the third generation since there is an additional shift coming from the admixture of the heavy top quark. The universal shift, coming from the mixing in the non-Abelian gauge boson sector is unobservable. (Redefinition of coupling constants by measured ones.

```
let yukawa =
   [ ((U (-3), H, U 3), FBF (1, Psibar, S, Psi), G_Htt);
     ((D (-3), H, D 3), FBF (1, Psibar, S, Psi), G_Hbb);
     ((U (-2), H, U 2), FBF (1, Psibar, S, Psi), G_Hcc);
     ((L (-3), H, L 3), FBF (1, Psibar, S, Psi), G_Htautau)]

let yukawa_add' =
   [ ((Toppb, H, Topp), FBF (1, Psibar, S, Psi), G_Hthth);
     ((Toppb, H, U 3), FBF (1, Psibar, SLR, Psi), G_Htht);
     ((U (-3), H, Topp), FBF (1, Psibar, SLR, Psi), G_Htht);
     ((U (-3), Psi0, U 3), FBF (1, Psibar, S, Psi), G_Psi0tt);
     ((D (-3), Psi0, D 3), FBF (1, Psibar, S, Psi), G_Psi0bb);
     ((U (-2), Psi0, U 2), FBF (1, Psibar, S, Psi), G_Psi0cc);
     ((L (-3), Psi0, L 3), FBF (1, Psibar, S, Psi), G_Psi0tautau);
     ((U (-3), Psi1, U 3), FBF (1, Psibar, P, Psi), G_Psi1tt);
     ((D (-3), Psi1, D 3), FBF (1, Psibar, P, Psi), G_Psi1bb);
     ((U (-2), Psi1, U 2), FBF (1, Psibar, P, Psi), G_Psi1cc);
     ((L (-3), Psi1, L 3), FBF (1, Psibar, P, Psi), G_Psi1tautau);
     ((U (-3), Psip, D 3), FBF (1, Psibar, SLR, Psi), G_Psipq3);
     ((U (-2), Psip, D 2), FBF (1, Psibar, SLR, Psi), G_Psipq2);
     ((N (-3), Psip, L 3), FBF (1, Psibar, SR, Psi), G_Psipl3);
     ((D (-3), Psim, U 3), FBF (1, Psibar, SLR, Psi), G_Psipq3);
     ((D (-2), Psim, U 2), FBF (1, Psibar, SLR, Psi), G_Psipq2);
     ((L (-3), Psim, N 3), FBF (1, Psibar, SL, Psi), G_Psipl3);
     ((Toppb, Psi0, U 3), FBF (1, Psibar, SL, Psi), G_Psi0tth);
     ((U (-3), Psi0, Topp), FBF (1, Psibar, SR, Psi), G_Psi0tth);
     ((Toppb, Psi1, U 3), FBF (1, Psibar, SL, Psi), G_Psi1tth);
     ((U (-3), Psi1, Topp), FBF (1, Psibar, SR, Psi), G_Psi1tth);
     ((Toppb, Psip, D 3), FBF (1, Psibar, SL, Psi), G_Psipbth);
     ((D (-3), Psim, Topp), FBF (1, Psibar, SR, Psi), G_Psipbth)]
```

```
let yukawa_add =
  if Flags.u1_gauged then
    yukawa_add'
  else
    yukawa_add' @
  [ ((U (−3), Eta, U 3), FBF (1, Psibar, P, Psi), G_Ett);
    ((Toppb, Eta, U 3), FBF (1, Psibar, SLR, Psi), G_Etht);
    ((D (−3), Eta, D 3), FBF (1, Psibar, P, Psi), G_Ebb);
    ((U (−3), Eta, Topp), FBF (1, Psibar, SLR, Psi), G_Etht)]
```

$$\mathcal{L}_{\mathrm{TGC}} = -e\partial_\mu A_\nu W_+^\mu W_-^\nu + \ldots - e\cot\theta_w \partial_\mu Z_\nu W_+^\mu W_-^\nu + \ldots \tag{13.45}$$

Check.

```
let standard_triple_gauge =
    [ ((Ga, Wm, Wp), Gauge_Gauge_Gauge 1, I_Q_W);
      ((Z, Wm, Wp), Gauge_Gauge_Gauge 1, I_G_ZWW);
      ((Gl, Gl, Gl), Gauge_Gauge_Gauge 1, I_Gs) ]

let heavy_triple_gauge =
   ([ ((Ga, WHm, WHp), Gauge_Gauge_Gauge 1, I_Q_W);
      ((Z, WHm, WHp), Gauge_Gauge_Gauge 1, I_G_ZWW);
      ((ZH, Wm, Wp), Gauge_Gauge_Gauge 1, I_G_ZHWW);
      ((Z, WHm, Wp), Gauge_Gauge_Gauge 1, I_G_ZWHW);
      ((Z, Wm, WHp), Gauge_Gauge_Gauge (−1), I_G_ZWHW);
      ((ZH, WHm, Wp), Gauge_Gauge_Gauge 1, I_G_WWW);
      ((ZH, Wm, WHp), Gauge_Gauge_Gauge (−1), I_G_WWW);
      ((ZH, WHm, WHp), Gauge_Gauge_Gauge (−1), I_G_ZHWHWH)]
      @
        (if Flags.u1_gauged then
    [ ((AH, Wm, Wp), Gauge_Gauge_Gauge 1, I_G_AHWW);
      ((AH, WHm, Wp), Gauge_Gauge_Gauge 1, I_G_AHWHW);
      ((AH, Wm, WHp), Gauge_Gauge_Gauge (−1), I_G_AHWHW);
      ((AH, WHm, WHp), Gauge_Gauge_Gauge 1, I_G_AHWHWH)]
        else
          []))

let triple_gauge =
    standard_triple_gauge @ heavy_triple_gauge

let gauge4 = Vector4 [(2, C_13_42); (−1, C_12_34); (−1, C_14_23)]
let minus_gauge4 = Vector4 [(−2, C_13_42); (1, C_12_34); (1, C_14_23)]
let standard_quartic_gauge =
    [ (Wm, Wp, Wm, Wp), gauge4, G_WWWW;
      (Wm, Z, Wp, Z), minus_gauge4, G_ZZWW;
      (Wm, Z, Wp, Ga), minus_gauge4, G_AZWW;
      (Wm, Ga, Wp, Ga), minus_gauge4, G_AAWW;
      (Gl, Gl, Gl, Gl), gauge4, G2]

let heavy_quartic_gauge =
 [ (WHm, Wp, WHm, Wp), gauge4, G_WWWW;
    (Wm, WHp, Wm, WHp), gauge4, G_WWWW;
```

```
    (WHm, WHp, WHm, WHp), gauge4, G_WH4;
    (Wm, Wp, WHm, WHp), gauge4, G_WHWHWW;
    (Wm, Wp, Wm, WHp), gauge4, G_WHWWW;
    (Wm, Wp, WHm, Wp), gauge4, G_WHWWW;
    (WHm, WHp, Wm, WHp), gauge4, G_WH3W;
    (WHm, WHp, WHm, Wp), gauge4, G_WH3W;
    (WHm, Z, WHp, Z), minus_gauge4, G_ZZWW;
    (WHm, Z, WHp, Ga), minus_gauge4, G_AZWW;
    (WHm, Ga, WHp, ZH), minus_gauge4, G_AAWW;
    (WHm, Z, WHp, ZH), minus_gauge4, G_ZZWW;
    (Wm, ZH, Wp, ZH), minus_gauge4, G_WWWW;
    (Wm, Ga, Wp, ZH), minus_gauge4, G_WWAZH;
    (Wm, Z, Wp, ZH), minus_gauge4, G_WWZZH;
    (WHm, Ga, WHp, ZH), minus_gauge4, G_WHWHAZH;
    (WHm, Z, WHp, ZH), minus_gauge4, G_WHWHZZH;
    (WHm, ZH, WHp, ZH), minus_gauge4, G_WH4;
    (WHm, Z, Wp, Z), minus_gauge4, G_WHWZZ;
    (Wm, Z, WHp, Z), minus_gauge4, G_WHWZZ;
    (WHm, Ga, Wp, Z), minus_gauge4, G_WHWAZ;
    (Wm, Ga, WHp, Z), minus_gauge4, G_WHWAZ;
    (WHm, ZH, Wp, ZH), minus_gauge4, G_WHWZHZH;
    (Wm, ZH, WHp, ZH), minus_gauge4, G_WHWZHZH;
    (WHm, Ga, Wp, ZH), minus_gauge4, G_WHWAZH;
    (Wm, Ga, WHp, ZH), minus_gauge4, G_WHWAZH;
    (WHm, Z, Wp, ZH), minus_gauge4, G_WHWZZH;
    (Wm, Z, WHp, ZH), minus_gauge4, G_WHWZZH]
    @
      (if Flags.u1_gauged then
  [ (Wm, Ga, Wp, AH), minus_gauge4, G_WWAAH;
    (Wm, Z, Wp, AH), minus_gauge4, G_WWZAH;
    (WHm, Ga, WHp, AH), minus_gauge4, G_WHWHAAH;
    (WHm, Z, WHp, AH), minus_gauge4, G_WHWHZAH;
    (Wm, ZH, Wp, AH), minus_gauge4, G_WWZHAH;
    (WHm, ZH, WHp, AH), minus_gauge4, G_WHWHZHAH;
    (WHm, Ga, Wp, AH), minus_gauge4, G_WHWAAH;
    (Wm, Ga, WHp, AH), minus_gauge4, G_WHWAAH;
    (WHm, Z, Wp, AH), minus_gauge4, G_WHWZAH;
    (Wm, Z, WHp, AH), minus_gauge4, G_WHWZAH;
    (WHm, ZH, Wp, AH), minus_gauge4, G_WHWZHAH;
    (Wm, ZH, WHp, AH), minus_gauge4, G_WHWZHAH]
      else
        [])

let quartic_gauge =
  standard_quartic_gauge @ heavy_quartic_gauge

let standard_gauge_higgs' =
  [ ((H, Wp, Wm), Scalar_Vector_Vector 1, G_HWW);
    ((H, Z, Z), Scalar_Vector_Vector 1, G_HZZ) ]

let heavy_gauge_higgs =
  [ ((H, Wp, WHm), Scalar_Vector_Vector 1, G_HWHW);
```

```
    ((H, WHp, Wm), Scalar_Vector_Vector 1, G_HWHW);
    ((H, WHp, WHm), Scalar_Vector_Vector 1, G_HWHWH);
    ((H, ZH, ZH), Scalar_Vector_Vector 1, G_HWHWH);
    ((H, ZH, Z), Scalar_Vector_Vector 1, G_HZHZ);
    ((H, Wp, Wm), Scalar_Vector_Vector 1, G_HZHAH)]
    @
      (if Flags.u1_gauged then
   [((H, AH, AH), Scalar_Vector_Vector 1, G_HAHAH);
    ((H, Z, AH), Scalar_Vector_Vector 1, G_HAHZ)]
      else
        [])

let triplet_gauge_higgs =
  [ ((Psi0, Wp, Wm), Scalar_Vector_Vector 1, G_PsiWW);
    ((Psi0, WHp, WHm), Scalar_Vector_Vector (-1), G_PsiWW);
    ((Psi0, WHp, Wm), Scalar_Vector_Vector 1, G_PsiWHW);
    ((Psi0, WHm, Wp), Scalar_Vector_Vector 1, G_PsiWHW);
    ((Psi0, Z, Z), Scalar_Vector_Vector 1, G_PsiZZ);
    ((Psi0, ZH, ZH), Scalar_Vector_Vector 1, G_PsiZHZH);
    ((Psi0, ZH, Z), Scalar_Vector_Vector 1, G_PsiZHZ);
    ((Psim, Wp, Z), Scalar_Vector_Vector 1, G_PsiZW);
    ((Psip, Wm, Z), Scalar_Vector_Vector 1, G_PsiZW);
    ((Psim, WHp, Z), Scalar_Vector_Vector 1, G_PsiZWH);
    ((Psip, WHm, Z), Scalar_Vector_Vector 1, G_PsiZWH);
    ((Psim, Wp, ZH), Scalar_Vector_Vector 1, G_PsiZHW);
    ((Psip, Wm, ZH), Scalar_Vector_Vector 1, G_PsiZHW);
    ((Psim, WHp, ZH), Scalar_Vector_Vector 1, G_PsiZHWH);
    ((Psip, WHm, ZH), Scalar_Vector_Vector 1, G_PsiZHWH);
    ((Psimm, Wp, Wp), Scalar_Vector_Vector 1, G_PsippWW);
    ((Psipp, Wm, Wm), Scalar_Vector_Vector 1, G_PsippWW);
    ((Psimm, WHp, Wp), Scalar_Vector_Vector 1, G_PsippWHW);
    ((Psipp, WHm, Wm), Scalar_Vector_Vector 1, G_PsippWHW);
    ((Psimm, WHp, WHp), Scalar_Vector_Vector 1, G_PsippWHWH);
    ((Psipp, WHm, WHm), Scalar_Vector_Vector 1, G_PsippWHWH)]
    @
      (if Flags.u1_gauged then
   [((Psi0, AH, Z), Scalar_Vector_Vector 1, G_PsiZAH);
    ((Psi0, AH, ZH), Scalar_Vector_Vector 1, G_PsiZHAH);
    ((Psi0, AH, AH), Scalar_Vector_Vector 1, G_PsiAHAH);
    ((Psim, Wp, AH), Scalar_Vector_Vector 1, G_PsiAHW);
    ((Psip, Wm, AH), Scalar_Vector_Vector 1, G_PsiAHW);
    ((Psim, WHp, AH), Scalar_Vector_Vector 1, G_PsiAHWH);
    ((Psip, WHm, AH), Scalar_Vector_Vector 1, G_PsiAHWH)]
      else
        [])

let triplet_gauge2_higgs =
  [ ((Wp, H, Psim), Vector_Scalar_Scalar 1, G_PsiHW);
    ((Wm, H, Psip), Vector_Scalar_Scalar 1, G_PsiHW);
    ((WHp, H, Psim), Vector_Scalar_Scalar 1, G_PsiHWH);
    ((WHm, H, Psip), Vector_Scalar_Scalar 1, G_PsiHWH);
```

```
    ((Wp, Psi0, Psim), Vector_Scalar_Scalar 1, G_Psi0W);
    ((Wm, Psi0, Psip), Vector_Scalar_Scalar 1, G_Psi0W);
    ((WHp, Psi0, Psim), Vector_Scalar_Scalar 1, G_Psi0WH);
    ((WHm, Psi0, Psip), Vector_Scalar_Scalar 1, G_Psi0WH);
    ((Wp, Psi1, Psim), Vector_Scalar_Scalar 1, G_Psi1W);
    ((Wm, Psi1, Psip), Vector_Scalar_Scalar (-1), G_Psi1W);
    ((WHp, Psi1, Psim), Vector_Scalar_Scalar 1, G_Psi1WH);
    ((WHm, Psi1, Psip), Vector_Scalar_Scalar (-1), G_Psi1WH);
    ((Wp, Psip, Psimm), Vector_Scalar_Scalar 1, G_PsiPPW);
    ((Wm, Psim, Psipp), Vector_Scalar_Scalar 1, G_PsiPPW);
    ((WHp, Psip, Psimm), Vector_Scalar_Scalar 1, G_PsiPPWH);
    ((WHm, Psim, Psipp), Vector_Scalar_Scalar 1, G_PsiPPWH);
    ((Ga, Psip, Psim), Vector_Scalar_Scalar 1, Q_lepton);
    ((Ga, Psipp, Psimm), Vector_Scalar_Scalar 2, Q_lepton);
    ((Z, H, Psi1), Vector_Scalar_Scalar 1, G_Psi1HZ);
    ((ZH, H, Psi1), Vector_Scalar_Scalar 1, G_Psi1HZH);
    ((Z, Psi0, Psi1), Vector_Scalar_Scalar 1, G_Psi01Z);
    ((ZH, Psi0, Psi1), Vector_Scalar_Scalar 1, G_Psi01ZH);
    ((Z, Psip, Psim), Vector_Scalar_Scalar 1, G_ZPsip);
    ((Z, Psipp, Psimm), Vector_Scalar_Scalar 2, G_ZPsipp);
    ((ZH, Psipp, Psimm), Vector_Scalar_Scalar 2, G_ZHPsipp)]
    @
      (if Flags.u1_gauged then
   [((AH, H, Psi1), Vector_Scalar_Scalar 1, G_Psi1HAH);
    ((AH, Psi0, Psi1), Vector_Scalar_Scalar 1, G_Psi01AH);
    ((AH, Psip, Psim), Vector_Scalar_Scalar 1, G_AHPsip);
    ((AH, Psipp, Psimm), Vector_Scalar_Scalar 2, G_AHPsip)]
      else [])

let standard_gauge_higgs =
  standard_gauge_higgs' @ heavy_gauge_higgs @ triplet_gauge_higgs @
  triplet_gauge2_higgs

let standard_gauge_higgs4 =
  [ (H, H, Wp, Wm), Scalar2_Vector2 1, G_HHWW;
    (H, H, Z, Z), Scalar2_Vector2 1, G_HHZZ ]

let littlest_gauge_higgs4 =
  [ (H, H, WHp, WHm), Scalar2_Vector2 (-1), G_HHWW;
    (H, H, ZH, ZH), Scalar2_Vector2 (-1), G_HHWW;
    (H, H, Wp, WHm), Scalar2_Vector2 1, G_HHWHW;
    (H, H, WHp, Wm), Scalar2_Vector2 1, G_HHWHW;
    (H, H, ZH, Z), Scalar2_Vector2 (-1), G_HHZHZ;
    (H, Psi0, Wp, Wm), Scalar2_Vector2 1, G_HPsi0WW;
    (H, Psi0, WHp, WHm), Scalar2_Vector2 (-1), G_HPsi0WW;
    (H, Psi0, WHp, Wm), Scalar2_Vector2 1, G_HPsi0WHW;
    (H, Psi0, Wp, WHm), Scalar2_Vector2 1, G_HPsi0WHW;
    (H, Psi0, Z, Z), Scalar2_Vector2 1, G_HPsi0ZZ;
    (H, Psi0, ZH, ZH), Scalar2_Vector2 1, G_HPsi0ZHZH;
    (H, Psi0, ZH, Z), Scalar2_Vector2 1, G_HPsi0ZHZ;
    (H, Psim, Wp, Ga), Scalar2_Vector2 1, G_HPsipWA;
    (H, Psip, Wm, Ga), Scalar2_Vector2 1, G_HPsipWA;
```

$(H,\ Psim,\ WHp,\ Ga),\ Scalar2\_Vector2\ 1,\ G\_HPsipWHA;$
$(H,\ Psip,\ WHm,\ Ga),\ Scalar2\_Vector2\ 1,\ G\_HPsipWHA;$
$(H,\ Psim,\ Wp,\ Z),\ Scalar2\_Vector2\ 1,\ G\_HPsipWZ;$
$(H,\ Psip,\ Wm,\ Z),\ Scalar2\_Vector2\ 1,\ G\_HPsipWZ;$
$(H,\ Psim,\ WHp,\ Z),\ Scalar2\_Vector2\ 1,\ G\_HPsipWHZ;$
$(H,\ Psip,\ WHm,\ Z),\ Scalar2\_Vector2\ 1,\ G\_HPsipWHZ;$
$(H,\ Psim,\ Wp,\ ZH),\ Scalar2\_Vector2\ 1,\ G\_HPsipWZH;$
$(H,\ Psip,\ Wm,\ ZH),\ Scalar2\_Vector2\ 1,\ G\_HPsipWZH;$
$(H,\ Psim,\ WHp,\ ZH),\ Scalar2\_Vector2\ 1,\ G\_HPsipWHZH;$
$(H,\ Psip,\ WHm,\ ZH),\ Scalar2\_Vector2\ 1,\ G\_HPsipWHZH;$
$(H,\ Psimm,\ Wp,\ Wp),\ Scalar2\_Vector2\ 1,\ G\_HPsippWW;$
$(H,\ Psipp,\ Wm,\ Wm),\ Scalar2\_Vector2\ 1,\ G\_HPsippWW;$
$(H,\ Psimm,\ WHp,\ WHp),\ Scalar2\_Vector2\ 1,\ G\_HPsippWHWH;$
$(H,\ Psipp,\ WHm,\ WHm),\ Scalar2\_Vector2\ 1,\ G\_HPsippWHWH;$
$(H,\ Psimm,\ WHp,\ Wp),\ Scalar2\_Vector2\ 1,\ G\_HPsippWHW;$
$(H,\ Psipp,\ WHm,\ Wm),\ Scalar2\_Vector2\ 1,\ G\_HPsippWHW;$
$(Psi0,\ Psi0,\ Wp,\ Wm),\ Scalar2\_Vector2\ 2,\ G\_HHWW;$
$(Psi0,\ Psi0,\ WHp,\ WHm),\ Scalar2\_Vector2\ (-2),\ G\_HHWW;$
$(Psi0,\ Psi0,\ Z,\ Z),\ Scalar2\_Vector2\ 4,\ G\_HHZZ;$
$(Psi0,\ Psi0,\ ZH,\ ZH),\ Scalar2\_Vector2\ 1,\ G\_Psi00ZH;$
$(Psi0,\ Psi0,\ WHp,\ Wm),\ Scalar2\_Vector2\ 2,\ G\_HHWHW;$
$(Psi0,\ Psi0,\ Wp,\ WHm),\ Scalar2\_Vector2\ 2,\ G\_HHWHW;$
$(Psi0,\ Psi0,\ Z,\ ZH),\ Scalar2\_Vector2\ 4,\ G\_HHZHZ;$
$(Psi0,\ Psim,\ Wp,\ Ga),\ Scalar2\_Vector2\ 1,\ G\_Psi0pWA;$
$(Psi0,\ Psip,\ Wm,\ Ga),\ Scalar2\_Vector2\ 1,\ G\_Psi0pWA;$
$(Psi0,\ Psim,\ WHp,\ Ga),\ Scalar2\_Vector2\ 1,\ G\_Psi0pWHA;$
$(Psi0,\ Psip,\ WHm,\ Ga),\ Scalar2\_Vector2\ 1,\ G\_Psi0pWHA;$
$(Psi0,\ Psim,\ Wp,\ Z),\ Scalar2\_Vector2\ 1,\ G\_Psi0pWZ;$
$(Psi0,\ Psip,\ Wm,\ Z),\ Scalar2\_Vector2\ 1,\ G\_Psi0pWZ;$
$(Psi0,\ Psim,\ WHp,\ Z),\ Scalar2\_Vector2\ 1,\ G\_Psi0pWHZ;$
$(Psi0,\ Psip,\ WHm,\ Z),\ Scalar2\_Vector2\ 1,\ G\_Psi0pWHZ;$
$(Psi0,\ Psim,\ Wp,\ ZH),\ Scalar2\_Vector2\ 1,\ G\_Psi0pWZH;$
$(Psi0,\ Psip,\ Wm,\ ZH),\ Scalar2\_Vector2\ 1,\ G\_Psi0pWZH;$
$(Psi0,\ Psim,\ WHp,\ ZH),\ Scalar2\_Vector2\ 1,\ G\_Psi0pWHZH;$
$(Psi0,\ Psip,\ WHm,\ ZH),\ Scalar2\_Vector2\ 1,\ G\_Psi0pWHZH;$
$(Psi0,\ Psimm,\ Wp,\ Wp),\ Scalar2\_Vector2\ 1,\ G\_Psi0ppWW;$
$(Psi0,\ Psipp,\ Wm,\ Wm),\ Scalar2\_Vector2\ 1,\ G\_Psi0ppWW;$
$(Psi0,\ Psimm,\ WHp,\ WHp),\ Scalar2\_Vector2\ 1,\ G\_Psi0ppWHWH;$
$(Psi0,\ Psipp,\ WHm,\ WHm),\ Scalar2\_Vector2\ 1,\ G\_Psi0ppWHWH;$
$(Psi0,\ Psimm,\ WHp,\ Wp),\ Scalar2\_Vector2\ 1,\ G\_Psi0ppWHW;$
$(Psi0,\ Psipp,\ WHm,\ Wm),\ Scalar2\_Vector2\ 1,\ G\_Psi0ppWHW;$
$(Psi1,\ Psi1,\ Wp,\ Wm),\ Scalar2\_Vector2\ 2,\ G\_HHWW;$
$(Psi1,\ Psi1,\ WHp,\ WHm),\ Scalar2\_Vector2\ (-2),\ G\_HHWW;$
$(Psi1,\ Psi1,\ Z,\ Z),\ Scalar2\_Vector2\ 4,\ G\_HHZZ;$
$(Psi1,\ Psi1,\ ZH,\ ZH),\ Scalar2\_Vector2\ 1,\ G\_Psi00ZH;$
$(Psi1,\ Psi1,\ WHp,\ Wm),\ Scalar2\_Vector2\ 2,\ G\_HHWHW;$
$(Psi1,\ Psi1,\ Wp,\ WHm),\ Scalar2\_Vector2\ 2,\ G\_HHWHW;$
$(Psi1,\ Psi1,\ Z,\ ZH),\ Scalar2\_Vector2\ 4,\ G\_HHZHZ;$
$(Psi1,\ Psim,\ Wp,\ Ga),\ Scalar2\_Vector2\ 1,\ I\_G\_Psi0pWA;$
$(Psi1,\ Psip,\ Wm,\ Ga),\ Scalar2\_Vector2\ (-1),\ I\_G\_Psi0pWA;$

```
(Psi1, Psim, WHp, Ga), Scalar2_Vector2 1, I_G_Psi0pWHA;
(Psi1, Psip, WHm, Ga), Scalar2_Vector2 (−1), I_G_Psi0pWHA;
(Psi1, Psim, Wp, Z), Scalar2_Vector2 1, I_G_Psi0pWZ;
(Psi1, Psip, Wm, Z), Scalar2_Vector2 (−1), I_G_Psi0pWZ;
(Psi1, Psim, WHp, Z), Scalar2_Vector2 1, I_G_Psi0pWHZ;
(Psi1, Psip, WHm, Z), Scalar2_Vector2 (−1), I_G_Psi0pWHZ;
(Psi1, Psim, Wp, ZH), Scalar2_Vector2 1, I_G_Psi0pWZH;
(Psi1, Psip, Wm, ZH), Scalar2_Vector2 (−1), I_G_Psi0pWZH;
(Psi1, Psim, WHp, ZH), Scalar2_Vector2 1, I_G_Psi0pWHZH;
(Psi1, Psip, WHm, ZH), Scalar2_Vector2 (−1), I_G_Psi0pWHZH;
(Psi1, Psimm, Wp, Wp), Scalar2_Vector2 1, I_G_Psi0ppWW;
(Psi1, Psipp, Wm, Wm), Scalar2_Vector2 (−1), I_G_Psi0ppWW;
(Psi1, Psimm, WHp, WHp), Scalar2_Vector2 1, I_G_Psi0ppWHWH;
(Psi1, Psipp, WHm, WHm), Scalar2_Vector2 (−1), I_G_Psi0ppWHWH;
(Psi1, Psimm, WHp, Wp), Scalar2_Vector2 1, I_G_Psi0ppWHW;
(Psi1, Psipp, WHm, Wm), Scalar2_Vector2 (−1), I_G_Psi0ppWHW;
(Psip, Psim, Wp, Wm), Scalar2_Vector2 4, G_HHWW;
(Psip, Psim, WHp, WHm), Scalar2_Vector2 1, G_Psi00ZH;
(Psip, Psim, WHp, Wm), Scalar2_Vector2 4, G_HHWHW;
(Psip, Psim, Wp, WHm), Scalar2_Vector2 4, G_HHWHW;
(Psip, Psim, Z, Z), Scalar2_Vector2 1, G_PsippZZ;
(Psip, Psim, Ga, Ga), Scalar2_Vector2 2, G_AAWW;
(Psip, Psim, ZH, ZH), Scalar2_Vector2 1, G_PsippZHZH;
(Psip, Psim, Ga, Z), Scalar2_Vector2 4, G_PsippAZ;
(Psip, Psimm, Wp, Ga), Scalar2_Vector2 1, G_PsippWA;
(Psim, Psipp, Wm, Ga), Scalar2_Vector2 1, G_PsippWA;
(Psip, Psimm, WHp, Ga), Scalar2_Vector2 1, G_PsippWHA;
(Psim, Psipp, WHm, Ga), Scalar2_Vector2 1, G_PsippWHA;
(Psip, Psimm, Wp, Z), Scalar2_Vector2 1, G_PsippWZ;
(Psim, Psipp, Wm, Z), Scalar2_Vector2 1, G_PsippWZ;
(Psip, Psimm, WHp, Z), Scalar2_Vector2 1, G_PsippWHZ;
(Psim, Psipp, WHm, Z), Scalar2_Vector2 1, G_PsippWHZ;
(Psip, Psimm, Wp, ZH), Scalar2_Vector2 1, G_PsippWZH;
(Psim, Psipp, Wm, ZH), Scalar2_Vector2 1, G_PsippWZH;
(Psip, Psimm, WHp, ZH), Scalar2_Vector2 1, G_PsippWHZH;
(Psim, Psipp, WHm, ZH), Scalar2_Vector2 1, G_PsippWHZH;
(Psipp, Psimm, Wp, Wm), Scalar2_Vector2 2, G_HHWW;
(Psipp, Psimm, WHp, WHm), Scalar2_Vector2 (−2), G_HHWW;
(Psipp, Psimm, WHp, Wm), Scalar2_Vector2 2, G_HHWHW;
(Psipp, Psimm, Wp, WHm), Scalar2_Vector2 2, G_HHWHW;
(Psipp, Psimm, Z, Z), Scalar2_Vector2 1, G_PsiccZZ;
(Psipp, Psimm, Ga, Ga), Scalar2_Vector2 8, G_AAWW;
(Psipp, Psimm, ZH, ZH), Scalar2_Vector2 1, G_Psi00ZH;
(Psipp, Psimm, Ga, Z), Scalar2_Vector2 1, G_PsiccAZ;
(Psipp, Psimm, Z, ZH), Scalar2_Vector2 4, G_PsiccZZH;
(Psipp, Psimm, Ga, ZH), Scalar2_Vector2 4, G_PsiccAZH]
@
   (if Flags.u1_gauged then
[(H, H, AH, AH), Scalar2_Vector2 1, G_HHAA;
 (H, H, AH, Z), Scalar2_Vector2 (−1), G_HHAHZ;
```

```
        (H, H, ZH, AH), Scalar2_Vector2 (-1), G_HHZHAH;
        (H, Psi0, AH, AH), Scalar2_Vector2 1, G_HPsi0AHAH;
        (H, Psi0, Z, AH), Scalar2_Vector2 1, G_HPsi0ZAH;
        (H, Psi0, ZH, AH), Scalar2_Vector2 1, G_HPsi0ZHAH;
        (H, Psim, Wp, AH), Scalar2_Vector2 1, G_HPsipWAH;
        (H, Psip, Wm, AH), Scalar2_Vector2 1, G_HPsipWAH;
        (H, Psim, WHp, AH), Scalar2_Vector2 1, G_HPsipWHAH;
        (H, Psip, WHm, AH), Scalar2_Vector2 1, G_HPsipWHAH;
        (Psi0, Psi0, AH, AH), Scalar2_Vector2 1, G_Psi00AH;
        (Psi0, Psi0, Z, AH), Scalar2_Vector2 4, G_HHAHZ;
        (Psi0, Psi0, AH, ZH), Scalar2_Vector2 1, G_Psi00ZHAH;
        (Psi0, Psim, Wp, AH), Scalar2_Vector2 1, G_Psi0pWAH;
        (Psi0, Psip, Wm, AH), Scalar2_Vector2 1, G_Psi0pWAH;
        (Psi0, Psim, WHp, AH), Scalar2_Vector2 1, G_Psi0pWHAH;
        (Psi0, Psip, WHm, AH), Scalar2_Vector2 1, G_Psi0pWHAH;
        (Psi1, Psi1, AH, AH), Scalar2_Vector2 1, G_Psi00AH;
        (Psi1, Psi1, Z, AH), Scalar2_Vector2 4, G_HHAHZ;
        (Psi1, Psi1, AH, ZH), Scalar2_Vector2 1, G_Psi00ZHAH;
        (Psi1, Psim, Wp, AH), Scalar2_Vector2 1, I_G_Psi0pWAH;
        (Psi1, Psip, Wm, AH), Scalar2_Vector2 (-1), I_G_Psi0pWAH;
        (Psi1, Psim, WHp, AH), Scalar2_Vector2 1, I_G_Psi0pWHAH;
        (Psi1, Psip, WHm, AH), Scalar2_Vector2 (-1), I_G_Psi0pWHAH;
        (Psip, Psim, AH, AH), Scalar2_Vector2 1, G_Psi00AH;
        (Psip, Psim, Ga, AH), Scalar2_Vector2 4, G_PsippAAH;
        (Psip, Psim, Z, AH), Scalar2_Vector2 4, G_PsippZAH;
        (Psip, Psimm, Wp, AH), Scalar2_Vector2 1, G_PsippWAH;
        (Psim, Psipp, Wm, AH), Scalar2_Vector2 1, G_PsippWAH;
        (Psip, Psimm, WHp, AH), Scalar2_Vector2 1, G_PsippWHAH;
        (Psim, Psipp, WHm, AH), Scalar2_Vector2 1, G_PsippWHAH;
        (Psipp, Psimm, AH, AH), Scalar2_Vector2 1, G_Psi00AH;
        (Psipp, Psimm, AH, ZH), Scalar2_Vector2 (-1), G_Psi00ZHAH;
        (Psipp, Psimm, Ga, AH), Scalar2_Vector2 4, G_PsiccAAH;
        (Psipp, Psimm, Z, AH), Scalar2_Vector2 4, G_PsiccZAH]
          else [])

    let standard_higgs =
      [ (H, H, H), Scalar_Scalar_Scalar 1, G_H3 ]

   let anomaly_higgs =
      [ (Eta, Gl, Gl), Dim5_Scalar_Gauge2_Skew 1, G_EGlGl;
        (Eta, Ga, Ga), Dim5_Scalar_Gauge2_Skew 1, G_EGaGa;
        (Eta, Ga, Z), Dim5_Scalar_Gauge2_Skew 1, G_EGaZ]
(* @ (H, Ga, Ga), Dim5_Scalar_Gauge2 1, G_HGaGa; (H, Ga, Z), Dim5_Scalar_Gauge2 1, G_HGaZ
*)

    let standard_higgs4 =
      [ (H, H, H, H), Scalar4 1, G_H4 ]

    let gauge_higgs =
        standard_gauge_higgs

    let gauge_higgs4 =
        standard_gauge_higgs4
```

```
let higgs =
    standard_higgs

let higgs4 =
    standard_higgs4

let top_quartic =
  [ ((U (-3), H, H, U 3), GBBG (1, Psibar, S2, Psi), G_HHtt);
((Toppb, H, H, Topp), GBBG (1, Psibar, S2, Psi), G_HHthth);
((U (-3), H, H, Topp), GBBG (1, Psibar, S2LR, Psi), G_HHtht);
((Toppb, H, H, U 3), GBBG (1, Psibar, S2LR, Psi), G_HHtht)]

let goldstone_vertices =
  [ ((Phi0, Wm, Wp), Scalar_Vector_Vector 1, I_G_ZWW);
    ((Phip, Ga, Wm), Scalar_Vector_Vector 1, I_Q_W);
    ((Phip, Z, Wm), Scalar_Vector_Vector 1, I_G_ZWW);
    ((Phim, Wp, Ga), Scalar_Vector_Vector 1, I_Q_W);
    ((Phim, Wp, Z), Scalar_Vector_Vector 1, I_G_ZWW) ]

let vertices3 =
  (ThoList.flatmap electromagnetic_currents [1; 2; 3] @
   ThoList.flatmap neutral_currents [1; 2; 3] @
   ThoList.flatmap color_currents [1; 2; 3] @
   ThoList.flatmap neutral_heavy_currents [1; 2; 3] @
   ThoList.flatmap charged_currents [1; 2; 3] @
   ThoList.flatmap quark_currents [1; 2] @
   heavy_top_currents @
   (if Flags.u1_gauged then [ ]
       else anomaly_higgs) @
   yukawa @ yukawa_add @ triple_gauge @
   gauge_higgs @ higgs @ goldstone_vertices)

let vertices4 =
  quartic_gauge @ gauge_higgs4 @ higgs4 @ top_quartic

let vertices () = (vertices3, vertices4, [])
```

For efficiency, make sure that *F.of_vertices vertices* is evaluated only once.

```
let table = F.of_vertices (vertices ())
let fuse2 = F.fuse2 table
let fuse3 = F.fuse3 table
let fuse = F.fuse table
let max_degree () = 4

let flavor_of_string = function
  | "e-" → L 1 | "e+" → L (-1)
  | "mu-" → L 2 | "mu+" → L (-2)
  | "tau-" → L 3 | "tau+" → L (-3)
  | "nue" → N 1 | "nuebar" → N (-1)
  | "numu" → N 2 | "numubar" → N (-2)
  | "nutau" → N 3 | "nutaubar" → N (-3)
  | "u" → U 1 | "ubar" → U (-1)
  | "c" → U 2 | "cbar" → U (-2)
  | "t" → U 3 | "tbar" → U (-3)
```

```
| "d" → D 1 | "dbar" → D (−1)
| "s" → D 2 | "sbar" → D (−2)
| "b" → D 3 | "bbar" → D (−3)
| "tp" → Topp | "tpbar" → Toppb
| "g" → Gl
| "A" → Ga | "Z" | "Z0" → Z
| "AH" | "AH0" | "Ah" | "Ah0" → AH
| "ZH" | "ZH0" | "Zh" | "Zh0" → ZH
| "W+" → Wp | "W-" → Wm
| "WH+" → WHp | "WH-" → WHm
| "H" | "h" → H | "eta" | "Eta" → Eta
| "Psi" | "Psi0" | "psi" | "psi0" → Psi0
| "Psi1" | "psi1" → Psi1
| "Psi+" | "psi+" | "Psip" | "psip" → Psip
| "Psi-" | "psi-" | "Psim" | "psim" → Psim
| "Psi++" | "psi++" | "Psipp" | "psipp" → Psipp
| "Psi--" | "psi--" | "Psimm" | "psimm" → Psimm
| _ → invalid_arg "Modellib_BSM.Littlest_Tpar.flavor_of_string"

let flavor_to_string = function
| L 1 → "e-" | L (−1) → "e+"
| L 2 → "mu-" | L (−2) → "mu+"
| L 3 → "tau-" | L (−3) → "tau+"
| L _ → invalid_arg "Modellib_BSM.Littlest_Tpar.flavor_to_string"
| N 1 → "nue" | N (−1) → "nuebar"
| N 2 → "numu" | N (−2) → "numubar"
| N 3 → "nutau" | N (−3) → "nutaubar"
| N _ → invalid_arg "Modellib_BSM.Littlest_Tpar.flavor_to_string"
| Lodd 1 → "l1odd-" | Lodd (−1) → "l1odd+"
| Lodd 2 → "l2odd-" | Lodd (−2) → "l2odd+"
| Lodd 3 → "l3odd-" | Lodd (−3) → "l3odd+"
| Lodd _ → invalid_arg "Modellib_BSM.Littlest_Tpar.flavor_to_string"
| Nodd 1 → "n1odd" | Nodd (−1) → "n1oddbar"
| Nodd 2 → "n2odd" | Nodd (−2) → "n2oddbar"
| Nodd 3 → "n3odd" | Nodd (−3) → "n3oddbar"
| Nodd _ → invalid_arg "Modellib_BSM.Littlest_Tpar.flavor_to_string"
| U 1 → "u" | U (−1) → "ubar"
| U 2 → "c" | U (−2) → "cbar"
| U 3 → "t" | U (−3) → "tbar"
| U _ → invalid_arg "Modellib_BSM.Littlest_Tpar.flavor_to_string"
| D 1 → "d" | D (−1) → "dbar"
| D 2 → "s" | D (−2) → "sbar"
| D 3 → "b" | D (−3) → "bbar"
| D _ → invalid_arg "Modellib_BSM.Littlest_Tpar.flavor_to_string"
| Uodd 1 → "uodd" | Uodd (−1) → "uoddbar"
| Uodd 2 → "codd" | Uodd (−2) → "coddbar"
| Uodd 3 → "t1odd" | Uodd (−3) → "t1oddbar"
| Uodd 4 → "t2odd" | Uodd (−4) → "t2oddbar"
| Uodd _ → invalid_arg "Modellib_BSM.Littlest_Tpar.flavor_to_string"
| Dodd 1 → "dodd" | Dodd (−1) → "doddbar"
```

```
| Dodd 2 → "sodd" | Dodd (−2) → "soddbar"
| Dodd 3 → "bodd" | Dodd (−3) → "boddbar"
| Dodd _ → invalid_arg "Modellib_BSM.Littlest_Tpar.flavor_to_string"
| Topp → "tp" | Toppb → "tpbar"
| Gl → "g"
| Ga → "A" | Z → "Z"
| Wp → "W+" | Wm → "W-"
| ZH → "ZH" | AH → "AH" | WHp → "WHp" | WHm → "WHm"
| Phip → "phi+" | Phim → "phi-" | Phi0 → "phi0"
| H → "H" | Eta → "Eta"
| Psi0 → "Psi0" | Psi1 → "Psi1" | Psip → "Psi+"
| Psim → "Psi-" | Psipp → "Psi++" | Psimm → "Psi--"

let flavor_to_TeX = function
  | L 1 → "e^-" | L (−1) → "e^+"
  | L 2 → "\\mu^-" | L (−2) → "\\mu^+"
  | L 3 → "\\tau^-" | L (−3) → "\\tau^+"
  | L _ → invalid_arg "Modellib_BSM.Littlest_Tpar.flavor_to_TeX"
  | N 1 → "\\nu_e" | N (−1) → "\\bar{\\nu}_e"
  | N 2 → "\\nu_\\mu" | N (−2) → "\\bar{\\nu}_\\mu"
  | N 3 → "\\nu_\\tau" | N (−3) → "\\bar{\\nu}_\\tau"
  | N _ → invalid_arg "Modellib_BSM.Littlest_Tpar.flavor_to_TeX"
  | Lodd 1 → "L_1^-" | Lodd (−1) → "L_1^+"
  | Lodd 2 → "L_2^-" | Lodd (−2) → "L_2^+"
  | Lodd 3 → "L_3^-" | Lodd (−3) → "L_3^+"
  | Lodd _ → invalid_arg "Modellib_BSM.Littlest_Tpar.flavor_to_TeX"
  | Nodd 1 → "N_1" | Nodd (−1) → "\\bar{N}_1"
  | Nodd 2 → "N_2" | Nodd (−2) → "\\bar{N}_2"
  | Nodd 3 → "N_3" | Nodd (−3) → "\\bar{N}_3"
  | Nodd _ → invalid_arg "Modellib_BSM.Littlest_Tpar.flavor_to_TeX"
  | U 1 → "u" | U (−1) → "\\bar{u}"
  | U 2 → "c" | U (−2) → "\\bar{c}"
  | U 3 → "t" | U (−3) → "\\bar{t}"
  | U _ → invalid_arg "Modellib_BSM.Littlest_Tpar.flavor_to_TeX"
  | D 1 → "d" | D (−1) → "\\bar{d}"
  | D 2 → "s" | D (−2) → "\\bar{s}"
  | D 3 → "b" | D (−3) → "\\bar{b}"
  | D _ → invalid_arg "Modellib_BSM.Littlest_Tpar.flavor_to_TeX"
  | Uodd 1 → "U" | Uodd (−1) → "\\bar{U}"
  | Uodd 2 → "C" | Uodd (−2) → "\\bar{C}"
  | Uodd 3 → "T_1" | Uodd (−3) → "\\bar{T}_1"
  | Uodd 4 → "T_2" | Uodd (−4) → "\\bar{T}_2"
  | Uodd _ → invalid_arg "Modellib_BSM.Littlest_Tpar.flavor_to_TeX"
  | Dodd 1 → "D" | Dodd (−1) → "\\bar{D}"
  | Dodd 2 → "S" | Dodd (−2) → "\\bar{S}"
  | Dodd 3 → "B" | Dodd (−3) → "\\bar{B}"
  | Dodd _ → invalid_arg "Modellib_BSM.Littlest_Tpar.flavor_to_TeX"
  | Topp → "T^\\prime" | Toppb → "\\bar{T}^\\prime"
  | Gl → "g"
  | Ga → "\\gamma" | Z → "Z"
```

$|$ *Wp* $\rightarrow$ "W^+" $|$ *Wm* $\rightarrow$ "W^-"
$|$ *ZH* $\rightarrow$ "Z_H" $|$ *AH* $\rightarrow$ "\\gamma_H" $|$ *WHp* $\rightarrow$ "W_H^+" $|$ *WHm* $\rightarrow$ "W_H^-"
$|$ *Phip* $\rightarrow$ "\\Phi^+" $|$ *Phim* $\rightarrow$ "\\Phi^-" $|$ *Phi0* $\rightarrow$ "\\Phi^0"
$|$ *H* $\rightarrow$ "H" $|$ *Eta* $\rightarrow$ "\\eta"
$|$ *Psi0* $\rightarrow$ "\\Psi_S" $|$ *Psi1* $\rightarrow$ "\\Psi_P" $|$ *Psip* $\rightarrow$ "\\Psi^+"
$|$ *Psim* $\rightarrow$ "\\Psi^-" $|$ *Psipp* $\rightarrow$ "\\Psi^{++}" $|$ *Psimm* $\rightarrow$ "\\Psi^{--}"

let *flavor_symbol* = function
$|$ *L n* when $n > 0 \rightarrow$ "l" ^ *string_of_int n*
$|$ *L n* $\rightarrow$ "l" ^ *string_of_int* (*abs n*) ^ "b"
$|$ *Lodd n* when $n > 0 \rightarrow$ "lodd" ^ *string_of_int n*
$|$ *Lodd n* $\rightarrow$ "lodd" ^ *string_of_int* (*abs n*) ^ "b"
$|$ *N n* when $n > 0 \rightarrow$ "n" ^ *string_of_int n*
$|$ *N n* $\rightarrow$ "n" ^ *string_of_int* (*abs n*) ^ "b"
$|$ *Nodd n* when $n > 0 \rightarrow$ "nodd" ^ *string_of_int n*
$|$ *Nodd n* $\rightarrow$ "nodd" ^ *string_of_int* (*abs n*) ^ "b"
$|$ *U n* when $n > 0 \rightarrow$ "u" ^ *string_of_int n*
$|$ *U n* $\rightarrow$ "u" ^ *string_of_int* (*abs n*) ^ "b"
$|$ *D n* when $n > 0 \rightarrow$ "d" ^ *string_of_int n*
$|$ *D n* $\rightarrow$ "d" ^ *string_of_int* (*abs n*) ^ "b"
$|$ *Uodd n* when $n > 0 \rightarrow$ "uodd" ^ *string_of_int n*
$|$ *Uodd n* $\rightarrow$ "uodd" ^ *string_of_int* (*abs n*) ^ "b"
$|$ *Dodd n* when $n > 0 \rightarrow$ "dodd" ^ *string_of_int n*
$|$ *Dodd n* $\rightarrow$ "dodd" ^ *string_of_int* (*abs n*) ^ "b"
$|$ *Topp* $\rightarrow$ "tp" $|$ *Toppb* $\rightarrow$ "tpb"
$|$ *Gl* $\rightarrow$ "gl"
$|$ *Ga* $\rightarrow$ "a" $|$ *Z* $\rightarrow$ "z"
$|$ *Wp* $\rightarrow$ "wp" $|$ *Wm* $\rightarrow$ "wm"
$|$ *ZH* $\rightarrow$ "zh" $|$ *AH* $\rightarrow$ "ah" $|$ *WHp* $\rightarrow$ "whp" $|$ *WHm* $\rightarrow$ "whm"
$|$ *Phip* $\rightarrow$ "pp" $|$ *Phim* $\rightarrow$ "pm" $|$ *Phi0* $\rightarrow$ "p0"
$|$ *H* $\rightarrow$ "h" $|$ *Eta* $\rightarrow$ "eta"
$|$ *Psi0* $\rightarrow$ "psi0" $|$ *Psi1* $\rightarrow$ "psi1" $|$ *Psip* $\rightarrow$ "psip"
$|$ *Psim* $\rightarrow$ "psim" $|$ *Psipp* $\rightarrow$ "psipp" $|$ *Psimm* $\rightarrow$ "psimm"

There are PDG numbers for Z', Z", W', 32-34, respectively. We just introduce a number 38 for Y0 as a Z"'. As well, there is the number 8 for a t'. But we cheat a little bit and take the number 35 which is reserved for a heavy scalar Higgs for the Eta scalar. For the heavy Higgs states we take 35 and 36 for the neutral ones, 37 for the charged and 38 for the doubly-charged. The pseudoscalar gets the 39. For the odd fermions we add 40 to the values for the SM particles.

let *pdg* = function
$|$ *L n* when $n > 0 \rightarrow 9 + 2 \times n$
$|$ *L n* $\rightarrow - 9 + 2 \times n$
$|$ *N n* when $n > 0 \rightarrow 10 + 2 \times n$
$|$ *N n* $\rightarrow - 10 + 2 \times n$
$|$ *U n* when $n > 0 \rightarrow 2 \times n$
$|$ *U n* $\rightarrow 2 \times n$
$|$ *D n* when $n > 0 \rightarrow - 1 + 2 \times n$
$|$ *D n* $\rightarrow 1 + 2 \times n$

| $Lodd\ n$ when $n\ >\ 0\ \rightarrow\ 49\ +\ 2 \times n$
| $Lodd\ n\ \rightarrow\ -\ 49\ +\ 2 \times n$
| $Nodd\ n$ when $n\ >\ 0\ \rightarrow\ 50\ +\ 2 \times n$
| $Nodd\ n\ \rightarrow\ -\ 50\ +\ 2 \times n$
| $Uodd\ n$ when $n\ >\ 0\ \rightarrow\ 40\ +\ 2 \times n$
| $Uodd\ n\ \rightarrow\ -40\ +\ 2 \times n$
| $Dodd\ n$ when $n\ >\ 0\ \rightarrow\ 39\ +\ 2 \times n$
| $Dodd\ n\ \rightarrow\ -39\ +\ 2 \times n$
| $Topp\ \rightarrow\ 8$ | $Toppb\ \rightarrow\ (-8)$
| $Gl\ \rightarrow\ 21$
| $Ga\ \rightarrow\ 22$ | $Z\ \rightarrow\ 23$
| $Wp\ \rightarrow\ 24$ | $Wm\ \rightarrow\ (-24)$
| $AH\ \rightarrow\ 32$ | $ZH\ \rightarrow\ 33$ | $WHp\ \rightarrow\ 34$ | $WHm\ \rightarrow\ (-34)$
| $Phip$ | $Phim\ \rightarrow\ 27$ | $Phi0\ \rightarrow\ 26$
| $Psi0\ \rightarrow\ 35$ | $Psi1\ \rightarrow\ 36$ | $Psip\ \rightarrow\ 37$ | $Psim\ \rightarrow\ (-37)$
| $Psipp\ \rightarrow\ 38$ | $Psimm\ \rightarrow\ (-38)$
| $H\ \rightarrow\ 25$ | $Eta\ \rightarrow\ 39$

let $mass\_symbol\ f\ =$
"mass(" ^ $string\_of\_int\ (abs\ (pdg\ f))$ ^ ")"

let $width\_symbol\ f\ =$
"width(" ^ $string\_of\_int\ (abs\ (pdg\ f))$ ^ ")"

let $constant\_symbol\ =$ function
| $Unit\ \rightarrow$ "unit" | $Pi\ \rightarrow$ "PI" | $VHeavy\ \rightarrow$ "vheavy"
| $Alpha\_QED\ \rightarrow$ "alpha" | $E\ \rightarrow$ "e" | $G\_weak\ \rightarrow$ "g" | $Vev\ \rightarrow$ "vev"
| $Sin2thw\ \rightarrow$ "sin2thw" | $Sinthw\ \rightarrow$ "sinthw" | $Costhw\ \rightarrow$ "costhw"
| $Sinpsi\ \rightarrow$ "sinpsi" | $Cospsi\ \rightarrow$ "cospsi"
| $Atpsi\ \rightarrow$ "atpsi" | $Sccs\ \rightarrow$ "sccs"
| $Supp\ \rightarrow$ "vF" | $Supp2\ \rightarrow$ "v2F2"
| $Q\_lepton\ \rightarrow$ "qlep" | $Q\_up\ \rightarrow$ "qup" | $Q\_down\ \rightarrow$ "qdwn"
| $Q\_Z\_up\ \rightarrow$ "qzup"
| $G\_ZHTHT\ \rightarrow$ "gzhtht" | $G\_ZTHT\ \rightarrow$ "gzhtht"
| $G\_AHTHTH\ \rightarrow$ "gahthth" | $G\_AHTHT\ \rightarrow$ "gahtht" | $G\_AHTT\ \rightarrow$ "gahtt"
| $G\_NC\_lepton\ \rightarrow$ "gnclep" | $G\_NC\_neutrino\ \rightarrow$ "gncneu"
| $G\_NC\_up\ \rightarrow$ "gncup" | $G\_NC\_down\ \rightarrow$ "gncdwn"
| $G\_CC\ \rightarrow$ "gcc" | $G\_CCtop\ \rightarrow$ "gcctop" | $G\_CC\_heavy\ \rightarrow$ "gcch"
| $G\_CC\_WH\ \rightarrow$ "gccwh" | $G\_CC\_W\ \rightarrow$ "gccw"
| $G\_NC\_h\_lepton\ \rightarrow$ "gnchlep" | $G\_NC\_h\_neutrino\ \rightarrow$ "gnchneu"
| $G\_NC\_h\_up\ \rightarrow$ "gnchup" | $G\_NC\_h\_down\ \rightarrow$ "gnchdwn"
| $G\_NC\_heavy\ \rightarrow$ "gnch"
| $I\_Q\_W\ \rightarrow$ "iqw" | $I\_G\_ZWW\ \rightarrow$ "igzww" | $I\_G\_WWW\ \rightarrow$ "igwww"
| $I\_G\_AHWW\ \rightarrow$ "igahww" | $I\_G\_ZHWW\ \rightarrow$ "igzhww" | $I\_G\_ZWHW\ \rightarrow$ "igzwhw"
| $I\_G\_AHWHWH\ \rightarrow$ "igahwhwh" | $I\_G\_ZHWHWH\ \rightarrow$ "igzhwhwh"
| $I\_G\_AHWHW\ \rightarrow$ "igahwhw"

```
| I_Q_H → "iqh"
| Gs → "gs" | I_Gs → "igs" | G2 → "gs**2"
| G_WWWW → "gw4" | G_ZZWW → "gzzww"
| G_AZWW → "gazww" | G_AAWW → "gaaww"
| G_WH4 → "gwh4" | G_WHWHWW → "gwhwhww" | G_WHWWW → "gwhwww"
| G_WH3W → "gwh3w"
| G_WWAAH → "gwwaah" | G_WWAZH → "gwwazh" | G_WWZZH → "gwwzzh"
| G_WWZAH → "gwwzah" | G_WHWHAAH → "gwhwhaah"
| G_WHWHAZH → "gwhwhazh" | G_WHWHZZH → "gwhwhzzh"
| G_WHWHZAH → "gwhwhzah"
| G_WWZHAH → "gwwzhah" | G_WHWHZHAH → "gwhwhzhah"
| G_WHWZZ → "gwhwzz" | G_WHWAZ → "gwhwaz"
| G_WHWAAH → "gwhwaah" | G_WHWZAH → "gwhwzah"
| G_WHWZHZH → "gwhwzhzh" | G_WHWZHAH → "gwhwzhah"
| G_WHWAZH → "gwhwazh" | G_WHWZZH → "gwhwzzh"
| G_HWW → "ghww" | G_HZZ → "ghzz"
| G_HHWW → "ghhww" | G_HHZZ → "ghhzz"
| G_HWHW → "ghwhw" | G_HWHWH → "ghwhwh" | G_HAHAH → "ghahah"
| G_HZHZ → "ghzhz" | G_HZHAH → "ghzhah"
| G_HAHZ → "ghahz"
| G_Htt → "ghtt" | G_Hbb → "ghbb"
| G_Htautau → "ghtautau" | G_Hcc → "ghcc"
| G_Hthth → "ghthth" | G_Htht → "ghtht"
| G_HHtt → "ghhtt" | G_HHthth → "ghhthth" | G_HHtht → "ghhtht"
| G_Psi0tt → "gpsi0tt" | G_Psi0bb → "gpsi0bb"
| G_Psi0cc → "gpsi0cc" | G_Psi0tautau → "gpsi0tautau"
| G_Psi1tt → "gpsi1tt" | G_Psi1bb → "gpsi1bb"
| G_Psi1cc → "gpsi1cc" | G_Psi1tautau → "gpsi1tautau"
| G_Psipq3 → "gpsipq3" | G_Psipq2 → "gpsipq2" | G_Psipl3 → "gpsil3"
| G_Psi0tth → "gpsi0tth" | G_Psi1tth → "gpsi1tth"
| G_Psipbth → "gpsipbth"
| G_Ethth → "gethth" | G_Etht → "getht"
| G_Ett → "gett" | G_Ebb → "gebb"
| G_HGaGa → "ghgaga" | G_HGaZ → "ghgaz"
| G_EGaGa → "geaa" | G_EGaZ → "geaz" | G_EGlGl → "gegg"
| G_H3 → "gh3" | G_H4 → "gh4"
| G_PsiWW → "gpsiww" | G_PsiWHW → "gpsiwhw"
| G_PsiZZ → "gpsizz" | G_PsiZHZH → "gpsizhzh"
| G_PsiZHZ → "gpsizhz" | G_PsiZAH → "gpsizah"
| G_PsiZHAH → "gpsizhah" | G_PsiAHAH → "gpsiahah"
| G_PsiZW → "gpsizw" | G_PsiZWH → "gpsizwh" | G_PsiAHW → "gpsiahw"
| G_PsiAHWH → "gpsiahwh" | G_PsiZHW → "gpsizhw"
| G_PsiZHWH → "gpsizhwh"
| G_PsippWW → "gpsippww" | G_PsippWHW → "gpsippwhw"
```

| $G\_PsippWHWH$ → "gpsippwhwh"
| G_PsiHW → "gpsihw" | G_PsiHWH → "gpsihwh"
| G_Psi0W → "gpsi0w" | G_Psi0WH → "gpsi0wh"
| G_Psi1W → "gpsi1w" | G_Psi1WH → "gpsi1wh"
| G_PsiPPW → "gpsippw" | $G\_PsiPPWH$ → "gpsippwh"
| $G\_Psi1HAH$ → "gpsihah" | $G\_Psi01AH$ → "gpsi0ah"
| G_AHPsip → "gahpsip" | G_Psi1HZ → "gpsi1hz"
| $G\_Psi1HZH$ → "gpsi1hzh" | G_Psi01Z → "gpsi01z"
| $G\_Psi01ZH$ → "gpsi01zh" | G_ZPsip → "gzpsip"
| G_ZPsipp → "gzpsipp" | $G\_ZHPsipp$ → "gzhpsipp"
| G_HHAA → "ghhaa" | G_HHWHW → "ghhwhw" | G_HHZHZ → "ghhzhz"
| G_HHAHZ → "ghhahz" | G_HHZHAH → "ghhzhah"
| $G\_HPsi0WW$ → "ghpsi0ww" | $G\_HPsi0WHW$ → "ghpsi0whw"
| $G\_HPsi0ZZ$ → "ghpsi0zz" | $G\_HPsi0ZHZH$ → "ghpsi0zhzh"
| $G\_HPsi0ZHZ$ → "ghpsi0zhz" | $G\_HPsi0AHAH$ → "ghpsi0ahah"
| $G\_HPsi0ZAH$ → "ghpsi0zah" | $G\_HPsi0ZHAH$ → "ghpsi0zhah"
| $G\_HPsipWA$ → "ghpsipwa" | $G\_HPsipWHA$ → "ghpsipwha"
| $G\_HPsipWZ$ → "ghpsipwz" | $G\_HPsipWHZ$ → "ghpsiwhz"
| $G\_HPsipWAH$ → "ghpsipwah" | $G\_HPsipWHAH$ → "ghpsipwhah"
| $G\_HPsipWZH$ → "ghpsipwzh" | $G\_HPsipWHZH$ → "ghpsipwhzh"
| $G\_HPsippWW$ → "ghpsippww" | $G\_HPsippWHWH$ → "ghpsippwhwh"
| $G\_HPsippWHW$ → "ghpsippwhw" | $G\_Psi00ZH$ → "gpsi00zh"
| $G\_Psi00AH$ → "gpsi00ah" | $G\_Psi00ZHAH$ → "gpsi00zhah"
| $G\_Psi0pWA$ → "gpsi0pwa" | $G\_Psi0pWHA$ → "gpsi0pwha"
| $G\_Psi0pWZ$ → "gpsi0pwz" | $G\_Psi0pWHZ$ → "gpsi0pwhz"
| $G\_Psi0pWAH$ → "gpsi0pwah" | $G\_Psi0pWHAH$ → "gpsi0pwhah"
| $G\_Psi0pWZH$ → "gpsi0pwzh" | $G\_Psi0pWHZH$ → "gpsi0pwhzh"
| $G\_Psi0ppWW$ → "gpsi0ppww" | $G\_Psi0ppWHWH$ → "gpsi0ppwhwh"
| $G\_Psi0ppWHW$ → "gpsi0ppwhw"
| $I\_G\_Psi0pWA$ → "i_gpsi0pwa" | $I\_G\_Psi0pWHA$ → "i_gpsi0pwha"
| $I\_G\_Psi0pWZ$ → "i_gpsi0pwz" | $I\_G\_Psi0pWHZ$ → "i_gpsi0pwhz"
| $I\_G\_Psi0pWAH$ → "i_gpsi0pwah" | $I\_G\_Psi0pWHAH$ → "i_gpsi0pwhah"
| $I\_G\_Psi0pWZH$ → "i_gpsi0pwzh" | $I\_G\_Psi0pWHZH$ → "i_gpsi0pwhzh"
| $I\_G\_Psi0ppWW$ → "i_gpsi0ppww" | $I\_G\_Psi0ppWHWH$ → "i_gpsi0ppwhwh"
| $I\_G\_Psi0ppWHW$ → "i_gpsi0ppwhw"
| $G\_PsippZZ$ → "gpsippzz" | $G\_PsippZHZH$ → "gpsippzhzh"
| $G\_PsippAZ$ → "gpsippaz" | $G\_PsippAAH$ → "gpsippaah"
| $G\_PsippZAH$ → "gpsippzah"
| $G\_PsippWA$ → "gpsippwa" | $G\_PsippWHA$ → "gpsippwha"
| $G\_PsippWZ$ → "gpsippwz" | $G\_PsippWHZ$ → "gpsippwhz"
| $G\_PsippWAH$ → "gpsippwah" | $G\_PsippWHAH$ → "gpsippwhah"
| $G\_PsippWZH$ → "gpsippwzh" | $G\_PsippWHZH$ → "gpsippwhzh"
| $G\_PsiccZZ$ → "gpsicczz" | $G\_PsiccAZ$ → "gpsiccaz"
| $G\_PsiccAAH$ → "gpsiccaah" | $G\_PsiccZZH$ → "gpsicczzh"
| $G\_PsiccAZH$ → "gpsiccazh" | $G\_PsiccZAH$ → "gpsicczah"
| *Mass f* → "mass" ^ *flavor_symbol f*
| *Width f* → "width" ^ *flavor_symbol f*
end

```
module Simplest (Flags : BSM_flags) =
  struct
    let rcs = rcs_file

    open Coupling

    let default_width = ref Timelike
    let use_fudged_width = ref false

    let options = Options.create
      [ "constant_width", Arg.Unit (fun () → default_width := Constant),
        "use␣constant␣width␣(also␣in␣t-channel)";
        "fudged_width", Arg.Set use_fudged_width,
        "use␣fudge␣factor␣for␣charge␣particle␣width";
        "custom_width", Arg.String (fun f → default_width := Custom f),
        "use␣custom␣width";
        "cancel_widths", Arg.Unit (fun () → default_width := Vanishing),
        "use␣vanishing␣width" ]
```

We do not introduce the Goldstones for the heavy vectors here. The heavy quarks are simply numerated by their generation, the assignments whether they are up- or down-type will be defined by the model.

```
    type flavor = L of int | N of int | U of int | D of int | QH of int
        | NH of int | Wp | Wm | Ga | Z | Xp | Xm | X0 | Y0 | ZH
        | Phip | Phim | Phi0 | H | Eta | Gl

    type gauge = unit

    let gauge_symbol () =
      failwith "Modellib_BSM.Simplest.gauge_symbol:␣internal␣error"

    let family n = [ L n; N n; U n; D n; QH n; NH n ]
```

Note that we add all heavy quarks, U, D, C, S, in order to have both embeddings included.

```
    let external_flavors () =
      [ "1st␣Generation␣(incl.␣heavy)", ThoList.flatmap family [1; −1];
        "2nd␣Generation␣(incl.␣heavy)", ThoList.flatmap family [2; −2];
        "3rd␣Generation␣(incl.␣heavy)", ThoList.flatmap family [3; −3];
        "Gauge␣Bosons", [Ga; Z; Wp; Wm; Gl; Xp; Xm; X0; Y0; ZH];
        "Higgs", [H; Eta];
        "Goldstone␣Bosons", [Phip; Phim; Phi0] ]

    let flavors () = ThoList.flatmap snd (external_flavors ())

    let spinor n =
      if n ≥ 0 then
        Spinor
      else
        ConjSpinor

    let lorentz = function
      | L n → spinor n | N n → spinor n
      | U n → spinor n | D n → spinor n
      | QH n → spinor n | NH n → spinor n
```

```
  | Ga | Gl → Vector
  | Wp | Wm | Z | Xp | Xm | X0 | Y0 | ZH → Massive_Vector
  | _ → Scalar

let color = function
  | U n → Color.SUN (if n > 0 then 3 else -3)
  | D n → Color.SUN (if n > 0 then 3 else -3)
  | QH n → Color.SUN (if n > 0 then 3 else -3)
  | Gl → Color.AdjSUN 3
  | _ → Color.Singlet

let prop_spinor n =
  if n ≥ 0 then
    Prop_Spinor
  else
    Prop_ConjSpinor

let propagator = function
  | L n → prop_spinor n | N n → prop_spinor n
  | U n → prop_spinor n | D n → prop_spinor n
  | QH n → prop_spinor n | NH n → prop_spinor n
  | Ga | Gl → Prop_Feynman
  | Wp | Wm | Z | Xp | Xm | X0 | Y0 | ZH → Prop_Unitarity
  | Phip | Phim | Phi0 → Only_Insertion
  | H | Eta → Prop_Scalar
```

Optionally, ask for the fudge factor treatment for the widths of charged particles. Currently, this only applies to $W^{\pm}$ and top.

```
let width f =
  if !use_fudged_width then
    match f with
    | Wp | Wm | U 3 | U (−3) | QH _ | NH _ → Fudged
    | _ → !default_width
  else
    !default_width

let goldstone = function
  | Wp → Some (Phip, Coupling.Const 1)
  | Wm → Some (Phim, Coupling.Const 1)
  | Z → Some (Phi0, Coupling.Const 1)
  | _ → None

let conjugate = function
  | L n → L (−n) | N n → N (−n)
  | U n → U (−n) | D n → D (−n)
  | QH n → QH (−n) | NH n → NH (−n)
  | Ga → Ga | Gl → Gl | Z → Z
  | Wp → Wm | Wm → Wp
  | Xp → Xm | Xm → Xp | X0 → X0 | Y0 → Y0 | ZH → ZH
  | Phip → Phim | Phim → Phip | Phi0 → Phi0
  | H → H | Eta → Eta

let fermion = function
```

```
  | L n → if n > 0 then 1 else -1
  | N n → if n > 0 then 1 else -1
  | U n → if n > 0 then 1 else -1
  | D n → if n > 0 then 1 else -1
  | QH n → if n > 0 then 1 else -1
  | NH n → if n > 0 then 1 else -1
  | Ga | Gl | Z | Wp | Wm | Xp | Xm | X0 | Y0 | ZH → 0
  | _ → 0

module Ch = Charges.QQ
let ( // ) = Algebra.Small_Rational.make

let charge = function
   | L n → if n > 0 then -1//1 else 1//1
   | N n | NH n → 0//1
   | U n → if n > 0 then 2//3 else -2//3
   | QH 3 → 2//3 | QH (−3) → − 2//3
   | QH (1 | 2) →
       if Flags.anom_ferm_ass then
          2//3
       else
          -1//3
   | QH ((−1) | (−2)) →
       if Flags.anom_ferm_ass then
          -2//3
       else
          1//3
   | QH n → invalid_arg ("Simplest.charge: QH " ^ string_of_int n)
   | D n → if n > 0 then -1//3 else 1//3
   | Gl | Ga | Z | ZH | X0 | Y0 → 0//1
   | Wp | Xp → 1//1
   | Wm | Xm → − 1//1
   | H | Phi0 | Eta → 0//1
   | Phip → 1//1
   | Phim → − 1//1

let lepton = function
   | L n | N n | NH n
       → if n > 0 then 1//1 else -1//1
   | U _ | D _ | _ → 0//1

let baryon = function
   | L _ | N _ → 0//1
   | U n | D n | QH n
       → if n > 0 then 1//1 else -1//1
   | _ → 0//1

let charges f =
  [ charge f; lepton f; baryon f]

type constant =
  | Unit | Pi | Alpha_QED | Sin2thw
  | Sinthw | Costhw | E | G_weak | Vev | VHeavy
```

```
  | Supp | Supp2
  | Sinpsi | Cospsi | Atpsi | Sccs (* Mixing angles of SU(2) *)
  | Q_lepton | Q_up | Q_down | Q_Z_up | G_CC | I_G_CC
  | G_NC_neutrino | G_NC_lepton | G_NC_up | G_NC_down
  | G_NC_X | G_NC_X_t | G_NC_Y | G_NC_Y_t | G_NC_H
  | G_NC_h_neutrino | G_NC_h_lepton | G_NC_h_up | G_NC_h_down
  | G_NC_h_top | G_NC_h_bot | G_NCH_N | G_NCH_U |
G_NCH_D | G_NCHt
  | G_zhthth
  | I_Q_W | I_G_ZWW | I_G_WWW
  | I_G_Z1 | I_G_Z2 | I_G_Z3 | I_G_Z4 | I_G_Z5 | I_G_Z6
  | I_Q_H | Gs | I_Gs | G2
  | G_WWWW | G_ZZWW | G_AZWW | G_AAWW
  | I_Q_ZH
  | G_HWW | G_HHWW | G_HZZ | G_HHZZ | G_HHZZH
  | G_heavy_HVV | G_heavy_HWW | G_heavy_HZZ | G_HHthth
  | G_Htt | G_Hbb | G_Hcc | G_Htautau | G_H3 | G_H4
  | G_Hthth | G_Htht | G_Ethth | G_Etht | G_Ett | G_Hqhq
  | G_Ebb | G_ZEH | G_ZHEH | G_Hgg
  | G_HGaGa | G_HGaZ | G_EGaGa | G_EGaZ | G_EGlGl
  | Mass of flavor | Width of flavor
```

Two integer counters for the QCD and EW order of the couplings.

```
type orders = int × int

let orders = function
  | _ → (0, 0)
```

The current abstract syntax for parameter dependencies is admittedly tedious. Later, there will be a parser for a convenient concrete syntax as a part of a concrete syntax for models. But as these examples show, it should include simple functions.

```
let input_parameters =
  []

let derived_parameters =
  []

let g_over_2_costh =
  Quot (Neg (Atom G_weak), Prod [Const 2; Atom Costhw])

let nc_coupling c t3 q =
  (Real_Array c,
   [Prod [g_over_2_costh; Diff (t3, Prod [Const 2; q; Atom Sin2thw])];
    Prod [g_over_2_costh; t3]])

let half = Quot (Const 1, Const 2)

let derived_parameter_arrays =
  [ nc_coupling G_NC_neutrino half (Const 0);
    nc_coupling G_NC_lepton (Neg half) (Const (−1));
    nc_coupling G_NC_up half (Quot (Const 2, Const 3));
```

```
      nc_coupling G_NC_down (Neg half) (Quot (Const (-1), Const 3));
      nc_coupling G_NC_h_neutrino half (Const 0);
      nc_coupling G_NC_h_lepton (Neg half) (Const (-1));
      nc_coupling G_NC_h_up half (Quot (Const 2, Const 3));
      nc_coupling G_NC_h_down (Neg half) (Quot (Const (-1), Const 3)) ]

let parameters () =
  { input = input_parameters;
    derived = derived_parameters;
    derived_arrays = derived_parameter_arrays }

module F = Modeltools.Fusions (struct
  type f = flavor
  type c = constant
  let compare = compare
  let conjugate = conjugate
end)

let electromagnetic_currents n =
    [ ((L (-n), Ga, L n), FBF (1, Psibar, V, Psi), Q_lepton);
      ((U (-n), Ga, U n), FBF (1, Psibar, V, Psi), Q_up);
      ((D (-n), Ga, D n), FBF (1, Psibar, V, Psi), Q_down) ]

let color_currents n =
  [ ((D (-n), Gl, D n), FBF ((-1), Psibar, V, Psi), Gs);
    ((U (-n), Gl, U n), FBF ((-1), Psibar, V, Psi), Gs);
    ((QH (-n), Gl, QH n), FBF ((-1), Psibar, V, Psi), Gs)]

let neutral_currents n =
    [ ((L (-n), Z, L n), FBF (1, Psibar, VA, Psi), G_NC_lepton);
      ((N (-n), Z, N n), FBF (1, Psibar, VA, Psi), G_NC_neutrino);
      ((U (-n), Z, U n), FBF (1, Psibar, VA, Psi), G_NC_up);
      ((D (-n), Z, D n), FBF (1, Psibar, VA, Psi), G_NC_down) ]

let xy_currents =
  ThoList.flatmap
    (fun n → [ ((N (-n), X0, N n), FBF ((-1), Psibar, VL, Psi), G_NC_X);
               ((L (-n), Xm, N n), FBF ((-1), Psibar, VL, Psi), G_NC_X);
               ((N (-n), Xp, L n), FBF ((-1), Psibar, VL, Psi), G_NC_X);
               ((N (-n), Y0, N n), FBF ((-1), Psibar, VL, Psi), G_NC_Y);
               ((NH (-n), X0, N n), FBF ((-1), Psibar, VL, Psi), G_CC);
               ((N (-n), X0, NH n), FBF ((-1), Psibar, VL, Psi), G_CC);
               ((NH (-n), Y0, N n), FBF ((-1), Psibar, VL, Psi), I_G_CC);
               ((N (-n), Y0, NH n), FBF ((-1), Psibar, VL, Psi), I_G_CC);
               ((L (-n), Xm, NH n), FBF ((-1), Psibar, VL, Psi), G_CC);
               ((NH (-n), Xp, L n), FBF ((-1), Psibar, VL, Psi), G_CC)])
    [1; 2; 3]
  @
    [ ((U (-3), X0, U 3), FBF (1, Psibar, VL, Psi), G_NC_X_t);
      ((U (-3), Y0, U 3), FBF (1, Psibar, VL, Psi), G_NC_Y_t);
      ((U (-3), X0, QH 3), FBF (1, Psibar, VL, Psi), G_CC);
      ((QH (-3), X0, U 3), FBF (1, Psibar, VL, Psi), G_CC);
      ((U (-3), Y0, QH 3), FBF (1, Psibar, VL, Psi), I_G_CC);
```

```
((QH (−3), Y0, U 3), FBF (1, Psibar, VL, Psi), I_G_CC);
((D (−3), Xm, U 3), FBF (1, Psibar, VL, Psi), G_NC_X_t);
((U (−3), Xp, D 3), FBF (1, Psibar, VL, Psi), G_NC_X_t);
((D (−3), Xm, QH 3), FBF (1, Psibar, VL, Psi), G_CC);
((QH (−3), Xp, D 3), FBF (1, Psibar, VL, Psi), G_CC);
((QH (−3), Wp, D 3), FBF (1, Psibar, VL, Psi), G_NC_X_t);
((D (−3), Wm, QH 3), FBF (1, Psibar, VL, Psi), G_NC_X_t);
((QH (−3), Z, U 3), FBF (1, Psibar, VL, Psi), G_NCHt);
((U (−3), Z, QH 3), FBF (1, Psibar, VL, Psi), G_NCHt)]
@
  ThoList.flatmap
    (fun n →
      if Flags.anom_ferm_ass then
        [ ((U (−n), X0, U n), FBF ((−1), Psibar, VL, Psi), G_NC_X);
          ((U (−n), Y0, U n), FBF ((−1), Psibar, VL, Psi), G_NC_Y);
          ((D (−n), Xm, U n), FBF ((−1), Psibar, VL, Psi), G_NC_X);
          ((U (−n), Xp, D n), FBF ((−1), Psibar, VL, Psi), G_NC_X);
          ((QH (−n), X0, U n), FBF ((−1), Psibar, VL, Psi), G_CC);
          ((U (−n), X0, QH n), FBF ((−1), Psibar, VL, Psi), G_CC);
          ((QH (−n), Y0, U n), FBF ((−1), Psibar, VL, Psi), I_G_CC);
          ((U (−n), Y0, QH n), FBF ((−1), Psibar, VL, Psi), I_G_CC);
          ((D (−n), Xm, QH n), FBF ((−1), Psibar, VL, Psi), G_CC);
          ((QH (−n), Xp, D n), FBF ((−1), Psibar, VL, Psi), G_CC);
          ((QH (−n), Wp, D n), FBF ((−1), Psibar, VL, Psi), G_NC_X);
          ((D (−n), Wm, QH n), FBF ((−1), Psibar, VL, Psi), G_NC_X);
          ((QH (−n), Z, U n), FBF (1, Psibar, VL, Psi), G_NC_H);
          ((U (−n), Z, QH n), FBF (1, Psibar, VL, Psi), G_NC_H)]
      else
        [ ((D (−n), X0, D n), FBF (1, Psibar, VL, Psi), G_NC_X);
          ((D (−n), Y0, D n), FBF (1, Psibar, VL, Psi), G_NC_Y);
          ((D (−n), Xm, U n), FBF (1, Psibar, VL, Psi), G_NC_X);
          ((U (−n), Xp, D n), FBF (1, Psibar, VL, Psi), G_NC_X);
          ((QH (−n), X0, D n), FBF ((−1), Psibar, VL, Psi), G_CC);
          ((D (−n), X0, QH n), FBF ((−1), Psibar, VL, Psi), G_CC);
          ((QH (−n), Y0, D n), FBF ((−1), Psibar, VL, Psi), I_G_CC);
          ((D (−n), Y0, QH n), FBF ((−1), Psibar, VL, Psi), I_G_CC);
          ((QH (−n), Xm, U n), FBF (1, Psibar, VL, Psi), G_CC);
          ((U (−n), Xp, QH n), FBF (1, Psibar, VL, Psi), G_CC);
          ((QH (−n), Wm, U n), FBF (1, Psibar, VL, Psi), G_NC_X);
          ((U (−n), Wp, QH n), FBF (1, Psibar, VL, Psi), G_NC_X);
          ((QH (−n), Z, D n), FBF (1, Psibar, VL, Psi), G_NC_H);
          ((D (−n), Z, QH n), FBF (1, Psibar, VL, Psi), G_NC_H)])
    [1; 2]
```

The sign of this coupling is just the one of the T3, being -(1/2) for leptons and down quarks, and +(1/2) for neutrinos and up quarks.

```
let neutral_heavy_currents n =
  [ ((L (−n), ZH, L n), FBF (1, Psibar, VLR, Psi), G_NC_h_lepton);
    ((N (−n), ZH, N n), FBF ((−1), Psibar, VLR, Psi), G_NC_h_neutrino);
    ((U (−n), ZH, U n), FBF ((−1), Psibar, VLR, Psi), (if n = 3 then
```

```
                              G_NC_h_top else G_NC_h_up));
    ((D (−n), ZH, D n), FBF (1, Psibar, VLR, Psi), (if n = 3 then
                              G_NC_h_bot else G_NC_h_down));
    ((NH (−n), ZH, NH n), FBF (1, Psibar, VLR, Psi), G_NCH_N);
    ((QH (−n), ZH, QH n), FBF (1, Psibar, VLR, Psi), (if n = 3 then
       G_NCH_U else if Flags.anom_ferm_ass then G_NCH_U else G_NCH_D))]

let heavy_currents n =
  [ ((QH (−n), Ga, QH n), FBF (1, Psibar, V, Psi), (if n = 3 then Q_up else
        if Flags.anom_ferm_ass then Q_up else Q_down))]

let charged_currents n =
  [ ((L (−n), Wm, N n), FBF (1, Psibar, VL, Psi), G_CC);
    ((N (−n), Wp, L n), FBF (1, Psibar, VL, Psi), G_CC);
    ((D (−n), Wm, U n), FBF (1, Psibar, VL, Psi), G_CC);
    ((U (−n), Wp, D n), FBF (1, Psibar, VL, Psi), G_CC) ]

let yukawa =
  [ ((U (−3), H, U 3), FBF (1, Psibar, S, Psi), G_Htt);
    ((D (−3), H, D 3), FBF (1, Psibar, S, Psi), G_Hbb);
    ((U (−2), H, U 2), FBF (1, Psibar, S, Psi), G_Hcc);
    ((L (−3), H, L 3), FBF (1, Psibar, S, Psi), G_Htautau) ]

let yukawa_add =
  [ ((QH (−3), H, U 3), FBF (1, Psibar, SL, Psi), G_Htht);
    ((U (−3), H, QH 3), FBF (1, Psibar, SR, Psi), G_Htht);
    ((QH (−3), Eta, U 3), FBF (1, Psibar, SR, Psi), G_Etht);
    ((U (−3), Eta, QH 3), FBF (1, Psibar, SL, Psi), G_Etht);
    ((D (−3), Eta, D 3), FBF (1, Psibar, P, Psi), G_Ebb);
    ((U (−3), Eta, U 3), FBF (1, Psibar, P, Psi), G_Ett)]
    @
      ThoList.flatmap
        (fun n →
      if Flags.anom_ferm_ass then
  [ ((QH (−n), H, U n), FBF (1, Psibar, SL, Psi), G_Hqhq);
    ((U (−n), H, QH n), FBF (1, Psibar, SR, Psi), G_Hqhq)]
      else
  [ ((QH (−n), H, D n), FBF (1, Psibar, SL, Psi), G_Hqhq);
    ((D (−n), H, QH n), FBF (1, Psibar, SR, Psi), G_Hqhq)])
        [1; 2]

let standard_triple_gauge =
  [ ((Ga, Wm, Wp), Gauge_Gauge_Gauge 1, I_Q_W);
    ((Z, Wm, Wp), Gauge_Gauge_Gauge 1, I_G_ZWW);
    ((Gl, Gl, Gl), Gauge_Gauge_Gauge 1, I_Gs)]

let heavy_triple_gauge =
  [ ((Ga, Xm, Xp), Gauge_Gauge_Gauge 1, I_Q_W);
    ((Z, Xm, Xp), Gauge_Gauge_Gauge 1, I_Q_ZH);
    ((Z, X0, Y0), Gauge_Gauge_Gauge 1, I_G_Z1);
    ((ZH, X0, Y0), Gauge_Gauge_Gauge 1, I_G_Z2);
    ((Y0, Wm, Xp), Gauge_Gauge_Gauge 1, I_G_Z3);
    ((Y0, Wp, Xm), Gauge_Gauge_Gauge (−1), I_G_Z3);
```

```
      ((X0, Wm, Xp), Gauge_Gauge_Gauge 1, I_G_Z4);
      ((X0, Wp, Xm), Gauge_Gauge_Gauge 1, I_G_Z4);
      ((ZH, Xm, Xp), Gauge_Gauge_Gauge 1, I_G_Z5);
      ((ZH, Wm, Wp), Gauge_Gauge_Gauge 1, I_G_Z6)]

  let triple_gauge =
    standard_triple_gauge @ heavy_triple_gauge

  let gauge4 = Vector4 [(2, C_13_42); (−1, C_12_34); (−1, C_14_23)]
  let minus_gauge4 = Vector4 [(−2, C_13_42); (1, C_12_34); (1, C_14_23)]
  let standard_quartic_gauge =
    [ (Wm, Wp, Wm, Wp), gauge4, G_WWWW;
      (Wm, Z, Wp, Z), minus_gauge4, G_ZZWW;
      (Wm, Z, Wp, Ga), minus_gauge4, G_AZWW;
      (Wm, Ga, Wp, Ga), minus_gauge4, G_AAWW;
      (Gl, Gl, Gl, Gl), gauge4, G2]

  let heavy_quartic_gauge =
    []

  let quartic_gauge =
    standard_quartic_gauge @ heavy_quartic_gauge

  let standard_gauge_higgs' =
    [ ((H, Wp, Wm), Scalar_Vector_Vector 1, G_HWW);
      ((H, Z, Z), Scalar_Vector_Vector 1, G_HZZ) ]

  let heavy_gauge_higgs =
    [ ((H, Wp, Xm), Scalar_Vector_Vector 1, G_heavy_HWW);
      ((H, Wm, Xp), Scalar_Vector_Vector 1, G_heavy_HWW);
      ((H, Z, X0), Scalar_Vector_Vector 1, G_heavy_HVV);
      ((H, ZH, X0), Scalar_Vector_Vector 1, G_heavy_HVV)]

  let standard_gauge_higgs =
    standard_gauge_higgs' @ heavy_gauge_higgs

  let standard_gauge_higgs4 =
    [ (H, H, Wp, Wm), Scalar2_Vector2 1, G_HHWW;
      (H, H, Z, Z), Scalar2_Vector2 1, G_HHZZ ]

  let heavy_gauge_higgs4 =
    [ (H, H, Z, ZH), Scalar2_Vector2 1, G_HHZZH;
      (H, H, Xp, Xm), Scalar2_Vector2 (−1), G_HHWW;
      (H, H, ZH, ZH), Scalar2_Vector2 (−1), G_HHZZ ]

  let standard_higgs =
    [ (H, H, H), Scalar_Scalar_Scalar 1, G_H3 ]

 let anomaly_higgs =
    [ (Eta, Gl, Gl), Dim5_Scalar_Gauge2_Skew 1, G_EGlGl;
      (Eta, Ga, Ga), Dim5_Scalar_Gauge2_Skew 1, G_EGaGa;
      (Eta, Ga, Z), Dim5_Scalar_Gauge2_Skew 1, G_EGaZ ]
(* @ (H, Ga, Ga), Dim5_Scalar_Gauge2 1, G_HGaGa; (H, Ga, Z), Dim5_Scalar_Gauge2 1, G_HGaZ
*)

  let standard_higgs4 =
```

```
  [ (H, H, H, H), Scalar4 1, G_H4 ]

let gauge_higgs =
    standard_gauge_higgs

let gauge_higgs4 =
    standard_gauge_higgs4 @ heavy_gauge_higgs4

let higgs =
    standard_higgs

let eta_higgs_gauge =
  [ (Z, Eta, H), Vector_Scalar_Scalar 1, G_ZEH;
    (ZH, Eta, H), Vector_Scalar_Scalar 1, G_ZHEH;
    (X0, Eta, H), Vector_Scalar_Scalar 1, G_CC ]

let top_quartic =
  [ ((QH (-3), H, H, QH 3), GBBG (1, Psibar, S2, Psi), G_HHthth)]

let higgs4 =
    standard_higgs4

let goldstone_vertices =
  [ ((Phi0, Wm, Wp), Scalar_Vector_Vector 1, I_G_ZWW);
    ((Phip, Ga, Wm), Scalar_Vector_Vector 1, I_Q_W);
    ((Phip, Z, Wm), Scalar_Vector_Vector 1, I_G_ZWW);
    ((Phim, Wp, Ga), Scalar_Vector_Vector 1, I_Q_W);
    ((Phim, Wp, Z), Scalar_Vector_Vector 1, I_G_ZWW) ]

let vertices3 =
  (ThoList.flatmap electromagnetic_currents [1; 2; 3] @
   ThoList.flatmap color_currents [1; 2; 3] @
   ThoList.flatmap neutral_currents [1; 2; 3] @
   ThoList.flatmap neutral_heavy_currents [1; 2; 3] @
   ThoList.flatmap heavy_currents [1; 2; 3] @
   ThoList.flatmap charged_currents [1; 2; 3] @
   xy_currents @ anomaly_higgs @
   eta_higgs_gauge @
   yukawa @ yukawa_add @
   triple_gauge @
   gauge_higgs @ higgs @ goldstone_vertices)

let vertices4 =
  quartic_gauge @ gauge_higgs4 @ higgs4

let vertices () = (vertices3, vertices4, [])
```

For efficiency, make sure that *F.of_vertices vertices* is evaluated only once.

```
let table = F.of_vertices (vertices ())
let fuse2 = F.fuse2 table
let fuse3 = F.fuse3 table
let fuse = F.fuse table
let max_degree () = 4

let flavor_of_string = function
  | "e-" → L 1 | "e+" → L (-1)
```

```
| "mu-" → L 2 | "mu+" → L (−2)
| "tau-" → L 3 | "tau+" → L (−3)
| "nue" → N 1 | "nuebar" → N (−1)
| "numu" → N 2 | "numubar" → N (−2)
| "nutau" → N 3 | "nutaubar" → N (−3)
| "nh1" → NH 1 | "nh1bar" → NH (−1)
| "nh2" → NH 2 | "nh2bar" → NH (−2)
| "nh3" → NH 3 | "nh3bar" → NH (−3)
| "u" → U 1 | "ubar" → U (−1)
| "c" → U 2 | "cbar" → U (−2)
| "t" → U 3 | "tbar" → U (−3)
| "d" → D 1 | "dbar" → D (−1)
| "s" → D 2 | "sbar" → D (−2)
| "b" → D 3 | "bbar" → D (−3)
| "uh" → if Flags.anom_ferm_ass then QH 1 else invalid_arg
    "Modellib_BSM.Simplest.flavor_of_string"
| "dh" → if Flags.anom_ferm_ass then invalid_arg
      "Modellib_BSM.Simplest.flavor_of_string" else QH 1
| "uhbar" → if Flags.anom_ferm_ass then QH (−1) else invalid_arg
    "Modellib_BSM.Simplest.flavor_of_string"
| "dhbar" → if Flags.anom_ferm_ass then invalid_arg
      "Modellib_BSM.Simplest.flavor_of_string" else QH (−1)
| "ch" → if Flags.anom_ferm_ass then QH 2 else invalid_arg
    "Modellib_BSM.Simplest.flavor_of_string"
| "sh" → if Flags.anom_ferm_ass then invalid_arg
      "Modellib_BSM.Simplest.flavor_of_string" else QH 2
| "chbar" → if Flags.anom_ferm_ass then QH (−2) else invalid_arg
    "Modellib_BSM.Simplest.flavor_of_string"
| "shbar" → if Flags.anom_ferm_ass then invalid_arg
      "Modellib_BSM.Simplest.flavor_of_string" else QH (−2)
| "th" → QH 3 | "thbar" → QH (−3)
| "eta" | "Eta" → Eta
| "A" → Ga | "Z" | "Z0" → Z | "g" | "gl" → Gl
| "ZH" | "ZH0" | "Zh" | "Zh0" → ZH
| "W+" → Wp | "W-" → Wm
| "X+" → Xp | "X-" → Xm
| "X0" → X0 | "Y0" → Y0
| "H" → H
| _ → invalid_arg "Modellib_BSM.Simplest.flavor_of_string"

let flavor_to_string = function
  | L 1 → "e-" | L (−1) → "e+"
  | L 2 → "mu-" | L (−2) → "mu+"
  | L 3 → "tau-" | L (−3) → "tau+"
  | L _ → invalid_arg
      "Modellib_BSM.Simplest.flavor_to_string:␣invalid␣lepton"
  | N 1 → "nue" | N (−1) → "nuebar"
  | N 2 → "numu" | N (−2) → "numubar"
  | N 3 → "nutau" | N (−3) → "nutaubar"
  | N _ → invalid_arg
```

```
        "Modellib_BSM.Simplest.flavor_to_string:␣invalid␣neutrino"
  | U 1 → "u" | U (−1) → "ubar"
  | U 2 → "c" | U (−2) → "cbar"
  | U 3 → "t" | U (−3) → "tbar"
  | U _ → invalid_arg
        "Modellib_BSM.Simplest.flavor_to_string:␣invalid␣up␣type␣quark"
  | D 1 → "d" | D (−1) → "dbar"
  | D 2 → "s" | D (−2) → "sbar"
  | D 3 → "b" | D (−3) → "bbar"
  | D _ → invalid_arg
        "Modellib_BSM.Simplest.flavor_to_string:␣invalid␣down␣type␣quark"
  | QH 1 → if Flags.anom_ferm_ass then "uh" else "dh"
  | QH 2 → if Flags.anom_ferm_ass then "ch" else "sh"
  | QH 3 → "th"
  | QH (−1) → if Flags.anom_ferm_ass then "uhbar" else "dhbar"
  | QH (−2) → if Flags.anom_ferm_ass then "chbar" else "shbar"
  | QH (−3) → "thbar"
  | QH _ → invalid_arg
        "Modellib_BSM.Simplest.flavor_to_string:␣invalid␣heavy␣quark"
  | NH n when n > 0 → "nh" ^ string_of_int n
  | NH n → "nh" ^ string_of_int (abs n) ^ "bar"
  | Ga → "A" | Z → "Z" | Gl → "gl"
  | Wp → "W+" | Wm → "W-"
  | Xp → "X+" | Xm → "X-" | X0 → "X0" | Y0 → "Y0" | ZH →
"ZH"
  | Phip → "phi+" | Phim → "phi-" | Phi0 → "phi0"
  | H → "H" | Eta → "Eta"

let flavor_to_TeX = function
  | L 1 → "e^-" | L (−1) → "\\e^+"
  | L 2 → "\\mu^-" | L (−2) → "\\mu^+"
  | L 3 → "\\tau^-" | L (−3) → "\\tau^+"
  | L _ → invalid_arg
        "Modellib_BSM.Simplest.flavor_to_TeX:␣invalid␣lepton"
  | N 1 → "\\nu_e" | N (−1) → "\\bar{\\nu}_e"
  | N 2 → "\\nu_\\mu" | N (−2) → "\\bar{\\nu}_\\mu"
  | N 3 → "\\nu_\\tau" | N (−3) → "\\bar{\\nu}_\\tau"
  | N _ → invalid_arg
        "Modellib_BSM.Simplest.flavor_to_TeX:␣invalid␣neutrino"
  | U 1 → "u" | U (−1) → "\\bar{u}"
  | U 2 → "c" | U (−2) → "\\bar{c}"
  | U 3 → "t" | U (−3) → "\\bar{t}"
  | U _ → invalid_arg
        "Modellib_BSM.Simplest.flavor_to_TeX:␣invalid␣up␣type␣quark"
  | D 1 → "d" | D (−1) → "\\bar{d}"
  | D 2 → "s" | D (−2) → "\\bar{s}"
  | D 3 → "b" | D (−3) → "\\bar{b}"
  | D _ → invalid_arg
        "Modellib_BSM.Simplest.flavor_to_TeX:␣invalid␣down␣type␣quark"
  | QH 1 → if Flags.anom_ferm_ass then "U" else "D"
```

```
    | QH 2 → if Flags.anom_ferm_ass then "C" else "S"
    | QH 3 → "T"
    | QH (−1) → if Flags.anom_ferm_ass then "\\bar{U}" else "\\bar{D}"
    | QH (−2) → if Flags.anom_ferm_ass then "\\bar{C}" else "\\bar{S}"
    | QH (−3) → "thbar"
    | QH _ → invalid_arg
        "Modellib_BSM.Simplest.flavor_to_TeX: invalid heavy quark"
    | NH n when n > 0 → "N_" ^ string_of_int n
    | NH n → "\\bar{N}_" ^ string_of_int (abs n)
    | Ga → "\\gamma" | Z → "Z" | Gl → "g"
    | Wp → "W^+" | Wm → "W^-"
    | Xp → "X^+" | Xm → "X^-" | X0 → "X^0" | Y0 → "Y^0" |
ZH → "Z_H"
    | Phip → "\\phi^+" | Phim → "\\phi^-" | Phi0 → "\\phi^0"
    | H → "H" | Eta → "\\eta"

  let flavor_symbol = function
    | L n when n > 0 → "l" ^ string_of_int n
    | L n → "l" ^ string_of_int (abs n) ^ "b"
    | N n when n > 0 → "n" ^ string_of_int n
    | N n → "n" ^ string_of_int (abs n) ^ "b"
    | U n when n > 0 → "u" ^ string_of_int n
    | U n → "u" ^ string_of_int (abs n) ^ "b"
    | D n when n > 0 → "d" ^ string_of_int n
    | D n → "d" ^ string_of_int (abs n) ^ "b"
    | NH n when n > 0 → "nh" ^ string_of_int n
    | NH n → "nh" ^ string_of_int (abs n) ^ "b"
    | QH n when n > 0 → "qh" ^ string_of_int n
    | QH n → "qh" ^ string_of_int (abs n) ^ "b"
    | Ga → "a" | Z → "z" | Gl → "gl"
    | Wp → "wp" | Wm → "wm"
    | Xp → "xp" | Xm → "xm" | X0 → "x0" | Y0 → "y0" | ZH →
"zh"
    | Phip → "pp" | Phim → "pm" | Phi0 → "p0"
    | H → "h" | Eta → "eta"
```

There are PDG numbers for Z', Z", W', 32-34, respectively. We just introduce a number 38 for Y0 as a Z"'. As well, there is the number 8 for a t'. But we cheat a little bit and take the number 35 which is reserved for a heavy scalar Higgs for the Eta scalar.

We abuse notation for the heavy quarks and take the PDG code for their SUSY partners!!! (What about an update of the PDG numbering scheme?) Thereby we take only those for up-type (s)quarks. The heavy neutrinos get the numbers of the sneutrinos.

```
  let pdg = function
    | L n when n > 0 → 9 + 2 × n
    | L n → − 9 + 2 × n
    | N n when n > 0 → 10 + 2 × n
    | N n → − 10 + 2 × n
    | U n when n > 0 → 2 × n
    | U n → 2 × n
```

| $D\ n$ when $n\ >\ 0\ \rightarrow\ -\ 1\ +\ 2 \times n$
| $D\ n\ \rightarrow\ 1\ +\ 2 \times n$
| $NH\ n$ when $n\ >\ 0\ \rightarrow\ 1000010\ +\ 2 \times n$
| $NH\ n\ \rightarrow\ -\ 1000010\ +\ 2 \times n$
| $QH\ 3\ \rightarrow\ 1000006$
| $QH\ (-3)\ \rightarrow\ -\ 1000006$
| $QH\ n$ when $n\ >\ 0\ \rightarrow$ if $Flags.anom\_ferm\_ass$ then
$1000000 + 2{\times}n$ else $999999 + 2{\times}n$
| $QH\ n\ \rightarrow$ if $Flags.anom\_ferm\_ass$ then
- $1000000 + 2{\times}n$ else - $999999 + 2{\times}n$
| $Gl\ \rightarrow\ 21$
| $Ga\ \rightarrow\ 22$ | $Z\ \rightarrow\ 23$
| $Wp\ \rightarrow\ 24$ | $Wm\ \rightarrow\ (-24)$
| $Xp\ \rightarrow\ 34$ | $Xm\ \rightarrow\ (-34)$ | $ZH\ \rightarrow\ 32$ | $X0\ \rightarrow\ 33$ | $Y0\ \rightarrow\ 38$
| $Phip$ | $Phim\ \rightarrow\ 27$ | $Phi0\ \rightarrow\ 26$
| $H\ \rightarrow\ 25$ | $Eta\ \rightarrow\ 36$

As in the case of SUSY we introduce an internal dummy pdf code in order to have manageable arrays. Heavy neutrinos get numbers 41,43,45, while the heavy quarks have the numbers 40,42,44. I take them all as up type here.

let pdg_mw = function
| $L\ n$ when $n\ >\ 0\ \rightarrow\ 9\ +\ 2 \times n$
| $L\ n\ \rightarrow\ -\ 9\ +\ 2 \times n$
| $N\ n$ when $n\ >\ 0\ \rightarrow\ 10\ +\ 2 \times n$
| $N\ n\ \rightarrow\ -\ 10\ +\ 2 \times n$
| $U\ n$ when $n\ >\ 0\ \rightarrow\ 2 \times n$
| $U\ n\ \rightarrow\ 2 \times n$
| $D\ n$ when $n\ >\ 0\ \rightarrow\ -\ 1\ +\ 2 \times n$
| $D\ n\ \rightarrow\ 1\ +\ 2 \times n$
| $NH\ n$ when $n\ >\ 0\ \rightarrow\ 39\ +\ 2 \times n$
| $NH\ n\ \rightarrow\ -\ 39\ +\ 2 \times n$
| $QH\ n$ when $n\ >\ 0\ \rightarrow\ 38\ +\ 2 \times n$
| $QH\ n\ \rightarrow\ -\ 38\ +\ 2 \times n$
| $Gl\ \rightarrow\ 21$
| $Ga\ \rightarrow\ 22$ | $Z\ \rightarrow\ 23$
| $Wp\ \rightarrow\ 24$ | $Wm\ \rightarrow\ (-24)$
| $Xp\ \rightarrow\ 34$ | $Xm\ \rightarrow\ (-34)$ | $ZH\ \rightarrow\ 32$ | $X0\ \rightarrow\ 33$ | $Y0\ \rightarrow\ 38$
| $Phip$ | $Phim\ \rightarrow\ 27$ | $Phi0\ \rightarrow\ 26$
| $H\ \rightarrow\ 25$ | $Eta\ \rightarrow\ 36$

let $mass\_symbol\ f$ =
"mass(" ^ $string\_of\_int\ (abs\ (pdg\_mw\ f))$ ^ ")"

let $width\_symbol\ f$ =
"width(" ^ $string\_of\_int\ (abs\ (pdg\_mw\ f))$ ^ ")"

let $constant\_symbol$ = function
| $Unit\ \rightarrow$ "unit" | $Pi\ \rightarrow$ "PI" | $VHeavy\ \rightarrow$ "vheavy"
| $Alpha\_QED\ \rightarrow$ "alpha" | $E\ \rightarrow$ "e" | $G\_weak\ \rightarrow$ "g" | $Vev\ \rightarrow$ "vev"
| $Sin2thw\ \rightarrow$ "sin2thw" | $Sinthw\ \rightarrow$ "sinthw" | $Costhw\ \rightarrow$ "costhw"

| $Sinpsi$ → "sinpsi" | $Cospsi$ → "cospsi"
| $Atpsi$ → "atpsi" | $Sccs$ → "sccs"
| $Supp$ → "vF" | $Supp2$ → "v2F2"
| Q_lepton → "qlep" | Q_up → "qup" | Q_down → "qdwn"
| Q_Z_up → "qzup"
| G_zhthth → "gzhthth"
| G_NC_lepton → "gnclep" | $G\_NC\_neutrino$ → "gncneu"
| G_NC_up → "gncup" | G_NC_down → "gncdwn"
| G_NC_X → "gncx" | $G\_NC\_X\_t$ → "gncxt"
| G_NC_Y → "gncy" | $G\_NC\_Y\_t$ → "gncyt" | G_NC_H → "gnch"
| G_CC → "gcc" | I_G_CC → "i_gcc"
| $G\_NC\_h\_lepton$ → "gnchlep" | $G\_NC\_h\_neutrino$ → "gnchneu"
| $G\_NC\_h\_up$ → "gnchup" | $G\_NC\_h\_down$ → "gnchdwn"
| $G\_NC\_h\_top$ → "gnchtop" | $G\_NC\_h\_bot$ → "gnchbot"
| G_NCH_N → "gnchn" | G_NCH_U → "gnchu" | G_NCH_D → "gnchd"
| G_NCHt → "gncht"
| I_Q_W → "iqw" | I_G_ZWW → "igzww" | I_G_WWW → "igwww"
| I_Q_H → "iqh" | I_Q_ZH → "iqzh"
| I_G_Z1 → "igz1" | I_G_Z2 → "igz2" | I_G_Z3 → "igz3"
| I_G_Z4 → "igz4" | I_G_Z5 → "igz5" | I_G_Z6 → "igz6"
| G_HHthth → "ghhthth"
| G_WWWW → "gw4" | G_ZZWW → "gzzww"
| G_AZWW → "gazww" | G_AAWW → "gaaww"
| G_HWW → "ghww" | G_HZZ → "ghzz"
| G_heavy_HVV → "ghyhvv"
| G_heavy_HWW → "ghyhww"
| G_heavy_HZZ → "ghyhzz"
| G_HHWW → "ghhww" | G_HHZZ → "ghhzz"
| G_HHZZH → "ghhzzh"
| G_Hgg → "ghgg"
| G_Htt → "ghtt" | G_Hbb → "ghbb"
| $G\_Htautau$ → "ghtautau" | G_Hcc → "ghcc"
| G_Hthth → "ghthth" | G_Htht → "ghtht"
| G_Hqhq → "ghqhq"
| G_Ethth → "gethth" | G_Etht → "getht"
| G_Ett → "gett" | G_Ebb → "gebb"
| G_HGaGa → "ghgaga" | G_HGaZ → "ghgaz"
| G_EGaGa → "geaa" | G_EGaZ → "geaz" | G_EGlGl → "gegg"
| G_ZEH → "gzeh" | G_ZHEH → "gzheh"
| G_H3 → "gh3" | G_H4 → "gh4"
| $Mass\ f$ → "mass" ^ $flavor\_symbol\ f$
| $Width\ f$ → "width" ^ $flavor\_symbol\ f$
| Gs → "gs" | I_Gs → "igs" | $G2$ → "gs**2"
end

module $Xdim$ ($Flags$: BSM_flags) =
struct

```
let rcs = RCS.rename rcs_file "Modellib_BSM.Xdim"
    [ "SM with extradimensional resonances"]

open Coupling

let default_width = ref Timelike
let use_fudged_width = ref false

let options = Options.create
  [ "constant_width", Arg.Unit (fun () → default_width := Constant),
    "use constant width (also in t-channel)";
    "fudged_width", Arg.Set use_fudged_width,
    "use fudge factor for charge particle width";
    "custom_width", Arg.String (fun f → default_width := Custom f),
    "use custom width";
    "cancel_widths", Arg.Unit (fun () → default_width := Vanishing),
    "use vanishing width"]

type matter_field = L of int | N of int | U of int | D of int
type gauge_boson = Ga | Wp | Wm | Z | Gl
type other = Phip | Phim | Phi0 | H | Grav
type flavor = M of matter_field | G of gauge_boson | O of other

let matter_field f = M f
let gauge_boson f = G f
let other f = O f

type field =
  | Matter of matter_field
  | Gauge of gauge_boson
  | Other of other

let field = function
  | M f → Matter f
  | G f → Gauge f
  | O f → Other f

type gauge = unit

let gauge_symbol () =
  failwith "Modellib_BSM.Xdim.gauge_symbol: internal error"

let family n = List.map matter_field [ L n; N n; U n; D n ]

let external_flavors () =
  [ "1st Generation", ThoList.flatmap family [1; − 1];
    "2nd Generation", ThoList.flatmap family [2; − 2];
    "3rd Generation", ThoList.flatmap family [3; − 3];
    "Gauge Bosons", List.map gauge_boson [Ga; Z; Wp; Wm; Gl];
    "Higgs", List.map other [H];
    "Graviton", List.map other [Grav];
    "Goldstone Bosons", List.map other [Phip; Phim; Phi0] ]

let flavors () = ThoList.flatmap snd (external_flavors ())

let spinor n =
  if n ≥ 0 then
```

```
    Spinor
  else
    ConjSpinor

let lorentz = function
  | M f →
      begin match f with
      | L n → spinor n | N n → spinor n
      | U n → spinor n | D n → spinor n
      end
  | G f →
      begin match f with
      | Ga | Gl → Vector
      | Wp | Wm | Z → Massive_Vector
      end
  | O f →
      begin match f with
      | Grav → Tensor_2
      | _ → Scalar
      end

let color = function
  | M (U n) → Color.SUN (if n > 0 then 3 else -3)
  | M (D n) → Color.SUN (if n > 0 then 3 else -3)
  | G Gl → Color.AdjSUN 3
  | _ → Color.Singlet

let prop_spinor n =
  if n ≥ 0 then
    Prop_Spinor
  else
    Prop_ConjSpinor

let propagator = function
  | M f →
      begin match f with
      | L n → prop_spinor n | N n → prop_spinor n
      | U n → prop_spinor n | D n → prop_spinor n
      end
  | G f →
      begin match f with
      | Ga | Gl → Prop_Feynman
      | Wp | Wm | Z → Prop_Unitarity
      end
  | O f →
      begin match f with
      | Phip | Phim | Phi0 → Only_Insertion
      | H → Prop_Scalar
      | Grav → Prop_Tensor_2
      end
```

Optionally, ask for the fudge factor treatment for the widths of charged particles. Currently, this only applies to $W^{\pm}$ and top.

```
let width f =
  if !use_fudged_width then
    match f with
    | G Wp | G Wm | M (U 3) | M (U (−3)) | O Grav → Fudged
    | _ → !default_width
  else
    !default_width

let goldstone = function
  | G f →
      begin match f with
      | Wp → Some (O Phip, Coupling.Const 1)
      | Wm → Some (O Phim, Coupling.Const 1)
      | Z → Some (O Phi0, Coupling.Const 1)
      | _ → None
      end
  | _ → None

let conjugate = function
  | M f →
      M (begin match f with
      | L n → L (−n) | N n → N (−n)
      | U n → U (−n) | D n → D (−n)
      end)
  | G f →
      G (begin match f with
      | Gl → Gl | Ga → Ga | Z → Z
      | Wp → Wm | Wm → Wp
      end)
  | O f →
      O (begin match f with
      | Phip → Phim | Phim → Phip | Phi0 → Phi0
      | H → H | Grav → Grav
      end)

let fermion = function
  | M f →
      begin match f with
      | L n → if n > 0 then 1 else -1
      | N n → if n > 0 then 1 else -1
      | U n → if n > 0 then 1 else -1
      | D n → if n > 0 then 1 else -1
      end
  | G f →
      begin match f with
      | Gl | Ga | Z | Wp | Wm → 0
      end
  | O _ → 0

module Ch = Charges.QQ
let ( // ) = Algebra.Small_Rational.make
```

```
let generation' = function
  | 1 → [ 1//1; 0//1; 0//1]
  | 2 → [ 0//1; 1//1; 0//1]
  | 3 → [ 0//1; 0//1; 1//1]
  | −1 → [−1//1; 0//1; 0//1]
  | −2 → [ 0//1; −1//1; 0//1]
  | −3 → [ 0//1; 0//1; −1//1]
  | n → invalid_arg ("Xdim.generation':␣" ^ string_of_int n)

let generation f =
  match f with
   | M (L n | N n | U n | D n) → generation' n
   | G _ | O _ → [0//1; 0//1; 0//1]

let charge = function
  | M f →
      begin match f with
      | L n → if n > 0 then -1//1 else 1//1
      | N n → 0//1
      | U n → if n > 0 then 2//3 else -2//3
      | D n → if n > 0 then -1//3 else 1//3
      end
  | G f →
      begin match f with
      | Gl | Ga | Z → 0//1
      | Wp → 1//1
      | Wm → −1//1
      end
  | O f →
      begin match f with
      | H | Phi0 | Grav → 0//1
      | Phip → 1//1
      | Phim → −1//1
      end

let lepton = function
  | M f →
      begin match f with
      | L n | N n → if n > 0 then 1//1 else -1//1
      | U _ | D _ → 0//1
      end
  | G _ | O _ → 0//1

let baryon = function
  | M f →
      begin match f with
      | L _ | N _ → 0//1
      | U n | D n → if n > 0 then 1//1 else -1//1
      end
  | G _ | O _ → 0//1

let charges f =
  [ charge f; lepton f; baryon f] @ generation f
```

```
type constant =
  | Unit | Pi | Alpha_QED | Sin2thw
  | Sinthw | Costhw | E | G_weak | Vev
  | Q_lepton | Q_up | Q_down | G_CC | G_CCQ of int×int
  | G_NC_neutrino | G_NC_lepton | G_NC_up | G_NC_down
  | Gs | I_Gs | G2
  | I_Q_W | I_G_ZWW
  | G_WWWW | G_ZZWW | G_AZWW | G_AAWW
  | G_HWW | G_HHWW | G_HZZ | G_HHZZ
  | G_Htt | G_Hbb | G_Hcc | G_Htautau | G_H3 | G_H4
  | G_HGaZ | G_HGaGa | G_Hgg | G_Grav
  | Mass of flavor | Width of flavor
```

Two integer counters for the QCD and EW order of the couplings.

```
type orders = int × int

let orders = function
  | _ → (0, 0)

let input_parameters =
  []

let derived_parameters =
  []

let g_over_2_costh =
  Quot (Neg (Atom G_weak), Prod [Const 2; Atom Costhw])

let nc_coupling c t3 q =
  (Real_Array c,
   [Prod [g_over_2_costh; Diff (t3, Prod [Const 2; q; Atom Sin2thw])];
    Prod [g_over_2_costh; t3]])

let half = Quot (Const 1, Const 2)

let derived_parameter_arrays =
  [ nc_coupling G_NC_neutrino half (Const 0);
    nc_coupling G_NC_lepton (Neg half) (Const (−1));
    nc_coupling G_NC_up half (Quot (Const 2, Const 3));
    nc_coupling G_NC_down (Neg half) (Quot (Const (−1), Const 3)) ]

let parameters () =
  { input = input_parameters;
    derived = derived_parameters;
    derived_arrays = derived_parameter_arrays }

module F = Modeltools.Fusions (struct
  type f = flavor
  type c = constant
  let compare = compare
  let conjugate = conjugate
end)

let mgm ((m1, g, m2), fbf, c) = ((M m1, G g, M m2), fbf, c)
let mom ((m1, o, m2), fbf, c) = ((M m1, O o, M m2), fbf, c)
```

```
let electromagnetic_currents n =
  List.map mgm
    [ ((L (−n), Ga, L n), FBF (1, Psibar, V, Psi), Q_lepton);
      ((U (−n), Ga, U n), FBF (1, Psibar, V, Psi), Q_up);
      ((D (−n), Ga, D n), FBF (1, Psibar, V, Psi), Q_down) ]

let neutral_currents n =
  List.map mgm
    [ ((L (−n), Z, L n), FBF (1, Psibar, VA, Psi), G_NC_lepton);
      ((N (−n), Z, N n), FBF (1, Psibar, VA, Psi), G_NC_neutrino);
      ((U (−n), Z, U n), FBF (1, Psibar, VA, Psi), G_NC_up);
      ((D (−n), Z, D n), FBF (1, Psibar, VA, Psi), G_NC_down) ]

let color_currents n =
  List.map mgm
    [ ((U (−n), Gl, U n), FBF ((−1), Psibar, V, Psi), Gs);
      ((D (−n), Gl, D n), FBF ((−1), Psibar, V, Psi), Gs) ]

let charged_currents n =
  List.map mgm
    [ ((L (−n), Wm, N n), FBF (1, Psibar, VL, Psi), G_CC);
      ((N (−n), Wp, L n), FBF (1, Psibar, VL, Psi), G_CC);
      ((D (−n), Wm, U n), FBF (1, Psibar, VL, Psi), G_CC);
      ((U (−n), Wp, D n), FBF (1, Psibar, VL, Psi), G_CC) ]

let gravity_currents n =
  List.map mom
    [ ((L (−n), Grav, L n), Graviton_Spinor_Spinor 1, G_Grav);
      ((N (−n), Grav, N n), Graviton_Spinor_Spinor 1, G_Grav);
      ((U (−n), Grav, U n), Graviton_Spinor_Spinor 1, G_Grav);
      ((D (−n), Grav, D n), Graviton_Spinor_Spinor 1, G_Grav) ]

let yukawa =
  List.map mom
    [ ((U (−3), H, U 3), FBF (1, Psibar, S, Psi), G_Htt);
      ((D (−3), H, D 3), FBF (1, Psibar, S, Psi), G_Hbb);
      ((U (−2), H, U 2), FBF (1, Psibar, S, Psi), G_Hcc);
      ((L (−3), H, L 3), FBF (1, Psibar, S, Psi), G_Htautau) ]

let tgc ((g1, g2, g3), t, c) = ((G g1, G g2, G g3), t, c)

let standard_triple_gauge =
  List.map tgc
    [ ((Ga, Wm, Wp), Gauge_Gauge_Gauge 1, I_Q_W);
      ((Z, Wm, Wp), Gauge_Gauge_Gauge 1, I_G_ZWW);
      ((Gl, Gl, Gl), Gauge_Gauge_Gauge 1, I_Gs) ]

let triple_gauge =
    standard_triple_gauge

let qgc ((g1, g2, g3, g4), t, c) = ((G g1, G g2, G g3, G g4), t, c)

let gauge4 = Vector4 [(2, C_13_42); (−1, C_12_34); (−1, C_14_23)]
let minus_gauge4 = Vector4 [(−2, C_13_42); (1, C_12_34); (1, C_14_23)]
let standard_quartic_gauge =
```

```
  List.map qgc
    [ (Wm, Wp, Wm, Wp), gauge4, G_WWWW;
      (Wm, Z, Wp, Z), minus_gauge4, G_ZZWW;
      (Wm, Z, Wp, Ga), minus_gauge4, G_AZWW;
      (Wm, Ga, Wp, Ga), minus_gauge4, G_AAWW;
      (Gl, Gl, Gl, Gl), gauge4, G2]

let quartic_gauge =
  standard_quartic_gauge

let gravity_gauge =
  [ (O Grav, G Z, G Z), Graviton_Vector_Vector 1, G_Grav;
    (O Grav, G Wp, G Wm), Graviton_Vector_Vector 1, G_Grav;
    (O Grav, G Ga, G Ga), Graviton_Vector_Vector 1, G_Grav;
    (O Grav, G Gl, G Gl), Graviton_Vector_Vector 1, G_Grav ]

let standard_gauge_higgs =
  [ ((O H, G Wp, G Wm), Scalar_Vector_Vector 1, G_HWW);
    ((O H, G Z, G Z), Scalar_Vector_Vector 1, G_HZZ) ]

let standard_gauge_higgs4 =
  [ (O H, O H, G Wp, G Wm), Scalar2_Vector2 1, G_HHWW;
    (O H, O H, G Z, G Z), Scalar2_Vector2 1, G_HHZZ ]

let standard_higgs =
  [ (O H, O H, O H), Scalar_Scalar_Scalar 1, G_H3 ]

let standard_higgs4 =
  [ (O H, O H, O H, O H), Scalar4 1, G_H4 ]

let gravity_higgs =
  [ (O Grav, O H, O H), Graviton_Scalar_Scalar 1, G_Grav]

let anomalous_gauge_higgs =
  []

let anomalous_gauge_higgs4 =
  []

let anomalous_higgs =
  []

let anomaly_higgs =
  [ (O H, G Ga, G Ga), Dim5_Scalar_Gauge2 1, G_HGaGa;
    (O H, G Ga, G Z), Dim5_Scalar_Gauge2 1, G_HGaZ;
    (O H, G Gl, G Gl), Dim5_Scalar_Gauge2 1, G_Hgg ]

let anomalous_higgs4 =
  []

let gauge_higgs =
    standard_gauge_higgs

let gauge_higgs4 =
    standard_gauge_higgs4

let higgs =
    standard_higgs @ gravity_higgs
```

```
let higgs4 =
    standard_higgs4

let goldstone_vertices =
  [ ((O Phi0, G Wm, G Wp), Scalar_Vector_Vector 1, I_G_ZWW);
    ((O Php, G Ga, G Wm), Scalar_Vector_Vector 1, I_Q_W);
    ((O Php, G Z, G Wm), Scalar_Vector_Vector 1, I_G_ZWW);
    ((O Phim, G Wp, G Ga), Scalar_Vector_Vector 1, I_Q_W);
    ((O Phim, G Wp, G Z), Scalar_Vector_Vector 1, I_G_ZWW) ]

let vertices3 =
  (ThoList.flatmap electromagnetic_currents [1; 2; 3] @
   ThoList.flatmap neutral_currents [1; 2; 3] @
   ThoList.flatmap color_currents [1; 2; 3] @
   ThoList.flatmap charged_currents [1; 2; 3] @
   ThoList.flatmap gravity_currents [1; 2; 3] @
   yukawa @ triple_gauge @ gravity_gauge @
   gauge_higgs @ higgs @ anomaly_higgs
   @ goldstone_vertices)

let vertices4 =
  quartic_gauge @ gauge_higgs4 @ higgs4

let vertices () = (vertices3, vertices4, [])
```

For efficiency, make sure that *F.of_vertices vertices* is evaluated only once.

```
let table = F.of_vertices (vertices ())
let fuse2 = F.fuse2 table
let fuse3 = F.fuse3 table
let fuse = F.fuse table
let max_degree () = 4

let flavor_of_string = function
  | "e-" → M (L 1) | "e+" → M (L (−1))
  | "mu-" → M (L 2) | "mu+" → M (L (−2))
  | "tau-" → M (L 3) | "tau+" → M (L (−3))
  | "nue" → M (N 1) | "nuebar" → M (N (−1))
  | "numu" → M (N 2) | "numubar" → M (N (−2))
  | "nutau" → M (N 3) | "nutaubar" → M (N (−3))
  | "u" → M (U 1) | "ubar" → M (U (−1))
  | "c" → M (U 2) | "cbar" → M (U (−2))
  | "t" → M (U 3) | "tbar" → M (U (−3))
  | "d" → M (D 1) | "dbar" → M (D (−1))
  | "s" → M (D 2) | "sbar" → M (D (−2))
  | "b" → M (D 3) | "bbar" → M (D (−3))
  | "g" | "gl" → G Gl
  | "A" → G Ga | "Z" | "Z0" → G Z
  | "W+" → G Wp | "W-" → G Wm
  | "H" → O H
  | "GG" → O Grav
  | _ → invalid_arg "Modellib_BSM.Xdim.flavor_of_string"

let flavor_to_string = function
```

```
  | M f →
      begin match f with
      | L 1 → "e-" | L (−1) → "e+"
      | L 2 → "mu-" | L (−2) → "mu+"
      | L 3 → "tau-" | L (−3) → "tau+"
      | L _ → invalid_arg
            "Modellib_BSM.Xdim.flavor_to_string:␣invalid␣lepton"
      | N 1 → "nue" | N (−1) → "nuebar"
      | N 2 → "numu" | N (−2) → "numubar"
      | N 3 → "nutau" | N (−3) → "nutaubar"
      | N _ → invalid_arg
            "Modellib_BSM.Xdim.flavor_to_string:␣invalid␣neutrino"
      | U 1 → "u" | U (−1) → "ubar"
      | U 2 → "c" | U (−2) → "cbar"
      | U 3 → "t" | U (−3) → "tbar"
      | U _ → invalid_arg
            "Modellib_BSM.Xdim.flavor_to_string:␣invalid␣up␣type␣quark"
      | D 1 → "d" | D (−1) → "dbar"
      | D 2 → "s" | D (−2) → "sbar"
      | D 3 → "b" | D (−3) → "bbar"
      | D _ → invalid_arg
            "Modellib_BSM.Xdim.flavor_to_string:␣invalid␣down␣type␣quark"
      end
  | G f →
      begin match f with
      | Gl → "g"
      | Ga → "A" | Z → "Z"
      | Wp → "W+" | Wm → "W-"
      end
  | O f →
      begin match f with
      | Phip → "phi+" | Phim → "phi-" | Phi0 → "phi0"
      | H → "H" | Grav → "GG"
      end

let flavor_to_TeX = function
  | M f →
      begin match f with
      | L 1 → "e^-" | L (−1) → "e^+"
      | L 2 → "\\mu^-" | L (−2) → "\\mu^+"
      | L 3 → "\\tau^-" | L (−3) → "\\tau^+"
      | L _ → invalid_arg
            "Modellib_BSM.Xdim.flavor_to_TeX:␣invalid␣lepton"
      | N 1 → "\\nu_e" | N (−1) → "\\bar{\\nu}_e"
      | N 2 → "\\nu_\\mu" | N (−2) → "\\bar{\\nu}_\\mu"
      | N 3 → "\\nu_\\tau" | N (−3) → "\\bar{\\nu}_\\tau"
      | N _ → invalid_arg
            "Modellib_BSM.Xdim.flavor_to_TeX:␣invalid␣neutrino"
      | U 1 → "u" | U (−1) → "\\bar{u}"
      | U 2 → "c" | U (−2) → "\\bar{c}"
```

```
      | U 3 → "t" | U (−3) → "\\bar{t}"
      | U _ → invalid_arg
          "Modellib_BSM.Xdim.flavor_to_TeX: invalid up type quark"
      | D 1 → "d" | D (−1) → "\\bar{d}"
      | D 2 → "s" | D (−2) → "\\bar{s}"
      | D 3 → "b" | D (−3) → "\\bar{b}"
      | D _ → invalid_arg
          "Modellib_BSM.Xdim.flavor_to_TeX: invalid down type quark"
      end
  | G f →
      begin match f with
      | Gl → "g"
      | Ga → "\\gamma" | Z → "Z"
      | Wp → "W^+" | Wm → "W^-"
      end
  | O f →
      begin match f with
      | Phip → "\\phi^+" | Phim → "\\phi^-" | Phi0 →
"\\phi^0"
      | H → "H" | Grav → "G"
      end

let flavor_symbol = function
  | M f →
      begin match f with
      | L n when n > 0 → "l" ^ string_of_int n
      | L n → "l" ^ string_of_int (abs n) ^ "b"
      | N n when n > 0 → "n" ^ string_of_int n
      | N n → "n" ^ string_of_int (abs n) ^ "b"
      | U n when n > 0 → "u" ^ string_of_int n
      | U n → "u" ^ string_of_int (abs n) ^ "b"
      | D n when n > 0 → "d" ^ string_of_int n
      | D n → "d" ^ string_of_int (abs n) ^ "b"
      end
  | G f →
      begin match f with
      | Gl → "gl"
      | Ga → "a" | Z → "z"
      | Wp → "wp" | Wm → "wm"
      end
  | O f →
      begin match f with
      | Phip → "pp" | Phim → "pm" | Phi0 → "p0"
      | H → "h" | Grav → "gv"
      end

let pdg = function
  | M f →
      begin match f with
      | L n when n > 0 → 9 + 2 × n
      | L n → − 9 + 2 × n
```

```
          | N n when n > 0 → 10 + 2 × n
          | N n → − 10 + 2 × n
          | U n when n > 0 → 2 × n
          | U n → 2 × n
          | D n when n > 0 → − 1 + 2 × n
          | D n → 1 + 2 × n
          end
      | G f →
          begin match f with
          | Gl → 21
          | Ga → 22 | Z → 23
          | Wp → 24 | Wm → (−24)
          end
      | O f →
          begin match f with
          | Phip | Phim → 27 | Phi0 → 26
          | H → 25 | Grav → 39
          end

  let mass_symbol f =
    "mass(" ^ string_of_int (abs (pdg f)) ^ ")"

  let width_symbol f =
    "width(" ^ string_of_int (abs (pdg f)) ^ ")"

  let constant_symbol = function
    | Unit → "unit" | Pi → "PI"
    | Alpha_QED → "alpha" | E → "e" | G_weak → "g" | Vev →
"vev"
    | Sin2thw → "sin2thw" | Sinthw → "sinthw" | Costhw →
"costhw"
    | Q_lepton → "qlep" | Q_up → "qup" | Q_down → "qdwn"
    | G_NC_lepton → "gnclep" | G_NC_neutrino → "gncneu"
    | G_NC_up → "gncup" | G_NC_down → "gncdwn"
    | Gs → "gs" | I_Gs → "igs" | G2 → "gs**2"
    | G_CC → "gcc"
    | G_CCQ (n1, n2) → "gccq" ^ string_of_int n1 ^ string_of_int n2
    | I_Q_W → "iqw" | I_G_ZWW → "igzww"
    | G_WWWW → "gw4" | G_ZZWW → "gzzww"
    | G_AZWW → "gazww" | G_AAWW → "gaaww"
    | G_HWW → "ghww" | G_HZZ → "ghzz"
    | G_HHWW → "ghhww" | G_HHZZ → "ghhzz"
    | G_Htt → "ghtt" | G_Hbb → "ghbb"
    | G_Htautau → "ghtautau" | G_Hcc → "ghcc"
    | G_HGaZ → "ghgaz" | G_HGaGa → "ghgaga" | G_Hgg →
"ghgg"
    | G_H3 → "gh3" | G_H4 → "gh4" | G_Grav → "ggrav"
    | Mass f → "mass" ^ flavor_symbol f
    | Width f → "width" ^ flavor_symbol f

end

module UED (Flags : BSM_flags) =
```

```
struct
  let rcs = RCS.rename rcs_file "Modellib_BSM.UED"
      [ "Universal Extra Dimensions"]

  open Coupling

  let default_width = ref Timelike
  let use_fudged_width = ref false

  let options = Options.create
    [ "constant_width", Arg.Unit (fun () → default_width := Constant),
      "use constant width (also in t-channel)";
      "fudged_width", Arg.Set use_fudged_width,
      "use fudge factor for charge particle width";
      "custom_width", Arg.String (fun f → default_width := Custom f),
      "use custom width";
      "cancel_widths", Arg.Unit (fun () → default_width := Vanishing),
      "use vanishing width"]

  type matter_field = L of int | N of int | U of int | D of int
        | L_K1_L of int | L_K1_R of int | N_K1 of int
        | L_K2_L of int | L_K2_R of int | N_K2 of int
        | U_K1_L of int | U_K2_L of int | D_K1_L of int | D_K2_L of int
        | U_K1_R of int | U_K2_R of int | D_K1_R of int | D_K2_R of int
  type gauge_boson = Ga | Wp | Wm | Z | Gl | Gl_K1 | Gl_K2
        | B1 | B2 | Z1 | Z2 | Wp1 | Wm1 | Wp2 | Wm2
  type other = Phip | Phim | Phi0 | H | H1up | H1um
        | H1dp | H1dm | H2up | H2um | H2dp | H2dm
        | Grav
  type flavor = M of matter_field | G of gauge_boson | O of other

  let matter_field f = M f
  let gauge_boson f = G f
  let other f = O f

  type field =
    | Matter of matter_field
    | Gauge of gauge_boson
    | Other of other

  let field = function
    | M f → Matter f
    | G f → Gauge f
    | O f → Other f

  type gauge = unit

  let gauge_symbol () =
    failwith "Modellib_BSM.UED.gauge_symbol: internal error"

  let family n = List.map matter_field [ L n; N n; U n; D n; L_K1_L n;
    L_K1_R n; L_K2_L n; L_K2_R n; N_K1 n; N_K2 n; U_K1_L n; U_K2_L n;
    D_K1_L n; D_K2_L n; U_K1_R n; U_K2_R n; D_K1_R n; D_K2_R n]
```

We don't introduce a special index for the higher excitations but make them parts of the particles' names.

```
let external_flavors () =
  [ "1st␣Generation", ThoList.flatmap family [1; −1];
    "2nd␣Generation", ThoList.flatmap family [2; −2];
    "3rd␣Generation", ThoList.flatmap family [3; −3];
    "Gauge␣Bosons", List.map gauge_boson [Ga; Z; Wp; Wm; Gl;
          Gl_K1; Gl_K2; B1; B2; Z1; Z2; Wp1 ; Wm1; Wp2; Wm2];
    "Higgs", List.map other [H; H1up; H1um; H1dp; H1dm;
          H2up; H2um; H2dp; H2dm];
    "Graviton", List.map other [Grav];
    "Goldstone␣Bosons", List.map other [Phip; Phim; Phi0] ]

let flavors () = ThoList.flatmap snd (external_flavors ())

let spinor n =
  if n ≥ 0 then
    Spinor
  else
    ConjSpinor

let lorentz = function
  | M f →
      begin match f with
      | L n → spinor n | N n → spinor n
      | U n → spinor n | D n → spinor n
      | L_K1_L n → spinor n | L_K1_R n → spinor n
      | L_K2_L n → spinor n | L_K2_R n → spinor n
      | N_K1 n → spinor n | N_K2 n → spinor n
      | U_K1_L n → spinor n | U_K1_R n → spinor n
      | U_K2_L n → spinor n | U_K2_R n → spinor n
      | D_K1_L n → spinor n | D_K1_R n → spinor n
      | D_K2_L n → spinor n | D_K2_R n → spinor n
      end
  | G f →
      begin match f with
      | Ga | Gl → Vector
      | Wp | Wm | Z | Gl_K1 | Gl_K2 | B1 | B2
      | Z1 | Z2 | Wp1 | Wm1 | Wp2 | Wm2 → Massive_Vector
      end
  | O f →
      begin match f with
      | Grav → Tensor_2
      | _ → Scalar
      end

let color = function
  | M (U n) → Color.SUN (if n > 0 then 3 else -3)
  | M (D n) → Color.SUN (if n > 0 then 3 else -3)
  | M (U_K1_L n) → Color.SUN (if n > 0 then 3 else -3)
  | M (D_K1_L n) → Color.SUN (if n > 0 then 3 else -3)
  | M (U_K1_R n) → Color.SUN (if n > 0 then 3 else -3)
  | M (D_K1_R n) → Color.SUN (if n > 0 then 3 else -3)
  | M (U_K2_L n) → Color.SUN (if n > 0 then 3 else -3)
```

```
    | M (D_K2_L n) → Color.SUN (if n > 0 then 3 else -3)
    | M (U_K2_R n) → Color.SUN (if n > 0 then 3 else -3)
    | M (D_K2_R n) → Color.SUN (if n > 0 then 3 else -3)
    | G Gl | G Gl_K1 | G Gl_K2 → Color.AdjSUN 3
    | _ → Color.Singlet

let prop_spinor n =
  if n ≥ 0 then
    Prop_Spinor
  else
    Prop_ConjSpinor

let propagator = function
  | M f →
      begin match f with
      | L n → prop_spinor n | N n → prop_spinor n
      | U n → prop_spinor n | D n → prop_spinor n
      | L_K1_L n → prop_spinor n | L_K1_R n → prop_spinor n
      | L_K2_L n → prop_spinor n | L_K2_R n → prop_spinor n
      | N_K1 n → prop_spinor n | N_K2 n → prop_spinor n
      | U_K1_L n → prop_spinor n | U_K1_R n → prop_spinor n
      | U_K2_L n → prop_spinor n | U_K2_R n → prop_spinor n
      | D_K1_L n → prop_spinor n | D_K1_R n → prop_spinor n
      | D_K2_L n → prop_spinor n | D_K2_R n → prop_spinor n
      end
  | G f →
      begin match f with
      | Ga | Gl → Prop_Feynman
      | Wp | Wm | Z | Gl_K1 | Gl_K2 | B1 | B2
      | Z1 | Z2 | Wp1 | Wm1 | Wp2 | Wm2 → Prop_Unitarity
      end
  | O f →
      begin match f with
      | Phip | Phim | Phi0 → Only_Insertion
      | H | H1up | H1um | H1dp | H1dm | H2up
      | H2um | H2dp | H2dm → Prop_Scalar
      | Grav → Prop_Tensor_2
      end
```

Optionally, ask for the fudge factor treatment for the widths of charged particles. Currently, this only applies to $W^{\pm}$ and top.

```
let width f =
  if !use_fudged_width then
    match f with
    | G Wp | G Wm | M (U 3) | M (U (−3)) | O Grav → Fudged
    | _ → !default_width
  else
    !default_width

let goldstone = function
  | G f →
      begin match f with
```

```
        | Wp → Some (O Phip, Coupling.Const 1)
        | Wm → Some (O Phim, Coupling.Const 1)
        | Z → Some (O Phi0, Coupling.Const 1)
        | _ → None
        end
    | _ → None

let conjugate = function
    | M f →
        M (begin match f with
        | L n → L (−n) | N n → N (−n)
        | U n → U (−n) | D n → D (−n)
        | L_K1_L n → L_K1_L (−n) | L_K1_R n → L_K1_R (−n)
        | L_K2_L n → L_K2_L (−n) | L_K2_R n → L_K2_R (−n)
        | N_K1 n → N_K1 (−n) | N_K2 n → N_K2 (−n)
        | U_K1_L n → U_K1_L (−n) | U_K1_R n → U_K1_R (−n)
        | U_K2_L n → U_K2_L (−n) | U_K2_R n → U_K2_R (−n)
        | D_K1_L n → D_K1_L (−n) | D_K1_R n → D_K1_R (−n)
        | D_K2_L n → D_K2_L (−n) | D_K2_R n → D_K2_R (−n)
        end)
    | G f →
        G (begin match f with
        | Gl → Gl | Ga → Ga | Z → Z
        | Wp → Wm | Wm → Wp
        | Gl_K1 → Gl_K1 | Gl_K2 → Gl_K2 | B1 → B1 | B2 →
B2
        | Z1 → Z1 | Z2 → Z2 | Wp1 → Wm1 | Wm1 → Wp1
        | Wp2 → Wm2 | Wm2 → Wp2
        end)
    | O f →
        O (begin match f with
        | Phip → Phim | Phim → Phip | Phi0 → Phi0
        | H → H | H1up → H1um | H1um → H1up
        | H1dp → H1dm | H1dm → H1dp
        | H2up → H2um | H2um → H2up
        | H2dp → H2dm | H2dm → H2dp
        | Grav → Grav
        end)

let fermion = function
    | M f →
        begin match f with
        | L n → if n > 0 then 1 else -1
        | N n → if n > 0 then 1 else -1
        | U n → if n > 0 then 1 else -1
        | D n → if n > 0 then 1 else -1
        | L_K1_L n → if n > 0 then 1 else -1
        | L_K2_L n → if n > 0 then 1 else -1
        | L_K1_R n → if n > 0 then 1 else -1
        | L_K2_R n → if n > 0 then 1 else -1
        | U_K1_L n → if n > 0 then 1 else -1
```

```
      | U_K2_L n → if n > 0 then 1 else -1
      | U_K1_R n → if n > 0 then 1 else -1
      | U_K2_R n → if n > 0 then 1 else -1
      | D_K1_L n → if n > 0 then 1 else -1
      | D_K2_L n → if n > 0 then 1 else -1
      | D_K1_R n → if n > 0 then 1 else -1
      | D_K2_R n → if n > 0 then 1 else -1
      | N_K1 n → if n > 0 then 1 else -1
      | N_K2 n → if n > 0 then 1 else -1
      end
  | G f →
      begin match f with
      | Gl | Ga | Z | Wp | Wm | Gl_K1 | Gl_K2
      | B1 | B2 | Z1 | Z2 | Wp1 | Wm1 | Wp2
      | Wm2 → 0
      end
  | O _ → 0

module Ch = Charges.QQ

let ( // ) = Algebra.Small_Rational.make

let generation' = function
  | 1 → [ 1//1; 0//1; 0//1]
  | 2 → [ 0//1; 1//1; 0//1]
  | 3 → [ 0//1; 0//1; 1//1]
  | -1 → [-1//1; 0//1; 0//1]
  | -2 → [ 0//1; -1//1; 0//1]
  | -3 → [ 0//1; 0//1; -1//1]
  | n → invalid_arg ("SM.generation': " ^ string_of_int n)

let generation f =
  match f with
  | M (L n | N n | U n | D n | L_K1_L n | L_K2_L n
     | L_K1_R n | L_K2_R n | N_K1 n | N_K2 n | U_K1_L n
     | U_K2_L n | U_K1_R n | U_K2_R n | D_K1_L n |
D_K2_L n
     | D_K1_R n | D_K2_R n ) → generation' n
  | G _ | O _ → [0//1; 0//1; 0//1]

let charge = function
  | M f →
      begin match f with
      | L n | L_K1_L n | L_K2_L n | L_K1_R n
      | L_K2_R n → if n > 0 then -1//1 else 1//1
      | N n | N_K1 n | N_K2 n → 0//1
      | U n | U_K1_L n | U_K2_L n | U_K1_R n
      | U_K2_R n → if n > 0 then 2//3 else -2//3
      | D n | D_K1_L n | D_K2_L n | D_K1_R n
      | D_K2_R n → if n > 0 then -1//3 else 1//3
      end
  | G f →
      begin match f with
```

```
        | Gl | Gl_K1 | Gl_K2 | Ga | Z
        | B1 | B2 | Z1 | Z2 → 0//1
        | Wp | Wp1 | Wp2 → 1//1
        | Wm | Wm1 | Wm2 → −1//1
        end
    | O f →
        begin match f with
        | H | Phi0 | Grav → 0//1
        | H1up | H1dp | H2up | H2dp | Phip → 1//1
        | H1um | H1dm | H2um | H2dm | Phim → −1//1
        end

let lepton = function
    | M f →
        begin match f with
        | L n | N n | L_K1_L n | L_K1_R n | L_K2_L n
        | L_K2_R n | N_K1 n | N_K2 n → if n > 0 then 1//1 else -
1//1
        | U _ | D _ | _ → 0//1
        end
    | G _ | O _ → 0//1

let baryon = function
    | M f →
        begin match f with
        | U n | D n | U_K1_L n | U_K1_R n | U_K2_L n
        | U_K2_R n | D_K1_L n | D_K1_R n | D_K2_L n
        | D_K2_R n → if n > 0 then 1//1 else -1//1
        | L _ | N _ | _ → 0//1
        end
    | G _ | O _ → 0//1

let charges f =
    [ charge f; lepton f; baryon f] @ generation f

type constant =
    | Unit | Pi | Alpha_QED | Sin2thw
    | Sinthw | Costhw | E | G_weak | Vev
    | Q_lepton | Q_up | Q_down | G_CC | G_CCQ of int×int
    | G_NC_neutrino | G_NC_lepton | G_NC_up | G_NC_down
    | I_Q_W | I_G_ZWW | I_Q_W_K | I_G_ZWW_K1 |
I_G_ZWW_K2
    | I_G_ZWW_K3
    | G_WWWW | G_ZZWW | G_AZWW | G_AAWW
    | G_HWW | G_HHWW | G_HZZ | G_HHZZ
    | G_Htt | G_Hbb | G_Hcc | G_Htautau | G_H3 | G_H4
    | G_HGaZ | G_HGaGa | G_Hgg
    | Gs | I_Gs | I_GsRt2 | G2 | G22 | G_Grav
    | Mass of flavor | Width of flavor
```

Two integer counters for the QCD and EW order of the couplings.

```
type orders = int × int
```

```
let orders = function
  | _ → (0,0)

let input_parameters =
  []

let derived_parameters =
  []

let g_over_2_costh =
  Quot (Neg (Atom G_weak), Prod [Const 2; Atom Costhw])

let nc_coupling c t3 q =
  (Real_Array c,
   [Prod [g_over_2_costh; Diff (t3, Prod [Const 2; q; Atom Sin2thw])];
    Prod [g_over_2_costh; t3]])

let half = Quot (Const 1, Const 2)

let derived_parameter_arrays =
  [ nc_coupling G_NC_neutrino half (Const 0);
    nc_coupling G_NC_lepton (Neg half) (Const (−1));
    nc_coupling G_NC_up half (Quot (Const 2, Const 3));
    nc_coupling G_NC_down (Neg half) (Quot (Const (−1), Const 3)) ]

let parameters () =
  { input = input_parameters;
    derived = derived_parameters;
    derived_arrays = derived_parameter_arrays }

module F = Modeltools.Fusions (struct
  type f = flavor
  type c = constant
  let compare = compare
  let conjugate = conjugate
end)

let mgm ((m1, g, m2), fbf, c) = ((M m1, G g, M m2), fbf, c)
let mom ((m1, o, m2), fbf, c) = ((M m1, O o, M m2), fbf, c)

let electromagnetic_currents n =
  List.map mgm
    [ ((L (−n), Ga, L n), FBF (1, Psibar, V, Psi), Q_lepton);
      ((U (−n), Ga, U n), FBF (1, Psibar, V, Psi), Q_up);
      ((D (−n), Ga, D n), FBF (1, Psibar, V, Psi), Q_down) ]

let neutral_currents n =
  List.map mgm
    [ ((L (−n), Z, L n), FBF (1, Psibar, VA, Psi), G_NC_lepton);
      ((N (−n), Z, N n), FBF (1, Psibar, VA, Psi), G_NC_neutrino);
      ((U (−n), Z, U n), FBF (1, Psibar, VA, Psi), G_NC_up);
      ((D (−n), Z, D n), FBF (1, Psibar, VA, Psi), G_NC_down) ]

let charged_currents n =
  List.map mgm
    [ ((L (−n), Wm, N n), FBF (1, Psibar, VL, Psi), G_CC);
```

```
      ((N (-n), Wp, L n), FBF (1, Psibar, VL, Psi), G_CC);
      ((D (-n), Wm, U n), FBF (1, Psibar, VL, Psi), G_CC);
      ((U (-n), Wp, D n), FBF (1, Psibar, VL, Psi), G_CC) ]

let color_currents n =
    List.map mgm
      [ ((U (-n), Gl, U n), FBF ((-1), Psibar, V, Psi), Gs);
        ((D (-n), Gl, D n), FBF ((-1), Psibar, V, Psi), Gs) ]

let gravity_currents n =
  List.map mom
    [ ((L (-n), Grav, L n), Graviton_Spinor_Spinor 1, G_Grav);
      ((N (-n), Grav, N n), Graviton_Spinor_Spinor 1, G_Grav);
      ((U (-n), Grav, U n), Graviton_Spinor_Spinor 1, G_Grav);
      ((D (-n), Grav, D n), Graviton_Spinor_Spinor 1, G_Grav) ]

let yukawa =
  List.map mom
    [ ((U (-3), H, U 3), FBF (1, Psibar, S, Psi), G_Htt);
      ((D (-3), H, D 3), FBF (1, Psibar, S, Psi), G_Hbb);
      ((U (-2), H, U 2), FBF (1, Psibar, S, Psi), G_Hcc);
      ((L (-3), H, L 3), FBF (1, Psibar, S, Psi), G_Htautau) ]

let tgc ((g1, g2, g3), t, c) = ((G g1, G g2, G g3), t, c)
```

Gluons should be included in just that way.

```
let standard_triple_gauge =
  List.map tgc
    [ ((Ga, Wm, Wp), Gauge_Gauge_Gauge 1, I_Q_W);
      ((Ga, Wm1, Wp1), Gauge_Gauge_Gauge 1, I_Q_W_K);
      ((Z, Wm, Wp), Gauge_Gauge_Gauge 1, I_G_ZWW);
      ((Z, Wm1, Wp1), Gauge_Gauge_Gauge 1, I_G_ZWW_K1);
      ((Z1, Wm, Wp1), Gauge_Gauge_Gauge 1, I_G_ZWW_K2);
      ((Z1, Wm1, Wp), Gauge_Gauge_Gauge 1, I_G_ZWW_K2);
      ((Z2, Wm1, Wp2), Gauge_Gauge_Gauge 1, I_G_ZWW_K3);
      ((Gl, Gl, Gl), Gauge_Gauge_Gauge 1, I_Gs);
      ((Gl, Gl_K2, Gl_K2), Gauge_Gauge_Gauge (-1), I_Gs);
      ((Gl, Gl_K1, Gl_K1), Gauge_Gauge_Gauge 1, I_Gs);
      ((Gl_K2, Gl_K1, Gl_K1), Gauge_Gauge_Gauge 1, I_GsRt2)]

let triple_gauge =
    standard_triple_gauge

let qgc ((g1, g2, g3, g4), t, c) = ((G g1, G g2, G g3, G g4), t, c)

let gauge4 = Vector4 [(2, C_13_42); (-1, C_12_34); (-1, C_14_23)]
let minus_gauge4 = Vector4 [(-2, C_13_42); (1, C_12_34); (1, C_14_23)]
let standard_quartic_gauge =
  List.map qgc
    [ (Wm, Wp, Wm, Wp), gauge4, G_WWWW;
      (Wm, Z, Wp, Z), minus_gauge4, G_ZZWW;
      (Wm, Z, Wp, Ga), minus_gauge4, G_AZWW;
      (Wm, Ga, Wp, Ga), minus_gauge4, G_AAWW;
      ((Gl, Gl, Gl, Gl), gauge4, G2);
```

```
      ((Gl, Gl, Gl_K1, Gl_K1), gauge4, G2);
      ((Gl, Gl, Gl_K2, Gl_K2), gauge4, G2);
      ((Gl_K1, Gl_K1, Gl_K2, Gl_K2), gauge4, G2);
      ((Gl_K2, Gl_K2, Gl_K2, Gl_K2), gauge4, G22)]

let quartic_gauge =
  standard_quartic_gauge

let gravity_gauge =
  [ (O Grav, G Z, G Z), Graviton_Vector_Vector 1, G_Grav;
    (O Grav, G Wp, G Wm), Graviton_Vector_Vector 1, G_Grav;
    (O Grav, G Ga, G Ga), Graviton_Vector_Vector 1, G_Grav;
    (O Grav, G Gl, G Gl), Graviton_Vector_Vector 1, G_Grav ]

let standard_gauge_higgs =
  [ ((O H, G Wp, G Wm), Scalar_Vector_Vector 1, G_HWW);
    ((O H, G Z, G Z), Scalar_Vector_Vector 1, G_HZZ) ]

let standard_gauge_higgs4 =
  [ (O H, O H, G Wp, G Wm), Scalar2_Vector2 1, G_HHWW;
    (O H, O H, G Z, G Z), Scalar2_Vector2 1, G_HHZZ ]

let standard_higgs =
  [ (O H, O H, O H), Scalar_Scalar_Scalar 1, G_H3 ]

let standard_higgs4 =
  [ (O H, O H, O H, O H), Scalar4 1, G_H4 ]

let gravity_higgs =
  [ (O Grav, O H, O H), Graviton_Scalar_Scalar 1, G_Grav]

let anomalous_gauge_higgs =
  []

let anomalous_gauge_higgs4 =
  []

let anomalous_higgs =
  []

let anomaly_higgs =
  [ (O H, G Ga, G Ga), Dim5_Scalar_Gauge2 1, G_HGaGa;
    (O H, G Ga, G Z), Dim5_Scalar_Gauge2 1, G_HGaZ;
    (O H, G Gl, G Gl), Dim5_Scalar_Gauge2 1, G_Hgg ]

let anomalous_higgs4 =
  []

let gauge_higgs =
    standard_gauge_higgs

let gauge_higgs4 =
    standard_gauge_higgs4

let higgs =
    standard_higgs @ gravity_higgs

let higgs4 =
```

```
    standard_higgs4

let goldstone_vertices =
  [ ((O Phi0, G Wm, G Wp), Scalar_Vector_Vector 1, I_G_ZWW);
    ((O Phip, G Ga, G Wm), Scalar_Vector_Vector 1, I_Q_W);
    ((O Phip, G Z, G Wm), Scalar_Vector_Vector 1, I_G_ZWW);
    ((O Phim, G Wp, G Ga), Scalar_Vector_Vector 1, I_Q_W);
    ((O Phim, G Wp, G Z), Scalar_Vector_Vector 1, I_G_ZWW) ]

let vertices3 =
  (ThoList.flatmap electromagnetic_currents [1; 2; 3] @
   ThoList.flatmap neutral_currents [1; 2; 3] @
   ThoList.flatmap charged_currents [1; 2; 3] @
   ThoList.flatmap color_currents [1; 2; 3] @
   ThoList.flatmap gravity_currents [1; 2; 3] @
   yukawa @ triple_gauge @ gravity_gauge @
   gauge_higgs @ higgs @ anomaly_higgs
   @ goldstone_vertices)

let vertices4 =
  quartic_gauge @ gauge_higgs4 @ higgs4

let vertices () = (vertices3, vertices4, [])
```

For efficiency, make sure that $F.of\_vertices$ $vertices$ is evaluated only once.

```
let table = F.of_vertices (vertices ())
let fuse2 = F.fuse2 table
let fuse3 = F.fuse3 table
let fuse = F.fuse table
let max_degree () = 4

let flavor_of_string = function
  | "e-" → M (L 1) | "e+" → M (L (−1))
  | "mu-" → M (L 2) | "mu+" → M (L (−2))
  | "tau-" → M (L 3) | "tau+" → M (L (−3))
  | "nue" → M (N 1) | "nuebar" → M (N (−1))
  | "numu" → M (N 2) | "numubar" → M (N (−2))
  | "nutau" → M (N 3) | "nutaubar" → M (N (−3))
  | "u" → M (U 1) | "ubar" → M (U (−1))
  | "c" → M (U 2) | "cbar" → M (U (−2))
  | "t" → M (U 3) | "tbar" → M (U (−3))
  | "d" → M (D 1) | "dbar" → M (D (−1))
  | "s" → M (D 2) | "sbar" → M (D (−2))
  | "b" → M (D 3) | "bbar" → M (D (−3))
  | "uk1l" → M (U_K1_L 1) | "uk1lbar" → M (U_K1_L (−1))
  | "ck1l" → M (U_K1_L 2) | "ck1lbar" → M (U_K1_L (−2))
  | "tk1l" → M (U_K1_L 3) | "tk1lbar" → M (U_K1_L (−3))
  | "dk1l" → M (D_K1_L 1) | "dk1lbar" → M (D_K1_L (−1))
  | "sk1l" → M (D_K1_L 2) | "sk1lbar" → M (D_K1_L (−2))
  | "bk1l" → M (D_K1_L 3) | "bk1lbar" → M (D_K1_L (−3))
  | "uk1r" → M (U_K1_R 1) | "uk1rbar" → M (U_K1_R (−1))
  | "ck1r" → M (U_K1_R 2) | "ck1rbar" → M (U_K1_R (−2))
  | "tk1r" → M (U_K1_R 3) | "tk1rbar" → M (U_K1_R (−3))
```

```
| "dk1r" → M (D_K1_R 1) | "dk1rbar" → M (D_K1_R (−1))
| "sk1r" → M (D_K1_R 2) | "sk1rbar" → M (D_K1_R (−2))
| "bk1r" → M (D_K1_R 3) | "bk1rbar" → M (D_K1_R (−3))
| "uk2l" → M (U_K2_L 1) | "uk2lbar" → M (U_K2_L (−1))
| "ck2l" → M (U_K2_L 2) | "ck2lbar" → M (U_K2_L (−2))
| "tk2l" → M (U_K2_L 3) | "tk2lbar" → M (U_K2_L (−3))
| "dk2l" → M (D_K2_L 1) | "dk2lbar" → M (D_K2_L (−1))
| "sk2l" → M (D_K2_L 2) | "sk2lbar" → M (D_K2_L (−2))
| "bk2l" → M (D_K2_L 3) | "bk2lbar" → M (D_K2_L (−3))
| "uk2r" → M (U_K2_R 1) | "uk2rbar" → M (U_K2_R (−1))
| "ck2r" → M (U_K2_R 2) | "ck2rbar" → M (U_K2_R (−2))
| "tk2r" → M (U_K2_R 3) | "tk2rbar" → M (U_K2_R (−3))
| "dk2r" → M (D_K2_R 1) | "dk2rbar" → M (D_K2_R (−1))
| "sk2r" → M (D_K2_R 2) | "sk2rbar" → M (D_K2_R (−2))
| "bk2r" → M (D_K2_R 3) | "bk2rbar" → M (D_K2_R (−3))
| "g" | "gl" → G Gl
| "g_k1" | "gl_k1" → G Gl_K1
| "g_k2" | "gl_k2" → G Gl_K2
| "b1" → G B1 | "b2" → G B2 | "z1" → G Z1 | "z2" → G Z2
| "W1+" → G Wp1 | "W1-" → G Wm1
| "W2+" → G Wp2 | "W2-" → G Wm2
| "A" → G Ga | "Z" | "Z0" → G Z
| "W+" → G Wp | "W-" → G Wm
| "H" → O H | "H1u+" → O H1up | "H1u-" → O H1um
| "H1d+" → O H1dp | "H1d-" → O H1dm
| "H2u+" → O H2up | "H2u-" → O H2um
| "H2d+" → O H2dp | "H2d-" → O H2dm
| "GG" → O Grav
| "ek1l-" → M (L_K1_L 1) | "ek1l+" → M (L_K1_L (−1))
| "muk1l-" → M (L_K1_L 2) | "mu1l+" → M (L_K1_L (−2))
| "tauk1l-" → M (L_K1_L 3) | "tauk1l+" → M (L_K1_L (−3))
| "ek1r-" → M (L_K1_R 1) | "ek1r+" → M (L_K1_R (−1))
| "muk1r-" → M (L_K1_R 2) | "mu1r+" → M (L_K1_R (−2))
| "tau1r-" → M (L_K1_R 3) | "tauk1r+" → M (L_K1_R (−3))
| "ek2l-" → M (L_K2_L 1) | "ek2l+" → M (L_K2_L (−1))
| "muk2l-" → M (L_K2_L 2) | "mu2l+" → M (L_K2_L (−2))
| "tauk2l-" → M (L_K2_L 3) | "tauk2l+" → M (L_K2_L (−3))
| "ek2r-" → M (L_K2_R 1) | "ek2r+" → M (L_K2_R (−1))
| "muk2r-" → M (L_K2_R 2) | "mu2r+" → M (L_K2_R (−2))
| "tau2r-" → M (L_K2_R 3) | "tauk2r+" → M (L_K2_R (−3))
| "nuek1" → M (N_K1 1) | "nuek1bar" → M (N_K1 (−1))
| "numuk1" → M (N_K1 2) | "numuk1bar" → M (N_K1 (−2))
| "nutauk1" → M (N_K1 3) | "nutauk1bar" → M (N_K1 (−3))
| "nuek2" → M (N_K2 1) | "nuek2bar" → M (N_K2 (−1))
| "numuk2" → M (N_K2 2) | "numuk2bar" → M (N_K2 (−2))
| "nutauk2" → M (N_K2 3) | "nutauk2bar" → M (N_K2 (−3))
| _ → invalid_arg "Modellib_BSM.UED.flavor_of_string"

let flavor_to_string = function
  | M f →
```

```
begin match f with
| L 1 → "e-" | L (−1) → "e+"
| L 2 → "mu-" | L (−2) → "mu+"
| L 3 → "tau-" | L (−3) → "tau+"
| L _ → invalid_arg
      "Modellib_BSM.UED.flavor_to_string:␣invalid␣lepton"
| N 1 → "nue" | N (−1) → "nuebar"
| N 2 → "numu" | N (−2) → "numubar"
| N 3 → "nutau" | N (−3) → "nutaubar"
| N _ → invalid_arg
      "Modellib_BSM.UED.flavor_to_string:␣invalid␣neutrino"
| U 1 → "u" | U (−1) → "ubar"
| U 2 → "c" | U (−2) → "cbar"
| U 3 → "t" | U (−3) → "tbar"
| U _ → invalid_arg
      "Modellib_BSM.UED.flavor_to_string:␣invalid␣up␣type␣quark"
| D 1 → "d" | D (−1) → "dbar"
| D 2 → "s" | D (−2) → "sbar"
| D 3 → "b" | D (−3) → "bbar"
| D _ → invalid_arg
      "Modellib_BSM.UED.flavor_to_string:␣invalid␣down␣type␣quark"
| U_K1_L 1 → "uk1l" | U_K1_L (−1) → "uk1lbar"
| U_K1_L 2 → "ck1l" | U_K1_L (−2) → "ck1lbar"
| U_K1_L 3 → "tk1l" | U_K1_L (−3) → "tk1lbar"
| U_K1_L _ → invalid_arg
      "Modellib_BSM.UED.flavor_to_string:␣invalid␣up␣type␣quark"
| D_K1_L 1 → "dk1l" | D_K1_L (−1) → "dk1lbar"
| D_K1_L 2 → "sk1l" | D_K1_L (−2) → "sk1lbar"
| D_K1_L 3 → "bk1l" | D_K1_L (−3) → "bk1lbar"
| D_K1_L _ → invalid_arg
      "Modellib_BSM.UED.flavor_to_string:␣invalid␣down␣type␣quark"
| U_K1_R 1 → "uk1r" | U_K1_R (−1) → "uk1rbar"
| U_K1_R 2 → "ck1r" | U_K1_R (−2) → "ck1rbar"
| U_K1_R 3 → "tk1r" | U_K1_R (−3) → "tk1rbar"
| U_K1_R _ → invalid_arg
      "Modellib_BSM.UED.flavor_to_string:␣invalid␣up␣type␣quark"
| D_K1_R 1 → "dk1r" | D_K1_R (−1) → "dk1rbar"
| D_K1_R 2 → "sk1r" | D_K1_R (−2) → "sk1rbar"
| D_K1_R 3 → "bk1r" | D_K1_R (−3) → "bk1rbar"
| D_K1_R _ → invalid_arg
      "Modellib_BSM.UED.flavor_to_string:␣invalid␣down␣type␣quark"
| U_K2_L 1 → "uk2l" | U_K2_L (−1) → "uk2lbar"
| U_K2_L 2 → "ck2l" | U_K2_L (−2) → "ck2lbar"
| U_K2_L 3 → "tk2l" | U_K2_L (−3) → "tk2lbar"
| U_K2_L _ → invalid_arg
      "Modellib_BSM.UED.flavor_to_string:␣invalid␣up␣type␣quark"
| D_K2_L 1 → "dk2l" | D_K2_L (−1) → "dk2lbar"
| D_K2_L 2 → "sk2l" | D_K2_L (−2) → "sk2lbar"
| D_K2_L 3 → "bk2l" | D_K2_L (−3) → "bk2lbar"
| D_K2_L _ → invalid_arg
```

```
        "Modellib_BSM.UED.flavor_to_string: invalid down type quark"
    | U_K2_R 1 → "uk2r" | U_K2_R (−1) → "uk2rbar"
    | U_K2_R 2 → "ck2r" | U_K2_R (−2) → "ck2rbar"
    | U_K2_R 3 → "tk2r" | U_K2_R (−3) → "tk2rbar"
    | U_K2_R _ → invalid_arg
        "Modellib_BSM.UED.flavor_to_string: invalid up type quark"
    | D_K2_R 1 → "dk2r" | D_K2_R (−1) → "dk2rbar"
    | D_K2_R 2 → "sk2r" | D_K2_R (−2) → "sk2rbar"
    | D_K2_R 3 → "bk2r" | D_K2_R (−3) → "bk2rbar"
    | D_K2_R _ → invalid_arg
        "Modellib_BSM.UED.flavor_to_string: invalid down type quark"
    | L_K1_L 1 → "ek1l-" | L_K1_L (−1) → "ek1l+"
    | L_K1_L 2 → "muk1l-" | L_K1_L (−2) → "muk1l+"
    | L_K1_L 3 → "tauk1l-" | L_K1_L (−3) → "tauk1l+"
    | L_K1_L _ → invalid_arg
        "Modellib_BSM.UED.flavor_to_string: invalid lepton"
    | L_K1_R 1 → "ek1r-" | L_K1_R (−1) → "ek1r+"
    | L_K1_R 2 → "muk1r-" | L_K1_R (−2) → "muk1r+"
    | L_K1_R 3 → "tauk1r-" | L_K1_R (−3) → "tauk1r+"
    | L_K1_R _ → invalid_arg
        "Modellib_BSM.UED.flavor_to_string: invalid lepton"
    | L_K2_L 1 → "ek2l-" | L_K2_L (−1) → "ek2l+"
    | L_K2_L 2 → "muk2l-" | L_K2_L (−2) → "muk2l+"
    | L_K2_L 3 → "tauk2l-" | L_K2_L (−3) → "tauk2l+"
    | L_K2_L _ → invalid_arg
        "Modellib_BSM.UED.flavor_to_string: invalid lepton"
    | L_K2_R 1 → "ek2r-" | L_K2_R (−1) → "ek2r+"
    | L_K2_R 2 → "muk2r-" | L_K2_R (−2) → "muk2r+"
    | L_K2_R 3 → "tauk2r-" | L_K2_R (−3) → "tauk2r+"
    | L_K2_R _ → invalid_arg
        "Modellib_BSM.UED.flavor_to_string: invalid lepton"
    | N_K1 1 → "nuek1" | N_K1 (−1) → "nuek1bar"
    | N_K1 2 → "numuk1" | N_K1 (−2) → "numuk1bar"
    | N_K1 3 → "nutauk1" | N_K1 (−3) → "nutauk1bar"
    | N_K1 _ → invalid_arg
        "Modellib_BSM.UED.flavor_to_string: invalid neutrino"
    | N_K2 1 → "nuek2" | N_K2 (−1) → "nuek2bar"
    | N_K2 2 → "numuk2" | N_K2 (−2) → "numuk2bar"
    | N_K2 3 → "nutauk2" | N_K2 (−3) → "nutauk2bar"
    | N_K2 _ → invalid_arg
        "Modellib_BSM.UED.flavor_to_string: invalid neutrino"
    end
| G f →
    begin match f with
    | Gl → "g"
    | Ga → "A" | Z → "Z"
    | Wp → "W+" | Wm → "W-"
    | Gl_K1 → "gk1" | Gl_K2 → "gk2"
    | B1 → "b1" | B2 → "b2" | Z1 → "z1" | Z2 → "z2"
    | Wp1 → "W1+" | Wm1 → "W1-"
```

```
      | Wp2 → "W2+" | Wm2 → "W2-"
      end
  | O f →
      begin match f with
      | Phip → "phi+" | Phim → "phi-" | Phi0 → "phi0"
      | H → "H" | H1up → "H1u+" | H1um → "H1u-"
      | H1dp → "H1d+" | H1dm → "H1d-"
      | H2up → "H2u+" | H2um → "H2u-"
      | H2dp → "H2d+" | H2dm → "H2d-"
      | Grav → "GG"
      end

let flavor_to_TeX = function
  | M f →
      begin match f with
      | L 1 → "e^-" | L (-1) → "e^+"
      | L 2 → "\\mu^-" | L (-2) → "\\mu^+"
      | L 3 → "\\tau^-" | L (-3) → "\\tau^+"
      | L _ → invalid_arg
          "Modellib_BSM.UED.flavor_to_TeX:␣invalid␣lepton"
      | N 1 → "\\nu_e" | N (-1) → "\\bar{\\nu}_e"
      | N 2 → "\\nu_\\mu" | N (-2) → "\\bar{\\nu}_\\mu"
      | N 3 → "\\nu_\\tau" | N (-3) → "\\bar{\\nu}_\\tau"
      | N _ → invalid_arg
          "Modellib_BSM.UED.flavor_to_TeX:␣invalid␣neutrino"
      | U 1 → "u" | U (-1) → "\\bar{u}"
      | U 2 → "c" | U (-2) → "\\bar{c}"
      | U 3 → "t" | U (-3) → "\\bar{t}"
      | U _ → invalid_arg
          "Modellib_BSM.UED.flavor_to_TeX:␣invalid␣up␣type␣quark"
      | D 1 → "d" | D (-1) → "dbar"
      | D 2 → "s" | D (-2) → "sbar"
      | D 3 → "b" | D (-3) → "bbar"
      | D _ → invalid_arg
          "Modellib_BSM.UED.flavor_to_TeX:␣invalid␣down␣type␣quark"
      | U_K1_L 1 → "u^\\prime_L" | U_K1_L (-1) → "\\bar{u}^\\prime_L"
      | U_K1_L 2 → "c^\\prime_L" | U_K1_L (-2) → "\\bar{c}^\\prime_L"
      | U_K1_L 3 → "t^\\prime_L" | U_K1_L (-3) → "\\bar{t}^\\prime_L"
      | U_K1_L _ → invalid_arg
          "Modellib_BSM.UED.flavor_to_TeX:␣invalid␣up␣type␣quark"
      | D_K1_L 1 → "d^\\prime_L" | D_K1_L (-1) → "\\bar{d}^\\prime_L"
      | D_K1_L 2 → "s^\\prime_L" | D_K1_L (-2) → "\\bar{s}^\\prime_L"
      | D_K1_L 3 → "b^\\prime_L" | D_K1_L (-3) → "\\bar{b}^\\prime_L"
      | D_K1_L _ → invalid_arg
          "Modellib_BSM.UED.flavor_to_TeX:␣invalid␣down␣type␣quark"
      | U_K1_R 1 → "u^\\prime_R" | U_K1_R (-1) → "\\bar{u}^\\prime_R"
      | U_K1_R 2 → "c^\\prime_R" | U_K1_R (-2) → "\\bar{c}^\\prime_R"
      | U_K1_R 3 → "t^\\prime_R" | U_K1_R (-3) → "\\bar{t}^\\prime_R"
      | U_K1_R _ → invalid_arg
          "Modellib_BSM.UED.flavor_to_TeX:␣invalid␣up␣type␣quark"
```

```
    | D_K1_R 1 → "d^\\prime_R" | D_K1_R (−1) → "\\bar{d}^\\prime_R"
    | D_K1_R 2 → "s^\\prime_R" | D_K1_R (−2) → "\\bar{s}^\\prime_R"
    | D_K1_R 3 → "b^\\prime_R" | D_K1_R (−3) → "\\bar{b}^\\prime_R"
    | D_K1_R _ → invalid_arg
        "Modellib_BSM.UED.flavor_to_TeX:␣invalid␣down␣type␣quark"
    | U_K2_L 1 → "u^{\\prime\\prime}_L" | U_K2_L (−1) →
"\\bar{u}^{\\prime\\prime}_L"
    | U_K2_L 2 → "c^{\\prime\\prime}_L" | U_K2_L (−2) →
"\\bar{c}^{\\prime\\prime}_L"
    | U_K2_L 3 → "t^{\\prime\\prime}_L" | U_K2_L (−3) →
"\\bar{t}^{\\prime\\prime}_L"
    | U_K2_L _ → invalid_arg
        "Modellib_BSM.UED.flavor_to_TeX:␣invalid␣up␣type␣quark"
    | D_K2_L 1 → "d^{\\prime\\prime}_L" | D_K2_L (−1) →
"\\bar{d}^{\\prime\\prime}_L"
    | D_K2_L 2 → "s^{\\prime\\prime}_L" | D_K2_L (−2) →
"\\bar{s}^{\\prime\\prime}_L"
    | D_K2_L 3 → "b^{\\prime\\prime}_L" | D_K2_L (−3) →
"\\bar{b}^{\\prime\\prime}_L"
    | D_K2_L _ → invalid_arg
        "Modellib_BSM.UED.flavor_to_TeX:␣invalid␣down␣type␣quark"
    | U_K2_R 1 → "u^{\\prime\\prime}_R" | U_K2_R (−1) →
"\\bar{u}^{\\prime\\prime}_R"
    | U_K2_R 2 → "c^{\\prime\\prime}_R" | U_K2_R (−2) →
"\\bar{c}^{\\prime\\prime}_R"
    | U_K2_R 3 → "t^{\\prime\\prime}_R" | U_K2_R (−3) →
"\\bar{t}^{\\prime\\prime}_R"
    | U_K2_R _ → invalid_arg
        "Modellib_BSM.UED.flavor_to_TeX:␣invalid␣up␣type␣quark"
    | D_K2_R 1 → "d^\\prime_R" | D_K2_R (−1) → "\\bar{d}^{\\prime\\prime}_R"
    | D_K2_R 2 → "s^\\prime_R" | D_K2_R (−2) → "\\bar{s}^{\\prime\\prime}_R"
    | D_K2_R 3 → "b^\\prime_R" | D_K2_R (−3) → "\\bar{b}^{\\prime\\prime}_R"
    | D_K2_R _ → invalid_arg
        "Modellib_BSM.UED.flavor_to_TeX:␣invalid␣down␣type␣quark"
    | L_K1_L 1 → "e_L^{\\prime,,-}" | L_K1_L (−1) → "\\bar{e}_L^{\\prime,,+}"
    | L_K1_L 2 → "\\mu_L^{\\prime,,-}" | L_K1_L (−2) →
"\\bar{\\mu}_L^{{\\prime,,+}"
    | L_K1_L 3 → "\\tau_L^{\\prime,,-}" | L_K1_L (−3) →
"\\bar{\\tau}_L^{\\prime,,+}"
    | L_K1_L _ → invalid_arg
        "Modellib_BSM.UED.flavor_to_TeX:␣invalid␣lepton"
    | L_K1_R 1 → "e_R^{\\prime,,-}" | L_K1_R (−1) →
"\\bar{e}_R^{\\prime,,+}"
    | L_K1_R 2 → "\\mu_R{\\prime,,-}" | L_K1_R (−2) →
"\\bar{\\mu}_R^{\\prime,,+}"
    | L_K1_R 3 → "\\tau_R{\\prime,,-}" | L_K1_R (−3) →
"\\bar{\\tau}_R{\\prime,,+}"
    | L_K1_R _ → invalid_arg
        "Modellib_BSM.UED.flavor_to_TeX:␣invalid␣lepton"
    | L_K2_L 1 → "e^{\\prime\\prime,,-}_L" | L_K2_L (−1) →
```

```
"\\bar{e}_L^{\\prime\\prime,,+}"
          | L_K2_L 2 → "\\mu_L^{\\prime\\prime,,-}" | L_K2_L (−2) →
"\\bar{\\mu}_L^{\\prime\\prime,,+}"
          | L_K2_L 3 → "\\tau_L^{\\prime\\prime,,-}" | L_K2_L (−3) →
"\\bar{\\tau}_L^{\\prime\\prime,,+}"
          | L_K2_L _ → invalid_arg
                "Modellib_BSM.UED.flavor_to_TeX:␣invalid␣lepton"
          | L_K2_R 1 → "e_R^{\\prime\\prime,,-}" | L_K2_R (−1) →
"\\bar{e}_R^{\\prime\\prime,,+}"
          | L_K2_R 2 → "\\mu_R^{\\prime\\prime,,-}" | L_K2_R (−2) →
"\\bar{\\mu}_R^{\\prime\\prime,,+}"
          | L_K2_R 3 → "\\tau_R{\\prime\\prime,,-}" | L_K2_R (−3) →
"\\bar{\\tau}_R^{\\prime\\prime,,+}"
          | L_K2_R _ → invalid_arg
                "Modellib_BSM.UED.flavor_to_TeX:␣invalid␣lepton"
          | N_K1 1 → "\\nu_e^\\prime" | N_K1 (−1) → "\\bar{\\nu}_e^\\prime"
          | N_K1 2 → "\\nu_\\mu^\\prime" | N_K1 (−2) → "\\bar{\\nu}_\\mu^\\prime"
          | N_K1 3 → "\\nu_\\tau^\\prime" | N_K1 (−3) → "\\bar{\\nu}_\\tau^\\prime"
          | N_K1 _ → invalid_arg
                "Modellib_BSM.UED.flavor_to_TeX:␣invalid␣neutrino"
          | N_K2 1 → "\\nu_e^{\\prime\\prime}" | N_K2 (−1) →
"\\bar{\\nu}_e^{\\prime\\prime}"
          | N_K2 2 → "\\nu_\\mu^{\\prime\\prime}" | N_K2 (−2) →
"\\bar{\\nu}_\\mu^{\\prime\\prime}"
          | N_K2 3 → "\\nu_\\tau^{\\prime\\prime}" | N_K2 (−3) →
"\\bar{\\nu}_\\tau^{\\prime\\prime}"
          | N_K2 _ → invalid_arg
                "Modellib_BSM.UED.flavor_to_TeX:␣invalid␣neutrino"
          end
      | G f →
          begin match f with
          | Gl → "g"
          | Ga → "\\gamma" | Z → "Z"
          | Wp → "W^+" | Wm → "W^-"
          | Gl_K1 → "g^\\prime" | Gl_K2 → "g^{\\prime\\prime}"
          | B1 → "B^\\prime" | B2 → "B^{\\prime\\prime}"
          | Z1 → "Z^\\prime" | Z2 → "Z^{\\prime\\prime}"
          | Wp1 → "W^{\\prime,,+}" | Wm1 → "W^{\\prime,,-}"
          | Wp2 → "W^{\\prime\\prime,,+}" | Wm2 → "W^{\\prime\\prime,,-}"
          end
      | O f →
          begin match f with
          | Phip → "\\phi^+" | Phim → "\\phi^-" | Phi0 →
"\\phi^0"
          | H → "H" | H1up → "H1u+" | H1um → "H1u-"
          | H1dp → "H1d+" | H1dm → "H1d-"
          | H2up → "H2u+" | H2um → "H2u-"
          | H2dp → "H2d+" | H2dm → "H2d-"
          | Grav → "G^\\prime"
          end
```

```
let flavor_symbol = function
  | M f ->
      begin match f with
      | L n when n > 0 -> "l" ^ string_of_int n
      | L n -> "l" ^ string_of_int (abs n) ^ "b"
      | N n when n > 0 -> "n" ^ string_of_int n
      | N n -> "n" ^ string_of_int (abs n) ^ "b"
      | U n when n > 0 -> "u" ^ string_of_int n
      | U n -> "u" ^ string_of_int (abs n) ^ "b"
      | D n when n > 0 -> "d" ^ string_of_int n
      | D n -> "d" ^ string_of_int (abs n) ^ "b"
      | L_K1_L n when n > 0 -> "lk1l" ^ string_of_int n
      | L_K1_L n -> "lk1l" ^ string_of_int (abs n) ^ "b"
      | L_K1_R n when n > 0 -> "lk1r" ^ string_of_int n
      | L_K1_R n -> "lk1r" ^ string_of_int (abs n) ^ "b"
      | L_K2_L n when n > 0 -> "lk2l" ^ string_of_int n
      | L_K2_L n -> "lk2l" ^ string_of_int (abs n) ^ "b"
      | L_K2_R n when n > 0 -> "lk2r" ^ string_of_int n
      | L_K2_R n -> "lk2r" ^ string_of_int (abs n) ^ "b"
      | U_K1_L n when n > 0 -> "uk1l" ^ string_of_int n
      | U_K1_L n -> "uk1l" ^ string_of_int (abs n) ^ "b"
      | U_K1_R n when n > 0 -> "uk1r" ^ string_of_int n
      | U_K1_R n -> "uk1r" ^ string_of_int (abs n) ^ "b"
      | U_K2_L n when n > 0 -> "uk2l" ^ string_of_int n
      | U_K2_L n -> "uk2l" ^ string_of_int (abs n) ^ "b"
      | U_K2_R n when n > 0 -> "uk2r" ^ string_of_int n
      | U_K2_R n -> "uk2r" ^ string_of_int (abs n) ^ "b"
      | D_K1_L n when n > 0 -> "dk1l" ^ string_of_int n
      | D_K1_L n -> "dk1l" ^ string_of_int (abs n) ^ "b"
      | D_K1_R n when n > 0 -> "dk1r" ^ string_of_int n
      | D_K1_R n -> "dk1r" ^ string_of_int (abs n) ^ "b"
      | D_K2_L n when n > 0 -> "dk2l" ^ string_of_int n
      | D_K2_L n -> "dk2l" ^ string_of_int (abs n) ^ "b"
      | D_K2_R n when n > 0 -> "dk2r" ^ string_of_int n
      | D_K2_R n -> "dk2r" ^ string_of_int (abs n) ^ "b"
      | N_K1 n when n > 0 -> "nk1" ^ string_of_int n
      | N_K1 n -> "nk1" ^ string_of_int (abs n) ^ "b"
      | N_K2 n when n > 0 -> "nk2" ^ string_of_int n
      | N_K2 n -> "nk2" ^ string_of_int (abs n) ^ "b"
      end
  | G f ->
      begin match f with
      | Gl -> "gl"
      | Ga -> "a" | Z -> "z"
      | Wp -> "wp" | Wm -> "wm"
      | Gl_K1 -> "gk1" | Gl_K2 -> "gk2"
      | B1 -> "b1" | B2 -> "b2" | Z1 -> "z1" | Z2 -> "z2"
      | Wp1 -> "wp1" | Wm1 -> "wm1"
      | Wp2 -> "wp2" | Wm2 -> "wm2"
      end
```

```
  | O f →
      begin match f with
      | Phip → "pp" | Phim → "pm" | Phi0 → "p0"
      | H → "h" | H1up → "h1up" | H1um → "h1um"
      | H1dp → "h1dp" | H1dm → "h1dm"
      | H2up → "h2up" | H2um → "h2um"
      | H2dp → "h2dp" | H2dm → "h2dm"
      | Grav → "gv"
      end

let pdg = function
  | M f →
      begin match f with
      | L n when n > 0 → 9 + 2 × n
      | L n → − 9 + 2 × n
      | N n when n > 0 → 10 + 2 × n
      | N n → − 10 + 2 × n
      | U n when n > 0 → 2 × n
      | U n → 2 × n
      | D n when n > 0 → − 1 + 2 × n
      | D n → 1 + 2 × n
      | U_K1_L n when n > 0 → 4000000 + 2 × n
      | U_K1_L n → − 4000000 + 2 × n
      | D_K1_L n when n > 0 → 3999999 + 2 × n
      | D_K1_L n → − 3999999 + 2 × n
      | U_K1_R n when n > 0 → 5000000 + 2 × n
      | U_K1_R n → − 5000000 + 2 × n
      | D_K1_R n when n > 0 → 4999999 + 2 × n
      | D_K1_R n → − 4999999 + 2 × n
      | U_K2_L n when n > 0 → 6000000 + 2 × n
      | U_K2_L n → − 6000000 + 2 × n
      | D_K2_L n when n > 0 → 5999999 + 2 × n
      | D_K2_L n → − 5999999 + 2 × n
      | U_K2_R n when n > 7000000 → 2 × n
      | U_K2_R n → − 7000000 + 2 × n
      | D_K2_R n when n > 0 → 6999999 + 2 × n
      | D_K2_R n → − 6999999 + 2 × n
      | L_K1_L n when n > 0 → 4000009 + 2 × n
      | L_K1_L n → − 4000009 + 2 × n
      | L_K1_R n when n > 0 → 5000009 + 2 × n
      | L_K1_R n → − 5000009 + 2 × n
      | L_K2_L n when n > 0 → 6000009 + 2 × n
      | L_K2_L n → − 6000009 + 2 × n
      | L_K2_R n when n > 0 → 7000009 + 2 × n
      | L_K2_R n → − 7000009 + 2 × n
      | N_K1 n when n > 0 → 4000010 + 2 × n
      | N_K1 n → − 4000010 + 2 × n
      | N_K2 n when n > 0 → 6000010 + 2 × n
      | N_K2 n → − 6000010 + 2 × n
      end
```

```
  | G f →
      begin match f with
      | Gl → 21
      | Ga → 22 | Z → 23
      | Wp → 24 | Wm → (−24)
      | Gl_K1 → 4000021 | Gl_K2 → 6000021
      | B1 → 4000022 | B2 → 6000022
      | Z1 → 4000023 | Z2 → 6000024
      | Wp1 → 4000024 | Wm1 → (−4000024)
      | Wp2 → 6000024 | Wm2 → (−6000024)
      end
  | O f →
      begin match f with
      | Phip | Phim → 27 | Phi0 → 26
      | H → 25 | H1up → 4000036 | H1um → (−4000036)
      | H1dp → 4000037 | H1dm → (−4000037)
      | H2up → 6000036 | H2um → (−6000036)
      | H2dp → 6000037 | H2dm → (−6000037)
      | Grav → 39
      end

let mass_symbol f =
  "mass(" ^ string_of_int (abs (pdg f)) ^ ")"

let width_symbol f =
  "width(" ^ string_of_int (abs (pdg f)) ^ ")"

let constant_symbol = function
  | Unit → "unit" | Pi → "PI"
  | Alpha_QED → "alpha" | E → "e" | G_weak → "g" | Vev →
"vev"
  | Sin2thw → "sin2thw" | Sinthw → "sinthw" | Costhw →
"costhw"
  | Q_lepton → "qlep" | Q_up → "qup" | Q_down → "qdwn"
  | G_NC_lepton → "gnclep" | G_NC_neutrino → "gncneu"
  | G_NC_up → "gncup" | G_NC_down → "gncdwn"
  | G_CC → "gcc"
  | G_CCQ (n1, n2) → "gccq" ^ string_of_int n1 ^ string_of_int n2
  | I_Q_W → "iqw" | I_G_ZWW → "igzww"
  | I_Q_W_K → "iqwk" | I_G_ZWW_K1 → "igzwwk1"
  | I_G_ZWW_K2 → "igzwwk2" | I_G_ZWW_K3 → "igzwwk3"
  | G_WWWW → "gw4" | G_ZZWW → "gzzww"
  | G_AZWW → "gazww" | G_AAWW → "gaaww"
  | G_HWW → "ghww" | G_HZZ → "ghzz"
  | G_HHWW → "ghhww" | G_HHZZ → "ghhzz"
  | G_Htt → "ghtt" | G_Hbb → "ghbb"
  | G_Htautau → "ghtautau" | G_Hcc → "ghcc"
  | G_HGaZ → "ghgaz" | G_HGaGa → "ghgaga" | G_Hgg →
"ghgg"
  | G_H3 → "gh3" | G_H4 → "gh4"
  | G2 → "gs**2" | Gs → "gs" | I_Gs → "igs" | I_GsRt2 →
"igs/sqrt(2.0_default)"
```

```
      | G22 → "gs**2/2.0_default"
      | G_Grav → "ggrav"
      | Mass f → "mass" ^ flavor_symbol f
      | Width f → "width" ^ flavor_symbol f

  end

module GravTest (Flags : BSM_flags) =
  struct
    let rcs = RCS.rename rcs_file "Modellib_BSM.GravTest"
        [ "Testing␣of␣Gravitinos"]

    open Coupling

    let default_width = ref Timelike
    let use_fudged_width = ref false

    let options = Options.create
      [ "constant_width", Arg.Unit (fun () → default_width := Constant),
        "use␣constant␣width␣(also␣in␣t-channel)";
        "fudged_width", Arg.Set use_fudged_width,
        "use␣fudge␣factor␣for␣charge␣particle␣width";
        "custom_width", Arg.String (fun f → default_width := Custom f),
        "use␣custom␣width";
        "cancel_widths", Arg.Unit (fun () → default_width := Vanishing),
        "use␣vanishing␣width"]

    type matter_field = L of int | N of int | U of int | D of int | SL of int
    type gauge_boson = Ga | Wp | Wm | Z | Gl | Phino
    type other = Phip | Phim | Phi0 | H | Grino
    type flavor = M of matter_field | G of gauge_boson | O of other

    let matter_field f = M f
    let gauge_boson f = G f
    let other f = O f

    type field =
      | Matter of matter_field
      | Gauge of gauge_boson
      | Other of other

    let field = function
      | M f → Matter f
      | G f → Gauge f
      | O f → Other f

    type gauge = unit

    let gauge_symbol () =
      failwith "Modellib_BSM.SM.gauge_symbol:␣internal␣error"

    let family n = List.map matter_field [ L n; SL n; N n; U n; D n ]

    let external_flavors () =
      [ "1st␣Generation", ThoList.flatmap family [1; − 1];
        "2nd␣Generation", ThoList.flatmap family [2; − 2];
        "3rd␣Generation", ThoList.flatmap family [3; − 3];
```

```
      "Gauge␣Bosons", List.map gauge_boson [Ga; Z; Wp; Wm; Gl; Phino];
      "Higgs", List.map other [H];
      "Gravitino", List.map other [Grino];
      "Goldstone␣Bosons", List.map other [Phip; Phim; Phi0] ]

let flavors () = ThoList.flatmap snd (external_flavors ())

let spinor n =
  if n ≥ 0 then
    Spinor
  else
    ConjSpinor

let lorentz = function
  | M f →
      begin match f with
      | L n → spinor n | N n → spinor n
      | U n → spinor n | D n → spinor n
      | SL _ → Scalar
      end
  | G f →
      begin match f with
      | Ga | Gl → Vector
      | Wp | Wm | Z → Massive_Vector
      | Phino → Majorana
      end
  | O f →
      begin match f with
      | Grino → Vectorspinor
      | _ → Scalar
      end

let color = function
  | M (U n) → Color.SUN (if n > 0 then 3 else -3)
  | M (D n) → Color.SUN (if n > 0 then 3 else -3)
  | G Gl → Color.AdjSUN 3
  | _ → Color.Singlet

let prop_spinor n =
  if n ≥ 0 then
    Prop_Spinor
  else
    Prop_ConjSpinor

let propagator = function
  | M f →
      begin match f with
      | L n → prop_spinor n | N n → prop_spinor n
      | U n → prop_spinor n | D n → prop_spinor n
      | SL n → Prop_Scalar
      end
  | G f →
      begin match f with
```

```
        | Ga | Gl → Prop_Feynman
        | Wp | Wm | Z → Prop_Unitarity
        | Phino → Prop_Majorana
        end
    | O f →
        begin match f with
        | Phip | Phim | Phi0 → Only_Insertion
        | H → Prop_Scalar
        | Grino → Prop_Vectorspinor
        end
```

Optionally, ask for the fudge factor treatment for the widths of charged particles. Currently, this only applies to $W^{\pm}$ and top.

```
let width f =
  if !use_fudged_width then
    match f with
    | G Wp | G Wm | M (U 3) | M (U (−3)) | O Grino → Fudged
    | _ → !default_width
  else
    !default_width

let goldstone = function
  | G f →
      begin match f with
      | Wp → Some (O Phip, Coupling.Const 1)
      | Wm → Some (O Phim, Coupling.Const 1)
      | Z → Some (O Phi0, Coupling.Const 1)
      | _ → None
      end
  | _ → None

let conjugate = function
  | M f →
      M (begin match f with
      | L n → L (−n) | N n → N (−n)
      | U n → U (−n) | D n → D (−n)
      | SL n → SL (−n)
      end)
  | G f →
      G (begin match f with
      | Gl → Gl | Ga → Ga | Z → Z
      | Wp → Wm | Wm → Wp | Phino → Phino
      end)
  | O f →
      O (begin match f with
      | Phip → Phim | Phim → Phip | Phi0 → Phi0
      | H → H | Grino → Grino
      end)

let fermion = function
  | M f →
      begin match f with
```

```
        | L n → if n > 0 then 1 else -1
        | N n → if n > 0 then 1 else -1
        | U n → if n > 0 then 1 else -1
        | D n → if n > 0 then 1 else -1
        | SL _ → 0
        end
  | G f →
        begin match f with
        | Gl | Ga | Z | Wp | Wm → 0
        | Phino → 2
        end
  | O f →
        begin match f with
        | Grino → 2
        | _ → 0
        end

module Ch = Charges.QQ

let ( // ) = Algebra.Small_Rational.make

let generation' = function
  | 1 → [ 1//1; 0//1; 0//1]
  | 2 → [ 0//1; 1//1; 0//1]
  | 3 → [ 0//1; 0//1; 1//1]
  | -1 → [-1//1; 0//1; 0//1]
  | -2 → [ 0//1; -1//1; 0//1]
  | -3 → [ 0//1; 0//1; -1//1]
  | n → invalid_arg ("SM3.generation':␣" ^ string_of_int n)

let generation f =
  match f with
  | M (L n | N n | U n | D n | SL n) → generation' n
  | G _ | O _ → [0//1; 0//1; 0//1]

let charge = function
  | M f →
        begin match f with
        | L n → if n > 0 then -1//1 else 1//1
        | SL n → if n > 0 then -1//1 else 1//1
        | N n → 0//1
        | U n → if n > 0 then 2//3 else -2//3
        | D n → if n > 0 then -1//3 else 1//3
        end
  | G f →
        begin match f with
        | Gl | Ga | Z | Phino → 0//1
        | Wp → 1//1
        | Wm → -1//1
        end
  | O f →
        begin match f with
        | H | Phi0 | Grino → 0//1
```

```
      | Phip → 1//1
      | Phim → − 1//1
      end

let lepton = function
  | M f →
      begin match f with
      | L n | N n | SL n → if n > 0 then 1//1 else -1//1
      | U _ | D _ → 0//1
      end
  | G _ | O _ → 0//1

let baryon = function
  | M f →
      begin match f with
      | L _ | N _ | SL _ → 0//1
      | U n | D n → if n > 0 then 1//1 else -1//1
      end
  | G _ | O _ → 0//1

let charges f =
  [ charge f; lepton f; baryon f] @ generation f

type constant =
  | Unit | Pi | Alpha_QED | Sin2thw
  | Sinthw | Costhw | E | G_weak | Vev
  | Q_lepton | Q_up | Q_down | G_CC | G_CCQ of int×int
  | G_NC_neutrino | G_NC_lepton | G_NC_up | G_NC_down
  | I_Q_W | I_G_ZWW
  | G_WWWW | G_ZZWW | G_AZWW | G_AAWW
  | G_HWW | G_HHWW | G_HZZ | G_HHZZ
  | G_Htt | G_Hbb | G_Hcc | G_Htautau | G_H3 | G_H4
  | G_HGaZ | G_HGaGa | G_Hgg
  | G_strong | G_Grav
  | Mass of flavor | Width of flavor
```

Two integer counters for the QCD and EW order of the couplings.

```
type orders = int × int

let orders = function
  | _ → (0, 0)

let input_parameters =
  []

let derived_parameters =
  []

let g_over_2_costh =
  Quot (Neg (Atom G_weak), Prod [Const 2; Atom Costhw])

let nc_coupling c t3 q =
  (Real_Array c,
   [Prod [g_over_2_costh; Diff (t3, Prod [Const 2; q; Atom Sin2thw])];
    Prod [g_over_2_costh; t3]])
```

```
let half = Quot (Const 1, Const 2)

let derived_parameter_arrays =
  [ nc_coupling G_NC_neutrino half (Const 0);
    nc_coupling G_NC_lepton (Neg half) (Const (−1));
    nc_coupling G_NC_up half (Quot (Const 2, Const 3));
    nc_coupling G_NC_down (Neg half) (Quot (Const (−1), Const 3)) ]

let parameters () =
  { input = input_parameters;
    derived = derived_parameters;
    derived_arrays = derived_parameter_arrays }

module F = Modeltools.Fusions (struct
  type f = flavor
  type c = constant
  let compare = compare
  let conjugate = conjugate
end)

let mgm ((m1, g, m2), fbf, c) = ((M m1, G g, M m2), fbf, c)
let mom ((m1, o, m2), fbf, c) = ((M m1, O o, M m2), fbf, c)

let electromagnetic_currents n =
  List.map mgm
    [ ((L (−n), Ga, L n), FBF (1, Psibar, V, Psi), Q_lepton);
      ((U (−n), Ga, U n), FBF (1, Psibar, V, Psi), Q_up);
      ((D (−n), Ga, D n), FBF (1, Psibar, V, Psi), Q_down) ]

let neutral_currents n =
  List.map mgm
    [ ((L (−n), Z, L n), FBF (1, Psibar, VA, Psi), G_NC_lepton);
      ((N (−n), Z, N n), FBF (1, Psibar, VA, Psi), G_NC_neutrino);
      ((U (−n), Z, U n), FBF (1, Psibar, VA, Psi), G_NC_up);
      ((D (−n), Z, D n), FBF (1, Psibar, VA, Psi), G_NC_down) ]

let charged_currents n =
  List.map mgm
    [ ((L (−n), Wm, N n), FBF (1, Psibar, VL, Psi), G_CC);
      ((N (−n), Wp, L n), FBF (1, Psibar, VL, Psi), G_CC);
      ((D (−n), Wm, U n), FBF (1, Psibar, VL, Psi), G_CC);
      ((U (−n), Wp, D n), FBF (1, Psibar, VL, Psi), G_CC) ]

let yukawa =
  List.map mom
    [ ((U (−3), H, U 3), FBF (1, Psibar, S, Psi), G_Htt);
      ((D (−3), H, D 3), FBF (1, Psibar, S, Psi), G_Hbb);
      ((U (−2), H, U 2), FBF (1, Psibar, S, Psi), G_Hcc);
      ((L (−3), H, L 3), FBF (1, Psibar, S, Psi), G_Htautau) ]

let tgc ((g1, g2, g3), t, c) = ((G g1, G g2, G g3), t, c)

let standard_triple_gauge =
  List.map tgc
    [ ((Ga, Wm, Wp), Gauge_Gauge_Gauge 1, I_Q_W);
```

```
      ((Z, Wm, Wp), Gauge_Gauge_Gauge 1, I_G_ZWW) ]

let triple_gauge =
    standard_triple_gauge

let qgc ((g1, g2, g3, g4), t, c) = ((G g1, G g2, G g3, G g4), t, c)

let gauge4 = Vector4 [(2, C_13_42); (−1, C_12_34); (−1, C_14_23)]
let minus_gauge4 = Vector4 [(−2, C_13_42); (1, C_12_34); (1, C_14_23)]
let standard_quartic_gauge =
  List.map qgc
    [ (Wm, Wp, Wm, Wp), gauge4, G_WWWW;
      (Wm, Z, Wp, Z), minus_gauge4, G_ZZWW;
      (Wm, Z, Wp, Ga), minus_gauge4, G_AZWW;
      (Wm, Ga, Wp, Ga), minus_gauge4, G_AAWW ]

let quartic_gauge =
  standard_quartic_gauge

let standard_gauge_higgs =
  [ ((O H, G Wp, G Wm), Scalar_Vector_Vector 1, G_HWW);
    ((O H, G Z, G Z), Scalar_Vector_Vector 1, G_HZZ) ]

let standard_gauge_higgs4 =
  [ (O H, O H, G Wp, G Wm), Scalar2_Vector2 1, G_HHWW;
    (O H, O H, G Z, G Z), Scalar2_Vector2 1, G_HHZZ ]

let standard_higgs =
  [ (O H, O H, O H), Scalar_Scalar_Scalar 1, G_H3 ]

let standard_higgs4 =
  [ (O H, O H, O H, O H), Scalar4 1, G_H4 ]

let anomalous_gauge_higgs =
  []

let anomalous_gauge_higgs4 =
  []

let anomalous_higgs =
  []

let anomaly_higgs =
  [ (O H, G Ga, G Ga), Dim5_Scalar_Gauge2 1, G_HGaGa;
    (O H, G Ga, G Z), Dim5_Scalar_Gauge2 1, G_HGaZ;
    (O H, G Gl, G Gl), Dim5_Scalar_Gauge2 1, G_Hgg ]

let gravitino_coup n =
  [ (O Grino, M (SL (−n)), M (L n)), GBG (1, Gravbar, POT, Psi), G_Grav;
    (M (L (−n)), M (SL n), O Grino), GBG (1, Psibar, POT, Grav), G_Grav]

let gravitino_gauge =
  [ (O Grino, G Ga, G Phino), GBG (1, Gravbar, V, Chi), G_Grav ]

let anomalous_higgs4 =
  []

let gauge_higgs =
```

```
    standard_gauge_higgs

let gauge_higgs4 =
    standard_gauge_higgs4

let higgs =
    standard_higgs

let higgs4 =
    standard_higgs4

let goldstone_vertices =
  [ ((O Phi0, G Wm, G Wp), Scalar_Vector_Vector 1, I_G_ZWW);
    ((O Phip, G Ga, G Wm), Scalar_Vector_Vector 1, I_Q_W);
    ((O Phip, G Z, G Wm), Scalar_Vector_Vector 1, I_G_ZWW);
    ((O Phim, G Wp, G Ga), Scalar_Vector_Vector 1, I_Q_W);
    ((O Phim, G Wp, G Z), Scalar_Vector_Vector 1, I_G_ZWW) ]

let vertices3 =
  (ThoList.flatmap electromagnetic_currents [1;2;3] @
   ThoList.flatmap neutral_currents [1;2;3] @
   ThoList.flatmap charged_currents [1;2;3] @
   ThoList.flatmap gravitino_coup [1;2;3] @
   gravitino_gauge @
   yukawa @ triple_gauge @
   gauge_higgs @ higgs @ anomaly_higgs
   @ goldstone_vertices)

let vertices4 =
  quartic_gauge @ gauge_higgs4 @ higgs4

let vertices () = (vertices3, vertices4, [])
```

For efficiency, make sure that $F.of\_vertices\ vertices$ is evaluated only once.

```
let table = F.of_vertices (vertices ())
let fuse2 = F.fuse2 table
let fuse3 = F.fuse3 table
let fuse = F.fuse table
let max_degree () = 4

let flavor_of_string = function
  | "e-" → M (L 1) | "e+" → M (L (−1))
  | "mu-" → M (L 2) | "mu+" → M (L (−2))
  | "tau-" → M (L 3) | "tau+" → M (L (−3))
  | "se-" → M (SL 1) | "se+" → M (SL (−1))
  | "smu-" → M (SL 2) | "smu+" → M (SL (−2))
  | "stau-" → M (SL 3) | "stau+" → M (SL (−3))
  | "nue" → M (N 1) | "nuebar" → M (N (−1))
  | "numu" → M (N 2) | "numubar" → M (N (−2))
  | "nutau" → M (N 3) | "nutaubar" → M (N (−3))
  | "u" → M (U 1) | "ubar" → M (U (−1))
  | "c" → M (U 2) | "cbar" → M (U (−2))
  | "t" → M (U 3) | "tbar" → M (U (−3))
  | "d" → M (D 1) | "dbar" → M (D (−1))
```

```
    | "s" → M (D 2) | "sbar" → M (D (−2))
    | "b" → M (D 3) | "bbar" → M (D (−3))
    | "g" | "gl" → G Gl
    | "A" → G Ga | "Z" | "Z0" → G Z
    | "W+" → G Wp | "W-" → G Wm
    | "H" → O H
    | "GG" → O Grino
    | "phino" | "Phino" → G Phino
    | _ → invalid_arg "Modellib_BSM.GravTest.flavor_of_string"

let flavor_to_string = function
  | M f →
      begin match f with
      | L 1 → "e-" | L (−1) → "e+"
      | L 2 → "mu-" | L (−2) → "mu+"
      | L 3 → "tau-" | L (−3) → "tau+"
      | L _ → invalid_arg
            "Modellib_BSM.GravTest.flavor_to_string: invalid lepton"
      | SL 1 → "se-" | SL (−1) → "se+"
      | SL 2 → "smu-" | SL (−2) → "smu+"
      | SL 3 → "stau-" | SL (−3) → "stau+"
      | SL _ → invalid_arg
            "Modellib_BSM.GravTest.flavor_to_string: invalid slepton"
      | N 1 → "nue" | N (−1) → "nuebar"
      | N 2 → "numu" | N (−2) → "numubar"
      | N 3 → "nutau" | N (−3) → "nutaubar"
      | N _ → invalid_arg
            "Modellib_BSM.GravTest.flavor_to_string: invalid neutrino"
      | U 1 → "u" | U (−1) → "ubar"
      | U 2 → "c" | U (−2) → "cbar"
      | U 3 → "t" | U (−3) → "tbar"
      | U _ → invalid_arg
            "Modellib_BSM.SM.flavor_to_string: invalid up type quark"
      | D 1 → "d" | D (−1) → "dbar"
      | D 2 → "s" | D (−2) → "sbar"
      | D 3 → "b" | D (−3) → "bbar"
      | D _ → invalid_arg
            "Modellib_BSM.GravTest.flavor_to_string: invalid down type quark"
      end
  | G f →
      begin match f with
      | Gl → "g"
      | Ga → "A" | Z → "Z"
      | Wp → "W+" | Wm → "W-"
      | Phino → "phino"
      end
  | O f →
      begin match f with
      | Phip → "phi+" | Phim → "phi-" | Phi0 → "phi0"
      | H → "H" | Grino → "GG"
```

```
        end

  let flavor_to_TeX = function
    | M f →
        begin match f with
        | L 1 → "e^-" | L (−1) → "e^+"
        | L 2 → "\\mu^-" | L (−2) → "\\mu^+"
        | L 3 → "\\tau^-" | L (−3) → "\\tau^+"
        | L _ → invalid_arg
              "Modellib_BSM.GravTest.flavor_to_TeX: invalid lepton"
        | SL 1 → "\\tilde{e}^-" | SL (−1) → "\\tilde{e}^+"
        | SL 2 → "\\tilde{\\mu}^-" | SL (−2) → "\\tilde{\\mu}^+"
        | SL 3 → "\\tilde{\\tau}^-" | SL (−3) → "\\tilde{\\tau}^+"
        | SL _ → invalid_arg
              "Modellib_BSM.GravTest.flavor_to_TeX: invalid slepton"
        | N 1 → "\\nu_e" | N (−1) → "\\bar{\\nu}_e"
        | N 2 → "\\nu_\\mu" | N (−2) → "\\bar{\\nu}_\\mu"
        | N 3 → "\\nu_\\tau" | N (−3) → "\\bar{\\nu}_\\tau"
        | N _ → invalid_arg
              "Modellib_BSM.GravTest.flavor_to_TeX: invalid neutrino"
        | U 1 → "u" | U (−1) → "\\bar{u}"
        | U 2 → "c" | U (−2) → "\\bar{c}"
        | U 3 → "t" | U (−3) → "\\bar{t}"
        | U _ → invalid_arg
              "Modellib_BSM.SM.flavor_to_TeX: invalid up type quark"
        | D 1 → "d" | D (−1) → "\\bar{d}"
        | D 2 → "s" | D (−2) → "\\bar{s}"
        | D 3 → "b" | D (−3) → "\\bar{b}"
        | D _ → invalid_arg
              "Modellib_BSM.GravTest.flavor_to_TeX: invalid down type quark"
        end
    | G f →
        begin match f with
        | Gl → "g"
        | Ga → "\\gamma" | Z → "Z"
        | Wp → "W^+" | Wm → "W^-"
        | Phino → "\\tilde{\\phi}"
        end
    | O f →
        begin match f with
        | Phip → "\\phi^+" | Phim → "\\phi^-" | Phi0 →
"\\phi^0"
        | H → "H" | Grino → "\\tilde{G}"
        end

  let flavor_symbol = function
    | M f →
        begin match f with
        | L n when n > 0 → "l" ^ string_of_int n
        | L n → "l" ^ string_of_int (abs n) ^ "b"
        | SL n when n > 0 → "sl" ^ string_of_int n
```

```
        | SL n → "sl" ^ string_of_int (abs n) ^ "b"
        | N n when n > 0 → "n" ^ string_of_int n
        | N n → "n" ^ string_of_int (abs n) ^ "b"
        | U n when n > 0 → "u" ^ string_of_int n
        | U n → "u" ^ string_of_int (abs n) ^ "b"
        | D n when n > 0 → "d" ^ string_of_int n
        | D n → "d" ^ string_of_int (abs n) ^ "b"
        end
  | G f →
        begin match f with
        | Gl → "gl"
        | Ga → "a" | Z → "z"
        | Wp → "wp" | Wm → "wm"
        | Phino → "phino"
        end
  | O f →
        begin match f with
        | Phip → "pp" | Phim → "pm" | Phi0 → "p0"
        | H → "h" | Grino → "gv"
        end

let pdg = function
  | M f →
        begin match f with
        | L n when n > 0 → 9 + 2 × n
        | L n → − 9 + 2 × n
        | SL n when n > 0 → 39 + 2 × n
        | SL n → − 39 + 2 × n
        | N n when n > 0 → 10 + 2 × n
        | N n → − 10 + 2 × n
        | U n when n > 0 → 2 × n
        | U n → 2 × n
        | D n when n > 0 → − 1 + 2 × n
        | D n → 1 + 2 × n
        end
  | G f →
        begin match f with
        | Gl → 21
        | Ga → 22 | Z → 23
        | Wp → 24 | Wm → (−24)
        | Phino → 46
        end
  | O f →
        begin match f with
        | Phip | Phim → 27 | Phi0 → 26
        | H → 25 | Grino → 39
        end

let mass_symbol f =
  "mass(" ^ string_of_int (abs (pdg f)) ^ ")"
```

```
let width_symbol f =
  "width(" ^ string_of_int (abs (pdg f)) ^ ")"

let constant_symbol = function
  | Unit → "unit" | Pi → "PI"
  | Alpha_QED → "alpha" | E → "e" | G_weak → "g" | Vev →
"vev"
  | Sin2thw → "sin2thw" | Sinthw → "sinthw" | Costhw →
"costhw"
  | Q_lepton → "qlep" | Q_up → "qup" | Q_down → "qdwn"
  | G_NC_lepton → "gnclep" | G_NC_neutrino → "gncneu"
  | G_NC_up → "gncup" | G_NC_down → "gncdwn"
  | G_CC → "gcc"
  | G_CCQ (n1, n2) → "gccq" ^ string_of_int n1 ^ string_of_int n2
  | I_Q_W → "iqw" | I_G_ZWW → "igzww"
  | G_WWWW → "gw4" | G_ZZWW → "gzzww"
  | G_AZWW → "gazww" | G_AAWW → "gaaww"
  | G_HWW → "ghww" | G_HZZ → "ghzz"
  | G_HHWW → "ghhww" | G_HHZZ → "ghhzz"
  | G_Htt → "ghtt" | G_Hbb → "ghbb"
  | G_Htautau → "ghtautau" | G_Hcc → "ghcc"
  | G_HGaZ → "ghgaz" | G_HGaGa → "ghgaga" | G_Hgg →
"ghgg"
  | G_H3 → "gh3" | G_H4 → "gh4"
  | G_strong → "gs" | G_Grav → "ggrav"
  | Mass f → "mass" ^ flavor_symbol f
  | Width f → "width" ^ flavor_symbol f

end

module Template (Flags : BSM_flags) =
 struct
  let rcs = RCS.rename rcs_file "Modellib_BSM.Template"
     [ "Template for user-defined BSM model"]

  open Coupling

  let default_width = ref Timelike
  let use_fudged_width = ref false

  let options = Options.create
   [ "constant_width", Arg.Unit (fun () → default_width := Constant),
     "use constant width (also in t-channel)";
     "fudged_width", Arg.Set use_fudged_width,
     "use fudge factor for charge particle width";
     "custom_width", Arg.String (fun f → default_width := Custom f),
     "use custom width";
     "cancel_widths", Arg.Unit (fun () → default_width := Vanishing),
     "use vanishing width"]

  type matter_field = L of int | N of int | U of int | D of int
  type gauge_boson = Ga | Wp | Wm | Z | Gl
  type other = Phip | Phim | Phi0 | H
  type flavor = M of matter_field | G of gauge_boson | O of other
```

```
let matter_field f = M f
let gauge_boson f = G f
let other f = O f

type field =
  | Matter of matter_field
  | Gauge of gauge_boson
  | Other of other

let field = function
  | M f → Matter f
  | G f → Gauge f
  | O f → Other f

type gauge = unit

let gauge_symbol () =
  failwith "Modellib_BSM.Template.gauge_symbol: internal error"

let family n = List.map matter_field [ L n; N n; U n; D n ]

let external_flavors () =
  [ "1st Generation", ThoList.flatmap family [1; − 1];
    "2nd Generation", ThoList.flatmap family [2; − 2];
    "3rd Generation", ThoList.flatmap family [3; − 3];
    "Gauge Bosons", List.map gauge_boson [Ga; Z; Wp; Wm; Gl];
    "Higgs", List.map other [H];
    "Goldstone Bosons", List.map other [Phip; Phim; Phi0] ]

let flavors () = ThoList.flatmap snd (external_flavors ())

let spinor n =
  if n ≥ 0 then
    Spinor
  else
    ConjSpinor

let lorentz = function
  | M f →
      begin match f with
      | L n → spinor n | N n → spinor n
      | U n → spinor n | D n → spinor n
      end
  | G f →
      begin match f with
      | Ga | Gl → Vector
      | Wp | Wm | Z → Massive_Vector
      end
  | O f → Scalar

let color = function
  | M (U n) → Color.SUN (if n > 0 then 3 else -3)
  | M (D n) → Color.SUN (if n > 0 then 3 else -3)
  | G Gl → Color.AdjSUN 3
  | _ → Color.Singlet
```

```
let prop_spinor n =
  if n ≥ 0 then
    Prop_Spinor
  else
    Prop_ConjSpinor

let propagator = function
  | M f →
      begin match f with
      | L n → prop_spinor n | N n → prop_spinor n
      | U n → prop_spinor n | D n → prop_spinor n
      end
  | G f →
      begin match f with
      | Ga | Gl → Prop_Feynman
      | Wp | Wm | Z → Prop_Unitarity
      end
  | O f →
      begin match f with
      | Phip | Phim | Phi0 → Only_Insertion
      | H → Prop_Scalar
      end
```

Optionally, ask for the fudge factor treatment for the widths of charged particles. Currently, this only applies to $W^{\pm}$ and top.

```
let width f =
  if !use_fudged_width then
    match f with
    | G Wp | G Wm | M (U 3) | M (U (−3)) → Fudged
    | _ → !default_width
  else
    !default_width

let goldstone = function
  | G f →
      begin match f with
      | Wp → Some (O Phip, Coupling.Const 1)
      | Wm → Some (O Phim, Coupling.Const 1)
      | Z → Some (O Phi0, Coupling.Const 1)
      | _ → None
      end
  | _ → None

let conjugate = function
  | M f →
      M (begin match f with
      | L n → L (−n) | N n → N (−n)
      | U n → U (−n) | D n → D (−n)
      end)
  | G f →
      G (begin match f with
      | Gl → Gl | Ga → Ga | Z → Z
```

```
        | Wp → Wm | Wm → Wp
        end)
    | O f →
        O (begin match f with
        | Phip → Phim | Phim → Phip | Phi0 → Phi0
        | H → H
        end)

let fermion = function
    | M f →
        begin match f with
        | L n → if n > 0 then 1 else -1
        | N n → if n > 0 then 1 else -1
        | U n → if n > 0 then 1 else -1
        | D n → if n > 0 then 1 else -1
        end
    | G f →
        begin match f with
        | Gl | Ga | Z | Wp | Wm → 0
        end
    | O _ → 0

module Ch = Charges.QQ

let ( // ) = Algebra.Small_Rational.make

let generation' = function
    | 1 → [ 1//1; 0//1; 0//1]
    | 2 → [ 0//1; 1//1; 0//1]
    | 3 → [ 0//1; 0//1; 1//1]
    | -1 → [-1//1; 0//1; 0//1]
    | -2 → [ 0//1; -1//1; 0//1]
    | -3 → [ 0//1; 0//1; -1//1]
    | n → invalid_arg ("Template.generation':␣" ^ string_of_int n)

let generation f =
    match f with
    | M (L n | N n | U n | D n) → generation' n
    | G _ | O _ → [0//1; 0//1; 0//1]

let charge = function
    | M f →
        begin match f with
        | L n → if n > 0 then -1//1 else 1//1
        | N n → 0//1
        | U n → if n > 0 then 2//3 else -2//3
        | D n → if n > 0 then -1//3 else 1//3
        end
    | G f →
        begin match f with
        | Gl | Ga | Z → 0//1
        | Wp → 1//1
        | Wm → -1//1
```

```
        end
    | O f →
        begin match f with
        | H | Phi0 → 0//1
        | Phip → 1//1
        | Phim → − 1//1
        end

  let lepton = function
    | M f →
        begin match f with
        | L n | N n → if n > 0 then 1//1 else -1//1
        | U _ | D _ → 0//1
        end
    | G _ | O _ → 0//1

  let baryon = function
    | M f →
        begin match f with
        | L _ | N _ → 0//1
        | U n | D n → if n > 0 then 1//1 else -1//1
        end
    | G _ | O _ → 0//1

  let charges f =
    [ charge f; lepton f; baryon f] @ generation f

  type constant =
    | Unit | Pi | Alpha_QED | Sin2thw
    | Sinthw | Costhw | E | G_weak | Vev
    | Q_lepton | Q_up | Q_down | G_CC
    | G_NC_neutrino | G_NC_lepton | G_NC_up | G_NC_down
    | I_Q_W | I_G_ZWW
    | G_WWWW | G_ZZWW | G_AZWW | G_AAWW
    | G_HWW | G_HHWW | G_HZZ | G_HHZZ
    | G_Htt | G_Hbb | G_Hcc | G_Hmm | G_Htautau | G_H3 |
G_H4
    | G_HGaZ | G_HGaGa | G_Hgg
    | Gs | I_Gs | G2
    | Mass of flavor | Width of flavor
```

Two integer counters for the QCD and EW order of the couplings.

```
  type orders = int × int

  let orders = function
    | _ → (0, 0)

  let input_parameters = []

  let derived_parameters = []

  let g_over_2_costh =
    Quot (Neg (Atom G_weak), Prod [Const 2; Atom Costhw])
```

```
let nc_coupling c t3 q =
  (Real_Array c,
   [Prod [g_over_2_costh; Diff (t3, Prod [Const 2; q; Atom Sin2thw])];
    Prod [g_over_2_costh; t3]])

let half = Quot (Const 1, Const 2)

let derived_parameter_arrays =
  [ nc_coupling G_NC_neutrino half (Const 0);
    nc_coupling G_NC_lepton (Neg half) (Const (-1));
    nc_coupling G_NC_up half (Quot (Const 2, Const 3));
    nc_coupling G_NC_down (Neg half) (Quot (Const (-1), Const 3)) ]

let parameters () =
  { input = input_parameters;
    derived = derived_parameters;
    derived_arrays = derived_parameter_arrays }

module F = Modeltools.Fusions (struct
  type f = flavor
  type c = constant
  let compare = compare
  let conjugate = conjugate
end)

let mgm ((m1, g, m2), fbf, c) = ((M m1, G g, M m2), fbf, c)

let electromagnetic_currents n =
  List.map mgm
    [ ((L (-n), Ga, L n), FBF (1, Psibar, V, Psi), Q_lepton);
      ((U (-n), Ga, U n), FBF (1, Psibar, V, Psi), Q_up);
      ((D (-n), Ga, D n), FBF (1, Psibar, V, Psi), Q_down) ]

let color_currents n =
    List.map mgm
      [ ((U (-n), Gl, U n), FBF ((-1), Psibar, V, Psi), Gs);
        ((D (-n), Gl, D n), FBF ((-1), Psibar, V, Psi), Gs) ]

let neutral_currents n =
  List.map mgm
    [ ((L (-n), Z, L n), FBF (1, Psibar, VA, Psi), G_NC_lepton);
      ((N (-n), Z, N n), FBF (1, Psibar, VA, Psi), G_NC_neutrino);
      ((U (-n), Z, U n), FBF (1, Psibar, VA, Psi), G_NC_up);
      ((D (-n), Z, D n), FBF (1, Psibar, VA, Psi), G_NC_down) ]

let charged_currents n =
  List.map mgm
    [ ((L (-n), Wm, N n), FBF (1, Psibar, VL, Psi), G_CC);
      ((N (-n), Wp, L n), FBF (1, Psibar, VL, Psi), G_CC);
      ((D (-n), Wm, U n), FBF (1, Psibar, VL, Psi), G_CC);
      ((U (-n), Wp, D n), FBF (1, Psibar, VL, Psi), G_CC) ]

let yukawa =
  [ ((M (U (-3)), O H, M (U 3)), FBF (1, Psibar, S, Psi), G_Htt);
    ((M (D (-3)), O H, M (D 3)), FBF (1, Psibar, S, Psi), G_Hbb);
```

```
        ((M (U (-2)), O H, M (U 2)), FBF (1, Psibar, S, Psi), G_Hcc);
        ((M (L (-2)), O H, M (L 2)), FBF (1, Psibar, S, Psi), G_Hmm);
        ((M (L (-3)), O H, M (L 3)), FBF (1, Psibar, S, Psi), G_Htautau) ]

  let tgc ((g1, g2, g3), t, c) = ((G g1, G g2, G g3), t, c)

  let triple_gauge =
    List.map tgc
      [ ((Ga, Wm, Wp), Gauge_Gauge_Gauge 1, I_Q_W);
        ((Z, Wm, Wp), Gauge_Gauge_Gauge 1, I_G_ZWW);
        ((Gl, Gl, Gl), Gauge_Gauge_Gauge 1, I_Gs) ]

  let qgc ((g1, g2, g3, g4), t, c) = ((G g1, G g2, G g3, G g4), t, c)

  let gauge4 = Vector4 [(2, C_13_42); (-1, C_12_34); (-1, C_14_23)]
  let minus_gauge4 = Vector4 [(-2, C_13_42); (1, C_12_34); (1, C_14_23)]
  let quartic_gauge =
    List.map qgc
      [ (Wm, Wp, Wm, Wp), gauge4, G_WWWW;
        (Wm, Z, Wp, Z), minus_gauge4, G_ZZWW;
        (Wm, Z, Wp, Ga), minus_gauge4, G_AZWW;
        (Wm, Ga, Wp, Ga), minus_gauge4, G_AAWW;
        (Gl, Gl, Gl, Gl), gauge4, G2]

  let gauge_higgs =
    [ ((O H, G Wp, G Wm), Scalar_Vector_Vector 1, G_HWW);
      ((O H, G Z, G Z), Scalar_Vector_Vector 1, G_HZZ) ]

  let gauge_higgs4 =
    [ (O H, O H, G Wp, G Wm), Scalar2_Vector2 1, G_HHWW;
      (O H, O H, G Z, G Z), Scalar2_Vector2 1, G_HHZZ ]

  let higgs =
    [ (O H, O H, O H), Scalar_Scalar_Scalar 1, G_H3 ]

  let higgs4 =
    [ (O H, O H, O H, O H), Scalar4 1, G_H4 ]

  let anomaly_higgs =
    []
(* (O H, G Ga, G Ga), Dim5_Scalar_Gauge2 1, G_HGaGa; (O H, G Ga, G Z), Dim5_Scalar_Gauge2 1,
*)

  let goldstone_vertices =
    [ ((O Phi0, G Wm, G Wp), Scalar_Vector_Vector 1, I_G_ZWW);
      ((O Phip, G Ga, G Wm), Scalar_Vector_Vector 1, I_Q_W);
      ((O Phip, G Z, G Wm), Scalar_Vector_Vector 1, I_G_ZWW);
      ((O Phim, G Wp, G Ga), Scalar_Vector_Vector 1, I_Q_W);
      ((O Phim, G Wp, G Z), Scalar_Vector_Vector 1, I_G_ZWW) ]

  let vertices3 =
    (ThoList.flatmap electromagnetic_currents [1;2;3] @
     ThoList.flatmap color_currents [1;2;3] @
     ThoList.flatmap neutral_currents [1;2;3] @
     ThoList.flatmap charged_currents [1;2;3] @
     yukawa @ triple_gauge @ gauge_higgs @ higgs @
```

```
    anomaly_higgs @ goldstone_vertices)

let vertices4 =
  quartic_gauge @ gauge_higgs4 @ higgs4

let vertices () = (vertices3, vertices4, [])

let table = F.of_vertices (vertices ())
let fuse2 = F.fuse2 table
let fuse3 = F.fuse3 table
let fuse = F.fuse table
let max_degree () = 4

let flavor_of_string = function
  | "e-" → M (L 1) | "e+" → M (L (−1))
  | "mu-" → M (L 2) | "mu+" → M (L (−2))
  | "tau-" → M (L 3) | "tau+" → M (L (−3))
  | "nue" → M (N 1) | "nuebar" → M (N (−1))
  | "numu" → M (N 2) | "numubar" → M (N (−2))
  | "nutau" → M (N 3) | "nutaubar" → M (N (−3))
  | "u" → M (U 1) | "ubar" → M (U (−1))
  | "c" → M (U 2) | "cbar" → M (U (−2))
  | "t" → M (U 3) | "tbar" → M (U (−3))
  | "d" → M (D 1) | "dbar" → M (D (−1))
  | "s" → M (D 2) | "sbar" → M (D (−2))
  | "b" → M (D 3) | "bbar" → M (D (−3))
  | "g" | "gl" → G Gl
  | "A" → G Ga | "Z" | "Z0" → G Z
  | "W+" → G Wp | "W-" → G Wm
  | "H" → O H
  | _ → invalid_arg "Modellib_BSM.Template.flavor_of_string"

let flavor_to_string = function
  | M f →
      begin match f with
      | L 1 → "e-" | L (−1) → "e+"
      | L 2 → "mu-" | L (−2) → "mu+"
      | L 3 → "tau-" | L (−3) → "tau+"
      | L _ → invalid_arg
            "Modellib_BSM.Template.flavor_to_string:␣invalid␣lepton"
      | N 1 → "nue" | N (−1) → "nuebar"
      | N 2 → "numu" | N (−2) → "numubar"
      | N 3 → "nutau" | N (−3) → "nutaubar"
      | N _ → invalid_arg
            "Modellib_BSM.Template.flavor_to_string:␣invalid␣neutrino"
      | U 1 → "u" | U (−1) → "ubar"
      | U 2 → "c" | U (−2) → "cbar"
      | U 3 → "t" | U (−3) → "tbar"
      | U _ → invalid_arg
            "Modellib_BSM.Template.flavor_to_string:␣invalid␣up␣type␣quark"
      | D 1 → "d" | D (−1) → "dbar"
      | D 2 → "s" | D (−2) → "sbar"
      | D 3 → "b" | D (−3) → "bbar"
```

```
        | D _ → invalid_arg
            "Modellib_BSM.Template.flavor_to_string: invalid down type quark"
        end
    | G f →
        begin match f with
        | Gl → "g"
        | Ga → "A" | Z → "Z"
        | Wp → "W+" | Wm → "W-"
        end
    | O f →
        begin match f with
        | Phip → "phi+" | Phim → "phi-" | Phi0 → "phi0"
        | H → "H"
        end

  let flavor_to_TeX = function
    | M f →
        begin match f with
        | L 1 → "e^-" | L (-1) → "e^+"
        | L 2 → "\\mu^-" | L (-2) → "\\mu^+"
        | L 3 → "\\tau^-" | L (-3) → "\\tau^+"
        | L _ → invalid_arg
            "Modellib_BSM.Template.flavor_to_TeX: invalid lepton"
        | N 1 → "\\nu_e" | N (-1) → "\\bar{\\nu}_e"
        | N 2 → "\\nu_\\mu" | N (-2) → "\\bar{\\nu}_\\mu"
        | N 3 → "\\nu_\\tau" | N (-3) → "\\bar{\\nu}_\\tau"
        | N _ → invalid_arg
            "Modellib_BSM.Template.flavor_to_TeX: invalid neutrino"
        | U 1 → "u" | U (-1) → "\\bar{u}"
        | U 2 → "c" | U (-2) → "\\bar{c}"
        | U 3 → "t" | U (-3) → "\\bar{t}"
        | U _ → invalid_arg
            "Modellib_BSM.Template.flavor_to_TeX: invalid up type quark"
        | D 1 → "d" | D (-1) → "\\bar{d}"
        | D 2 → "s" | D (-2) → "\\bar{s}"
        | D 3 → "b" | D (-3) → "\\bar{b}"
        | D _ → invalid_arg
            "Modellib_BSM.Template.flavor_to_TeX: invalid down type quark"
        end
    | G f →
        begin match f with
        | Gl → "g"
        | Ga → "\\gamma" | Z → "Z"
        | Wp → "W^+" | Wm → "W^-"
        end
    | O f →
        begin match f with
        | Phip → "\\phi^+" | Phim → "\\phi^-" | Phi0 →
"\\phi^0"
        | H → "H"
```

```
    end
let flavor_symbol = function
  | M f ->
      begin match f with
      | L n when n > 0 -> "l" ^ string_of_int n
      | L n -> "l" ^ string_of_int (abs n) ^ "b"
      | N n when n > 0 -> "n" ^ string_of_int n
      | N n -> "n" ^ string_of_int (abs n) ^ "b"
      | U n when n > 0 -> "u" ^ string_of_int n
      | U n -> "u" ^ string_of_int (abs n) ^ "b"
      | D n when n > 0 -> "d" ^ string_of_int n
      | D n -> "d" ^ string_of_int (abs n) ^ "b"
      end
  | G f ->
      begin match f with
      | Gl -> "gl"
      | Ga -> "a" | Z -> "z"
      | Wp -> "wp" | Wm -> "wm"
      end
  | O f ->
      begin match f with
      | Phip -> "pp" | Phim -> "pm" | Phi0 -> "p0"
      | H -> "h"
      end
let pdg = function
  | M f ->
      begin match f with
      | L n when n > 0 -> 9 + 2 × n
      | L n -> - 9 + 2 × n
      | N n when n > 0 -> 10 + 2 × n
      | N n -> - 10 + 2 × n
      | U n when n > 0 -> 2 × n
      | U n -> 2 × n
      | D n when n > 0 -> - 1 + 2 × n
      | D n -> 1 + 2 × n
      end
  | G f ->
      begin match f with
      | Gl -> 21
      | Ga -> 22 | Z -> 23
      | Wp -> 24 | Wm -> (-24)
      end
  | O f ->
      begin match f with
      | Phip | Phim -> 27 | Phi0 -> 26
      | H -> 25
      end
let mass_symbol f =
  "mass(" ^ string_of_int (abs (pdg f)) ^ ")"
```

```
let width_symbol f =
  "width(" ^ string_of_int (abs (pdg f)) ^ ")"

let constant_symbol = function
  | Unit → "unit" | Pi → "PI"
  | Alpha_QED → "alpha" | E → "e" | G_weak → "g" | Vev →
"vev"
  | Sin2thw → "sin2thw" | Sinthw → "sinthw" | Costhw →
"costhw"
  | Q_lepton → "qlep" | Q_up → "qup" | Q_down → "qdwn"
  | G_NC_lepton → "gnclep" | G_NC_neutrino → "gncneu"
  | G_NC_up → "gncup" | G_NC_down → "gncdwn"
  | G_CC → "gcc"
  | I_Q_W → "iqw" | I_G_ZWW → "igzww"
  | G_WWWW → "gw4" | G_ZZWW → "gzzww"
  | G_AZWW → "gazww" | G_AAWW → "gaaww"
  | G_HWW → "ghww" | G_HZZ → "ghzz"
  | G_HHWW → "ghhww" | G_HHZZ → "ghhzz"
  | G_Htt → "ghtt" | G_Hbb → "ghbb"
  | G_Htautau → "ghtautau" | G_Hcc → "ghcc" | G_Hmm →
"ghmm"
  | G_HGaZ → "ghgaz" | G_HGaGa → "ghgaga" | G_Hgg →
"ghgg"
  | G_H3 → "gh3" | G_H4 → "gh4"
  | Gs → "gs" | I_Gs → "igs" | G2 → "gs**2"
  | Mass f → "mass" ^ flavor_symbol f
  | Width f → "width" ^ flavor_symbol f

end

module HSExt (Flags : BSM_flags) =
 struct
  let rcs = RCS.rename rcs_file "Modellib_BSM.HSExt"
     [ "Higgs␣Singlet␣Extension␣of␣the␣SM"]

  open Coupling

  let default_width = ref Timelike
  let use_fudged_width = ref false

  let options = Options.create
    [ "constant_width", Arg.Unit (fun () → default_width := Constant),
      "use␣constant␣width␣(also␣in␣t-channel)";
      "fudged_width", Arg.Set use_fudged_width,
      "use␣fudge␣factor␣for␣charge␣particle␣width";
      "custom_width", Arg.String (fun f → default_width := Custom f),
      "use␣custom␣width";
      "cancel_widths", Arg.Unit (fun () → default_width := Vanishing),
      "use␣vanishing␣width"]

  type matter_field = L of int | N of int | U of int | D of int
  type gauge_boson = Ga | Wp | Wm | Z | Gl
  type other = Phip | Phim | Phi0 | H | S
  type flavor = M of matter_field | G of gauge_boson | O of other
```

```
let matter_field f = M f
let gauge_boson f = G f
let other f = O f

type field =
  | Matter of matter_field
  | Gauge of gauge_boson
  | Other of other

let field = function
  | M f → Matter f
  | G f → Gauge f
  | O f → Other f

type gauge = unit

let gauge_symbol () =
  failwith "Modellib_BSM.HSExt.gauge_symbol: internal error"

let family n = List.map matter_field [ L n; N n; U n; D n ]

let external_flavors () =
  [ "1st Generation", ThoList.flatmap family [1; −1];
    "2nd Generation", ThoList.flatmap family [2; −2];
    "3rd Generation", ThoList.flatmap family [3; −3];
    "Gauge Bosons", List.map gauge_boson [Ga; Z; Wp; Wm; Gl];
    "Higgs", List.map other [H; S];
    "Goldstone Bosons", List.map other [Phip; Phim; Phi0] ]

let flavors () = ThoList.flatmap snd (external_flavors ())

let spinor n =
  if n ≥ 0 then
    Spinor
  else
    ConjSpinor

let lorentz = function
  | M f →
      begin match f with
      | L n → spinor n | N n → spinor n
      | U n → spinor n | D n → spinor n
      end
  | G f →
      begin match f with
      | Ga | Gl → Vector
      | Wp | Wm | Z → Massive_Vector
      end
  | O f → Scalar

let color = function
  | M (U n) → Color.SUN (if n > 0 then 3 else -3)
  | M (D n) → Color.SUN (if n > 0 then 3 else -3)
  | G Gl → Color.AdjSUN 3
  | _ → Color.Singlet
```

```
let prop_spinor n =
  if n ≥ 0 then
    Prop_Spinor
  else
    Prop_ConjSpinor

let propagator = function
  | M f →
      begin match f with
      | L n → prop_spinor n | N n → prop_spinor n
      | U n → prop_spinor n | D n → prop_spinor n
      end
  | G f →
      begin match f with
      | Ga | Gl → Prop_Feynman
      | Wp | Wm | Z → Prop_Unitarity
      end
  | O f →
      begin match f with
      | Phip | Phim | Phi0 → Only_Insertion
      | H | S → Prop_Scalar
      end
```

Optionally, ask for the fudge factor treatment for the widths of charged particles. Currently, this only applies to $W^{\pm}$ and top.

```
let width f =
  if !use_fudged_width then
    match f with
    | G Wp | G Wm | M (U 3) | M (U (−3)) → Fudged
    | _ → !default_width
  else
    !default_width

let goldstone = function
  | G f →
      begin match f with
      | Wp → Some (O Phip, Coupling.Const 1)
      | Wm → Some (O Phim, Coupling.Const 1)
      | Z → Some (O Phi0, Coupling.Const 1)
      | _ → None
      end
  | _ → None

let conjugate = function
  | M f →
      M (begin match f with
      | L n → L (−n) | N n → N (−n)
      | U n → U (−n) | D n → D (−n)
      end)
  | G f →
      G (begin match f with
      | Gl → Gl | Ga → Ga | Z → Z
```

```
        | Wp → Wm | Wm → Wp
        end)
  | O f →
        O (begin match f with
        | Phip → Phim | Phim → Phip | Phi0 → Phi0
        | H → H | S → S
        end)

let fermion = function
  | M f →
        begin match f with
        | L n → if n > 0 then 1 else -1
        | N n → if n > 0 then 1 else -1
        | U n → if n > 0 then 1 else -1
        | D n → if n > 0 then 1 else -1
        end
  | G f →
        begin match f with
        | Gl | Ga | Z | Wp | Wm → 0
        end
  | O _ → 0

module Ch = Charges.QQ

let ( // ) = Algebra.Small_Rational.make

let generation' = function
  | 1 → [ 1//1; 0//1; 0//1]
  | 2 → [ 0//1; 1//1; 0//1]
  | 3 → [ 0//1; 0//1; 1//1]
  | −1 → [−1//1; 0//1; 0//1]
  | −2 → [ 0//1; −1//1; 0//1]
  | −3 → [ 0//1; 0//1; −1//1]
  | n → invalid_arg ("HSExt.generation':␣" ^ string_of_int n)

let generation f =
  match f with
  | M (L n | N n | U n | D n) → generation' n
  | G _ | O _ → [0//1; 0//1; 0//1]

let charge = function
  | M f →
        begin match f with
        | L n → if n > 0 then -1//1 else 1//1
        | N n → 0//1
        | U n → if n > 0 then 2//3 else -2//3
        | D n → if n > 0 then -1//3 else 1//3
        end
  | G f →
        begin match f with
        | Gl | Ga | Z → 0//1
        | Wp → 1//1
        | Wm → −1//1
```

```
      end
  | O f →
      begin match f with
      | H | Phi0 | S → 0//1
      | Phip → 1//1
      | Phim → − 1//1
      end

let lepton = function
  | M f →
      begin match f with
      | L n | N n → if n > 0 then 1//1 else -1//1
      | U _ | D _ → 0//1
      end
  | G _ | O _ → 0//1

let baryon = function
  | M f →
      begin match f with
      | L _ | N _ → 0//1
      | U n | D n → if n > 0 then 1//1 else -1//1
      end
  | G _ | O _ → 0//1

let charges f =
  [ charge f; lepton f; baryon f] @ generation f

type constant =
  | Unit | Pi | Alpha_QED | Sin2thw
  | Sinthw | Costhw | E | G_weak | Vev
  | Q_lepton | Q_up | Q_down | G_CC
  | G_NC_neutrino | G_NC_lepton | G_NC_up | G_NC_down
  | I_Q_W | I_G_ZWW
  | G_WWWW | G_ZZWW | G_AZWW | G_AAWW
  | G_HWW | G_HHWW | G_HZZ | G_HHZZ
  | G_SWW | G_SSWW | G_SZZ | G_SSZZ | G_HSWW |
G_HSZZ
  | G_Htt | G_Hbb | G_Hcc | G_Hmm | G_Htautau | G_H3 |
G_H4_1
  | G_H4_2 | G_H4_3 | G_H4_4 | G_H4_5
  | G_Stt | G_Sbb | G_Scc | G_Smm | G_Stautau | G_HSS |
G_HHS
  | G_HGaZ | G_HGaGa | G_Hgg | G_SGaZ | G_SGaGa | G_Sgg
  | Gs | I_Gs | G2
  | Mass of flavor | Width of flavor
```

Two integer counters for the QCD and EW order of the couplings.

```
type orders = int × int

let orders = function
  | _ → (0,0)

let input_parameters = []
```

```
let derived_parameters = []

let g_over_2_costh =
  Quot (Neg (Atom G_weak), Prod [Const 2; Atom Costhw])

let nc_coupling c t3 q =
  (Real_Array c,
   [Prod [g_over_2_costh; Diff (t3, Prod [Const 2; q; Atom Sin2thw])];
    Prod [g_over_2_costh; t3]])

let half = Quot (Const 1, Const 2)

let derived_parameter_arrays =
  [ nc_coupling G_NC_neutrino half (Const 0);
    nc_coupling G_NC_lepton (Neg half) (Const (-1));
    nc_coupling G_NC_up half (Quot (Const 2, Const 3));
    nc_coupling G_NC_down (Neg half) (Quot (Const (-1), Const 3)) ]

let parameters () =
  { input = input_parameters;
    derived = derived_parameters;
    derived_arrays = derived_parameter_arrays }

module F = Modeltools.Fusions (struct
  type f = flavor
  type c = constant
  let compare = compare
  let conjugate = conjugate
end)

let mgm ((m1, g, m2), fbf, c) = ((M m1, G g, M m2), fbf, c)

let electromagnetic_currents n =
  List.map mgm
    [ ((L (-n), Ga, L n), FBF (1, Psibar, V, Psi), Q_lepton);
      ((U (-n), Ga, U n), FBF (1, Psibar, V, Psi), Q_up);
      ((D (-n), Ga, D n), FBF (1, Psibar, V, Psi), Q_down) ]

let color_currents n =
    List.map mgm
      [ ((U (-n), Gl, U n), FBF ((-1), Psibar, V, Psi), Gs);
        ((D (-n), Gl, D n), FBF ((-1), Psibar, V, Psi), Gs) ]

let neutral_currents n =
  List.map mgm
    [ ((L (-n), Z, L n), FBF (1, Psibar, VA, Psi), G_NC_lepton);
      ((N (-n), Z, N n), FBF (1, Psibar, VA, Psi), G_NC_neutrino);
      ((U (-n), Z, U n), FBF (1, Psibar, VA, Psi), G_NC_up);
      ((D (-n), Z, D n), FBF (1, Psibar, VA, Psi), G_NC_down) ]

let charged_currents n =
  List.map mgm
    [ ((L (-n), Wm, N n), FBF (1, Psibar, VL, Psi), G_CC);
      ((N (-n), Wp, L n), FBF (1, Psibar, VL, Psi), G_CC);
      ((D (-n), Wm, U n), FBF (1, Psibar, VL, Psi), G_CC);
      ((U (-n), Wp, D n), FBF (1, Psibar, VL, Psi), G_CC) ]
```

```
let yukawa =
  [ ((M (U (−3)), O H, M (U 3)), FBF (1, Psibar, Coupling.S, Psi), G_Htt);
    ((M (D (−3)), O H, M (D 3)), FBF (1, Psibar, Coupling.S, Psi), G_Hbb);
    ((M (U (−2)), O H, M (U 2)), FBF (1, Psibar, Coupling.S, Psi), G_Hcc);
    ((M (L (−2)), O H, M (L 2)), FBF (1, Psibar, Coupling.S, Psi), G_Hmm);
    ((M (L (−3)), O H, M (L 3)),
          FBF (1, Psibar, Coupling.S, Psi), G_Htautau);
    ((M (U (−3)), O S, M (U 3)), FBF (1, Psibar, Coupling.S, Psi), G_Stt);
    ((M (D (−3)), O S, M (D 3)), FBF (1, Psibar, Coupling.S, Psi), G_Sbb);
    ((M (U (−2)), O S, M (U 2)), FBF (1, Psibar, Coupling.S, Psi), G_Scc);
    ((M (L (−2)), O S, M (L 2)), FBF (1, Psibar, Coupling.S, Psi), G_Smm);
    ((M (L (−3)), O S, M (L 3)),
          FBF (1, Psibar, Coupling.S, Psi), G_Stautau) ]

let tgc ((g1, g2, g3), t, c) = ((G g1, G g2, G g3), t, c)

let triple_gauge =
  List.map tgc
    [ ((Ga, Wm, Wp), Gauge_Gauge_Gauge 1, I_Q_W);
      ((Z, Wm, Wp), Gauge_Gauge_Gauge 1, I_G_ZWW);
      ((Gl, Gl, Gl), Gauge_Gauge_Gauge 1, I_Gs) ]

let qgc ((g1, g2, g3, g4), t, c) = ((G g1, G g2, G g3, G g4), t, c)

let gauge4 = Vector4 [(2, C_13_42); (−1, C_12_34); (−1, C_14_23)]
let minus_gauge4 = Vector4 [(−2, C_13_42); (1, C_12_34); (1, C_14_23)]
let quartic_gauge =
  List.map qgc
    [ (Wm, Wp, Wm, Wp), gauge4, G_WWWW;
      (Wm, Z, Wp, Z), minus_gauge4, G_ZZWW;
      (Wm, Z, Wp, Ga), minus_gauge4, G_AZWW;
      (Wm, Ga, Wp, Ga), minus_gauge4, G_AAWW;
      (Gl, Gl, Gl, Gl), gauge4, G2]

let gauge_higgs =
  [ ((O H, G Wp, G Wm), Scalar_Vector_Vector 1, G_HWW);
    ((O H, G Z, G Z), Scalar_Vector_Vector 1, G_HZZ);
    ((O S, G Wp, G Wm), Scalar_Vector_Vector 1, G_SWW);
    ((O S, G Z, G Z), Scalar_Vector_Vector 1, G_SZZ) ]

let gauge_higgs4 =
  [ (O H, O H, G Wp, G Wm), Scalar2_Vector2 1, G_HHWW;
    (O H, O H, G Z, G Z), Scalar2_Vector2 1, G_HHZZ;
    (O H, O S, G Wp, G Wm), Scalar2_Vector2 1, G_HSWW;
    (O H, O S, G Z, G Z), Scalar2_Vector2 1, G_HSZZ;
    (O S, O S, G Wp, G Wm), Scalar2_Vector2 1, G_SSWW;
    (O S, O S, G Z, G Z), Scalar2_Vector2 1, G_SSZZ ]

let higgs =
  [ (O H, O H, O H), Scalar_Scalar_Scalar 1, G_H3;
    (O S, O H, O H), Scalar_Scalar_Scalar 1, G_HHS;
    (O S, O S, O H), Scalar_Scalar_Scalar 1, G_HSS ]

let higgs4 =
```

```
  [ (O H, O H, O H, O H), Scalar4 1, G_H4_1;
    (O H, O H, O H, O S), Scalar4 1, G_H4_2;
    (O H, O H, O S, O S), Scalar4 1, G_H4_3;
    (O H, O S, O S, O S), Scalar4 1, G_H4_4;
    (O S, O S, O S, O S), Scalar4 1, G_H4_5 ]

let anomaly_higgs =
  [ (O H, G Ga, G Ga), Dim5_Scalar_Gauge2 1, G_HGaGa;
    (O H, G Ga, G Z), Dim5_Scalar_Gauge2 1, G_HGaZ;
    (O H, G Gl, G Gl), Dim5_Scalar_Gauge2 1, G_Hgg;
    (O S, G Ga, G Ga), Dim5_Scalar_Gauge2 1, G_SGaGa;
    (O S, G Ga, G Z), Dim5_Scalar_Gauge2 1, G_SGaZ;
    (O S, G Gl, G Gl), Dim5_Scalar_Gauge2 1, G_Sgg ]

let goldstone_vertices =
  [ ((O Phi0, G Wm, G Wp), Scalar_Vector_Vector 1, I_G_ZWW);
    ((O Phip, G Ga, G Wm), Scalar_Vector_Vector 1, I_Q_W);
    ((O Phip, G Z, G Wm), Scalar_Vector_Vector 1, I_G_ZWW);
    ((O Phim, G Wp, G Ga), Scalar_Vector_Vector 1, I_Q_W);
    ((O Phim, G Wp, G Z), Scalar_Vector_Vector 1, I_G_ZWW) ]

let vertices3 =
  (ThoList.flatmap electromagnetic_currents [1;2;3] @
   ThoList.flatmap color_currents [1;2;3] @
   ThoList.flatmap neutral_currents [1;2;3] @
   ThoList.flatmap charged_currents [1;2;3] @
   yukawa @ triple_gauge @ gauge_higgs @ higgs @
   anomaly_higgs @ goldstone_vertices)

let vertices4 =
  quartic_gauge @ gauge_higgs4 @ higgs4

let vertices () = (vertices3, vertices4, [])

let table = F.of_vertices (vertices ())
let fuse2 = F.fuse2 table
let fuse3 = F.fuse3 table
let fuse = F.fuse table
let max_degree () = 4

let flavor_of_string = function
  | "e-" → M (L 1) | "e+" → M (L (−1))
  | "mu-" → M (L 2) | "mu+" → M (L (−2))
  | "tau-" → M (L 3) | "tau+" → M (L (−3))
  | "nue" → M (N 1) | "nuebar" → M (N (−1))
  | "numu" → M (N 2) | "numubar" → M (N (−2))
  | "nutau" → M (N 3) | "nutaubar" → M (N (−3))
  | "u" → M (U 1) | "ubar" → M (U (−1))
  | "c" → M (U 2) | "cbar" → M (U (−2))
  | "t" → M (U 3) | "tbar" → M (U (−3))
  | "d" → M (D 1) | "dbar" → M (D (−1))
  | "s" → M (D 2) | "sbar" → M (D (−2))
  | "b" → M (D 3) | "bbar" → M (D (−3))
  | "g" | "gl" → G Gl
```

```
  | "A" → G Ga | "Z" | "Z0" → G Z
  | "W+" → G Wp | "W-" → G Wm
  | "H" → O H
  | "S" → O S
  | _ → invalid_arg "Modellib_BSM.HSExt.flavor_of_string"

let flavor_to_string = function
  | M f →
      begin match f with
      | L 1 → "e-" | L (-1) → "e+"
      | L 2 → "mu-" | L (-2) → "mu+"
      | L 3 → "tau-" | L (-3) → "tau+"
      | L _ → invalid_arg
            "Modellib_BSM.HSExt.flavor_to_string: invalid lepton"
      | N 1 → "nue" | N (-1) → "nuebar"
      | N 2 → "numu" | N (-2) → "numubar"
      | N 3 → "nutau" | N (-3) → "nutaubar"
      | N _ → invalid_arg
            "Modellib_BSM.HSExt.flavor_to_string: invalid neutrino"
      | U 1 → "u" | U (-1) → "ubar"
      | U 2 → "c" | U (-2) → "cbar"
      | U 3 → "t" | U (-3) → "tbar"
      | U _ → invalid_arg
            "Modellib_BSM.HSExt.flavor_to_string: invalid up type quark"
      | D 1 → "d" | D (-1) → "dbar"
      | D 2 → "s" | D (-2) → "sbar"
      | D 3 → "b" | D (-3) → "bbar"
      | D _ → invalid_arg
            "Modellib_BSM.HSExt.flavor_to_string: invalid down type quark"
      end
  | G f →
      begin match f with
      | Gl → "g"
      | Ga → "A" | Z → "Z"
      | Wp → "W+" | Wm → "W-"
      end
  | O f →
      begin match f with
      | Phip → "phi+" | Phim → "phi-" | Phi0 → "phi0"
      | H → "H" | S → "S"
      end

let flavor_to_TeX = function
  | M f →
      begin match f with
      | L 1 → "e^-" | L (-1) → "e^+"
      | L 2 → "\\mu^-" | L (-2) → "\\mu^+"
      | L 3 → "\\tau^-" | L (-3) → "\\tau^+"
      | L _ → invalid_arg
            "Modellib_BSM.HSExt.flavor_to_TeX: invalid lepton"
      | N 1 → "\\nu_e" | N (-1) → "\\bar{\\nu}_e"
```

```
      | N 2 → "\\nu_\\mu" | N (−2) → "\\bar{\\nu}_\\mu"
      | N 3 → "\\nu_\\tau" | N (−3) → "\\bar{\\nu}_\\tau"
      | N _ → invalid_arg
          "Modellib_BSM.HSExt.flavor_to_TeX: invalid neutrino"
      | U 1 → "u" | U (−1) → "\\bar{u}"
      | U 2 → "c" | U (−2) → "\\bar{c}"
      | U 3 → "t" | U (−3) → "\\bar{t}"
      | U _ → invalid_arg
          "Modellib_BSM.HSExt.flavor_to_TeX: invalid up type quark"
      | D 1 → "d" | D (−1) → "\\bar{d}"
      | D 2 → "s" | D (−2) → "\\bar{s}"
      | D 3 → "b" | D (−3) → "\\bar{b}"
      | D _ → invalid_arg
          "Modellib_BSM.HSExt.flavor_to_TeX: invalid down type quark"
      end
  | G f →
      begin match f with
      | Gl → "g"
      | Ga → "\\gamma" | Z → "Z"
      | Wp → "W^+" | Wm → "W^-"
      end
  | O f →
      begin match f with
      | Phip → "\\phi^+" | Phim → "\\phi^-" | Phi0 →
"\\phi^0"
      | H → "H" | S → "S"
      end

let flavor_symbol = function
  | M f →
      begin match f with
      | L n when n > 0 → "l" ^ string_of_int n
      | L n → "l" ^ string_of_int (abs n) ^ "b"
      | N n when n > 0 → "n" ^ string_of_int n
      | N n → "n" ^ string_of_int (abs n) ^ "b"
      | U n when n > 0 → "u" ^ string_of_int n
      | U n → "u" ^ string_of_int (abs n) ^ "b"
      | D n when n > 0 → "d" ^ string_of_int n
      | D n → "d" ^ string_of_int (abs n) ^ "b"
      end
  | G f →
      begin match f with
      | Gl → "gl"
      | Ga → "a" | Z → "z"
      | Wp → "wp" | Wm → "wm"
      end
  | O f →
      begin match f with
      | Phip → "pp" | Phim → "pm" | Phi0 → "p0"
      | H → "h" | S → "s"
```

```
        end

  let pdg = function
    | M f →
        begin match f with
        | L n when n > 0 → 9 + 2 × n
        | L n → − 9 + 2 × n
        | N n when n > 0 → 10 + 2 × n
        | N n → − 10 + 2 × n
        | U n when n > 0 → 2 × n
        | U n → 2 × n
        | D n when n > 0 → − 1 + 2 × n
        | D n → 1 + 2 × n
        end
    | G f →
        begin match f with
        | Gl → 21
        | Ga → 22 | Z → 23
        | Wp → 24 | Wm → (−24)
        end
    | O f →
        begin match f with
        | Phip | Phim → 27 | Phi0 → 26
        | H → 25 | S → 35
        end

  let mass_symbol f =
    "mass(" ^ string_of_int (abs (pdg f)) ^ ")"

  let width_symbol f =
    "width(" ^ string_of_int (abs (pdg f)) ^ ")"

  let constant_symbol = function
    | Unit → "unit" | Pi → "PI"
    | Alpha_QED → "alpha" | E → "e" | G_weak → "g" | Vev →
"vev"
    | Sin2thw → "sin2thw" | Sinthw → "sinthw" | Costhw →
"costhw"
    | Q_lepton → "qlep" | Q_up → "qup" | Q_down → "qdwn"
    | G_NC_lepton → "gnclep" | G_NC_neutrino → "gncneu"
    | G_NC_up → "gncup" | G_NC_down → "gncdwn"
    | G_CC → "gcc"
    | I_Q_W → "iqw" | I_G_ZWW → "igzww"
    | G_WWWW → "gw4" | G_ZZWW → "gzzww"
    | G_AZWW → "gazww" | G_AAWW → "gaaww"
    | G_HWW → "ghww" | G_HZZ → "ghzz"
    | G_HHWW → "ghhww" | G_HHZZ → "ghhzz"
    | G_SWW → "gsww" | G_SZZ → "gszz"
    | G_SSWW → "gssww" | G_SSZZ → "gsszz"
    | G_HSWW → "ghsww" | G_HSZZ → "ghszz"
    | G_Htt → "ghtt" | G_Hbb → "ghbb"
```

```
    | G_Htautau → "ghtautau" | G_Hcc → "ghcc" | G_Hmm →
"ghmm"
    | G_Stt → "gstt" | G_Sbb → "gsbb"
    | G_Stautau → "gstautau" | G_Scc → "gscc" | G_Smm →
"gsmm"
    | G_HGaZ → "ghgaz" | G_HGaGa → "ghgaga" | G_Hgg →
"ghgg"
    | G_SGaZ → "gsgaz" | G_SGaGa → "gsgaga" | G_Sgg → "gsgg"
    | G_H3 → "gh3" | G_H4_1 → "gh4_1" | G_H4_2 → "gh4_2"
    | G_H4_3 → "gh4_3" | G_H4_4 → "gh4_4" | G_H4_5 →
"gh4_5"
    | G_HHS → "ghhs" | G_HSS → "ghss"
    | Gs → "gs" | I_Gs → "igs" | G2 → "gs**2"
    | Mass f → "mass" ^ flavor_symbol f
    | Width f → "width" ^ flavor_symbol f
 end
```

13.6.2 Three-Site Higgsless Model

```
module type Threeshl_options =
  sig
    val include_ckm : bool
    val include_hf : bool
    val diet : bool
  end

module Threeshl_no_ckm : Threeshl_options =
  struct
    let include_ckm = false
    let include_hf = true
    let diet = false
  end

module Threeshl_ckm : Threeshl_options =
  struct
    let include_ckm = true
    let include_hf = true
    let diet = false
  end

module Threeshl_no_ckm_no_hf : Threeshl_options =
  struct
    let include_ckm = false
    let include_hf = false
    let diet = false
  end

module Threeshl_ckm_no_hf : Threeshl_options =
  struct
    let include_ckm = true
    let include_hf = false
```

```
    let diet = false
  end

module Threeshl_diet_no_hf : Threeshl_options =
  struct
    let include_ckm = false
    let include_hf = false
    let diet = true
  end

module Threeshl_diet : Threeshl_options =
  struct
    let include_ckm = false
    let include_hf = true
    let diet = true
  end
```

We use one generic implementation of the model and implement different features via option modules given to a functor

```
module Threeshl (Module_options : Threeshl_options) =
  struct

    open Coupling

    let modname = "Modellib_BSM.Threeshl"

    let rcs =
    let renderbool = function true → "true" | false → "false"
    in RCS.rename rcs_file "Modellib_BSM.Threeshl"
      ["Three-Site␣Higgsless␣Model,␣" ^
        "flavor␣mixing:␣" ^ (renderbool Module_options.include_ckm) ^
        ",␣heavy␣fermions:␣" ^ (renderbool Module_options.include_hf) ^
        ",␣reduced␣set␣of␣couplings:␣" ^ (renderbool Module_options.diet)
      ]
```

Shamelessly stolen from Modellib.SM3, but with no support for fudged width yet

```
    let default_width = ref Timelike
```

If this flag is set true, all gauge bosons are assumed to be massless and are assigned feynman gauge propagators. This in conjunction with the unbroken three site model is intended for checking gauge invariance via the ward identites.

```
    let all_feynman = ref false

    let options = Options.create [
      "constant_width", Arg.Unit (fun _ → default_width := Constant),
        "use␣constant␣width␣(also␣in␣t-channel)";
      "custom_width", Arg.String (fun x → default_width := Custom x),
        "use␣custom␣width";
      "cancel_widths", Arg.Unit (fun _ → default_width := Vanishing),
        "use␣vanishing␣width";
      "all_feynman", Arg.Unit (fun _ → all_feynman := true),
        "assign␣feynman␣gauge␣propagators␣to␣all␣gauge␣bosons\n"
```

```
^ "\t(for␣checking␣the␣ward␣identities);␣use␣only␣if␣you␣*really*␣know\n"
^ "\twhat␣you␣are␣doing"]
```

The quantum numbers that are carried by the particles. `csign` is *not* the charge carried by the particle, but differentiates between particles (`Pos`) and antiparticles (`Neg`)

```
type kkmode = Light | Heavy
type generation = Gen0 | Gen1 | Gen2
type csign = Pos | Neg
type isospin = Iso_up | Iso_down
```

Necessary to represent the indices of the couplings defined in FORTRAN

```
type kk2 = Light2 | Heavy2 | Light_Heavy
```

Map the different types to the constants used in the FORTRAN module

```
let fspec_of_kkmode = function Light → "l_mode" | Heavy →
"h_mode"
let fspec_of_kk2 = function
  Light2 → "l_mode" | Heavy2 → "h_mode" | Light_Heavy →
"lh_mode"
let fspec_of_gen = function Gen0 → "gen_0" | Gen1 → "gen_1" |
Gen2 → "gen_2"
let fspec_of_iso = function Iso_up → "iso_up" | Iso_down →
"iso_down"
```

Covert the "charge sign" into a numeric sign (used e.g. in the determination of the MCID codes)

```
let int_of_csign = function Pos → 1 | Neg → −1
```

Convert the generation into an integer (dito)

```
let int_of_gen = function Gen0 → 1 | Gen1 → 2 | Gen2 → 3
```

The type `flavor` is implemented as a variant. Fermions are implemented as a variant differentating between leptons and quarks (seemed the most natural way as this is also the way in which the FORTRAN code is structured). Bosons are implemented as a variant the differentiates between W, Z and A. All other quantum numbers that are required for identifying the particles are carried by the variant constructors.

```
type fermion =
  | Lepton of (kkmode × csign × generation × isospin)
  | Quark of (kkmode × csign × generation × isospin)

type boson =
  | W of (kkmode × csign)
  | Z of kkmode
  | A
  | G

type flavor = Fermion of fermion | Boson of boson
```

Helpers to construct particles from quantum numbers

```
let lepton kk cs gen iso = Lepton (kk, cs, gen, iso)
let quark kk cs gen iso = Quark (kk, cs, gen, iso)
let w kk cs = W (kk, cs)
let z kk = Z kk
let flavor_of_f x = Fermion x
let flavor_of_b x = Boson x
```

Map a list of functions to the list (partially) applied to a value

```
let revmap funs v = List.map (fun x → x v) funs
```

The same for a list of values; the result is flattened

```
let revmap2 funs vals = ThoList.flatmap (revmap funs) vals
```

Functions to loop the constructors over quantum numbers for list creation purposes

```
let loop_kk flist = revmap2 flist [Light; Heavy]
let loop_cs flist = revmap2 flist [Pos; Neg]
let loop_gen flist = revmap2 flist [Gen0; Gen1; Gen2]
let loop_iso flist = revmap2 flist [Iso_up; Iso_down]
let loop_kk2 flist = revmap2 flist [Light2; Heavy2; Light_Heavy]
```

Conditional looping over kk modes depending on whether to include heavy fermions

```
let cloop_kk flist = match Module_options.include_hf with
  | true → loop_kk flist
  | false → revmap flist Light
let cloop_kk2 flist = match Module_options.include_hf with
  | true → loop_kk2 flist
  | false → revmap flist Light2
```

Having defined the necessary helpers, the magic of currying makes building lists of particles as easy as nesting the loop functions in the correct order...

```
let all_leptons = loop_iso (loop_gen (loop_cs (cloop_kk [lepton] )))
let all_quarks = loop_iso( loop_gen (loop_cs (cloop_kk [quark] )))
let all_bosons = (loop_cs (loop_kk [w] )) @ [Z Light; Z Heavy; A; G]
```

Converts a flavor spec to the BCD identifier defined in the FORTRAN module. Splitting the function into two parts `prefix` and `rump` removes a lot of redundancy.

```
let bcdi_of_flavor =
let prefix = function
  | Fermion (Lepton (Heavy, _, _, _)) | Fermion (Quark (Heavy, _, _, _))
  | Boson (W (Heavy, _)) | Boson (Z Heavy) → "h"
  | _ → ""
in let rump = function
  | Fermion (Lepton spec) → (match spec with
    | (_, _, Gen0, Iso_up) → "nue"
    | (_, _, Gen0, Iso_down) → "e"
    | (_, _, Gen1, Iso_up) → "numu"
    | (_, _, Gen1, Iso_down) → "mu"
```

```
        | (_, _, Gen2, Iso_up) → "nutau"
        | (_, _, Gen2, Iso_down) → "tau")
    | Fermion (Quark spec) → (match spec with
        | (_, _, Gen0, Iso_up) → "u"
        | (_, _, Gen0, Iso_down) → "d"
        | (_, _, Gen1, Iso_up) → "c"
        | (_, _, Gen1, Iso_down) → "s"
        | (_, _, Gen2, Iso_up) → "t"
        | (_, _, Gen2, Iso_down) → "b")
    | Boson (W _) → "w" | Boson (Z _) → "z"
    | Boson A → invalid_arg (modname ^ ".bcd_of_flavor: no bcd for photon!")
    | Boson G → invalid_arg (modname ^ ".bcd_of_flavor: no bcd for gluon!")
in function x → (prefix x) ^ (rump x) ^ "_bcd"
```

The function defined in the model signature which returns the colour representation of a particle

```
let color =
let quarkrep = function
    | (_, Pos, _, _) → Color.SUN 3
    | (_, Neg, _, _) → Color.SUN (−3)
in function
    | Fermion (Quark x) → quarkrep x
    | Boson G → Color.AdjSUN 3
    | _ → Color.Singlet
```

Function for calculating the MCID code of a particle. Convenctions have been choosen such that the heavy modes are identified by the same numbers as the light ones, prefixed with 99. This is supposedly in accord with the conventions for adding new particles to the list of MCID codes. This function is required by the signature.

```
let pdg =
let iso_delta = function Iso_down → 0 | Iso_up → 1
in let gen_delta = function Gen0 → 0 | Gen1 → 2 | Gen2 → 4
in let kk_delta = function Light → 0 | Heavy → 9900
in function
    | Fermion ( Lepton (kk, cs, gen, iso)) →
        (int_of_csign cs) × (11 + (gen_delta gen) + (iso_delta iso) + (kk_delta kk))
    | Fermion ( Quark (kk, cs, gen, iso)) →
        (int_of_csign cs) × (1 + (gen_delta gen) + (iso_delta iso)+ (kk_delta kk))
    | Boson (W (kk, cs)) → (int_of_csign cs) × (24 + (kk_delta kk))
    | Boson (Z kk) → 23 + (kk_delta kk)
    | Boson A → 22
    | Boson G → 21
```

Returns the lorentz representation of a particle; required by the signature.

```
let lorentz =
let spinor = function
    | (_, Pos, _, _) → Spinor
    | (_, Neg, _, _) → ConjSpinor
in function
```

```
    | Fermion (Lepton x) | Fermion (Quark x) → spinor x
    | Boson (W _) | Boson (Z _) → Massive_Vector
    | Boson A → Vector
    | Boson G → Vector
```

O'Mega supports models that allow different gauges; however, we only implement unitary gauge and therefore stub this (SM3 does the same thing). The `gauge` type as well as `gauge_symbol` are required by the signature.

```
type gauge = unit

let gauge_symbol () =
  failwith (modname ^ ".gauge_symbol: internal error")
```

Returns the propagator for a given particle type. Required by signature.

```
let propagator =
let spinorprop = function
    | (_, Pos, _, _) → Prop_Spinor
    | (_, Neg, _, _) → Prop_ConjSpinor
in function
    | Fermion (Lepton x) | Fermion (Quark x) → spinorprop x
    | Boson (W _) | Boson (Z _) →
       (match !all_feynman with false → Prop_Unitarity | true →
Prop_Feynman)
    | Boson A → Prop_Feynman
    | Boson G → Prop_Feynman
```

Return the width of a particle, required by signature.
TODO: Refine such that stable particles always are treade via vanishing width, as this might speed up the generated code a bit.

```
let width _ = !default_width
```

Returns the conjugate particle; required by signature.

```
let conjugate =
let conj_csign = function
    | Pos → Neg
    | Neg → Pos
in function
    | Fermion (Lepton (kk, cs, gen, iso)) → Fermion (Lepton (kk, conj_csign cs, gen, iso))
    | Fermion (Quark (kk, cs, gen, iso)) → Fermion (Quark (kk, conj_csign cs, gen, iso))
    | Boson (W (kk, cs)) → Boson (W (kk, conj_csign cs))
    | x → x
```

Tells the diagram generator whether a particle is a fermion, a conjugate fermion or a boson. Required by signature

```
let fermion = function
    | Fermion (Lepton (_, cs, _, _)) | Fermion (Quark (_, cs, _, _)) →
int_of_csign cs
    | Boson _ → 0
```

Charges are: charge, lepton number, baryon number, generation. Required by signature

```
module Ch = Charges.QQ
let ( // ) = Algebra.Small_Rational.make

let qn_charge = function
  | Boson b → (match b with
    | W (_, c) → (int_of_csign (c)) // 1
    | _ → 0//1)
  | Fermion f → (match f with
    | Lepton (_, c, _, Iso_up) → 0//1
    | Lepton (_, c, _, Iso_down) → (−1 × int_of_csign (c)) // 1
    | Quark (_, c, _, Iso_up) → (2 × int_of_csign (c)) // 3
    | Quark (_, c, _, Iso_down) → (−1 × int_of_csign (c)) // 3)

let qn_lepton = function
  | Fermion (Lepton (_, c, _, _)) → int_of_csign (c) // 1
  | _ → 0//1

let qn_baryon = function
  | Fermion (Quark (_, c, _, _)) → int_of_csign (c) // 1
  | _ → 0//1
```

Generation is conditional: if we enable the nontrivial CKM matrix, all particles carry generation 0; 0; 0

```
let qn_generation x =
  let qn cs gen =
    let c = int_of_csign (cs) in
    match gen with
      | Gen0 → [c//1; 0//1; 0//1]
      | Gen1 → [0//1; c//1; 0//1]
      | Gen2 → [0//1; 0//1; c//1]
  in
  if Module_options.include_ckm then
    [0//1; 0//1; 0//1]
  else
    match x with
      | Fermion (Lepton (_, c, g, _)) → qn c g
      | Fermion (Quark (_, c, g, _)) → qn c g
      | _ → [0//1; 0//1; 0//1]

let charges x =
  [qn_charge x; qn_lepton x; qn_baryon x] @ (qn_generation x)
```

A variant to represent the different coupling constants, choosen to mimic the FORTRAN part. Required by signature.

```
type constant =
  | G_a_lep | G_a_quark of isospin
  | G_aww | G_aaww
  | G_w_lep of (kkmode × kkmode × generation × kkmode × generation)
  | G_w_quark of (kkmode × kkmode × generation × kkmode × generation)
  | G_z_lep of (kkmode × kk2 × generation × isospin)
```

```
| G_z_quark of (kkmode × kk2 × generation × isospin)
| G_wwz of (kk2 × kkmode)
| G_wwzz of (kk2 × kk2)
| G_wwza of (kk2 × kkmode)
| G_wwww of int
| G_s
| IG_s
| G_s2
```

Two integer counters for the QCD and EW order of the couplings.

```
type orders = int × int

let orders = function
  | _ → (0,0)
```

Functions for the construction of constants from indices

```
let g_a_quark x = G_a_quark x
let g_w_lep kk1 kk2 gen1 kk3 gen2 = G_w_lep (kk1, kk2, gen1, kk3, gen2)
let g_w_quark kk1 kk2 gen1 kk3 gen2 = G_w_quark (kk1, kk2, gen1, kk3, gen2)
let g_z_lep kk1 kk2 gen iso = G_z_lep (kk1, kk2, gen, iso)
let g_z_quark kk1 kk2 gen iso = G_z_quark (kk1, kk2, gen, iso)
let g_wwz kk1 kk2 = G_wwz (kk1, kk2)
let g_wwzz kk1 kk2 = G_wwzz (kk1, kk2)
let g_wwza kk1 kk2 = G_wwza (kk1, kk2)
let g_wwww nhw = if (nhw ≥ 0) ∧ (nhw ≤ 4) then G_wwww nhw
    else failwith (modname ^ ".g_wwww: invalid integer, very bad")
```

Build a list of the different constants

```
let clist = [G_a_lep; G_aww; G_aaww] @ (loop_iso [g_a_quark]) @
  (loop_gen (cloop_kk (loop_gen (cloop_kk (loop_kk [g_w_lep] ))))) @
  (loop_gen (cloop_kk (loop_gen (cloop_kk (loop_kk [g_w_quark] ))))) @
  (loop_iso (loop_gen (cloop_kk2 (loop_kk [g_z_lep] )))) @
  (loop_iso (loop_gen (cloop_kk2 (loop_kk [g_z_quark] )))) @
  (loop_kk (loop_kk2 [g_wwz] )) @ (loop_kk2 (loop_kk2 [g_wwzz] )) @
  (loop_kk (loop_kk2 [g_wwza] )) @ (List.map g_wwww [0; 1; 2; 3; 4])
```

Maximum number of lines meeting at a vertex, required by signature.

```
let max_degree () = 4
```

Transform a pair of kk identifiers into a kk2 identifier

```
let get_kk2 = function (Light, Light) → Light2 | (Heavy, Heavy) →
Heavy2
  | (Light, Heavy) | (Heavy, Light) → Light_Heavy
```

Flip isospin

```
let conj_iso = function Iso_up → Iso_down | Iso_down → Iso_up
```

Below, lists of couplings are generated which ultimately are joined into a list of all couplings in the model. The generated lists can be viewed using the `dump.ml` script in the O'Mega toplevel directory.
The individual couplings are defined as 5-tupels resp. 6-tupels consisting in this

order of the particles meeting at the vertex, the coupling type (see `couplings.ml`) and the coupling constant.
List of llA type vertices

```
let vertices_all =
let vgen kk gen =
  ((Fermion (Lepton (kk, Neg, gen, Iso_down)), Boson A, Fermion (Lepton (kk, Pos, gen,
    Iso_down))), FBF(1, Psibar, V, Psi), G_a_lep)
in loop_gen (cloop_kk [vgen])
```

List of qqA type vertices

```
let vertices_aqq =
let vgen kk gen iso =
  ((Fermion (Quark (kk, Neg, gen, iso)), Boson A, Fermion (Quark (kk, Pos, gen,
    iso))), FBF(1, Psibar, V, Psi), G_a_quark iso)
in loop_iso (loop_gen (cloop_kk [vgen]))
```

List of νlW type vertices

```
let vertices_wll =
let vgen kkw kk_f kk_fbar iso_f gen =
  ((Fermion (Lepton (kk_fbar, Neg, gen, conj_iso iso_f)),
    Boson (W (kkw, (match iso_f with Iso_up → Neg | _ →
Pos))),
    Fermion (Lepton (kk_f, Pos, gen, iso_f))),
    FBF (1, Psibar, VA2, Psi),
    G_w_lep (kkw, (match iso_f with Iso_up → kk_f | _ →
kk_fbar), gen,
      (match iso_f with Iso_up → kk_fbar | _ → kk_f), gen) )
in loop_gen (loop_iso (cloop_kk (cloop_kk (loop_kk [vgen] ))))
```

The same list, but without couplings between the W' and light fermions

```
let vertices_wll_diet =
let filter = function
  | ((Fermion (Lepton (Light, _, _, _)), Boson (W (Heavy, _)),
    Fermion (Lepton (Light, _, _, _))), _, _) → false
  | _ → true
in List.filter filter vertices_wll
```

List of udW type vertices, flavor-diagonal

```
let vertices_wqq_no_ckm =
let vgen kkw kk_f kk_fbar iso_f gen =
  ((Fermion (Quark (kk_fbar, Neg, gen, conj_iso iso_f)),
    Boson (W (kkw, (match iso_f with Iso_up → Neg | _ →
Pos))),
    Fermion (Quark (kk_f, Pos, gen, iso_f))),
    FBF (1, Psibar, VA2, Psi),
    G_w_quark (kkw, (match iso_f with Iso_up → kk_f | _ →
kk_fbar), gen,
      (match iso_f with Iso_up → kk_fbar | _ → kk_f), gen) )
in loop_gen (loop_iso (cloop_kk (cloop_kk (loop_kk [vgen] ))))
```

The same list, but without couplings between the W' and the first two generations of quarks

```
let vertices_wqq_no_ckm_diet =
let filter = function
  | ((Fermion (Quark (Light, _, gen, _)), Boson (W (Heavy, _)),
     Fermion (Quark (Light, _, _, _))), _, _) →
       (match gen with Gen2 → true | _ → false)
  | _ → true
in List.filter filter vertices_wqq_no_ckm
```

List of udW type vertices, including non flavor-diagonal couplings

```
let vertices_wqq =
let vgen kkw kk_f gen_f kk_fbar gen_fbar iso_f =
  ((Fermion (Quark (kk_fbar, Neg, gen_fbar, conj_iso iso_f)),
     Boson (W (kkw, (match iso_f with Iso_up → Neg | _ →
Pos))),
     Fermion (Quark (kk_f, Pos, gen_f, iso_f))),
     FBF (1, Psibar, VA2, Psi),
     G_w_quark (match iso_f with
       | Iso_up → (kkw, kk_f, gen_f, kk_fbar, gen_fbar)
       | Iso_down → (kkw, kk_fbar, gen_fbar, kk_f, gen_f)))
in loop_iso (loop_gen (cloop_kk (loop_gen (cloop_kk (loop_kk [vgen] )))))
```

List of llZ / $\nu\nu Z$ type vertices

```
let vertices_zll =
let vgen kkz kk_f kk_fbar gen iso =
  ((Fermion (Lepton (kk_fbar, Neg, gen, iso)), Boson (Z kkz),
     Fermion (Lepton (kk_f, Pos, gen, iso))),
     FBF (1, Psibar, VA2, Psi),
     G_z_lep (kkz, get_kk2 (kk_f, kk_fbar), gen, iso))
in loop_iso (loop_gen (cloop_kk (cloop_kk (loop_kk [vgen] ))))
```

List of qqZ type vertices

```
let vertices_zqq =
let vgen kkz kk_f kk_fbar gen iso =
  ((Fermion (Quark (kk_fbar, Neg, gen, iso)), Boson (Z kkz),
     Fermion (Quark (kk_f, Pos, gen, iso))),
     FBF (1, Psibar, VA2, Psi),
     G_z_quark (kkz, get_kk2 (kk_f, kk_fbar), gen, iso))
in loop_iso (loop_gen (cloop_kk (cloop_kk (loop_kk [vgen] ))))
```

$gq\bar{q}$

```
let vertices_gqq =
let vgen kk gen iso =
  ((Fermion (Quark (kk, Neg, gen, iso)), Boson G, Fermion (Quark (kk, Pos, gen, iso))),
     FBF (1, Psibar, V, Psi), G_s)
in loop_iso (loop_gen (cloop_kk [vgen]))
```

AWW

```
let vertices_aww =
```

```
let vgen kk =
  ( (Boson A, Boson (W (kk, Pos)), Boson (W (kk, Neg))), Gauge_Gauge_Gauge 1, G_aww)
in loop_kk [vgen]
```

ZWW

```
let vertices_zww =
let vgen kkz kkwp kkwm =
  ((Boson (Z kkz), Boson (W (kkwp, Pos)), Boson (W (kkwm, Neg))), Gauge_Gauge_Gauge 1,
    G_wwz (get_kk2 (kkwp, kkwm), kkz))
in loop_kk (loop_kk (loop_kk [vgen]))
```

ggg

```
let vertices_ggg = [(Boson G, Boson G, Boson G), Gauge_Gauge_Gauge (−1), IG_s]
```

Stolen from Modellib.SM; the signs seem to be OK. See `couplings.ml` for more docs.

```
let gauge4 = Vector4 [(2, C_13_42); (−1, C_12_34); (−1, C_14_23)]
let minus_gauge4 = Vector4 [(−2, C_13_42); (1, C_12_34); (1, C_14_23)]
```

AAWW

```
let vertices_aaww =
let vgen kk =
  ((Boson A, Boson (W (kk, Pos)), Boson A, Boson (W (kk, Neg))), minus_gauge4, G_aaww)
in loop_kk [vgen]
```

WWZZ

```
let vertices_wwzz =
let vgen kkwp kkwm kk2z =
  ((Boson (Z (match kk2z with Heavy2 → Heavy | Light2 |
Light_Heavy → Light)),
    Boson (W (kkwp, Pos)),
    Boson (Z (match kk2z with Heavy2 | Light_Heavy → Heavy |
Light2 → Light)),
    Boson (W (kkwm, Neg))), minus_gauge4, G_wwzz (get_kk2 (kkwp, kkwm), kk2z))
in loop_kk2 (loop_kk (loop_kk [vgen]))
```

WWZA

```
let vertices_wwza =
let vgen kkwp kkwm kkz =
  ((Boson A, Boson (W (kkwp, Pos)), Boson (Z kkz), Boson (W (kkwm, Neg))),
    minus_gauge4, G_wwza (get_kk2 (kkwp, kkwm), kkz))
in loop_kk (loop_kk (loop_kk [vgen]))
```

WWWW

```
let vertices_wwww =
let count = function Light2 → 0 | Light_Heavy → 1 | Heavy2 → 2
in let vgen kk2wp kk2wm =
  ((Boson (W ((match kk2wp with Heavy2 → Heavy | Light2 |
Light_Heavy → Light), Pos)),
    Boson (W ((match kk2wm with Heavy2 → Heavy | Light2 |
Light_Heavy → Light), Neg)),
```

```
        Boson (W ((match kk2wp with Heavy2 | Light_Heavy → Heavy |
Light2 → Light), Pos)),
        Boson (W ((match kk2wm with Heavy2 | Light_Heavy →
Heavy | Light2 → Light), Neg))),
        gauge4, G_wwww ((count kk2wp) + (count kk2wm)))
    in loop_kk2 (loop_kk2 [vgen])
```

gggg

```
    let vertices_gggg = [(Boson G, Boson G, Boson G, Boson G), gauge4, G_s2]
```

The list of couplings is transformed into the fusion lists required by the generator by the Model.Fusions functor.
This is copy& paste from the other models; check again with Thorsten if it is correct

```
    module F = Modeltools.Fusions (struct
      type f = flavor
      type c = constant
      let compare = compare
      let conjugate = conjugate
    end )
```

Not sure yet whether F.fusex also creates the conjugate vertices; by looking at the implementation of the other models, I assume it doesn't. Still, better ask Thorsten to be sure!!!
Update: Still didn't get to ask, but since the results are consistent, I suspect my assertion is correct.
The stuff below is required by the signature.

```
    let vertices () = (vertices_all @ vertices_aqq @
      (match Module_options.diet with
        | false → vertices_wll
        | true → vertices_wll_diet) @
      (match (Module_options.include_ckm, Module_options.diet) with
        | (true, false) → vertices_wqq
        | (false, false) → vertices_wqq_no_ckm
        | (false, true) → vertices_wqq_no_ckm_diet
        | (true, true) → raise (Failure
          ("Modules4.Threeshl.vertices: CKM matrix together with option diet is not" ^
          " implemented yet!"))) @
      vertices_zll @ vertices_zqq @ vertices_aww @ vertices_zww @ vertices_gqq @ vertices_ggg,
      vertices_aaww @ vertices_wwzz @ vertices_wwza @ vertices_wwww @ vertices_gggg
      , [])
    let table = F.of_vertices (vertices ())
    let fuse2 = F.fuse2 table
    let fuse3 = F.fuse3 table
    let fuse = F.fuse table
```

A function that returns a list of a flavours known to the model, required by the signature.

```
    let flavors () = (List.map flavor_of_f (all_leptons @ all_quarks)) @
      (List.map flavor_of_b all_bosons)
```

dito, external flavours, also required.

```
let external_flavors () = [
  "light␣leptons", List.map flavor_of_f (loop_iso (loop_gen( loop_cs [lepton Light])));
  "light␣quarks", List.map flavor_of_f (loop_iso (loop_gen( loop_cs [quark Light])));
  "light␣gauge␣bosons", List.map flavor_of_b [W (Light, Pos); W (Light, Neg); Z Light; A];
  "heavy␣gauge␣bosons", List.map flavor_of_b [W (Heavy, Pos); W (Heavy, Neg); Z Heavy]] @
  (match Module_options.include_hf with
    | true → [
        "heavy␣leptons", List.map flavor_of_f (loop_iso (loop_gen( loop_cs [lepton Heavy])));
        "heavy␣quarks", List.map flavor_of_f (loop_iso (loop_gen( loop_cs [quark Heavy])))]
    | false → [] ) @ ["gluons", [Boson G]]
```

Which of the particles are goldstones? → none. Required by the signature.

```
let goldstone x = None
```

This is wrong but handy for debugging the constant identifier generation via -params. Usually, this function would return a record consisting of the parameters as well as expression for the dependent quantities that can be used to generate FORTRAN code for calculating them. However, we have a seperate module for the threeshl, so we can abuse this for debugging. Required by signature.

```
let parameters () = {input = List.map (fun x → (x, 0.)) clist;
  derived = []; derived_arrays = []}
```

Convert a flavour into a ID string with which it will be referred by the user interface of the compiled generator. Required by signature

```
let flavor_to_string =
let prefix = function
  | Fermion (Lepton (Heavy, _, _, _)) | Fermion (Quark (Heavy, _, _, _))
  | Boson (W (Heavy, _)) | Boson (Z Heavy) → "H"
  | _ → ""
in let postfix = function
  | Fermion (Lepton (_, cs, _, Iso_down)) → (match cs with Pos →
"-" | Neg → "+")
  | Fermion (Quark (_, Neg, _, _)) | Fermion (Lepton (_, Neg, _, Iso_up)) →
"bar"
  | Boson (W (_, cs)) → (match cs with Pos → "+" | Neg → "-")
  | _ → ""
in let rump = function
  | Fermion (Lepton desc) → (match desc with
    | (_, _, Gen0, Iso_up) → "nue"
    | (_, _, Gen0, Iso_down) → "e"
    | (_, _, Gen1, Iso_up) → "numu"
    | (_, _, Gen1, Iso_down) → "mu"
    | (_, _, Gen2, Iso_up) → "nutau"
    | (_, _, Gen2, Iso_down) → "tau")
  | Fermion (Quark desc) → (match desc with
    | (_, _, Gen0, Iso_up) → "u"
    | (_, _, Gen0, Iso_down) → "d"
    | (_, _, Gen1, Iso_up) → "c"
```

```
        | (_, _, Gen1, Iso_down) → "s"
        | (_, _, Gen2, Iso_up) → "t"
        | (_, _, Gen2, Iso_down) → "b")
    | Boson (W _) → "W" | Boson (Z _) → "Z" | Boson A → "A" |
Boson G → "gl"
  in function x → (prefix x) ^ (rump x) ^ (postfix x)
```

Conversion of the ID string into a particle flavor. Instead of going through all cases again, we generate a "dictionary" of flavor / ID pairs which we use to identify the correct flavor. Required by signature.

```
let flavor_of_string x =
let dict = List.map (fun x → (x, flavor_to_string x)) (flavors ())
in let get_ident = function (x, _) → x
in try
    get_ident (List.find (fun (_, y) → (x = y)) dict)
  with
    Not_found → (match x with
      | "g" → Boson G
      | _ → invalid_arg (modname ^ ".flavor_of_string")
    )
```

Converts a flavor into a symbol used as identification in the generated FORTRAN code (has to comply to the conventions of valid FORTRAN identifiers therefore). We stick to the same convenctions as SM3, prefixing heavy modes with a H. Required by signature.

```
let flavor_symbol =
let prefix = function
    | Fermion (Lepton (Heavy, _, _, _)) | Fermion (Quark (Heavy, _, _, _))
    | Boson (W (Heavy, _)) | Boson (Z Heavy) → "H"
    | _ → ""
in let postfix = function
    | Fermion (Lepton (_, Neg, _, _)) | Fermion (Quark (_, Neg, _, _)) →
"b"
    | _ → ""
in let rump = function
    | Fermion spec → (match spec with
      | Lepton (_, _, gen, Iso_up) → "n" ^ (string_of_int (int_of_gen gen))
      | Lepton (_, _, gen, Iso_down) → "l" ^ (string_of_int (int_of_gen gen))
      | Quark (_, _, gen, Iso_up) → "u" ^ (string_of_int (int_of_gen gen))
      | Quark (_, _, gen, Iso_down) → "d"^ (string_of_int (int_of_gen gen)))
    | Boson spec → (match spec with
      | W (_, Pos) → "wp" | W (_, Neg) → "wm"
      | Z _ → "z" | A → "a" | G → "gl" )
in function
    x → (prefix x) ^ (rump x) ^ (postfix x)
```

Generate TeX for a flavor

```
let flavor_to_TeX =
let bar x y = match x with Neg → "\\overline{" ^ y ^ "}" | Pos →
y
```

```
in let pm x y = match x with Neg → "{" ^ y ^ "}^+" | Pos →
"{" ^ y ^ "}^-"
in let prime x y = match x with Light → y | Heavy → "{" ^ y ^ "}^\\prime"
in function
  | Fermion (Lepton desc) → (match desc with
    | (kk, cs, gen, Iso_up) → prime kk (bar cs (match gen with
      | Gen0 → "\\nu_e"
      | Gen1 → "\\nu_\\mu"
      | Gen2 → "\\nu_\\tau"))
    | (kk, cs, gen, Iso_down) → prime kk (pm cs (match gen with
      | Gen0 → "e" | Gen1 → "\\mu" | Gen2 → "\\tau")))
  | Fermion (Quark (kk, cs, gen, iso)) → prime kk (bar cs (match (gen, iso) with
    | (Gen0, Iso_up) → "u"
    | (Gen0, Iso_down) → "d"
    | (Gen1, Iso_up) → "c"
    | (Gen1, Iso_down) → "s"
    | (Gen2, Iso_up) → "t"
    | (Gen2, Iso_down) → "b"))
  | Boson spec → (match spec with
    | W (kk, cs) → prime kk (pm (match cs with Pos → Neg |
Neg → Pos) "W")
    | Z kk → prime kk "Z"
    | A → "A" | G → "g")
```

Returns the string referring to the particle mass in the generated FORTRAN code. Required by signature.

```
let mass_symbol = function
  | Boson A | Boson G → "0._default"
  | x → "mass_array(" ^ (bcdi_of_flavor x) ^ ")"
```

Dito, for width. Required by signature.

```
let width_symbol = function
  | Boson A | Boson G → "0._default"
  | x → "width_array(" ^ (bcdi_of_flavor x) ^ ")"
```

Determines the string referring to a coupling constant in the generated FORTRAN code. Required by signature.

```
let constant_symbol =
let c = ",␣"
in let g_w_ferm = function
  (kk1, kk2, gen1, kk3, gen2) →
    ":,␣" ^ (fspec_of_kkmode kk1) ^ c ^ (fspec_of_kkmode kk2) ^ c ^ (fspec_of_gen gen1) ^ c ^
    (fspec_of_kkmode kk3) ^ c ^ (fspec_of_gen gen2)
in let g_z_ferm = function
  (kk1, kk2, gen, iso) →
    ":,␣" ^ (fspec_of_kkmode kk1) ^ c ^ (fspec_of_kk2 kk2) ^ c ^ (fspec_of_gen gen) ^ c ^
    (fspec_of_iso iso)
in function
  | G_a_lep → "g_a_lep"
  | G_s → "g_s_norm"
```

```
        | IG_s → "ig_s_norm"
        | G_s2 → "g_s_norm2"
        | G_a_quark iso → "g_a_quark(" ^ (fspec_of_iso iso) ^ ")"
        | G_aww → "ig_aww"
        | G_aaww → "g_aaww"
        | G_w_lep spec → "g_w_lep_va(" ^ (g_w_ferm spec) ^ ")"
        | G_w_quark spec → "g_w_quark_va(" ^ (g_w_ferm spec) ^ ")"
        | G_z_lep spec → "g_z_lep_va(" ^ (g_z_ferm spec) ^ ")"
        | G_z_quark spec → "g_z_quark_va(" ^ (g_z_ferm spec) ^ ")"
        | G_wwz (kk1, kk2) → "ig_wwz(" ^ (fspec_of_kk2 kk1) ^ c ^
            (fspec_of_kkmode kk2) ^ ")"
        | G_wwzz (kk1, kk2) → "g_wwzz(" ^ (fspec_of_kk2 kk1) ^ c ^
            (fspec_of_kk2 kk2) ^ ")"
        | G_wwza (kk1, kk2) → "g_wwza(" ^(fspec_of_kk2 kk1) ^ c ^
            (fspec_of_kkmode kk2) ^ ")"
        | G_wwww nhw → if (0 ≤ nhw) ∧ (nhw ≤ 4) then
            "g_wwww(" ^ (string_of_int nhw) ^ ")"
            else failwith "Modules4.Threeshl.constant_symbol: invalid int for G_wwww; very bad"
  end
```

13.6.3 2HDM with and without non-trivial flavor structure

```
module type THDM_flags =
  sig
    val ckm_present : bool
  end

module THDM : THDM_flags =
  struct
    let ckm_present = false
  end

module THDM_CKM : THDM_flags =
  struct
    let ckm_present = true
  end

module TwoHiggsDoublet (Flags : THDM_flags) =
  struct
    let rcs = rcs_file

    open Coupling

    let default_width = ref Timelike
    let use_fudged_width = ref false

    let options = Options.create
      [ "constant_width", Arg.Unit (fun () → default_width := Constant),
        "use constant width (also in t-channel)";
        "fudged_width", Arg.Set use_fudged_width,
        "use fudge factor for charge particle width";
        "custom_width", Arg.String (fun f → default_width := Custom f),
```

```
    "use␣custom␣width";
    "cancel_widths", Arg.Unit (fun () → default_width := Vanishing),
    "use␣vanishing␣width" ]

type matter_field = L of int | N of int | U of int | D of int
type gauge_boson = Ga | Wp | Wm | Z | Gl
type other = Phip | Phim | Phi0 | Hh | HA | HH | Hp | Hm
type flavor = M of matter_field | G of gauge_boson | O of other

let matter_field f = M f
let gauge_boson f = G f
let other f = O f

type field =
  | Matter of matter_field
  | Gauge of gauge_boson
  | Other of other

let field = function
  | M f → Matter f
  | G f → Gauge f
  | O f → Other f

type gauge = unit

let gauge_symbol () =
  failwith "Modellib_BSM.TwoHiggsDoublet.gauge_symbol:␣internal␣error"

let family n = List.map matter_field [ L n; N n; U n; D n ]

let external_flavors () =
  [ "1st␣Generation", ThoList.flatmap family [1; − 1];
    "2nd␣Generation", ThoList.flatmap family [2; − 2];
    "3rd␣Generation", ThoList.flatmap family [3; − 3];
    "Gauge␣Bosons", List.map gauge_boson [Ga; Z; Wp; Wm; Gl];
    "Higgs", List.map other [Hh; HH; HA; Hp; Hm];
    "Goldstone␣Bosons", List.map other [Phip; Phim; Phi0] ]

let flavors () = ThoList.flatmap snd (external_flavors ())

let spinor n =
  if n ≥ 0 then
    Spinor
  else
    ConjSpinor

let lorentz = function
  | M f →
      begin match f with
      | L n → spinor n | N n → spinor n
      | U n → spinor n | D n → spinor n
      end
  | G f →
      begin match f with
      | Ga | Gl → Vector
      | Wp | Wm | Z → Massive_Vector
```

```
      end
    | O f → Scalar

  let color = function
    | M (U n) → Color.SUN (if n > 0 then 3 else -3)
    | M (D n) → Color.SUN (if n > 0 then 3 else -3)
    | G Gl → Color.AdjSUN 3
    | _ → Color.Singlet

  let prop_spinor n =
    if n ≥ 0 then
      Prop_Spinor
    else
      Prop_ConjSpinor

  let propagator = function
    | M f →
        begin match f with
        | L n → prop_spinor n | N n → prop_spinor n
        | U n → prop_spinor n | D n → prop_spinor n
        end
    | G f →
        begin match f with
        | Ga | Gl → Prop_Feynman
        | Wp | Wm | Z → Prop_Unitarity
        end
    | O f →
        begin match f with
        | Phip | Phim | Phi0 → Only_Insertion
        | Hh | HH | HA | Hp | Hm → Prop_Scalar
        end
```

Optionally, ask for the fudge factor treatment for the widths of charged particles. Currently, this only applies to $W^{\pm}$ and top.

```
  let width f =
    if !use_fudged_width then
      match f with
      | G Wp | G Wm | M (U 3) | M (U (−3)) → Fudged
      | _ → !default_width
    else
      !default_width

  let goldstone = function
    | G f →
        begin match f with
        | Wp → Some (O Phip, Coupling.Const 1)
        | Wm → Some (O Phim, Coupling.Const 1)
        | Z → Some (O Phi0, Coupling.Const 1)
        | _ → None
        end
    | _ → None
```

```
let conjugate = function
  | M f →
      M (begin match f with
      | L n → L (−n) | N n → N (−n)
      | U n → U (−n) | D n → D (−n)
      end)
  | G f →
      G (begin match f with
      | Gl → Gl | Ga → Ga | Z → Z
      | Wp → Wm | Wm → Wp
      end)
  | O f →
      O (begin match f with
      | Phip → Phim | Phim → Phip | Phi0 → Phi0
      | Hh → Hh | HH → HH | HA → HA
      | Hp → Hm | Hm → Hp
      end)

let fermion = function
  | M f →
      begin match f with
      | L n → if n > 0 then 1 else -1
      | N n → if n > 0 then 1 else -1
      | U n → if n > 0 then 1 else -1
      | D n → if n > 0 then 1 else -1
      end
  | G f →
      begin match f with
      | Gl | Ga | Z | Wp | Wm → 0
      end
  | O _ → 0
```

Electrical charge, lepton number, baryon number. We could avoid the rationals altogether by multiplying the first and last by 3 ...

```
module Ch = Charges.QQ
let ( // ) = Algebra.Small_Rational.make

let generation' = function
  | 1 → [ 1//1; 0//1; 0//1]
  | 2 → [ 0//1; 1//1; 0//1]
  | 3 → [ 0//1; 0//1; 1//1]
  | −1 → [−1//1; 0//1; 0//1]
  | −2 → [ 0//1; −1//1; 0//1]
  | −3 → [ 0//1; 0//1; −1//1]
  | n → invalid_arg ("Modellib_BSM.TwoHiggsDoublet.generation':␣" ^ string_of_int n)

let generation f =
  if Flags.ckm_present then
    []
  else
    match f with
      | M (L n | N n | U n | D n) → generation' n
```

```
      | G _ | O _ → [ 0//1; 0//1; 0//1]
let charge = function
  | M f →
      begin match f with
      | L n → if n > 0 then -1//1 else 1//1
      | N n → 0//1
      | U n → if n > 0 then 2//3 else -2//3
      | D n → if n > 0 then -1//3 else 1//3
      end
  | G f →
      begin match f with
      | Gl | Ga | Z → 0//1
      | Wp → 1//1
      | Wm → − 1//1
      end
  | O f →
      begin match f with
      | Hh | HH | HA | Phi0 → 0//1
      | Hp | Phip → 1//1
      | Hm | Phim → − 1//1
      end
let lepton = function
  | M f →
      begin match f with
      | L n | N n → if n > 0 then 1//1 else -1//1
      | U _ | D _ → 0//1
      end
  | G _ | O _ → 0//1
let baryon = function
  | M f →
      begin match f with
      | L _ | N _ → 0//1
      | U n | D n → if n > 0 then 1//1 else -1//1
      end
  | G _ | O _ → 0//1
let charges f =
    [ charge f; lepton f; baryon f] @ generation f
type constant =
  | Unit | Pi | Alpha_QED | Sin2thw
  | Sinthw | Costhw | E | G_weak | Vev
  | Q_lepton | Q_up | Q_down | G_CC
  | G_NC_neutrino | G_NC_lepton | G_NC_up | G_NC_down
  | I_Q_W | I_G_ZWW | I_G_WWW
  | G_WWWW | G_ZZWW | G_AZWW | G_AAWW
  | G_htt | G_hbb | G_hcc | G_htautau | G_hmumu
  | G_Htt | G_Hbb | G_Hcc | G_Htautau | G_Hmumu
  | I_G_Att | I_G_Abb | I_G_Acc | I_G_Atautau | I_G_Amumu
  | G_Htb | G_Hcs | G_Htaunu | G_Hmunu
```

| *Gs* | *I_Gs* | *G2*
| *G_AHpHm* | *G_ZHpHm* | *G_Zh1h2* | *G_Zh1h3* | *G_Zh2h3*
| *G_WpHmh1* | *G_WpHmh2* | *G_WpHmh3* | *G_WmHph1* | *G_WmHph2*
| *G_WmHph3* | *G_h1ZZ* | *G_h2ZZ* | *G_h3ZZ* | *G_h1WpWm* | *G_h2WpWm*
| *G_h3WpWm* | *G_hhWpWm* | *G_hhZZ* | *G_HpHmAA* | *G_HpHmZZ* | *G_HpHmAZ*
| *G_HpHmWpWm* | *G_h1HpAWm* | *G_h2HpAWm* | *G_h3HpAWm* | *G_h1HpZWm*
| *G_h2HpZWm* | *G_h3HpZWm* | *G_h1HpAWmC* | *G_h2HpAWmC* | *G_h3HpAWmC*
| *G_h1HpZWmC* | *G_h2HpZWmC* | *G_h3HpZWmC* | *G_h1HpHm* | *G_h2HpHm* | *G_h3HpHm*
| *G_h111* | *G_h112* | *G_h113* | *G_h221* | *G_h222* | *G_h223* | *G_h331* | *G_h332*
| *G_h333* | *G_h123* | *G_HpHmHpHm* | *G_HpHm11* | *G_HpHm12* | *G_HpHm13*
| *G_HpHm22* | *G_HpHm23* | *G_HpHm33* | *G_h1111* | *G_h1112* | *G_h1113*
| *G_h1122* | *G_h1123* | *G_h1133* | *G_h1222* | *G_h1223* | *G_h1233* | *G_h1333*
| *G_h2222* | *G_h2223* | *G_h2233* | *G_h2333* | *G_h3333*
| *G_h1uu* | *G_h2uu* | *G_h3uu* | *G_h1uc* | *G_h2uc* | *G_h3uc* | *G_h1ut*
| *G_h2ut* | *G_h3ut* | *G_h1cu* | *G_h2cu* | *G_h3cu* | *G_h1cc* | *G_h2cc*
| *G_h3cc* | *G_h1ct* | *G_h2ct* | *G_h3ct* | *G_h1tu* | *G_h2tu* | *G_h3tu*
| *G_h1tc* | *G_h2tc* | *G_h3tc* | *G_h1tt* | *G_h2tt* | *G_h3tt*
| *G_h1dd* | *G_h2dd* | *G_h3dd* | *G_h1ds* | *G_h2ds* | *G_h3ds* | *G_h1db*
| *G_h2db* | *G_h3db* | *G_h1sd* | *G_h2sd* | *G_h3sd* | *G_h1ss* | *G_h2ss*
| *G_h3ss* | *G_h1sb* | *G_h2sb* | *G_h3sb* | *G_h1bd* | *G_h2bd* | *G_h3bd*
| *G_h1bs* | *G_h2bs* | *G_h3bs* | *G_h1bb* | *G_h2bb* | *G_h3bb*
| *G_hud* | *G_hus* | *G_hub* | *G_hcd* | *G_hcs* | *G_hcb* | *G_htd* | *G_hts* | *G_htb*
| *G_hdu* | *G_hdc* | *G_hdt* | *G_hsu* | *G_hsc* | *G_hst* | *G_hbu* | *G_hbc* | *G_hbt*
| *G_he1n1* | *G_he1n2* | *G_he1n3* | *G_he2n1* | *G_he2n2* | *G_he2n3* | *G_he3n1*
| *G_he3n2* | *G_he3n3* | *G_hn1e1* | *G_hn1e2* | *G_hn1e3* | *G_hn2e1* | *G_hn2e2*
| *G_hn2e3* | *G_hn3e1* | *G_hn3e2* | *G_hn3e3*
| *G_h1e1e1* | *G_h2e1e1* | *G_h3e1e1* | *G_h1e1e2* | *G_h2e1e2* | *G_h3e1e2*
| *G_h1e1e3* | *G_h2e1e3* | *G_h3e1e3* | *G_h1e2e1* | *G_h2e2e1* | *G_h3e2e1*

```
    | G_h1e2e2 | G_h2e2e2 | G_h3e2e2 | G_h1e2e3 | G_h2e2e3 |
G_h3e2e3
    | G_h1e3e1 | G_h2e3e1 | G_h3e3e1 | G_h1e3e2 | G_h2e3e2 |
G_h3e3e2
    | G_h1e3e3 | G_h2e3e3 | G_h3e3e3
    | Mass of flavor | Width of flavor
```

Two integer counters for the QCD and EW order of the couplings.

```
type orders = int × int

let orders = function
  | _ → (0,0)

let g_over_2_costh =
  Quot (Neg (Atom G_weak), Prod [Const 2; Atom Costhw])

let nc_coupling c t3 q =
  (Real_Array c,
   [Prod [g_over_2_costh; Diff (t3, Prod [Const 2; q; Atom Sin2thw])];
    Prod [g_over_2_costh; t3]])

let half = Quot (Const 1, Const 2)

let array_list = [G_h1uu; G_h2uu; G_h3uu; G_h1uc; G_h2uc; G_h3uc; G_h1ut;
    G_h2ut; G_h3ut; G_h1cu; G_h2cu; G_h3cu; G_h1cc; G_h2cc;
    G_h3cc; G_h1ct; G_h2ct; G_h3ct; G_h1tu; G_h2tu; G_h3tu;
    G_h1tc; G_h2tc; G_h3tc; G_h1tt; G_h2tt; G_h3tt;
    G_h1dd; G_h2dd; G_h3dd; G_h1ds; G_h2ds; G_h3ds; G_h1db;
    G_h2db; G_h3db; G_h1sd; G_h2sd; G_h3sd; G_h1ss; G_h2ss;
    G_h3ss; G_h1sb; G_h2sb; G_h3sb; G_h1bd; G_h2bd; G_h3bd;
    G_h1bs; G_h2bs; G_h3bs; G_h1bb; G_h2bb; G_h3bb;
    G_hud; G_hus; G_hub; G_hcd; G_hcs; G_hcb; G_htd; G_hts; G_htb;
    G_hdu; G_hdc; G_hdt; G_hsu; G_hsc; G_hst; G_hbu; G_hbc; G_hbt;
    G_he1n1; G_he1n2; G_he1n3; G_he2n1; G_he2n2; G_he2n3; G_he3n1;
    G_he3n2; G_he3n3; G_hn1e1; G_hn1e2; G_hn1e3; G_hn2e1; G_hn2e2;
    G_hn2e3; G_hn3e1; G_hn3e2; G_hn3e3;
    G_h1e1e1; G_h2e1e1; G_h3e1e1; G_h1e1e2; G_h2e1e2; G_h3e1e2;
    G_h1e1e3; G_h2e1e3; G_h3e1e3; G_h1e2e1; G_h2e2e1; G_h3e2e1;
    G_h1e2e2; G_h2e2e2; G_h3e2e2; G_h1e2e3; G_h2e2e3; G_h3e2e3;
    G_h1e3e1; G_h2e3e1; G_h3e3e1; G_h1e3e2; G_h2e3e2; G_h3e3e2;
    G_h1e3e3; G_h2e3e3; G_h3e3e3]

let add_complex_array_tag c = (Complex_Array c, [Const 0; Const 0])

let derived_parameter_arrays =
  [ nc_coupling G_NC_neutrino half (Const 0);
    nc_coupling G_NC_lepton (Neg half) (Const (−1));
    nc_coupling G_NC_up half (Quot (Const 2, Const 3));
    nc_coupling G_NC_down (Neg half) (Quot (Const (−1), Const 3));
   ] @ (List.map add_complex_array_tag array_list)

let parameters () =
  { input = []; derived = []; derived_arrays = derived_parameter_arrays}
```

```
module F = Modeltools.Fusions (struct
  type f = flavor
  type c = constant
  let compare = compare
  let conjugate = conjugate
end)
```

$$\mathcal{L}_{\mathrm{EM}} = -e\sum_i q_i\bar\psi_i \not{A}\psi_i \tag{13.46}$$

```
let mgm ((m1, g, m2), fbf, c) = ((M m1, G g, M m2), fbf, c)

let electromagnetic_currents n =
  List.map mgm
    [ ((L (-n), Ga, L n), FBF (1, Psibar, V, Psi), Q_lepton);
      ((U (-n), Ga, U n), FBF (1, Psibar, V, Psi), Q_up);
      ((D (-n), Ga, D n), FBF (1, Psibar, V, Psi), Q_down) ]

let color_currents n =
  List.map mgm
    [ ((U (-n), Gl, U n), FBF (1, Psibar, V, Psi), Gs);
      ((D (-n), Gl, D n), FBF (1, Psibar, V, Psi), Gs) ]
```

$$\mathcal{L}_{\mathrm{NC}} = -\frac{g}{2\cos\theta_W}\sum_i \bar\psi_i \not{Z}(g_V^i - g_A^i\gamma_5)\psi_i \tag{13.47}$$

```
let neutral_currents n =
  List.map mgm
    [ ((L (-n), Z, L n), FBF (1, Psibar, VA, Psi), G_NC_lepton);
      ((N (-n), Z, N n), FBF (1, Psibar, VA, Psi), G_NC_neutrino);
      ((U (-n), Z, U n), FBF (1, Psibar, VA, Psi), G_NC_up);
      ((D (-n), Z, D n), FBF (1, Psibar, VA, Psi), G_NC_down) ]
```

$$\mathcal{L}_{\mathrm{CC}} = -\frac{g}{2\sqrt{2}}\sum_i \bar\psi_i (T^+\not{W}^+ + T^-\not{W}^-)(1-\gamma_5)\psi_i \tag{13.48}$$

```
let charged_currents n =
  List.map mgm
    [ ((L (-n), Wm, N n), FBF (1, Psibar, VL, Psi), G_CC);
      ((N (-n), Wp, L n), FBF (1, Psibar, VL, Psi), G_CC);
      ((D (-n), Wm, U n), FBF (1, Psibar, VL, Psi), G_CC);
      ((U (-n), Wp, D n), FBF (1, Psibar, VL, Psi), G_CC) ]

let yukawa =
  [ ((M (U (-1)), O Hh, M (U 1)), FBF (1, Psibar, SP, Psi), G_h1uu);
    ((M (U (-1)), O HH, M (U 1)), FBF (1, Psibar, SP, Psi), G_h2uu);
    ((M (U (-1)), O HA, M (U 1)), FBF (1, Psibar, SP, Psi), G_h3uu);
    ((M (U (-2)), O Hh, M (U 2)), FBF (1, Psibar, SP, Psi), G_h1cc);
    ((M (U (-2)), O HH, M (U 2)), FBF (1, Psibar, SP, Psi), G_h2cc);
    ((M (U (-2)), O HA, M (U 2)), FBF (1, Psibar, SP, Psi), G_h3cc);
    ((M (U (-3)), O Hh, M (U 3)), FBF (1, Psibar, SP, Psi), G_h1tt);
```

$((M\ (U\ (-3)), O\ HH, M\ (U\ 3)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h2tt);$
$((M\ (U\ (-3)), O\ HA, M\ (U\ 3)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h3tt);$

$((M\ (D\ (-1)), O\ Hh, M\ (D\ 1)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h1dd);$
$((M\ (D\ (-1)), O\ HH, M\ (D\ 1)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h2dd);$
$((M\ (D\ (-1)), O\ HA, M\ (D\ 1)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h3dd);$
$((M\ (D\ (-2)), O\ Hh, M\ (D\ 2)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h1ss);$
$((M\ (D\ (-2)), O\ HH, M\ (D\ 2)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h2ss);$
$((M\ (D\ (-2)), O\ HA, M\ (D\ 2)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h3ss);$
$((M\ (D\ (-3)), O\ Hh, M\ (D\ 3)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h1bb);$
$((M\ (D\ (-3)), O\ HH, M\ (D\ 3)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h2bb);$
$((M\ (D\ (-3)), O\ HA, M\ (D\ 3)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h3bb);$

$((M\ (U\ (-1)), O\ Hp, M\ (D\ 1)),\ FBF\ (1, Psibar, SP, Psi),\ G\_hud);$
$((M\ (U\ (-2)), O\ Hp, M\ (D\ 2)),\ FBF\ (1, Psibar, SP, Psi),\ G\_hcs);$
$((M\ (U\ (-3)), O\ Hp, M\ (D\ 3)),\ FBF\ (1, Psibar, SP, Psi),\ G\_htb);$

$((M\ (D\ (-1)), O\ Hm, M\ (U\ 1)),\ FBF\ (1, Psibar, SP, Psi),\ G\_hdu);$
$((M\ (D\ (-2)), O\ Hm, M\ (U\ 2)),\ FBF\ (1, Psibar, SP, Psi),\ G\_hsc);$
$((M\ (D\ (-3)), O\ Hm, M\ (U\ 3)),\ FBF\ (1, Psibar, SP, Psi),\ G\_hbt);$

$((M\ (L\ (-1)), O\ Hh, M\ (L\ 1)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h1e1e1);$
$((M\ (L\ (-1)), O\ HH, M\ (L\ 1)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h2e1e1);$
$((M\ (L\ (-1)), O\ HA, M\ (L\ 1)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h3e1e1);$
$((M\ (L\ (-2)), O\ Hh, M\ (L\ 2)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h1e2e2);$
$((M\ (L\ (-2)), O\ HH, M\ (L\ 2)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h2e2e2);$
$((M\ (L\ (-2)), O\ HA, M\ (L\ 2)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h3e2e2);$
$((M\ (L\ (-3)), O\ Hh, M\ (L\ 3)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h1e3e3);$
$((M\ (L\ (-3)), O\ HH, M\ (L\ 3)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h2e3e3);$
$((M\ (L\ (-3)), O\ HA, M\ (L\ 3)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h3e3e3)$

] @

if $Flags.ckm\_present$ then
$[((M\ (U\ (-1)), O\ Hh,\ M\ (U\ 2)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h1uc);$
$((M\ (U\ (-1)), O\ Hh,\ M\ (U\ 3)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h1ut);$
$((M\ (U\ (-2)), O\ Hh, M\ (U\ 1)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h1cu);$
$((M\ (U\ (-2)), O\ Hh, M\ (U\ 3)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h1ct);$
$((M\ (U\ (-1)), O\ HH, M\ (U\ 2)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h2uc);$
$((M\ (U\ (-1)), O\ HH, M\ (U\ 3)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h2ut);$
$((M\ (U\ (-1)), O\ HA, M\ (U\ 2)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h3uc);$
$((M\ (U\ (-1)), O\ HA, M\ (U\ 3)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h3ut);$
$((M\ (U\ (-2)), O\ HH, M\ (U\ 1)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h2cu);$
$((M\ (U\ (-2)), O\ HH, M\ (U\ 3)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h2ct);$
$((M\ (U\ (-2)), O\ HA, M\ (U\ 1)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h3cu);$
$((M\ (U\ (-2)), O\ HA, M\ (U\ 3)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h3ct);$
$((M\ (U\ (-3)), O\ Hh, M\ (U\ 1)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h1tu);$
$((M\ (U\ (-3)), O\ Hh, M\ (U\ 2)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h1tc);$
$((M\ (U\ (-3)), O\ HH, M\ (U\ 1)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h2tu);$
$((M\ (U\ (-3)), O\ HH, M\ (U\ 2)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h2tc);$
$((M\ (U\ (-3)), O\ HA, M\ (U\ 1)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h3tu);$
$((M\ (U\ (-3)), O\ HA, M\ (U\ 2)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h3tc);$

$((M\ (D\ (-1)), O\ Hh, M\ (D\ 2)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h1ds);$

$((M\ (D\ (-1)), O\ Hh, M\ (D\ 3)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h1db);$
$((M\ (D\ (-1)), O\ HH, M\ (D\ 2)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h2ds);$
$((M\ (D\ (-1)), O\ HH, M\ (D\ 3)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h2db);$
$((M\ (D\ (-1)), O\ HA, M\ (D\ 2)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h3ds);$
$((M\ (D\ (-1)), O\ HA, M\ (D\ 3)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h3db);$
$((M\ (D\ (-2)), O\ Hh, M\ (D\ 1)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h1sd);$
$((M\ (D\ (-2)), O\ Hh, M\ (D\ 3)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h1sb);$
$((M\ (D\ (-2)), O\ HH, M\ (D\ 1)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h2sd);$
$((M\ (D\ (-2)), O\ HH, M\ (D\ 3)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h2sb);$
$((M\ (D\ (-2)), O\ HA, M\ (D\ 1)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h3sd);$

$((M\ (L\ (-1)), O\ Hh, M\ (L\ 2)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h1e1e2);$
$((M\ (L\ (-1)), O\ Hh, M\ (L\ 3)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h1e1e3);$
$((M\ (L\ (-1)), O\ HH, M\ (L\ 2)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h2e1e2);$
$((M\ (L\ (-1)), O\ HH, M\ (L\ 3)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h2e1e3);$
$((M\ (L\ (-1)), O\ HA, M\ (L\ 2)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h3e1e2);$
$((M\ (L\ (-1)), O\ HA, M\ (L\ 3)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h3e1e3);$
$((M\ (L\ (-2)), O\ Hh, M\ (L\ 1)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h1e2e1);$
$((M\ (L\ (-2)), O\ Hh, M\ (L\ 3)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h1e2e3);$
$((M\ (L\ (-2)), O\ HH, M\ (L\ 1)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h2e2e1);$
$((M\ (L\ (-2)), O\ HH, M\ (L\ 3)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h2e2e3);$
$((M\ (L\ (-2)), O\ HA, M\ (L\ 1)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h3e2e1);$
$((M\ (L\ (-2)), O\ HA, M\ (L\ 3)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h3e2e3);$
$((M\ (L\ (-3)), O\ Hh, M\ (L\ 1)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h1e3e1);$
$((M\ (L\ (-3)), O\ Hh, M\ (L\ 2)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h1e3e2);$
$((M\ (L\ (-3)), O\ HH, M\ (L\ 1)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h2e3e1);$
$((M\ (L\ (-3)), O\ HH, M\ (L\ 2)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h2e3e2);$
$((M\ (L\ (-3)), O\ HA, M\ (L\ 1)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h3e3e1);$
$((M\ (L\ (-3)), O\ HA, M\ (L\ 2)),\ FBF\ (1, Psibar, SP, Psi),\ G\_h3e3e2)\ ]$
else
 [ ]

$$\mathcal{L}_{\mathrm{TGC}} = -e\partial_\mu A_\nu W_+^\mu W_-^\nu + \ldots - e\cot\theta_w \partial_\mu Z_\nu W_+^\mu W_-^\nu + \ldots \tag{13.49}$$

```
let tgc ((g1, g2, g3), t, c) = ((G g1, G g2, G g3), t, c)
```

```
let triple_gauge =
  List.map tgc
    [ ((Ga, Wm, Wp), Gauge_Gauge_Gauge 1, I_Q_W);
      ((Z, Wm, Wp), Gauge_Gauge_Gauge 1, I_G_ZWW);
      ((Gl, Gl, Gl), Gauge_Gauge_Gauge 1, I_Gs) ]
```

$$\mathcal{L}_{\mathrm{QGC}} = -g^2 W_{+,\mu} W_{-,\nu} W_+^\mu W_-^\nu + \ldots \tag{13.50}$$

Actually, quartic gauge couplings are a little bit more straightforward using auxiliary fields. Here we have to impose the antisymmetry manually:

$$\begin{aligned}(W_1^{+,\mu} W_2^{-,\nu} - W_1^{+,\nu} W_2^{-,\mu})(W_{3,\mu}^+ W_{4,\nu}^- - W_{3,\nu}^+ W_{4,\mu}^-) \\ = 2(W_1^+ W_3^+)(W_2^- W_4^-) - 2(W_1^+ W_4^-)(W_2^- W_3^+)\end{aligned} \tag{13.51a}$$

also (V can be A or Z)

$$(W_1^{+,\mu} V_2^\nu - W_1^{+,\nu} V_2^\mu)(W_{3,\mu}^- V_{4,\nu} - W_{3,\nu}^- V_{4,\mu}) \\ = 2(W_1^+ W_3^-)(V_2 V_4) - 2(W_1^+ V_4)(V_2 W_3^-) \quad (13.51b)$$

$$W^{+,\mu} W^{-,\nu} W_\mu^+ W_\nu^- \quad (13.52a)$$

```
let qgc ((g1, g2, g3, g4), t, c) = ((G g1, G g2, G g3, G g4), t, c)

let gauge4 = Vector4 [(2, C_13_42); (−1, C_12_34); (−1, C_14_23)]
let minus_gauge4 = Vector4 [(−2, C_13_42); (1, C_12_34); (1, C_14_23)]
let quartic_gauge =
  List.map qgc
    [ (Wm, Wp, Wm, Wp), gauge4, G_WWWW;
      (Wm, Z, Wp, Z), minus_gauge4, G_ZZWW;
      (Wm, Z, Wp, Ga), minus_gauge4, G_AZWW;
      (Wm, Ga, Wp, Ga), minus_gauge4, G_AAWW;
      (Gl, Gl, Gl, Gl), gauge4, G2]

let gauge_higgs =
  [ (G Ga, O Hp, O Hm), Vector_Scalar_Scalar 1, G_AHpHm;
    (G Z, O Hp, O Hm), Vector_Scalar_Scalar 1, G_ZHpHm;
    (G Z, O Hh, O HH), Vector_Scalar_Scalar 1, G_Zh1h2;
    (G Z, O Hh, O HA), Vector_Scalar_Scalar 1, G_Zh1h3;
    (G Z, O HH, O HA), Vector_Scalar_Scalar 1, G_Zh2h3;
    (G Wp, O Hm, O Hh), Vector_Scalar_Scalar 1, G_WpHmh1;
    (G Wp, O Hm, O HH), Vector_Scalar_Scalar 1, G_WpHmh2;
    (G Wp, O Hm, O HA), Vector_Scalar_Scalar 1, G_WpHmh3;
    (G Wm, O Hp, O Hh), Vector_Scalar_Scalar 1, G_WmHph1;
    (G Wm, O Hp, O HH), Vector_Scalar_Scalar 1, G_WmHph2;
    (G Wm, O Hp, O HA), Vector_Scalar_Scalar 1, G_WmHph3;
    (O Hh, G Z, G Z), Scalar_Vector_Vector 1, G_h1ZZ;
    (O HH, G Z, G Z), Scalar_Vector_Vector 1, G_h2ZZ;
    (O HA, G Z, G Z), Scalar_Vector_Vector 1, G_h3ZZ;
    (O Hh, G Wp, G Wm), Scalar_Vector_Vector 1, G_h1WpWm;
    (O HH, G Wp, G Wm), Scalar_Vector_Vector 1, G_h2WpWm;
    (O HA, G Wp, G Wm), Scalar_Vector_Vector 1, G_h3WpWm ]

let gauge_higgs4 =
  [ (O Hh, O Hh, G Wp, G Wm), Scalar2_Vector2 1, G_hhWpWm;
    (O HH, O HH, G Wp, G Wm), Scalar2_Vector2 1, G_hhWpWm;
    (O HA, O HA, G Wp, G Wm), Scalar2_Vector2 1, G_hhWpWm;
    (O Hh, O Hh, G Z, G Z), Scalar2_Vector2 1, G_hhZZ;
    (O HH, O HH, G Z, G Z), Scalar2_Vector2 1, G_hhZZ;
    (O HA, O HA, G Z, G Z), Scalar2_Vector2 1, G_hhZZ;
    (O Hp, O Hm, G Ga, G Ga), Scalar2_Vector2 1, G_HpHmAA;
    (O Hp, O Hm, G Z, G Z), Scalar2_Vector2 1, G_HpHmZZ;
    (O Hp, O Hm, G Ga, G Z), Scalar2_Vector2 1, G_HpHmAZ;
    (O Hp, O Hm, G Wp, G Wm), Scalar2_Vector2 1, G_HpHmWpWm;
```

```
(O Hh, O Hp, G Ga, G Wm), Scalar2_Vector2 1, G_h1HpAWm;
(O HH, O Hp, G Ga, G Wm), Scalar2_Vector2 1, G_h2HpAWm;
(O HA, O Hp, G Ga, G Wm), Scalar2_Vector2 1, G_h3HpAWm;
(O Hh, O Hp, G Z, G Wm), Scalar2_Vector2 1, G_h1HpZWm;
(O HH, O Hp, G Z, G Wm), Scalar2_Vector2 1, G_h2HpZWm;
(O HA, O Hp, G Z, G Wm), Scalar2_Vector2 1, G_h3HpZWm;
(O Hh, O Hm, G Ga, G Wp), Scalar2_Vector2 1, G_h1HpAWmC;
(O HH, O Hm, G Ga, G Wp), Scalar2_Vector2 1, G_h2HpAWmC;
(O HA, O Hm, G Ga, G Wp), Scalar2_Vector2 1, G_h3HpAWmC;
(O Hh, O Hm, G Z, G Wp), Scalar2_Vector2 1, G_h1HpZWmC;
(O HH, O Hm, G Z, G Wp), Scalar2_Vector2 1, G_h2HpZWmC;
(O HA, O Hm, G Z, G Wp), Scalar2_Vector2 1, G_h3HpZWmC ]

let higgs =
  [ (O Hh, O Hp, O Hm), Scalar_Scalar_Scalar 1, G_h1HpHm;
    (O HH, O Hp, O Hm), Scalar_Scalar_Scalar 1, G_h2HpHm;
    (O HA, O Hp, O Hm), Scalar_Scalar_Scalar 1, G_h3HpHm;
    (O Hh, O Hh, O Hh), Scalar_Scalar_Scalar 1, G_h111;
    (O Hh, O Hh, O HH), Scalar_Scalar_Scalar 1, G_h112;
    (O Hh, O Hh, O HA), Scalar_Scalar_Scalar 1, G_h113;
    (O HH, O HH, O Hh), Scalar_Scalar_Scalar 1, G_h221;
    (O HH, O HH, O HH), Scalar_Scalar_Scalar 1, G_h222;
    (O HH, O HH, O HA), Scalar_Scalar_Scalar 1, G_h223;
    (O HA, O HA, O Hh), Scalar_Scalar_Scalar 1, G_h331;
    (O HA, O HA, O HH), Scalar_Scalar_Scalar 1, G_h332;
    (O HA, O HA, O HA), Scalar_Scalar_Scalar 1, G_h333;
    (O Hh, O HH, O HA), Scalar_Scalar_Scalar 1, G_h123 ]

let higgs4 =
  [ (O Hp, O Hm, O Hp, O Hm), Scalar4 1, G_HpHmHpHm;
    (O Hp, O Hm, O Hh, O Hh), Scalar4 1, G_HpHm11;
    (O Hp, O Hm, O Hh, O HH), Scalar4 1, G_HpHm12;
    (O Hp, O Hm, O Hh, O HA), Scalar4 1, G_HpHm13;
    (O Hp, O Hm, O HH, O HH), Scalar4 1, G_HpHm22;
    (O Hp, O Hm, O HH, O HA), Scalar4 1, G_HpHm23;
    (O Hp, O Hm, O HA, O HA), Scalar4 1, G_HpHm33;
    (O Hh, O Hh, O Hh, O Hh), Scalar4 1, G_h1111;
    (O Hh, O Hh, O Hh, O HH), Scalar4 1, G_h1112;
    (O Hh, O Hh, O Hh, O HA), Scalar4 1, G_h1113;
    (O Hh, O Hh, O HH, O HH), Scalar4 1, G_h1122;
    (O Hh, O Hh, O HH, O HA), Scalar4 1, G_h1123;
    (O Hh, O Hh, O HA, O HA), Scalar4 1, G_h1133;
    (O Hh, O HH, O HH, O HH), Scalar4 1, G_h1222;
    (O Hh, O HH, O HH, O HA), Scalar4 1, G_h1223;
    (O Hh, O HH, O HA, O HA), Scalar4 1, G_h1233;
    (O Hh, O HA, O HA, O HA), Scalar4 1, G_h1333;
    (O HH, O HH, O HH, O HH), Scalar4 1, G_h2222;
    (O HH, O HH, O HH, O HA), Scalar4 1, G_h2223;
    (O HH, O HH, O HA, O HA), Scalar4 1, G_h2233;
    (O HH, O HA, O HA, O HA), Scalar4 1, G_h2333;
    (O HA, O HA, O HA, O HA), Scalar4 1, G_h3333 ]
```

```
let goldstone_vertices =
  [ ((O Phi0, G Wm, G Wp), Scalar_Vector_Vector 1, I_G_ZWW);
    ((O Phip, G Ga, G Wm), Scalar_Vector_Vector 1, I_Q_W);
    ((O Phip, G Z, G Wm), Scalar_Vector_Vector 1, I_G_ZWW);
    ((O Phim, G Wp, G Ga), Scalar_Vector_Vector 1, I_Q_W);
    ((O Phim, G Wp, G Z), Scalar_Vector_Vector 1, I_G_ZWW) ]

let vertices3 =
  (ThoList.flatmap electromagnetic_currents [1; 2; 3] @
   ThoList.flatmap color_currents [1; 2; 3] @
   ThoList.flatmap neutral_currents [1; 2; 3] @
   ThoList.flatmap charged_currents [1; 2; 3] @
   yukawa @ triple_gauge @
   gauge_higgs @ higgs @ goldstone_vertices)

let vertices4 =
  quartic_gauge @ gauge_higgs4 @ higgs4

let vertices () = (vertices3, vertices4, [])
```

For efficiency, make sure that *F.of_vertices vertices* is evaluated only once.

```
let table = F.of_vertices (vertices ())
let fuse2 = F.fuse2 table
let fuse3 = F.fuse3 table
let fuse = F.fuse table
let max_degree () = 4

let flavor_of_string = function
  | "e-" → M (L 1) | "e+" → M (L (−1))
  | "mu-" → M (L 2) | "mu+" → M (L (−2))
  | "tau-" → M (L 3) | "tau+" → M (L (−3))
  | "nue" → M (N 1) | "nuebar" → M (N (−1))
  | "numu" → M (N 2) | "numubar" → M (N (−2))
  | "nutau" → M (N 3) | "nutaubar" → M (N (−3))
  | "u" → M (U 1) | "ubar" → M (U (−1))
  | "c" → M (U 2) | "cbar" → M (U (−2))
  | "t" → M (U 3) | "tbar" → M (U (−3))
  | "d" → M (D 1) | "dbar" → M (D (−1))
  | "s" → M (D 2) | "sbar" → M (D (−2))
  | "b" → M (D 3) | "bbar" → M (D (−3))
  | "g" → G Gl
  | "A" → G Ga | "Z" | "Z0" → G Z
  | "W+" → G Wp | "W-" → G Wm
  | "h0" → O Hh
  | "H0" → O HH
  | "A0" → O HA
  | _ → invalid_arg "Modellib_BSM.TwoHiggsDoublet.flavor_of_string"

let flavor_to_string = function
  | M f →
      begin match f with
      | L 1 → "e-" | L (−1) → "e+"
      | L 2 → "mu-" | L (−2) → "mu+"
```

```
      | L 3 → "tau-" | L (−3) → "tau+"
      | L _ → invalid_arg
            "Modellib_BSM.TwoHiggsDoublet.flavor_to_string:␣invalid␣lepton"
      | N 1 → "nue" | N (−1) → "nuebar"
      | N 2 → "numu" | N (−2) → "numubar"
      | N 3 → "nutau" | N (−3) → "nutaubar"
      | N _ → invalid_arg
            "Modellib_BSM.TwoHiggsDoublet.flavor_to_string:␣invalid␣neutrino"
      | U 1 → "u" | U (−1) → "ubar"
      | U 2 → "c" | U (−2) → "cbar"
      | U 3 → "t" | U (−3) → "tbar"
      | U _ → invalid_arg
            "Modellib_BSM.TwoHiggsDoublet.flavor_to_string:␣invalid␣up␣type␣quark"
      | D 1 → "d" | D (−1) → "dbar"
      | D 2 → "s" | D (−2) → "sbar"
      | D 3 → "b" | D (−3) → "bbar"
      | D _ → invalid_arg
            "Modellib_BSM.TwoHiggsDoublet.flavor_to_string:␣invalid␣down␣type␣quark"
      end
  | G f →
      begin match f with
      | Gl → "g"
      | Ga → "A" | Z → "Z"
      | Wp → "W+" | Wm → "W-"
      end
  | O f →
      begin match f with
      | Phip → "phi+" | Phim → "phi-" | Phi0 → "phi0"
      | Hh → "h0" | HH → "H0" | HA → "A0"
      | Hp → "H+" | Hm → "H-"
      end

let flavor_to_TeX = function
  | M f →
      begin match f with
      | L 1 → "e^-" | L (−1) → "e^+"
      | L 2 → "\\mu^-" | L (−2) → "\\mu^+"
      | L 3 → "\\tau^-" | L (−3) → "\\tau^+"
      | L _ → invalid_arg
            "Modellib_BSM.TwoHiggsDoublet.flavor_to_TeX:␣invalid␣lepton"
      | N 1 → "\\nu_e" | N (−1) → "\\bar{\\nu}_e"
      | N 2 → "\\nu_\\mu" | N (−2) → "\\bar{\\nu}_\\mu"
      | N 3 → "\\nu_\\tau" | N (−3) → "\\bar{\\nu}_\\tau"
      | N _ → invalid_arg
            "Modellib_BSM.TwoHiggsDoublet.flavor_to_TeX:␣invalid␣neutrino"
      | U 1 → "u" | U (−1) → "\\bar{u}"
      | U 2 → "c" | U (−2) → "\\bar{c}"
      | U 3 → "t" | U (−3) → "\\bar{t}"
      | U _ → invalid_arg
            "Modellib_BSM.TwoHiggsDoublet.flavor_to_TeX:␣invalid␣up␣type␣quark"
```

```
        | D 1 → "d" | D (−1) → "\\bar{d}"
        | D 2 → "s" | D (−2) → "\\bar{s}"
        | D 3 → "b" | D (−3) → "\\bar{b}"
        | D _ → invalid_arg
              "Modellib_BSM.TwoHiggsDoublet.flavor_to_TeX: invalid down type quark"
        end
    | G f →
        begin match f with
        | Gl → "g"
        | Ga → "\\gamma" | Z → "Z"
        | Wp → "W^+" | Wm → "W^-"
        end
    | O f →
        begin match f with
        | Phip → "\\phi^+" | Phim → "\\phi^-" | Phi0 →
"\\phi^0"
        | Hh → "h^0" | HH → "H^0" | HA → "A^0"
        | Hp → "H^+" | Hm → "H^-"
        end

  let flavor_symbol = function
    | M f →
        begin match f with
        | L n when n > 0 → "l" ^ string_of_int n
        | L n → "l" ^ string_of_int (abs n) ^ "b"
        | N n when n > 0 → "n" ^ string_of_int n
        | N n → "n" ^ string_of_int (abs n) ^ "b"
        | U n when n > 0 → "u" ^ string_of_int n
        | U n → "u" ^ string_of_int (abs n) ^ "b"
        | D n when n > 0 → "d" ^ string_of_int n
        | D n → "d" ^ string_of_int (abs n) ^ "b"
        end
    | G f →
        begin match f with
        | Gl → "gl"
        | Ga → "a" | Z → "z"
        | Wp → "wp" | Wm → "wm"
        end
    | O f →
        begin match f with
        | Phip → "pp" | Phim → "pm" | Phi0 → "p0"
        | Hh → "h" | HH → "h0" | HA → "a0"
        | Hp → "hp" | Hm → "hm"
        end

  let pdg = function
    | M f →
        begin match f with
        | L n when n > 0 → 9 + 2 × n
        | L n → − 9 + 2 × n
        | N n when n > 0 → 10 + 2 × n
```

```
          | N n → − 10 + 2 × n
          | U n when n > 0 → 2 × n
          | U n → 2 × n
          | D n when n > 0 → − 1 + 2 × n
          | D n → 1 + 2 × n
          end
      | G f →
          begin match f with
          | Gl → 21
          | Ga → 22 | Z → 23
          | Wp → 24 | Wm → (−24)
          end
      | O f →
          begin match f with
          | Phip → 27 | Phim → − 27 | Phi0 → 26
          | Hh → 25
          | HH → 35
          | HA → 36
          | Hp → 37
          | Hm → − 37
          end
  let mass_symbol f =
    "mass(" ^ string_of_int (abs (pdg f)) ^ ")"
  let width_symbol f =
    "width(" ^ string_of_int (abs (pdg f)) ^ ")"
  let constant_symbol = function
      | Unit → "unit" | Pi → "PI"
      | Alpha_QED → "alpha" | E → "e" | G_weak → "g" | Vev →
"vev"
      | Sin2thw → "sin2thw" | Sinthw → "sinthw" | Costhw →
"costhw"
      | Q_lepton → "qlep" | Q_up → "qup" | Q_down → "qdwn"
      | G_NC_lepton → "gnclep" | G_NC_neutrino → "gncneu"
      | G_NC_up → "gncup" | G_NC_down → "gncdwn"
      | G_CC → "gcc"
      | I_Q_W → "iqw" | I_G_ZWW → "igzww" | I_G_WWW →
"igwww"
      | G_WWWW → "gw4" | G_ZZWW → "gzzww"
      | G_AZWW → "gazww" | G_AAWW → "gaaww"
      | G_htt → "ghtt" | G_hbb → "ghbb" | G_hcc → "ghcc"
      | G_Htt → "gh0tt" | G_Hbb → "gh0bb" | G_Hcc → "gh0cc"
      | I_G_Att → "iga0tt" | I_G_Abb → "iga0bb" | I_G_Acc →
"iga0cc"
      | G_htautau → "ghtautau" | G_hmumu → "ghmumu"
      | G_Htautau → "gh0tautau" | G_Hmumu → "gh0mumu"
      | I_G_Atautau → "iga0tautau" | I_G_Amumu → "iga0mumu"
      | G_Htb → "ghptb" | G_Hcs → "ghpcs"
      | G_Htaunu → "ghptaunu" | G_Hmunu → "ghpmunu"
      | Gs → "gs" | I_Gs → "igs" | G2 → "gs**2"
```

| G_AHpHm $\rightarrow$ "gAHpHm" | G_ZHpHm $\rightarrow$ "gZHpHm"
| G_Zh1h2 $\rightarrow$ "gZh1h2" | G_Zh1h3 $\rightarrow$ "gZh1h3"
| G_Zh2h3 $\rightarrow$ "gZh2h3" | G_WpHmh1 $\rightarrow$ "gWpHmh1"
| G_WpHmh2 $\rightarrow$ "gWpHmh2" | G_WpHmh3 $\rightarrow$ "gWpHmh3"
| G_WmHph1 $\rightarrow$ "gWmHph1" | G_WmHph2 $\rightarrow$ "gWmHph2"
| G_WmHph3 $\rightarrow$ "gWmHph3" | G_h1ZZ $\rightarrow$ "gh1ZZ"
| G_h2ZZ $\rightarrow$ "gh2ZZ" | G_h3ZZ $\rightarrow$ "gh3ZZ"
| G_h1WpWm $\rightarrow$ "gh1WpWm" | G_h2WpWm $\rightarrow$ "gh2WpWm"
| G_h3WpWm $\rightarrow$ "gh3WpWm" | G_hhWpWm $\rightarrow$ "ghhWpWm"
| G_hhZZ $\rightarrow$ "ghhZZ" | G_HpHmAA $\rightarrow$ "gHpHmAA"
| G_HpHmZZ $\rightarrow$ "gHpHmZZ" | G_HpHmAZ $\rightarrow$ "gHpHmAZ"
| $G\_HpHmWpWm$ $\rightarrow$ "gHpHmWpWm" | $G\_h1HpAWm$ $\rightarrow$ "gh1HpAWm"
| $G\_h2HpAWm$ $\rightarrow$ "gh2HpAWm" | $G\_h3HpAWm$ $\rightarrow$ "gh3HpAWm"
| $G\_h1HpZWm$ $\rightarrow$ "gh1HpZWm" | $G\_h2HpZWm$ $\rightarrow$ "gh2HpZWm"
| $G\_h3HpZWm$ $\rightarrow$ "gh3HpZWm" | $G\_h1HpAWmC$ $\rightarrow$ "gh1HpAWmC"
| $G\_h2HpAWmC$ $\rightarrow$ "gh2HpAWmC" | $G\_h3HpAWmC$ $\rightarrow$ "gh3HpAWmC"
| $G\_h1HpZWmC$ $\rightarrow$ "gh1HpZWmC" | $G\_h2HpZWmC$ $\rightarrow$ "gh2HpZWmC"
| $G\_h3HpZWmC$ $\rightarrow$ "gh3HpZWmC" | G_h1HpHm $\rightarrow$ "gh1HpHm"
| G_h2HpHm $\rightarrow$ "gh2HpHm" | G_h3HpHm $\rightarrow$ "gh3HpHm"
| G_h111 $\rightarrow$ "gh111" | G_h112 $\rightarrow$ "gh112" | G_h113 $\rightarrow$ "gh113"
| G_h221 $\rightarrow$ "gh221" | G_h222 $\rightarrow$ "gh222" | G_h223 $\rightarrow$ "gh223"
| G_h331 $\rightarrow$ "gh331" | G_h332 $\rightarrow$ "gh332"
| G_h333 $\rightarrow$ "gh333" | G_h123 $\rightarrow$ "gh123"
| $G\_HpHmHpHm$ $\rightarrow$ "gHpHmHpHm" | G_HpHm11 $\rightarrow$ "gHpHm11"
| G_HpHm12 $\rightarrow$ "gHpHm12" | G_HpHm13 $\rightarrow$ "gHpHm13"
| G_HpHm22 $\rightarrow$ "gHpHm22" | G_HpHm23 $\rightarrow$ "gHpHm23"
| G_HpHm33 $\rightarrow$ "gHpHm33" | G_h1111 $\rightarrow$ "gh1111"
| G_h1112 $\rightarrow$ "gh1112" | G_h1113 $\rightarrow$ "gh1113"
| G_h1122 $\rightarrow$ "gh1122" | G_h1123 $\rightarrow$ "gh1123"
| G_h1133 $\rightarrow$ "gh1133" | G_h1222 $\rightarrow$ "gh1222"
| G_h1223 $\rightarrow$ "gh1223" | G_h1233 $\rightarrow$ "gh1233"
| G_h1333 $\rightarrow$ "gh1333" | G_h2222 $\rightarrow$ "gh2222"
| G_h2223 $\rightarrow$ "gh2223" | G_h2233 $\rightarrow$ "gh2233"
| G_h2333 $\rightarrow$ "gh2333" | G_h3333 $\rightarrow$ "gh3333"
| G_h1uu $\rightarrow$ "gh1uu" | G_h2uu $\rightarrow$ "gh2uu"
| G_h3uu $\rightarrow$ "gh3uu" | G_h1uc $\rightarrow$ "gh1uc"
| G_h2uc $\rightarrow$ "gh2uc" | G_h3uc $\rightarrow$ "gh3uc"
| G_h1ut $\rightarrow$ "gh1ut" | G_h2ut $\rightarrow$ "gh2ut"
| G_h3ut $\rightarrow$ "gh3ut" | G_h1cu $\rightarrow$ "gh1cu"
| G_h2cu $\rightarrow$ "gh2cu" | G_h3cu $\rightarrow$ "gh3cu"
| G_h1cc $\rightarrow$ "gh1cc" | G_h2cc $\rightarrow$ "gh2cc"
| G_h3cc $\rightarrow$ "gh3cc" | G_h1ct $\rightarrow$ "gh1ct"
| G_h2ct $\rightarrow$ "gh2ct" | G_h3ct $\rightarrow$ "gh3ct"
| G_h1tu $\rightarrow$ "gh1tu" | G_h2tu $\rightarrow$ "gh2tu"
| G_h3tu $\rightarrow$ "gh3tu" | G_h1tc $\rightarrow$ "gh1tc"
| G_h2tc $\rightarrow$ "gh2tc" | G_h3tc $\rightarrow$ "gh3tc"
| G_h1tt $\rightarrow$ "gh1tt" | G_h2tt $\rightarrow$ "gh2tt"
| G_h3tt $\rightarrow$ "gh3tt"
| G_h1dd $\rightarrow$ "gh1dd" | G_h2dd $\rightarrow$ "gh2dd"
| G_h3dd $\rightarrow$ "gh3dd" | G_h1ds $\rightarrow$ "gh1ds"

```
| G_h2ds → "gh2ds" | G_h3ds → "gh3ds"
| G_h1db → "gh1db" | G_h2db → "gh2db"
| G_h3db → "gh3db" | G_h1sd → "gh1sd"
| G_h2sd → "gh2sd" | G_h3sd → "gh3sd"
| G_h1ss → "gh1ss" | G_h2ss → "gh2ss"
| G_h3ss → "gh3ss" | G_h1sb → "gh1sb"
| G_h2sb → "gh2sb" | G_h3sb → "gh3sb"
| G_h1bd → "gh1bd" | G_h2bd → "gh2bd"
| G_h3bd → "gh3bd" | G_h1bs → "gh1bs"
| G_h2bs → "gh2bs" | G_h3bs → "gh3bs"
| G_h1bb → "gh1bb" | G_h2bb → "gh2bb"
| G_h3bb → "gh3bb"
| G_hud → "ghud" | G_hus → "ghus"
| G_hub → "ghub" | G_hcd → "ghcd"
| G_hcs → "ghcs" | G_hcb → "ghcb"
| G_htd → "ghtd" | G_hts → "ghts"
| G_htb → "ghtb"
| G_hdu → "ghdu" | G_hdc → "ghdc"
| G_hdt → "ghdt" | G_hsu → "ghsu"
| G_hsc → "ghsc" | G_hst → "ghst"
| G_hbu → "ghbu" | G_hbc → "ghbc"
| G_hbt → "ghbt"
| G_he1n1 → "ghe1n1" | G_he1n2 → "ghe1n2"
| G_he1n3 → "ghe1n3" | G_he2n1 → "ghe2n1"
| G_he2n2 → "ghe2n2" | G_he2n3 → "ghe2n3"
| G_he3n1 → "ghe3n1" | G_he3n2 → "ghe3n2"
| G_he3n3 → "ghe3n3" | G_hn1e1 → "ghn1e1"
| G_hn1e2 → "ghn1e2" | G_hn1e3 → "ghn1e3"
| G_hn2e1 → "ghn2e1" | G_hn2e2 → "ghn2e2"
| G_hn2e3 → "ghn2e3" | G_hn3e1 → "ghn3e1"
| G_hn3e2 → "ghn3e2" | G_hn3e3 → "ghn3e3"
| G_h1e1e1 → "gh1e1e1" | G_h2e1e1 → "gh2e1e1"
| G_h3e1e1 → "gh3e1e1" | G_h1e1e2 → "gh1e1e2"
| G_h2e1e2 → "gh2e1e2" | G_h3e1e2 → "gh3e1e2"
| G_h1e1e3 → "gh1e1e3" | G_h2e1e3 → "gh2e1e3"
| G_h3e1e3 → "gh3e1e3" | G_h1e2e1 → "gh1e2e1"
| G_h2e2e1 → "gh2e2e1" | G_h3e2e1 → "gh3e2e1"
| G_h1e2e2 → "gh1e2e2" | G_h2e2e2 → "gh2e2e2"
| G_h3e2e2 → "gh3e2e2" | G_h1e2e3 → "gh1e2e3"
| G_h2e2e3 → "gh2e2e3" | G_h3e2e3 → "gh3e2e3"
| G_h1e3e1 → "gh1e3e1" | G_h2e3e1 → "gh2e3e1"
| G_h3e3e1 → "gh3e3e1" | G_h1e3e2 → "gh1e3e2"
| G_h2e3e2 → "gh2e3e2" | G_h3e3e2 → "gh3e3e2"
| G_h1e3e3 → "gh1e3e3" | G_h2e3e3 → "gh2e3e3"
| G_h3e3e3 → "gh3e3e3"
| Mass f → "mass" ^ flavor_symbol f
| Width f → "width" ^ flavor_symbol f
end
module type SSC_flags =
```

```
sig
  val higgs_triangle : bool (* Hγγ, Hgγ and Hgg couplings *)
  val higgs_hmm : bool
  val triple_anom : bool
  val quartic_anom : bool
  val higgs_anom : bool
  val k_matrix : bool
  val ckm_present : bool
  val top_anom : bool
  val top_anom_4f : bool
end

module SSC_kmatrix : SSC_flags =
struct
  let higgs_triangle = false
  let higgs_hmm = false
  let triple_anom = false
  let quartic_anom = true
  let higgs_anom = false
  let k_matrix = true
  let ckm_present = false
  let top_anom = false
  let top_anom_4f = false
end
```

13.6.4 Complete Minimal Standard Model including additional Resonances

```
module SSC (Flags : SSC_flags) =
struct
  let rcs = RCS.rename rcs_file "Modellib_BSM.SSC"
      [ "minimal␣electroweak␣standard␣model␣in␣unitarity␣gauge␣with␣additional␣Vectorboso
  open Coupling

  let default_width = ref Timelike
  let use_fudged_width = ref false

  let options = Options.create
    [ "constant_width", Arg.Unit (fun () → default_width := Constant),
      "use␣constant␣width␣(also␣in␣t-channel)";
      "fudged_width", Arg.Set use_fudged_width,
      "use␣fudge␣factor␣for␣charge␣particle␣width";
      "custom_width", Arg.String (fun f → default_width := Custom f),
      "use␣custom␣width";
      "cancel_widths", Arg.Unit (fun () → default_width := Vanishing),
      "use␣vanishing␣width"]

  type f_aux_top = TTGG | TBWA | TBWZ | TTWW | BBWW |
  let quartic_gauge =
    standard_quartic_gauge @ anomalous_quartic_gauge @ k_matrix_quartic_gauge
```

```
let standard_gauge_higgs =
  [ ((O H, G Wp, G Wm), Scalar_Vector_Vector 1, G_HWW);
    ((O H, G Z, G Z), Scalar_Vector_Vector 1, G_HZZ) ]
let standard_gauge_higgs4 =
  [ (O H, O H, G Wp, G Wm), Scalar2_Vector2 1, G_HHWW;
    (O H, O H, G Z, G Z), Scalar2_Vector2 1, G_HHZZ ]
let standard_higgs =
  [ (O H, O H, O H), Scalar_Scalar_Scalar 1, G_H3 ]
let standard_higgs4 =
  [ (O H, O H, O H, O H), Scalar4 1, G_H4 ]
```

WK's couplings (apparently, he still intends to divide by $\Lambda_{\text{EWSB}}^2 = 16\pi^2 v_{\text{F}}^2$):

$$\mathcal{L}_4^\tau = \left[(\partial_\mu H)(\partial^\mu H) + \frac{g^2 v_{\text{F}}^2}{4} V_\mu V^\mu\right]^2 \tag{13.53a}$$

$$\mathcal{L}_5^\tau = \left[(\partial_\mu H)(\partial_\nu H) + \frac{g^2 v_{\text{F}}^2}{4} V_\mu V_\nu\right]^2 \tag{13.53b}$$

with

$$V_\mu V_\nu = \frac{1}{2}\left(W_\mu^+ W_\nu^- + W_\nu^+ W_\mu^-\right) + \frac{1}{2\cos^2\theta_w} Z_\mu Z_\nu \tag{13.54}$$

(note the symmetrization!), i. e.

$$\mathcal{L}_4 = \alpha_4 \frac{g^4 v_{\text{F}}^4}{16} (V_\mu V_\nu)^2 \tag{13.55a}$$

$$\mathcal{L}_5 = \alpha_5 \frac{g^4 v_{\text{F}}^4}{16} (V_\mu V^\mu)^2 \tag{13.55b}$$

Breaking thinks up

$$\mathcal{L}_4^{\tau,H^4} = \left[(\partial_\mu H)(\partial^\mu H)\right]^2 \tag{13.56a}$$

$$\mathcal{L}_5^{\tau,H^4} = \left[(\partial_\mu H)(\partial^\mu H)\right]^2 \tag{13.56b}$$

and

$$\mathcal{L}_4^{\tau,H^2V^2} = \frac{g^2 v_{\text{F}}^2}{2} (\partial_\mu H)(\partial^\mu H) V_\mu V^\mu \tag{13.57a}$$

$$\mathcal{L}_5^{\tau,H^2V^2} = \frac{g^2 v_{\text{F}}^2}{2} (\partial_\mu H)(\partial_\nu H) V_\mu V_\nu \tag{13.57b}$$

i. e.

$$\mathcal{L}_4^{\tau,H^2V^2} = \frac{g^2 v_{\text{F}}^2}{2} \left[(\partial_\mu H)(\partial^\mu H) W_\nu^+ W^{-,\nu} + \frac{1}{2\cos^2\theta_w} (\partial_\mu H)(\partial^\mu H) Z_\nu Z^\nu\right] \tag{13.58a}$$

$$\mathcal{L}_5^{\tau,H^2V^2} = \frac{g^2 v_{\text{F}}^2}{2} \left[(W^{+,\mu}\partial_\mu H)(W^{-,\nu}\partial_\nu H) + \frac{1}{2\cos^2\theta_w} (Z^\mu \partial_\mu H)(Z^\nu \partial_\nu H)\right] \tag{13.58b}$$

$$
\begin{aligned}
\tau_8^4 \mathcal{L}_4^{\tau,H^2V^2} + \tau_8^5 \mathcal{L}_5^{\tau,H^2V^2} = \\
- \frac{g^2 v_{\mathrm{F}}^2}{2} \Bigg[2\tau_8^4 \frac{1}{2} (\mathrm{i}\partial_\mu H)(\mathrm{i}\partial^\mu H) W_\nu^+ W^{-,\nu} + \tau_8^5 (W^{+,\mu}\mathrm{i}\partial_\mu H)(W^{-,\nu}\mathrm{i}\partial_\nu H) \\
+ \frac{2\tau_8^4}{\cos^2\theta_w} \frac{1}{4} (\mathrm{i}\partial_\mu H)(\mathrm{i}\partial^\mu H) Z_\nu Z^\nu + \frac{\tau_8^5}{\cos^2\theta_w} \frac{1}{2} (Z^\mu \mathrm{i}\partial_\mu H)(Z^\nu \mathrm{i}\partial_\nu H) \Bigg]
\end{aligned} \tag{13.59}
$$

where the two powers of i make the sign conveniently negative, i. e.

$$\alpha^2_{(\partial H)^2 W^2} = \tau_8^4 g^2 v_{\mathrm{F}}^2 \tag{13.60a}$$

$$\alpha^2_{(\partial H W)^2} = \frac{\tau_8^5 g^2 v_{\mathrm{F}}^2}{2} \tag{13.60b}$$

$$\alpha^2_{(\partial H)^2 Z^2} = \frac{\tau_8^4 g^2 v_{\mathrm{F}}^2}{\cos^2\theta_w} \tag{13.60c}$$

$$\alpha^2_{(\partial H Z)^2} = \frac{\tau_8^5 g^2 v_{\mathrm{F}}^2}{2\cos^2\theta_w} \tag{13.60d}$$

```
let anomalous_gauge_higgs =
  [ (O H, G Ga, G Ga), Dim5_Scalar_Gauge2 1, G_HGaGa_anom;
    (O H, G Ga, G Z), Dim5_Scalar_Gauge2 1, G_HGaZ_anom;
    (O H, G Z, G Z), Dim5_Scalar_Gauge2 1, G_HZZ_anom;
    (O H, G Wp, G Wm), Dim5_Scalar_Gauge2 1, G_HWW_anom;
    (O H, G Ga, G Z), Dim5_Scalar_Vector_Vector_U 1, G_HGaZ_u;
    (O H, G Z, G Z), Dim5_Scalar_Vector_Vector_U 1, G_HZZ_u;
    (O H, G Wp, G Wm), Dim5_Scalar_Vector_Vector_U 1, G_HWW_u;
    (O H, G Wm, G Wp), Dim5_Scalar_Vector_Vector_U 1, G_HWW_u
  ]

let anomalous_gauge_higgs4 =
  []

let anomalous_higgs =
  []

let higgs_triangle_vertices =
  if Flags.higgs_triangle then
    [ (O H, G Ga, G Ga), Dim5_Scalar_Gauge2 1, G_HGaGa;
      (O H, G Ga, G Z), Dim5_Scalar_Gauge2 1, G_HGaZ;
      (O H, G Gl, G Gl), Dim5_Scalar_Gauge2 1, G_Hgg ]
  else
    []

let anomalous_higgs4 =
  []

let gauge_higgs =
  if Flags.higgs_anom then
    standard_gauge_higgs @ anomalous_gauge_higgs
  else
    standard_gauge_higgs

let gauge_higgs4 =
```

```
    if Flags.higgs_anom then
      standard_gauge_higgs4 @ anomalous_gauge_higgs4
    else
      standard_gauge_higgs4

  let higgs =
    if Flags.higgs_anom then
      standard_higgs @ anomalous_higgs
    else
      standard_higgs

  let higgs4 =
    if Flags.higgs_anom then
      standard_higgs4 @ anomalous_higgs4
    else
      standard_higgs4

  let goldstone_vertices =
    [ ((O Phi0, G Wm, G Wp), Scalar_Vector_Vector 1, I_G_ZWW);
      ((O Phip, G Ga, G Wm), Scalar_Vector_Vector 1, I_Q_W);
      ((O Phip, G Z, G Wm), Scalar_Vector_Vector 1, I_G_ZWW);
      ((O Phim, G Wp, G Ga), Scalar_Vector_Vector 1, I_Q_W);
      ((O Phim, G Wp, G Z), Scalar_Vector_Vector 1, I_G_ZWW) ]
```

New Resonances

There is an extra minus in the Lagrangian to have the same sign as HWW or HZZ vertex. Effectivly this doesn't matter for SSC, because $(-1)^2 = 1$. This is only for completeness.

$$\mathbf{V}_\mu = -\mathrm{i}g\mathbf{W}_\mu + \mathrm{i}g'\mathbf{B}_\mu \tag{13.61a}$$

$$\mathbf{W}_\mu = W_\mu^a \frac{\tau^a}{2} \tag{13.61b}$$

$$\mathbf{B}_\mu = W_\mu^a \frac{\tau^3}{2} \tag{13.61c}$$

$$\tau^{++} = \tau^+ \otimes \tau^+ \tag{13.61d}$$

$$\tau^+ = \frac{1}{2}\left(\tau^+ \otimes \tau^3 + \tau^3 + \tau^+\right) \tag{13.61e}$$

$$\tau^0 = \frac{1}{\sqrt{6}}\left(\tau^3 \otimes \tau^3 - \tau^+ \otimes \tau^- - \tau^- + \tau^+\right) \tag{13.61f}$$

$$\tau^- = \frac{1}{2}\left(\tau^- \otimes \tau^3 + \tau^3 + \tau^-\right) \tag{13.61g}$$

$$\tau^{--} = \tau^- \otimes \tau^- \tag{13.61h}$$

Scalar Isoscalar

$$\mathcal{L}_\sigma = -\frac{g_\sigma v}{2} \mathrm{tr}\left[\mathbf{V}_\mu \mathbf{V}^\mu\right] \sigma \tag{13.62}$$

```
  let rsigma3 =
    [ ((O Rsigma, G Wp, G Wm), Scalar_Vector_Vector 1, G_SWW);
      ((O Rsigma, G Z, G Z), Scalar_Vector_Vector 1, G_SZZ) ]
```

let $rsigma3t$ $=$
  $[\,((O\ Rsigma,\ G\ Wp,\ G\ Wm),\ Scalar\_Vector\_Vector\_t\ 1,\ G\_SWW\_T);$
    $((O\ Rsigma,\ G\ Z,\ G\ Z),\ Scalar\_Vector\_Vector\_t\ 1,\ G\_SZZ\_T)\,]$

let $rsigma4$ $=$
  $[\,(O\ Rsigma,\ O\ Rsigma,\ G\ Wp,\ G\ Wm),\ Scalar2\_Vector2\ 1,\ G\_SSWW;$
    $(O\ Rsigma,\ O\ Rsigma,\ G\ Z,\ G\ Z),\ Scalar2\_Vector2\ 1,\ G\_SSZZ\,]$

Scalar Isotensor

$$\mathcal{L}_\phi = \frac{g_\phi v}{4}\mathrm{Tr}\left[\left(\mathbf{V}_\mu \otimes \mathbf{V}^\mu - \frac{\tau^{aa}}{6}\mathrm{Tr}\left[\mathbf{V}_\mu\mathbf{V}^\mu\right]\right)\phi\right] \tag{13.63a}$$

$$\phi = \sqrt{2}\left(\phi^{++}\tau^{++} + \phi^{+}\tau^{+} + \phi^{0}\tau^{0} + \phi^{-}\tau^{-} + \phi^{--}\tau^{--}\right) \tag{13.63b}$$

let $rphi3$ $=$
  $[\,((O\ Rphin,\ G\ Wp,\ G\ Wm),\ Scalar\_Vector\_Vector\ 1,\ G\_PNWW);$
    $((O\ Rphin,\ G\ Z,\ G\ Z),\ Scalar\_Vector\_Vector\ 1,\ G\_PNZZ)\ ;$
    $((O\ Rphip,\ G\ Z,\ G\ Wm),\ Scalar\_Vector\_Vector\ 1,\ G\_PWZ)\ ;$
    $((O\ Rphipp,\ G\ Wm,\ G\ Wm),\ Scalar\_Vector\_Vector\ 1,\ G\_PWW)\ ;$
    $((O\ Rphim,\ G\ Wp,\ G\ Z),\ Scalar\_Vector\_Vector\ 1,\ G\_PWZ)\ ;$
    $((O\ Rphimm,\ G\ Wp,\ G\ Wp),\ Scalar\_Vector\_Vector\ 1,\ G\_PWW)\,]$

Tensor IsoScalar

let $rf3$ $=$
  $[\,((O\ Rf,\ G\ Wp,\ G\ Wm),\ Tensor\_2\_Vector\_Vector\_1\ 1,\ G\_FWW);$
    $((O\ Rf,\ G\ Z,\ G\ Z),\ Tensor\_2\_Vector\_Vector\_1\ 1,\ G\_FZZ)\,]$

let $rf3t$ $=$
  $[\,((O\ Rf,\ G\ Wp,\ G\ Wm),\ Tensor\_2\_Vector\_Vector\_t\ 1,\ G\_FWW\_T);$
    $((O\ Rf,\ G\ Z,\ G\ Z),\ Tensor\_2\_Vector\_Vector\_t\ 1,\ G\_FZZ\_T)\,]$

Tensor Isotensor

$$\mathcal{L}_t \tag{13.64a}$$

let $rt3$ $=$
  $[\,((O\ Rtn,\ G\ Wp,\ G\ Wm),\ Tensor\_2\_Vector\_Vector\_1\ 1,\ G\_TNWW);$
    $((O\ Rtn,\ G\ Z,\ G\ Z),\ Tensor\_2\_Vector\_Vector\_1\ 1,\ G\_TNZZ)\ ;$
    $((O\ Rtp,\ G\ Z,\ G\ Wm),\ Tensor\_2\_Vector\_Vector\_1\ 1,\ G\_TWZ)\ ;$
    $((O\ Rtpp,\ G\ Wm,\ G\ Wm),\ Tensor\_2\_Vector\_Vector\_1\ 1,\ G\_TWW)\ ;$
    $((O\ Rtm,\ G\ Wp,\ G\ Z),\ Tensor\_2\_Vector\_Vector\_1\ 1,\ G\_TWZ)\ ;$
    $((O\ Rtmm,\ G\ Wp,\ G\ Wp),\ Tensor\_2\_Vector\_Vector\_1\ 1,\ G\_TWW)\,]$

Anomalous trilinear interactions $f_i f_j V$ and ttH:

$$\Delta\mathcal{L}_{tt\gamma} = -e\frac{v}{\Lambda^2}\bar{t}i\sigma^{\mu\nu}k_\nu(d_V(k^2) + id_A(k^2)\gamma_5)tA_\mu \tag{13.65}$$

let $anomalous\_ttA$ $=$
  if $Flags.top\_anom$ then
    $[\,((M\ (U\ (-3)),\ G\ Ga,\ M\ (U\ 3)),\ FBF\ (1,\ Psibar,\ TVAM,\ Psi),\ G\_TVA\_ttA)\,]$
  else
    $[\,]$

$$\Delta\mathcal{L}_{bb\gamma} = -e\frac{v}{\Lambda^2}\bar{b}i\sigma^{\mu\nu}k_\nu(d_V(k^2) + id_A(k^2)\gamma_5)bA_\mu \tag{13.66}$$

```
let anomalous_bbA =
  if Flags.top_anom then
    [((M (D (-3)), G Ga, M (D 3)), FBF (1, Psibar, TVAM, Psi), G_TVA_bbA)]
  else
    []
```

$$\Delta\mathcal{L}_{ttg} = -g_s\frac{v}{\Lambda^2}\bar{t}\lambda^a i\sigma^{\mu\nu}k_\nu(d_V(k^2) + id_A(k^2)\gamma_5)tG^a_\mu \tag{13.67}$$

```
let anomalous_ttG =
  if Flags.top_anom then
    [((M (U (-3)), G Gl, M (U 3)), FBF (1, Psibar, TVAM, Psi), G_TVA_ttG)]
  else
    []
```

$$\Delta\mathcal{L}_{ttZ} = -\frac{g}{2c_W}\frac{v^2}{\Lambda^2}\left[\bar{t}\not{Z}(X_L(k^2)P_L + X_R(k^2)P_R)t + \bar{t}\frac{i\sigma^{\mu\nu}k_\nu}{m_Z}(d_V(k^2) + id_A(k^2)\gamma_5)tZ_\mu\right] \tag{13.68}$$

```
let anomalous_ttZ =
  if Flags.top_anom then
    [((M (U (-3)), G Z, M (U 3)), FBF (1, Psibar, VLRM, Psi), G_VLR_ttZ);
      ((M (U (-3)), G Z, M (U 3)), FBF (1, Psibar, TVAM, Psi), G_TVA_ttZ)]
  else
    []
```

$$\Delta\mathcal{L}_{bbZ} = -\frac{g}{2c_W}\frac{v^2}{\Lambda^2}\bar{b}\frac{i\sigma^{\mu\nu}k_\nu}{m_Z}(d_V(k^2) + id_A(k^2)\gamma_5)bZ_\mu \tag{13.69}$$

```
let anomalous_bbZ =
  if Flags.top_anom then
    [((M (D (-3)), G Z, M (D 3)), FBF (1, Psibar, TVAM, Psi), G_TVA_bbZ)]
  else
    []
```

$$\Delta\mathcal{L}_{tbW} = -\frac{g}{\sqrt{2}}\frac{v^2}{\Lambda^2}\left[\bar{b}\not{W}^-(V_L(k^2)P_L + V_R(k^2)P_R)t + \bar{b}\frac{i\sigma^{\mu\nu}k_\nu}{m_W}(g_L(k^2)P_L + g_R(k^2)P_R)tW^-_\mu\right] + \text{H.c.} \tag{13.70}$$

```
let anomalous_tbW =
  if Flags.top_anom then
    [((M (D (-3)), G Wm, M (U 3)), FBF (1, Psibar, VLRM, Psi), G_VLR_btW);
      ((M (U (-3)), G Wp, M (D 3)), FBF (1, Psibar, VLRM, Psi), G_VLR_tbW);
      ((M (D (-3)), G Wm, M (U 3)), FBF (1, Psibar, TLRM, Psi), G_TLR_btW);
      ((M (U (-3)), G Wp, M (D 3)), FBF (1, Psibar, TRLM, Psi), G_TRL_tbW)]
  else
```

```
    []
```

$$\Delta\mathcal{L}_{ttH} = -\frac{1}{\sqrt{2}}\bar{t}(Y_V(k^2) + iY_A(k^2)\gamma_5)tH \tag{13.71}$$

```
let anomalous_ttH =
  if Flags.top_anom then
    [ ((M (U (−3)), O H, M (U 3)), FBF (1, Psibar, SPM, Psi), G_SP_ttH) ]
  else
    []
```

quartic fermion-gauge interactions $f_i f_j V_1 V_2$ emerging from gauge-invariant effective operators:

$$\Delta\mathcal{L}_{ttgg} = -\frac{g_s^2}{2} f_{abc} \frac{\upsilon}{\Lambda^2} \bar{t}\lambda^a \sigma^{\mu\nu}(d_V(k^2) + id_A(k^2)\gamma_5)tG_\mu^b G_\nu^c \tag{13.72}$$

```
let anomalous_ttGG =
  if Flags.top_anom then
    [ ((M (U (−3)), O (Aux_top (2, 1, 0,true,TTGG)), M (U 3)), FBF (1, Psibar, TVA, Psi), G_TVA
      ((O (Aux_top (2, 1, 0,false,TTGG)), G Gl, G Gl), Aux_Gauge_Gauge 1, I_Gs) ]
  else
    []
```

$$\Delta\mathcal{L}_{tbWA} = -i\sin\theta_w \frac{g^2}{2\sqrt{2}} \frac{\upsilon^2}{\Lambda^2} \left[\bar{b}\frac{\sigma^{\mu\nu}}{m_W}(g_L(k^2)P_L + g_R(k^2)P_R)tA_\mu W_\nu^-\right] + \text{H.c.} \tag{13.73}$$

```
let anomalous_tbWA =
  if Flags.top_anom then
    [ ((M (D (−3)), O (Aux_top (2, 0, −1,true,TBWA)), M (U 3)), FBF (1, Psibar, TLR, Psi), G_TL
      ((O (Aux_top (2, 0, 1,false,TBWA)), G Ga, G Wm), Aux_Gauge_Gauge 1, I_G_weak);
      ((M (U (−3)), O (Aux_top (2, 0, 1,true,TBWA)), M (D 3)), FBF (1, Psibar, TRL, Psi), G_TR
      ((O (Aux_top (2, 0, −1,false,TBWA)), G Wp, G Ga), Aux_Gauge_Gauge 1, I_G_weak) ]
  else
    []
```

$$\Delta\mathcal{L}_{tbWZ} = -i\cos\theta_w \frac{g^2}{2\sqrt{2}} \frac{\upsilon^2}{\Lambda^2} \left[\bar{b}\frac{\sigma^{\mu\nu}}{m_W}(g_L(k^2)P_L + g_R(k^2)P_R)tZ_\mu W_\nu^-\right] + \text{H.c.} \tag{13.74}$$

```
let anomalous_tbWZ =
  if Flags.top_anom then
    [ ((M (D (−3)), O (Aux_top (2, 0, −1,true,TBWZ)), M (U 3)), FBF (1, Psibar, TLR, Psi), G_TL
      ((O (Aux_top (2, 0, 1,false,TBWZ)), G Z, G Wm), Aux_Gauge_Gauge 1, I_G_weak);
      ((M (U (−3)), O (Aux_top (2, 0, 1,true,TBWZ)), M (D 3)), FBF (1, Psibar, TRL, Psi), G_TR
      ((O (Aux_top (2, 0, −1,false,TBWZ)), G Wp, G Z), Aux_Gauge_Gauge 1, I_G_weak) ]
  else
    []
```

$$\Delta\mathcal{L}_{ttWW} = -i\frac{g^2}{2}\frac{\upsilon^2}{\Lambda^2}\bar{t}\frac{\sigma^{\mu\nu}}{m_W}(d_V(k^2) + id_A(k^2)\gamma_5)tW^-_\mu W^+_\nu \tag{13.75}$$

```
let anomalous_ttWW =
  if Flags.top_anom then
    [ ((M (U (−3)), O (Aux_top (2, 0, 0,true,TTWW)), M (U 3)), FBF (1, Psibar, TVA, Psi), G_TVA
      ((O (Aux_top (2, 0, 0,false,TTWW)), G Wm, G Wp), Aux_Gauge_Gauge 1, I_G_weak) ]
  else
    []
```

$$\Delta\mathcal{L}_{bbWW} = -i\frac{g^2}{2}\frac{\upsilon^2}{\Lambda^2}\bar{b}\frac{\sigma^{\mu\nu}}{m_W}(d_V(k^2) + id_A(k^2)\gamma_5)bW^-_\mu W^+_\nu \tag{13.76}$$

```
let anomalous_bbWW =
  if Flags.top_anom then
    [ ((M (D (−3)), O (Aux_top (2, 0, 0,true,BBWW)), M (D 3)), FBF (1, Psibar, TVA, Psi), G_TVA
      ((O (Aux_top (2, 0, 0,false,BBWW)), G Wm, G Wp), Aux_Gauge_Gauge 1, I_G_weak) ]
  else
    []
```

4-fermion contact terms emerging from operator rewriting:

```
let anomalous_top_qGuG_tt =
  [ ((M (U (−3)), O (Aux_top (1, 1, 0,true,QGUG)), M (U 3)), FBF (1, Psibar, VLR, Psi), G_VLR_q

let anomalous_top_qGuG_ff n =
  List.map mom
    [ ((U (−n), Aux_top (1, 1, 0,false,QGUG), U n), FBF (1, Psibar, V, Psi), Unit);
      ((D (−n), Aux_top (1, 1, 0,false,QGUG), D n), FBF (1, Psibar, V, Psi), Unit) ]

let anomalous_top_qGuG =
  if Flags.top_anom_4f then
    anomalous_top_qGuG_tt @ ThoList.flatmap anomalous_top_qGuG_ff [1; 2; 3]
  else
    []

let anomalous_top_qBuB_tt =
  [ ((M (U (−3)), O (Aux_top (1, 0, 0,true,QBUB)), M (U 3)), FBF (1, Psibar, VLR, Psi), G_VLR_q

let anomalous_top_qBuB_ff n =
  List.map mom
    [ ((U (−n), Aux_top (1, 0, 0,false,QBUB), U n), FBF (1, Psibar, VLR, Psi), G_VLR_qBuB_u);
      ((D (−n), Aux_top (1, 0, 0,false,QBUB), D n), FBF (1, Psibar, VLR, Psi), G_VLR_qBuB_d);
      ((L (−n), Aux_top (1, 0, 0,false,QBUB), L n), FBF (1, Psibar, VLR, Psi), G_VLR_qBuB_e);
      ((N (−n), Aux_top (1, 0, 0,false,QBUB), N n), FBF (1, Psibar, VL, Psi), G_VL_qBuB_n) ]

let anomalous_top_qBuB =
  if Flags.top_anom_4f then
    anomalous_top_qBuB_tt @ ThoList.flatmap anomalous_top_qBuB_ff [1; 2; 3]
  else
    []

let anomalous_top_qW_tq =
```

```
    [ ((M (U (−3)), O (Aux_top (1,0,0,true,QW)), M (U 3)), FBF (1, Psibar, VL, Psi), G_VL_qW);
      ((M (D (−3)), O (Aux_top (1,0,−1,true,QW)), M (U 3)), FBF (1, Psibar, VL, Psi), G_VL_qW)
      ((M (U (−3)), O (Aux_top (1,0,1,true,QW)), M (D 3)), FBF (1, Psibar, VL, Psi), G_VL_qW) ]

  let anomalous_top_qW_ff n =
    List.map mom
      [ ((U (−n), Aux_top (1,0,0,false,QW), U n), FBF (1, Psibar, VL, Psi), G_VL_qW_u);
        ((D (−n), Aux_top (1,0,0,false,QW), D n), FBF (1, Psibar, VL, Psi), G_VL_qW_d);
        ((N (−n), Aux_top (1,0,0,false,QW), N n), FBF (1, Psibar, VL, Psi), G_VL_qW_u);
        ((L (−n), Aux_top (1,0,0,false,QW), L n), FBF (1, Psibar, VL, Psi), G_VL_qW_d);
        ((D (−n), Aux_top (1,0,−1,false,QW), U n), FBF (1, Psibar, VL, Psi), Half);
        ((U (−n), Aux_top (1,0,1,false,QW), D n), FBF (1, Psibar, VL, Psi), Half);
        ((L (−n), Aux_top (1,0,−1,false,QW), N n), FBF (1, Psibar, VL, Psi), Half);
        ((N (−n), Aux_top (1,0,1,false,QW), L n), FBF (1, Psibar, VL, Psi), Half) ]

  let anomalous_top_qW =
    if Flags.top_anom_4f then
      anomalous_top_qW_tq @ ThoList.flatmap anomalous_top_qW_ff [1;2;3]
    else
      []

  let anomalous_top_DuDd =
    if Flags.top_anom_4f then
      [ ((M (U (−3)), O (Aux_top (0,0,0,true,DR)), M (U 3)), FBF (1, Psibar, SR, Psi), Half);
        ((M (U (−3)), O (Aux_top (0,0,0,false,DR)), M (U 3)), FBF (1, Psibar, SL, Psi), G_SL_DttR
        ((M (D (−3)), O (Aux_top (0,0,0,false,DR)), M (D 3)), FBF (1, Psibar, SR, Psi), G_SR_DttR
        ((M (U (−3)), O (Aux_top (0,0,0,true,DL)), M (U 3)), FBF (1, Psibar, SL, Psi), Half);
        ((M (D (−3)), O (Aux_top (0,0,0,false,DL)), M (D 3)), FBF (1, Psibar, SL, Psi), G_SL_DttL)
        ((M (D (−3)), O (Aux_top (0,0,−1,true,DR)), M (U 3)), FBF (1, Psibar, SR, Psi), Half);
        ((M (U (−3)), O (Aux_top (0,0,1,false,DR)), M (D 3)), FBF (1, Psibar, SLR, Psi), G_SLR_D
        ((M (D (−3)), O (Aux_top (0,0,−1,true,DL)), M (U 3)), FBF (1, Psibar, SL, Psi), Half);
        ((M (U (−3)), O (Aux_top (0,0,1,false,DL)), M (D 3)), FBF (1, Psibar, SL, Psi), G_SL_DbtL)
    else
      []

  let vertices3 =
    (ThoList.flatmap electromagnetic_currents [1;2;3] @
     ThoList.flatmap color_currents [1;2;3] @
     ThoList.flatmap neutral_currents [1;2;3] @
     (if Flags.ckm_present then
        charged_currents_ckm
      else
        charged_currents_triv) @
     yukawa @ triple_gauge @
     gauge_higgs @ higgs @ higgs_triangle_vertices
     @ goldstone_vertices @
     rsigma3 @ rsigma3t @ rphi3 @ rf3 @ rf3t @ rt3 @
     anomalous_ttA @ anomalous_bbA @
     anomalous_ttZ @ anomalous_bbZ @
     anomalous_tbW @ anomalous_tbWA @ anomalous_tbWZ @
     anomalous_ttWW @ anomalous_bbWW @
     anomalous_ttG @ anomalous_ttGG @
```

```
    anomalous_ttH @
    anomalous_top_qGuG @ anomalous_top_qBuB @
    anomalous_top_qW @ anomalous_top_DuDd)

let vertices4 =
  quartic_gauge @ gauge_higgs4 @ higgs4 @ rsigma4

let vertices () = (vertices3, vertices4, [])
```

For efficiency, make sure that *F.of_vertices vertices* is evaluated only once.

```
let table = F.of_vertices (vertices ())
let fuse2 = F.fuse2 table
let fuse3 = F.fuse3 table
let fuse = F.fuse table
let max_degree () = 4

let flavor_of_string = function
  | "e-" → M (L 1) | "e+" → M (L (−1))
  | "mu-" → M (L 2) | "mu+" → M (L (−2))
  | "tau-" → M (L 3) | "tau+" → M (L (−3))
  | "nue" → M (N 1) | "nuebar" → M (N (−1))
  | "numu" → M (N 2) | "numubar" → M (N (−2))
  | "nutau" → M (N 3) | "nutaubar" → M (N (−3))
  | "u" → M (U 1) | "ubar" → M (U (−1))
  | "c" → M (U 2) | "cbar" → M (U (−2))
  | "t" → M (U 3) | "tbar" → M (U (−3))
  | "d" → M (D 1) | "dbar" → M (D (−1))
  | "s" → M (D 2) | "sbar" → M (D (−2))
  | "b" → M (D 3) | "bbar" → M (D (−3))
  | "g" | "gl" → G Gl
  | "A" → G Ga | "Z" | "Z0" → G Z
  | "W+" → G Wp | "W-" → G Wm
  | "H" → O H | "Rsigma" → O Rsigma
  | "Rphi0" → O Rphin
  | "Rphi+" → O Rphip | "Rphi-" → O Rphim
  | "Rphi++" → O Rphip | "Rphi--" → O Rphimm
  | "Rf" → O Rf
  | "Rt0" → O Rtn
  | "Rt+" → O Rtp | "Rt-" → O Rtm
  | "Rt++" → O Rtp | "Rt--" → O Rtmm
  | "Aux_t_ttGG0" → O (Aux_top (2,1, 0,true,TTGG)) | "Aux_ttGG0" →
O (Aux_top (2,1, 0,false,TTGG))
  | "Aux_t_tbWA+" → O (Aux_top (2,0, 1,true,TBWA)) | "Aux_tbWA+" →
O (Aux_top (2,0, 1,false,TBWA))
  | "Aux_t_tbWA-" → O (Aux_top (2,0, −1,true,TBWA)) | "Aux_tbWA-" →
O (Aux_top (2,0, −1,false,TBWA))
  | "Aux_t_tbWZ+" → O (Aux_top (2,0, 1,true,TBWZ)) | "Aux_tbWZ+" →
O (Aux_top (2,0, 1,false,TBWZ))
  | "Aux_t_tbWZ-" → O (Aux_top (2,0, −1,true,TBWZ)) | "Aux_tbWZ-" →
O (Aux_top (2,0, −1,false,TBWZ))
  | "Aux_t_ttWW0" → O (Aux_top (2,0, 0,true,TTWW)) | "Aux_ttWW0" →
O (Aux_top (2,0, 0,false,TTWW))
```

```
| "Aux_t_bbWW0" → O (Aux_top (2,0, 0,true,BBWW)) | "Aux_bbWW0" →
O (Aux_top (2,0, 0,false,BBWW))
  | "Aux_t_qGuG0" → O (Aux_top (1,1, 0,true,QGUG)) | "Aux_qGuG0" →
O (Aux_top (1,1, 0,false,QGUG))
  | "Aux_t_qBuB0" → O (Aux_top (1,0, 0,true,QBUB)) | "Aux_qBuB0" →
O (Aux_top (1,0, 0,false,QBUB))
  | "Aux_t_qW0" → O (Aux_top (1,0, 0,true,QW)) | "Aux_qW0" →
O (Aux_top (1,0, 0,false,QW))
  | "Aux_t_qW+" → O (Aux_top (1,0, 1,true,QW)) | "Aux_qW+" →
O (Aux_top (1,0, 1,false,QW))
  | "Aux_t_qW-" → O (Aux_top (1,0,−1,true,QW)) | "Aux_qW-" →
O (Aux_top (1,0,−1,false,QW))
  | "Aux_t_dL0" → O (Aux_top (0,0, 0,true,DL)) | "Aux_dL0" →
O (Aux_top (0,0, 0,false,DL))
  | "Aux_t_dL+" → O (Aux_top (0,0, 1,true,DL)) | "Aux_dL+" →
O (Aux_top (0,0, 1,false,DL))
  | "Aux_t_dL-" → O (Aux_top (0,0,−1,true,DL)) | "Aux_dL-" →
O (Aux_top (0,0,−1,false,DL))
  | "Aux_t_dR0" → O (Aux_top (0,0, 0,true,DR)) | "Aux_dR0" →
O (Aux_top (0,0, 0,false,DR))
  | "Aux_t_dR+" → O (Aux_top (0,0, 1,true,DR)) | "Aux_dR+" →
O (Aux_top (0,0, 1,false,DR))
  | "Aux_t_dR-" → O (Aux_top (0,0,−1,true,DR)) | "Aux_dR-" →
O (Aux_top (0,0,−1,false,DR))
  | _ → invalid_arg "Modellib_BSM.SSC.flavor_of_string"

let flavor_to_string = function
  | M f →
      begin match f with
      | L 1 → "e-" | L (−1) → "e+"
      | L 2 → "mu-" | L (−2) → "mu+"
      | L 3 → "tau-" | L (−3) → "tau+"
      | L _ → invalid_arg
            "Modellib_BSM.SSC.flavor_to_string:␣invalid␣lepton"
      | N 1 → "nue" | N (−1) → "nuebar"
      | N 2 → "numu" | N (−2) → "numubar"
      | N 3 → "nutau" | N (−3) → "nutaubar"
      | N _ → invalid_arg
            "Modellib_BSM.SSC.flavor_to_string:␣invalid␣neutrino"
      | U 1 → "u" | U (−1) → "ubar"
      | U 2 → "c" | U (−2) → "cbar"
      | U 3 → "t" | U (−3) → "tbar"
      | U _ → invalid_arg
            "Modellib_BSM.SSC.flavor_to_string:␣invalid␣up␣type␣quark"
      | D 1 → "d" | D (−1) → "dbar"
      | D 2 → "s" | D (−2) → "sbar"
      | D 3 → "b" | D (−3) → "bbar"
      | D _ → invalid_arg
            "Modellib_BSM.SSC.flavor_to_string:␣invalid␣down␣type␣quark"
      end
```

```
| G f →
    begin match f with
    | Gl → "gl"
    | Ga → "A" | Z → "Z"
    | Wp → "W+" | Wm → "W-"
    end
| O f →
    begin match f with
    | Phip → "phi+" | Phim → "phi-" | Phi0 → "phi0"
    | H → "H" | Rsigma → "Rsigma"
    | Rphin → "Rphin" | Rphip → "Rphi+" | Rphim → "Rphi-"
    | Rphipp → "Rphi++" | Rphimm → "Rphi--"
    | Rf → "Rf"
    | Rtn → "Rtn" | Rtp → "Rt+" | Rtm → "Rt-"
    | Rtpp → "Rt++" | Rtmm → "Rt--"
    | Aux_top (_, _, ch, n, v) → "Aux_" ^ (if n then "t_" else "") ^ (
        begin match v with
        | TTGG → "ttGG" | TBWA → "tbWA" | TBWZ →
"tbWZ"
        | TTWW → "ttWW" | BBWW → "bbWW"
        | QGUG → "qGuG" | QBUB → "qBuB"
        | QW → "qW" | DL → "dL" | DR → "dR"
        end ) ^ ( if ch > 0 then "+" else if ch < 0 then "-" else "0" )
    end

let flavor_to_TeX = function
  | M f →
    begin match f with
    | L 1 → "e^-" | L (−1) → "e^+"
    | L 2 → "\\mu^-" | L (−2) → "\\mu^+"
    | L 3 → "\\tau^-" | L (−3) → "\\tau^+"
    | L _ → invalid_arg
        "Modellib_BSM.SSC.flavor_to_TeX:␣invalid␣lepton"
    | N 1 → "\\nu_e" | N (−1) → "\\bar{\\nu}_e"
    | N 2 → "\\nu_\\mu" | N (−2) → "\\bar{\\nu}_\\mu"
    | N 3 → "\\nu_\\tau" | N (−3) → "\\bar{\\nu}_\\tau"
    | N _ → invalid_arg
        "Modellib_BSM.SSC.flavor_to_TeX:␣invalid␣neutrino"
    | U 1 → "u" | U (−1) → "\\bar{u}"
    | U 2 → "c" | U (−2) → "\\bar{c}"
    | U 3 → "t" | U (−3) → "\\bar{t}"
    | U _ → invalid_arg
        "Modellib_BSM.SSC.flavor_to_TeX:␣invalid␣up␣type␣quark"
    | D 1 → "d" | D (−1) → "\\bar{d}"
    | D 2 → "s" | D (−2) → "\\bar{s}"
    | D 3 → "b" | D (−3) → "\\bar{b}"
    | D _ → invalid_arg
        "Modellib_BSM.SSC.flavor_to_TeX:␣invalid␣down␣type␣quark"
    end
  | G f →
```

```
        begin match f with
        | Gl → "g"
        | Ga → "\\gamma" | Z → "Z"
        | Wp → "W^+" | Wm → "W^-"
        end
    | O f →
        begin match f with
        | Phip → "\\phi^+" | Phim → "\\phi^-" | Phi0 →
"\\phi^0"
        | H → "H" | Rsigma → "\\sigma"
        | Rphip → "\\phi^+" | Rphim → "\\phi^-" | Rphin →
"\\phi^0"
        | Rphipp → "\\phi^{++}" | Rphimm → "\\phi^{--}"
        | Rf → "f"
        | Rtp → "t^+" | Rtm → "t^-" | Rtn → "t^0"
        | Rtpp → "t^{++}" | Rtmm → "t^{--}"
        | Aux_top (_, _, ch, n, v) → "\\textnormal{Aux_" ^ (if n then "t_" else "") ^ (
            begin match v with
            | TTGG → "ttGG" | TBWA → "tbWA" | TBWZ →
"tbWZ"
            | TTWW → "ttWW" | BBWW → "bbWW"
            | QGUG → "qGuG" | QBUB → "qBuB"
            | QW → "qW" | DL → "dL" | DR → "dR"
            end ) ^ ( if ch > 0 then "^+" else if ch < 0 then "^-" else "^0" ) ^ "}"
        end

  let flavor_symbol = function
    | M f →
        begin match f with
        | L n when n > 0 → "l" ^ string_of_int n
        | L n → "l" ^ string_of_int (abs n) ^ "b"
        | N n when n > 0 → "n" ^ string_of_int n
        | N n → "n" ^ string_of_int (abs n) ^ "b"
        | U n when n > 0 → "u" ^ string_of_int n
        | U n → "u" ^ string_of_int (abs n) ^ "b"
        | D n when n > 0 → "d" ^ string_of_int n
        | D n → "d" ^ string_of_int (abs n) ^ "b"
        end
    | G f →
        begin match f with
        | Gl → "gl"
        | Ga → "a" | Z → "z"
        | Wp → "wp" | Wm → "wm"
        end
    | O f →
        begin match f with
        | Phip → "pp" | Phim → "pm" | Phi0 → "p0"
        | H → "h" | Rsigma → "rsi"
        | Rphip → "rpp" | Rphim → "rpm" | Rphin → "rpn"
        | Rphipp → "rppp" | Rphimm → "rpmm"
```

```
      | Rf → "rf"
      | Rtp → "rtp" | Rtm → "rtm" | Rtn → "rtn"
      | Rtpp → "rtpp" | Rtmm → "rtmm"
      | Aux_top (_, _, ch, n, v) → "aux_" ^ (if n then "t_" else "") ^ (
          begin match v with
          | TTGG → "ttgg" | TBWA → "tbwa" | TBWZ →
"tbwz"
          | TTWW → "ttww" | BBWW → "bbww"
          | QGUG → "qgug" | QBUB → "qbub"
          | QW → "qw" | DL → "dl" | DR → "dr"
          end ) ^ "_" ^ ( if ch > 0 then "p" else if ch < 0 then "m" else "0" )
      end
```

Introducing new Resonances from 45, there are no PDG values

```
let pdg = function
  | M f →
      begin match f with
      | L n when n > 0 → 9 + 2 × n
      | L n → - 9 + 2 × n
      | N n when n > 0 → 10 + 2 × n
      | N n → - 10 + 2 × n
      | U n when n > 0 → 2 × n
      | U n → 2 × n
      | D n when n > 0 → - 1 + 2 × n
      | D n → 1 + 2 × n
      end
  | G f →
      begin match f with
      | Gl → 21
      | Ga → 22 | Z → 23
      | Wp → 24 | Wm → (−24)
      end
  | O f →
      begin match f with
      | Phip | Phim → 27 | Phi0 → 26
      | H → 25 | Rsigma → 45
      | Rphin → 46 | Rphip | Rphim → 47
      | Rphipp | Rphimm → 48
      | Rf → 52
      | Rtn → 53 | Rtp | Rtm → 54
      | Rtpp | Rtmm → 55
      | Aux_top (_, _, _, _, _) → 81
      end

let mass_symbol f =
  "mass(" ^ string_of_int (abs (pdg f)) ^ ")"

let width_symbol f =
  "width(" ^ string_of_int (abs (pdg f)) ^ ")"

let constant_symbol = function
  | Unit → "unit" | Half → "half" | Pi → "PI"
```

```
| Alpha_QED → "alpha" | E → "e" | G_weak → "g" | Vev → "vev"
| I_G_weak → "ig"
| Sin2thw → "sin2thw" | Sinthw → "sinthw" | Costhw → "costhw"
| Q_lepton → "qlep" | Q_up → "qup" | Q_down → "qdwn"
| G_NC_lepton → "gnclep" | G_NC_neutrino → "gncneu"
| G_NC_up → "gncup" | G_NC_down → "gncdwn"
| G_TVA_ttA → "gtva_tta" | G_TVA_bbA → "gtva_bba"
| G_VLR_ttZ → "gvlr_ttz" | G_TVA_ttZ → "gtva_ttz" | G_TVA_bbZ → "gtva_bbz"
| G_VLR_btW → "gvlr_btw" | G_VLR_tbW → "gvlr_tbw"
| G_TLR_btW → "gtlr_btw" | G_TRL_tbW → "gtrl_tbw"
| G_TLR_btWA → "gtlr_btwa" | G_TRL_tbWA → "gtrl_tbwa"
| G_TLR_btWZ → "gtlr_btwz" | G_TRL_tbWZ → "gtrl_tbwz"
| G_TVA_ttWW → "gtva_ttww" | G_TVA_bbWW → "gtva_bbww"
| G_TVA_ttG → "gtva_ttg" | G_TVA_ttGG → "gtva_ttgg"
| G_SP_ttH → "gsp_tth"
| G_VLR_qGuG → "gvlr_qgug"
| G_VLR_qBuB → "gvlr_qbub"
| G_VLR_qBuB_u → "gvlr_qbub_u" | G_VLR_qBuB_d → "gvlr_qbub_d"
| G_VLR_qBuB_e → "gvlr_qbub_e" | G_VL_qBuB_n → "gvl_qbub_n"
| G_VL_qW → "gvl_qw"
| G_VL_qW_u → "gvl_qw_u" | G_VL_qW_d → "gvl_qw_d"
| G_SL_DttR → "gsl_dttr" | G_SR_DttR → "gsr_dttr" | G_SL_DttL → "gsl_dttl"
| G_SLR_DbtR → "gslr_dbtr" | G_SL_DbtL → "gsl_dbtl"
| G_CC → "gcc"
| G_CCQ (n1, n2) → "gccq" ^ string_of_int n1 ^ string_of_int n2
| I_Q_W → "iqw" | I_G_ZWW → "igzww"
| G_WWWW → "gw4" | G_ZZWW → "gzzww"
| G_AZWW → "gazww" | G_AAWW → "gaaww"
| I_G1_AWW → "ig1a" | I_G1_ZWW → "ig1z"
| I_G1_plus_kappa_plus_G4_AWW → "ig1pkpg4a"
| I_G1_plus_kappa_plus_G4_ZWW → "ig1pkpg4z"
| I_G1_plus_kappa_minus_G4_AWW → "ig1pkmg4a"
| I_G1_plus_kappa_minus_G4_ZWW → "ig1pkmg4z"
| I_G1_minus_kappa_plus_G4_AWW → "ig1mkpg4a"
| I_G1_minus_kappa_plus_G4_ZWW → "ig1mkpg4z"
| I_G1_minus_kappa_minus_G4_AWW → "ig1mkmg4a"
| I_G1_minus_kappa_minus_G4_ZWW → "ig1mkmg4z"
| I_lambda_AWW → "ila"
| I_lambda_ZWW → "ilz"
| G5_AWW → "rg5a"
| G5_ZWW → "rg5z"
| I_kappa5_AWW → "ik5a"
| I_kappa5_ZWW → "ik5z"
| I_lambda5_AWW → "il5a" | I_lambda5_ZWW → "il5z"
| Alpha_WWWW0 → "alww0" | Alpha_WWWW2 → "alww2"
| Alpha_ZZWW0 → "alzw0" | Alpha_ZZWW1 → "alzw1"
```

```
    | Alpha_ZZZZ → "alzz"
    | D_Alpha_ZZWW0_S → "dalzz0_s(gkm,mkm,"
    | D_Alpha_ZZWW0_T → "dalzz0_t(gkm,mkm,"
    | D_Alpha_ZZWW1_S → "dalzz1_s(gkm,mkm,"
    | D_Alpha_ZZWW1_T → "dalzz1_t(gkm,mkm,"
    | D_Alpha_ZZWW1_U → "dalzz1_u(gkm,mkm,"
    | D_Alpha_WWWW0_S → "dalww0_s(gkm,mkm,"
    | D_Alpha_WWWW0_T → "dalww0_t(gkm,mkm,"
    | D_Alpha_WWWW0_U → "dalww0_u(gkm,mkm,"
    | D_Alpha_WWWW2_S → "dalww2_s(gkm,mkm,"
    | D_Alpha_WWWW2_T → "dalww2_t(gkm,mkm,"
    | D_Alpha_ZZZZ_S → "dalz4_s(gkm,mkm,"
    | D_Alpha_ZZZZ_T → "dalz4_t(gkm,mkm,"
    | G_HWW → "ghww" | G_HZZ → "ghzz"
    | G_HHWW → "ghhww" | G_HHZZ → "ghhzz"
    | G_SWW → "gsww" | G_SZZ → "gszz"
    | G_SWW_T → "gswwt" | G_SZZ_T → "gszzt"
    | G_PNWW → "gpnww" | G_PNZZ → "gpnzz"
    | G_PWZ → "gpwz" | G_PWW → "gpww"
    | G_FWW → "gfww" | G_FZZ → "gfzz"
    | G_FWW_T → "gfwwt" | G_FZZ_T → "gfzzt"
    | G_TNWW → "gtnww" | G_TNZZ → "gtnzz"
    | G_TWZ → "gtwz" | G_TWW → "gtww"
    | G_SSWW → "gssww" | G_SSZZ → "gsszz"
    | G_Htt → "ghtt" | G_Hbb → "ghbb"
    | G_Htautau → "ghtautau" | G_Hcc → "ghcc" | G_Hmm →
"ghmm"
    | G_HGaZ → "ghgaz" | G_HGaGa → "ghgaga" | G_Hgg →
"ghgg"
    | G_HGaGa_anom → "ghgaga_ac" | G_HGaZ_anom → "ghgaz_ac"
    | G_HZZ_anom → "ghzz_ac" | G_HWW_anom → "ghww_ac"
    | G_HGaZ_u → "ghgaz_u" | G_HZZ_u → "ghzz_u"
    | G_HWW_u → "ghww_u"
    | G_H3 → "gh3" | G_H4 → "gh4"
    | Gs → "gs" | I_Gs → "igs" | G2 → "gs**2"
    | Mass f → "mass" ^ flavor_symbol f
    | Width f → "width" ^ flavor_symbol f
    | K_Matrix_Coeff i → "kc" ^ string_of_int i
    | K_Matrix_Pole i → "kp" ^ string_of_int i
  end
```

13.7 Interface of Modellib_MSSM

13.7.1 More Hardcoded Models

```
module type MSSM_flags =
  sig
    val include_goldstone : bool
    val include_four : bool
```

```
    val ckm_present : bool
    val gravitino : bool
    val higgs_triangle : bool
  end

module MSSM_no_goldstone : MSSM_flags
module MSSM_goldstone : MSSM_flags
module MSSM_no_4 : MSSM_flags
module MSSM_no_4_ckm : MSSM_flags
module MSSM_Grav : MSSM_flags
module MSSM_Hgg : MSSM_flags
module MSSM : functor (F : MSSM_flags) → Model.T with module Ch = Charges.QQ
```

13.8 Implementation of Modellib_MSSM

$Id : modellib_MSSM.ml 6465 2015-01-10 15:22:31Z jr_reuter$

```
let rcs_file = RCS.parse "Modellib_MSSM" ["MSSM"]
    { RCS.revision = "$Revision: 6465 $";
      RCS.date = "$Date: 2015-01-10 16:22:31 +0100 (Sat, 10 Jan 2015) $";
      RCS.author = "$Author: jr_reuter $";
      RCS.source
        = "$URL: svn+ssh://jr_reuter@login.hepforge.org/hepforge/svn/whizard/trunk/omega/sr
```

13.8.1 Minimal Supersymmetric Standard Model

```
module type MSSM_flags =
  sig
    val include_goldstone : bool
    val include_four : bool
    val ckm_present : bool
    val gravitino : bool
    val higgs_triangle : bool
  end

module MSSM_no_goldstone : MSSM_flags =
  struct
    let include_goldstone = false
    let include_four = true
    let ckm_present = false
    let gravitino = false
    let higgs_triangle = false
  end

module MSSM_goldstone : MSSM_flags =
  struct
    let include_goldstone = true
    let include_four = true
    let ckm_present = false
    let gravitino = false
```

```
    let higgs_triangle = false
  end

module MSSM_no_4 : MSSM_flags =
  struct
    let include_goldstone = false
    let include_four = false
    let ckm_present = false
    let gravitino = false
    let higgs_triangle = false
  end

module MSSM_no_4_ckm : MSSM_flags =
  struct
    let include_goldstone = false
    let include_four = false
    let ckm_present = true
    let gravitino = false
    let higgs_triangle = false
  end

module MSSM_Grav : MSSM_flags =
  struct
    let include_goldstone = false
    let include_four = false
    let ckm_present = false
    let gravitino = true
    let higgs_triangle = false
  end

module MSSM_Hgg : MSSM_flags =
  struct
    let include_goldstone = false
    let include_four = false
    let ckm_present = false
    let gravitino = false
    let higgs_triangle = true
  end

module MSSM (Flags : MSSM_flags) =
  struct
    let rcs = RCS.rename rcs_file "Modellib_MSSM.MSSM"
        [ "MSSM" ]

    open Coupling

    let default_width = ref Timelike
    let use_fudged_width = ref false

    let options = Options.create
      [ "constant_width", Arg.Unit (fun () → default_width := Constant),
        "use constant width (also in t-channel)";
        "fudged_width", Arg.Set use_fudged_width,
        "use fudge factor for charge particle width";
```

```
      "custom_width", Arg.String (fun f → default_width := Custom f),
      "use custom width";
      "cancel_widths", Arg.Unit (fun () → default_width := Vanishing),
      "use vanishing width"]

type gen =
  | G of int | GG of gen × gen

let rec string_of_gen = function
  | G n when n > 0 → string_of_int n
  | G n → string_of_int (abs n) ^ "c"
  | GG (g1, g2) → string_of_gen g1 ^ "_" ^ string_of_gen g2
```

With this we distinguish the flavour.

```
type sff =
  | SL | SN | SU | SD

let string_of_sff = function
  | SL → "sl" | SN → "sn" | SU → "su" | SD → "sd"
```

With this we distinguish the mass eigenstates. At the moment we have to cheat a little bit for the sneutrinos. Because we are dealing with massless neutrinos there is only one sort of sneutrino.

```
type sfm =
  | M1 | M2

let string_of_sfm = function
  | M1 → "1" | M2 → "2"
```

We also introduce special types for the charginos and neutralinos.

```
type char =
  | C1 | C2 | C1c | C2c

type neu =
  | N1 | N2 | N3 | N4

let int_of_char = function
  | C1 → 1 | C2 → 2 | C1c → −1 | C2c → −2

let string_of_char = function
  | C1 → "1" | C2 → "2" | C1c → "-1" | C2c → "-2"

let conj_char = function
  | C1 → C1c | C2 → C2c | C1c → C1 | C2c → C2

let string_of_neu = function
  | N1 → "1" | N2 → "2" | N3 → "3" | N4 → "4"
```

Also we need types to distinguish the Higgs bosons. We follow the conventions of Kuroda, which means

$$H_1 = \begin{pmatrix} \frac{1}{\sqrt{2}}\bigl(v_1 + H^0\cos\alpha - h^0\sin\alpha + \mathrm{i}A^0\sin\beta - \mathrm{i}\phi^0\cos\beta\bigr) \\ H^-\sin\beta - \phi^-\cos\beta \end{pmatrix}, \tag{13.77}$$

$$H_2 = \begin{pmatrix} H^+ \cos\beta + \phi^+ \sin\beta \\ \frac{1}{\sqrt{2}}\bigl(v_2 + H^0 \sin\alpha + h^0 \cos\alpha + \mathrm{i}A^0 \cos\beta + \mathrm{i}\phi^0 \sin\beta\bigr) \end{pmatrix} \tag{13.78}$$

This is a different sign convention compared to, e.g., Weinberg's volume iii. We will refer to it as $GS+$.

```
type higgs =
  | H1 (* the light scalar Higgs *)
  | H2 (* the heavy scalar Higgs *)
  | H3 (* the pseudoscalar Higgs *)
  | H4 (* the charged Higgs *)
  | H5 (* the neutral Goldstone boson *)
  | H6 (* the charged Goldstone boson *)
  | DH of higgs × higgs

let rec string_of_higgs = function
  | H1 → "h1" | H2 → "h2" | H3 → "h3" | H4 → "h4"
  | H5 → "p1" | H6 → "p2"
  | DH (h1, h2) → string_of_higgs h1 ^ string_of_higgs h2

type flavor =
  | L of int | N of int
  | U of int | D of int
  | Sup of sfm × int | Sdown of sfm × int
  | Ga | Wp | Wm | Z | Gl
  | Slepton of sfm × int | Sneutrino of int
  | Neutralino of neu | Chargino of char
  | Gluino | Grino
  | Phip | Phim | Phi0 | H_Heavy | H_Light | Hp | Hm | A

type gauge = unit

let gauge_symbol () =
  failwith "Modellib_MSSM.MSSM.gauge_symbol: internal error"
```

At this point we will forget graviton and -tino.

```
let lep_family g = [ L g; N g; Slepton (M1, g);
                     Slepton (M2, g); Sneutrino g ]
let family g =
  [ L g; N g; Slepton (M1, g); Slepton (M2, g); Sneutrino g;
    U g; D g; Sup (M1, g); Sup (M2, g); Sdown (M1, g);
    Sdown (M2, g)]

let external_flavors'' =
    [ "1st Generation", ThoList.flatmap family [1; −1];
      "2nd Generation", ThoList.flatmap family [2; −2];
      "3rd Generation", ThoList.flatmap family [3; −3];
      "Gauge Bosons", [Ga; Z; Wp; Wm; Gl];
      "Charginos", [Chargino C1; Chargino C2; Chargino C1c; Chargino C2c];
      "Neutralinos", [Neutralino N1; Neutralino N2; Neutralino N3;
                      Neutralino N4];
      "Higgs Bosons", [H_Heavy; H_Light; Hp; Hm; A];
      "Gluinos", [Gluino]]
```

```
let external_flavors' =
  if Flags.gravitino then external_flavors'' @ ["Gravitino", [Grino]]
  else
    external_flavors''
let external_flavors () =
  if Flags.include_goldstone then external_flavors' @ ["Goldstone Bosons",
                                                        [Phip; Phim; Phi0]]
  else
    external_flavors'

let flavors () = ThoList.flatmap snd (external_flavors ())

let spinor n =
  if n ≥ 0 then
    Spinor
  else if
    n ≤ 0 then
    ConjSpinor
  else
    invalid_arg "Modellib_MSSM.MSSM.spinor: internal error"

let lorentz = function
  | L g → spinor g | N g → spinor g
  | U g → spinor g | D g → spinor g
  | Chargino c → spinor (int_of_char c)
  | Ga → Vector
  | Gl → Vector
  | Wp | Wm | Z → Massive_Vector
  | H_Heavy | H_Light | Hp | Hm | A → Scalar
  | Phip | Phim | Phi0 → Scalar
  | Sup _ | Sdown _ | Slepton _ | Sneutrino _ → Scalar
  | Neutralino _ → Majorana
  | Gluino → Majorana
  | Grino → Vectorspinor

let color = function
  | U g → Color.SUN (if g > 0 then 3 else -3)
  | Sup (m, g) → Color.SUN (if g > 0 then 3 else -3)
  | D g → Color.SUN (if g > 0 then 3 else -3)
  | Sdown (m, g) → Color.SUN (if g > 0 then 3 else -3)
  | Gl | Gluino → Color.AdjSUN 3
  | _ → Color.Singlet

let prop_spinor n =
  if n ≥ 0 then
    Prop_Spinor
  else if
    n ≤ 0 then
    Prop_ConjSpinor
  else
    invalid_arg "Modellib_MSSM.MSSM.prop_spinor: internal error"

let propagator = function
```

```
| L g → prop_spinor g | N g → prop_spinor g
| U g → prop_spinor g | D g → prop_spinor g
| Chargino c → prop_spinor (int_of_char c)
| Ga | Gl → Prop_Feynman
| Wp | Wm | Z → Prop_Unitarity
| H_Heavy | H_Light | Hp | Hm | A → Prop_Scalar
| Phip | Phim | Phi0 → if Flags.include_goldstone then Prop_Scalar
                                   else Only_Insertion
| Slepton _ | Sneutrino _ | Sup _ | Sdown _ → Prop_Scalar
| Gluino → Prop_Majorana | Neutralino _ → Prop_Majorana
| Grino → Only_Insertion
```

Note, that we define the gravitino only as an insertion since when using propagators we are effectively going to a higher order in the gravitational coupling. This would enforce us to also include higher-dimensional vertices with two gravitinos for a consistent power counting in $1/M_{\text{Planck}}$.
Optionally, ask for the fudge factor treatment for the widths of charged particles. Currently, this only applies to $W^{\pm}$ and top.

```
let width f =
  if !use_fudged_width then
    match f with
    | Wp | Wm | U 3 | U (−3) → Fudged
    | _ → !default_width
  else
    !default_width
```

For the Goldstone bosons we adopt the conventions of the Kuroda paper.

$$H_1 \equiv \begin{pmatrix} \left(v_1 + H^0\cos\alpha - h^0\sin\alpha + \mathrm{i}A^0\sin\beta - \mathrm{i}\cos\beta\phi^0\right)/\sqrt{2} \\ H^-\sin\beta - \phi^-\cos\beta \end{pmatrix} \tag{13.79a}$$

$$H_2 \equiv \begin{pmatrix} H^+\cos\beta + \phi^+\sin\beta \\ \left(v_2 + H^0\sin\alpha + h^0\cos\alpha + \mathrm{i}A^0\cos\beta + \mathrm{i}\phi^0\sin\beta\right)/\sqrt{2} \end{pmatrix} \tag{13.79b}$$

```
let goldstone = function
  | Wp → Some (Phip, Coupling.Const 1)
  | Wm → Some (Phim, Coupling.Const 1)
  | Z → Some (Phi0, Coupling.Const 1)
  | _ → None
```

```
let conjugate = function
  | L g → L (−g) | N g → N (−g)
  | U g → U (−g) | D g → D (−g)
  | Sup (m, g) → Sup (m, −g)
  | Sdown (m, g) → Sdown (m, −g)
  | Slepton (m, g) → Slepton (m, −g)
  | Sneutrino g → Sneutrino (−g)
  | Gl → Gl (* — Gl0 -¿ Gl0 *)
  | Ga → Ga | Z → Z | Wp → Wm | Wm → Wp
  | H_Heavy → H_Heavy | H_Light → H_Light | A → A
```

```
  | Hp → Hm | Hm → Hp
  | Phip → Phim | Phim → Phip | Phi0 → Phi0
  | Gluino → Gluino
  | Grino → Grino
  | Neutralino n → Neutralino n | Chargino c → Chargino (conj_char c)
let fermion = function
  | L g → if g > 0 then 1 else -1
  | N g → if g > 0 then 1 else -1
  | U g → if g > 0 then 1 else -1
  | D g → if g > 0 then 1 else -1
  | Gl | Ga | Z | Wp | Wm → 0 (* — Gl0 -¿ 0 *)
  | H_Heavy | H_Light | Hp | Hm | A → 0
  | Phip | Phim | Phi0 → 0
  | Neutralino _ → 2
  | Chargino c → if (int_of_char c) > 0 then 1 else -1
  | Sup _ → 0 | Sdown _ → 0
  | Slepton _ → 0 | Sneutrino _ → 0
  | Gluino | Grino → 2
```

Because the O'Caml compiler only allows 248 constructors we must divide the constants into subgroups of constants, e.g. for the Higgs couplings. In the MSSM there are a lot of angles among the parameters, the Weinberg-angle, the angle describing the Higgs vacuum structure, the mixing angle of the real parts of the Higgs dubletts, the mixing angles of the sfermions. Therefore we are going to define the trigonometric functions of those angles not as constants but as functors of the angels. Sums and differences of angles are only used as arguments for the α and β angles, so it makes no sense to define special functions for differences and sums of angles.

```
type angle =
  | Thw | Al | Be | Th_SF of sff × int | Delta | CKM_12 | CKM_13 | CKM_23
let string_of_angle = function
  | Thw → "thw" | Al → "al" | Be → "be" | Delta → "d"
  | CKM_12 → "ckm12" | CKM_13 → "ckm13" | CKM_23 → "ckm23"
  | Th_SF (f, g) → "th" ^ string_of_sff f ^ string_of_int g
```

We introduce a Boolean type vc as a pseudonym for Vertex Conjugator to distinguish between vertices containing complex mixing matrices like the CKM–matrix or the sfermion or neutralino/chargino–mixing matrices, which have to become complex conjugated. The true–option stands for the conjugated vertex, the false–option for the unconjugated vertex.

```
type vc = bool
type constant =
  | Unit | Pi | Alpha_QED | Sin2thw
  | Sin of angle | Cos of angle | E | G | Vev | Tanb | Tana
  | Cos2be | Cos2al | Sin2be | Sin2al | Sin4al | Sin4be | Cos4be
  | Cosapb | Cosamb | Sinapb | Sinamb | Cos2am2b | Sin2am2b
  | Eidelta
```

| Mu | AU of int | AD of int | AL of int
| V_CKM of $int \times int$ | M_SF of $sff \times int \times sfm \times sfm$
| M_V of $char \times char$ (* left chargino mixing matrix *)
| M_U of $char \times char$ (* right chargino mixing matrix *)
| M_N of $neu \times neu$ (* neutralino mixing matrix *)
| V_0 of $neu \times neu$ | A_0 of $neu \times neu$ | V_P of $char \times char$ | A_P of $char \times char$
| L_CN of $char \times neu$ | R_CN of $char \times neu$ | L_NC of $neu \times char$ | R_NC of $neu \times char$
| S_NNH1 of $neu \times neu$ | P_NNH1 of $neu \times neu$
| S_NNH2 of $neu \times neu$ | P_NNH2 of $neu \times neu$
| S_NNA of $neu \times neu$ | P_NNA of $neu \times neu$
| S_NNG of $neu \times neu$ | P_NNG of $neu \times neu$
| L_CNG of $char \times neu$ | R_CNG of $char \times neu$
| L_NCH of $neu \times char$ | R_NCH of $neu \times char$
| Q_lepton | Q_up | Q_down | Q_charg
| G_Z | G_CC | G_CCQ of $vc \times int \times int$
| $G\_NC\_neutrino$ | G_NC_lepton | G_NC_up | G_NC_down
| I_Q_W | I_G_ZWW | G_WWWW | G_ZZWW | G_PZWW | G_PPWW
| G_strong | G_SS | I_G_S | G_S_Sqrt
| Gs
| M of $flavor$ | W of $flavor$
| G_NZN of $neu \times neu$ | G_CZC of $char \times char$
| G_YUK of $int \times int$
| G_YUK_1 of $int \times int$ | G_YUK_2 of $int \times int$ | G_YUK_3 of $int \times int$
| G_YUK_4 of $int \times int$ | G_NHC of $neu \times char$ | G_CHN of $char \times neu$
| G_YUK_C of $vc \times int \times char \times sff \times sfm$
| G_YUK_Q of $vc \times int \times int \times char \times sff \times sfm$
| G_YUK_N of $vc \times int \times neu \times sff \times sfm$
| G_YUK_G of $vc \times int \times sff \times sfm$
| G_NGC of $neu \times char$ | G_CGN of $char \times neu$
| SUM_1
| G_NWC of $neu \times char$ | G_CWN of $char \times neu$
| G_CH1C of $char \times char$ | G_CH2C of $char \times char$ | G_CAC of $char \times char$
| G_CGC of $char \times char$
| G_SWS of $vc \times int \times int \times sfm \times sfm$
| G_SLSNW of $vc \times int \times sfm$
| G_ZSF of $sff \times int \times sfm \times sfm$
| G_CICIH1 of $neu \times neu$ | G_CICIH2 of $neu \times neu$ | G_CICIA of $neu \times neu$
| G_CICIG of $neu \times neu$
| G_GH of int | G_GHGo of int
| G_GLGLH | G_GLGLHH | G_GLGLA | G_PPH | G_PPHH | G_PPA
| G_WWSFSF of $sff \times int \times sfm \times sfm$
| G_WPSLSN of $vc \times int \times sfm$
| G_H3 of int | G_H4 of int
| G_HGo3 of int | G_HGo4 of int | G_GG4 of int
| G_H1SFSF of $sff \times int \times sfm \times sfm$ | G_H2SFSF of $sff \times int \times sfm \times sfm$

| *G_ASFSF* of *sff* × *int*×*sfm* × *sfm*
| *G_HSNSL* of *vc* × *int*×*sfm*
| *G_GoSFSF* of *sff* × *int*×*sfm* × *sfm*
| *G_GoSNSL* of *vc* × *int*×*sfm*
| *G_HSUSD* of *vc* × *sfm* × *sfm* × *int*×*int* | *G_GSUSD* of *vc* × *sfm* × *sfm* × *int*×*int*
| *G_WPSUSD* of *vc* × *sfm* × *sfm* × *int*×*int*
| *G_WZSUSD* of *vc* × *sfm* × *sfm* × *int*×*int*
| *G_WZSLSN* of *vc* × *int*×*sfm* | *G_GlGlSQSQ*
| *G_PPSFSF* of *sff*
| *G_ZZSFSF* of *sff* ×*int*×*sfm*×*sfm* | *G_ZPSFSF* of *sff* ×*int*×*sfm*×*sfm*
| *G_GlZSFSF* of *sff* × *int*×*sfm* × *sfm* | *G_GlPSQSQ*
| *G_GlWSUSD* of *vc* × *sfm* × *sfm* × *int*×*int*
| *G_GH4* of *int* | *G_GHGo4* of *int*
| *G_H1H2SFSF* of *sff* × *sfm* × *sfm* × *int*
| *G_H1H1SFSF* of *sff* × *sfm* × *sfm* × *int*
| *G_H2H2SFSF* of *sff* × *sfm* × *sfm* × *int*
| *G_HHSFSF* of *sff* × *sfm* × *sfm* × *int*
| *G_AASFSF* of *sff* × *sfm* × *sfm* × *int*
| *G_HH1SLSN* of *vc* × *sfm* × *int* | *G_HH2SLSN* of *vc* × *sfm* × *int*
| *G_HASLSN* of *vc* × *sfm* × *int*
| *G_HH1SUSD* of *vc* × *sfm* × *sfm* × *int*×*int*
| *G_HH2SUSD* of *vc* × *sfm* × *sfm* × *int*×*int*
| *G_HASUSD* of *vc* × *sfm* × *sfm* × *int*×*int*
| *G_AG0SFSF* of *sff* × *sfm* × *sfm* × *int*
| *G_HGSFSF* of *sff* × *sfm* × *sfm* × *int*
| *G_GGSFSF* of *sff* × *sfm* × *sfm* × *int*
| *G_G0G0SFSF* of *sff* × *sfm* × *sfm* × *int*
| *G_HGSNSL* of *vc* × *sfm* × *int* | *G_H1GSNSL* of *vc* × *sfm* × *int*
| *G_H2GSNSL* of *vc* × *sfm* × *int* | *G_AGSNSL* of *vc* × *sfm* × *int*
| *G_GGSNSL* of *vc* × *sfm* × *int*
| *G_HGSUSD* of *vc* × *sfm* × *sfm* × *int*×*int*
| *G_H1GSUSD* of *vc* × *sfm* × *sfm* × *int*×*int*
| *G_H2GSUSD* of *vc* × *sfm* × *sfm* × *int*×*int*
| *G_AGSUSD* of *vc* × *sfm* × *sfm* × *int*×*int*
| *G_GGSUSD* of *vc* × *sfm* × *sfm* × *int*×*int*
| *G_SN4* of *int*×*int*
| *G_SN2SL2_1* of *sfm* × *sfm* × *int*×*int* | *G_SN2SL2_2* of *sfm* × *sfm* × *int*×*int*
| *G_SF4* of *sff* × *sff* × *sfm* × *sfm* × *sfm* × *sfm* × *int*×*int*
| *G_SF4_3* of *sff* × *sff* × *sfm* × *sfm* × *sfm* × *sfm* × *int*×*int*×*int*
| *G_SF4_4* of *sff* × *sff* × *sfm* × *sfm* × *sfm* × *sfm* × *int*×*int*×*int*×*int*
| *G_SL4* of *sfm* × *sfm* × *sfm* × *sfm* × *int*
| *G_SL4_2* of *sfm* × *sfm* × *sfm* × *sfm* × *int*×*int*
| *G_SN2SQ2* of *sff* × *sfm* × *sfm* × *int*×*int*
| *G_SL2SQ2* of *sff* × *sfm* × *sfm* × *sfm* × *sfm* × *int*×*int*
| *G_SUSDSNSL* of *vc* × *sfm* × *sfm* × *sfm* × *int*×*int*×*int*
| *G_SU4* of *sfm* × *sfm* × *sfm* × *sfm* × *int*
| *G_SU4_2* of *sfm* × *sfm* × *sfm* × *sfm* × *int*×*int*
| *G_SD4* of *sfm* × *sfm* × *sfm* × *sfm* × *int*

```
| G_SD4_2 of sfm × sfm × sfm × sfm × int×int
| G_SU2SD2 of sfm × sfm × sfm × sfm × int×int×int×int
| G_HSF31 of higgs × int×sfm × sfm × sff × sff
| G_HSF32 of higgs × int×int×sfm × sfm × sff × sff
| G_HSF41 of higgs × int×sfm × sfm × sff × sff
| G_HSF42 of higgs × int×int×sfm × sfm × sff × sff
| G_Grav | G_Gr_Ch of char | G_Gr_Z_Neu of neu
| G_Gr_A_Neu of neu | G_Gr4_Neu of neu
| G_Gr4_A_Ch of char | G_Gr4_Z_Ch of char
| G_Grav_N | G_Grav_U of int×sfm | G_Grav_D of int×sfm
| G_Grav_L of int×sfm | G_Grav_Uc of int×sfm | G_Grav_Dc of int×sfm
| G_Grav_Lc of int×sfm | G_GravGl
| G_Gr_H_Ch of char | G_Gr_H1_Neu of neu
| G_Gr_H2_Neu of neu | G_Gr_H3_Neu of neu
| G_Gr4A_Sl of int×sfm | G_Gr4A_Slc of int×sfm
| G_Gr4A_Su of int×sfm | G_Gr4A_Suc of int×sfm
| G_Gr4A_Sd of int×sfm | G_Gr4A_Sdc of int×sfm
| G_Gr4Z_Sn | G_Gr4Z_Snc
| G_Gr4Z_Sl of int×sfm | G_Gr4Z_Slc of int×sfm
| G_Gr4Z_Su of int×sfm | G_Gr4Z_Suc of int×sfm
| G_Gr4Z_Sd of int×sfm | G_Gr4Z_Sdc of int×sfm
| G_Gr4W_Sl of int×sfm | G_Gr4W_Slc of int×sfm
| G_Gr4W_Su of int×sfm | G_Gr4W_Suc of int×sfm
| G_Gr4W_Sd of int×sfm | G_Gr4W_Sdc of int×sfm
| G_Gr4W_Sn | G_Gr4W_Snc
| G_Gr4Gl_Su of int×sfm | G_Gr4Gl_Suc of int×sfm
| G_Gr4Gl_Sd of int×sfm | G_Gr4Gl_Sdc of int×sfm
| G_Gr4_Z_H1 of neu | G_Gr4_Z_H2 of neu | G_Gr4_Z_H3 of neu
| G_Gr4_W_H of neu | G_Gr4_W_Hc of neu | G_Gr4_H_A of char
| G_Gr4_H_Z of char
```

Two integer counters for the QCD and EW order of the couplings.

```
type orders = int × int

let orders = function
  | _ → (0,0)

let ferm_of_sff = function
  | SL, g → (L g) | SN, g → (N g)
  | SU, g → (U g) | SD, g → (D g)
```

$$\alpha_{\text{QED}} = \frac{1}{137.0359895} \tag{13.80a}$$

$$\sin^2\theta_w = 0.23124 \tag{13.80b}$$

Here we must perhaps allow for complex input parameters. So split them into their modulus and their phase. At first, we leave them real; the generalization to complex parameters is obvious.

```
module Ch = Charges.QQ
```

```
let ( // ) = Algebra.Small_Rational.make

let generation' = function
  | 1 → [ 1//1; 0//1; 0//1]
  | 2 → [ 0//1; 1//1; 0//1]
  | 3 → [ 0//1; 0//1; 1//1]
  | − 1 → [−1//1; 0//1; 0//1]
  | − 2 → [ 0//1; − 1//1; 0//1]
  | − 3 → [ 0//1; 0//1; − 1//1]
  | n → invalid_arg ("MSSM.generation':␣" ^ string_of_int n)

let generation f =
  if Flags.ckm_present then
    []
  else
    match f with
    | L n | N n | U n | D n | Sup (_, n)
    | Sdown (_, n) | Slepton (_, n)
    | Sneutrino n → generation' n
    | _ → [0//1; 0//1; 0//1]

let charge = function
  | L n → if n > 0 then -1//1 else 1//1
  | Slepton (_, n) → if n > 0 then -1//1 else 1//1
  | N n → 0//1
  | Sneutrino n → 0//1
  | U n → if n > 0 then 2//3 else -2//3
  | Sup (_, n) → if n > 0 then 2//3 else -2//3
  | D n → if n > 0 then -1//3 else 1//3
  | Sdown (_, n) → if n > 0 then -1//3 else 1//3
  | Gl | Ga | Z | Neutralino _ | Gluino → 0//1
  | Wp → 1//1
  | Wm → − 1//1
  | H_Heavy | H_Light | Phi0 → 0//1
  | Hp | Phip → 1//1
  | Hm | Phim → − 1//1
  | Chargino (C1 | C2) → 1//1
  | Chargino (C1c | C2c) → − 1//1
  | _ → 0//1

let lepton = function
  | L n | N n → if n > 0 then 1//1 else -1//1
  | Slepton (_, n)
  | Sneutrino n → if n > 0 then 1//1 else -1//1
  | _ → 0//1

let baryon = function
  | U n | D n → if n > 0 then 1//1 else -1//1
  | Sup (_, n) | Sdown (_, n) → if n > 0 then 1//1 else -1//1
  | _ → 0//1

let charges f =
  [ charge f; lepton f; baryon f] @ generation f
```

```
let parameters () =
  { input = [];
    derived = [];
    derived_arrays = [] }

module F = Modeltools.Fusions (struct
  type f = flavor
  type c = constant
  let compare = compare
  let conjugate = conjugate
end)
```

For the couplings there are generally two possibilities concerning the sign of the covariant derivative.

$$\mathrm{CD}^{\pm} = \partial_\mu \pm \mathrm{i} g T^a A^a_\mu \tag{13.81}$$

The particle data group defines the signs consistently to be positive. Since the convention for that signs also influence the phase definitions of the gaugino/higgsino fields via the off-diagonal entries in their mass matrices it would be the best to adopt that convention.
** REVISED: Compatible with CD+. **

```
let electromagnetic_currents_3 g =
 [((U (-g), Ga, U g), FBF (1, Psibar, V, Psi), Q_up);
  ((D (-g), Ga, D g), FBF (1, Psibar, V, Psi), Q_down);
  ((L (-g), Ga, L g), FBF (1, Psibar, V, Psi), Q_lepton) ]
```

** REVISED: Compatible with CD+. **

```
let electromagnetic_sfermion_currents g m =
    [ ((Ga, Slepton (m, -g), Slepton (m, g)), Vector_Scalar_Scalar 1, Q_lepton);
      ((Ga, Sup (m, -g), Sup (m, g)), Vector_Scalar_Scalar 1, Q_up);
      ((Ga, Sdown (m, -g), Sdown (m, g)), Vector_Scalar_Scalar 1, Q_down) ]
```

** REVISED: Compatible with CD+. **

```
let electromagnetic_currents_2 c =
  let cc = conj_char c in
  [ ((Chargino cc, Ga, Chargino c), FBF (1, Psibar, V, Psi), Q_charg) ]
```

** REVISED: Compatible with CD+. **

```
let neutral_currents g =
  [ ((L (-g), Z, L g), FBF (1, Psibar, VA, Psi), G_NC_lepton);
    ((N (-g), Z, N g), FBF (1, Psibar, VA, Psi), G_NC_neutrino);
    ((U (-g), Z, U g), FBF (1, Psibar, VA, Psi), G_NC_up);
    ((D (-g), Z, D g), FBF (1, Psibar, VA, Psi), G_NC_down) ]
```

$$\mathcal{L}_{\mathrm{CC}} = \mp \frac{g}{2\sqrt{2}} \sum_i \bar\psi_i \gamma^\mu (1-\gamma_5)(T^+ W^+_\mu + T^- W^-_\mu)\psi_i, \tag{13.82}$$

where the sign corresponds to $\mathrm{CD}_\pm$, respectively.
** REVISED: Compatible with CD+. **
Remark: The definition with the other sign compared to the SM files comes from the fact that $g_{cc} = 1/(2\sqrt{2})$ is used overwhelmingly often in the SUSY

Feynman rules, so that JR decided to use a different definiton for g_cc in SM and MSSM.

```
let charged_currents g =
  [ ((L (-g), Wm, N g), FBF ((-1), Psibar, VL, Psi), G_CC);
    ((N (-g), Wp, L g), FBF ((-1), Psibar, VL, Psi), G_CC) ]
```

The quark with the inverted generation (the antiparticle) is the outgoing one, the other the incoming. The vertex attached to the outgoing up-quark contains the CKM matrix element *not* complex conjugated, while the vertex with the outgoing down-quark has the conjugated CKM matrix element.
** REVISED: Compatible with CD+. **

```
let charged_quark_currents g h =
  [ ((D (-g), Wm, U h), FBF ((-1), Psibar, VL, Psi), G_CCQ (true,g, h));
    ((U (-g), Wp, D h), FBF ((-1), Psibar, VL, Psi), G_CCQ (false,h, g))]
```

** REVISED: Compatible with CD+. **

```
let charged_chargino_currents n c =
  let cc = conj_char c in
  [ ((Chargino cc, Wp, Neutralino n),
                    FBF (1, Psibar, VLR, Chi), G_CWN (c, n));
    ((Neutralino n, Wm, Chargino c),
                    FBF (1, Chibar, VLR, Psi), G_NWC (n, c)) ]
```

** REVISED: Compatible with CD+. **

```
let charged_slepton_currents g m =
  [ ((Wm, Slepton (m, -g), Sneutrino g), Vector_Scalar_Scalar (-1), G_SLSNW
       (true,g, m));
    ((Wp, Slepton (m, g), Sneutrino (-g)), Vector_Scalar_Scalar 1, G_SLSNW
       (false,g, m)) ]
```

** REVISED: Compatible with CD+. **

```
let charged_squark_currents' g h m1 m2 =
  [ ((Wm, Sup (m1, g), Sdown (m2, -h)), Vector_Scalar_Scalar (-1), G_SWS
       (true,g, h, m1, m2));
    ((Wp, Sup (m1, -g), Sdown (m2, h)), Vector_Scalar_Scalar 1, G_SWS
       (false,g, h, m1, m2)) ]
let charged_squark_currents g h = List.flatten (Product.list2
          (charged_squark_currents' g h) [M1; M2] [M1; M2])
```

** REVISED: Compatible with CD+. **

```
let neutral_sfermion_currents' g m1 m2 =
  [ ((Z, Slepton (m1, -g), Slepton (m2, g)), Vector_Scalar_Scalar (-1), G_ZSF
       (SL, g, m1, m2));
    ((Z, Sup (m1, -g), Sup (m2, g)), Vector_Scalar_Scalar (-1), G_ZSF
       (SU, g, m1, m2));
    ((Z, Sdown (m1, -g), Sdown (m2, g)), Vector_Scalar_Scalar (-1), G_ZSF
       (SD, g, m1, m2)) ]
let neutral_sfermion_currents g =
  List.flatten (Product.list2 (neutral_sfermion_currents'
                g) [M1; M2] [M1; M2]) @
```

```
    [ ((Z, Sneutrino (−g), Sneutrino g), Vector_Scalar_Scalar (−1), G_ZSF
        (SN, g, M1, M1)) ]
```

The reality of the coupling of the Z-boson to two identical neutralinos makes the vector part of the coupling vanish. So we distinguish them not by the name but by the structure of the couplings.
** REVISED: Compatible with CD+. **

```
  let neutral_Z_1 (n, m) =
    [ ((Neutralino n, Z, Neutralino m), FBF (1, Chibar, VA, Chi),
          (G_NZN (n, m))) ]
(*** REVISED: Compatible with CD+. ***)
  let neutral_Z_2 n =
    [ ((Neutralino n, Z, Neutralino n), FBF (1, Chibar, Coupling.A, Chi),
          (G_NZN (n, n)) )]
```

** REVISED: Compatible with CD+. **

```
  let charged_Z c1 c2 =
    let cc1 = conj_char c1 in
    ((Chargino cc1, Z, Chargino c2), FBF ((−1), Psibar, VA, Psi),
          G_CZC (c1, c2))
```

** REVISED: Compatible with CD+. **

```
  let yukawa_v =
    [ ((Gluino, Gl, Gluino), FBF (1, Chibar, V, Chi), Gs) ]
```

** REVISED: Independent of the sign of CD. **

```
  let yukawa_higgs g =
    [ ((N (−g), Hp, L g), FBF (1, Psibar, Coupling.SR, Psi), G_YUK (6, g));
      ((L (−g), Hm, N g), FBF (1, Psibar, Coupling.SL, Psi), G_YUK (6, g));
      ((L (−g), H_Heavy, L g), FBF (1, Psibar, S, Psi), G_YUK (7, g));
      ((L (−g), H_Light, L g), FBF (1, Psibar, S, Psi), G_YUK (8, g));
      ((L (−g), A, L g), FBF (1, Psibar, P, Psi), G_YUK (9, g));
      ((U (−g), H_Heavy, U g), FBF (1, Psibar, S, Psi), G_YUK (10, g));
      ((U (−g), H_Light, U g), FBF (1, Psibar, S, Psi), G_YUK (11, g));
      ((U (−g), A, U g), FBF (1, Psibar, P, Psi), G_YUK (12, g));
      ((D (−g), H_Heavy, D g), FBF (1, Psibar, S, Psi), G_YUK (13, g));
      ((D (−g), H_Light, D g), FBF (1, Psibar, S, Psi), G_YUK (14, g));
      ((D (−g), A, D g), FBF (1, Psibar, P, Psi), G_YUK (15, g)) ]
```

** REVISED: Compatible with CD+ and GS+. **

```
  let yukawa_goldstone g =
    [ ((N (−g), Phip, L g), FBF (1, Psibar, Coupling.SR, Psi), G_YUK (19, g));
      ((L (−g), Phim, N g), FBF (1, Psibar, Coupling.SL, Psi), G_YUK (19, g));
      ((L (−g), Phi0, L g), FBF (1, Psibar, P, Psi), G_YUK (16, g));
      ((U (−g), Phi0, U g), FBF (1, Psibar, P, Psi), G_YUK (17, g));
      ((D (−g), Phi0, D g), FBF (1, Psibar, P, Psi), G_YUK (18, g)) ]
```

** REVISED: Independent of the sign of CD. **

```
  let yukawa_higgs_quark (g, h) =
    [ ((U (−g), Hp, D h), FBF (1, Psibar, SLR, Psi), G_YUK_1 (g, h));
```

```
        ((D (−h), Hm, U g), FBF (1, Psibar, SLR, Psi), G_YUK_2 (g, h)) ]
```

** REVISED: Compatible with CD+ and GS+. **

```
  let yukawa_goldstone_quark g h =
      [ (((U (−g), Phip, D h), FBF (1, Psibar, SLR, Psi), G_YUK_3 (g, h));
        ((D (−h), Phim, U g), FBF (1, Psibar, SLR, Psi), G_YUK_4 (g, h)) ]
```

** REVISED: Compatible with CD+.

```
  let yukawa_higgs_2' (c1, c2) =
    let cc1 = conj_char c1 in
    [ ((Chargino cc1, H_Heavy, Chargino c2), FBF (1, Psibar, SLR, Psi),
              G_CH2C (c1, c2));
      ((Chargino cc1, H_Light, Chargino c2), FBF (1, Psibar, SLR, Psi),
              G_CH1C (c1, c2));
      ((Chargino cc1, A, Chargino c2), FBF (1, Psibar, SLR, Psi),
              G_CAC (c1, c2)) ]
  let yukawa_higgs_2'' c =
    let cc = conj_char c in
    [ ((Chargino cc, H_Heavy, Chargino c), FBF (1, Psibar, S, Psi),
              G_CH2C (c, c));
      ((Chargino cc, H_Light, Chargino c), FBF (1, Psibar, S, Psi),
              G_CH1C (c, c));
      ((Chargino cc, A, Chargino c), FBF (1, Psibar, P, Psi),
              G_CAC (c, c)) ]
  let yukawa_higgs_2 =
    ThoList.flatmap yukawa_higgs_2' [(C1, C2); (C2, C1)] @
    ThoList.flatmap yukawa_higgs_2'' [C1; C2]
```

** REVISED: Compatible with CD+ and GS+. **

```
  let yukawa_goldstone_2' (c1, c2) =
    let cc1 = conj_char c1 in
    [ ((Chargino cc1, Phi0, Chargino c2), FBF (1, Psibar, SLR, Psi),
              G_CGC (c1, c2)) ]
  let yukawa_goldstone_2'' c =
    let cc = conj_char c in
    [ ((Chargino cc, Phi0, Chargino c), FBF (1, Psibar, P, Psi),
              G_CGC (c, c)) ]
  let yukawa_goldstone_2 =
    ThoList.flatmap yukawa_goldstone_2' [(C1, C2); (C2, C1)] @
    ThoList.flatmap yukawa_goldstone_2'' [C1; C2]
```

** REVISED: Compatible with CD+. **

```
  let higgs_charg_neutr n c =
    let cc = conj_char c in
    [ ((Neutralino n, Hm, Chargino c), FBF (−1, Chibar, SLR, Psi),
                G_NHC (n, c));
      ((Chargino cc, Hp, Neutralino n), FBF (−1, Psibar, SLR, Chi),
                G_CHN (c, n)) ]
```

** REVISED: Compatible with CD+ and GS+. **

```
let goldstone_charg_neutr n c =
  let cc = conj_char c in
  [ ((Neutralino n, Phim, Chargino c), FBF (1, Chibar, SLR, Psi),
          G_NGC (n, c));
    ((Chargino cc, Phip, Neutralino n), FBF (1, Psibar, SLR, Chi),
              G_CGN (c, n)) ]
```

** REVISED: Compatible with CD+. **

```
let higgs_neutr' (n, m) =
  [ ((Neutralino n, H_Heavy, Neutralino m), FBF (1, Chibar, SP, Chi),
          G_CICIH2 (n, m));
    ((Neutralino n, H_Light, Neutralino m), FBF (1, Chibar, SP, Chi),
          G_CICIH1 (n, m));
    ((Neutralino n, A, Neutralino m), FBF (1, Chibar, SP, Chi),
          G_CICIA (n, m)) ]
let higgs_neutr'' n =
  [ ((Neutralino n, H_Heavy, Neutralino n), FBF (1, Chibar, S, Chi),
          G_CICIH2 (n, n));
    ((Neutralino n, H_Light, Neutralino n), FBF (1, Chibar, S, Chi),
          G_CICIH1 (n, n));
    ((Neutralino n, A, Neutralino n), FBF (1, Chibar, P, Chi),
          G_CICIA (n, n)) ]
let higgs_neutr =
  ThoList.flatmap higgs_neutr' [(N1, N2); (N1, N3); (N1, N4);
                                (N2, N3); (N2, N4); (N3, N4)] @
  ThoList.flatmap higgs_neutr'' [N1; N2; N3; N4]
```

** REVISED: Compatible with CD+ and GS+. **

```
let goldstone_neutr' (n, m) =
  [ ((Neutralino n, Phi0, Neutralino m), FBF (1, Chibar, SP, Chi),
          G_CICIG (n, m)) ]
let goldstone_neutr'' n =
  [ ((Neutralino n, Phi0, Neutralino n), FBF (1, Chibar, P, Chi),
          G_CICIG (n, n)) ]
let goldstone_neutr =
  ThoList.flatmap goldstone_neutr' [(N1, N2); (N1, N3); (N1, N4);
                                    (N2, N3); (N2, N4); (N3, N4)] @
  ThoList.flatmap goldstone_neutr'' [N1; N2; N3; N4]
```

** REVISED: Compatible with CD+. **

```
let yukawa_n_1 n g =
    [ ((Neutralino n, Slepton (M1, -g), L g), FBF (1, Chibar, Coupling.SL,
        Psi), G_YUK_N (true, g, n, SL, M1));
      ((Neutralino n, Slepton (M2, -g), L g), FBF (1, Chibar, SR, Psi),
        G_YUK_N (true, g, n, SL, M2));
      ((L (-g), Slepton (M1, g), Neutralino n), FBF (1, Psibar, SR, Chi),
        G_YUK_N (false, g, n, SL, M1));
      ((L (-g), Slepton (M2, g), Neutralino n), FBF (1, Psibar, Coupling.SL,
        Chi), G_YUK_N (false, g, n, SL, M2));
      ((Neutralino n, Sup (M1, -g), U g), FBF (1, Chibar, Coupling.SL,
```

```
            Psi), G_YUK_N (true,g, n, SU, M1));
        ((Neutralino n, Sup (M2, −g), U g), FBF (1, Chibar, SR, Psi),
            G_YUK_N (true,g, n, SU, M2));
        ((U (−g), Sup (M1, g), Neutralino n), FBF (1, Psibar, SR, Chi),
            G_YUK_N (false,g, n, SU, M1));
        ((U (−g), Sup (M2, g), Neutralino n), FBF (1, Psibar, Coupling.SL,
            Chi), G_YUK_N (false,g, n, SU, M2));
        ((Neutralino n, Sdown (M1, −g), D g), FBF (1, Chibar, Coupling.SL,
            Psi), G_YUK_N (true,g, n, SD, M1));
        ((Neutralino n, Sdown (M2, −g), D g), FBF (1, Chibar, SR, Psi),
            G_YUK_N (true,g, n, SD, M2));
        ((D (−g), Sdown (M1, g), Neutralino n), FBF (1, Psibar, SR, Chi),
            G_YUK_N (false,g, n, SD, M1));
        ((D (−g), Sdown (M2, g), Neutralino n), FBF (1, Psibar, Coupling.SL,
            Chi), G_YUK_N (false,g, n, SD, M2)) ]
  let yukawa_n_2 n m =
      [ ((Neutralino n, Slepton (m, −3), L 3), FBF (1, Chibar, SLR, Psi),
            G_YUK_N (true,3,n, SL, m));
        ((L (−3), Slepton (m, 3), Neutralino n), FBF (1, Psibar, SLR, Chi),
            G_YUK_N (false,3,n, SL, m));
        ((Neutralino n, Sup (m, −3), U 3), FBF (1, Chibar, SLR, Psi),
            G_YUK_N (true,3,n, SU, m));
        ((U (−3), Sup (m, 3), Neutralino n), FBF (1, Psibar, SLR, Chi),
            G_YUK_N (false,3,n, SU, m));
        ((Neutralino n, Sdown (m, −3), D 3), FBF (1, Chibar, SLR, Psi),
            G_YUK_N (true,3,n, SD, m));
        ((D (−3), Sdown (m, 3), Neutralino n), FBF (1, Psibar, SLR, Chi),
            G_YUK_N (false,3,n, SD, m)) ]
  let yukawa_n_3 n g =
      [ ((Neutralino n, Sneutrino (−g), N g), FBF (1, Chibar, Coupling.SL,
            Psi), G_YUK_N (true,g, n, SN, M1));
        ((N (−g), Sneutrino g, Neutralino n), FBF (1, Psibar, SR, Chi),
            G_YUK_N (false,g, n, SN, M1)) ]
  let yukawa_n_4 g =
      [ ((U (−g), Sup (M1, g), Gluino), FBF ((−1), Psibar, SR, Chi), G_S_Sqrt);
        ((D (−g), Sdown (M1, g), Gluino), FBF ((−1), Psibar, SR, Chi), G_S_Sqrt);
        ((Gluino, Sup (M1, −g), U g), FBF ((−1), Chibar, Coupling.SL, Psi), G_S_Sqrt);
        ((Gluino, Sdown (M1, −g), D g), FBF ((−1), Chibar, Coupling.SL, Psi), G_S_Sqrt);
        ((U (−g), Sup (M2, g), Gluino), FBF (1, Psibar, Coupling.SL, Chi), G_S_Sqrt);
        ((D (−g), Sdown (M2, g), Gluino), FBF (1, Psibar, Coupling.SL, Chi), G_S_Sqrt);
        ((Gluino, Sup (M2, −g), U g), FBF (1, Chibar, SR, Psi), G_S_Sqrt);
        ((Gluino, Sdown (M2, −g), D g), FBF (1, Chibar, SR, Psi), G_S_Sqrt)]
  let yukawa_n_5 m =
        [ ((U (−3), Sup (m, 3), Gluino), FBF (1, Psibar, SLR, Chi),
              G_YUK_G (false,3,SU, m));
          ((D (−3), Sdown (m, 3), Gluino), FBF (1, Psibar, SLR, Chi),
              G_YUK_G (false,3,SD, m));
          ((Gluino, Sup (m, −3), U 3), FBF (1, Chibar, SLR, Psi),
              G_YUK_G (true,3,SU, m));
          ((Gluino, Sdown (m, −3), D 3), FBF (1, Chibar, SLR, Psi),
```

```
                  G_YUK_G (true,3,SD, m))]
  let yukawa_n =
    List.flatten (Product.list2 yukawa_n_1 [N1; N2; N3; N4] [1; 2]) @
    List.flatten (Product.list2 yukawa_n_2 [N1; N2; N3; N4] [M1; M2]) @
    List.flatten (Product.list2 yukawa_n_3 [N1; N2; N3; N4] [1; 2; 3]) @
    ThoList.flatmap yukawa_n_4 [1; 2] @
    ThoList.flatmap yukawa_n_5 [M1; M2]
```

** REVISED: Compatible with CD+. **

```
  let yukawa_c_1 c g =
      let cc = conj_char c in
      [ ((L (-g), Sneutrino g, Chargino cc), BBB (1, Psibar, Coupling.SR,
            Psibar), G_YUK_C (true,g, c, SN, M1));
        ((Chargino c, Sneutrino (-g), L g), PBP (1, Psi, Coupling.SL, Psi),
            G_YUK_C (false,g, c, SN, M1)) ]
  let yukawa_c_2 c =
      let cc = conj_char c in
      [ ((L (-3), Sneutrino 3, Chargino cc), BBB (1, Psibar, SLR,
            Psibar), G_YUK_C (true,3,c, SN, M1));
        ((Chargino c, Sneutrino (-3), L 3), PBP (1, Psi, SLR, Psi),
            G_YUK_C (false,3,c, SN, M1)) ]
  let yukawa_c_3 c m g =
      let cc = conj_char c in
      [ ((N (-g), Slepton (m, g), Chargino c), FBF (1, Psibar, Coupling.SR,
            Psi), G_YUK_C (true,g, c, SL, m));
        ((Chargino cc, Slepton (m, -g), N g), FBF (1, Psibar, Coupling.SL,
            Psi), G_YUK_C (false,g, c, SL, m)) ]
  let yukawa_c c =
    ThoList.flatmap (yukawa_c_1 c) [1; 2] @
    yukawa_c_2 c @
    List.flatten (Product.list2 (yukawa_c_3 c) [M1] [1; 2]) @
    List.flatten (Product.list2 (yukawa_c_3 c) [M1; M2] [3])
```

** REVISED: Compatible with CD+. **

```
  let yukawa_cq' c (g, h) m =
      let cc = conj_char c in
       [ ((Chargino c, Sup (m, -g), D h), PBP (1, Psi, SLR, Psi),
          G_YUK_Q (false,g, h, c, SU, m));
         ((D (-h), Sup (m, g), Chargino cc), BBB (1, Psibar, SLR, Psibar),
          G_YUK_Q (true,g, h, c, SU, m));
         ((Chargino cc, Sdown (m, -h), U g), FBF (1, Psibar, SLR, Psi),
          G_YUK_Q (true,g, h, c, SD, m));
         ((U (-g), Sdown (m, h), Chargino c), FBF (1, Psibar, SLR, Psi),
          G_YUK_Q (false,g, h, c, SD, m)) ]
    let yukawa_cq'' c (g, h) =
      let cc = conj_char c in
        [ ((Chargino c, Sup (M1, -g), D h), PBP (1, Psi, Coupling.SL, Psi),
              G_YUK_Q (false,g, h, c, SU, M1));
          ((D (-h), Sup (M1, g), Chargino cc),
              BBB (1, Psibar, Coupling.SR, Psibar), G_YUK_Q (true,g, h, c, SU, M1));
```

```
          ((Chargino cc, Sdown (M1, −h), U g),
              FBF (1, Psibar, Coupling.SL, Psi), G_YUK_Q (true,g, h, c, SD, M1));
          ((U (−g), Sdown (M1, h), Chargino c),
              FBF (1, Psibar, Coupling.SR, Psi), G_YUK_Q (false,g, h, c, SD, M1)) ]
  let yukawa_cq c =
    if Flags.ckm_present then
      List.flatten (Product.list2 (yukawa_cq' c) [(1, 3); (2, 3); (3, 3);
                                                  (3, 2); (3, 1)] [M1; M2]) @
      ThoList.flatmap (yukawa_cq'' c) [(1, 1); (1, 2); (2, 1); (2, 2)]
    else
      ThoList.flatmap (yukawa_cq' c (3, 3)) [M1; M2] @
      ThoList.flatmap (yukawa_cq'' c) [(1, 1); (2, 2)]
```

** REVISED: Compatible with CD+. Remark: Singlet and octet gluon exchange. The coupling is divided by sqrt(2) to account for the correct normalization of the Lie algebra generators. **

```
  let col_currents g =
    [ ((D (−g), Gl, D g), FBF ((−1), Psibar, V, Psi), Gs);
      ((U (−g), Gl, U g), FBF ((−1), Psibar, V, Psi), Gs)]
```

** REVISED: Compatible with CD+. Remark: Singlet and octet gluon exchange. The coupling is divided by sqrt(2) to account for the correct normalization of the Lie algebra generators. **

```
  let col_sfermion_currents g m =
    [ ((Gl, Sup (m, −g), Sup (m, g)), Vector_Scalar_Scalar (−1), Gs);
      ((Gl, Sdown (m, −g), Sdown (m, g)), Vector_Scalar_Scalar (−1), Gs)]
```

The gravitino coupling is generically $1/(4M_{Pl.})$
** Triple vertices containing graivitinos. **

```
  let triple_gravitino' g =
    [ ((Grino, Sneutrino (−g), N g), GBG (1, Gravbar, Coupling.SL, Psi), G_Grav_N);
      ((N (−g), Sneutrino g, Grino), GBG (1, Psibar, Coupling.SL, Grav), G_Grav_N)]

  let triple_gravitino'' g m =
    [ ((Grino, Slepton (m, −g), L g), GBG (1, Gravbar, SLR, Psi), G_Grav_L (g, m));
      ((L (−g), Slepton (m, g), Grino), GBG (1, Psibar, SLR, Grav), G_Grav_Lc (g, m));
      ((Grino, Sup (m, −g), U g), GBG (1, Gravbar, SLR, Psi), G_Grav_U (g, m));
      ((U (−g), Sup (m, g), Grino), GBG (1, Psibar, SLR, Grav), G_Grav_Uc (g, m));
      ((Grino, Sdown (m, −g), D g), GBG (1, Gravbar, SLR, Psi), G_Grav_D (g, m));
      ((D (−g), Sdown (m, g), Grino), GBG (1, Psibar, SLR, Grav), G_Grav_Dc (g, m)) ]

  let higgs_ch_gravitino c =
    let cc = conj_char c in
    [ ((Grino, Hm, Chargino c), GBG (1, Gravbar, SLR, Psi), G_Gr_H_Ch c);
      ((Chargino cc, Hp, Grino), GBG (1, Psibar, SLR, Grav), G_Gr_H_Ch cc) ]

  let higgs_neu_gravitino n =
    [ ((Grino, H_Light, Neutralino n), GBG (1, Gravbar, SLR, Chi), G_Gr_H1_Neu n);
      ((Grino, H_Heavy, Neutralino n), GBG (1, Gravbar, SLR, Chi), G_Gr_H2_Neu n);
      ((Grino, A, Neutralino n), GBG (1, Gravbar, SLR, Chi), G_Gr_H3_Neu n) ]

  let gravitino_gaugino_3 =
```

```
[ ((Grino, Gl, Gluino), GBG (1, Gravbar, V, Chi), G_Grav);
  ((Gluino, Gl, Grino), GBG (1, Chibar, V, Grav), G_Grav);
  ((Chargino C1c, Wp, Grino), GBG (1, Psibar, VLR, Grav), G_Gr_Ch C1);
  ((Chargino C2c, Wp, Grino), GBG (1, Psibar, VLR, Grav), G_Gr_Ch C2);
  ((Grino, Wm, Chargino C1), GBG (1, Gravbar, VLR, Psi), G_Gr_Ch C1c);
  ((Grino, Wm, Chargino C2), GBG (1, Gravbar, VLR, Psi), G_Gr_Ch C2c);
  ((Grino, Z, Neutralino N1), GBG (1, Gravbar, VLR, Chi), G_Gr_Z_Neu N1);
  ((Grino, Z, Neutralino N2), GBG (1, Gravbar, VLR, Chi), G_Gr_Z_Neu N2);
  ((Grino, Z, Neutralino N3), GBG (1, Gravbar, VLR, Chi), G_Gr_Z_Neu N3);
  ((Grino, Z, Neutralino N4), GBG (1, Gravbar, VLR, Chi), G_Gr_Z_Neu N4);
  ((Grino, Ga, Neutralino N1), GBG (1, Gravbar, VLR, Chi), G_Gr_A_Neu N1);
  ((Grino, Ga, Neutralino N2), GBG (1, Gravbar, VLR, Chi), G_Gr_A_Neu N2);
  ((Grino, Ga, Neutralino N3), GBG (1, Gravbar, VLR, Chi), G_Gr_A_Neu N3);
  ((Grino, Ga, Neutralino N4), GBG (1, Gravbar, VLR, Chi), G_Gr_A_Neu N4) ]

let triple_gravitino =
  ThoList.flatmap triple_gravitino' [1;2;3] @
  List.flatten (Product.list2 triple_gravitino'' [1;2;3] [M1; M2]) @
  ThoList.flatmap higgs_ch_gravitino [C1; C2] @
  ThoList.flatmap higgs_neu_gravitino [N1; N2; N3; N4] @
  gravitino_gaugino_3
```

** REVISED: Compatible with CD+. **

```
let triple_gauge =
  [ ((Ga, Wm, Wp), Gauge_Gauge_Gauge 1, I_Q_W);
    ((Z, Wm, Wp), Gauge_Gauge_Gauge 1, I_G_ZWW);
    ((Gl, Gl, Gl), Gauge_Gauge_Gauge 1, I_G_S)]
```

** REVISED: Independent of the sign of CD. **

```
let gauge4 = Vector4 [(2, C_13_42); (-1, C_12_34); (-1, C_14_23)]
let gluon4 = Vector4 [(-1, C_13_42); (-1, C_12_34); (-1, C_14_23)]
let minus_gauge4 = Vector4 [(-2, C_13_42); (1, C_12_34); (1, C_14_23)]
let quartic_gauge =
  [ (Wm, Wp, Wm, Wp), gauge4, G_WWWW;
    (Wm, Z, Wp, Z), minus_gauge4, G_ZZWW;
    (Wm, Z, Wp, Ga), minus_gauge4, G_PZWW;
    (Wm, Ga, Wp, Ga), minus_gauge4, G_PPWW;
    (Gl, Gl, Gl, Gl), gauge4, G_SS]
```

The *Scalar_Vector_Vector* couplings do not depend on the choice of the sign of the covariant derivative since they are quadratic in the gauge couplings.
** REVISED: Compatible with CD+. **
** Revision: 2005-03-10: first two vertices corrected. **

```
let gauge_higgs =
  [ ((Wm, Hp, A), Vector_Scalar_Scalar 1, G_GH 1);
    ((Wp, Hm, A), Vector_Scalar_Scalar 1, G_GH 1);
    ((Z, H_Heavy, A), Vector_Scalar_Scalar 1, G_GH 3);
    ((Z, H_Light, A), Vector_Scalar_Scalar 1, G_GH 2);
    ((H_Heavy, Wp, Wm), Scalar_Vector_Vector 1, G_GH 5);
    ((H_Light, Wp, Wm), Scalar_Vector_Vector 1, G_GH 4);
    ((Wm, Hp, H_Heavy), Vector_Scalar_Scalar 1, G_GH 7);
```

```
    ((Wp, Hm, H_Heavy), Vector_Scalar_Scalar (-1), G_GH 7);
    ((Wm, Hp, H_Light), Vector_Scalar_Scalar 1, G_GH 6);
    ((Wp, Hm, H_Light), Vector_Scalar_Scalar (-1), G_GH 6);
    ((H_Heavy, Z, Z), Scalar_Vector_Vector 1, G_GH 9);
    ((H_Light, Z, Z), Scalar_Vector_Vector 1, G_GH 8);
    ((Z, Hp, Hm), Vector_Scalar_Scalar 1, G_GH 10);
    ((Ga, Hp, Hm), Vector_Scalar_Scalar 1, G_GH 11) ] @
  (if Flags.higgs_triangle then
   [((H_Light, Gl, Gl), Dim5_Scalar_Gauge2 1, G_GLGLH);
    ((H_Heavy, Gl, Gl), Dim5_Scalar_Gauge2 1, G_GLGLHH);
    ((A, Gl, Gl), Dim5_Scalar_Gauge2_Skew 1, G_GLGLA);
    ((H_Light, Ga, Ga), Dim5_Scalar_Gauge2 1, G_PPH);
    ((H_Heavy, Ga, Ga), Dim5_Scalar_Gauge2 1, G_PPHH);
    ((A, Ga, Ga), Dim5_Scalar_Gauge2 1, G_PPA)]
   else
     [])
```

** REVISED: Compatible with CD+ and GS+. **

```
let gauge_higgs_gold =
  [ ((Wp, Phi0, Phim), Vector_Scalar_Scalar 1, G_GH 1);
    ((Wm, Phi0, Phip), Vector_Scalar_Scalar 1, G_GH 1);
    ((Z, H_Heavy, Phi0), Vector_Scalar_Scalar 1, G_GH 2);
    ((Z, H_Light, Phi0), Vector_Scalar_Scalar (-1), G_GH 3);
    ((Wp, H_Heavy, Phim), Vector_Scalar_Scalar 1, G_GH 6);
    ((Wm, H_Heavy, Phip), Vector_Scalar_Scalar (-1), G_GH 6);
    ((Wp, H_Light, Phim), Vector_Scalar_Scalar (-1), G_GH 7);
    ((Wm, H_Light, Phip), Vector_Scalar_Scalar 1, G_GH 7);
    ((Phim, Wp, Ga), Scalar_Vector_Vector 1, G_GHGo 1);
    ((Phip, Wm, Ga), Scalar_Vector_Vector 1, G_GHGo 1);
    ((Phim, Wp, Z), Scalar_Vector_Vector 1, G_GHGo 2);
    ((Phip, Wm, Z), Scalar_Vector_Vector 1, G_GHGo 2);
    ((Z, Phip, Phim), Vector_Scalar_Scalar 1, G_GH 10);
    ((Ga, Phip, Phim), Vector_Scalar_Scalar 1, G_GH 11) ]

let gauge_higgs4 =
  [ ((A, A, Z, Z), Scalar2_Vector2 1, G_GH4 1);
    ((H_Heavy, H_Heavy, Z, Z), Scalar2_Vector2 1, G_GH4 3);
    ((H_Light, H_Light, Z, Z), Scalar2_Vector2 1, G_GH4 2);
    ((Hp, Hm, Z, Z), Scalar2_Vector2 1, G_GH4 4);
    ((Hp, Hm, Ga, Ga), Scalar2_Vector2 1, G_GH4 5);
    ((Hp, Hm, Ga, Z), Scalar2_Vector2 1, G_GH4 6);
    ((Hp, H_Heavy, Wm, Z), Scalar2_Vector2 1, G_GH4 8);
    ((Hm, H_Heavy, Wp, Z), Scalar2_Vector2 1, G_GH4 8);
    ((Hp, H_Light, Wm, Z), Scalar2_Vector2 1, G_GH4 7);
    ((Hm, H_Light, Wp, Z), Scalar2_Vector2 1, G_GH4 7);
    ((Hp, H_Heavy, Wm, Ga), Scalar2_Vector2 1, G_GH4 10);
    ((Hm, H_Heavy, Wp, Ga), Scalar2_Vector2 1, G_GH4 10);
    ((Hp, H_Light, Wm, Ga), Scalar2_Vector2 1, G_GH4 9);
    ((Hm, H_Light, Wp, Ga), Scalar2_Vector2 1, G_GH4 9);
    ((A, A, Wp, Wm), Scalar2_Vector2 1, G_GH4 11);
    ((H_Heavy, H_Heavy, Wp, Wm), Scalar2_Vector2 1, G_GH4 13);
```

```
  ((H_Light, H_Light, Wp, Wm), Scalar2_Vector2 1, G_GH4 12);
  ((Hp, Hm, Wp, Wm), Scalar2_Vector2 1, G_GH4 14);
  ((Hp, A, Wm, Z), Scalar2_Vector2 1, G_GH4 15);
  ((Hm, A, Wp, Z), Scalar2_Vector2 (-1), G_GH4 15);
  ((Hp, A, Wm, Ga), Scalar2_Vector2 1, G_GH4 16);
  ((Hm, A, Wp, Ga), Scalar2_Vector2 (-1), G_GH4 16) ]

let gauge_higgs_gold4 =
 [ ((Z, Z, Phi0, Phi0), Scalar2_Vector2 1, G_GHGo4 1);
  ((Z, Z, Phip, Phim), Scalar2_Vector2 1, G_GHGo4 2);
  ((Ga, Ga, Phip, Phim), Scalar2_Vector2 1, G_GHGo4 3);
  ((Z, Ga, Phip, Phim), Scalar2_Vector2 1, G_GHGo4 4);
  ((Wp, Wm, Phip, Phim), Scalar2_Vector2 1, G_GHGo4 5);
  ((Wp, Wm, Phi0, Phi0), Scalar2_Vector2 1, G_GHGo4 5);
  ((Wp, Z, Phim, Phi0), Scalar2_Vector2 1, G_GHGo4 6);
  ((Wm, Z, Phip, Phi0), Scalar2_Vector2 (-1), G_GHGo4 6);
  ((Wp, Ga, Phim, Phi0), Scalar2_Vector2 1, G_GHGo4 7);
  ((Wm, Ga, Phip, Phi0), Scalar2_Vector2 (-1), G_GHGo4 7);
  ((Wp, Z, Phim, H_Heavy), Scalar2_Vector2 1, G_GHGo4 9);
  ((Wm, Z, Phip, H_Heavy), Scalar2_Vector2 1, G_GHGo4 9);
  ((Wp, Ga, Phim, H_Heavy), Scalar2_Vector2 1, G_GHGo4 11);
  ((Wm, Ga, Phip, H_Heavy), Scalar2_Vector2 1, G_GHGo4 11);
  ((Wp, Z, Phim, H_Light), Scalar2_Vector2 1, G_GHGo4 8);
  ((Wm, Z, Phip, H_Light), Scalar2_Vector2 1, G_GHGo4 8);
  ((Wp, Ga, Phim, H_Light), Scalar2_Vector2 1, G_GHGo4 10);
  ((Wm, Ga, Phip, H_Light), Scalar2_Vector2 1, G_GHGo4 10) ]

let gauge_sfermion4' g m1 m2 =
  [ ((Wp, Wm, Slepton (m1, g), Slepton (m2, -g)), Scalar2_Vector2 1,
       G_WWSFSF (SL, g, m1, m2));
   ((Z, Ga, Slepton (m1, g), Slepton (m2, -g)), Scalar2_Vector2 1,
      G_ZPSFSF (SL, g, m1, m2));
   ((Z, Z, Slepton (m1, g), Slepton (m2, -g)), Scalar2_Vector2 1, G_ZZSFSF
      (SL, g, m1, m2));
   ((Wp, Wm, Sup (m1, g), Sup (m2, -g)), Scalar2_Vector2 1, G_WWSFSF
      (SU, g, m1, m2));
   ((Wp, Wm, Sdown (m1, g), Sdown (m2, -g)), Scalar2_Vector2 1, G_WWSFSF
      (SD, g, m1, m2));
   ((Z, Z, Sup (m1, g), Sup (m2, -g)), Scalar2_Vector2 1, G_ZZSFSF
      (SU, g, m1, m2));
   ((Z, Z, Sdown (m1, g), Sdown (m2, -g)), Scalar2_Vector2 1, G_ZZSFSF
      (SD, g, m1, m2));
   ((Z, Ga, Sup (m1, g), Sup (m2, -g)), Scalar2_Vector2 1, G_ZPSFSF
      (SU, g, m1, m2));
   ((Z, Ga, Sdown (m1, g), Sdown (m2, -g)), Scalar2_Vector2 1, G_ZPSFSF
      (SD, g, m1, m2)) ]
let gauge_sfermion4'' g m =
 [ ((Wp, Ga, Slepton (m, g), Sneutrino (-g)), Scalar2_Vector2 1, G_WPSLSN
      (false, g, m));
   ((Wm, Ga, Slepton (m, -g), Sneutrino g), Scalar2_Vector2 1,
      G_WPSLSN (true, g, m));
```

```
    ((Wp, Z, Slepton (m, g), Sneutrino (-g)), Scalar2_Vector2 1, G_WZSLSN
        (false,g, m));
    ((Wm, Z, Slepton (m, -g), Sneutrino g), Scalar2_Vector2 1,
        G_WZSLSN (true,g, m));
    ((Ga, Ga, Slepton (m, g), Slepton (m, -g)), Scalar2_Vector2 1, G_PPSFSF SL);
    ((Ga, Ga, Sup (m, g), Sup (m, -g)), Scalar2_Vector2 1, G_PPSFSF SU);
    ((Ga, Ga, Sdown (m, g), Sdown (m, -g)), Scalar2_Vector2 1, G_PPSFSF SD)]
let gauge_sfermion4 g =
  List.flatten (Product.list2 (gauge_sfermion4' g) [M1; M2] [M1; M2]) @
  ThoList.flatmap (gauge_sfermion4'' g) [M1; M2] @
  [ ((Wp, Wm, Sneutrino g, Sneutrino (-g)), Scalar2_Vector2 1, G_WWSFSF
        (SN, g, M1, M1));
    ((Z, Z, Sneutrino g, Sneutrino (-g)), Scalar2_Vector2 1, G_ZZSFSF
        (SN, g, M1, M1)) ]

let gauge_squark4'' g h m1 m2 =
  [ ((Wp, Ga, Sup (m1, -g), Sdown (m2, h)), Scalar2_Vector2 1, G_WPSUSD
        (false,m1, m2, g, h));
    ((Wm, Ga, Sup (m1, g), Sdown (m2, -h)), Scalar2_Vector2 1, G_WPSUSD
        (true,m1, m2, g, h));
    ((Wp, Z, Sup (m1, -g), Sdown (m2, h)), Scalar2_Vector2 1, G_WZSUSD
        (false,m1, m2, g, h));
    ((Wm, Z, Sup (m1, g), Sdown (m2, -h)), Scalar2_Vector2 1, G_WZSUSD
        (true,m1, m2, g, h)) ]
let gauge_squark4' g h = List.flatten (Product.list2 (gauge_squark4'' g h)
                                          [M1; M2] [M1; M2])
let gauge_squark4 =
  if Flags.ckm_present then
    List.flatten (Product.list2 gauge_squark4' [1; 2; 3] [1; 2; 3])
  else
    ThoList.flatmap (fun g -> gauge_squark4' g g) [1; 2; 3]

let gluon_w_squark'' g h m1 m2 =
  [ ((Gl, Wp, Sup (m1, -g), Sdown (m2, h)),
         Scalar2_Vector2 1, G_GlWSUSD (false,m1, m2, g, h));
    ((Gl, Wm, Sup (m1, g), Sdown (m2, -h)),
         Scalar2_Vector2 1, G_GlWSUSD (true,m1, m2, g, h)) ]
let gluon_w_squark' g h =
  List.flatten (Product.list2 (gluon_w_squark'' g h) [M1; M2] [M1; M2])
let gluon_w_squark =
  if Flags.ckm_present then
    List.flatten (Product.list2 gluon_w_squark' [1; 2; 3] [1; 2; 3])
  else
    ThoList.flatmap (fun g -> gluon_w_squark' g g) [1; 2; 3]

let gluon_gauge_squark' g m1 m2 =
  [ ((Gl, Z, Sup (m1, g), Sup (m2, -g)), Scalar2_Vector2 2, G_GlZSFSF (SU, g, m1, m2));
    ((Gl, Z, Sdown (m1, g), Sdown (m2, -g)), Scalar2_Vector2 2, G_GlZSFSF (SD, g, m1, m2)) ]
let gluon_gauge_squark'' g m =
  [ ((Gl, Ga, Sup (m, g), Sup (m, -g)), Scalar2_Vector2 2, G_GlPSQSQ);
    ((Gl, Ga, Sdown (m, g), Sdown (m, -g)), Scalar2_Vector2 (-1), G_GlPSQSQ) ]
let gluon_gauge_squark g =
```

$List.flatten\ (Product.list2\ (gluon\_gauge\_squark'\ g)\ [M1; M2]\ [M1; M2])$ @
$ThoList.flatmap\ (gluon\_gauge\_squark''\ g)\ [M1; M2]$

let $gluon2\_squark2\ g\ m\ =$
$[\ ((Gl,\ Gl,\ Sup\ (m, g),\ Sup\ (m, -g)),\ Scalar2\_Vector2\ 1,\ G\_GlGlSQSQ);$
$((Gl,\ Gl,\ Sdown\ (m, g),\ Sdown\ (m, -g)),\ Scalar2\_Vector2\ 1,\ G\_GlGlSQSQ)]$

** REVISED: Independent of the sign of CD. **

let $higgs\ =$
$[\ ((Hp,\ Hm,\ H\_Heavy),\ Scalar\_Scalar\_Scalar\ 1,\ G\_H3\ 1);$
$((Hp,\ Hm,\ H\_Light),\ Scalar\_Scalar\_Scalar\ 1,\ G\_H3\ 2);$
$((H\_Heavy,\ H\_Heavy,\ H\_Light),\ Scalar\_Scalar\_Scalar\ 1,\ G\_H3\ 3);$
$((H\_Heavy,\ H\_Heavy,\ H\_Heavy),\ Scalar\_Scalar\_Scalar\ 1,\ G\_H3\ 4);$
$((H\_Light,\ H\_Light,\ H\_Light),\ Scalar\_Scalar\_Scalar\ 1,\ G\_H3\ 5);$
$((H\_Heavy,\ H\_Light,\ H\_Light),\ Scalar\_Scalar\_Scalar\ 1,\ G\_H3\ 6);$
$((H\_Heavy,\ A,\ A),\ Scalar\_Scalar\_Scalar\ 1,\ G\_H3\ 7);$
$((H\_Light,\ A,\ A),\ Scalar\_Scalar\_Scalar\ 1,\ G\_H3\ 8)\ ]$

** REVISED: Compatible with GS+, independent of the sign of CD. **

let $higgs\_gold\ =$
$[\ ((H\_Heavy,\ A,\ Phi0),\ Scalar\_Scalar\_Scalar\ 1,\ G\_HGo3\ 1);$
$((H\_Light,\ A,\ Phi0),\ Scalar\_Scalar\_Scalar\ 1,\ G\_HGo3\ 2);$
$((H\_Heavy,\ Hp,\ Phim),\ Scalar\_Scalar\_Scalar\ 1,\ G\_HGo3\ 3);$
$((H\_Heavy,\ Hm,\ Phip),\ Scalar\_Scalar\_Scalar\ 1,\ G\_HGo3\ 3);$
$((H\_Light,\ Hp,\ Phim),\ Scalar\_Scalar\_Scalar\ 1,\ G\_HGo3\ 4);$
$((H\_Light,\ Hm,\ Phip),\ Scalar\_Scalar\_Scalar\ 1,\ G\_HGo3\ 4);$
$((A,\ Hp,\ Phim),\ Scalar\_Scalar\_Scalar\ (-1),\ G\_HGo3\ 5);$
$((A,\ Hm,\ Phip),\ Scalar\_Scalar\_Scalar\ 1,\ G\_HGo3\ 5);$
$((H\_Heavy,\ Phi0,\ Phi0),\ Scalar\_Scalar\_Scalar\ (-1),\ G\_H3\ 7);$
$((H\_Heavy,\ Phip,\ Phim),\ Scalar\_Scalar\_Scalar\ (-1),\ G\_H3\ 7);$
$((H\_Light,\ Phi0,\ Phi0),\ Scalar\_Scalar\_Scalar\ (-1),\ G\_H3\ 8);$
$((H\_Light,\ Phip,\ Phim),\ Scalar\_Scalar\_Scalar\ (-1),\ G\_H3\ 8)\ ]$

Here follow purely scalar quartic vertices which are only available for the no-Whizard colored version.
** REVISED: Independent of the sign of CD. **

let $higgs4\ =$
$[\ ((Hp,\ Hm,\ Hp,\ Hm),\ Scalar4\ 1,\ G\_H4\ 1);$
$((Hp,\ Hm,\ H\_Heavy,\ H\_Heavy),\ Scalar4\ 1,\ G\_H4\ 2);$
$((Hp,\ Hm,\ H\_Light,\ H\_Light),\ Scalar4\ 1,\ G\_H4\ 3);$
$((Hp,\ Hm,\ H\_Heavy,\ H\_Light),\ Scalar4\ 1,\ G\_H4\ 4);$
$((Hp,\ Hm,\ A,\ A),\ Scalar4\ 1,\ G\_H4\ 5);$
$((H\_Heavy,\ H\_Heavy,\ H\_Heavy,\ H\_Heavy),\ Scalar4\ 1,\ G\_H4\ 6);$
$((H\_Light,\ H\_Light,\ H\_Light,\ H\_Light),\ Scalar4\ 1,\ G\_H4\ 6);$
$((H\_Heavy,\ H\_Heavy,\ H\_Light,\ H\_Light),\ Scalar4\ 1,\ G\_H4\ 7);$
$((H\_Heavy,\ H\_Light,\ H\_Light,\ H\_Light),\ Scalar4\ 1,\ G\_H4\ 8);$
$((H\_Heavy,\ H\_Heavy,\ H\_Heavy,\ H\_Light),\ Scalar4\ (-1),\ G\_H4\ 8);$
$((H\_Heavy,\ H\_Heavy,\ A,\ A),\ Scalar4\ 1,\ G\_H4\ 9);$
$((H\_Light,\ H\_Light,\ A,\ A),\ Scalar4\ (-1),\ G\_H4\ 9);$
$((H\_Heavy,\ H\_Light,\ A,\ A),\ Scalar4\ 1,\ G\_H4\ 10);$
$((A,\ A,\ A,\ A),\ Scalar4\ 1,\ G\_H4\ 11)\ ]$

** REVISED: Compatible with GS+, independent of the sign of CD. **

```
let higgs_gold4 =
  [ ((H_Heavy, H_Heavy, A, Phi0), Scalar4 1, G_HGo4 1);
    ((H_Heavy, H_Light, A, Phi0), Scalar4 1, G_HGo4 2);
    ((H_Light, H_Light, A, Phi0), Scalar4 (-1), G_HGo4 1);
    ((A, A, A, Phi0), Scalar4 3, G_HGo4 3);
    ((Hp, Hm, A, Phi0), Scalar4 1, G_HGo4 3);
    ((H_Heavy, H_Heavy, Hp, Phim), Scalar4 1, G_HGo4 4);
    ((H_Heavy, H_Heavy, Hm, Phip), Scalar4 1, G_HGo4 4);
    ((H_Heavy, H_Light, Hp, Phim), Scalar4 1, G_HGo4 5);
    ((H_Heavy, H_Light, Hm, Phip), Scalar4 1, G_HGo4 5);
    ((H_Light, H_Light, Hp, Phim), Scalar4 (-1), G_HGo4 4);
    ((H_Light, H_Light, Hm, Phip), Scalar4 (-1), G_HGo4 4);
    ((A, A, Hp, Phim), Scalar4 1, G_HGo4 6);
    ((A, A, Hm, Phip), Scalar4 1, G_HGo4 6);
    ((H_Heavy, A, Hp, Phim), Scalar4 1, G_HGo4 7);
    ((H_Heavy, A, Hm, Phip), Scalar4 (-1), G_HGo4 7);
    ((H_Light, A, Hp, Phim), Scalar4 1, G_HGo4 8);
    ((H_Light, A, Hm, Phip), Scalar4 (-1), G_HGo4 8);
    ((Hp, Hm, Hp, Phim), Scalar4 2, G_HGo4 6);
    ((Hp, Hm, Hm, Phip), Scalar4 2, G_HGo4 6);
    ((H_Heavy, H_Heavy, Phi0, Phi0), Scalar4 (-1), G_H4 9);
    ((H_Heavy, H_Light, Phi0, Phi0), Scalar4 (-1), G_H4 10);
    ((H_Light, H_Light, Phi0, Phi0), Scalar4 1, G_H4 9);
    ((A, A, Phi0, Phi0), Scalar4 1, G_HGo4 9);
    ((Hp, Hm, Phi0, Phi0), Scalar4 1, G_HGo4 10);
    ((H_Heavy, Hp, Phim, Phi0), Scalar4 1, G_HGo4 8);
    ((H_Heavy, Hm, Phip, Phi0), Scalar4 (-1), G_HGo4 8);
    ((H_Light, Hp, Phim, Phi0), Scalar4 (-1), G_HGo4 7);
    ((H_Light, Hm, Phip, Phi0), Scalar4 1, G_HGo4 7);
    ((A, Hp, Phim, Phi0), Scalar4 1, G_HGo4 11);
    ((A, Hm, Phip, Phi0), Scalar4 1, G_HGo4 11);
    ((H_Heavy, H_Heavy, Phip, Phim), Scalar4 1, G_HGo4 12);
    ((H_Heavy, H_Light, Phip, Phim), Scalar4 1, G_HGo4 13);
    ((H_Light, H_Light, Phip, Phim), Scalar4 1, G_HGo4 14);
    ((A, A, Phip, Phim), Scalar4 1, G_HGo4 15);
    ((Hp, Hm, Phip, Phim), Scalar4 1, G_HGo4 16);
    ((Hp, Hp, Phim, Phim), Scalar4 1, G_HGo4 17);
    ((Hm, Hm, Phip, Phip), Scalar4 1, G_HGo4 17);
    ((Hp, Phim, Phi0, Phi0), Scalar4 (-1), G_HGo4 6);
    ((Hm, Phip, Phi0, Phi0), Scalar4 (-1), G_HGo4 6);
    ((A, Phi0, Phi0, Phi0), Scalar4 (-3), G_HGo4 6);
    ((A, Phi0, Phip, Phim), Scalar4 (-1), G_HGo4 6);
    ((Hp, Phim, Phip, Phim), Scalar4 (-2), G_HGo4 6);
    ((Hm, Phip, Phip, Phim), Scalar4 (-2), G_HGo4 6) ]
```

** REVISED: Independent of the sign of CD and GS. **

```
let goldstone4 =
  [ ((Phi0, Phi0, Phi0, Phi0), Scalar4 1, G_GG4 1);
    ((Phip, Phim, Phi0, Phi0), Scalar4 1, G_GG4 2);
```

```
    ((Phip, Phim, Phip, Phim), Scalar4 1, G_GG4 3) ]
```

The vertices of the type Higgs - Sfermion - Sfermion are independent of the choice of the CD sign since they are quadratic in the gauge coupling.
** REVISED: Independent of the sign of CD. **

```
  let higgs_sneutrino' g =
    [ ((H_Heavy, Sneutrino g, Sneutrino (-g)), Scalar_Scalar_Scalar 1,
                      G_H2SFSF (SN, g, M1, M1));
      ((H_Light, Sneutrino g, Sneutrino (-g)), Scalar_Scalar_Scalar 1,
                      G_H1SFSF (SN, g, M1, M1));
      ((Hp, Sneutrino (-g), Slepton (M1, g)), Scalar_Scalar_Scalar 1,
            G_HSNSL (false,g, M1));
      ((Hm, Sneutrino g, Slepton (M1, -g)), Scalar_Scalar_Scalar 1,
            G_HSNSL (true,g, M1)) ]
    let higgs_sneutrino'' =
    [ ((Hp, Sneutrino (-3), Slepton (M2, 3)), Scalar_Scalar_Scalar 1,
            G_HSNSL (false,3,M2));
      ((Hm, Sneutrino 3, Slepton (M2, -3)), Scalar_Scalar_Scalar 1,
            G_HSNSL (false,3,M2)) ]
    let higgs_sneutrino =
      ThoList.flatmap higgs_sneutrino' [1; 2; 3] @ higgs_sneutrino''
```

Under the assumption that there is no mixing between the left- and right-handed sfermions for the first two generations there is only a coupling of the form Higgs - sfermion1 - sfermion2 for the third generation. All the others are suppressed by m_f/M_W.
** REVISED: Independent of the sign of CD. **

```
  let higgs_sfermion' g m1 m2 =
    [ ((H_Heavy, Slepton (m1, g), Slepton (m2, -g)), Scalar_Scalar_Scalar 1,
          G_H2SFSF (SL, g, m1, m2));
      ((H_Light, Slepton (m1, g), Slepton (m2, -g)), Scalar_Scalar_Scalar 1,
          G_H1SFSF (SL, g, m1, m2));
      ((H_Heavy, Sup (m1, g), Sup (m2, -g)), Scalar_Scalar_Scalar 1,
          G_H2SFSF (SU, g, m1, m2));
      ((H_Heavy, Sdown (m1, g), Sdown (m2, -g)), Scalar_Scalar_Scalar 1,
          G_H2SFSF (SD, g, m1, m2));
      ((H_Light, Sup (m1, g), Sup (m2, -g)), Scalar_Scalar_Scalar 1,
          G_H1SFSF (SU, g, m1, m2));
      ((H_Light, Sdown (m1, g), Sdown (m2, -g)), Scalar_Scalar_Scalar 1,
          G_H1SFSF (SD, g, m1, m2)) ]
    let higgs_sfermion'' m1 m2 =
      [ ((A, Slepton (m1, 3), Slepton (m2, -3)), Scalar_Scalar_Scalar 1,
          G_ASFSF (SL, 3, m1, m2));
        ((A, Sup (m1, 3), Sup (m2, -3)), Scalar_Scalar_Scalar 1,
          G_ASFSF (SU, 3, m1, m2));
        ((A, Sdown (m1, 3), Sdown (m2, -3)), Scalar_Scalar_Scalar 1,
          G_ASFSF (SD, 3, m1, m2)) ]
  let higgs_sfermion = List.flatten (Product.list2 (higgs_sfermion' 3)
                                       [M1; M2] [M1; M2]) @
      (higgs_sfermion' 1 M1 M1) @ (higgs_sfermion' 1 M2 M2) @
```

```
    (higgs_sfermion' 2 M1 M1) @ (higgs_sfermion' 2 M2 M2) @
    List.flatten (Product.list2 higgs_sfermion'' [M1; M2] [M1; M2])
```

** REVISED: Independent of the sign of CD, compatible with GS+. **

```
let goldstone_sfermion' g m1 m2 =
  [ ((Phi0, Slepton (m1, g), Slepton (m2, -g)), Scalar_Scalar_Scalar 1,
                     G_GoSFSF (SL, g, m1, m2));
    ((Phi0, Sup (m1, g), Sup (m2, -g)), Scalar_Scalar_Scalar 1,
                     G_GoSFSF (SU, g, m1, m2));
    ((Phi0, Sdown (m1, g), Sdown (m2, -g)), Scalar_Scalar_Scalar 1,
                     G_GoSFSF (SD, g, m1, m2))]
let goldstone_sfermion'' g =
  [ ((Phip, Sneutrino (-g), Slepton (M1, g)), Scalar_Scalar_Scalar 1,
                     G_GoSNSL (false,g, M1));
    ((Phim, Sneutrino g, Slepton (M1, -g)), Scalar_Scalar_Scalar 1,
                     G_GoSNSL (true,g, M1)) ]
let goldstone_sfermion''' g =
  [ ((Phip, Sneutrino (-g), Slepton (M2, g)), Scalar_Scalar_Scalar 1,
                     G_GoSNSL (false,g, M2));
    ((Phim, Sneutrino g, Slepton (M2, -g)), Scalar_Scalar_Scalar 1,
                     G_GoSNSL (true,g, M2))]
let goldstone_sfermion =
  List.flatten (Product.list2 (goldstone_sfermion' 3) [M1; M2] [M1; M2]) @
  ThoList.flatmap goldstone_sfermion'' [1; 2; 3] @
  goldstone_sfermion''' 3
```

** REVISED: Independent of the sign of CD. **

```
let higgs_squark' g h m1 m2 =
  [ ((Hp, Sup (m1, -g), Sdown (m2, h)), Scalar_Scalar_Scalar 1,
          G_HSUSD (false,m1, m2, g, h));
    ((Hm, Sup (m1, g), Sdown (m2, -h)), Scalar_Scalar_Scalar 1,
          G_HSUSD (true,m1, m2, g, h)) ]
let higgs_squark_a g h = higgs_squark' g h M1 M1
let higgs_squark_b (g, h) = List.flatten (Product.list2 (higgs_squark' g h)
                                                     [M1; M2] [M1; M2])
let higgs_squark =
  if Flags.ckm_present then
    List.flatten (Product.list2 higgs_squark_a [1; 2] [1; 2]) @
    ThoList.flatmap higgs_squark_b [(1, 3); (2, 3); (3, 3); (3, 1); (3, 2)]
  else
    higgs_squark_a 1 1 @ higgs_squark_a 2 2 @ higgs_squark_b (3, 3)
```

** REVISED: Independent of the sign of CD, compatible with GS+. **

```
let goldstone_squark' g h m1 m2 =
  [ ((Phip, Sup (m1, -g), Sdown (m2, h)), Scalar_Scalar_Scalar 1,
          G_GSUSD (false,m1, m2, g, h));
    ((Phim, Sup (m1, g), Sdown (m2, -h)), Scalar_Scalar_Scalar 1,
          G_GSUSD (true,m1, m2, g, h)) ]
let goldstone_squark_a g h = goldstone_squark' g h M1 M1
let goldstone_squark_b (g, h) = List.flatten (Product.list2
```

```
        (goldstone_squark' g h) [M1; M2] [M1; M2])
  let goldstone_squark =
      List.flatten (Product.list2 goldstone_squark_a [1; 2] [1; 2]) @
      ThoList.flatmap goldstone_squark_b [(1, 3); (2, 3); (3, 3); (3, 1); (3, 2)]
```

BAUSTELLE: For the quartic scalar coupligs we does not allow *whiz_col*.

```
  let higgs_sneutrino4' g m =
    [ ((Hp, H_Heavy, Slepton (m, g), Sneutrino (-g)), Scalar4 1,
          G_HH2SLSN (false, m, g));
      ((Hm, H_Heavy, Slepton (m, -g), Sneutrino g), Scalar4 1,
          G_HH2SLSN (true, m, g));
      ((Hp, H_Light, Slepton (m, g), Sneutrino (-g)), Scalar4 1,
          G_HH1SLSN (false, m, g));
      ((Hm, H_Light, Slepton (m, -g), Sneutrino g), Scalar4 1,
          G_HH1SLSN (true, m, g));
      ((Hp, A, Slepton (m, g), Sneutrino (-g)), Scalar4 1,
          G_HASLSN (false, m, g));
      ((Hm, A, Slepton (m, -g), Sneutrino g), Scalar4 1,
          G_HASLSN (true, m, g)) ]
  let higgs_sneutrino4 g =
    ThoList.flatmap (higgs_sneutrino4' g) [M1; M2] @
    [ ((H_Heavy, H_Heavy, Sneutrino g, Sneutrino (-g)), Scalar4 1,
        G_H2H2SFSF (SN, M1, M1, g));
      ((H_Heavy, H_Light, Sneutrino g, Sneutrino (-g)), Scalar4 1,
        G_H1H2SFSF (SN, M1, M1, g));
      ((H_Light, H_Light, Sneutrino g, Sneutrino (-g)), Scalar4 1,
        G_H1H1SFSF (SN, M1, M1, g));
      ((Hp, Hm, Sneutrino g, Sneutrino (-g)), Scalar4 1, G_HHSFSF (SN, M1, M1, g)) ]
  let higgs_sfermion4' g m1 m2 =
    [ ((H_Heavy, H_Heavy, Slepton (m1, g), Slepton (m2, -g)), Scalar4 1,
        G_H2H2SFSF (SL, m1, m2, g));
      ((H_Heavy, H_Light, Slepton (m1, g), Slepton (m2, -g)), Scalar4 1,
        G_H1H2SFSF (SL, m1, m2, g));
      ((H_Light, H_Light, Slepton (m1, g), Slepton (m2, -g)), Scalar4 1,
        G_H1H1SFSF (SL, m1, m2, g));
      ((A, A, Slepton (m1, g), Slepton (m2, -g)), Scalar4 1,
        G_AASFSF (SL, m1, m2, g));
      ((Hp, Hm, Slepton (m1, g), Slepton (m2, -g)), Scalar4 1,
        G_HHSFSF (SL, m1, m2, g));
      ((H_Heavy, H_Heavy, Sup (m1, g), Sup (m2, -g)), Scalar4 1,
        G_H2H2SFSF (SU, m1, m2, g));
      ((H_Heavy, H_Heavy, Sdown (m1, g), Sdown (m2, -g)), Scalar4 1,
        G_H2H2SFSF (SD, m1, m2, g));
      ((H_Light, H_Light, Sup (m1, g), Sup (m2, -g)), Scalar4 1,
        G_H1H1SFSF (SU, m1, m2, g));
      ((H_Light, H_Light, Sdown (m1, g), Sdown (m2, -g)), Scalar4 1,
        G_H1H1SFSF (SD, m1, m2, g));
      ((H_Light, H_Heavy, Sup (m1, g), Sup (m2, -g)), Scalar4 1,
        G_H1H2SFSF (SU, m1, m2, g));
      ((H_Light, H_Heavy, Sdown (m1, g), Sdown (m2, -g)), Scalar4 1,
```

```
            G_H1H2SFSF (SD, m1, m2, g));
        ((Hp, Hm, Sup (m1, g), Sup (m2, −g)), Scalar4 1, G_HHSFSF (SU, m1, m2, g));
        ((Hp, Hm, Sdown (m1, g), Sdown (m2, −g)), Scalar4 1, G_HHSFSF (SD, m1, m2, g));
        ((A, A, Sup (m1, g), Sup (m2, −g)), Scalar4 1, G_AASFSF (SU, m1, m2, g));
        ((A, A, Sdown (m1, g), Sdown (m2, −g)), Scalar4 1, G_AASFSF (SD, m1, m2, g)) ]
let higgs_sfermion4 g = List.flatten (Product.list2 (higgs_sfermion4′ g)
                                        [M1; M2] [M1; M2])

let higgs_squark4′ g h m1 m2 =
    [ ((Hp, H_Light, Sup (m1, −g), Sdown (m2, h)), Scalar4 1,
          G_HH1SUSD (false, m1, m2, g, h));
      ((Hm, H_Light, Sup (m1, g), Sdown (m2, −h)), Scalar4 1,
          G_HH1SUSD (true, m1, m2, g, h));
      ((Hp, H_Heavy, Sup (m1, −g), Sdown (m2, h)), Scalar4 1,
          G_HH2SUSD (false, m1, m2, g, h));
      ((Hm, H_Heavy, Sup (m1, g), Sdown (m2, −h)), Scalar4 1,
          G_HH2SUSD (true, m1, m2, g, h));
      ((Hp, A, Sup (m1, −g), Sdown (m2, h)), Scalar4 1,
          G_HASUSD (false, m1, m2, g, h));
      ((Hm, A, Sup (m1, g), Sdown (m2, −h)), Scalar4 1,
          G_HASUSD (true, m1, m2, g, h)) ]
let higgs_squark4 g h = List.flatten (Product.list2 (higgs_squark4′ g h)
                                        [M1; M2] [M1; M2])

let higgs_gold_sneutrino′ g m =
    [ ((Hp, Phi0, Sneutrino (−g), Slepton (m, g)), Scalar4 1, G_HGSNSL (false, m, g));
      ((Hm, Phi0, Sneutrino g, Slepton (m, −g)), Scalar4 1, G_HGSNSL (true, m, g));
      ((H_Heavy, Phip, Sneutrino (−g), Slepton (m, g)), Scalar4 1,
          G_H2GSNSL (false, m, g));
      ((H_Heavy, Phim, Sneutrino g, Slepton (m, −g)), Scalar4 1,
          G_H2GSNSL (true, m, g));
      ((H_Light, Phip, Sneutrino (−g), Slepton (m, g)), Scalar4 1,
          G_H1GSNSL (false, m, g));
      ((H_Light, Phim, Sneutrino g, Slepton (m, −g)), Scalar4 1,
          G_H1GSNSL (true, m, g));
      ((A, Phip, Sneutrino (−g), Slepton (m, g)), Scalar4 1, G_AGSNSL (false, m, g));
      ((A, Phim, Sneutrino g, Slepton (m, −g)), Scalar4 1, G_AGSNSL (true, m, g));
      ((Phi0, Phip, Sneutrino (−g), Slepton (m, g)), Scalar4 1, G_GGSNSL (false, m, g));
      ((Phi0, Phim, Sneutrino g, Slepton (m, −g)), Scalar4 1, G_GGSNSL (true, m, g))]
let higgs_gold_sneutrino g =
    ThoList.flatmap (higgs_gold_sneutrino′ g) [M1; M2] @
    [ ((A, Phi0, Sneutrino g, Sneutrino (−g)), Scalar4 1,
          G_AG0SFSF (SN, M1, M1, g));
      ((Hp, Phim, Sneutrino g, Sneutrino (−g)), Scalar4 1,
          G_HGSFSF (SN, M1, M1, g));
      ((Hm, Phip, Sneutrino g, Sneutrino (−g)), Scalar4 1,
          G_HGSFSF (SN, M1, M1, g));
      ((Phip, Phim, Sneutrino g, Sneutrino (−g)), Scalar4 1,
          G_GGSFSF (SN, M1, M1, g));
      ((Phi0, Phi0, Sneutrino g, Sneutrino (−g)), Scalar4 1,
          G_G0G0SFSF (SN, M1, M1, g)) ]
```

let $higgs\_gold\_sfermion'\ g\ m1\ m2\ =$
$[\ ((A,\ Phi0,\ Slepton\ (m1, g),\ Slepton\ (m2, -g)),\ Scalar4\ 1,$
$G\_AG0SFSF\ (SL, m1, m2, g));$
$((Hp,\ Phim,\ Slepton\ (m1, g),\ Slepton\ (m2, -g)),\ Scalar4\ 1,$
$G\_HGSFSF\ (SL, m1, m2, g));$
$((Hm,\ Phip,\ Slepton\ (m1, g),\ Slepton\ (m2, -g)),\ Scalar4\ 1,$
$G\_HGSFSF\ (SL, m1, m2, g));$
$((Phip,\ Phim,\ Slepton\ (m1, g),\ Slepton\ (m2, -g)),\ Scalar4\ 1,$
$G\_GGSFSF\ (SL, m1, m2, g));$
$((Phi0,\ Phi0,\ Slepton\ (m1, g),\ Slepton\ (m2, -g)),\ Scalar4\ 1,$
$G\_G0G0SFSF\ (SL, m1, m2, g));$
$((A,\ Phi0,\ Sup\ (m1, g),\ Sup\ (m2, -g)),\ Scalar4\ 1,\ G\_AG0SFSF\ (SU, m1, m2, g));$
$((A,\ Phi0,\ Sdown\ (m1, g),\ Sdown\ (m2, -g)),\ Scalar4\ 1,$
$G\_AG0SFSF\ (SD, m1, m2, g));$
$((Hp,\ Phim,\ Sup\ (m1, g),\ Sup\ (m2, -g)),\ Scalar4\ 1,\ G\_HGSFSF\ (SU, m1, m2, g));$
$((Hm,\ Phip,\ Sup\ (m1, g),\ Sup\ (m2, -g)),\ Scalar4\ 1,\ G\_HGSFSF\ (SU, m1, m2, g));$
$((Hp,\ Phim,\ Sdown\ (m1, g),\ Sdown\ (m2, -g)),\ Scalar4\ 1,$
$G\_HGSFSF\ (SD, m1, m2, g));$
$((Hm,\ Phip,\ Sdown\ (m1, g),\ Sdown\ (m2, -g)),\ Scalar4\ 1,$
$G\_HGSFSF\ (SD, m1, m2, g));$
$((Phip,\ Phim,\ Sup\ (m1, g),\ Sup\ (m2, -g)),\ Scalar4\ 1,$
$G\_GGSFSF\ (SU, m1, m2, g));$
$((Phip,\ Phim,\ Sdown\ (m1, g),\ Sdown\ (m2, -g)),\ Scalar4\ 1,$
$G\_GGSFSF\ (SD, m1, m2, g));$
$((Phi0,\ Phi0,\ Sup\ (m1, g),\ Sup\ (m2, -g)),\ Scalar4\ 1,$
$G\_G0G0SFSF\ (SU, m1, m2, g));$
$((Phi0,\ Phi0,\ Sdown\ (m1, g),\ Sdown\ (m2, -g)),\ Scalar4\ 1,$
$G\_G0G0SFSF\ (SD, m1, m2, g))\ ]$
let $higgs\_gold\_sfermion\ g\ =\ List.flatten\ (Product.list2$
$(higgs\_gold\_sfermion'\ g)\ [M1; M2]\ [M1; M2])$

let $higgs\_gold\_squark'\ g\ h\ m1\ m2\ =$
$[\ ((Hp,\ Phi0,\ Sup\ (m1, -g),\ Sdown\ (m2, h)),\ Scalar4\ 1,$
$G\_HGSUSD\ (\mathsf{false}, m1, m2, g, h));$
$((Hm,\ Phi0,\ Sup\ (m1, g),\ Sdown\ (m2, -h)),\ Scalar4\ 1,$
$G\_HGSUSD\ (\mathsf{true}, m1, m2, g, h));$
$((H\_Heavy,\ Phip,\ Sup\ (m1, -g),\ Sdown\ (m2, h)),\ Scalar4\ 1,$
$G\_H2GSUSD\ (\mathsf{false}, m1, m2, g, h));$
$((H\_Heavy,\ Phim,\ Sup\ (m1, g),\ Sdown\ (m2, -h)),\ Scalar4\ 1,$
$G\_H2GSUSD\ (\mathsf{true}, m1, m2, g, h));$
$((H\_Light,\ Phip,\ Sup\ (m1, -g),\ Sdown\ (m2, h)),\ Scalar4\ 1,$
$G\_H1GSUSD\ (\mathsf{false}, m1, m2, g, h));$
$((H\_Light,\ Phim,\ Sup\ (m1, g),\ Sdown\ (m2, -h)),\ Scalar4\ 1,$
$G\_H1GSUSD\ (\mathsf{true}, m1, m2, g, h));$
$((A,\ Phip,\ Sup\ (m1, -g),\ Sdown\ (m2, h)),\ Scalar4\ 1,$
$G\_AGSUSD\ (\mathsf{false}, m1, m2, g, h));$
$((A,\ Phim,\ Sup\ (m1, g),\ Sdown\ (m2, -h)),\ Scalar4\ 1,$
$G\_AGSUSD\ (\mathsf{true}, m1, m2, g, h));$
$((Phi0,\ Phip,\ Sup\ (m1, -g),\ Sdown\ (m2, h)),\ Scalar4\ 1,$
$G\_GGSUSD\ (\mathsf{false}, m1, m2, g, h));$

```
      ((Phi0, Phim, Sup (m1, g), Sdown (m2, -h)), Scalar4 1,
           G_GGSUSD (true,m1, m2, g, h)) ]
  let higgs_gold_squark g h = List.flatten (Product.list2 (higgs_gold_squark'
               g h) [M1; M2] [M1; M2])

  let sneutrino4' (g, h) =
    [ ((Sneutrino g, Sneutrino h, Sneutrino (-g), Sneutrino (-h)), Scalar4 1,
          G_SN4 (g, h))]
  let sneutrino4 = ThoList.flatmap sneutrino4'
                  [(1, 1); (1, 2); (1, 3); (2, 2); (2, 3); (3, 3)]

  let sneu2_slep2_1' g h m1 m2 =
     ((Sneutrino (-g), Sneutrino g, Slepton (m1, -h), Slepton (m2, h)), Scalar4 1,
           G_SN2SL2_1 (m1, m2, g, h))
  let sneu2_slep2_2' (g, h) m1 m2 =
      ((Sneutrino g, Sneutrino (-h), Slepton (m1, -g), Slepton (m2, h)), Scalar4 1,
           G_SN2SL2_2 (m1, m2, g, h))
  let sneu2_slep2_1 g h = Product.list2 (sneu2_slep2_1' g h) [M1; M2] [M1; M2]
  let sneu2_slep2_2 (g, h) = Product.list2 (sneu2_slep2_2' (g, h)) [M1; M2] [M1; M2]
```

The 4-slepton-vertices have the following structure: The sleptons come up in pairs of a positive and a negative slepton of the same generation; there is no vertex with e.g. two negative selectrons and two positive smuons, that of course would be a contradiction to the conservation of the separate slepton numbers of each generation which is not implemented in the MSSM. Because there is no CKM-mixing for the sleptons (in case of massless neutrinos) we maximally have two different generations of sleptons in a 4-slepton-vertex.

```
  let slepton4_1gen' g (m1, m2, m3, m4) =
    [ ((Slepton (m1, -g), Slepton (m2, g), Slepton (m3, -g), Slepton (m4, g)),
          Scalar4 1, G_SL4 (m1, m2, m3, m4, g)) ]
  let slepton4_1gen g = ThoList.flatmap (slepton4_1gen' g) [(M1, M1, M1, M1);
      (M1, M1, M1, M2); (M1, M1, M2, M1); (M1, M1, M2, M2); (M1, M2, M1, M2); (M1, M2, M2, M
      (M1, M2, M2, M2); (M2, M1, M2, M2); (M2, M2, M2, M2) ]
  let slepton4_2gen' (g, h) (m1, m2) (m3, m4) =
     ((Slepton (m1, -g), Slepton (m2, g), Slepton (m3, -h), Slepton (m4, h)),
         Scalar4 1, G_SL4_2 (m1, m2, m3, m4, g, h))
  let slepton4_2gen (g, h) =
    Product.list2 (slepton4_2gen' (g, h)) [(M1, M1); (M1, M2); (M2, M1); (M2, M2)]
      [(M1, M1); (M1, M2); (M2, M1); (M2, M2)]

  let sneu2_squark2' g h m1 m2 =
     [ ((Sneutrino (-g), Sneutrino g, Sup (m1, -h), Sup (m2, h)), Scalar4 1,
            G_SN2SQ2 (SU, m1, m2, g, h));
        ((Sneutrino (-g), Sneutrino g, Sdown (m1, -h), Sdown (m2, h)), Scalar4 1,
            G_SN2SQ2 (SD, m1, m2, g, h)) ]
  let sneu2_squark2 g h = List.flatten (Product.list2 (sneu2_squark2' g h)
                                        [M1; M2] [M1; M2])

  let slepton2_squark2'' g h m1 m2 m3 m4 =
     [ ((Slepton (m1, -g), Slepton (m2, g), Sup (m3, -h), Sup (m4, h)), Scalar4 1,
            G_SL2SQ2 (SU, m1, m2, m3, m4, g, h));
        ((Slepton (m1, -g), Slepton (m2, g), Sdown (m3, -h), Sdown (m4, h)),
```

```
            Scalar4 1, G_SL2SQ2 (SD, m1, m2, m3, m4, g, h)) ]
  let slepton2_squark2' g h m1 m2 =
    List.flatten (Product.list2 (slepton2_squark2'' g h m1 m2) [M1; M2] [M1; M2])
  let slepton2_squark2 g h =
    List.flatten (Product.list2 (slepton2_squark2' g h) [M1; M2] [M1; M2])

  let slep_sneu_squark2'' g1 g2 g3 m1 m2 m3 =
    [ ((Sup (m1, -g1), Sdown (m2, g2), Slepton (m3, -g3), Sneutrino g3),
          Scalar4 1, G_SUSDSNSL (false,m1, m2, m3, g1, g2, g3));
      ((Sup (m1, g1), Sdown (m2, -g2), Slepton (m3, g3), Sneutrino (-g3)),
          Scalar4 1, G_SUSDSNSL (true,m1, m2, m3, g1, g2, g3)) ]
  let slep_sneu_squark2' g1 g2 g3 m1 =
    List.flatten (Product.list2 (slep_sneu_squark2'' g1 g2 g3 m1)
                    [M1; M2] [M1; M2])
  let slep_sneu_squark2 g1 g2 =
    List.flatten (Product.list2 (slep_sneu_squark2' g1 g2) [1; 2; 3] [M1; M2])
```

There are three kinds of 4-squark-vertices: Four up-Squarks, four down-squarks or two up- and two down-squarks.

```
  let sup4_1gen' g (m1, m2, m3, m4) =
    [ ((Sup (m1, -g), Sup (m2, g), Sup (m3, -g), Sup (m4, g)), Scalar4 1,
          G_SU4 (m1, m2, m3, m4, g)) ]
  let sup4_1gen g = ThoList.flatmap (sup4_1gen' g) [(M1, M1, M1, M1);
      (M1, M1, M1, M2); (M1, M1, M2, M1); (M1, M1, M2, M2); (M1, M2, M1, M2); (M1, M2, M2, M
      (M1, M2, M2, M2); (M2, M1, M2, M2); (M2, M2, M2, M2) ]
  let sup4_2gen' (g, h) (m1, m2) (m3, m4) =
     ((Sup (m1, -g), Sup (m2, g), Sup (m3, -h), Sup (m4, h)), Scalar4 1,
          G_SU4_2 (m1, m2, m3, m4, g, h))
  let sup4_2gen (g, h) =
    Product.list2 (sup4_2gen' (g, h)) [(M1, M1); (M1, M2); (M2, M1); (M2, M2)]
      [(M1, M1); (M1, M2); (M2, M1); (M2, M2)]

  let sdown4_1gen' g (m1, m2, m3, m4) =
    [ ((Sdown (m1, -g), Sdown (m2, g), Sdown (m3, -g), Sdown (m4, g)), Scalar4 1,
          G_SD4 (m1, m2, m3, m4, g)) ]
  let sdown4_1gen g = ThoList.flatmap (sdown4_1gen' g) [(M1, M1, M1, M1);
      (M1, M1, M1, M2); (M1, M1, M2, M1); (M1, M1, M2, M2); (M1, M2, M1, M2); (M1, M2, M2, M
      (M1, M2, M2, M2); (M2, M1, M2, M2); (M2, M2, M2, M2) ]
  let sdown4_2gen' (g, h) (m1, m2) (m3, m4) =
     ((Sdown (m1, -g), Sdown (m2, g), Sdown (m3, -h), Sdown (m4, h)), Scalar4 1,
          G_SD4_2 (m1, m2, m3, m4, g, h))
  let sdown4_2gen (g, h) =
    Product.list2 (sdown4_2gen' (g, h)) [(M1, M1); (M1, M2); (M2, M1); (M2, M2)]
      [(M1, M1); (M1, M2); (M2, M1); (M2, M2)]

  let sup2_sdown2_3 g1 g2 g3 g4 m1 m2 m3 m4 =
      ((Sup (m1, -g1), Sup (m2, g2), Sdown (m3, -g3), Sdown
          (m4, g4)), Scalar4 1, G_SU2SD2 (m1, m2, m3, m4, g1, g2, g3, g4))
  let sup2_sdown2_2 g1 g2 g3 g4 m1 m2 =
    Product.list2 (sup2_sdown2_3 g1 g2 g3 g4 m1 m2) [M1; M2] [M1; M2]
  let sup2_sdown2_1 g1 g2 g3 g4 =
    List.flatten (Product.list2 (sup2_sdown2_2 g1 g2 g3 g4) [M1; M2] [M1; M2])
```

```
let sup2_sdown2 g1 g2 =
  List.flatten (Product.list2 (sup2_sdown2_1 g1 g2) [1;2;3] [1;2;3])

let quartic_grav_gauge g m =
  [ ((Grino, Slepton (m, -g), Ga, L g), GBBG (1, Gravbar, SLRV, Psi), G_Gr4A_Sl (g,m));
    ((L (-g), Slepton (m, g), Ga, Grino), GBBG (1, Psibar, SLRV, Grav), G_Gr4A_Slc (g,m));
    ((Grino, Sup (m, -g), Ga, U g), GBBG (1, Gravbar, SLRV, Psi), G_Gr4A_Su (g,m));
    ((U (-g), Sup (m, g), Ga, Grino), GBBG (1, Psibar, SLRV, Grav), G_Gr4A_Suc (g,m));
    ((Grino, Sdown (m, -g), Ga, D g), GBBG (1, Gravbar, SLRV, Psi), G_Gr4A_Sd (g,m));
    ((D (-g), Sdown (m, g), Ga, Grino), GBBG (1, Psibar, SLRV, Grav), G_Gr4A_Sdc (g,m));
    ((Grino, Slepton (m, -g), Z, L g), GBBG (1, Gravbar, SLRV, Psi), G_Gr4Z_Sl (g,m));
    ((L (-g), Slepton (m, g), Z, Grino), GBBG (1, Psibar, SLRV, Grav), G_Gr4Z_Slc (g,m));
    ((Grino, Sup (m, -g), Z, U g), GBBG (1, Gravbar, SLRV, Psi), G_Gr4Z_Su (g,m));
    ((U (-g), Sup (m, g), Z, Grino), GBBG (1, Psibar, SLRV, Grav), G_Gr4Z_Suc (g,m));
    ((Grino, Sdown (m, -g), Z, D g), GBBG (1, Gravbar, SLRV, Psi), G_Gr4Z_Sd (g,m));
    ((D (-g), Sdown (m, g), Z, Grino), GBBG (1, Psibar, SLRV, Grav), G_Gr4Z_Sdc (g,m));
    ((Grino, Sup (m, -g), Gl, U g), GBBG (1, Gravbar, SLRV, Psi), G_Gr4Gl_Su (g,m));
    ((U (-g), Sup (m, g), Gl, Grino), GBBG (1, Psibar, SLRV, Grav), G_Gr4Gl_Suc (g,m));
    ((Grino, Sdown (m, -g), Gl, D g), GBBG (1, Gravbar, SLRV, Psi), G_Gr4Gl_Sd (g,m));
    ((D (-g), Sdown (m, g), Gl, Grino), GBBG (1, Psibar, SLRV, Grav), G_Gr4Gl_Sdc (g,m));
    ((Grino, Slepton (m, -g), Wm, N g), GBBG (1, Gravbar, SLV, Psi), G_Gr4W_Sl (g,m));
    ((N (-g), Slepton (m, g), Wp, Grino), GBBG (1, Psibar, SLV, Grav), G_Gr4Z_Slc (g,m));
    ((Grino, Sup (m, -g), Wp, D g), GBBG (1, Gravbar, SLV, Psi), G_Gr4W_Su (g,m));
    ((D (-g), Sup (m, g), Wm, Grino), GBBG (1, Psibar, SLV, Grav), G_Gr4W_Suc (g,m));
    ((Grino, Sdown (m, -g), Wm, U g), GBBG (1, Gravbar, SLV, Psi), G_Gr4W_Sd (g,m));
    ((U (-g), Sdown (m, g), Wp, Grino), GBBG (1, Psibar, SLV, Grav), G_Gr4W_Sdc (g,m)) ]

let quartic_grav_sneutrino g =
  [ ((Grino, Sneutrino (-g), Z, N g), GBBG (1, Gravbar, SLV, Psi), G_Gr4Z_Sn);
    ((N (-g), Sneutrino g, Z, Grino), GBBG (1, Psibar, SLV, Grav), G_Gr4Z_Snc);
    ((Grino, Sneutrino (-g), Wp, L g), GBBG (1, Gravbar, SLV, Psi), G_Gr4W_Sn);
    ((L (-g), Sneutrino g, Wm, Grino), GBBG (1, Psibar, SLV, Grav), G_Gr4W_Snc) ]

let quartic_grav_neu n =
  [ ((Grino, Wp, Wm, Neutralino n), GBBG (1, Gravbar, V2LR, Chi), G_Gr4_Neu n);
    ((Grino, H_Light, Z, Neutralino n), GBBG (1, Gravbar, SLRV, Chi), G_Gr4_Z_H1 n);
    ((Grino, H_Heavy, Z, Neutralino n), GBBG (1, Gravbar, SLRV, Chi), G_Gr4_Z_H2 n);
    ((Grino, A, Z, Neutralino n), GBBG (1, Gravbar, SLRV, Chi), G_Gr4_Z_H3 n);
    ((Grino, Hm, Wp, Neutralino n), GBBG (1, Gravbar, SLRV, Chi), G_Gr4_W_H n);
    ((Grino, Hp, Wm, Neutralino n), GBBG (1, Gravbar, SLRV, Chi), G_Gr4_W_Hc n) ]

let quartic_grav_char c =
  let cc = conj_char c in
  [ ((Grino, Wm, Ga, Chargino c), GBBG (1, Gravbar, V2LR, Psi), G_Gr4_A_Ch c);
    ((Grino, Wm, Z, Chargino c), GBBG (1, Gravbar, V2LR, Psi), G_Gr4_Z_Ch c);
    ((Chargino cc, Wp, Ga, Grino), GBBG ((-1), Psibar, V2LR, Grav), G_Gr4_A_Ch cc);
    ((Chargino cc, Wp, Z, Grino), GBBG ((-1), Psibar, V2LR, Grav), G_Gr4_Z_Ch cc);
    ((Grino, Hm, Ga, Chargino c), GBBG (1, Gravbar, SLRV, Psi), G_Gr4_H_A c);
    ((Chargino cc, Hp, Ga, Grino), GBBG (1, Psibar, SLRV, Grav), G_Gr4_H_A cc);
    ((Grino, Hm, Z, Chargino c), GBBG (1, Gravbar, SLRV, Psi), G_Gr4_H_Z c);
    ((Chargino cc, Hp, Z, Grino), GBBG (1, Psibar, SLRV, Grav), G_Gr4_H_Z cc)]
```

```
let quartic_gravitino =
  [((Grino, Gl, Gl, Gluino), GBBG (1, Gravbar, V2, Chi), G_GravGl)] @
  ThoList.flatmap quartic_grav_neu [N1; N2; N3; N4] @
  ThoList.flatmap quartic_grav_char [C1; C2] @
  List.flatten (Product.list2 quartic_grav_gauge [1; 2; 3] [M1; M2]) @
  ThoList.flatmap quartic_grav_sneutrino [1; 2; 3]

let vertices3'' =
  if Flags.ckm_present then
    (ThoList.flatmap electromagnetic_currents_3 [1;2;3] @
     ThoList.flatmap electromagnetic_currents_2 [C1; C2] @
     List.flatten (Product.list2
             electromagnetic_sfermion_currents [1;2;3] [M1; M2]) @
     ThoList.flatmap neutral_currents [1;2;3] @
     ThoList.flatmap neutral_sfermion_currents [1;2;3] @
     ThoList.flatmap charged_currents [1;2;3] @
     List.flatten (Product.list2 charged_slepton_currents [1;2;3]
                        [M1; M2]) @
     List.flatten (Product.list2 charged_quark_currents [1;2;3]
                        [1;2;3]) @
     List.flatten (Product.list2 charged_squark_currents [1;2;3]
                        [1;2;3]) @
     ThoList.flatmap yukawa_higgs_quark [(1,3);(2,3);(3,3);(3,1);(3,2)] @
     yukawa_higgs 3 @ yukawa_n @
     ThoList.flatmap yukawa_c [C1; C2] @
     ThoList.flatmap yukawa_cq [C1; C2] @
     List.flatten (Product.list2 charged_chargino_currents [N1; N2; N3; N4]
                        [C1; C2]) @ triple_gauge @
     ThoList.flatmap neutral_Z_1 [(N1, N2); (N1, N3); (N1, N4); (N2, N3); (N2, N4);
                                 (N3, N4)] @
     ThoList.flatmap neutral_Z_2 [N1; N2; N3; N4] @
     Product.list2 charged_Z [C1; C2] [C1; C2] @
     gauge_higgs @ higgs @ yukawa_higgs_2 @
     List.flatten (Product.list2 higgs_charg_neutr [N1; N2; N3; N4] [C1; C2]) @
     higgs_neutr @ higgs_sneutrino @ higgs_sfermion @
     higgs_squark @ yukawa_v @
     ThoList.flatmap col_currents [1;2;3] @
     List.flatten (Product.list2 col_sfermion_currents [1;2;3] [M1; M2]))
  else
    (ThoList.flatmap electromagnetic_currents_3 [1;2;3] @
     ThoList.flatmap electromagnetic_currents_2 [C1; C2] @
     List.flatten (Product.list2
             electromagnetic_sfermion_currents [1;2;3] [M1; M2]) @
     ThoList.flatmap neutral_currents [1;2;3] @
     ThoList.flatmap neutral_sfermion_currents [1;2;3] @
     ThoList.flatmap charged_currents [1;2;3] @
     List.flatten (Product.list2 charged_slepton_currents [1;2;3]
                        [M1; M2]) @
     charged_quark_currents 1 1 @
     charged_quark_currents 2 2 @
```

```
        charged_quark_currents 3 3 @
        charged_squark_currents 1 1 @
        charged_squark_currents 2 2 @
        charged_squark_currents 3 3 @
        ThoList.flatmap yukawa_higgs_quark [(3, 3)] @
        yukawa_higgs 3 @ yukawa_n @
        ThoList.flatmap yukawa_c [C1; C2] @
        ThoList.flatmap yukawa_cq [C1; C2] @
        List.flatten (Product.list2 charged_chargino_currents [N1; N2; N3; N4]
                          [C1; C2]) @ triple_gauge @
        ThoList.flatmap neutral_Z_1 [(N1, N2); (N1, N3); (N1, N4); (N2, N3); (N2, N4);
                                      (N3, N4)] @
        ThoList.flatmap neutral_Z_2 [N1; N2; N3; N4] @
        Product.list2 charged_Z [C1; C2] [C1; C2] @
        gauge_higgs @ higgs @ yukawa_higgs_2 @
        List.flatten (Product.list2 higgs_charg_neutr [N1; N2; N3; N4] [C1; C2]) @
        higgs_neutr @ higgs_sneutrino @ higgs_sfermion @
        higgs_squark @ yukawa_v @
        ThoList.flatmap col_currents [1; 2; 3] @
        List.flatten (Product.list2 col_sfermion_currents [1; 2; 3] [M1; M2]))
  let vertices3' =
    if Flags.gravitino then (vertices3'' @ triple_gravitino)
    else vertices3''
  let vertices3 =
    if Flags.include_goldstone then
      (vertices3' @ yukawa_goldstone 3 @
        gauge_higgs_gold @ higgs_gold @ yukawa_goldstone_2 @
        (if Flags.ckm_present then
          List.flatten (Product.list2 yukawa_goldstone_quark [1; 2; 3]
                          [1; 2; 3]) @
          List.flatten (Product.list2 goldstone_charg_neutr [N1; N2; N3; N4]
                          [C1; C2])
        else
          yukawa_goldstone_quark 1 1 @
          yukawa_goldstone_quark 2 2 @
          yukawa_goldstone_quark 3 3) @
        goldstone_neutr @ goldstone_sfermion @ goldstone_squark)
    else vertices3'
  let vertices4''' =
    (quartic_gauge @ higgs4 @ gauge_higgs4 @
     ThoList.flatmap gauge_sfermion4 [1; 2; 3] @
     gauge_squark4 @ gluon_w_squark @
     List.flatten (Product.list2 gluon2_squark2 [1; 2; 3] [M1; M2]) @
     ThoList.flatmap gluon_gauge_squark [1; 2; 3])
  let vertices4'' =
    if Flags.gravitino then (vertices4''' @ quartic_gravitino)
      else vertices4'''
 let vertices4' =
  if Flags.include_four then
```

```
    (vertices4'' @
      ThoList.flatmap higgs_sfermion4 [1; 2; 3] @
      ThoList.flatmap higgs_sneutrino4 [1; 2; 3] @
      List.flatten (Product.list2 higgs_squark4 [1; 2; 3] [1; 2; 3]) @
      sneutrino4 @
      List.flatten (Product.list2 sneu2_slep2_1 [1; 2; 3] [1; 2; 3]) @
      ThoList.flatmap sneu2_slep2_2 [(1, 2); (1, 3); (2, 3); (2, 1); (3, 1); (3, 2)] @
      ThoList.flatmap slepton4_1gen [1; 2; 3] @
      ThoList.flatmap slepton4_2gen [(1, 2); (1, 3); (2, 3)] @
      List.flatten (Product.list2 sneu2_squark2 [1; 2; 3] [1; 2; 3]) @
      List.flatten (Product.list2 slepton2_squark2 [1; 2; 3] [1; 2; 3]) @
      List.flatten (Product.list2 slep_sneu_squark2 [1; 2; 3] [1; 2; 3]) @
      ThoList.flatmap sup4_1gen [1; 2; 3] @
      ThoList.flatmap sup4_2gen [(1, 2); (1, 3); (2, 3)] @
      ThoList.flatmap sdown4_1gen [1; 2; 3] @
      ThoList.flatmap sdown4_2gen [(1, 2); (1, 3); (2, 3)] @
      List.flatten (Product.list2 sup2_sdown2 [1; 2; 3] [1; 2; 3]))
      else
    vertices4''
let vertices4 =
  if Flags.include_goldstone then
    (vertices4' @ higgs_gold4 @ gauge_higgs_gold4 @ goldstone4 @
      ThoList.flatmap higgs_gold_sneutrino [1; 2; 3] @
      ThoList.flatmap higgs_gold_sfermion [1; 2; 3] @
      List.flatten (Product.list2 higgs_gold_squark [1; 2; 3] [1; 2; 3]))
  else
    vertices4'

 let vertices () = (vertices3, vertices4, [])

 let table = F.of_vertices (vertices ())
 let fuse2 = F.fuse2 table
 let fuse3 = F.fuse3 table
 let fuse = F.fuse table
 let max_degree () = 4

 let flavor_of_string s =
   match s with
   | "e-" → L 1 | "e+" → L (−1)
   | "mu-" → L 2 | "mu+" → L (−2)
   | "tau-" → L 3 | "tau+" → L (−3)
   | "nue" → N 1 | "nuebar" → N (−1)
   | "numu" → N 2 | "numubar" → N (−2)
   | "nutau" → N 3 | "nutaubar" → N (−3)
   | "se1-" → Slepton (M1, 1) | "se1+" → Slepton (M1, −1)
   | "smu1-" → Slepton (M1, 2) | "smu1+" → Slepton (M1, −2)
   | "stau1-" → Slepton (M1, 3) | "stau1+" → Slepton (M1, −3)
   | "se2-" → Slepton (M2, 1) | "se2+" → Slepton (M2, −1)
   | "smu2-" → Slepton (M2, 2) | "smu2+" → Slepton (M2, −2)
   | "stau2-" → Slepton (M2, 3) | "stau2+" → Slepton (M2, −3)
   | "snue" → Sneutrino 1 | "snue*" → Sneutrino (−1)
   | "snumu" → Sneutrino 2 | "snumu*" → Sneutrino (−2)
```

```
    | "snutau" → Sneutrino 3 | "snutau*" → Sneutrino (−3)
    | "u" → U 1 | "ubar" → U (−1)
    | "c" → U 2 | "cbar" → U (−2)
    | "t" → U 3 | "tbar" → U (−3)
    | "d" → D 1 | "dbar" → D (−1)
    | "s" → D 2 | "sbar" → D (−2)
    | "b" → D 3 | "bbar" → D (−3)
    | "A" → Ga | "Z" | "Z0" → Z
    | "W+" → Wp | "W-" → Wm
    | "gl" | "g" → Gl
    | "H" → H_Heavy | "h" → H_Light | "A0" → A
    | "H+" → Hp | "H-" → Hm
    | "phi0" → Phi0 | "phi+" → Phip | "phim" → Phim
    | "su1" → Sup (M1, 1) | "su1c" → Sup (M1, −1)
    | "sc1" → Sup (M1, 2) | "sc1c" → Sup (M1, −2)
    | "st1" → Sup (M1, 3) | "st1c" → Sup (M1, −3)
    | "su2" → Sup (M2, 1) | "su2c" → Sup (M2, −1)
    | "sc2" → Sup (M2, 2) | "sc2c" → Sup (M2, −2)
    | "st2" → Sup (M2, 3) | "st2c" → Sup (M2, −3)
    | "sgl" | "sg" → Gluino
    | "sd1" → Sdown (M1, 1) | "sd1c" → Sdown (M1, −1)
    | "ss1" → Sdown (M1, 2) | "ss1c" → Sdown (M1, −2)
    | "sb1" → Sdown (M1, 3) | "sb1c" → Sdown (M1, −3)
    | "sd2" → Sdown (M2, 1) | "sd2c" → Sdown (M2, −1)
    | "ss2" → Sdown (M2, 2) | "ss2c" → Sdown (M2, −2)
    | "sb2" → Sdown (M2, 3) | "sb2c" → Sdown (M2, −3)
    | "neu1" → Neutralino N1 | "neu2" → Neutralino N2
    | "neu3" → Neutralino N3 | "neu4" → Neutralino N4
    | "ch1+" → Chargino C1 | "ch2+" → Chargino C2
    | "ch1-" → Chargino C1c | "ch2-" → Chargino C2c
    | "GR" → Grino
    | _ → invalid_arg "Modellib_MSSM.MSSM.flavor_of_string"

let flavor_to_string = function
    | L 1 → "e-" | L (−1) → "e+"
    | L 2 → "mu-" | L (−2) → "mu+"
    | L 3 → "tau-" | L (−3) → "tau+"
    | N 1 → "nue" | N (−1) → "nuebar"
    | N 2 → "numu" | N (−2) → "numubar"
    | N 3 → "nutau" | N (−3) → "nutaubar"
    | U 1 → "u" | U (−1) → "ubar"
    | U 2 → "c" | U (−2) → "cbar"
    | U 3 → "t" | U (−3) → "tbar"
    | D 1 → "d" | D (−1) → "dbar"
    | D 2 → "s" | D (−2) → "sbar"
    | D 3 → "b" | D (−3) → "bbar"
    | L _ → invalid_arg
          "Modellib_MSSM.MSSM.flavor_to_string: invalid lepton"
    | N _ → invalid_arg
          "Modellib_MSSM.MSSM.flavor_to_string: invalid neutrino"
```

```
  | U _ → invalid_arg
      "Modellib_MSSM.MSSM.flavor_to_string: invalid up type quark"
  | D _ → invalid_arg
      "Modellib_MSSM.MSSM.flavor_to_string: invalid down type quark"
  | Gl → "gl" | Gluino → "sgl"
  | Ga → "A" | Z → "Z" | Wp → "W+" | Wm → "W-"
  | Phip → "phi+" | Phim → "phi-" | Phi0 → "phi0"
  | H_Heavy → "H" | H_Light → "h" | A → "A0"
  | Hp → "H+" | Hm → "H-"
  | Slepton (M1, 1) → "se1-" | Slepton (M1, -1) → "se1+"
  | Slepton (M1, 2) → "smu1-" | Slepton (M1, -2) → "smu1+"
  | Slepton (M1, 3) → "stau1-" | Slepton (M1, -3) → "stau1+"
  | Slepton (M2, 1) → "se2-" | Slepton (M2, -1) → "se2+"
  | Slepton (M2, 2) → "smu2-" | Slepton (M2, -2) → "smu2+"
  | Slepton (M2, 3) → "stau2-" | Slepton (M2, -3) → "stau2+"
  | Sneutrino 1 → "snue" | Sneutrino (-1) → "snue*"
  | Sneutrino 2 → "snumu" | Sneutrino (-2) → "snumu*"
  | Sneutrino 3 → "snutau" | Sneutrino (-3) → "snutau*"
  | Sup (M1, 1) → "su1" | Sup (M1, -1) → "su1c"
  | Sup (M1, 2) → "sc1" | Sup (M1, -2) → "sc1c"
  | Sup (M1, 3) → "st1" | Sup (M1, -3) → "st1c"
  | Sup (M2, 1) → "su2" | Sup (M2, -1) → "su2c"
  | Sup (M2, 2) → "sc2" | Sup (M2, -2) → "sc2c"
  | Sup (M2, 3) → "st2" | Sup (M2, -3) → "st2c"
  | Sdown (M1, 1) → "sd1" | Sdown (M1, -1) → "sd1c"
  | Sdown (M1, 2) → "ss1" | Sdown (M1, -2) → "ss1c"
  | Sdown (M1, 3) → "sb1" | Sdown (M1, -3) → "sb1c"
  | Sdown (M2, 1) → "sd2" | Sdown (M2, -1) → "sd2c"
  | Sdown (M2, 2) → "ss2" | Sdown (M2, -2) → "ss2c"
  | Sdown (M2, 3) → "sb2" | Sdown (M2, -3) → "sb2c"
  | Neutralino N1 → "neu1"
  | Neutralino N2 → "neu2"
  | Neutralino N3 → "neu3"
  | Neutralino N4 → "neu4"
  | Slepton _ → invalid_arg
      "Modellib_MSSM.MSSM.flavor_to_string: invalid slepton"
  | Sneutrino _ → invalid_arg
      "Modellib_MSSM.MSSM.flavor_to_string: invalid sneutrino"
  | Sup _ → invalid_arg
      "Modellib_MSSM.MSSM.flavor_to_string: invalid up type squark"
  | Sdown _ → invalid_arg
      "Modellib_MSSM.MSSM.flavor_to_string: invalid down type squark"
  | Chargino C1 → "ch1+" | Chargino C1c → "ch1-"
  | Chargino C2 → "ch2+" | Chargino C2c → "ch2-"
  | Grino → "GR"

let flavor_symbol = function
  | L g when g > 0 → "l" ^ string_of_int g
  | L g → "l" ^ string_of_int (abs g) ^ "b"
  | N g when g > 0 → "n" ^ string_of_int g
```

```
  | N g → "n" ^ string_of_int (abs g) ^ "b"
  | U g when g > 0 → "u" ^ string_of_int g
  | U g → "u" ^ string_of_int (abs g) ^ "b"
  | D g when g > 0 → "d" ^ string_of_int g
  | D g → "d" ^ string_of_int (abs g) ^ "b"
  | Gl → "gl" | Ga → "a" | Z → "z"
  | Wp → "wp" | Wm → "wm"
  | Slepton (M1, g) when g > 0 → "sl1" ^ string_of_int g
  | Slepton (M1, g) → "sl1c" ^ string_of_int (abs g)
  | Slepton (M2, g) when g > 0 → "sl2" ^ string_of_int g
  | Slepton (M2, g) → "sl2c" ^ string_of_int (abs g)
  | Sneutrino g when g > 0 → "sn" ^ string_of_int g
  | Sneutrino g → "snc" ^ string_of_int (abs g)
  | Sup (M1, g) when g > 0 → "su1" ^ string_of_int g
  | Sup (M1, g) → "su1c" ^ string_of_int (abs g)
  | Sup (M2, g) when g > 0 → "su2" ^ string_of_int g
  | Sup (M2, g) → "su2c" ^ string_of_int (abs g)
  | Sdown (M1, g) when g > 0 → "sd1" ^ string_of_int g
  | Sdown (M1, g) → "sd1c" ^ string_of_int (abs g)
  | Sdown (M2, g) when g > 0 → "sd2" ^ string_of_int g
  | Sdown (M2, g) → "sd2c" ^ string_of_int (abs g)
  | Neutralino n → "neu" ^ (string_of_neu n)
  | Chargino c when (int_of_char c) > 0 → "cp" ^ string_of_char c
  | Chargino c → "cm" ^ string_of_int (abs (int_of_char c))
  | Gluino → "sgl" | Phip → "pp" | Phim → "pm" | Phi0 → "p0"
  | H_Heavy → "h0h" | H_Light → "h0l" | A → "a0"
  | Hp → "hp" | Hm → "hm" | Grino → "gv"

let flavor_to_TeX = function
  | L 1 → "e^-" | L (−1) → "e^+"
  | L 2 → "\\mu^-" | L (−2) → "\\mu^+"
  | L 3 → "\\tau^-" | L (−3) → "\\tau^+"
  | N 1 → "\\nu_e" | N (−1) → "\\bar{\\nu}_e"
  | N 2 → "\\nu_\\mu" | N (−2) → "\\bar{\\nu}_\\mu"
  | N 3 → "\\nu_\\tau" | N (−3) → "\\bar{\\nu}_\\tau"
  | U 1 → "u" | U (−1) → "\\bar{u}"
  | U 2 → "c" | U (−2) → "\\bar{c}"
  | U 3 → "t" | U (−3) → "\\bar{t}"
  | D 1 → "d" | D (−1) → "\\bar{d}"
  | D 2 → "s" | D (−2) → "\\bar{s}"
  | D 3 → "b" | D (−3) → "\\bar{b}"
  | L _ → invalid_arg
      "Modellib_MSSM.MSSM.flavor_to_TeX:␣invalid␣lepton"
  | N _ → invalid_arg
      "Modellib_MSSM.MSSM.flavor_to_TeX:␣invalid␣neutrino"
  | U _ → invalid_arg
      "Modellib_MSSM.MSSM.flavor_to_TeX:␣invalid␣up␣type␣quark"
  | D _ → invalid_arg
      "Modellib_MSSM.MSSM.flavor_to_TeX:␣invalid␣down␣type␣quark"
  | Gl → "g" | Gluino → "\\widetilde{g}"
```

```
| Ga → "\\gamma" | Z → "Z" | Wp → "W^+" | Wm → "W^-"
| Phip → "\\phi^+" | Phim → "\\phi^-" | Phi0 → "\\phi^0"
| H_Heavy → "H^0" | H_Light → "h^0" | A → "A^0"
| Hp → "H^+" | Hm → "H^-"
| Slepton (M1, 1) → "\\widetilde{e}_1^-"
| Slepton (M1, −1) → "\\widetilde{e}_1^+"
| Slepton (M1, 2) → "\\widetilde{\\mu}_1^-"
| Slepton (M1, −2) → "\\widetilde{\\mu}_1^+"
| Slepton (M1, 3) → "\\widetilde{\\tau}_1^-"
| Slepton (M1, −3) → "\\widetilde{\\tau}_1^+"
| Slepton (M2, 1) → "\\widetilde{e}_2^-"
| Slepton (M2, −1) → "\\widetilde{e}_2^+"
| Slepton (M2, 2) → "\\widetilde{\\mu}_2^-"
| Slepton (M2, −2) → "\\widetilde{\\mu}_2^+"
| Slepton (M2, 3) → "\\widetilde{\\tau}_2^-"
| Slepton (M2, −3) → "\\widetilde{\\tau}_2^+"
| Sneutrino 1 → "\\widetilde{\\nu}_e"
| Sneutrino (−1) → "\\widetilde{\\nu}_e^*"
| Sneutrino 2 → "\\widetilde{\\nu}_\\mu"
| Sneutrino (−2) → "\\widetilde{\\nu}_\\mu^*"
| Sneutrino 3 → "\\widetilde{\\nu}_\\tau"
| Sneutrino (−3) → "\\widetilde{\\nu}_\\tau^*"
| Sup (M1, 1) → "\\widetilde{u}_1"
| Sup (M1, −1) → "\\widetilde{u}_1^*"
| Sup (M1, 2) → "\\widetilde{c}_1"
| Sup (M1, −2) → "\\widetilde{c}_1^*"
| Sup (M1, 3) → "\\widetilde{t}_1"
| Sup (M1, −3) → "\\widetilde{t}_1^*"
| Sup (M2, 1) → "\\widetilde{u}_2"
| Sup (M2, −1) → "\\widetilde{u}_2^*"
| Sup (M2, 2) → "\\widetilde{c}_2"
| Sup (M2, −2) → "\\widetilde{c}_2^*"
| Sup (M2, 3) → "\\widetilde{t}_2"
| Sup (M2, −3) → "\\widetilde{t}_2^*"
| Sdown (M1, 1) → "\\widetilde{d}_1"
| Sdown (M1, −1) → "\\widetilde{d}_1^*"
| Sdown (M1, 2) → "\\widetilde{s}_1"
| Sdown (M1, −2) → "\\widetilde{s}_1^*"
| Sdown (M1, 3) → "\\widetilde{b}_1"
| Sdown (M1, −3) → "\\widetilde{b}_1^*"
| Sdown (M2, 1) → "\\widetilde{d}_2"
| Sdown (M2, −1) → "\\widetilde{d}_2^*"
| Sdown (M2, 2) → "\\widetilde{s}_2"
| Sdown (M2, −2) → "\\widetilde{s}_2^*"
| Sdown (M2, 3) → "\\widetilde{b}_2"
| Sdown (M2, −3) → "\\widetilde{b}_2^*"
| Neutralino N1 → "\\widetilde{\\chi}^0_1"
| Neutralino N2 → "\\widetilde{\\chi}^0_2"
| Neutralino N3 → "\\widetilde{\\chi}^0_3"
| Neutralino N4 → "\\widetilde{\\chi}^0_4"
```

```
  | Slepton _ → invalid_arg
      "Modellib_MSSM.MSSM.flavor_to_TeX:␣invalid␣slepton"
  | Sneutrino _ → invalid_arg
      "Modellib_MSSM.MSSM.flavor_to_TeX:␣invalid␣sneutrino"
  | Sup _ → invalid_arg
      "Modellib_MSSM.MSSM.flavor_to_TeX:␣invalid␣up␣type␣squark"
  | Sdown _ → invalid_arg
      "Modellib_MSSM.MSSM.flavor_to_TeX:␣invalid␣down␣type␣squark"
  | Chargino C1 → "\\widetilde{\\chi}_1^+"
  | Chargino C1c → "\\widetilde{\\chi}_1^-"
  | Chargino C2 → "\\widetilde{\\chi}_2^+"
  | Chargino C2c → "\\widetilde{\\chi}_2^-"
  | Grino → "\\widetilde{G}"

let pdg = function
  | L g when g > 0 → 9 + 2 × g
  | L g → − 9 + 2 × g
  | N g when g > 0 → 10 + 2 × g
  | N g → − 10 + 2 × g
  | U g when g > 0 → 2 × g
  | U g → 2 × g
  | D g when g > 0 → − 1 + 2 × g
  | D g → 1 + 2 × g
  | Gl → 21 | Ga → 22 | Z → 23
  | Wp → 24 | Wm → (−24)
  | H_Light → 25 | H_Heavy → 35 | A → 36
  | Hp → 37 | Hm → (−37)
  | Phip | Phim → 27 | Phi0 → 26
  | Slepton (M1, g) when g > 0 → 1000009 + 2 × g
  | Slepton (M1, g) → − 1000009 + 2 × g
  | Slepton (M2, g) when g > 0 → 2000009 + 2 × g
  | Slepton (M2, g) → − 2000009 + 2 × g
  | Sneutrino g when g > 0 → 1000010 + 2 × g
  | Sneutrino g → − 1000010 + 2 × g
  | Sup (M1, g) when g > 0 → 1000000 + 2 × g
  | Sup (M1, g) → − 1000000 + 2 × g
  | Sup (M2, g) when g > 0 → 2000000 + 2 × g
  | Sup (M2, g) → − 2000000 + 2 × g
  | Sdown (M1, g) when g > 0 → 999999 + 2 × g
  | Sdown (M1, g) → − 999999 + 2 × g
  | Sdown (M2, g) when g > 0 → 1999999 + 2 × g
  | Sdown (M2, g) → − 1999999 + 2 × g
  | Gluino → 1000021
  | Grino → 1000039
  | Chargino C1 → 1000024 | Chargino C1c → (−1000024)
  | Chargino C2 → 1000037 | Chargino C2c → (−1000037)
  | Neutralino N1 → 1000022 | Neutralino N2 → 1000023
  | Neutralino N3 → 1000025 | Neutralino N4 → 1000035
```

We must take care of the pdg numbers for the two different kinds of sfermions in the MSSM. The particle data group in its Monte Carlo particle numbering scheme takes only into account mixtures of the third generation squarks and the stau. For the other sfermions we will use the number of the lefthanded field for the lighter mixed state and the one for the righthanded for the heavier. Below are the official pdg numbers from the Particle Data Group. In order not to produce arrays with some million entries in the Fortran code for the masses and the widths we introduce our private pdg numbering scheme which only extends not too far beyond 42. Our private scheme then has the following pdf numbers (for the sparticles the subscripts L/R and $1/2$ are taken synonymously):

d	down-quark	1
u	up-quark	2
s	strange-quark	3
c	charm-quark	4
b	bottom-quark	5
t	top-quark	6
e^-	electron	11
ν_e	electron-neutrino	12
μ^-	muon	13
ν_μ	muon-neutrino	14
τ^-	tau	15
ν_τ	tau-neutrino	16
g	gluon	(9) 21
γ	photon	22
Z^0	Z-boson	23
W^+	W-boson	24
h^0	light Higgs boson	25
H^0	heavy Higgs boson	35
A^0	pseudoscalar Higgs	36
H^+	charged Higgs	37
$\widetilde{\psi}_\mu$	gravitino	39
$\widetilde{d}_L$	down-squark 1	41
$\widetilde{u}_L$	up-squark 1	42
$\widetilde{s}_L$	strange-squark 1	43
$\widetilde{c}_L$	charm-squark 1	44
$\widetilde{b}_L$	bottom-squark 1	45
$\widetilde{t}_L$	top-squark 1	46
$\widetilde{d}_R$	down-squark 2	47
$\widetilde{u}_R$	up-squark 2	48
$\widetilde{s}_R$	strange-squark 2	49
$\widetilde{c}_R$	charm-squark 2	50
$\widetilde{b}_R$	bottom-squark 2	51
$\widetilde{t}_R$	top-squark 2	52
$\widetilde{e}_L$	selectron 1	53
$\widetilde{\nu}_{e,L}$	electron-sneutrino	54
$\widetilde{\mu}_L$	smuon 1	55
$\widetilde{\nu}_{\mu,L}$	muon-sneutrino	56
$\widetilde{\tau}_L$	stau 1	57
$\widetilde{\nu}_{\tau,L}$	tau-sneutrino	58
$\widetilde{e}_R$	selectron 2	59
$\widetilde{\mu}_R$	smuon 2	61
$\widetilde{\tau}_R$	stau 2	63
$\widetilde{g}$	gluino	64
$\widetilde{\chi}^0_1$	neutralino 1	65
$\widetilde{\chi}^0_2$	neutralino 2	66
$\widetilde{\chi}^0_3$	neutralino 3	67
$\widetilde{\chi}^0_4$	neutralino 4	68
$\widetilde{\chi}^+_1$	chargino 1	69

```
let pdg_mw = function
  | L g when g > 0 → 9 + 2 × g
  | L g → − 9 + 2 × g
  | N g when g > 0 → 10 + 2 × g
  | N g → − 10 + 2 × g
  | U g when g > 0 → 2 × g
  | U g → 2 × g
  | D g when g > 0 → − 1 + 2 × g
  | D g → 1 + 2 × g
  | Gl → 21 | Ga → 22 | Z → 23
  | Wp → 24 | Wm → (−24)
  | H_Light → 25 | H_Heavy → 35 | A → 36
  | Hp → 37 | Hm → (−37)
  | Phip | Phim → 27 | Phi0 → 26
  | Sup (M1, g) when g > 0 → 40 + 2 × g
  | Sup (M1, g) → − 40 + 2 × g
  | Sup (M2, g) when g > 0 → 46 + 2 × g
  | Sup (M2, g) → − 46 + 2 × g
  | Sdown (M1, g) when g > 0 → 39 + 2 × g
  | Sdown (M1, g) → − 39 + 2 × g
  | Sdown (M2, g) when g > 0 → 45 + 2 × g
  | Sdown (M2, g) → − 45 + 2 × g
  | Slepton (M1, g) when g > 0 → 51 + 2 × g
  | Slepton (M1, g) → − 51 + 2 × g
  | Slepton (M2, g) when g > 0 → 57 + 2 × g
  | Slepton (M2, g) → − 57 + 2 × g
  | Sneutrino g when g > 0 → 52 + 2 × g
  | Sneutrino g → − 52 + 2 × g
  | Grino → 39
  | Gluino → 64
  | Chargino C1 → 69 | Chargino C1c → (−69)
  | Chargino C2 → 70 | Chargino C2c → (−70)
  | Neutralino N1 → 65 | Neutralino N2 → 66
  | Neutralino N3 → 67 | Neutralino N4 → 68

let mass_symbol f =
  "mass(" ^ string_of_int (abs (pdg_mw f)) ^ ")"

let width_symbol f =
  "width(" ^ string_of_int (abs (pdg_mw f)) ^ ")"

let conj_symbol = function
  | false, str → str
  | true, str → str ^ "_c"

let constant_symbol = function
  | Unit → "unit" | Pi → "PI"
  | Alpha_QED → "alpha" | E → "e" | G → "g" | Vev → "vev"
  | Sin2thw → "sin2thw" | Eidelta → "eidelta" | Mu → "mu" |
G_Z → "gz"
  | Sin a → "sin" ^ string_of_angle a | Cos a → "cos" ^ string_of_angle a
```

```
    | Sin2am2b → "sin2am2b" | Cos2am2b → "cos2am2b" | Sinamb →
"sinamb"
    | Sinapb → "sinapb" | Cosamb → "cosamb" | Cosapb → "cosapb"
    | Cos4be → "cos4be" | Sin4be → "sin4be" | Sin4al → "sin4al"
    | Sin2al → "sin2al" | Cos2al → "cos2al" | Sin2be → "sin2be"
    | Cos2be → "cos2be" | Tana → "tana" | Tanb → "tanb"
    | Q_lepton → "qlep" | Q_up → "qup" | Q_down → "qdwn"
    | Q_charg → "qchar"
    | V_CKM (g1, g2) → "vckm_" ^ string_of_int g1 ^ string_of_int g2
    | M_SF (f, g, m1, m2) → "mix_" ^ string_of_sff f ^ string_of_int g
        ^ string_of_sfm m1 ^ string_of_sfm m2
    | AL g → "al_" ^ string_of_int g
    | AD g → "ad_" ^ string_of_int g
    | AU g → "au_" ^ string_of_int g
    | A_0 (n1, n2) → "a0_" ^ string_of_neu n1 ^ string_of_neu n2
    | A_P (c1, c2) → "ap_" ^ string_of_char c1 ^ string_of_char c2
    | V_0 (n1, n2) → "v0_" ^ string_of_neu n1 ^ string_of_neu n2
    | V_P (c1, c2) → "vp_" ^ string_of_char c1 ^ string_of_char c2
    | M_N (n1, n2) → "mn_" ^ string_of_neu n1 ^ string_of_neu n2
    | M_U (c1, c2) → "mu_" ^ string_of_char c1 ^ string_of_char c2
    | M_V (c1, c2) → "mv_" ^ string_of_char c1 ^ string_of_char c2
    | L_NC (n, c) → "lnc_" ^ string_of_neu n ^ string_of_char c
    | R_NC (n, c) → "rnc_" ^ string_of_neu n ^ string_of_char c
    | L_CN (c, n) → "lcn_" ^ string_of_char c ^ string_of_neu n
    | R_CN (c, n) → "rcn_" ^ string_of_char c ^ string_of_neu n
    | L_NCH (n, c) → "lnch_" ^ string_of_neu n ^ string_of_char c
    | R_NCH (n, c) → "rnch_" ^ string_of_neu n ^ string_of_char c
    | L_CNG (c, n) → "lcng_" ^ string_of_char c ^ string_of_neu n
    | R_CNG (c, n) → "rcng_" ^ string_of_char c ^ string_of_neu n
    | S_NNA (n1, n2) → "snna_" ^ string_of_neu n1 ^ string_of_neu n2
    | P_NNA (n1, n2) → "pnna_" ^ string_of_neu n1 ^ string_of_neu n2
    | S_NNG (n1, n2) → "snng_" ^ string_of_neu n1 ^ string_of_neu n2
    | P_NNG (n1, n2) → "pnng_" ^ string_of_neu n1 ^ string_of_neu n2
    | S_NNH1 (n1, n2) → "snnh1_" ^ string_of_neu n1 ^ string_of_neu n2
    | P_NNH1 (n1, n2) → "pnnh1_" ^ string_of_neu n1 ^ string_of_neu n2
    | S_NNH2 (n1, n2) → "snnh2_" ^ string_of_neu n1 ^ string_of_neu n2
    | P_NNH2 (n1, n2) → "pnnh2_" ^ string_of_neu n1 ^ string_of_neu n2
    | G_NC_lepton → "gnclep" | G_NC_neutrino → "gncneu"
    | G_NC_up → "gncup" | G_NC_down → "gncdwn"
    | G_CC → "gcc"
    | G_CCQ (vc, g1, g2) → conj_symbol (vc, "gccq_" ^ string_of_int g1 ^ "_"
        ^ string_of_int g2)
    | I_Q_W → "iqw" | I_G_ZWW → "igzww"
    | G_WWWW → "gw4" | G_ZZWW → "gzzww"
    | G_PZWW → "gpzww" | G_PPWW → "gppww"
    | G_GH 1 → "ghaw"
    | G_GH 2 → "gh1az" | G_GH 3 → "gh2az"
    | G_GH 4 → "gh1ww" | G_GH 5 → "gh2ww"
    | G_GH 6 → "ghh1w" | G_GH 7 → "ghh2w"
    | G_GH 8 → "gh1zz" | G_GH 9 → "gh2zz"
```

```
| G_GH 10 → "ghhz" | G_GH 11 → "ghhp"
| G_GH _ → failwith "this␣G_GH␣coupling␣is␣not␣available"
| G_GLGLH → "gglglh" | G_GLGLHH → "gglglhh"
| G_GLGLA → "gglgla" | G_PPH → "gpph"
| G_PPHH → "gpphh" | G_PPA → "gppa"
| G_GHGo n → "g_hgh(" ^ string_of_int n ^ ")"
| G_GH4 1 → "gaazz" | G_GH4 2 → "gh1h1zz" | G_GH4 3 →
"gh2h2zz"
| G_GH4 4 → "ghphmzz" | G_GH4 5 → "ghphmpp" | G_GH4 6 →
"ghphmpz"
| G_GH4 7 → "ghh1wz" | G_GH4 8 → "ghh2wz"
| G_GH4 9 → "ghh1wp" | G_GH4 10 → "ghh2wp"
| G_GH4 11 → "gaaww" | G_GH4 12 → "gh1h1ww" | G_GH4 13 →
"gh2h2ww"
| G_GH4 14 → "ghhww" | G_GH4 15 → "ghawz" | G_GH4 16 →
"ghawp"
| G_GH4 _ → failwith "this␣G_GH4␣coupling␣is␣not␣available"
| G_CICIH1 (n1, n2) → "gcicih1_" ^ string_of_neu n1 ^ "_"
    ^ string_of_neu n2
| G_CICIH2 (n1, n2) → "gcicih2_" ^ string_of_neu n1 ^ "_"
    ^ string_of_neu n2
| G_CICIA (n1, n2) → "gcicia_" ^ string_of_neu n1 ^ "_"
    ^ string_of_neu n2
| G_CICIG (n1, n2) → "gcicig_" ^ string_of_neu n1 ^ "_"
    ^ string_of_neu n2
| G_H3 n → "gh3_" ^ string_of_int n
| G_H4 n → "gh4_" ^ string_of_int n
| G_HGo3 n → "ghg3_" ^ string_of_int n
| G_HGo4 n → "ghg4_" ^ string_of_int n
| G_GG4 n → "ggg4_" ^ string_of_int n
| G_strong → "gs" | G_SS → "gs**2"
| Gs → "gs"
| I_G_S → "igs"
| G_S_Sqrt → "gssq"
| G_NWC (n, c) → "gnwc_" ^ string_of_neu n ^ "_" ^ string_of_char c
| G_CWN (c, n) → "gcwn_" ^ string_of_char c ^ "_" ^ string_of_neu n
| G_CH1C (c1, c2) → "gch1c_" ^ string_of_char c1 ^ "_" ^ string_of_char c2
| G_CH2C (c1, c2) → "gch2c_" ^ string_of_char c1 ^ "_" ^ string_of_char c2
| G_CAC (c1, c2) → "gcac_" ^ string_of_char c1 ^ "_" ^ string_of_char c2
| G_CGC (c1, c2) → "gcgc_" ^ string_of_char c1 ^ "_" ^ string_of_char c2
| G_YUK (i, g) → "g_yuk" ^ string_of_int i ^ "_" ^ string_of_int g
| G_NZN (n1, n2) → "gnzn_" ^ string_of_neu n1 ^ "_" ^ string_of_neu n2
| G_CZC (c1, c2) → "gczc_" ^ string_of_char c1 ^ "_" ^ string_of_char
    c2
| G_YUK_1 (n, m) → "g_yuk1_" ^ string_of_int n ^ "_" ^ string_of_int m
| G_YUK_2 (n, m) → "g_yuk2_" ^ string_of_int n ^ "_" ^ string_of_int m
| G_YUK_3 (n, m) → "g_yuk3_" ^ string_of_int n ^ "_" ^ string_of_int m
| G_YUK_4 (n, m) → "g_yuk4_" ^ string_of_int n ^ "_" ^ string_of_int m
| G_YUK_C (vc, g, c, sf, m) → conj_symbol (vc, "g_yuk_ch" ^ string_of_char c
    ^ "_" ^ string_of_sff sf ^ string_of_sfm m ^ "_" ^ string_of_int g )
```

| $G\_YUK\_N\ (vc, g, n, sf, m) \rightarrow conj\_symbol\ (vc,$ "g_yuk_n" ^ $string\_of\_neu\ n$
  ^ "_" ^ $string\_of\_sff\ sf$ ^ $string\_of\_sfm\ m$ ^ "_" ^ $string\_of\_int\ g$)
| $G\_YUK\_G\ (vc, g, sf, m) \rightarrow conj\_symbol\ (vc,$ "g_yuk_g" ^ $string\_of\_sff\ sf$
  ^ $string\_of\_sfm\ m$ ^ "_" ^ $string\_of\_int\ g)$
| $G\_YUK\_Q\ (vc, g1, g2, c, sf, m) \rightarrow conj\_symbol\ (vc,$ "g_yuk_ch" ^ $string\_of\_char\ c$
  ^ "_" ^ $string\_of\_sff\ sf$ ^ $string\_of\_sfm\ m$ ^ "_" ^ $string\_of\_int\ g1$
  ^ "_" ^ $string\_of\_int\ g2)$
| $G\_NHC\ (n, c) \rightarrow$ "g_nhc_" ^ $string\_of\_neu\ n$ ^ "_" ^ $string\_of\_char\ c$
| $G\_CHN\ (c, n) \rightarrow$ "g_chn_" ^ $string\_of\_neu\ n$ ^ "_" ^ $string\_of\_char\ c$
| $G\_NGC\ (n, c) \rightarrow$ "g_ngc_" ^ $string\_of\_neu\ n$ ^ $string\_of\_char\ c$
| $G\_CGN\ (c, n) \rightarrow$ "g_cgn_" ^ $string\_of\_char\ c$ ^ $string\_of\_neu\ n$
| $SUM\_1 \rightarrow$ "sum1"
| $G\_SLSNW\ (vc, g, m) \rightarrow conj\_symbol\ (vc,$ "gsl" ^ $string\_of\_sfm\ m$ ^ "_"
  ^ $string\_of\_int\ g$ ^ "snw")
| $G\_ZSF\ (f, g, m1, m2) \rightarrow$ "g" ^ $string\_of\_sff\ f$ ^ $string\_of\_sfm\ m1$ ^ "z"
  ^ $string\_of\_sff\ f$ ^ $string\_of\_sfm\ m2$ ^ "_" ^ $string\_of\_int\ g$
| $G\_WWSFSF\ (f, g, m1, m2) \rightarrow$ "gww" ^ $string\_of\_sff\ f$ ^ $string\_of\_sfm\ m1$
  ^ $string\_of\_sff\ f$ ^ $string\_of\_sfm\ m2$ ^ "_" ^ $string\_of\_int\ g$
| $G\_WPSLSN\ (vc, g, m) \rightarrow conj\_symbol\ (vc,$ "gpwsl" ^ $string\_of\_sfm\ m$
  ^ "sn_" ^ $string\_of\_int\ g)$
| $G\_WZSLSN\ (vc, g, m) \rightarrow conj\_symbol\ (vc,$ "gwzsl" ^ $string\_of\_sfm\ m$
  ^ "sn_" ^ $string\_of\_int\ g)$
| $G\_H1SFSF\ (f, g, m1, m2) \rightarrow$ "gh1" ^ $string\_of\_sff\ f$ ^ $string\_of\_sfm\ m1$
  ^ $string\_of\_sff\ f$ ^ $string\_of\_sfm\ m2$ ^ "_" ^ $string\_of\_int\ g$
| $G\_H2SFSF\ (f, g, m1, m2) \rightarrow$ "gh2" ^ $string\_of\_sff\ f$ ^ $string\_of\_sfm\ m1$
  ^ $string\_of\_sff\ f$ ^ $string\_of\_sfm\ m2$ ^ "_" ^ $string\_of\_int\ g$
| $G\_ASFSF\ (f, g, m1, m2) \rightarrow$ "ga" ^ $string\_of\_sff\ f$ ^ $string\_of\_sfm\ m1$
  ^ $string\_of\_sff\ f$ ^ $string\_of\_sfm\ m2$ ^ "_" ^ $string\_of\_int\ g$
| $G\_HSNSL\ (vc, g, m) \rightarrow conj\_symbol\ (vc,$ "ghsnsl" ^ $string\_of\_sfm\ m$ ^ "_"
  ^ $string\_of\_int\ g)$
| $G\_GoSFSF\ (f, g, m1, m2) \rightarrow$ "ggo" ^ $string\_of\_sff\ f$ ^ $string\_of\_sfm\ m1$
  ^ $string\_of\_sff\ f$ ^ $string\_of\_sfm\ m2$ ^ "_" ^ $string\_of\_int\ g$
| $G\_GoSNSL\ (vc, g, m) \rightarrow conj\_symbol\ (vc,$ "ggosnsl" ^ $string\_of\_sfm\ m$ ^ "_"
  ^ $string\_of\_int\ g)$
| $G\_HSUSD\ (vc, m1, m2, g1, g2) \rightarrow conj\_symbol\ (vc,$ "ghsu" ^ $string\_of\_sfm\ m1$
  ^ "sd" ^ $string\_of\_sfm\ m2$ ^ "_" ^ $string\_of\_int\ g1$ ^ "_"
  ^ $string\_of\_int\ g2)$
| $G\_GSUSD\ (vc, m1, m2, g1, g2) \rightarrow conj\_symbol\ (vc,$ "ggsu" ^ $string\_of\_sfm\ m1$
  ^ "sd" ^ $string\_of\_sfm\ m2$ ^ "_" ^ $string\_of\_int\ g1$ ^ "_"
  ^ $string\_of\_int\ g2)$
| $G\_WPSUSD\ (vc, m1, m2, n, m) \rightarrow conj\_symbol\ (vc,$ "gpwpsu" ^ $string\_of\_sfm\ m1$
  ^ "sd" ^ $string\_of\_sfm\ m2$ ^ "_" ^ $string\_of\_int\ n$ ^ "_"
  ^ $string\_of\_int\ m)$
| $G\_WZSUSD\ (vc, m1, m2, n, m) \rightarrow conj\_symbol\ (vc,$ "gzwpsu" ^ $string\_of\_sfm\ m1$
  ^ "sd" ^ $string\_of\_sfm\ m2$ ^ "_" ^ $string\_of\_int\ n$ ^ "_"
  ^ $string\_of\_int\ m)$
| $G\_SWS\ (vc, g1, g2, m1, m2) \rightarrow conj\_symbol\ (vc,$ "gs" ^ $string\_of\_sfm\ m1$ ^ "ws"
  ^ $string\_of\_sfm\ m2$ ^ "_" ^ $string\_of\_int\ g1$ ^ "_" ^ $string\_of\_int\ g2)$
| $G\_GlGlSQSQ \rightarrow$ "gglglsqsq"
| $G\_PPSFSF\ f \rightarrow$ "gpp" ^ $string\_of\_sff\ f$ ^ $string\_of\_sff\ f$

```
| G_ZZSFSF (f, g, m1, m2) → "gzz" ^ string_of_sff f ^ string_of_sfm m1
    ^ string_of_sff f ^ string_of_sfm m2 ^ "_" ^ string_of_int g
| G_ZPSFSF (f, g, m1, m2) → "gzp" ^ string_of_sff f ^ string_of_sfm m1
    ^ string_of_sff f ^ string_of_sfm m2 ^ "_" ^ string_of_int g
| G_GlPSQSQ → "gglpsqsq"
| G_GlZSFSF (f, g, m1, m2) → "ggl" ^ string_of_sff f ^ string_of_sfm m1
    ^ string_of_sff f ^ string_of_sfm m2 ^ "_" ^ string_of_int g
| G_GlWSUSD (vc, m1, m2, g1, g2) → conj_symbol (vc, "gglwsu"
    ^ string_of_sfm m1 ^ "sd" ^ string_of_sfm m2 ^ "_" ^ string_of_int g1
    ^ "_" ^ string_of_int g2)
| G_GHGo4 1 → "gzzg0g0" | G_GHGo4 2 → "gzzgpgm"
| G_GHGo4 3 → "gppgpgm" | G_GHGo4 4 → "gzpgpgm"
| G_GHGo4 5 → "gwwgpgm" | G_GHGo4 6 → "gwwg0g0"
| G_GHGo4 7 → "gwzg0g" | G_GHGo4 8 → "gwzg0g"
| G_GHGo4 9 → "gwzh1g" | G_GHGo4 10 → "gwzh2g"
| G_GHGo4 11 → "gwph1g" | G_GHGo4 12 → "gwph2g"
| G_GHGo4 _ → failwith "Coupling G_GHGo4 is not available"
| G_HSF31 (h, g, m1, m2, f1, f2) → "g_" ^ string_of_higgs h ^
    string_of_int g ^ string_of_sfm m1 ^ string_of_sfm m2 ^
    string_of_sff f1 ^ string_of_sff f2
| G_HSF32 (h, g1, g2, m1, m2, f1, f2) → "g_" ^ string_of_higgs h ^
    string_of_int g1 ^ "_" ^ string_of_int g2 ^ string_of_sfm m1 ^
    string_of_sfm m2 ^ string_of_sff f1 ^ string_of_sff f2
| G_HSF41 (h, g, m1, m2, f1, f2) → "g_" ^ string_of_higgs h ^
    string_of_int g ^ string_of_sfm m1 ^ string_of_sfm m2 ^
    string_of_sff f1 ^ string_of_sff f2
| G_HSF42 (h, g1, g2, m1, m2, f1, f2) → "g_" ^ string_of_higgs h ^
    string_of_int g1 ^ "_" ^ string_of_int g2 ^ string_of_sfm m1 ^
    string_of_sfm m2 ^ string_of_sff f1 ^ string_of_sff f2
| G_H1H1SFSF (f, m1, m2, n) → "gh1h1" ^ string_of_sff f ^ string_of_sfm
    m1 ^ string_of_sff f ^ string_of_sfm m2 ^ "_" ^ string_of_int n
| G_H1H2SFSF (f, m1, m2, n) → "gh1h2" ^ string_of_sff f ^ string_of_sfm
    m1 ^ string_of_sff f ^ string_of_sfm m2 ^ "_" ^ string_of_int n
| G_H2H2SFSF (f, m1, m2, n) → "gh2h2" ^ string_of_sff f ^ string_of_sfm
    m1 ^ string_of_sff f ^ string_of_sfm m2 ^ "_" ^ string_of_int n
| G_HHSFSF (f, m1, m2, n) → "ghh" ^ string_of_sff f ^ string_of_sfm m1
    ^ string_of_sff f ^ string_of_sfm m2 ^ "_" ^ string_of_int n
| G_AASFSF (f, m1, m2, n) → "gaa" ^ string_of_sff f ^ string_of_sfm m1
    ^ string_of_sff f ^ string_of_sfm m2 ^ "_" ^ string_of_int n
| G_HH1SUSD (vc, m1, m2, g1, g2) → conj_symbol (vc, "ghh1su"
    ^ string_of_sfm m1 ^ "sd" ^ string_of_sfm m2 ^ "_" ^ string_of_int g1
    ^ "_" ^ string_of_int g2)
| G_HH2SUSD (vc, m1, m2, g1, g2) → conj_symbol (vc, "ghh2su"
    ^ string_of_sfm m1 ^ "sd" ^ string_of_sfm m2 ^ "_" ^ string_of_int g1
    ^ "_" ^ string_of_int g2)
| G_HASUSD (vc, m1, m2, g1, g2) → conj_symbol (vc, "ghasu"
    ^ string_of_sfm m1 ^ "sd" ^ string_of_sfm m2 ^ "_"
    ^ string_of_int g1 ^ "_" ^ string_of_int g2 ^ "_c")
| G_HH1SLSN (vc, m, g) → conj_symbol (vc, "ghh1sl" ^ string_of_sfm m
                                      ^ "sn_" ^ string_of_int g)
```

```
| G_HH2SLSN (vc, m, g) → conj_symbol (vc, "ghh2sl" ^ string_of_sfm m
                                       ^ "sn_" ^ string_of_int g)
| G_HASLSN (vc, m, g) → conj_symbol (vc, "ghasl" ^ string_of_sfm m
                                       ^ "sn_" ^ string_of_int g)
| G_AG0SFSF (f, m1, m2, n) → "gag0" ^ string_of_sff f ^ string_of_sfm m1
    ^ string_of_sff f ^ string_of_sfm m2 ^ "_" ^ string_of_int n
| G_HGSFSF (f, m1, m2, n) → "ghg" ^ string_of_sff f ^ string_of_sfm m1
    ^ string_of_sff f ^ string_of_sfm m1 ^ "_" ^ string_of_int n
| G_GGSFSF (f, m1, m2, n) → "ggg" ^ string_of_sff f ^ string_of_sfm m1
    ^ string_of_sff f ^ string_of_sfm m2 ^ "_" ^ string_of_int n
| G_G0G0SFSF (f, m1, m2, n) → "gg0g0" ^ string_of_sff f ^ string_of_sfm m1
    ^ string_of_sff f ^ string_of_sfm m2 ^ "_" ^ string_of_int n
| G_HGSNSL (vc, m, n) → conj_symbol (vc, "ghgsnsl" ^ string_of_sfm m ^ "_"
    ^ string_of_int n)
| G_H1GSNSL (vc, m, n) → conj_symbol (vc, "gh1gsnsl" ^ string_of_sfm m ^ "_"
    ^ string_of_int n)
| G_H2GSNSL (vc, m, n) → conj_symbol (vc, "gh2gsnsl" ^ string_of_sfm m ^ "_"
    ^ string_of_int n)
| G_AGSNSL (vc, m, n) → conj_symbol (vc, "gagsnsl" ^ string_of_sfm m ^ "_"
    ^ string_of_int n)
| G_GGSNSL (vc, m, n) → conj_symbol (vc, "gggsnsl" ^ string_of_sfm m ^ "_"
    ^ string_of_int n)
| G_HGSUSD (vc, m1, m2, g1, g2) → conj_symbol (vc, "gghpsu" ^ string_of_sfm m1
    ^ "sd" ^ string_of_sfm m2 ^ "_" ^ string_of_int g1 ^ "_"
    ^ string_of_int g2)
| G_H1GSUSD (vc, m1, m2, g1, g2) → conj_symbol (vc, "gh1gpsu" ^ string_of_sfm m1
    ^ "sd" ^ string_of_sfm m2 ^ "_" ^ string_of_int g1 ^ "_"
    ^ string_of_int g2)
| G_H2GSUSD (vc, m1, m2, g1, g2) → conj_symbol (vc, "gh2gpsu" ^ string_of_sfm m1
    ^ "sd" ^ string_of_sfm m2 ^ "_" ^ string_of_int g1 ^ "_"
    ^ string_of_int g2)
| G_AGSUSD (vc, m1, m2, g1, g2) → conj_symbol (vc, "gagpsu" ^ string_of_sfm m1
    ^ "sd" ^ string_of_sfm m2 ^ "_" ^ string_of_int g1 ^ "_"
    ^ string_of_int g2)
| G_GGSUSD (vc, m1, m2, g1, g2) → conj_symbol (vc, "gggpsu" ^ string_of_sfm m1
    ^ "sd" ^ string_of_sfm m2 ^ "_" ^ string_of_int g1 ^ "_"
    ^ string_of_int g2)
| G_SN4 (g1, g2) → "gsn4_" ^ string_of_int g1 ^ "_" ^ string_of_int g2
| G_SN2SL2_1 (m1, m2, g1, g2) → "gsl_" ^ string_of_int g1 ^ "_sl_"
    ^ string_of_int g1 ^ "_sl" ^ string_of_sfm m1 ^ "_" ^ string_of_int g2
    ^ "_sl" ^ string_of_sfm m2 ^ "_" ^ string_of_int g2
| G_SN2SL2_2 (m1, m2, g1, g2) → "gsl_" ^ string_of_int g1 ^ "_sl_"
    ^ string_of_int g2 ^ "_sl" ^ string_of_sfm m1 ^ "_" ^ string_of_int g1
    ^ "_sl" ^ string_of_sfm m2 ^ "_" ^ string_of_int g2 ^ "_mix"
| G_SF4 (f1, f2, m1, m2, m3, m4, g1, g2) → "gsf" ^ string_of_sff f1 ^
    string_of_sff f2 ^ string_of_sfm m1 ^ string_of_sfm m2 ^
    string_of_sfm m3 ^ string_of_sfm m4 ^ string_of_int g1 ^
    string_of_int g2
| G_SF4_3 (f1, f2, m1, m2, m3, m4, g1, g2, g3) → "gsf" ^ string_of_sff f1 ^
    string_of_sff f2 ^ string_of_sfm m1 ^ string_of_sfm m2 ^
```

```
      string_of_sfm m3 ^ string_of_sfm m4 ^ string_of_int g1 ^
      string_of_int g2 ^ "_" ^ string_of_int g3
  | G_SF4_4 (f1, f2, m1, m2, m3, m4, g1, g2, g3, g4) → "gsf" ^ string_of_sff f1 ^
      string_of_sff f2 ^ string_of_sfm m1 ^ string_of_sfm m2 ^
      string_of_sfm m3 ^ string_of_sfm m4 ^ string_of_int g1 ^ "_" ^
      string_of_int g2 ^ string_of_int g3 ^ "_" ^ string_of_int g4
  | G_SL4 (m1, m2, m3, m4, g) → "gsl" ^ string_of_sfm m1 ^ "_"
      ^ "sl" ^ string_of_sfm m2 ^ "_" ^ "sl" ^ string_of_sfm m3 ^ "_"
      ^ "sl" ^ string_of_sfm m4 ^ "_" ^ string_of_int g
  | G_SL4_2 (m1, m2, m3, m4, g1, g2) → "gsl" ^ string_of_sfm m1 ^ "_"
      ^ "sl" ^ string_of_sfm m2 ^ "_" ^ "sl" ^ string_of_sfm m3 ^ "_"
      ^ "sl" ^ string_of_sfm m4 ^ "_" ^ string_of_int g1 ^ "_" ^
      string_of_int g2
  | G_SN2SQ2 (f, m1, m2, g1, g2) → "gsn_" ^ string_of_int g1 ^ "_sn_"
      ^ string_of_int g1 ^ "_" ^ string_of_sff f ^ string_of_sfm m1 ^ "_"
      ^ string_of_int g2 ^ "_" ^ string_of_sff f ^ string_of_sfm m2 ^ "_"
      ^ string_of_int g2
  | G_SL2SQ2 (f, m1, m2, m3, m4, g1, g2) → "gsl" ^ string_of_sfm m1 ^ "_"
      ^ string_of_int g1 ^ "_sl" ^ string_of_sfm m2 ^ "_" ^ string_of_int g1
      ^ "_" ^ string_of_sff f ^ string_of_sfm m3 ^ "_" ^ string_of_int g2
      ^ "_" ^ string_of_sff f ^ string_of_sfm m4 ^ "_" ^ string_of_int g2
  | G_SUSDSNSL (vc, m1, m2, m3, g1, g2, g3) → conj_symbol (vc, "gsl"
      ^ string_of_sfm m3 ^ "_" ^ string_of_int g3 ^ "_sn_" ^ string_of_int g3
      ^ "_su" ^ string_of_sfm m1 ^ "_" ^ string_of_int g1 ^ "_sd"
      ^ string_of_sfm m2 ^ "_" ^ string_of_int g2)
  | G_SU4 (m1, m2, m3, m4, g) → "gsu" ^ string_of_sfm m1 ^ "_"
      ^ "_su" ^ string_of_sfm m2 ^ "_" ^ "_su" ^ string_of_sfm m3 ^ "_" ^
      "_su" ^ string_of_sfm m4 ^ "_" ^ string_of_int g
  | G_SU4_2 (m1, m2, m3, m4, g1, g2) → "gsu" ^ string_of_sfm m1 ^ "_"
      ^ "_su" ^ string_of_sfm m2 ^ "_" ^ "_su" ^ string_of_sfm m3 ^ "_" ^
      "_su" ^ string_of_sfm m4 ^ "_" ^ string_of_int g1 ^ "_" ^
      string_of_int g2
  | G_SD4 (m1, m2, m3, m4, g) → "gsd" ^ string_of_sfm m1 ^ "_"
      ^ "_sd" ^ string_of_sfm m2 ^ "_" ^ "_sd" ^ string_of_sfm m3 ^ "_"
      ^ "_sd" ^ string_of_sfm m4 ^ "_" ^ string_of_int g
  | G_SD4_2 (m1, m2, m3, m4, g1, g2) → "gsd" ^ string_of_sfm m1 ^ "_"
      ^ "_sd" ^ string_of_sfm m2 ^ "_" ^ "_sd" ^ string_of_sfm m3 ^ "_"
      ^ "_sd" ^ string_of_sfm m4 ^ "_" ^ string_of_int g1 ^ "_" ^
      string_of_int g2
  | G_SU2SD2 (m1, m2, m3, m4, g1, g2, g3, g4) → "gsu" ^ string_of_sfm m1
      ^ "_" ^ string_of_int g1 ^ "_su" ^ string_of_sfm m2 ^ "_"
      ^ string_of_int g2 ^ "_sd" ^ string_of_sfm m3 ^ "_" ^ string_of_int g3
      ^ "_sd" ^ string_of_sfm m4 ^ "_" ^ string_of_int g4
  | M f → "mass" ^ flavor_symbol f
  | W f → "width" ^ flavor_symbol f
  | G_Grav → "ggrav" | G_Gr_Ch C1 → "ggrch1" | G_Gr_Ch C2 →
"ggrch2"
  | G_Gr_Ch C1c → "ggrch1c" | G_Gr_Ch C2c → "ggrch2c"
  | G_Gr_Z_Neu n → "ggrzneu" ^ string_of_neu n
  | G_Gr_A_Neu n → "ggraneu" ^ string_of_neu n
```

```
| G_Gr4_Neu n → "ggr4neu" ^ string_of_neu n
| G_Gr4_A_Ch C1 → "ggr4ach1" | G_Gr4_A_Ch C2 → "ggr4ach2"
| G_Gr4_A_Ch C1c → "ggr4ach1c" | G_Gr4_A_Ch C2c →
"ggr4ach2c"
| G_Gr4_Z_Ch C1 → "ggr4zch1" | G_Gr4_Z_Ch C2 → "ggr4zch2"
| G_Gr4_Z_Ch C1c → "ggr4zch1c" | G_Gr4_Z_Ch C2c →
"ggr4zch2c"
| G_Grav_N → "ggravn"
| G_GravGl → "gs * ggrav"
| G_Grav_L (g, m) → "ggravl" ^ string_of_int g ^ string_of_sfm m
| G_Grav_Lc (g, m) → "ggravl" ^ string_of_int g ^ string_of_sfm m ^ "c"
| G_Grav_U (g, m) → "ggravu" ^ string_of_int g ^ string_of_sfm m
| G_Grav_Uc (g, m) → "ggravu" ^ string_of_int g ^ string_of_sfm m ^ "c"
| G_Grav_D (g, m) → "ggravd" ^ string_of_int g ^ string_of_sfm m
| G_Grav_Dc (g, m) → "ggravd" ^ string_of_int g ^ string_of_sfm m ^ "c"
| G_Gr_H_Ch C1 → "ggrhch1" | G_Gr_H_Ch C2 → "ggrhch2"
| G_Gr_H_Ch C1c → "ggrhch1c" | G_Gr_H_Ch C2c → "ggrhch2c"
| G_Gr_H1_Neu n → "ggrh1neu" ^ string_of_neu n
| G_Gr_H2_Neu n → "ggrh2neu" ^ string_of_neu n
| G_Gr_H3_Neu n → "ggrh3neu" ^ string_of_neu n
| G_Gr4A_Sl (g, m) → "ggr4asl" ^ string_of_int g ^ string_of_sfm m
| G_Gr4A_Slc (g, m) → "ggr4asl" ^ string_of_int g ^ string_of_sfm m ^ "c"
| G_Gr4A_Su (g, m) → "ggr4asu" ^ string_of_int g ^ string_of_sfm m
| G_Gr4A_Suc (g, m) → "ggr4asu" ^ string_of_int g ^ string_of_sfm m ^ "c"
| G_Gr4A_Sd (g, m) → "ggr4asd" ^ string_of_int g ^ string_of_sfm m
| G_Gr4A_Sdc (g, m) → "ggr4asd" ^ string_of_int g ^ string_of_sfm m ^ "c"
| G_Gr4Z_Sn → "ggr4zsn" | G_Gr4Z_Snc → "ggr4zsnc"
| G_Gr4Z_Sl (g, m) → "ggr4zsl" ^ string_of_int g ^ string_of_sfm m
| G_Gr4Z_Slc (g, m) → "ggr4zsl" ^ string_of_int g ^ string_of_sfm m ^ "c"
| G_Gr4Z_Su (g, m) → "ggr4zsu" ^ string_of_int g ^ string_of_sfm m
| G_Gr4Z_Suc (g, m) → "ggr4zsu" ^ string_of_int g ^ string_of_sfm m ^ "c"
| G_Gr4Z_Sd (g, m) → "ggr4zsd" ^ string_of_int g ^ string_of_sfm m
| G_Gr4Z_Sdc (g, m) → "ggr4zsd" ^ string_of_int g ^ string_of_sfm m ^ "c"
| G_Gr4W_Sl (g, m) → "ggr4wsl" ^ string_of_int g ^ string_of_sfm m
| G_Gr4W_Slc (g, m) → "ggr4wsl" ^ string_of_int g ^ string_of_sfm m ^ "c"
| G_Gr4W_Su (g, m) → "ggr4wsu" ^ string_of_int g ^ string_of_sfm m
| G_Gr4W_Suc (g, m) → "ggr4wsu" ^ string_of_int g ^ string_of_sfm m ^ "c"
| G_Gr4W_Sd (g, m) → "ggr4wsd" ^ string_of_int g ^ string_of_sfm m
| G_Gr4W_Sdc (g, m) → "ggr4wsd" ^ string_of_int g ^ string_of_sfm m ^ "c"
| G_Gr4Gl_Su (g, m) → "ggr4glsu" ^ string_of_int g ^ string_of_sfm m
| G_Gr4Gl_Suc (g, m) → "ggr4glsu" ^ string_of_int g ^ string_of_sfm m ^ "c"
| G_Gr4Gl_Sd (g, m) → "ggr4glsd" ^ string_of_int g ^ string_of_sfm m
| G_Gr4Gl_Sdc (g, m) → "ggr4glsd" ^ string_of_int g ^ string_of_sfm m ^ "c"
| G_Gr4_Z_H1 n → "ggr4zh1_" ^ string_of_neu n
| G_Gr4_Z_H2 n → "ggr4zh2_" ^ string_of_neu n
| G_Gr4_Z_H3 n → "ggr4zh3_" ^ string_of_neu n
| G_Gr4_W_H n → "ggr4wh_" ^ string_of_neu n
| G_Gr4_W_Hc n → "ggr4whc_" ^ string_of_neu n
| G_Gr4_H_A C1 → "ggr4ha1" | G_Gr4_H_A C2 → "ggr4ha2"
| G_Gr4_H_A C1c → "ggr4ha1c" | G_Gr4_H_A C2c → "ggr4ha2c"
```

```
  | G_Gr4_H_Z C1 → "ggr4hz1" | G_Gr4_H_Z C2 → "ggr4hz2"
  | G_Gr4_H_Z C1c → "ggr4hz1c" | G_Gr4_H_Z C2c → "ggr4hz2c"
  | G_Gr4W_Sn → "ggr4wsn"
  | G_Gr4W_Snc → "ggr4wsnc"
end
```

13.9 Interface of Modellib_NMSSM

13.9.1 Extended Supersymmetric Models

We do not introduce the possibility here of using four point couplings or not. We simply add the relevant and leave the rest out. No possibility for Goldstone bosons is given. But we allow for CKM mixing.

```
module type NMSSM_flags =
  sig
    val ckm_present : bool
    val higgs_triangle : bool (* Hγγ, Hgγ and Hggcouplings*)
  end

module NMSSM : NMSSM_flags
module NMSSM_CKM : NMSSM_flags
module NMSSM_Hgg : NMSSM_flags
module NMSSM_func : functor (F : NMSSM_flags) → Model.T with module Ch = Charges.QQ
```

13.10 Implementation of Modellib_NMSSM

```
let rcs_file = RCS.parse "Modellib_NMSSM" ["NMSSM"]
    { RCS.revision = "$Revision:␣6465␣$";
      RCS.date = "$Date:␣2015-01-10␣16:22:31␣+0100␣(Sat,␣10␣Jan␣2015)␣$";
      RCS.author = "$Author:␣jr_reuter␣$";
      RCS.source
        = "$URL:␣svn+ssh://jr_reuter@login.hepforge.org/hepforge/svn/whizard/trunk/omega/sr
```

13.10.1 Next-to-Minimal Supersymmetric Standard Model

This is based on the NMSSM implementation by Felix Braam. Note that for the Higgs sector vertices the conventions of the Franke/Fraas paper have been used.

```
module type NMSSM_flags =
  sig
    val ckm_present : bool
    val higgs_triangle : bool
  end

module NMSSM : NMSSM_flags =
  struct
    let ckm_present = false
```

```
    let higgs_triangle = false
  end

module NMSSM_CKM : NMSSM_flags =
  struct
    let ckm_present = true
    let higgs_triangle = false
  end

module NMSSM_Hgg : NMSSM_flags =
  struct
    let ckm_present = false
    let higgs_triangle = true
  end

module NMSSM_func (Flags : NMSSM_flags) =
  struct
    let rcs = RCS.rename rcs_file "Modellib_NMSSM.NMSSM"
        [ "NMSSM" ]

    open Coupling

    let default_width = ref Timelike
    let use_fudged_width = ref false

    let options = Options.create
      [ "constant_width", Arg.Unit (fun () → default_width := Constant),
        "use␣constant␣width␣(also␣in␣t-channel)";
        "fudged_width", Arg.Set use_fudged_width,
        "use␣fudge␣factor␣for␣charge␣particle␣width";
        "custom_width", Arg.String (fun f → default_width := Custom f),
        "use␣custom␣width";
        "cancel_widths", Arg.Unit (fun () → default_width := Vanishing),
        "use␣vanishing␣width"]
```

Yields a list of tuples consistig of the off-diag combinations of the elements in ”set”.

```
    let choose2 set =
      List.map (function [x; y] → (x, y) | _ → failwith "choose2")
        (Combinatorics.choose 2 set)
```

pairs appends the diagonal combinations to *choose2*.

```
    let rec diag = function
      | [] → []
      | x1 :: rest → (x1, x1) :: diag rest

   let pairs l = choose2 l @ diag l

   let rec cloop set i j k =
     if i > ((List.length set) − 1) then [ ]
     else if j > i then cloop set (succ i) (j − i − 1) (j − i − 1)
     else if k > j then cloop set i (succ j) (k − j − 1)
     else (List.nth set i, List.nth set j, List.nth set k) :: cloop set i j (succ k)
```

```
let triples set = cloop set 0 0 0

let rec two_and_one' l1 z n =
  if n < 0 then []
  else
  ((fst (List.nth (pairs l1) n)), (snd (List.nth (pairs l1) n)), z) :: two_and_one' l1 z (pred n)

let two_and_one l1 l2 =
  let f z = two_and_one' l1 z ((List.length (pairs l1)) - 1)
  in
  List.flatten ( List.map f l2 )

type gen =
  | G of int | GG of gen × gen

let rec string_of_gen = function
  | G n when n > 0 → string_of_int n
  | G n → string_of_int (abs n) ^ "c"
  | GG (g1, g2) → string_of_gen g1 ^ "_" ^ string_of_gen g2
```

With this we distinguish the flavour.

```
type sff =
  | SL | SN | SU | SD

let string_of_sff = function
  | SL → "sl" | SN → "sn" | SU → "su" | SD → "sd"
```

With this we distinguish the mass eigenstates. At the moment we have to cheat a little bit for the sneutrinos. Because we are dealing with massless neutrinos there is only one sort of sneutrino.

```
type sfm =
  | M1 | M2

let string_of_sfm = function
  | M1 → "1" | M2 → "2"
```

We also introduce special types for the charginos and neutralinos.

```
type char =
  | C1 | C2 | C1c | C2c

type neu =
  | N1 | N2 | N3 | N4 | N5

let int_of_char = function
  | C1 → 1 | C2 → 2 | C1c → -1 | C2c → -2

let string_of_char = function
  | C1 → "1" | C2 → "2" | C1c → "-1" | C2c → "-2"

let conj_char = function
  | C1 → C1c | C2 → C2c | C1c → C1 | C2c → C2

let string_of_neu = function
  | N1 → "1" | N2 → "2" | N3 → "3" | N4 → "4" | N5 → "5"
```

For the Higgs bosons, we follow the conventions of Franke/Fraas.

```
type shiggs =
  | S1 | S2 | S3

type phiggs =
  | P1 | P2

let string_of_shiggs = function
  | S1 → "1" | S2 → "2" | S3 → "3"

let string_of_phiggs = function
  | P1 → "1" | P2 → "2"

type flavor =
  | L of int | N of int
  | U of int | D of int
  | Sup of sfm × int | Sdown of sfm × int
  | Ga | Wp | Wm | Z | Gl
  | Slepton of sfm × int | Sneutrino of int
  | Neutralino of neu | Chargino of char
  | Gluino
  | SHiggs of shiggs | Hp | Hm | PHiggs of phiggs

let string_of_fermion_type = function
  | L _ → "l" | U _ → "u" | D _ → "d" | N _ → "n"
  | _ → failwith "Modellib_NMSSM.NMSSM.string_of_fermion_type:␣invalid␣fermion␣type"

let string_of_fermion_gen = function
  | L g | U g | D g | N g → string_of_int (abs (g))
  | _ → failwith "Modellib_NMSSM.NMSSM.string_of_fermion_gen:␣invalid␣fermion␣type"

type gauge = unit

let gauge_symbol () =
  failwith "Modellib_NMSSM.NMSSM.gauge_symbol:␣internal␣error"
```

At this point we will forget graviton and -ino.

```
let family g = [ L g; N g; Slepton (M1, g);
                 Slepton (M2, g); Sneutrino g;
                 U g; D g; Sup (M1, g); Sup (M2, g);
                 Sdown (M1, g); Sdown (M2, g)]

let external_flavors () =
    [ "1st␣Generation␣matter", ThoList.flatmap family [1; − 1];
      "2nd␣Generation␣matter", ThoList.flatmap family [2; − 2];
      "3rd␣Generation␣matter", ThoList.flatmap family [3; − 3];
      "Gauge␣Bosons", [Ga; Z; Wp; Wm; Gl];
      "Charginos", [Chargino C1; Chargino C2; Chargino C1c; Chargino C2c];
      "Neutralinos", [Neutralino N1; Neutralino N2; Neutralino N3;
                      Neutralino N4; Neutralino N5];
      "Higgs␣Bosons", [SHiggs S1; SHiggs S2; SHiggs S3; Hp; Hm; PHiggs P1; PHiggs P2];
      "Gluino", [Gluino]]

let flavors () = ThoList.flatmap snd (external_flavors ())

let spinor n m =
  if n ≥ 0 ∧ m ≥ 0 then
```

```
        Spinor
      else if
        n ≤ 0 ∧ m ≤ 0 then
        ConjSpinor
      else
        invalid_arg "Modellib_NMSSM.NMSSM.spinor:␣internal␣error"

  let lorentz = function
    | L g → spinor g 0 | N g → spinor g 0
    | U g → spinor g 0 | D g → spinor g 0
    | Chargino c → spinor (int_of_char c) 0
    | Ga | Gl → Vector
    | Wp | Wm | Z → Massive_Vector
    | SHiggs _ | PHiggs _ | Hp | Hm
    | Sup _ | Sdown _ | Slepton _ | Sneutrino _ → Scalar
    | Neutralino _ | Gluino → Majorana

  let color = function
    | U g → Color.SUN (if g > 0 then 3 else -3)
    | Sup (m, g) → Color.SUN (if g > 0 then 3 else -3)
    | D g → Color.SUN (if g > 0 then 3 else -3)
    | Sdown (m, g) → Color.SUN (if g > 0 then 3 else -3)
    | Gl | Gluino → Color.AdjSUN 3
    | _ → Color.Singlet

  let prop_spinor n m =
    if n ≥ 0 ∧ m ≥ 0 then
      Prop_Spinor
    else if
      n ≤ 0 ∧ m ≤ 0 then
      Prop_ConjSpinor
    else
      invalid_arg "Modellib_NMSSM.NMSSM.prop_spinor:␣internal␣error"

  let propagator = function
    | L g → prop_spinor g 0 | N g → prop_spinor g 0
    | U g → prop_spinor g 0 | D g → prop_spinor g 0
    | Chargino c → prop_spinor (int_of_char c) 0
    | Ga | Gl → Prop_Feynman
    | Wp | Wm | Z → Prop_Unitarity
    | SHiggs _ | PHiggs _ → Prop_Scalar
    | Hp | Hm → Prop_Scalar
    | Sup _ | Sdown _ | Slepton _ | Sneutrino _ → Prop_Scalar
    | Gluino → Prop_Majorana
    | Neutralino _ → Prop_Majorana
```

Optionally, ask for the fudge factor treatment for the widths of charged particles. Currently, this only applies to $W^{\pm}$ and top.

```
  let width f =
    if !use_fudged_width then
      match f with
      | Wp | Wm | U 3 | U (−3) → Fudged
```

```
      | _ → !default_width
    else
      !default_width
  let goldstone _ = None
  let conjugate = function
    | L g → L (−g) | N g → N (−g)
    | U g → U (−g) | D g → D (−g)
    | Sup (m, g) → Sup (m, −g)
    | Sdown (m, g) → Sdown (m, −g)
    | Slepton (m, g) → Slepton (m, −g)
    | Sneutrino g → Sneutrino (−g)
    | Gl → Gl | Ga → Ga | Z → Z
    | Wp → Wm | Wm → Wp
    | SHiggs s → SHiggs s
    | PHiggs p → PHiggs p
    | Hp → Hm | Hm → Hp
    | Gluino → Gluino
    | Neutralino n → Neutralino n | Chargino c → Chargino (conj_char c)
let fermion = function
  | L g → if g > 0 then 1 else -1
  | N g → if g > 0 then 1 else -1
  | U g → if g > 0 then 1 else -1
  | D g → if g > 0 then 1 else -1
  | Gl | Ga | Z | Wp | Wm → 0
  | SHiggs _ | Hp | Hm | PHiggs _ → 0
  | Neutralino _ → 2
  | Chargino c → if (int_of_char c) > 0 then 1 else -1
  | Sup _ → 0 | Sdown _ → 0
  | Slepton _ → 0 | Sneutrino _ → 0
  | Gluino → 2
  module Ch = Charges.QQ
  let ( // ) = Algebra.Small_Rational.make
  let generation' = function
    | 1 → [ 1//1; 0//1; 0//1]
    | 2 → [ 0//1; 1//1; 0//1]
    | 3 → [ 0//1; 0//1; 1//1]
    | − 1 → [−1//1; 0//1; 0//1]
    | − 2 → [ 0//1; − 1//1; 0//1]
    | − 3 → [ 0//1; 0//1; − 1//1]
    | n → invalid_arg ("NMSSM.generation':␣" ^ string_of_int n)
  let generation f =
    if Flags.ckm_present then
      []
    else
      match f with
      | L n | N n | U n | D n | Sup (_, n)
      | Sdown (_, n) | Slepton (_, n)
```

```
      | Sneutrino n → generation' n
      | _ → [0//1; 0//1; 0//1]

let charge = function
  | L n → if n > 0 then -1//1 else 1//1
  | Slepton (_, n) → if n > 0 then -1//1 else 1//1
  | N n → 0//1
  | Sneutrino n → 0//1
  | U n → if n > 0 then 2//3 else -2//3
  | Sup (_, n) → if n > 0 then 2//3 else -2//3
  | D n → if n > 0 then -1//3 else 1//3
  | Sdown (_, n) → if n > 0 then -1//3 else 1//3
  | Gl | Ga | Z | Neutralino _ | Gluino → 0//1
  | Wp → 1//1
  | Wm → −1//1
  | SHiggs _ | PHiggs _ → 0//1
  | Hp → 1//1
  | Hm → −1//1
  | Chargino (C1 | C2) → 1//1
  | Chargino (C1c | C2c) → −1//1

let lepton = function
  | L n | N n → if n > 0 then 1//1 else -1//1
  | Slepton (_, n)
  | Sneutrino n → if n > 0 then 1//1 else -1//1
  | _ → 0//1

let baryon = function
  | U n | D n → if n > 0 then 1//1 else -1//1
  | Sup (_, n) | Sdown (_, n) → if n > 0 then 1//1 else -1//1
  | _ → 0//1

let charges f =
  [ charge f; lepton f; baryon f] @ generation f
```

We introduce a Boolean type vc as a pseudonym for Vertex Conjugator to distinguish between vertices containing complex mixing matrices like the CKM–matrix or the sfermion or neutralino/chargino–mixing matrices, which have to become complex conjugated. The true–option stands for the conjugated vertex, the false–option for the unconjugated vertex.

```
type vc = bool

type constant =
  | E | G
  | Mu (*lambda*¡s¿*) | Lambda
  | Q_lepton | Q_up | Q_down | Q_charg
  | G_Z | G_CC | G_CCQ of vc × int×int
  | G_NC_neutrino | G_NC_lepton | G_NC_up | G_NC_down
  | I_Q_W | I_G_ZWW | G_WWWW | G_ZZWW | G_PZWW |
G_PPWW
  | G_SS | I_G_S | Gs
  | G_NZN of neu × neu | G_CZC of char×char
  | G_YUK_FFS of flavor × flavor × shiggs
```

| G_YUK_FFP of $flavor \times flavor \times phiggs$
| G_YUK_LCN of int
| G_YUK_UCD of $int \times int$ | G_YUK_DCU of $int \times int$
| G_NHC of $vc \times neu \times char$
| G_YUK_C of $vc \times flavor \times char \times sff \times sfm$
| G_YUK_Q of $vc \times int \times flavor \times char \times sff \times sfm$
| G_YUK_N of $vc \times flavor \times neu \times sff \times sfm$
| G_YUK_G of $vc \times flavor \times sff \times sfm$
| G_NWC of $neu \times char$ | G_CWN of $char \times neu$
| G_CSC of $char \times char \times shiggs$
| G_CPC of $char \times char \times phiggs$
| G_WSQ of $vc \times int \times int \times sfm \times sfm$
| G_SLSNW of $vc \times int \times sfm$
| G_ZSF of $sff \times int \times sfm \times sfm$
| G_CICIS of $neu \times neu \times shiggs$
| G_CICIP of $neu \times neu \times phiggs$
| G_GH_WPC of $phiggs$ | G_GH_WSC of $shiggs$
| G_GH_ZSP of $shiggs \times phiggs$ | G_GH_WWS of $shiggs$
| G_GH_ZZS of $shiggs$ | G_GH_ZCC
| G_GH_GaCC
| G_GH4_ZZPP of $phiggs \times phiggs$
| G_GH4_ZZSS of $shiggs \times shiggs$
| G_GH4_ZZCC | G_GH4_GaGaCC
| G_GH4_ZGaCC | G_GH4_WWCC
| G_GH4_WWPP of $phiggs \times phiggs$
| G_GH4_WWSS of $shiggs \times shiggs$
| G_GH4_ZWSC of $shiggs$
| G_GH4_GaWSC of $shiggs$
| G_GH4_ZWPC of $phiggs$
| G_GH4_GaWPC of $phiggs$
| G_WWSFSF of $sff \times int \times sfm \times sfm$
| G_WPSLSN of $vc \times int \times sfm$
| G_H3_SCC of $shiggs$
| G_H3_SSS of $shiggs \times shiggs \times shiggs$
| G_H3_SPP of $shiggs \times phiggs \times phiggs$
| G_SFSFS of $shiggs \times sff \times int \times sfm \times sfm$
| G_SFSFP of $phiggs \times sff \times int \times sfm \times sfm$
| G_HSNSL of $vc \times int \times sfm$
| G_HSUSD of $vc \times sfm \times sfm \times int \times int$
| G_WPSUSD of $vc \times sfm \times sfm \times int \times int$
| G_WZSUSD of $vc \times sfm \times sfm \times int \times int$
| G_WZSLSN of $vc \times int \times sfm$ | $G\_GlGlSQSQ$
| G_PPSFSF of sff
| G_ZZSFSF of $sff \times int \times sfm \times sfm$ | G_ZPSFSF of $sff \times int \times sfm \times sfm$
| $G\_GlZSFSF$ of $sff \times int \times sfm \times sfm$ | $G\_GlPSQSQ$
| $G\_GlWSUSD$ of $vc \times sfm \times sfm \times int \times int$
| $G\_GLUGLUA0$ of $phiggs$ | $G\_GLUGLUH0$ of $shiggs$

Two integer counters for the QCD and EW order of the couplings.

type $orders = int \times int$

```
let orders = function
  | _ → (0,0)
```

$$\alpha_{\mathrm{QED}} = \frac{1}{137.0359895} \tag{13.83a}$$

$$\sin^2\theta_w = 0.23124 \tag{13.83b}$$

Here we must perhaps allow for complex input parameters. So split them into their modulus and their phase. At first, we leave them real; the generalization to complex parameters is obvious.

```
let parameters () =
  { input = [];
    derived = [];
    derived_arrays = [] }

module F = Modeltools.Fusions (struct
  type f = flavor
  type c = constant
  let compare = compare
  let conjugate = conjugate
end)
```

For the couplings there are generally two possibilities concerning the sign of the covariant derivative.

$$\mathrm{CD}^{\pm} = \partial_\mu \pm \mathrm{i} g T^a A^a_\mu \tag{13.84}$$

The particle data group defines the signs consistently to be positive. Since the convention for that signs also influence the phase definitions of the gaugino/higgsino fields via the off-diagonal entries in their mass matrices it would be the best to adopt that convention.
** REVISED: Compatible with CD+. FB **

```
let electromagnetic_currents_3 g =
    [ ((L (−g), Ga, L g), FBF (1, Psibar, V, Psi), Q_lepton);
      ((U (−g), Ga, U g), FBF (1, Psibar, V, Psi), Q_up);
      ((D (−g), Ga, D g), FBF (1, Psibar, V, Psi), Q_down)]
```

** REVISED: Compatible with CD+. FB**

```
let electromagnetic_sfermion_currents g m =
    [ ((Ga, Slepton (m, −g), Slepton (m, g)), Vector_Scalar_Scalar 1, Q_lepton);
      ((Ga, Sup (m, −g), Sup (m, g)), Vector_Scalar_Scalar 1, Q_up);
      ((Ga, Sdown (m, −g), Sdown (m, g)), Vector_Scalar_Scalar 1, Q_down)]
```

** REVISED: Compatible with CD+. FB**

```
let electromagnetic_currents_2 c =
  let cc = conj_char c in
  [ ((Chargino cc, Ga, Chargino c), FBF (1, Psibar, V, Psi), Q_charg) ]
```

** REVISED: Compatible with CD+. FB**

```
let neutral_currents g =
  [ ((L (−g), Z, L g), FBF (1, Psibar, VA, Psi), G_NC_lepton);
```

$((N\ (-g),\ Z,\ N\ g),\ \mathit{FBF}\ (1,\ \mathit{Psibar},\ \mathit{VA},\ \mathit{Psi}),\ \mathit{G\_NC\_neutrino});$
$((U\ (-g),\ Z,\ U\ g),\ \mathit{FBF}\ (1,\ \mathit{Psibar},\ \mathit{VA},\ \mathit{Psi}),\ \mathit{G\_NC\_up});$
$((D\ (-g),\ Z,\ D\ g),\ \mathit{FBF}\ (1,\ \mathit{Psibar},\ \mathit{VA},\ \mathit{Psi}),\ \mathit{G\_NC\_down})]$

$$\mathcal{L}_{\mathrm{CC}} = \mp\frac{g}{2\sqrt{2}}\sum_i \bar\psi_i\gamma^\mu(1-\gamma_5)(T^+W^+_\mu + T^-W^-_\mu)\psi_i, \tag{13.85}$$

where the sign corresponds to $\mathrm{CD}_\pm$, respectively.
** REVISED: Compatible with CD+. **
Remark: The definition with the other sign compared to the SM files comes from the fact that $g_{cc} = 1/(2\sqrt{2})$ is used overwhelmingly often in the SUSY Feynman rules, so that JR decided to use a different definiton for *g_cc* in SM and MSSM.
* FB *

```
let charged_currents g =
  [ ((L (−g), Wm, N g), FBF ((−1), Psibar, VL, Psi), G_CC);
    ((N (−g), Wp, L g), FBF ((−1), Psibar, VL, Psi), G_CC) ]
```

The quark with the inverted generation (the antiparticle) is the outgoing one, the other the incoming. The vertex attached to the outgoing up-quark contains the CKM matrix element *not* complex conjugated, while the vertex with the outgoing down-quark has the conjugated CKM matrix element.
** REVISED: Compatible with CD+. FB **

```
let charged_quark_currents g h =
    [ ((D (−g), Wm, U h), FBF ((−1), Psibar, VL, Psi), G_CCQ (true,g, h));
      ((U (−g), Wp, D h), FBF ((−1), Psibar, VL, Psi), G_CCQ (false,h, g))]
```

** REVISED: Compatible with CD+.FB **

```
let charged_chargino_currents n c =
  let cc = conj_char c in
  [ ((Chargino cc, Wp, Neutralino n),
                 FBF (1, Psibar, VLR, Chi), G_CWN (c, n));
    ((Neutralino n, Wm, Chargino c),
                 FBF (1, Chibar, VLR, Psi), G_NWC (n, c)) ]
```

** REVISED: Compatible with CD+. FB**

```
let charged_slepton_currents g m =
  [ ((Wm, Slepton (m, −g), Sneutrino g), Vector_Scalar_Scalar (−1), G_SLSNW
      (true,g, m));
    ((Wp, Slepton (m, g), Sneutrino (−g)), Vector_Scalar_Scalar 1, G_SLSNW
      (false,g, m)) ]
```

** REVISED: Compatible with CD+. FB**

```
let charged_squark_currents' g h m1 m2 =
  [ ((Wm, Sup (m1, g), Sdown (m2, −h)), Vector_Scalar_Scalar (−1), G_WSQ
        (true,g, h, m1, m2));
      ((Wp, Sup (m1, −g), Sdown (m2, h)), Vector_Scalar_Scalar 1, G_WSQ
        (false,g, h, m1, m2)) ]
let charged_squark_currents g h =
List.flatten (Product.list2 (charged_squark_currents' g h) [M1; M2] [M1; M2] )
```

** REVISED: Compatible with CD+. FB **

```
let neutral_sfermion_currents' g m1 m2 =
  [ ((Z, Slepton (m1, -g), Slepton (m2, g)), Vector_Scalar_Scalar (-1),
      G_ZSF (SL, g, m1, m2));
    ((Z, Sup (m1, -g), Sup (m2, g)), Vector_Scalar_Scalar (-1),
      G_ZSF(SU, g, m1, m2));
    ((Z, Sdown (m1, -g), Sdown (m2, g)), Vector_Scalar_Scalar (-1),
      G_ZSF (SD, g, m1, m2))]
let neutral_sfermion_currents g =
  List.flatten (Product.list2 (neutral_sfermion_currents'
                g) [M1; M2] [M1; M2]) @
  [ ((Z, Sneutrino (-g), Sneutrino g), Vector_Scalar_Scalar (-1),
      G_ZSF (SN, g, M1, M1)) ]
```

** REVISED: Compatible with CD+. FB**

```
let neutral_Z (n, m) =
  [ ((Neutralino n, Z, Neutralino m), FBF (1, Chibar, VLR, Chi),
          (G_NZN (n, m))) ]
```

** REVISED: Compatible with CD+. FB**

```
let charged_Z c1 c2 =
  let cc1 = conj_char c1 in
  ((Chargino cc1, Z, Chargino c2), FBF ((-1), Psibar, VA , Psi),
          G_CZC (c1, c2))
```

** REVISED: Compatible with CD+. Remark: This is pure octet. FB**

```
let yukawa_v =
  [ (Gluino, Gl, Gluino), FBF (1, Chibar, V, Chi), Gs]
```

** REVISED: Independent of the sign of CD. **
** REVISED: Felix Braam: Compact version using new COMBOS + FF-Couplings

```
let yukawa_higgs_FFS f s =
    [((conjugate f, SHiggs s, f ), FBF (1, Psibar, S, Psi),
        G_YUK_FFS (conjugate f, f, s))]
let yukawa_higgs_FFP f p =
    [((conjugate f, PHiggs p, f), FBF (1, Psibar, P, Psi),
        G_YUK_FFP (conjugate f ,f , p))]
let yukawa_higgs_NLC g =
  [ ((N (-g), Hp, L g), FBF (1, Psibar, Coupling.SR, Psi), G_YUK_LCN g);
    ((L (-g), Hm, N g), FBF (1, Psibar, Coupling.SL, Psi), G_YUK_LCN g)]

let yukawa_higgs g =
    yukawa_higgs_NLC g @
    List.flatten ( Product.list2 yukawa_higgs_FFS [L g; U g; D g] [S1; S2; S3]) @
    List.flatten ( Product.list2 yukawa_higgs_FFP [L g; U g; D g] [P1; P2])
```

** REVISED: Independent of the sign of CD. FB**

```
let yukawa_higgs_quark (g, h) =
  [ ((U (-g), Hp, D h), FBF (1, Psibar, SLR, Psi), G_YUK_UCD (g, h));
```

$((D\ (-h),\ Hm,\ U\ g),\ FBF\ (1,\ Psibar,\ SLR,\ Psi),\ G\_YUK\_DCU\ (g,\ h))\ ]$

** REVISED: Compatible with CD+. **
** REVISED: Felix Braam: Compact version using new COMBOS

```
let yukawa_shiggs_2 c1 c2 s =
  let cc1 = conj_char c1 in
   ((Chargino cc1, SHiggs s, Chargino c2), FBF (1, Psibar, SLR, Psi),
       G_CSC (c1, c2, s))

let yukawa_phiggs_2 c1 c2 p =
  let cc1 = conj_char c1 in
  ((Chargino cc1, PHiggs p, Chargino c2), FBF (1, Psibar, SLR, Psi),
      G_CPC (c1, c2, p))

let yukawa_higgs_2 =
  Product.list3 yukawa_shiggs_2 [C1; C2] [C1; C2] [S1; S2; S3] @
  Product.list3 yukawa_phiggs_2 [C1; C2] [C1; C2] [P1; P2]
```

** REVISED: Compatible with CD+.FB **

```
let higgs_charg_neutr n c =
  let cc = conj_char c in
  [ ((Neutralino n, Hm, Chargino c), FBF (-1, Chibar, SLR, Psi),
              G_NHC (false,n, c));
    ((Chargino cc, Hp, Neutralino n), FBF (-1, Psibar, SLR, Chi),
              G_NHC (true,n, c)) ]
```

** REVISED: Compatible with CD+. **
** REVISED: Felix Braam: Compact version using new COMBOS

```
let shiggs_neutr (n, m, s) =
   ((Neutralino n, SHiggs s, Neutralino m), FBF (1, Chibar, SLR, Chi),
    G_CICIS (n, m, s))
let phiggs_neutr (n, m, p) =
   ((Neutralino n, PHiggs p, Neutralino m), FBF (1, Chibar, SLR, Chi),
    G_CICIP (n, m, p))

let higgs_neutr =
  List.map shiggs_neutr (two_and_one [N1; N2; N3; N4; N5] [S1; S2; S3]) @
  List.map phiggs_neutr (two_and_one [N1; N2; N3; N4; N5] [P1; P2])
```

** REVISED: Compatible with CD+. FB**

```
let yukawa_n_2 n m g =
  [ ((Neutralino n, Slepton (m, -g), L g), FBF (1, Chibar, SLR, Psi),
        G_YUK_N (true,L g, n, SL, m));
    ((L (-g), Slepton (m, g), Neutralino n), FBF (1, Psibar, SLR, Chi),
        G_YUK_N (false,L g, n, SL, m));
    ((Neutralino n, Sup (m, -g), U g), FBF (1, Chibar, SLR, Psi),
        G_YUK_N (true,U g, n, SU, m));
    ((U (-g), Sup (m, g), Neutralino n), FBF (1, Psibar, SLR, Chi),
        G_YUK_N (false,U g, n, SU, m));
    ((Neutralino n, Sdown (m, -g), D g), FBF (1, Chibar, SLR, Psi),
        G_YUK_N (true,D g, n, SD, m));
    ((D (-g), Sdown (m, g), Neutralino n), FBF (1, Psibar, SLR, Chi),
```

```
            G_YUK_N (false,D g, n, SD, m)) ]
  let yukawa_n_3 n g =
      [ ((Neutralino n, Sneutrino (-g), N g), FBF (1, Chibar, SLR, Psi),
            G_YUK_N (true,N g, n, SN, M1));
        ((N (-g), Sneutrino g, Neutralino n), FBF (1, Psibar, SLR, Chi),
            G_YUK_N (false,N g, n, SN, M1)) ]

  let yukawa_n_5 g m =
      [ ((U (-g), Sup (m, g), Gluino), FBF (1, Psibar, SLR, Chi),
            G_YUK_G (false,U g, SU, m));
        ((D (-g), Sdown (m, g), Gluino), FBF (1, Psibar, SLR, Chi),
            G_YUK_G (false,D g, SD, m));
        ((Gluino, Sup (m, -g), U g), FBF (1, Chibar, SLR, Psi),
            G_YUK_G (true,U g, SU, m));
        ((Gluino, Sdown (m, -g), D g), FBF (1, Chibar, SLR, Psi),
            G_YUK_G (true,D g, SD, m))]
  let yukawa_n =
    List.flatten (Product.list3 yukawa_n_2 [N1; N2; N3; N4; N5] [M1; M2] [1; 2; 3]) @
    List.flatten (Product.list2 yukawa_n_3 [N1; N2; N3; N4; N5] [1; 2; 3]) @
    List.flatten (Product.list2 yukawa_n_5 [1; 2; 3] [M1; M2])
```

** REVISED: Compatible with CD+.FB **

```
  let yukawa_c_2 c g =
      let cc = conj_char c in
      [ ((L (-g), Sneutrino g, Chargino cc), BBB (1, Psibar, SLR,
            Psibar), G_YUK_C (true,L g, c, SN, M1));
        ((Chargino c, Sneutrino (-g), L g), PBP (1, Psi, SLR, Psi),
            G_YUK_C (false,L g, c, SN, M1)) ]
  let yukawa_c_3 c m g =
      let cc = conj_char c in
      [ ((N (-g), Slepton (m, g), Chargino c), FBF (1, Psibar, SLR,
            Psi), G_YUK_C (true,N g, c, SL, m));
        ((Chargino cc, Slepton (m, -g), N g), FBF (1, Psibar, SLR,
            Psi), G_YUK_C (false,N g, c, SL, m)) ]
  let yukawa_c c =
    ThoList.flatmap (yukawa_c_2 c) [1; 2; 3] @
    List.flatten (Product.list2 (yukawa_c_3 c) [M1; M2] [1; 2; 3])
```

** REVISED: Compatible with CD+. FB**

```
  let yukawa_cq' c (g, h) m =
      let cc = conj_char c in
        [ ((Chargino c, Sup (m, -g), D h), PBP (1, Psi, SLR, Psi),
            G_YUK_Q (false,g, D h, c, SU, m));
          ((D (-h), Sup (m, g), Chargino cc), BBB (1, Psibar, SLR, Psibar),
            G_YUK_Q (true,g, D h, c, SU, m));
          ((Chargino cc, Sdown (m, -g), U h), FBF (1, Psibar, SLR, Psi),
            G_YUK_Q (true,g, U h, c, SD, m));
          ((U (-h), Sdown (m, g), Chargino c), FBF (1, Psibar, SLR, Psi),
            G_YUK_Q (false,g, U h, c, SD, m)) ]
  let yukawa_cq c =
   if Flags.ckm_present then
```

```
      List.flatten (Product.list2 (yukawa_cq' c) [(1,1);(1,2);(2,1);(2,2);(1,3);(2,3);(3,3);(3,2);(3,1)] [M
    else
      List.flatten (Product.list2 (yukawa_cq' c) [(1,1);(2,2);(3,3)] [M1;M2])
```

** REVISED: Compatible with CD+. Remark: Singlet and octet gluon exchange. The coupling is divided by sqrt(2) to account for the correct normalization of the Lie algebra generators. **FB

```
  let col_currents g =
    [ ((D (-g), Gl, D g), FBF ((-1), Psibar, V, Psi), Gs);
      ((U (-g), Gl, U g), FBF ((-1), Psibar, V, Psi), Gs)]
```

** REVISED: Compatible with CD+. Remark: Singlet and octet gluon exchange. The coupling is divided by sqrt(2) to account for the correct normalization of the Lie algebra generators. **FB

```
  let chg = function
    | M1 -> M2 | M2 -> M1

  let col_sfermion_currents g m =
    [ ((Gl, Sup (m,-g), Sup (m,g)), Vector_Scalar_Scalar (-1), Gs);
      ((Gl, Sdown (m,-g), Sdown (m,g)), Vector_Scalar_Scalar (-1), Gs)]
```

** REVISED: Compatible with CD+. **FB

```
  let triple_gauge =
    [ ((Ga, Wm, Wp), Gauge_Gauge_Gauge 1, I_Q_W);
      ((Z, Wm, Wp), Gauge_Gauge_Gauge 1, I_G_ZWW);
      ((Gl, Gl, Gl), Gauge_Gauge_Gauge 1, I_G_S)]
```

** REVISED: Independent of the sign of CD. **FB

```
  let gauge4 = Vector4 [(2, C_13_42); (-1, C_12_34); (-1, C_14_23)]
  let minus_gauge4 = Vector4 [(-2, C_13_42); (1, C_12_34); (1, C_14_23)]
  let quartic_gauge =
    [ (Wm, Wp, Wm, Wp), gauge4, G_WWWW;
      (Wm, Z, Wp, Z), minus_gauge4, G_ZZWW;
      (Wm, Z, Wp, Ga), minus_gauge4, G_PZWW;
      (Wm, Ga, Wp, Ga), minus_gauge4, G_PPWW;
      (Gl, Gl, Gl, Gl), gauge4, G_SS]
```

The *Scalar_Vector_Vector* couplings do not depend on the choice of the sign of the covariant derivative since they are quadratic in the gauge couplings.
* Effective Higgs-Gluon-Gluon coupling. *

```
    let gauge_higgs_GlGlS s =
      ((SHiggs s, Gl, Gl), Dim5_Scalar_Gauge2 1, G_GLUGLUH0 s)
    let gauge_higgs_GlGlP p =
      ((PHiggs p, Gl, Gl), Dim5_Scalar_Gauge2_Skew 1, G_GLUGLUA0 p)
```

** REVISED: Compatible with CD+. FB**
** Revision: 2005-03-10: first two vertices corrected. **
** REVISED: Compact version using new COMBOS
** REVISED: Couplings adjusted to FF-convention

```
    let gauge_higgs_WPC p =
    [ ((Wm, Hp, PHiggs p), Vector_Scalar_Scalar 1, G_GH_WPC p);
```

```
        ((Wp, Hm, PHiggs p), Vector_Scalar_Scalar 1, G_GH_WPC p)]
    let gauge_higgs_WSC s =
      [((Wm, Hp, SHiggs s), Vector_Scalar_Scalar 1, G_GH_WSC s);
        ((Wp, Hm, SHiggs s), Vector_Scalar_Scalar (-1), G_GH_WSC s)]
    let gauge_higgs_ZSP s p =
        [((Z, SHiggs s, PHiggs p), Vector_Scalar_Scalar 1, G_GH_ZSP (s, p))]
    let gauge_higgs_WWS s =
        ((SHiggs s, Wp, Wm), Scalar_Vector_Vector 1, G_GH_WWS s)
    let gauge_higgs_ZZS s =
        ((SHiggs s, Z, Z), Scalar_Vector_Vector 1, G_GH_ZZS s)
    let gauge_higgs_ZCC =
        ((Z, Hp, Hm), Vector_Scalar_Scalar 1, G_GH_ZCC )
    let gauge_higgs_GaCC =
        ((Ga, Hp, Hm), Vector_Scalar_Scalar 1, G_GH_GaCC )

    let gauge_higgs =
      ThoList.flatmap gauge_higgs_WPC [P1; P2] @
      ThoList.flatmap gauge_higgs_WSC [S1; S2; S3] @
      List.flatten (Product.list2 gauge_higgs_ZSP [S1; S2; S3] [P1; P2]) @
      List.map gauge_higgs_WWS [S1; S2; S3] @
      List.map gauge_higgs_ZZS [S1; S2; S3] @
      [gauge_higgs_ZCC] @ [gauge_higgs_GaCC] @
     (if Flags.higgs_triangle then
        List.map gauge_higgs_GlGlS [S1; S2; S3] @
        List.map gauge_higgs_GlGlP [P1; P2]
      else
        [])
```

** REVISED: Compact version using new COMBOS
** REVISED: Couplings adjusted to FF-convention

```
    let gauge_higgs4_ZZPP (p1, p2) =
      ((PHiggs p1, PHiggs p2, Z, Z), Scalar2_Vector2 1, G_GH4_ZZPP (p1, p2))

    let gauge_higgs4_ZZSS (s1, s2) =
        ((SHiggs s1, SHiggs s2 , Z, Z), Scalar2_Vector2 1, G_GH4_ZZSS (s1, s2))

    let gauge_higgs4_ZZCC =
        ((Hp, Hm, Z, Z), Scalar2_Vector2 1, G_GH4_ZZCC)

    let gauge_higgs4_GaGaCC =
        ((Hp, Hm, Ga, Ga), Scalar2_Vector2 1, G_GH4_GaGaCC)

    let gauge_higgs4_ZGaCC =
        ((Hp, Hm, Ga, Z), Scalar2_Vector2 1, G_GH4_ZGaCC )

    let gauge_higgs4_WWCC =
        ((Hp, Hm, Wp, Wm), Scalar2_Vector2 1, G_GH4_WWCC )

    let gauge_higgs4_WWPP (p1, p2) =
        ((PHiggs p1, PHiggs p2, Wp, Wm), Scalar2_Vector2 1, G_GH4_WWPP (p1, p2))

    let gauge_higgs4_WWSS (s1, s2) =
        ((SHiggs s1, SHiggs s2, Wp, Wm), Scalar2_Vector2 1, G_GH4_WWSS (s1, s2))
```

```
let gauge_higgs4_ZWSC s =
  [ ((Hp, SHiggs s, Wm, Z), Scalar2_Vector2 1, G_GH4_ZWSC s);
    ((Hm, SHiggs s, Wp, Z), Scalar2_Vector2 1, G_GH4_ZWSC s)]

let gauge_higgs4_GaWSC s =
  [ ((Hp, SHiggs s, Wm, Ga), Scalar2_Vector2 1, G_GH4_GaWSC s);
    ((Hm, SHiggs s, Wp, Ga), Scalar2_Vector2 1, G_GH4_GaWSC s) ]

let gauge_higgs4_ZWPC p =
  [ ((Hp, PHiggs p, Wm, Z), Scalar2_Vector2 1, G_GH4_ZWPC p);
    ((Hm, PHiggs p, Wp, Z), Scalar2_Vector2 (-1), G_GH4_ZWPC p)]

let gauge_higgs4_GaWPC p =
  [ ((Hp, PHiggs p, Wm, Ga), Scalar2_Vector2 1, G_GH4_GaWPC p);
    ((Hm, PHiggs p, Wp, Ga), Scalar2_Vector2 (-1), G_GH4_GaWPC p) ]

let gauge_higgs4 =
  List.map gauge_higgs4_ZZPP (pairs [P1; P2]) @
  List.map gauge_higgs4_ZZSS (pairs [S1; S2; S3]) @
  [gauge_higgs4_ZZCC] @ [gauge_higgs4_GaGaCC] @
  [gauge_higgs4_ZGaCC] @ [gauge_higgs4_WWCC] @
  List.map gauge_higgs4_WWPP (pairs [P1; P2]) @
  List.map gauge_higgs4_WWSS (pairs [S1; S2; S3]) @
  ThoList.flatmap gauge_higgs4_ZWSC [S1; S2; S3] @
  ThoList.flatmap gauge_higgs4_GaWSC [S1; S2; S3] @
  ThoList.flatmap gauge_higgs4_ZWPC [P1; P2] @
  ThoList.flatmap gauge_higgs4_GaWPC [P1; P2]
```

FB

```
let gauge_sfermion4' g m1 m2 =
  [ ((Wp, Wm, Slepton (m1, g), Slepton (m2, -g)), Scalar2_Vector2 1,
      G_WWSFSF (SL, g, m1, m2));
    ((Z, Ga, Slepton (m1, g), Slepton (m2, -g)), Scalar2_Vector2 1,
      G_ZPSFSF (SL, g, m1, m2));
    ((Z, Z, Slepton (m1, g), Slepton (m2, -g)), Scalar2_Vector2 1,
      G_ZZSFSF(SL, g, m1, m2));
    ((Wp, Wm, Sup (m1, g), Sup (m2, -g)), Scalar2_Vector2 1, G_WWSFSF
      (SU, g, m1, m2));
    ((Wp, Wm, Sdown (m1, g), Sdown (m2, -g)), Scalar2_Vector2 1,
      G_WWSFSF(SD, g, m1, m2));
    ((Z, Z, Sup (m1, g), Sup (m2, -g)), Scalar2_Vector2 1, G_ZZSFSF
      (SU, g, m1, m2));
    ((Z, Z, Sdown (m1, g), Sdown (m2, -g)), Scalar2_Vector2 1, G_ZZSFSF
      (SD, g, m1, m2));
    ((Z, Ga, Sup (m1, g), Sup (m2, -g)), Scalar2_Vector2 1, G_ZPSFSF
      (SU, g, m1, m2));
    ((Z, Ga, Sdown (m1, g), Sdown (m2, -g)), Scalar2_Vector2 1, G_ZPSFSF
      (SD, g, m1, m2)) ]

let gauge_sfermion4'' g m =
  [ ((Wp, Ga, Slepton (m, g), Sneutrino (-g)), Scalar2_Vector2 1,
      G_WPSLSN (false,g, m));
    ((Wm, Ga, Slepton (m, -g), Sneutrino g), Scalar2_Vector2 1,
```

```
        G_WPSLSN (true,g, m));
      ((Wp, Z, Slepton (m, g), Sneutrino (−g)), Scalar2_Vector2 1,
        G_WZSLSN(false,g, m));
      ((Wm, Z, Slepton (m, −g), Sneutrino g), Scalar2_Vector2 1,
        G_WZSLSN (true,g, m));
      ((Ga, Ga, Slepton (m, g), Slepton (m, −g)), Scalar2_Vector2 1,
        G_PPSFSF SL);
      ((Ga, Ga, Sup (m, g), Sup (m, −g)), Scalar2_Vector2 1, G_PPSFSF SU);
      ((Ga, Ga, Sdown (m, g), Sdown (m, −g)), Scalar2_Vector2 1, G_PPSFSF SD)]

  let gauge_sfermion4 g =
    List.flatten (Product.list2 (gauge_sfermion4' g) [M1; M2] [M1; M2]) @
    ThoList.flatmap (gauge_sfermion4'' g) [M1; M2] @
    [ ((Wp, Wm, Sneutrino g, Sneutrino (−g)), Scalar2_Vector2 1, G_WWSFSF
          (SN, g, M1, M1));
      ((Z, Z, Sneutrino g, Sneutrino (−g)), Scalar2_Vector2 1, G_ZZSFSF
          (SN, g, M1, M1)) ]
```

** Added by Felix Braam. **

```
  let gauge_squark4'' g h m1 m2 =
    [ ((Wp, Ga, Sup (m1, −g), Sdown (m2, h)), Scalar2_Vector2 1, G_WPSUSD
          (false,m1, m2, g, h));
      ((Wm, Ga, Sup (m1, g), Sdown (m2, −h)), Scalar2_Vector2 1, G_WPSUSD
          (true,m1, m2, g, h));
      ((Wp, Z, Sup (m1, −g), Sdown (m2, h)), Scalar2_Vector2 1, G_WZSUSD
          (false,m1, m2, g, h));
      ((Wm, Z, Sup (m1, g), Sdown (m2, −h)), Scalar2_Vector2 1, G_WZSUSD
          (true,m1, m2, g, h)) ]
  let gauge_squark4' g h = List.flatten (Product.list2 (gauge_squark4'' g h)
                                           [M1; M2] [M1; M2])
  let gauge_squark4 =
    if Flags.ckm_present then
      List.flatten (Product.list2 gauge_squark4' [1; 2; 3] [1; 2; 3])
    else
      ThoList.flatmap (fun g → gauge_squark4' g g) [1; 2; 3]
```

*******************************FB********************

```
  let gluon_w_squark'' g h m1 m2 =
    [ ((Gl, Wp, Sup (m1, −g), Sdown (m2, h)),
          Scalar2_Vector2 1, G_GlWSUSD (false,m1, m2, g, h));
      ((Gl, Wm, Sup (m1, g), Sdown (m2, −h)),
          Scalar2_Vector2 1, G_GlWSUSD (true,m1, m2, g, h)) ]
  let gluon_w_squark' g h =
    List.flatten (Product.list2 (gluon_w_squark'' g h) [M1; M2] [M1; M2])
  let gluon_w_squark =
    if Flags.ckm_present then
      List.flatten (Product.list2 gluon_w_squark' [1; 2; 3] [1; 2; 3])
    else
      ThoList.flatmap (fun g → gluon_w_squark' g g) [1; 2; 3]
```

********************************FB*******************

```
  let gluon_gauge_squark' g m1 m2 =
    [ ((Gl, Z, Sup (m1, g), Sup (m2, -g)),
         Scalar2_Vector2 2, G_GlZSFSF (SU, g, m1, m2));
      ((Gl, Z, Sdown (m1, g), Sdown (m2, -g)),
         Scalar2_Vector2 2, G_GlZSFSF (SD, g, m1, m2)) ]
  let gluon_gauge_squark'' g m =
    [ ((Gl, Ga, Sup (m, g), Sup (m, -g)), Scalar2_Vector2 2, G_GlPSQSQ);
      ((Gl, Ga, Sdown (m, g), Sdown (m, -g)), Scalar2_Vector2 (-1), G_GlPSQSQ) ]

  let gluon_gauge_squark g =
    List.flatten (Product.list2 (gluon_gauge_squark' g) [M1; M2] [M1; M2]) @
    ThoList.flatmap (gluon_gauge_squark'' g) [M1; M2]
(*********************************FB*****************)

  let gluon2_squark2' g m =
    [ ((Gl, Gl, Sup (m, g), Sup (m, -g)), Scalar2_Vector2 2, G_GlGlSQSQ);
      ((Gl, Gl, Sdown (m, g), Sdown (m, -g)), Scalar2_Vector2 2, G_GlGlSQSQ) ]
  let gluon2_squark2 g =
    ThoList.flatmap (gluon2_squark2' g) [M1; M2]
```

** REVISED: Independent of the sign of CD. *FB*
** REVISED: Compact version using new COMBOS
** REVISED: Couplings adjusted to FF-convention

```
  let higgs_SCC s =
      ((Hp, Hm, SHiggs s), Scalar_Scalar_Scalar 1, G_H3_SCC s )
  let higgs_SSS (s1, s2, s3) =
      ((SHiggs s1, SHiggs s2, SHiggs s3), Scalar_Scalar_Scalar 1,
      G_H3_SSS (s1, s2, s3))
  let higgs_SPP (p1, p2, s) =
      ((SHiggs s, PHiggs p1, PHiggs p2), Scalar_Scalar_Scalar 1,
      G_H3_SPP (s, p1, p2))

  let higgs =
      List.map higgs_SCC [S1; S2; S3]@
      List.map higgs_SSS (triples [S1; S2; S3])@
      List.map higgs_SPP (two_and_one [P1; P2] [S1; S2; S3])

  let higgs4 = []
```

(* The vertices of the type Higgs - Sfermion - Sfermion are independent of the choice of the CD sign since they are quadratic in the gauge coupling. *)

** REVISED: Independent of the sign of CD. **

```
  let higgs_sneutrino' s g =
      ((SHiggs s, Sneutrino g, Sneutrino (-g)), Scalar_Scalar_Scalar 1,
                      G_SFSFS (s, SN, g, M1, M1))
    let higgs_sneutrino'' g m =
      [((Hp, Sneutrino (-g), Slepton (m, g)), Scalar_Scalar_Scalar 1,
           G_HSNSL (false, g, m));
      ((Hm, Sneutrino g, Slepton (m, -g)), Scalar_Scalar_Scalar 1,
           G_HSNSL (true, g, m))]
    let higgs_sneutrino =
      Product.list2 higgs_sneutrino' [S1; S2; S3] [1; 2; 3] @
```

```
    List.flatten ( Product.list2 higgs_sneutrino'' [1; 2; 3] [M1; M2] )
```

Under the assumption that there is no mixing between the left- and right-handed sfermions for the first two generations there is only a coupling of the form Higgs - sfermion1 - sfermion2 for the third generation. All the others are suppressed by m_f/M_W.
** REVISED: Independent of the sign of CD. **

```
    let higgs_sfermion_S s g m1 m2 =
      [ ((SHiggs s, Slepton (m1, g), Slepton (m2, -g)), Scalar_Scalar_Scalar 1,
            G_SFSFS (s, SL, g, m1, m2));
        ((SHiggs s, Sup (m1, g), Sup (m2, -g)), Scalar_Scalar_Scalar 1,
            G_SFSFS (s, SU, g, m1, m2));
        ((SHiggs s, Sdown (m1, g), Sdown (m2, -g)), Scalar_Scalar_Scalar 1,
            G_SFSFS (s, SD, g, m1, m2))]

  let higgs_sfermion' g m1 m2 =
      (higgs_sfermion_S S1 g m1 m2) @ (higgs_sfermion_S S2 g m1 m2) @ (higgs_sfermion_S S3 g m1

  let higgs_sfermion_P p g m1 m2 =
    [ ((PHiggs p, Slepton (m1, g), Slepton (m2, -g)), Scalar_Scalar_Scalar 1,
          G_SFSFP (p, SL, g, m1, m2));
      ((PHiggs p, Sup (m1, g), Sup (m2, -g)), Scalar_Scalar_Scalar 1,
          G_SFSFP (p, SU, g, m1, m2));
      ((PHiggs p, Sdown (m1, g), Sdown (m2, -g)), Scalar_Scalar_Scalar 1,
          G_SFSFP (p, SD, g, m1, m2)) ]

  let higgs_sfermion'' g m1 m2 =
      (higgs_sfermion_P P1 g m1 m2) @ (higgs_sfermion_P P2 g m1 m2)
  let higgs_sfermion = List.flatten (Product.list3 higgs_sfermion' [1; 2; 3] [M1; M2] [M1; M2]) @
    List.flatten (Product.list3 higgs_sfermion'' [1; 2; 3] [M1; M2] [M1; M2])
```

** REVISED: Independent of the sign of CD. **

```
  let higgs_squark' g h m1 m2 =
    [ ((Hp, Sup (m1, -g), Sdown (m2, h)), Scalar_Scalar_Scalar 1,
          G_HSUSD (false, m1, m2, g, h));
      ((Hm, Sup (m1, g), Sdown (m2, -h)), Scalar_Scalar_Scalar 1,
          G_HSUSD (true, m1, m2, g, h)) ]
  let higgs_squark_a g h = higgs_squark' g h M1 M1
  let higgs_squark_b (g, h) = List.flatten (Product.list2 (higgs_squark' g h)
                                  [M1; M2] [M1; M2])
  let higgs_squark =
    if Flags.ckm_present then
      List.flatten (Product.list2 higgs_squark_a [1; 2] [1; 2]) @
      ThoList.flatmap higgs_squark_b [(1, 3); (2, 3); (3, 3); (3, 1); (3, 2)]
    else
      higgs_squark_a 1 1 @ higgs_squark_a 2 2 @ higgs_squark_b (3, 3)

  let vertices3 =
    (ThoList.flatmap electromagnetic_currents_3 [1; 2; 3] @
     ThoList.flatmap electromagnetic_currents_2 [C1; C2] @
     List.flatten (Product.list2 electromagnetic_sfermion_currents [1; 2; 3]
                [M1; M2]) @
```

```
      ThoList.flatmap neutral_currents [1; 2; 3] @
      ThoList.flatmap neutral_sfermion_currents [1; 2; 3] @
      ThoList.flatmap charged_currents [1; 2; 3] @
      List.flatten (Product.list2 charged_slepton_currents [1; 2; 3]
                          [M1; M2]) @
      (if Flags.ckm_present then
        List.flatten (Product.list2 charged_quark_currents [1; 2; 3]
                           [1; 2; 3]) @
        List.flatten (Product.list2 charged_squark_currents [1; 2; 3]
                           [1; 2; 3]) @
        ThoList.flatmap yukawa_higgs_quark [(1, 3); (2, 3); (3, 3); (3, 1); (3, 2)]
      else
        charged_quark_currents 1 1 @
        charged_quark_currents 2 2 @
        charged_quark_currents 3 3 @
        charged_squark_currents 1 1 @
        charged_squark_currents 2 2 @
        charged_squark_currents 3 3 @
        ThoList.flatmap yukawa_higgs_quark [(3, 3)]) @
      yukawa_higgs 3 @ yukawa_n @
      ThoList.flatmap yukawa_c [C1; C2] @
      ThoList.flatmap yukawa_cq [C1; C2] @
      List.flatten (Product.list2 charged_chargino_currents [N1; N2; N3; N4; N5]
                          [C1; C2]) @ triple_gauge @
      ThoList.flatmap neutral_Z (pairs [N1; N2; N3; N4; N5]) @
      Product.list2 charged_Z [C1; C2] [C1; C2] @
      gauge_higgs @ higgs @ yukawa_higgs_2 @
      List.flatten (Product.list2 higgs_charg_neutr [N1; N2; N3; N4; N5] [C1; C2]) @
      higgs_neutr @ higgs_sneutrino @ higgs_sfermion @
      higgs_squark @ yukawa_v @
      ThoList.flatmap col_currents [1; 2; 3] @
      List.flatten (Product.list2 col_sfermion_currents [1; 2; 3] [M1; M2]))

  let vertices4 =
    (quartic_gauge @ higgs4 @ gauge_higgs4 @
     ThoList.flatmap gauge_sfermion4 [1; 2; 3] @
     gauge_squark4 @ gluon_w_squark @
     ThoList.flatmap gluon2_squark2 [1; 2; 3] @
     ThoList.flatmap gluon_gauge_squark [1; 2; 3])

  let vertices () = (vertices3, vertices4, [])

  let table = F.of_vertices (vertices ())
  let fuse2 = F.fuse2 table
  let fuse3 = F.fuse3 table
  let fuse = F.fuse table
  let max_degree () = 4
```

SLHA2-Nomenclature for neutral Higgses

```
  let flavor_of_string s =
    match s with
      | "e-" → L 1 | "e+" → L (−1)
```

```
| "mu-" → L 2 | "mu+" → L (−2)
| "tau-" → L 3 | "tau+" → L (−3)
| "nue" → N 1 | "nuebar" → N (−1)
| "numu" → N 2 | "numubar" → N (−2)
| "nutau" → N 3 | "nutaubar" → N (−3)
| "se1-" → Slepton (M1, 1) | "se1+" → Slepton (M1, −1)
| "smu1-" → Slepton (M1, 2) | "smu1+" → Slepton (M1, −2)
| "stau1-" → Slepton (M1, 3) | "stau1+" → Slepton (M1, −3)
| "se2-" → Slepton (M2, 1) | "se2+" → Slepton (M2, −1)
| "smu2-" → Slepton (M2, 2) | "smu2+" → Slepton (M2, −2)
| "stau2-" → Slepton (M2, 3) | "stau2+" → Slepton (M2, −3)
| "snue" → Sneutrino 1 | "snue*" → Sneutrino (−1)
| "snumu" → Sneutrino 2 | "snumu*" → Sneutrino (−2)
| "snutau" → Sneutrino 3 | "snutau*" → Sneutrino (−3)
| "u" → U 1 | "ubar" → U (−1)
| "c" → U 2 | "cbar" → U (−2)
| "t" → U 3 | "tbar" → U (−3)
| "d" → D 1 | "dbar" → D (−1)
| "s" → D 2 | "sbar" → D (−2)
| "b" → D 3 | "bbar" → D (−3)
| "A" → Ga | "Z" | "Z0" → Z
| "W+" → Wp | "W-" → Wm
| "gl" | "g" → Gl
| "h01" → SHiggs S1 | "h02" → SHiggs S2 | "h03" →
SHiggs S3
| "A01" → PHiggs P1 | "A02" → PHiggs P2
| "H+" → Hp | "H-" → Hm
| "su1" → Sup (M1, 1) | "su1c" → Sup (M1, −1)
| "sc1" → Sup (M1, 2) | "sc1c" → Sup (M1, −2)
| "st1" → Sup (M1, 3) | "st1c" → Sup (M1, −3)
| "su2" → Sup (M2, 1) | "su2c" → Sup (M2, −1)
| "sc2" → Sup (M2, 2) | "sc2c" → Sup (M2, −2)
| "st2" → Sup (M2, 3) | "st2c" → Sup (M2, −3)
| "sgl" | "sg" → Gluino
| "sd1" → Sdown (M1, 1) | "sd1c" → Sdown (M1, −1)
| "ss1" → Sdown (M1, 2) | "ss1c" → Sdown (M1, −2)
| "sb1" → Sdown (M1, 3) | "sb1c" → Sdown (M1, −3)
| "sd2" → Sdown (M2, 1) | "sd2c" → Sdown (M2, −1)
| "ss2" → Sdown (M2, 2) | "ss2c" → Sdown (M2, −2)
| "sb2" → Sdown (M2, 3) | "sb2c" → Sdown (M2, −3)
| "neu1" → Neutralino N1 | "neu2" → Neutralino N2
| "neu3" → Neutralino N3 | "neu4" → Neutralino N4
| "neu5" → Neutralino N5
| "ch1+" → Chargino C1 | "ch2+" → Chargino C2
| "ch1-" → Chargino C1c | "ch2-" → Chargino C2c
| s → invalid_arg ("Fatal␣error:␣%s␣Modellib_NMSSM.NMSSM.flavor_of_string:" ^ s)

let flavor_to_string = function
  | L 1 → "e-" | L (−1) → "e+"
  | L 2 → "mu-" | L (−2) → "mu+"
```

```
| L 3 → "tau-" | L (−3) → "tau+"
| N 1 → "nue" | N (−1) → "nuebar"
| N 2 → "numu" | N (−2) → "numubar"
| N 3 → "nutau" | N (−3) → "nutaubar"
| U 1 → "u" | U (−1) → "ubar"
| U 2 → "c" | U (−2) → "cbar"
| U 3 → "t" | U (−3) → "tbar"
| U _ → invalid_arg
    "Modellib_NMSSM.NMSSM.flavor_to_string:␣invalid␣up␣type␣quark"
| D 1 → "d" | D (−1) → "dbar"
| D 2 → "s" | D (−2) → "sbar"
| D 3 → "b" | D (−3) → "bbar"
| D _ → invalid_arg
    "Modellib_NMSSM.NMSSM.flavor_to_string:␣invalid␣down␣type␣quark"
| Gl → "gl" | Gluino → "sgl"
| Ga → "A" | Z → "Z"
| Wp → "W+" | Wm → "W-"
| SHiggs S1 → "h01" | SHiggs S2 → "h02" | SHiggs S3 → "h03"
| PHiggs P1 → "A01" | PHiggs P2 → "A02"
| Hp → "H+" | Hm → "H-"
| Slepton (M1, 1) → "se1-" | Slepton (M1, −1) → "se1+"
| Slepton (M1, 2) → "smu1-" | Slepton (M1, −2) → "smu1+"
| Slepton (M1, 3) → "stau1-" | Slepton (M1, −3) → "stau1+"
| Slepton (M2, 1) → "se2-" | Slepton (M2, −1) → "se2+"
| Slepton (M2, 2) → "smu2-" | Slepton (M2, −2) → "smu2+"
| Slepton (M2, 3) → "stau2-" | Slepton (M2, −3) → "stau2+"
| Sneutrino 1 → "snue" | Sneutrino (−1) → "snue*"
| Sneutrino 2 → "snumu" | Sneutrino (−2) → "snumu*"
| Sneutrino 3 → "snutau" | Sneutrino (−3) → "snutau*"
| Sup (M1, 1) → "su1" | Sup (M1, −1) → "su1c"
| Sup (M1, 2) → "sc1" | Sup (M1, −2) → "sc1c"
| Sup (M1, 3) → "st1" | Sup (M1, −3) → "st1c"
| Sup (M2, 1) → "su2" | Sup (M2, −1) → "su2c"
| Sup (M2, 2) → "sc2" | Sup (M2, −2) → "sc2c"
| Sup (M2, 3) → "st2" | Sup (M2, −3) → "st2c"
| Sdown (M1, 1) → "sd1" | Sdown (M1, −1) → "sd1c"
| Sdown (M1, 2) → "ss1" | Sdown (M1, −2) → "ss1c"
| Sdown (M1, 3) → "sb1" | Sdown (M1, −3) → "sb1c"
| Sdown (M2, 1) → "sd2" | Sdown (M2, −1) → "sd2c"
| Sdown (M2, 2) → "ss2" | Sdown (M2, −2) → "ss2c"
| Sdown (M2, 3) → "sb2" | Sdown (M2, −3) → "sb2c"
| Neutralino N1 → "neu1"
| Neutralino N2 → "neu2"
| Neutralino N3 → "neu3"
| Neutralino N4 → "neu4"
| Neutralino N5 → "neu5"
| Chargino C1 → "ch1+" | Chargino C1c → "ch1-"
| Chargino C2 → "ch2+" | Chargino C2c → "ch2-"
| _ → invalid_arg "Modellib_NMSSM.NMSSM.flavor_to_string"
```

```
let flavor_to_TeX = function
  | L 1 → "e^-" | L (−1) → "e^+"
  | L 2 → "\\mu^-" | L (−2) → "\\mu^+"
  | L 3 → "\\tau^-" | L (−3) → "\\tau^+"
  | N 1 → "\\nu_e" | N (−1) → "\\bar{\\nu}_e"
  | N 2 → "\\nu_\\mu" | N (−2) → "\\bar{\\nu}_\\mu"
  | N 3 → "\\nu_\\tau" | N (−3) → "\\bar{\\nu}_\\tau"
  | U 1 → "u" | U (−1) → "\\bar{u}"
  | U 2 → "c" | U (−2) → "\\bar{c}"
  | U 3 → "t" | U (−3) → "\\bar{t}"
  | D 1 → "d" | D (−1) → "\\bar{d}"
  | D 2 → "s" | D (−2) → "\\bar{s}"
  | D 3 → "b" | D (−3) → "\\bar{b}"
  | L _ → invalid_arg
      "Modellib_NMSSM.NMSSM.flavor_to_TeX:␣invalid␣lepton"
  | N _ → invalid_arg
      "Modellib_NMSSM.NMSSM.flavor_to_TeX:␣invalid␣neutrino"
  | U _ → invalid_arg
      "Modellib_NMSSM.NMSSM.flavor_to_TeX:␣invalid␣up␣type␣quark"
  | D _ → invalid_arg
      "Modellib_NMSSM.NMSSM.flavor_to_TeX:␣invalid␣down␣type␣quark"
  | Gl → "g" | Gluino → "\\widetilde{g}"
  | Ga → "\\gamma" | Z → "Z" | Wp → "W^+" | Wm → "W^-"
  | SHiggs S1 → "S_1" | SHiggs S2 → "S_2" | SHiggs S3 → "S_3"
  | PHiggs P1 → "P_1" | PHiggs P2 → "P_2"
  | Hp → "H^+" | Hm → "H^-"
  | Slepton (M1, 1) → "\\widetilde{e}_1^-"
  | Slepton (M1, −1) → "\\widetilde{e}_1^+"
  | Slepton (M1, 2) → "\\widetilde{\\mu}_1^-"
  | Slepton (M1, −2) → "\\widetilde{\\mu}_1^+"
  | Slepton (M1, 3) → "\\widetilde{\\tau}_1^-"
  | Slepton (M1, −3) → "\\widetilde{\\tau}_1^+"
  | Slepton (M2, 1) → "\\widetilde{e}_2^-"
  | Slepton (M2, −1) → "\\widetilde{e}_2^+"
  | Slepton (M2, 2) → "\\widetilde{\\mu}_2^-"
  | Slepton (M2, −2) → "\\widetilde{\\mu}_2^+"
  | Slepton (M2, 3) → "\\widetilde{\\tau}_2^-"
  | Slepton (M2, −3) → "\\widetilde{\\tau}_2^+"
  | Sneutrino 1 → "\\widetilde{\\nu}_e"
  | Sneutrino (−1) → "\\widetilde{\\nu}_e^*"
  | Sneutrino 2 → "\\widetilde{\\nu}_\\mu"
  | Sneutrino (−2) → "\\widetilde{\\nu}_\\mu^*"
  | Sneutrino 3 → "\\widetilde{\\nu}_\\tau"
  | Sneutrino (−3) → "\\widetilde{\\nu}_\\tau^*"
  | Sup (M1, 1) → "\\widetilde{u}_1"
  | Sup (M1, −1) → "\\widetilde{u}_1^*"
  | Sup (M1, 2) → "\\widetilde{c}_1"
  | Sup (M1, −2) → "\\widetilde{c}_1^*"
  | Sup (M1, 3) → "\\widetilde{t}_1"
  | Sup (M1, −3) → "\\widetilde{t}_1^*"
```

```
  | Sup (M2, 1) → "\\widetilde{u}_2"
  | Sup (M2, -1) → "\\widetilde{u}_2^*"
  | Sup (M2, 2) → "\\widetilde{c}_2"
  | Sup (M2, -2) → "\\widetilde{c}_2^*"
  | Sup (M2, 3) → "\\widetilde{t}_2"
  | Sup (M2, -3) → "\\widetilde{t}_2^*"
  | Sdown (M1, 1) → "\\widetilde{d}_1"
  | Sdown (M1, -1) → "\\widetilde{d}_1^*"
  | Sdown (M1, 2) → "\\widetilde{s}_1"
  | Sdown (M1, -2) → "\\widetilde{s}_1^*"
  | Sdown (M1, 3) → "\\widetilde{b}_1"
  | Sdown (M1, -3) → "\\widetilde{b}_1^*"
  | Sdown (M2, 1) → "\\widetilde{d}_2"
  | Sdown (M2, -1) → "\\widetilde{d}_2^*"
  | Sdown (M2, 2) → "\\widetilde{s}_2"
  | Sdown (M2, -2) → "\\widetilde{s}_2^*"
  | Sdown (M2, 3) → "\\widetilde{b}_2"
  | Sdown (M2, -3) → "\\widetilde{b}_2^*"
  | Neutralino N1 → "\\widetilde{\\chi}^0_1"
  | Neutralino N2 → "\\widetilde{\\chi}^0_2"
  | Neutralino N3 → "\\widetilde{\\chi}^0_3"
  | Neutralino N4 → "\\widetilde{\\chi}^0_4"
  | Neutralino N5 → "\\widetilde{\\chi}^0_5"
  | Slepton _ → invalid_arg
      "Modellib_NMSSM.NMSSM.flavor_to_TeX: invalid slepton"
  | Sneutrino _ → invalid_arg
      "Modellib_NMSSM.NMSSM.flavor_to_TeX: invalid sneutrino"
  | Sup _ → invalid_arg
      "Modellib_NMSSM.NMSSM.flavor_to_TeX: invalid up type squark"
  | Sdown _ → invalid_arg
      "Modellib_NMSSM.NMSSM.flavor_to_TeX: invalid down type squark"
  | Chargino C1 → "\\widetilde{\\chi}_1^+"
  | Chargino C1c → "\\widetilde{\\chi}_1^-"
  | Chargino C2 → "\\widetilde{\\chi}_2^+"
  | Chargino C2c → "\\widetilde{\\chi}_2^-"

let flavor_symbol = function
  | L g when g > 0 → "l" ^ string_of_int g
  | L g → "l" ^ string_of_int (abs g) ^ "b"
  | N g when g > 0 → "n" ^ string_of_int g
  | N g → "n" ^ string_of_int (abs g) ^ "b"
  | U g when g > 0 → "u" ^ string_of_int g
  | U g → "u" ^ string_of_int (abs g) ^ "b"
  | D g when g > 0 → "d" ^ string_of_int g
  | D g → "d" ^ string_of_int (abs g) ^ "b"
  | Gl → "gl"
  | Ga → "a" | Z → "z"
  | Wp → "wp" | Wm → "wm"
  | Slepton (M1, g) when g > 0 → "sl1" ^ string_of_int g
  | Slepton (M1, g) → "sl1c" ^ string_of_int (abs g)
```

```
| Slepton (M2, g) when g > 0 → "sl2" ^ string_of_int g
| Slepton (M2, g) → "sl2c" ^ string_of_int (abs g)
| Sneutrino g when g > 0 → "sn" ^ string_of_int g
| Sneutrino g → "snc" ^ string_of_int (abs g)
| Sup (M1, g) when g > 0 → "su1" ^ string_of_int g
| Sup (M1, g) → "su1c" ^ string_of_int (abs g)
| Sup (M2, g) when g > 0 → "su2" ^ string_of_int g
| Sup (M2, g) → "su2c" ^ string_of_int (abs g)
| Sdown (M1, g) when g > 0 → "sd1" ^ string_of_int g
| Sdown (M1, g) → "sd1c" ^ string_of_int (abs g)
| Sdown (M2, g) when g > 0 → "sd2" ^ string_of_int g
| Sdown (M2, g) → "sd2c" ^ string_of_int (abs g)
| Neutralino n → "neu" ^ (string_of_neu n)
| Chargino c when (int_of_char c) > 0 → "cp" ^ string_of_char c
| Chargino c → "cm" ^ string_of_int (abs (int_of_char c))
| Gluino → "sgl"
| SHiggs s → "h0" ^ (string_of_shiggs s)
| PHiggs p → "A0" ^ (string_of_phiggs p)
| Hp → "hp" | Hm → "hm"

let pdg = function
| L g when g > 0 → 9 + 2 × g
| L g → − 9 + 2 × g
| N g when g > 0 → 10 + 2 × g
| N g → − 10 + 2 × g
| U g when g > 0 → 2 × g
| U g → 2 × g
| D g when g > 0 → − 1 + 2 × g
| D g → 1 + 2 × g
| Gl → 21
| Ga → 22 | Z → 23
| Wp → 24 | Wm → (−24)
| SHiggs S1 → 25 | SHiggs S2 → 35 | SHiggs S3 → 45
| PHiggs P1 → 36 | PHiggs P2 → 46
| Hp → 37 | Hm → (−37)
| Slepton (M1, g) when g > 0 → 1000009 + 2 × g
| Slepton (M1, g) → − 1000009 + 2 × g
| Slepton (M2, g) when g > 0 → 2000009 + 2 × g
| Slepton (M2, g) → − 2000009 + 2 × g
| Sneutrino g when g > 0 → 1000010 + 2 × g
| Sneutrino g → − 1000010 + 2 × g
| Sup (M1, g) when g > 0 → 1000000 + 2 × g
| Sup (M1, g) → − 1000000 + 2 × g
| Sup (M2, g) when g > 0 → 2000000 + 2 × g
| Sup (M2, g) → − 2000000 + 2 × g
| Sdown (M1, g) when g > 0 → 999999 + 2 × g
| Sdown (M1, g) → − 999999 + 2 × g
| Sdown (M2, g) when g > 0 → 1999999 + 2 × g
| Sdown (M2, g) → − 1999999 + 2 × g
| Gluino → 1000021
```

| *Chargino C1* $\rightarrow$ 1000024 | *Chargino C1c* $\rightarrow$ ($-$1000024)
| *Chargino C2* $\rightarrow$ 1000037 | *Chargino C2c* $\rightarrow$ ($-$1000037)
| *Neutralino N1* $\rightarrow$ 1000022 | *Neutralino N2* $\rightarrow$ 1000023
| *Neutralino N3* $\rightarrow$ 1000025 | *Neutralino N4* $\rightarrow$ 1000035
| *Neutralino N5* $\rightarrow$ 1000045

We must take care of the pdg numbers for the two different kinds of sfermions in the MSSM. The particle data group in its Monte Carlo particle numbering scheme takes only into account mixtures of the third generation squarks and the stau. For the other sfermions we will use the number of the lefthanded field for the lighter mixed state and the one for the righthanded for the heavier. Below are the official pdg numbers from the Particle Data Group. In order not to produce arrays with some million entries in the Fortran code for the masses and the widths we introduce our private pdg numbering scheme which only extends not too far beyond 42. Our private scheme then has the following pdf numbers (for the sparticles the subscripts L/R and $1/2$ are taken synonymously):

d	down-quark	1
u	up-quark	2
s	strange-quark	3
c	charm-quark	4
b	bottom-quark	5
t	top-quark	6
e^-	electron	11
ν_e	electron-neutrino	12
μ^-	muon	13
ν_μ	muon-neutrino	14
τ^-	tau	15
ν_τ	tau-neutrino	16
g	gluon	(9) 21
γ	photon	22
Z^0	Z-boson	23
W^+	W-boson	24
h^0	light Higgs boson	25
H^0	heavy Higgs boson	35
A^0	pseudoscalar Higgs	36
H^+	charged Higgs	37
$\tilde{d}_L$	down-squark 1	41
$\tilde{u}_L$	up-squark 1	42
$\tilde{s}_L$	strange-squark 1	43
$\tilde{c}_L$	charm-squark 1	44
$\tilde{b}_L$	bottom-squark 1	45
$\tilde{t}_L$	top-squark 1	46
$\tilde{d}_R$	down-squark 2	47
$\tilde{u}_R$	up-squark 2	48
$\tilde{s}_R$	strange-squark 2	49
$\tilde{c}_R$	charm-squark 2	50
$\tilde{b}_R$	bottom-squark 2	51
$\tilde{t}_R$	top-squark 2	52
$\tilde{e}_L$	selectron 1	53
$\tilde{\nu}_{e,L}$	electron-sneutrino	54
$\tilde{\mu}_L$	smuon 1	55
$\tilde{\nu}_{\mu,L}$	muon-sneutrino	56
$\tilde{\tau}_L$	stau 1	57
$\tilde{\nu}_{\tau,L}$	tau-sneutrino	58
$\tilde{e}_R$	selectron 2	59
$\tilde{\mu}_R$	smuon 2	61
$\tilde{\tau}_R$	stau 2	63
$\tilde{g}$	gluino	64
$\tilde{\chi}^0_1$	neutralino 1	65
$\tilde{\chi}^0_2$	neutralino 2	66
$\tilde{\chi}^0_3$	neutralino 3	67
$\tilde{\chi}^0_4$	neutralino 4	68
$\tilde{\chi}^0_5$	neutralino 5	69
$\tilde{\chi}^+_1$	chargino 1	70

```
let pdg_mw = function
  | L g when g > 0 → 9 + 2 × g
  | L g → − 9 + 2 × g
  | N g when g > 0 → 10 + 2 × g
  | N g → − 10 + 2 × g
  | U g when g > 0 → 2 × g
  | U g → 2 × g
  | D g when g > 0 → − 1 + 2 × g
  | D g → 1 + 2 × g
  | Gl → 21
  | Ga → 22 | Z → 23
  | Wp → 24 | Wm → (−24)
  | SHiggs S1 → 25 | SHiggs S2 → 35 | PHiggs P1 → 36
  | Hp → 37 | Hm → (−37)
  | Sup (M1, g) when g > 0 → 40 + 2 × g
  | Sup (M1, g) → − 40 + 2 × g
  | Sup (M2, g) when g > 0 → 46 + 2 × g
  | Sup (M2, g) → − 46 + 2 × g
  | Sdown (M1, g) when g > 0 → 39 + 2 × g
  | Sdown (M1, g) → − 39 + 2 × g
  | Sdown (M2, g) when g > 0 → 45 + 2 × g
  | Sdown (M2, g) → − 45 + 2 × g
  | Slepton (M1, g) when g > 0 → 51 + 2 × g
  | Slepton (M1, g) → − 51 + 2 × g
  | Slepton (M2, g) when g > 0 → 57 + 2 × g
  | Slepton (M2, g) → − 57 + 2 × g
  | Sneutrino g when g > 0 → 52 + 2 × g
  | Sneutrino g → − 52 + 2 × g
  | Gluino → 64
  | Chargino C1 → 70 | Chargino C1c → (−70)
  | Chargino C2 → 71 | Chargino C2c → (−71)
  | Neutralino N1 → 65 | Neutralino N2 → 66
  | Neutralino N3 → 67 | Neutralino N4 → 68
  | Neutralino N5 → 69
  | PHiggs P2 → 72 | SHiggs S3 → 73

let mass_symbol f =
  "mass(" ^ string_of_int (abs (pdg_mw f)) ^ ")"

let width_symbol f =
  "width(" ^ string_of_int (abs (pdg_mw f)) ^ ")"

let conj_symbol = function
  | false, str → str
  | true, str → str ^ "_c"

let constant_symbol = function
  | E → "e" | G → "g"
  | Mu → "mu" | Lambda → "lambda" | G_Z → "gz"
  | Q_lepton → "qlep" | Q_up → "qup" | Q_down → "qdwn"
  | Q_charg → "qchar"
  | G_NC_lepton → "gnclep" | G_NC_neutrino → "gncneu"
```

```
| G_NC_up -> "gncup" | G_NC_down -> "gncdwn"
| G_CC -> "gcc"
| G_CCQ (vc, g1, g2) -> conj_symbol (vc, "g_ccq" ) ^ "(" ^
    string_of_int g1 ^ "," ^ string_of_int g2 ^ ")"
| I_Q_W -> "iqw" | I_G_ZWW -> "igzww"
| G_WWWW -> "gw4" | G_ZZWW -> "gzzww"
| G_PZWW -> "gpzww" | G_PPWW -> "gppww"
| G_GH4_ZZPP (p1, p2) -> "g_ZZA0A0(" ^ string_of_phiggs p1 ^ ","
    ^ string_of_phiggs p2 ^ ")"
| G_GH4_ZZSS (s1, s2) -> "g_ZZh0h0(" ^ string_of_shiggs s1 ^ ","
    ^ string_of_shiggs s2 ^ ")"
| G_GH4_ZZCC -> "g_zzhphm"
| G_GH4_GaGaCC -> "g_AAhphm"
| G_GH4_ZGaCC -> "g_zAhphm"
| G_GH4_WWCC -> "g_wwhphm"
| G_GH4_WWPP (p1, p2) -> "g_WWA0A0(" ^ string_of_phiggs p1 ^ "," ^
    string_of_phiggs p2 ^ ")"
| G_GH4_WWSS (s1, s2) -> "g_WWh0h0(" ^ string_of_shiggs s1 ^ "," ^
    string_of_shiggs s2 ^ ")"
| G_GH4_ZWSC s -> "g_ZWhph0(" ^ string_of_shiggs s ^")"
| G_GH4_GaWSC s -> "g_AWhph0(" ^ string_of_shiggs s ^")"
| G_GH4_ZWPC p -> "g_ZWhpA0(" ^ string_of_phiggs p ^")"
| G_GH4_GaWPC p -> "g_AWhpA0(" ^ string_of_phiggs p ^")"
| G_CICIS (n1, n2, s) -> "g_neuneuh0(" ^ string_of_neu n1 ^ "," ^
    string_of_neu n2 ^ "," ^ string_of_shiggs s ^ ")"
| G_CICIP (n1, n2, p) -> "g_neuneuA0(" ^ string_of_neu n1 ^ "," ^
    string_of_neu n2 ^ "," ^ string_of_phiggs p ^ ")"
| G_H3_SCC s -> "g_h0hphm(" ^ string_of_shiggs s ^ ")"
| G_H3_SPP (s, p1, p2) -> "g_h0A0A0(" ^ string_of_shiggs s ^ "," ^
    string_of_phiggs p1 ^ "," ^ string_of_phiggs p2 ^ ")"
| G_H3_SSS (s1, s2, s3) -> "g_h0h0h0(" ^ string_of_shiggs s1 ^ "," ^
    string_of_shiggs s2 ^ "," ^ string_of_shiggs s3 ^ ")"
| G_CSC (c1, c2, s) -> "g_chchh0(" ^ string_of_char c1 ^ "," ^
    string_of_char c2 ^ "," ^ string_of_shiggs s ^")"
| G_CPC (c1, c2, p) -> "g_chchA0(" ^ string_of_char c1 ^ "," ^
    string_of_char c2 ^ "," ^ string_of_phiggs p ^")"
| G_YUK_FFS (f1, f2, s) -> "g_yuk_h0_" ^ string_of_fermion_type f1 ^
    string_of_fermion_type f2 ^ "(" ^ string_of_shiggs s ^ "," ^
    string_of_fermion_gen f1 ^ ")"
| G_YUK_FFP (f1, f2, p) -> "g_yuk_A0_" ^ string_of_fermion_type f1 ^
    string_of_fermion_type f2 ^ "(" ^ string_of_phiggs p ^ "," ^
    string_of_fermion_gen f1 ^ ")"
| G_YUK_LCN g -> "g_yuk_hp_ln(" ^ string_of_int g ^ ")"
| G_NWC (n, c) -> "g_nwc(" ^ string_of_char c ^ "," ^ string_of_neu n
    ^ ")"
| G_CWN (c, n) -> "g_cwn(" ^ string_of_char c ^ "," ^ string_of_neu n
    ^ ")"
| G_SLSNW (vc, g, m) -> conj_symbol (vc, "g_wslsn") ^ "(" ^
    string_of_int g ^ "," ^ string_of_sfm m ^ ")"
| G_NZN (n1, n2) -> "g_zneuneu(" ^ string_of_neu n1 ^ ","
```

```
      ^ string_of_neu n2 ^ ")"
  | G_CZC (c1, c2) → "g_zchch(" ^ string_of_char c1 ^ "," ^
      string_of_char c2 ^ ")"
  | Gs → "gs"
  | G_YUK_UCD (n, m) → "g_yuk_hp_ud(" ^ string_of_int n ^ "," ^
      string_of_int m ^ ")"
  | G_YUK_DCU (n, m) → "g_yuk_hm_du(" ^ string_of_int n ^ "," ^
      string_of_int m ^ ")"
  | G_YUK_N (vc, f, n, sf, m) → conj_symbol (vc, "g_yuk_neu_" ^
      string_of_fermion_type f ^ string_of_sff sf) ^ "(" ^
      string_of_fermion_gen f ^ "," ^ string_of_neu n ^ "," ^
      string_of_sfm m ^ ")"
  | G_YUK_G (vc, f, sf, m) → conj_symbol (vc, "g_yuk_gluino_" ^
      string_of_fermion_type f ^ string_of_sff sf) ^ "(" ^
      string_of_fermion_gen f ^ "," ^ string_of_sfm m ^ ")"
  | G_YUK_C (vc, f, c, sf, m) → conj_symbol (vc, "g_yuk_char_" ^
      string_of_fermion_type f ^ string_of_sff sf) ^ "(" ^
      string_of_fermion_gen f ^ "," ^ string_of_char c ^ "," ^
      string_of_sfm m ^ ")"
  | G_YUK_Q (vc, g1, f, c, sf, m) → conj_symbol (vc, "g_yuk_char_" ^
      string_of_fermion_type f ^ string_of_sff sf) ^ "(" ^ string_of_int
      g1 ^ "," ^ string_of_fermion_gen f ^ "," ^ string_of_char c ^ ","
      ^ string_of_sfm m ^ ")"
  | G_WPSUSD (vc, m1, m2, g1, g2) → conj_symbol (vc, "g_wA_susd") ^ "(" ^
      string_of_int g1 ^ "," ^ string_of_int g2 ^ "," ^ string_of_sfm m1 ^
      "," ^ string_of_sfm m2 ^ ")"
  | G_WZSUSD (vc, m1, m2, g1, g2) → conj_symbol (vc, "g_wz_susd") ^ "(" ^
      string_of_int g1 ^ "," ^ string_of_int g2 ^ "," ^ string_of_sfm m1 ^
      "," ^ string_of_sfm m2 ^ ")"
  | G_GH_ZSP (s, p) → "g_zh0a0(" ^ string_of_shiggs s ^ "," ^
      string_of_phiggs p ^ ")"
  | G_GH_WSC s → "g_Whph0(" ^ string_of_shiggs s ^ ")"
  | G_GH_WPC p → "g_WhpA0(" ^ string_of_phiggs p ^ ")"
  | G_GH_ZZS s → "g_ZZh0(" ^ string_of_shiggs s ^ ")"
  | G_GH_WWS s → "g_WWh0(" ^ string_of_shiggs s ^ ")"
  | G_GLUGLUH0 s → "g_glugluh0(" ^ string_of_shiggs s ^ ")"
  | G_GLUGLUA0 p → "g_gluglua0(" ^ string_of_phiggs p ^ ")"
  | G_GH_ZCC → "g_Zhmhp"
  | G_GH_GaCC → "g_Ahmhp"
  | G_ZSF (f, g, m1, m2) → "g_z" ^ string_of_sff f ^ string_of_sff f ^ "(" ^
      string_of_int g ^ "," ^ string_of_sfm m1 ^ "," ^ string_of_sfm m2
      ^ ")"
  | G_HSNSL (vc, g, m) → conj_symbol (vc, "g_hp_sl" ^ string_of_sfm m ^
      "sn1") ^ "(" ^ string_of_int g ^ ")"
  | G_GlGlSQSQ → "g_gg_sqsq"
  | G_PPSFSF f → "g_AA_" ^ string_of_sff f ^ string_of_sff f
  | G_ZZSFSF (f, g, m1, m2) → "g_zz_" ^ string_of_sff f ^ string_of_sff f ^
      "(" ^ string_of_int g ^"," ^ string_of_sfm m1
      ^ "," ^ string_of_sfm m2 ^ ")"
  | G_ZPSFSF (f, g, m1, m2) → "g_zA_" ^ string_of_sff f ^ string_of_sff f ^
```

```
        "(" ^ string_of_int g ^"","^ string_of_sfm m1
        ^ "," ^ string_of_sfm m2 ^ ")"
    | G_GlPSQSQ → "g_gA_sqsq"
    | G_GlZSFSF (f, g, m1, m2) → "g_gz_" ^ string_of_sff f ^ string_of_sff f ^
        "(" ^ string_of_int g ^ "," ^ string_of_sfm m1 ^ "," ^ string_of_sfm
        m2 ^ ")"
    | G_GlWSUSD (vc, m1, m2, g1, g2) → conj_symbol (vc, "g_gw_susd") ^ "(" ^
        string_of_int g1 ^ "," ^ string_of_int g2 ^ "," ^ string_of_sfm m1 ^
        "," ^ string_of_sfm m2 ^ ")"
    | G_SS → "gs**2"
    | I_G_S → "igs"
    | G_NHC (vc, n, c) → conj_symbol(vc,"g_neuhmchar") ^ "(" ^
        string_of_neu n ^ "," ^ string_of_char c ^")"
    | G_WWSFSF (f, g, m1, m2) → "g_ww_" ^ string_of_sff f
        ^ string_of_sff f ^"(" ^ string_of_int g ^ "," ^ string_of_sfm m1 ^
        "," ^ string_of_sfm m2 ^ ")"
    | G_WPSLSN (vc, g, m) → conj_symbol (vc, "g_wA_slsn") ^ "(" ^
        string_of_int g ^ "," ^ string_of_sfm m ^ ")"
    | G_WZSLSN (vc, g, m) → conj_symbol (vc, "g_wz_slsn") ^"("^ string_of_int
        g ^ "," ^ string_of_sfm m ^ ")"
    | G_SFSFS (s, f, g, m1, m2) → "g_h0_"^ string_of_sff f ^ string_of_sfm m1
        ^ string_of_sff f ^ string_of_sfm m2 ^ "(" ^ string_of_shiggs s ^ ","
        ^ string_of_int g ^ ")"
    | G_SFSFP (p, f, g, m1, m2) → "g_A0_"^ string_of_sff f ^ string_of_sfm m1
        ^ string_of_sff f ^ string_of_sfm m2 ^ "(" ^ string_of_phiggs p ^ ","
        ^ string_of_int g ^ ")"
    | G_HSUSD (vc, m1, m2, g1, g2) → conj_symbol (vc, "g_hp_su" ^ string_of_sfm
        m1 ^ "sd" ^ string_of_sfm m2 )^ "(" ^ string_of_int g1 ^ ","
        ^ string_of_int g2 ^")"
    | G_WSQ (vc, g1, g2, m1, m2) → conj_symbol (vc, "g_wsusd") ^ "("
        ^ string_of_int g1 ^ "," ^ string_of_int g2 ^ "," ^ string_of_sfm m1
        ^ "," ^ string_of_sfm m2 ^ ")"
end
```

13.11 Interface of *Modellib_PSSSM*

13.11.1 Extended Supersymmetric Models

We do not introduce the possibility here of using four point couplings or not. We simply add the relevant and leave the rest out. No possibility for Goldstone bosons is given. But we allow for CKM mixing.

```
module type extMSSM_flags =
  sig
    val ckm_present : bool
  end

module PSSSM : extMSSM_flags
module PSSSM_QCD : extMSSM_flags
```

module *ExtMSSM* : functor (*F* : *extMSSM_flags*) → *Model.T* with module *Ch* = *Charges.QQ*

13.12 Implementation of Modellib_PSSSM

```
let rcs_file = RCS.parse "Modellib_PSSSM" ["Extended␣SUSY␣models"]
  { RCS.revision = "$Revision:␣6465␣$";
    RCS.date = "$Date:␣2015-01-10␣16:22:31␣+0100␣(Sat,␣10␣Jan␣2015)␣$";
    RCS.author = "$Author:␣jr_reuter␣$";
    RCS.source
      = "$URL:␣svn+ssh://jr_reuter@login.hepforge.org/hepforge/svn/whizard/trunk/omega/sr
```

13.12.1 Extended Supersymmetric Standard Model(s)

This is based on the NMSSM implementation by Felix Braam, and extended to the exotica – leptoquarks, leptoquarkinos, additional Higgses etc. – by Daniel Wiesler. Note that for the Higgs sector vertices the conventions of the Franke/Fraas paper have been used.

```
module type extMSSM_flags =
  sig
    val ckm_present : bool
  end

module PSSSM : extMSSM_flags =
  struct
    let ckm_present = false
  end

module PSSSM_QCD : extMSSM_flags =
  struct
    let ckm_present = false
  end

module ExtMSSM (Flags : extMSSM_flags) =
  struct
    let rcs = RCS.rename rcs_file "Modellib_PSSSM.NMSSM"
        [ "Extended␣SUSY␣models" ]

    open Coupling

    let default_width = ref Timelike
    let use_fudged_width = ref false

    let options = Options.create
      [ "constant_width", Arg.Unit (fun () → default_width := Constant),
        "use␣constant␣width␣(also␣in␣t-channel)";
        "fudged_width", Arg.Set use_fudged_width,
        "use␣fudge␣factor␣for␣charge␣particle␣width";
        "custom_width", Arg.String (fun f → default_width := Custom f),
        "use␣custom␣width";
        "cancel_widths", Arg.Unit (fun () → default_width := Vanishing),
```

```
    "use␣vanishing␣width"]
```

additional combinatorics
yields a list of tuples consistig of the off-diag combinations of the elements in "set"

```
let choose2 set =
  List.map (function [x; y] -> (x, y) | _ -> failwith "choose2")
    (Combinatorics.choose 2 set)
```

pairs
pairs appends the diagonal combinations to *choose2*

```
let rec diag = function
  | [] -> []
  | x1 :: rest -> (x1, x1) :: diag rest

let pairs l = choose2 l @ diag l
```

triples
rank 3 generalization of *pairs*

```
let rec cloop set i j k =
  if i > ((List.length set) - 1) then []
  else if j > i then cloop set (succ i) (j - i - 1) (j - i - 1)
  else if k > j then cloop set i (succ j) (k - j - 1)
  else (List.nth set i, List.nth set j, List.nth set k) :: cloop set i j (succ k)

let triples set = cloop set 0 0 0
```

(* *two_and_one* *)

```
let rec two_and_one' l1 z n =
    if n < 0 then []
    else
    ((fst (List.nth (pairs l1) n)), (snd (List.nth (pairs l1) n)), z) :: two_and_one' l1 z (pred n)

let two_and_one l1 l2 =
    let f z = two_and_one' l1 z ((List.length (pairs l1)) - 1)
    in
    List.flatten ( List.map f l2 )

type gen =
  | G of int | GG of gen × gen

let rec string_of_gen = function
  | G n when n > 0 -> string_of_int n
  | G n -> string_of_int (abs n) ^ "c"
  | GG (g1, g2) -> string_of_gen g1 ^ "_" ^ string_of_gen g2
```

With this we distinguish the flavour.

```
type sff =
  | SL | SN | SU | SD

let string_of_sff = function
  | SL -> "sl" | SN -> "sn" | SU -> "su" | SD -> "sd"
```

With this we distinguish the mass eigenstates. At the moment we have to cheat a little bit for the sneutrinos. Because we are dealing with massless neutrinos there is only one sort of sneutrino.

```
type sfm =
  | M1 | M2

let string_of_sfm = function
  | M1 → "1" | M2 → "2"
```

We also introduce special types for the charginos and neutralinos.

```
type char =
  | C1 | C2 | C1c | C2c | C3 | C3c | C4 | C4c

type neu =
  | N1 | N2 | N3 | N4 | N5 | N6 | N7 | N8 | N9 | N10 | N11

let int_of_char = function
  | C1 → 1 | C2 → 2 | C1c → − 1 | C2c → − 2
  | C3 → 3 | C4 → 4 | C3c → − 3 | C4c → − 4

let string_of_char c = string_of_int (int_of_char c)

let conj_char = function
  | C1 → C1c | C2 → C2c | C1c → C1 | C2c → C2
  | C3 → C3c | C4 → C4c | C3c → C3 | C4c → C4

let string_of_neu = function
  | N1 → "1" | N2 → "2" | N3 → "3" | N4 → "4" | N5 →
"5" | N6 → "6"
  | N7 → "7" | N8 → "8" | N9 → "9" | N10 → "10"| N11 → "11"
```

For NMSSM-like the Higgs bosons, we follow the conventions of Franke/Fraas. Daniel Wiesler: extended to E6 models.

```
type shiggs =
  | S1 | S2 | S3 | S4 | S5 | S6 | S7 | S8 | S9

type phiggs =
  | P1 | P2 | P3 | P4 | P5 | P6 | P7
```

HCx is always the H^+, *HCxc* the H^-.

```
type chiggs =
  | HC1 | HC2 | HC3 | HC4 | HC5 | HC1c | HC2c | HC3c |
HC4c | HC5c

let conj_chiggs = function
  | HC1 → HC1c | HC2 → HC2c | HC1c → HC1 | HC2c → HC2
  | HC3 → HC3c | HC4 → HC4c | HC3c → HC3 | HC4c → HC4
  | HC5 → HC5c | HC5c → HC5

let string_of_shiggs = function
  | S1 → "1" | S2 → "2" | S3 → "3" | S4 → "4" | S5 → "5"
  | S6 → "6" | S7 → "7" | S8 → "8" | S9 → "9"

let string_of_phiggs = function
  | P1 → "1" | P2 → "2" | P3 → "3" | P4 → "4" | P5 → "5"
```

```
  | P6 → "6" | P7 → "7"
let nlist = [ N1; N2; N3; N4; N5; N6; N7; N8; N9; N10; N11 ]
let slist = [ S1; S2; S3; S4; S5; S6; S7; S8; S9 ]
let plist = [ P1; P2; P3; P4; P5; P6; P7 ]
let clist = [ HC1; HC2; HC3; HC4; HC5; HC1c; HC2c; HC3c; HC4c; HC5c ]
let charlist = [ C1; C2; C3; C4; C1c; C2c; C3c; C4c ]

type flavor =
  | L of int | N of int
  | U of int | D of int
  | Sup of sfm × int | Sdown of sfm × int
  | Ga | Wp | Wm | Z | Gl
  | Slepton of sfm × int | Sneutrino of int
  | Neutralino of neu | Chargino of char
  | Gluino | SHiggs of shiggs | PHiggs of phiggs
  | CHiggs of chiggs
  | LQ of sfm × int
  | LQino of int

let string_of_fermion_type = function
  | L _ → "l" | U _ → "u" | D _ → "d" | N _ → "n"
  | _ → failwith
        "Modellib_PSSSM.ExtMSSM.string_of_fermion_type: invalid fermion type"

let string_of_fermion_gen = function
  | L g | U g | D g | N g → string_of_int (abs (g))
  | _ → failwith
        "Modellib_PSSSM.ExtMSSM.string_of_fermion_gen: invalid fermion type"

type gauge = unit

let gauge_symbol () =
  failwith "Modellib_PSSSM.ExtMSSM.gauge_symbol: internal error"
```

At this point we will forget graviton and -ino.

```
let family g = [ L g; N g; Slepton (M1, g);
                 Slepton (M2, g); Sneutrino g;
                 U g; D g; Sup (M1, g); Sup (M2, g);
                 Sdown (M1, g); Sdown (M2, g);
                 LQ (M1, g); LQ (M2, g); LQino g ]

let external_flavors () =
    [ "1st Generation matter", ThoList.flatmap family [1; − 1];
      "2nd Generation matter", ThoList.flatmap family [2; − 2];
      "3rd Generation matter", ThoList.flatmap family [3; − 3];
      "Gauge Bosons", [Ga; Z; Wp; Wm; Gl];
      "Charginos", List.map (fun a → Chargino a) charlist;
      "Neutralinos", List.map (fun a → Neutralino a) nlist;
      "Higgs Bosons", List.map (fun a → SHiggs a) slist @
                      List.map (fun a → PHiggs a) plist @
                      List.map (fun a → CHiggs a) clist;
      "Gluino", [Gluino]]
```

```
let flavors () = ThoList.flatmap snd (external_flavors ())

let spinor n m =
  if n ≥ 0 ∧ m ≥ 0 then
    Spinor
  else if
    n ≤ 0 ∧ m ≤ 0 then
    ConjSpinor
  else
    invalid_arg "Modellib_PSSSM.ExtMSSM.spinor:␣internal␣error"

let lorentz = function
  | L g → spinor g 0 | N g → spinor g 0
  | U g → spinor g 0 | D g → spinor g 0
  | LQino g → spinor g 0
  | Chargino c → spinor (int_of_char c) 0
  | Ga | Gl → Vector
  | Wp | Wm | Z → Massive_Vector
  | SHiggs _ | PHiggs _ | CHiggs _
  | Sup _ | Sdown _ | Slepton _ | Sneutrino _ | LQ _ → Scalar
  | Neutralino _ | Gluino → Majorana

let color = function
  | U g → Color.SUN (if g > 0 then 3 else -3)
  | Sup (m, g) → Color.SUN (if g > 0 then 3 else -3)
  | D g → Color.SUN (if g > 0 then 3 else -3)
  | Sdown (m, g) → Color.SUN (if g > 0 then 3 else -3)
  | LQ (m, g) → Color.SUN (if g > 0 then 3 else -3)
  | LQino g → Color.SUN (if g > 0 then 3 else -3)
  | Gl | Gluino → Color.AdjSUN 3
  | _ → Color.Singlet

let prop_spinor n m =
  if n ≥ 0 ∧ m ≥ 0 then
    Prop_Spinor
  else if
    n ≤ 0 ∧ m ≤ 0 then
    Prop_ConjSpinor
  else
    invalid_arg "Modellib_PSSSM.ExtMSSM.prop_spinor:␣internal␣error"

let propagator = function
  | L g → prop_spinor g 0 | N g → prop_spinor g 0
  | U g → prop_spinor g 0 | D g → prop_spinor g 0
  | LQino g → prop_spinor g 0
  | Chargino c → prop_spinor (int_of_char c) 0
  | Ga | Gl → Prop_Feynman
  | Wp | Wm | Z → Prop_Unitarity
  | SHiggs _ | PHiggs _ | CHiggs _ → Prop_Scalar
  | Sup _ | Sdown _ | Slepton _ | Sneutrino _ → Prop_Scalar
  | LQ _ → Prop_Scalar
  | Gluino → Prop_Majorana
  | Neutralino _ → Prop_Majorana
```

Optionally, ask for the fudge factor treatment for the widths of charged particles. Currently, this only applies to $W^{\pm}$ and top.

```
let width f =
  if !use_fudged_width then
    match f with
    | Wp | Wm | U 3 | U (-3) → Fudged
    | _ → !default_width
  else
    !default_width

let goldstone _ = None

let conjugate = function
  | L g → L (-g) | N g → N (-g)
  | U g → U (-g) | D g → D (-g)
  | Sup (m, g) → Sup (m, -g)
  | Sdown (m, g) → Sdown (m, -g)
  | Slepton (m, g) → Slepton (m, -g)
  | Sneutrino g → Sneutrino (-g)
  | Gl → Gl | Ga → Ga | Z → Z
  | Wp → Wm | Wm → Wp
  | SHiggs s → SHiggs s
  | PHiggs p → PHiggs p
  | CHiggs c → CHiggs (conj_chiggs c)
  | Gluino → Gluino
  | Neutralino n → Neutralino n | Chargino c → Chargino (conj_char c)
  | LQino g → LQino (-g)
  | LQ (m, g) → LQ (m, -g)

let fermion = function
  | L g → if g > 0 then 1 else -1
  | N g → if g > 0 then 1 else -1
  | U g → if g > 0 then 1 else -1
  | D g → if g > 0 then 1 else -1
  | Gl | Ga | Z | Wp | Wm → 0
  | SHiggs _ | PHiggs _ | CHiggs _ → 0
  | Neutralino _ → 2
  | Chargino c → if (int_of_char c) > 0 then 1 else -1
  | Sup _ → 0 | Sdown _ → 0
  | Slepton _ → 0 | Sneutrino _ → 0
  | Gluino → 2
  | LQ _ → 0
  | LQino g → if g > 0 then 1 else -1
```

This model does NOT have a conserved generation quantum number.

```
module Ch = Charges.QQ

let ( // ) = Algebra.Small_Rational.make

let charge = function
  | L n → if n > 0 then -1//1 else 1//1
  | Slepton (_, n) → if n > 0 then -1//1 else 1//1
```

```
      | N n → 0//1
      | Sneutrino n → 0//1
      | U n → if n > 0 then 2//3 else -2//3
      | Sup (_, n) → if n > 0 then 2//3 else -2//3
      | D n | LQ (_, n) | LQino n → if n > 0 then -1//3 else 1//3
      | Sdown (_, n) → if n > 0 then -1//3 else 1//3
      | Gl | Ga | Z | Neutralino _ | Gluino → 0//1
      | Wp → 1//1
      | Wm → -1//1
      | SHiggs _ | PHiggs _ → 0//1
      | CHiggs (HC1 | HC2 | HC3 | HC4 | HC5) → 1//1
      | CHiggs (HC1c | HC2c | HC3c | HC4c | HC5c) → -1//1
      | Chargino (C1 | C2 | C3 | C4) → 1//1
      | Chargino (C1c | C2c | C3c | C4c) → -1//1

    let lepton = function
      | L n | N n → if n > 0 then 1//1 else -1//1
      | Slepton (_, n)
      | Sneutrino n → if n > 0 then 1//1 else -1//1
      | LQ (_, n) | LQino n → if n > 0 then 1//1 else -1//1
      | _ → 0//1

    let baryon = function
      | U n | D n → if n > 0 then 1//1 else -1//1
      | Sup (_, n) | Sdown (_, n) → if n > 0 then 1//1 else -1//1
      | LQ (_, n) | LQino n → if n > 0 then 1//1 else -1//1
      | _ → 0//1

    let charges f =
      [ charge f; lepton f; baryon f]
```

We introduce a Boolean type vc as a pseudonym for Vertex Conjugator to distinguish between vertices containing complex mixing matrices like the CKM–matrix or the sfermion or neutralino/chargino–mixing matrices, which have to become complex conjugated. The true–option stands for the conjugated vertex, the false–option for the unconjugated vertex.

```
    type vc = bool

    type constant =
      | E | G
      | Q_lepton | Q_up | Q_down | Q_charg
      | G_Z | G_CC | G_CCQ of vc × int×int
      | G_NC_neutrino | G_NC_lepton | G_NC_up | G_NC_down
      | I_Q_W | I_G_ZWW | G_WWWW | G_ZZWW | G_PZWW |
 G_PPWW
      | G_strong | G_SS | I_G_S
      | Gs
      | G_NZN of neu × neu | G_CZC of char×char
      | G_YUK_FFS of flavor × flavor × shiggs
      | G_YUK_FFP of flavor × flavor × phiggs
      | G_YUK_LCN of int
      | G_YUK_UCD of int×int | G_YUK_DCU of int×int
```

$|$ *G_NHC* of $vc \times neu \times char$
$|$ *G_YUK_C* of $vc \times flavor \times char \times sff \times sfm$
$|$ *G_YUK_Q* of $vc \times int \times flavor \times char \times sff \times sfm$
$|$ *G_YUK_N* of $vc \times flavor \times neu \times sff \times sfm$
$|$ *G_YUK_G* of $vc \times flavor \times sff \times sfm$
$|$ *G_NWC* of $neu \times char$ $|$ *G_CWN* of $char \times neu$
$|$ *G_CSC* of $char \times char \times shiggs$
$|$ *G_CPC* of $char \times char \times phiggs$
$|$ *G_WSQ* of $vc \times int \times int \times sfm \times sfm$
$|$ *G_SLSNW* of $vc \times int \times sfm$
$|$ *G_ZSF* of $sff \times int \times sfm \times sfm$
$|$ *G_CICIS* of $neu \times neu \times shiggs$
$|$ *G_CICIP* of $neu \times neu \times phiggs$
$|$ *G_GH_WPC* of $phiggs$ $|$ *G_GH_WSC* of $shiggs$
$|$ *G_GH_ZSP* of $shiggs \times phiggs$ $|$ *G_GH_WWS* of $shiggs$
$|$ *G_GH_ZZS* of $shiggs$ $|$ *G_GH_ZCC*
$|$ *G_GH_GaCC*
$|$ *G_GH4_ZZPP* of $phiggs \times phiggs$
$|$ *G_GH4_ZZSS* of $shiggs \times shiggs$
$|$ *G_GH4_ZZCC* $|$ *G_GH4_GaGaCC*
$|$ *G_GH4_ZGaCC* $|$ *G_GH4_WWCC*
$|$ *G_GH4_WWPP* of $phiggs \times phiggs$
$|$ *G_GH4_WWSS* of $shiggs \times shiggs$
$|$ *G_GH4_ZWSC* of $shiggs$
$|$ *G_GH4_GaWSC* of $shiggs$
$|$ *G_GH4_ZWPC* of $phiggs$
$|$ *G_GH4_GaWPC* of $phiggs$
$|$ *G_WWSFSF* of $sff \times int \times sfm \times sfm$
$|$ *G_WPSLSN* of $vc \times int \times sfm$
$|$ *G_H3_SCC* of $shiggs$
$|$ *G_H3_SSS* of $shiggs \times shiggs \times shiggs$
$|$ *G_H3_SPP* of $shiggs \times phiggs \times phiggs$
$|$ *G_SFSFS* of $shiggs \times sff \times int \times sfm \times sfm$
$|$ *G_SFSFP* of $phiggs \times sff \times int \times sfm \times sfm$
$|$ *G_HSNSL* of $vc \times int \times sfm$
$|$ *G_HSUSD* of $vc \times sfm \times sfm \times int \times int$
$|$ *G_WPSUSD* of $vc \times sfm \times sfm \times int \times int$
$|$ *G_WZSUSD* of $vc \times sfm \times sfm \times int \times int$
$|$ *G_WZSLSN* of $vc \times int \times sfm$ $|$ *G_GlGlSQSQ*
$|$ *G_PPSFSF* of sff
$|$ *G_ZZSFSF* of $sff \times int \times sfm \times sfm$ $|$ *G_ZPSFSF* of $sff \times int \times sfm \times sfm$
$|$ *G_GlZSFSF* of $sff \times int \times sfm \times sfm$ $|$ *G_GlPSQSQ*
$|$ *G_GlWSUSD* of $vc \times sfm \times sfm \times int \times int$
$|$ *G_YUK_LQ_S* of $int \times shiggs \times int$
$|$ *G_YUK_LQ_P* of $int \times phiggs \times int$
$|$ *G_LQ_NEU* of $sfm \times int \times int \times neu$
$|$ *G_LQ_EC_UC* of $vc \times sfm \times int \times int \times int$
$|$ *G_LQ_GG* of $sfm \times int \times int$
$|$ *G_LQ_SSU* of $sfm \times sfm \times sfm \times int \times int \times int$
$|$ *G_LQ_SSD* of $sfm \times sfm \times int \times int \times int$

```
    | G_LQ_S of sfm × sfm × int×shiggs × int
    | G_LQ_P of sfm × sfm × int×phiggs × int
    | G_ZLQ of int×sfm × sfm
    | G_ZZLQLQ | G_ZPLQLQ | G_PPLQLQ | G_ZGlLQLQ |
G_PGlLQLQ | G_NLQC | G_GlGlLQLQ
```

Two integer counters for the QCD and EW order of the couplings.

```
type orders = int × int

let orders = function
  | _ → (0,0)
```

$$\alpha_{\text{QED}} = \frac{1}{137.0359895} \tag{13.86a}$$

$$\sin^2\theta_w = 0.23124 \tag{13.86b}$$

Here we must perhaps allow for complex input parameters. So split them into their modulus and their phase. At first, we leave them real; the generalization to complex parameters is obvious.

```
let parameters () =
  { input = [];
    derived = [];
    derived_arrays = [] }

module F = Modeltools.Fusions (struct
  type f = flavor
  type c = constant
  let compare = compare
  let conjugate = conjugate
end)
```

For the couplings there are generally two possibilities concerning the sign of the covariant derivative.

$$\text{CD}^{\pm} = \partial_\mu \pm \text{i}gT^a A^a_\mu \tag{13.87}$$

The particle data group defines the signs consistently to be positive. Since the convention for that signs also influence the phase definitions of the gaugino/higgsino fields via the off-diagonal entries in their mass matrices it would be the best to adopt that convention.
** REVISED: Compatible with CD+. FB **

```
let electromagnetic_currents_3 g =
    [ ((L (−g), Ga, L g), FBF (1, Psibar, V, Psi), Q_lepton);
      ((U (−g), Ga, U g), FBF (1, Psibar, V, Psi), Q_up);
      ((D (−g), Ga, D g), FBF (1, Psibar, V, Psi), Q_down)]
```

** REVISED: Compatible with CD+. FB**

```
let electromagnetic_sfermion_currents g m =
    [ ((Ga, Slepton (m, −g), Slepton (m, g)), Vector_Scalar_Scalar 1, Q_lepton);
      ((Ga, Sup (m, −g), Sup (m, g)), Vector_Scalar_Scalar 1, Q_up);
      ((Ga, Sdown (m, −g), Sdown (m, g)), Vector_Scalar_Scalar 1, Q_down)]
```

** REVISED: Compatible with CD+. FB**

$\mathsf{let}\ \mathit{electromagnetic\_currents\_2}\ c\ =$
$\quad\mathsf{let}\ cc\ =\ \mathit{conj\_char}\ c\ \mathsf{in}$
$\quad[\ ((\mathit{Chargino}\ cc,\ \mathit{Ga},\ \mathit{Chargino}\ c),\ \mathit{FBF}\ (1,\ \mathit{Psibar},\ V,\ \mathit{Psi}),\ \mathit{Q\_charg})\ ]$

** REVISED: Compatible with CD+. FB**

$\mathsf{let}\ \mathit{neutral\_currents}\ g\ =$
$\quad[\ ((L\ (-g),\ Z,\ L\ g),\ \mathit{FBF}\ (1,\ \mathit{Psibar},\ \mathit{VA},\ \mathit{Psi}),\ \mathit{G\_NC\_lepton});$
$\quad\ ((N\ (-g),\ Z,\ N\ g),\ \mathit{FBF}\ (1,\ \mathit{Psibar},\ \mathit{VA},\ \mathit{Psi}),\ \mathit{G\_NC\_neutrino});$
$\quad\ ((U\ (-g),\ Z,\ U\ g),\ \mathit{FBF}\ (1,\ \mathit{Psibar},\ \mathit{VA},\ \mathit{Psi}),\ \mathit{G\_NC\_up});$
$\quad\ ((D\ (-g),\ Z,\ D\ g),\ \mathit{FBF}\ (1,\ \mathit{Psibar},\ \mathit{VA},\ \mathit{Psi}),\ \mathit{G\_NC\_down})]$

$$\mathcal{L}_{\mathrm{CC}} = \mp\frac{g}{2\sqrt{2}}\sum_i \bar\psi_i\gamma^\mu(1-\gamma_5)(T^+W^+_\mu + T^-W^-_\mu)\psi_i, \tag{13.88}$$

where the sign corresponds to $\mathrm{CD}_\pm$, respectively.
** REVISED: Compatible with CD+. **
Remark: The definition with the other sign compared to the SM files comes from the fact that $g_{cc} = 1/(2\sqrt{2})$ is used overwhelmingly often in the SUSY Feynman rules, so that JR decided to use a different definiton for g_cc in SM and MSSM.
* FB *

$\mathsf{let}\ \mathit{charged\_currents}\ g\ =$
$\quad[\ ((L\ (-g),\ \mathit{Wm},\ N\ g),\ \mathit{FBF}\ ((-1),\ \mathit{Psibar},\ \mathit{VL},\ \mathit{Psi}),\ \mathit{G\_CC});$
$\quad\ ((N\ (-g),\ \mathit{Wp},\ L\ g),\ \mathit{FBF}\ ((-1),\ \mathit{Psibar},\ \mathit{VL},\ \mathit{Psi}),\ \mathit{G\_CC})\ ]$

The quark with the inverted generation (the antiparticle) is the outgoing one, the other the incoming. The vertex attached to the outgoing up-quark contains the CKM matrix element *not* complex conjugated, while the vertex with the outgoing down-quark has the conjugated CKM matrix element.
** REVISED: Compatible with CD+. FB **

$\mathsf{let}\ \mathit{charged\_quark\_currents}\ g\ h\ =$
$\quad[\ ((D\ (-g),\ \mathit{Wm},\ U\ h),\ \mathit{FBF}\ ((-1),\ \mathit{Psibar},\ \mathit{VL},\ \mathit{Psi}),\ \mathit{G\_CCQ}\ (\mathsf{true},g,h));$
$\quad\ ((U\ (-g),\ \mathit{Wp},\ D\ h),\ \mathit{FBF}\ ((-1),\ \mathit{Psibar},\ \mathit{VL},\ \mathit{Psi}),\ \mathit{G\_CCQ}\ (\mathsf{false},h,g))]$

** REVISED: Compatible with CD+.FB **

$\mathsf{let}\ \mathit{charged\_chargino\_currents}\ n\ c\ =$
$\quad\mathsf{let}\ cc\ =\ \mathit{conj\_char}\ c\ \mathsf{in}$
$\quad[\ ((\mathit{Chargino}\ cc,\ \mathit{Wp},\ \mathit{Neutralino}\ n),$
$\qquad\qquad \mathit{FBF}\ (1,\ \mathit{Psibar},\ \mathit{VLR},\ \mathit{Chi}),\ \mathit{G\_CWN}\ (c,n));$
$\quad\ ((\mathit{Neutralino}\ n,\ \mathit{Wm},\ \mathit{Chargino}\ c),$
$\qquad\qquad \mathit{FBF}\ (1,\ \mathit{Chibar},\ \mathit{VLR},\ \mathit{Psi}),\ \mathit{G\_NWC}\ (n,c))\ ]$

** REVISED: Compatible with CD+. FB**

$\mathsf{let}\ \mathit{charged\_slepton\_currents}\ g\ m\ =$
$\quad[\ ((\mathit{Wm},\ \mathit{Slepton}\ (m,-g),\ \mathit{Sneutrino}\ g),\ \mathit{Vector\_Scalar\_Scalar}\ (-1),\ \mathit{G\_SLSNW}$
$\qquad(\mathsf{true},g,m));$
$\quad\ ((\mathit{Wp},\ \mathit{Slepton}\ (m,g),\ \mathit{Sneutrino}\ (-g)),\ \mathit{Vector\_Scalar\_Scalar}\ 1,\ \mathit{G\_SLSNW}$
$\qquad(\mathsf{false},g,m))\ ]$

** REVISED: Compatible with CD+. FB**

```
let charged_squark_currents' g h m1 m2 =
  [ ((Wm, Sup (m1, g), Sdown (m2, -h)), Vector_Scalar_Scalar (-1), G_WSQ
        (true,g, h, m1, m2));
      ((Wp, Sup (m1, -g), Sdown (m2, h)), Vector_Scalar_Scalar 1, G_WSQ
        (false,g, h, m1, m2)) ]
let charged_squark_currents g h =
List.flatten (Product.list2 (charged_squark_currents' g h) [M1; M2] [M1; M2] )
```

** REVISED: Compatible with CD+. FB **

```
let neutral_sfermion_currents' g m1 m2 =
  [ ((Z, Slepton (m1, -g), Slepton (m2, g)), Vector_Scalar_Scalar (-1),
        G_ZSF (SL, g, m1, m2));
    ((Z, Sup (m1, -g), Sup (m2, g)), Vector_Scalar_Scalar (-1),
        G_ZSF (SU, g, m1, m2));
    ((Z, Sdown (m1, -g), Sdown (m2, g)), Vector_Scalar_Scalar (-1),
        G_ZSF (SD, g, m1, m2))]
let neutral_sfermion_currents g =
  List.flatten (Product.list2 (neutral_sfermion_currents'
                g) [M1; M2] [M1; M2]) @
  [ ((Z, Sneutrino (-g), Sneutrino g), Vector_Scalar_Scalar (-1),
        G_ZSF (SN, g, M1, M1)) ]
```

The reality of the coupling of the Z-boson to two identical neutralinos makes the vector part of the coupling vanish. So we distinguish them not by the name but by the structure of the couplings.
** REVISED: Compatible with CD+. FB**

```
let neutral_Z (n, m) =
  [ ((Neutralino n, Z, Neutralino m), FBF (1, Chibar, VA, Chi),
          (G_NZN (n, m))) ]
```

** REVISED: Compatible with CD+. FB**

```
let charged_Z c1 c2 =
  let cc1 = conj_char c1 in
  ((Chargino cc1, Z, Chargino c2), FBF ((-1), Psibar, VA , Psi),
          G_CZC (c1, c2))
```

** REVISED: Compatible with CD+. Remark: This is pure octet. FB**

```
let yukawa_v =
  [ (Gluino, Gl, Gluino), FBF (1, Chibar, V, Chi), Gs]
```

** REVISED: Independent of the sign of CD. **
** REVISED: Felix Braam: Compact version using new COMBOS + FF-Couplings

```
let yukawa_higgs_FFS f s =
    [((conjugate f, SHiggs s, f ), FBF (1, Psibar, S, Psi),
        G_YUK_FFS (conjugate f, f, s))]
let yukawa_higgs_FFP f p =
    [((conjugate f, PHiggs p, f), FBF (1, Psibar, P, Psi),
        G_YUK_FFP (conjugate f ,f , p))]
```

JR: Only the first charged Higgs.

```
let yukawa_higgs_NLC g =
  [ ((N (-g), CHiggs HC1, L g), FBF (1, Psibar, Coupling.SR, Psi),
        G_YUK_LCN g);
    ((L (-g), CHiggs HC1c, N g), FBF (1, Psibar, Coupling.SL, Psi),
        G_YUK_LCN g)]

let yukawa_higgs g =
   yukawa_higgs_NLC g @
   List.flatten ( Product.list2 yukawa_higgs_FFS [L g; U g; D g] [S1; S2; S3]) @
   List.flatten ( Product.list2 yukawa_higgs_FFP [L g; U g; D g] [P1; P2])
```

JR: Only the first charged Higgs.
** REVISED: Independent of the sign of CD. FB**

```
let yukawa_higgs_quark (g, h) =
  [ ((U (-g), CHiggs HC1, D h), FBF (1, Psibar, SLR, Psi),
        G_YUK_UCD (g, h));
    ((D (-h), CHiggs HC1c, U g), FBF (1, Psibar, SLR, Psi),
        G_YUK_DCU (g, h)) ]
```

** REVISED: Compatible with CD+.FB
** REVISED: Compact version using new COMBOS

```
let yukawa_shiggs_2 c1 c2 s =
  let cc1 = conj_char c1 in
   ((Chargino cc1, SHiggs s, Chargino c2), FBF (1, Psibar, SLR, Psi),
       G_CSC (c1, c2, s))

let yukawa_phiggs_2 c1 c2 p =
  let cc1 = conj_char c1 in
  ((Chargino cc1, PHiggs p, Chargino c2), FBF (1, Psibar, SLR, Psi),
       G_CPC (c1, c2, p))

let yukawa_higgs_2 =
  Product.list3 yukawa_shiggs_2 [C1; C2] [C1; C2] [S1; S2; S3] @
  Product.list3 yukawa_phiggs_2 [C1; C2] [C1; C2] [P1; P2]
```

JR: Only the first charged Higgs.
** REVISED: Compatible with CD+.FB **

```
let higgs_charg_neutr n c =
  let cc = conj_char c in
  [ ((Neutralino n, CHiggs HC1c, Chargino c), FBF (-1, Chibar, SLR, Psi),
              G_NHC (false, n, c));
    ((Chargino cc, CHiggs HC1, Neutralino n), FBF (-1, Psibar, SLR, Chi),
              G_NHC (true, n, c)) ]
```

** REVISED: Compatible with CD+. FB**
** REVISED: Compact version using new COMBOS

```
let shiggs_neutr (n, m, s) =
   ((Neutralino n, SHiggs s, Neutralino m), FBF (1, Chibar, SP, Chi),
       G_CICIS (n, m, s))
let phiggs_neutr (n, m, p) =
```

$((Neutralino\ n,\ PHiggs\ p,\ Neutralino\ m),\ FBF\ (1,\ Chibar,\ SP,\ Chi),$
$G\_CICIP\ (n,m,p))$

let $higgs\_neutr\ =$
$List.map\ shiggs\_neutr\ (two\_and\_one\ [N1;N2;N3;N4;N5]\ [S1;S2;S3])$ @
$List.map\ phiggs\_neutr\ (two\_and\_one\ [N1;N2;N3;N4;N5]\ [P1;P2])$

** REVISED: Compatible with CD+. FB**

let $yukawa\_n\_2\ n\ m\ g\ =$
$[\ ((Neutralino\ n,\ Slepton\ (m,-g),\ L\ g),\ FBF\ (1,\ Chibar,\ SLR,\ Psi),$
G_YUK_N (true,$L\ g,n,SL,m$));
$((L\ (-g),\ Slepton\ (m,g),\ Neutralino\ n),\ FBF\ (1,\ Psibar,\ SLR,\ Chi),$
G_YUK_N (false,$L\ g,n,SL,m$));
$((Neutralino\ n,\ Sup\ (m,-g),\ U\ g),\ FBF\ (1,\ Chibar,\ SLR,\ Psi),$
G_YUK_N (true,$U\ g,n,SU,m$));
$((U\ (-g),\ Sup\ (m,g),\ Neutralino\ n),\ FBF\ (1,\ Psibar,\ SLR,\ Chi),$
G_YUK_N (false,$U\ g,n,SU,m$));
$((Neutralino\ n,\ Sdown\ (m,-g),\ D\ g),\ FBF\ (1,\ Chibar,\ SLR,\ Psi),$
G_YUK_N (true,$D\ g,n,SD,m$));
$((D\ (-g),\ Sdown\ (m,g),\ Neutralino\ n),\ FBF\ (1,\ Psibar,\ SLR,\ Chi),$
G_YUK_N (false,$D\ g,n,SD,m$))]
let $yukawa\_n\_3\ n\ g\ =$
$[\ ((Neutralino\ n,\ Sneutrino\ (-g),\ N\ g),\ FBF\ (1,\ Chibar,\ SLR,\ Psi),$
G_YUK_N (true,$N\ g,n,SN,M1$));
$((N\ (-g),\ Sneutrino\ g,\ Neutralino\ n),\ FBF\ (1,\ Psibar,\ SLR,\ Chi),$
G_YUK_N (false,$N\ g,\ n,SN,M1$))]

let $yukawa\_n\_5\ g\ m\ =$
$[\ ((U\ (-g),\ Sup\ (m,g),\ Gluino),\ FBF\ (1,\ Psibar,\ SLR,\ Chi),$
G_YUK_G (false,$U\ g,SU,m$));
$((D\ (-g),\ Sdown\ (m,g),\ Gluino),\ FBF\ (1,\ Psibar,\ SLR,\ Chi),$
G_YUK_G (false,$D\ g,SD,m$));
$((Gluino,\ Sup\ (m,-g),\ U\ g),\ FBF\ (1,\ Chibar,\ SLR,\ Psi),$
G_YUK_G (true,$U\ g,SU,m$));
$((Gluino,\ Sdown\ (m,-g),\ D\ g),\ FBF\ (1,\ Chibar,\ SLR,\ Psi),$
G_YUK_G (true,$D\ g,SD,m$))]
let $yukawa\_n\ =$
$List.flatten\ (Product.list3\ yukawa\_n\_2\ [N1;N2;N3;N4;N5]\ [M1;M2]\ [1;2;3])$ @
$List.flatten\ (Product.list2\ yukawa\_n\_3\ [N1;N2;N3;N4;N5]\ [1;2;3])$ @
$List.flatten\ (Product.list2\ yukawa\_n\_5\ [1;2;3]\ [M1;M2])$

** REVISED: Compatible with CD+.FB **

let $yukawa\_c\_2\ c\ g\ =$
let $cc\ =\ conj\_char\ c$ in
$[\ ((L\ (-g),\ Sneutrino\ g,\ Chargino\ cc),\ BBB\ (1,\ Psibar,\ SLR,$
$Psibar),\ G\_YUK\_C$ (true,$L\ g,c,SN,M1$));
$((Chargino\ c,\ Sneutrino\ (-g),\ L\ g),\ PBP\ (1,\ Psi,\ SLR,\ Psi),$
G_YUK_C (false,$L\ g,c,SN,M1$))]
let $yukawa\_c\_3\ c\ m\ g\ =$
let $cc\ =\ conj\_char\ c$ in
$[\ ((N\ (-g),\ Slepton\ (m,g),\ Chargino\ c),\ FBF\ (1,\ Psibar,\ SLR,$

```
            Psi), G_YUK_C (true,N g, c, SL, m));
          ((Chargino cc, Slepton (m, −g), N g), FBF (1, Psibar, SLR,
            Psi), G_YUK_C (false,N g, c, SL, m)) ]
  let yukawa_c c =
    ThoList.flatmap (yukawa_c_2 c) [1; 2; 3] @
    List.flatten (Product.list2 (yukawa_c_3 c) [M1; M2] [1; 2; 3])
```

** REVISED: Compatible with CD+. FB**

```
  let yukawa_cq' c (g, h) m =
      let cc = conj_char c in
        [ ((Chargino c, Sup (m, −g), D h), PBP (1, Psi, SLR, Psi),
            G_YUK_Q (false,g, D h, c, SU, m));
          ((D (−h), Sup (m, g), Chargino cc), BBB (1, Psibar, SLR, Psibar),
            G_YUK_Q (true,g, D h, c, SU, m));
          ((Chargino cc, Sdown (m, −g), U h), FBF (1, Psibar, SLR, Psi),
            G_YUK_Q (true,g, U h, c, SD, m));
          ((U (−h), Sdown (m, g), Chargino c), FBF (1, Psibar, SLR, Psi),
            G_YUK_Q (false,g, U h, c, SD, m)) ]
  let yukawa_cq c =
   if Flags.ckm_present then
     List.flatten (Product.list2 (yukawa_cq' c) [(1, 1); (1, 2); (2, 1); (2, 2); (1, 3); (2, 3); (3, 3); (3, 2); (3, 1)] [M
   else
     List.flatten (Product.list2 (yukawa_cq' c) [(1, 1); (2, 2); (3, 3)] [M1; M2])
```

** REVISED: Compatible with CD+. Remark: Singlet and octet gluon exchange. The coupling is divided by sqrt(2) to account for the correct normalization of the Lie algebra generators. **FB

```
  let col_currents g =
    [ ((D (−g), Gl, D g), FBF ((−1), Psibar, V, Psi), Gs);
      ((U (−g), Gl, U g), FBF ((−1), Psibar, V, Psi), Gs)]
```

** REVISED: Compatible with CD+. Remark: Singlet and octet gluon exchange. The coupling is divided by sqrt(2) to account for the correct normalization of the Lie algebra generators. **FB
* LQ-coupl. **DW*

```
  let chg = function
    | M1 → M2 | M2 → M1
```

* LQ - Yuk's *

```
  let yuk_lqino_se_uc1' g1 g2 g3 m =
    let cm = chg m in
      [ ((U (−g3), Slepton (m, −g2), LQino g1), FBF (1, Psibar, SLR, Psi),
            G_LQ_EC_UC (true,cm, g1, g2, g3)) ]

  let yuk_lqino_se_uc1 g1 g2 g3 =
      ThoList.flatmap (yuk_lqino_se_uc1' g1 g2 g3) [M1; M2]

  let yuk_lqino_se_uc2' g1 g2 g3 m =
    let cm = chg m in
      [ ((LQino (−g1), Slepton (m, g2), U g3), FBF (1, Psibar, SLR, Psi),
            G_LQ_EC_UC (false,cm, g1, g2, g3)) ]
```

```
let yuk_lqino_se_uc2 g1 g2 g3 =
    ThoList.flatmap (yuk_lqino_se_uc2' g1 g2 g3) [M1; M2]

let yuk_lqino_sn_dc1 g1 g2 g3 =
  [ ((D (-g3), Sneutrino (-g2), LQino g1), FBF (-1, Psibar, SLR, Psi),
        G_LQ_EC_UC (true,M2, g1, g2, g3)) ]

let yuk_lqino_sn_dc2 g1 g2 g3 =
  [ ((LQino (-g1), Sneutrino g2, D g3), FBF (-1, Psibar, SLR, Psi),
        G_LQ_EC_UC (false,M2, g1, g2, g3)) ]

let yuk_lqino_ec_su1' g1 g2 g3 m =
  let cm = chg m in
    [ ((LQino (-g1), Sup (m, g3), L g2), FBF (1, Psibar, SLR, Psi),
          G_LQ_EC_UC (true,cm, g1, g2, g3)) ]

let yuk_lqino_ec_su1 g1 g2 g3 =
    ThoList.flatmap (yuk_lqino_ec_su1' g1 g2 g3) [M1; M2]

let yuk_lqino_ec_su2' g1 g2 g3 m =
  let cm = chg m in
    [ ((L (-g2), Sup (m, -g3), LQino (g1)), FBF (1, Psibar, SLR, Psi),
          G_LQ_EC_UC (false,cm, g1, g2, g3)) ]

let yuk_lqino_ec_su2 g1 g2 g3 =
    ThoList.flatmap (yuk_lqino_ec_su2' g1 g2 g3) [M1; M2]

let yuk_lqino_nc_sd1 g1 g2 g3 =
  [ ((LQino (-g1), Sdown (M1, g3), N g2), FBF (-1, Psibar, SLR, Psi),
        G_LQ_EC_UC (true,M2, g1, g2, g3)) ]

let yuk_lqino_nc_sd2 g1 g2 g3 =
  [ ((N (-g2), Sdown (M1, -g3), LQino (g1)), FBF (-1, Psibar, SLR, Psi),
        G_LQ_EC_UC (false,M2, g1, g2, g3)) ]

let yuk_lq_ec_uc' g1 g2 g3 m =
  [ ((L (-g2), LQ (m, g1), U (-g3)), BBB (1, Psibar, SLR, Psibar),
        G_LQ_EC_UC (false,m, g1, g2, g3)) ]

let yuk_lq_ec_uc g1 g2 g3 =
    ThoList.flatmap (yuk_lq_ec_uc' g1 g2 g3) [M1; M2]

let yuk_lq_ec_uc2' g1 g2 g3 m =
  [ ((L (g2), LQ (m, -g1), U (g3)), PBP (1, Psi, SLR, Psi),
        G_LQ_EC_UC (true,m, g1, g2, g3)) ]

let yuk_lq_ec_uc2 g1 g2 g3 =
    ThoList.flatmap (yuk_lq_ec_uc2' g1 g2 g3) [M1; M2]

let yuk_lq_nc_dc g1 g2 g3 =
  [ ((N (-g2), LQ (M2, g1), D (-g3)), BBB (-1, Psibar, SLR, Psibar),
        G_LQ_EC_UC (false,M2, g1, g2, g3)) ]

let yuk_lq_nc_dc2 g1 g2 g3 =
  [ ((N (g2), LQ (M2, -g1), D (g3)), PBP (-1, Psi, SLR, Psi),
        G_LQ_EC_UC (true,M2, g1, g2, g3)) ]
```

** Daniel Wiesler: LQ - F-Term w/ vev **

let $lq\_se\_su'\ g1\ g2\ g3\ m1\ m2\ m3\ =$
$[\ ((LQ\ (m1, g1),\ Slepton\ (m2, -g2),\ Sup\ (m3, -g3)),\ Scalar\_Scalar\_Scalar\ 1,$
$G\_LQ\_SSU\ (m1, m2, m3, g1, g2, g3))\ ]$

let $lq\_se\_su\ g1\ g2\ g3\ =$
$List.flatten\ (Product.list3\ (lq\_se\_su'\ g1\ g2\ g3)\ [M1; M2]\ [M1; M2]\ [M1; M2]\ )$

let $lq\_snu\_sd'\ g1\ g2\ g3\ m1\ m2\ =$
$[\ ((LQ\ (m1, g1),\ Sdown\ (m2, -g2),\ Sneutrino\ (-g3)),\ Scalar\_Scalar\_Scalar\ 1,$
$G\_LQ\_SSD\ (m1, m2, g1, g2, g3))\ ]$

let $lq\_snu\_sd\ g1\ g2\ g3\ =$
$List.flatten\ (Product.list2\ (lq\_snu\_sd'\ g1\ g2\ g3)\ [M1; M2]\ [M1; M2]\ )$

** Daniel Wiesler: LQ - Higgs **

let $lq\_shiggs'\ g1\ s\ g2\ m1\ m2\ =$
$[\ ((LQ\ (m1, g1),\ SHiggs\ s,\ LQ\ (m2, -g2)),\ Scalar\_Scalar\_Scalar\ 1,\ G\_LQ\_S\ (m1, m2, g1, s, g2))]$

let $lq\_shiggs\ g1\ s\ g2\ =$
$List.flatten\ (\ Product.list2\ (lq\_shiggs'\ g1\ s\ g2)\ [M1; M2]\ [M1; M2])$

let $lq\_phiggs'\ g1\ p\ g2\ m1\ m2\ =$
$[\ ((LQ\ (m1, g1),\ PHiggs\ p,\ LQ\ (m2, -g2)),\ Scalar\_Scalar\_Scalar\ 1,\ G\_LQ\_P\ (m1, m2, g1, p, g2))]$

let $lq\_phiggs\ g1\ p\ g2\ =$
$List.flatten\ (\ Product.list2\ (lq\_phiggs'\ g1\ p\ g2)\ [M1; M2]\ [M1; M2])$

let $yuk\_lqino\_shiggs\ g1\ s\ g2\ =$
$[\ ((LQino\ (-g1),\ SHiggs\ s,\ LQino\ g2),\ FBF\ (1,\ Psibar,\ SLR,\ Psi),$
$G\_YUK\_LQ\_S\ (g1, s, g2))\ ]$

let $yuk\_lqino\_phiggs\ g1\ p\ g2\ =$
$[\ ((LQino\ (-g1),\ PHiggs\ p,\ LQino\ g2),\ FBF\ (1,\ Psibar,\ SLR,\ Psi),$
$G\_YUK\_LQ\_P\ (g1, p, g2))\ ]$

** Daniel Wiesler: LQ - Neutralinos. **

let $lqino\_lq\_neu'\ n\ g1\ g2\ m\ =$
$[\ ((Neutralino\ n,\ LQ\ (m, -g1),\ LQino\ g2),\ FBF\ (1,\ Chibar,\ SLR,\ Psi),$
$G\_LQ\_NEU\ (m, g1, g2, n))\ ]$

let $lqino\_lq\_neu\ n\ g1\ g2\ =$
$ThoList.flatmap\ (lqino\_lq\_neu'\ n\ g1\ g2)\ [M1; M2]$

let $lqino\_lq\_neu2'\ n\ g1\ g2\ m\ =$
$[\ ((LQino\ (-g2),\ LQ\ (m, g1),\ Neutralino\ n),\ FBF\ (1,\ Psibar,\ SLR,\ Chi),$
$G\_LQ\_NEU\ (m, g1, g2, n))\ ]$

let $lqino\_lq\_neu2\ n\ g1\ g2\ =$
$ThoList.flatmap\ (lqino\_lq\_neu2'\ n\ g1\ g2)\ [M1; M2]$

** Daniel Wiesler: LQ-LQino-Gluino **

let $lqino\_lq\_gg'\ g1\ g2\ m\ =$
$[\ ((Gluino,\ LQ\ (m, -g1),\ LQino\ g2),\ FBF\ (1,\ Chibar,\ SLR,\ Psi),$
$G\_LQ\_GG\ (m, g1, g2))\ ]$

let $lqino\_lq\_gg\ g1\ g2\ =$

```
    ThoList.flatmap (lqino_lq_gg' g1 g2) [M1; M2]
```

** Daniel Wiesler: LQ - Gauge **

```
  let col_lqino_currents g =
    [ ((LQino (-g), Gl, LQino g), FBF ((-1), Psibar, V, Psi), Gs)]

  let neutr_lqino_current g =
    [ ((LQino (-g), Z, LQino g), FBF (1, Psibar, V, Psi), G_NLQC)]

  let col_lq_currents m g =
    [ ((Gl, LQ (m, -g), LQ (m, g)), Vector_Scalar_Scalar (-1), Gs)]

  let lq_neutr_Z g m1 m2 =
    [ ((Z, LQ (m1, -g), LQ (m2, g)), Vector_Scalar_Scalar (-1), G_ZLQ (g, m1, m2))]

  let em_lq_currents g m =
    [ ((Ga, LQ (m, -g), LQ (m, g)), Vector_Scalar_Scalar 1, Q_down)]

  let em_lqino_currents g =
    [ ((LQino (-g), Ga, LQino g), FBF (1, Psibar, V, Psi), Q_down)]

  let gluon2_lq2' g m =
    [ ((LQ (m, g), LQ (m, -g), Gl, Gl), Scalar2_Vector2 2, G_GlGlLQLQ)]
  let gluon2_lq2 g =
    ThoList.flatmap (gluon2_lq2' g) [M1; M2]

  let lq_gauge4' g m =
    [ ((Z, Z, LQ (m, g), LQ (m, -g)), Scalar2_Vector2 1, G_ZZLQLQ);
      ((Z, Ga, LQ (m, g), LQ (m, -g)), Scalar2_Vector2 1, G_ZPLQLQ);
      ((Ga, Ga, LQ (m, g), LQ (m, -g)), Scalar2_Vector2 1, G_PPLQLQ)]

  let lq_gauge4 g =
    ThoList.flatmap (lq_gauge4' g) [M1; M2]

  let lq_gg_gauge2' g m =
    [ ((Z, Gl, LQ (m, g), LQ (m, -g)), Scalar2_Vector2 1, G_ZGlLQLQ);
      ((Ga, Gl, LQ (m, g), LQ (m, -g)), Scalar2_Vector2 1, G_PGlLQLQ)]

  let lq_gg_gauge2 g =
    ThoList.flatmap (lq_gg_gauge2' g) [M1; M2]

  let col_sfermion_currents g m =
    [ ((Gl, Sup (m, -g), Sup (m, g)), Vector_Scalar_Scalar (-1), Gs);
      ((Gl, Sdown (m, -g), Sdown (m, g)), Vector_Scalar_Scalar (-1), Gs)]
```

** REVISED: Compatible with CD+. **FB

```
  let triple_gauge =
    [ ((Ga, Wm, Wp), Gauge_Gauge_Gauge 1, I_Q_W);
      ((Z, Wm, Wp), Gauge_Gauge_Gauge 1, I_G_ZWW);
      ((Gl, Gl, Gl), Gauge_Gauge_Gauge 1, I_G_S)]
```

** REVISED: Independent of the sign of CD. **FB

```
  let gauge4 = Vector4 [(2, C_13_42); (-1, C_12_34); (-1, C_14_23)]
  let minus_gauge4 = Vector4 [(-2, C_13_42); (1, C_12_34); (1, C_14_23)]
  let quartic_gauge =
    [ (Wm, Wp, Wm, Wp), gauge4, G_WWWW;
```

```
    (Wm, Z, Wp, Z), minus_gauge4, G_ZZWW;
    (Wm, Z, Wp, Ga), minus_gauge4, G_PZWW;
    (Wm, Ga, Wp, Ga), minus_gauge4, G_PPWW;
    (Gl, Gl, Gl, Gl), gauge4, G_SS]
```

The *Scalar_Vector_Vector* couplings do not depend on the choice of the sign of the covariant derivative since they are quadratic in the gauge couplings.
JR: Only the first charged Higgs.
** REVISED: Compatible with CD+. **
** Revision: 2005-03-10: first two vertices corrected. **
** REVISED: Felix Braam: Compact version using new COMBOS
** REVISED: Felix Braam: Couplings adjusted to FF-convention

```
let gauge_higgs_WPC p =
  [((Wm, CHiggs HC1, PHiggs p), Vector_Scalar_Scalar 1, G_GH_WPC p);
    ((Wp, CHiggs HC1c, PHiggs p), Vector_Scalar_Scalar 1, G_GH_WPC p)]
let gauge_higgs_WSC s =
   [((Wm, CHiggs HC1, SHiggs s), Vector_Scalar_Scalar 1, G_GH_WSC s);
    ((Wp, CHiggs HC1c, SHiggs s), Vector_Scalar_Scalar (-1), G_GH_WSC s)]
let gauge_higgs_ZSP s p =
    [((Z, SHiggs s, PHiggs p), Vector_Scalar_Scalar 1, G_GH_ZSP (s, p))]
let gauge_higgs_WWS s =
    ((SHiggs s, Wp, Wm), Scalar_Vector_Vector 1, G_GH_WWS s)
let gauge_higgs_ZZS s =
    ((SHiggs s, Z, Z), Scalar_Vector_Vector 1, G_GH_ZZS s)
let gauge_higgs_ZCC =
    ((Z, CHiggs HC1, CHiggs HC1c), Vector_Scalar_Scalar 1, G_GH_ZCC )
let gauge_higgs_GaCC =
    ((Ga, CHiggs HC1, CHiggs HC1c), Vector_Scalar_Scalar 1, G_GH_GaCC )

let gauge_higgs =
    ThoList.flatmap gauge_higgs_WPC [P1; P2] @
    ThoList.flatmap gauge_higgs_WSC [S1; S2; S3] @
    List.flatten (Product.list2 gauge_higgs_ZSP [S1; S2; S3] [P1; P2]) @
    List.map gauge_higgs_WWS [S1; S2; S3] @
    List.map gauge_higgs_ZZS [S1; S2; S3] @
    [gauge_higgs_ZCC] @ [gauge_higgs_GaCC]
```

** REVISED: Compact version using new COMBOS
** REVISED: Couplings adjusted to FF-convention

```
let gauge_higgs4_ZZPP (p1, p2) =
   ((PHiggs p1, PHiggs p2, Z, Z), Scalar2_Vector2 1, G_GH4_ZZPP (p1, p2))

let gauge_higgs4_ZZSS (s1, s2) =
    ((SHiggs s1, SHiggs s2 , Z, Z), Scalar2_Vector2 1, G_GH4_ZZSS (s1, s2))
```

JR: Only the first charged Higgs.

```
let gauge_higgs4_ZZCC =
    ((CHiggs HC1, CHiggs HC1c, Z, Z), Scalar2_Vector2 1, G_GH4_ZZCC)

let gauge_higgs4_GaGaCC =
    ((CHiggs HC1, CHiggs HC1c, Ga, Ga), Scalar2_Vector2 1, G_GH4_GaGaCC)
```

```
let gauge_higgs4_ZGaCC =
  ((CHiggs HC1, CHiggs HC1c, Ga, Z), Scalar2_Vector2 1, G_GH4_ZGaCC )

let gauge_higgs4_WWCC =
  ((CHiggs HC1, CHiggs HC1c, Wp, Wm), Scalar2_Vector2 1, G_GH4_WWCC )

let gauge_higgs4_WWPP (p1, p2) =
  ((PHiggs p1, PHiggs p2, Wp, Wm), Scalar2_Vector2 1, G_GH4_WWPP (p1, p2))

let gauge_higgs4_WWSS (s1, s2) =
  ((SHiggs s1, SHiggs s2, Wp, Wm), Scalar2_Vector2 1, G_GH4_WWSS (s1, s2))
```

JR: Only the first charged Higgs.

```
let gauge_higgs4_ZWSC s =
  [ ((CHiggs HC1, SHiggs s, Wm, Z), Scalar2_Vector2 1, G_GH4_ZWSC s);
    ((CHiggs HC1c, SHiggs s, Wp, Z), Scalar2_Vector2 1, G_GH4_ZWSC s)]

let gauge_higgs4_GaWSC s =
  [ ((CHiggs HC1, SHiggs s, Wm, Ga), Scalar2_Vector2 1, G_GH4_GaWSC s);
    ((CHiggs HC1c, SHiggs s, Wp, Ga), Scalar2_Vector2 1, G_GH4_GaWSC s) ]

let gauge_higgs4_ZWPC p =
  [ ((CHiggs HC1, PHiggs p, Wm, Z), Scalar2_Vector2 1, G_GH4_ZWPC p);
    ((CHiggs HC1c, PHiggs p, Wp, Z), Scalar2_Vector2 (-1), G_GH4_ZWPC p)]

let gauge_higgs4_GaWPC p =
  [ ((CHiggs HC1, PHiggs p, Wm, Ga), Scalar2_Vector2 1, G_GH4_GaWPC p);
    ((CHiggs HC1c, PHiggs p, Wp, Ga), Scalar2_Vector2 (-1),
        G_GH4_GaWPC p) ]

let gauge_higgs4 =
  List.map gauge_higgs4_ZZPP (pairs [P1; P2]) @
  List.map gauge_higgs4_ZZSS (pairs [S1; S2; S3]) @
  [gauge_higgs4_ZZCC] @ [gauge_higgs4_GaGaCC] @
  [gauge_higgs4_ZGaCC] @ [gauge_higgs4_WWCC] @
  List.map gauge_higgs4_WWPP (pairs [P1; P2]) @
  List.map gauge_higgs4_WWSS (pairs [S1; S2; S3]) @
  ThoList.flatmap gauge_higgs4_ZWSC [S1; S2; S3] @
  ThoList.flatmap gauge_higgs4_GaWSC [S1; S2; S3] @
  ThoList.flatmap gauge_higgs4_ZWPC [P1; P2] @
  ThoList.flatmap gauge_higgs4_GaWPC [P1; P2]
```

** Added by Felix Braam. **

```
let gauge_sfermion4' g m1 m2 =
  [ ((Wp, Wm, Slepton (m1, g), Slepton (m2, -g)), Scalar2_Vector2 1,
       G_WWSFSF (SL, g, m1, m2));
    ((Z, Ga, Slepton (m1, g), Slepton (m2, -g)), Scalar2_Vector2 1,
      G_ZPSFSF (SL, g, m1, m2));
    ((Z, Z, Slepton (m1, g), Slepton (m2, -g)), Scalar2_Vector2 1,
      G_ZZSFSF (SL, g, m1, m2));
    ((Wp, Wm, Sup (m1, g), Sup (m2, -g)), Scalar2_Vector2 1, G_WWSFSF
      (SU, g, m1, m2));
    ((Wp, Wm, Sdown (m1, g), Sdown (m2, -g)), Scalar2_Vector2 1, G_WWSFSF
      (SD, g, m1, m2));
```

```
((Z, Z, Sup (m1, g), Sup (m2, -g)), Scalar2_Vector2 1, G_ZZSFSF
    (SU, g, m1, m2));
((Z, Z, Sdown (m1, g), Sdown (m2, -g)), Scalar2_Vector2 1, G_ZZSFSF
    (SD, g, m1, m2));
((Z, Ga, Sup (m1, g), Sup (m2, -g)), Scalar2_Vector2 1, G_ZPSFSF
    (SU, g, m1, m2));
((Z, Ga, Sdown (m1, g), Sdown (m2, -g)), Scalar2_Vector2 1, G_ZPSFSF
    (SD, g, m1, m2)) ]

let gauge_sfermion4'' g m =
  [ ((Wp, Ga, Slepton (m, g), Sneutrino (-g)), Scalar2_Vector2 1,
      G_WPSLSN (false,g, m));
    ((Wm, Ga, Slepton (m, -g), Sneutrino g), Scalar2_Vector2 1,
      G_WPSLSN (true,g, m));
    ((Wp, Z, Slepton (m, g), Sneutrino (-g)), Scalar2_Vector2 1,
      G_WZSLSN (false,g, m));
    ((Wm, Z, Slepton (m, -g), Sneutrino g), Scalar2_Vector2 1,
      G_WZSLSN (true,g, m));
    ((Ga, Ga, Slepton (m, g), Slepton (m, -g)), Scalar2_Vector2 1, G_PPSFSF SL);
    ((Ga, Ga, Sup (m, g), Sup (m, -g)), Scalar2_Vector2 1, G_PPSFSF SU);
    ((Ga, Ga, Sdown (m, g), Sdown (m, -g)), Scalar2_Vector2 1, G_PPSFSF SD)]

let gauge_sfermion4 g =
  List.flatten (Product.list2 (gauge_sfermion4' g) [M1; M2] [M1; M2]) @
  ThoList.flatmap (gauge_sfermion4'' g) [M1; M2] @
  [ ((Wp, Wm, Sneutrino g, Sneutrino (-g)), Scalar2_Vector2 1, G_WWSFSF
      (SN, g, M1, M1));
    ((Z, Z, Sneutrino g, Sneutrino (-g)), Scalar2_Vector2 1, G_ZZSFSF
      (SN, g, M1, M1)) ]
```

** Modified by Felix Braam. **

```
let gauge_squark4'' g h m1 m2 =
  [ ((Wp, Ga, Sup (m1, -g), Sdown (m2, h)), Scalar2_Vector2 1, G_WPSUSD
      (false,m1, m2, g, h));
    ((Wm, Ga, Sup (m1, g), Sdown (m2, -h)), Scalar2_Vector2 1, G_WPSUSD
      (true,m1, m2, g, h));
    ((Wp, Z, Sup (m1, -g), Sdown (m2, h)), Scalar2_Vector2 1, G_WZSUSD
      (false,m1, m2, g, h));
    ((Wm, Z, Sup (m1, g), Sdown (m2, -h)), Scalar2_Vector2 1, G_WZSUSD
      (true,m1, m2, g, h)) ]
let gauge_squark4' g h = List.flatten (Product.list2 (gauge_squark4'' g h)
                                         [M1; M2] [M1; M2])
let gauge_squark4 =
  if Flags.ckm_present then
    List.flatten (Product.list2 gauge_squark4' [1; 2; 3] [1; 2; 3])
  else
    ThoList.flatmap (fun g -> gauge_squark4' g g) [1; 2; 3]

let gluon_w_squark'' g h m1 m2 =
  [ ((Gl, Wp, Sup (m1, -g), Sdown (m2, h)),
        Scalar2_Vector2 1, G_GlWSUSD (false,m1, m2, g, h));
    ((Gl, Wm, Sup (m1, g), Sdown (m2, -h)),
```

```
          Scalar2_Vector2 1, G_GlWSUSD (true,m1, m2, g, h)) ]
  let gluon_w_squark' g h =
    List.flatten (Product.list2 (gluon_w_squark'' g h) [M1; M2] [M1; M2])
  let gluon_w_squark =
    if Flags.ckm_present then
      List.flatten (Product.list2 gluon_w_squark' [1; 2; 3] [1; 2; 3])
    else
      ThoList.flatmap (fun g → gluon_w_squark' g g) [1; 2; 3]
```

** Modified by Felix Braam. **

```
  let gluon_gauge_squark' g m1 m2 =
    [ ((Gl, Z, Sup (m1, g), Sup (m2, −g)),
          Scalar2_Vector2 2, G_GlZSFSF (SU, g, m1, m2));
      ((Gl, Z, Sdown (m1, g), Sdown (m2, −g)),
          Scalar2_Vector2 2, G_GlZSFSF (SD, g, m1, m2)) ]
  let gluon_gauge_squark'' g m =
    [ ((Gl, Ga, Sup (m, g), Sup (m, −g)), Scalar2_Vector2 2, G_GlPSQSQ);
      ((Gl, Ga, Sdown (m, g), Sdown (m, −g)), Scalar2_Vector2 (−1), G_GlPSQSQ) ]
```

** Modified by Felix Braam. **

```
  let gluon_gauge_squark g =
    List.flatten (Product.list2 (gluon_gauge_squark' g) [M1; M2] [M1; M2]) @
    ThoList.flatmap (gluon_gauge_squark'' g) [M1; M2]

  let gluon2_squark2' g m =
    [ ((Gl, Gl, Sup (m, g), Sup (m, −g)), Scalar2_Vector2 2, G_GlGlSQSQ);
      ((Gl, Gl, Sdown (m, g), Sdown (m, −g)), Scalar2_Vector2 2, G_GlGlSQSQ) ]
  let gluon2_squark2 g =
    ThoList.flatmap (gluon2_squark2' g) [M1; M2]
```

JR: Only the first charged Higgs.
** REVISED: Independent of the sign of CD. **
** REVISED: Felix Braam: Compact version using new COMBOS
** REVISED: Felix Braam: Couplings adjusted to FF-convention

```
  let higgs_SCC s =
     ((CHiggs HC1, CHiggs HC1c, SHiggs s), Scalar_Scalar_Scalar 1,
        G_H3_SCC s )
  let higgs_SSS (s1, s2, s3) =
      ((SHiggs s1, SHiggs s2, SHiggs s3), Scalar_Scalar_Scalar 1,
        G_H3_SSS (s1, s2, s3))
  let higgs_SPP (p1, p2, s) =
      ((SHiggs s, PHiggs p1, PHiggs p2), Scalar_Scalar_Scalar 1,
        G_H3_SPP (s, p1, p2))

  let higgs =
     List.map higgs_SCC [S1; S2; S3]@
     List.map higgs_SSS (triples [S1; S2; S3])@
     List.map higgs_SPP (two_and_one [P1; P2] [S1; S2; S3])

  let higgs4 = []
```

(∗ The vertices of the type Higgs - Sfermion - Sfermion are independent of the choice of the CD sign since they are quadratic in the gauge coupling. ∗)

JR: Only the first charged Higgs.
** REVISED: Independent of the sign of CD. **

```
let higgs_sneutrino' s g =
    ((SHiggs s, Sneutrino g, Sneutrino (-g)), Scalar_Scalar_Scalar 1,
        G_SFSFS (s, SN, g, M1, M1))
  let higgs_sneutrino'' g m =
    [((CHiggs HC1, Sneutrino (-g), Slepton (m, g)),
          Scalar_Scalar_Scalar 1, G_HSNSL (false, g, m));
     ((CHiggs HC1c, Sneutrino g, Slepton (m, -g)), Scalar_Scalar_Scalar 1,
          G_HSNSL (true, g, m))]
  let higgs_sneutrino =
    Product.list2 higgs_sneutrino' [S1; S2; S3] [1; 2; 3] @
    List.flatten ( Product.list2 higgs_sneutrino'' [1; 2; 3] [M1; M2] )
```

Under the assumption that there is no mixing between the left- and right-handed sfermions for the first two generations there is only a coupling of the form Higgs - sfermion1 - sfermion2 for the third generation. All the others are suppressed by m_f/M_W.
** REVISED: Independent of the sign of CD. **

```
  let higgs_sfermion_S s g m1 m2 =
    [ ((SHiggs s, Slepton (m1, g), Slepton (m2, -g)), Scalar_Scalar_Scalar 1,
          G_SFSFS (s, SL, g, m1, m2));
      ((SHiggs s, Sup (m1, g), Sup (m2, -g)), Scalar_Scalar_Scalar 1,
          G_SFSFS (s, SU, g, m1, m2));
      ((SHiggs s, Sdown (m1, g), Sdown (m2, -g)), Scalar_Scalar_Scalar 1,
          G_SFSFS (s, SD, g, m1, m2))]

let higgs_sfermion' g m1 m2 =
      (higgs_sfermion_S S1 g m1 m2) @ (higgs_sfermion_S S2 g m1 m2) @ (higgs_sfermion_S S3 g m1

let higgs_sfermion_P p g m1 m2 =
    [ ((PHiggs p, Slepton (m1, g), Slepton (m2, -g)), Scalar_Scalar_Scalar 1,
          G_SFSFP (p, SL, g, m1, m2));
      ((PHiggs p, Sup (m1, g), Sup (m2, -g)), Scalar_Scalar_Scalar 1,
          G_SFSFP (p, SU, g, m1, m2));
      ((PHiggs p, Sdown (m1, g), Sdown (m2, -g)), Scalar_Scalar_Scalar 1,
          G_SFSFP (p, SD, g, m1, m2)) ]

let higgs_sfermion'' g m1 m2 =
      (higgs_sfermion_P P1 g m1 m2) @ (higgs_sfermion_P P2 g m1 m2)
let higgs_sfermion = List.flatten (Product.list3 higgs_sfermion' [1; 2; 3] [M1; M2] [M1; M2]) @
    List.flatten (Product.list3 higgs_sfermion'' [1; 2; 3] [M1; M2] [M1; M2])
```

JR: Only the first charged Higgs.
** REVISED: Independent of the sign of CD. **

```
let higgs_squark' g h m1 m2 =
  [ ((CHiggs HC1, Sup (m1, -g), Sdown (m2, h)), Scalar_Scalar_Scalar 1,
          G_HSUSD (false, m1, m2, g, h));
    ((CHiggs HC1c, Sup (m1, g), Sdown (m2, -h)), Scalar_Scalar_Scalar 1,
          G_HSUSD (true, m1, m2, g, h)) ]
let higgs_squark_a g h = higgs_squark' g h M1 M1
```

```
let higgs_squark_b (g, h) = List.flatten (Product.list2 (higgs_squark' g h)
                                [M1; M2] [M1; M2])
let higgs_squark =
  if Flags.ckm_present then
    List.flatten (Product.list2 higgs_squark_a [1; 2] [1; 2]) @
    ThoList.flatmap higgs_squark_b [(1, 3); (2, 3); (3, 3); (3, 1); (3, 2)]
  else
    higgs_squark_a 1 1 @ higgs_squark_a 2 2 @ higgs_squark_b (3, 3)

let vertices3 =
    (ThoList.flatmap electromagnetic_currents_3 [1; 2; 3] @
     ThoList.flatmap electromagnetic_currents_2 [C1; C2] @
     List.flatten (Product.list2 electromagnetic_sfermion_currents [1; 2; 3]
                     [M1; M2]) @
     ThoList.flatmap neutral_currents [1; 2; 3] @
     ThoList.flatmap neutral_sfermion_currents [1; 2; 3] @
     ThoList.flatmap charged_currents [1; 2; 3] @
     List.flatten (Product.list2 charged_slepton_currents [1; 2; 3]
                     [M1; M2]) @
     (if Flags.ckm_present then
        List.flatten (Product.list2 charged_quark_currents [1; 2; 3]
                        [1; 2; 3]) @
        List.flatten (Product.list2 charged_squark_currents [1; 2; 3]
                        [1; 2; 3]) @
        ThoList.flatmap yukawa_higgs_quark [(1, 3); (2, 3); (3, 3); (3, 1); (3, 2)]
     else
        charged_quark_currents 1 1 @
        charged_quark_currents 2 2 @
        charged_quark_currents 3 3 @
        charged_squark_currents 1 1 @
        charged_squark_currents 2 2 @
        charged_squark_currents 3 3 @
        ThoList.flatmap yukawa_higgs_quark [(3, 3)]) @
     yukawa_higgs 3 @ yukawa_n @
     ThoList.flatmap yukawa_c [C1; C2] @
     ThoList.flatmap yukawa_cq [C1; C2] @
     List.flatten (Product.list2 charged_chargino_currents [N1; N2; N3; N4; N5]
                     [C1; C2]) @ triple_gauge @
     ThoList.flatmap neutral_Z (pairs [N1; N2; N3; N4; N5]) @
     Product.list2 charged_Z [C1; C2] [C1; C2] @
     gauge_higgs @ higgs @ yukawa_higgs_2 @
     List.flatten (Product.list2 higgs_charg_neutr [N1; N2; N3; N4; N5] [C1; C2]) @
     higgs_neutr @ higgs_sneutrino @ higgs_sfermion @
     higgs_squark @ yukawa_v @
     ThoList.flatmap col_currents [1; 2; 3] @
     List.flatten (Product.list2 col_sfermion_currents [1; 2; 3] [M1; M2])) @
     List.flatten (Product.list2 col_lq_currents [M1; M2] [1; 2; 3]) @
     ThoList.flatmap col_lqino_currents [1; 2; 3] @
     ThoList.flatmap em_lqino_currents [1; 2; 3] @
     ThoList.flatmap neutr_lqino_current [1; 2; 3] @
```

```
      List.flatten (Product.list3 yuk_lqino_se_uc1 [1; 2; 3] [1; 2; 3] [1; 2; 3]) @
      List.flatten (Product.list3 yuk_lqino_se_uc2 [1; 2; 3] [1; 2; 3] [1; 2; 3]) @
      List.flatten (Product.list3 yuk_lqino_ec_su1 [1; 2; 3] [1; 2; 3] [1; 2; 3]) @
      List.flatten (Product.list3 yuk_lqino_ec_su2 [1; 2; 3] [1; 2; 3] [1; 2; 3]) @
      List.flatten (Product.list3 yuk_lqino_sn_dc1 [1; 2; 3] [1; 2; 3] [1; 2; 3]) @
      List.flatten (Product.list3 yuk_lqino_sn_dc2 [1; 2; 3] [1; 2; 3] [1; 2; 3]) @
      List.flatten (Product.list3 yuk_lqino_nc_sd1 [1; 2; 3] [1; 2; 3] [1; 2; 3]) @
      List.flatten (Product.list3 yuk_lqino_nc_sd2 [1; 2; 3] [1; 2; 3] [1; 2; 3]) @
      List.flatten (Product.list3 yuk_lq_ec_uc [1; 2; 3] [1; 2; 3] [1; 2; 3]) @
      List.flatten (Product.list3 yuk_lq_ec_uc2 [1; 2; 3] [1; 2; 3] [1; 2; 3]) @
      List.flatten (Product.list3 yuk_lq_nc_dc [1; 2; 3] [1; 2; 3] [1; 2; 3]) @
      List.flatten (Product.list3 yuk_lq_nc_dc2 [1; 2; 3] [1; 2; 3] [1; 2; 3]) @
      List.flatten (Product.list3 lq_neutr_Z [1; 2; 3] [M1; M2] [M1; M2]) @
      List.flatten (Product.list2 em_lq_currents [1; 2; 3] [M1; M2]) @
      List.flatten (Product.list3 lq_shiggs [1; 2; 3] [S1; S2; S3; S4; S5; S6; S7; S8; S9] [1; 2; 3]) @
      List.flatten (Product.list3 lq_phiggs [1; 2; 3] [P1; P2; P3; P4; P5; P6; P7] [1; 2; 3]) @
      List.flatten (Product.list3 yuk_lqino_shiggs [1; 2; 3] [S1; S2; S3; S4; S5; S6; S7; S8; S9] [1; 2; 3]) @
      List.flatten (Product.list3 yuk_lqino_phiggs [1; 2; 3] [P1; P2; P3; P4; P5; P6; P7] [1; 2; 3]) @
      List.flatten (Product.list3 lqino_lq_neu nlist [1; 2; 3] [1; 2; 3]) @
      List.flatten (Product.list3 lqino_lq_neu2 nlist [1; 2; 3] [1; 2; 3]) @
      List.flatten (Product.list3 lq_se_su [1; 2; 3] [1; 2; 3] [1; 2; 3]) @
      List.flatten (Product.list3 lq_snu_sd [1; 2; 3] [1; 2; 3] [1; 2; 3]) @
      List.flatten (Product.list2 lqino_lq_gg [1; 2; 3] [1; 2; 3])

  let vertices4 =
    (quartic_gauge @ higgs4 @ gauge_higgs4 @
     ThoList.flatmap gauge_sfermion4 [1; 2; 3] @
     gauge_squark4 @ gluon_w_squark @
     ThoList.flatmap gluon2_squark2 [1; 2; 3] @
     ThoList.flatmap gluon_gauge_squark [1; 2; 3] @
     ThoList.flatmap gluon2_lq2 [1; 2; 3] @
     ThoList.flatmap lq_gauge4 [1; 2; 3] @
     ThoList.flatmap lq_gg_gauge2 [1; 2; 3])

  let vertices () = (vertices3, vertices4, [])

  let table = F.of_vertices (vertices ())
  let fuse2 = F.fuse2 table
  let fuse3 = F.fuse3 table
  let fuse = F.fuse table
  let max_degree () = 4
```

SLHA2-Nomenclature for neutral Higgses

```
  let flavor_of_string s =
    match s with
        | "e-" → L 1 | "e+" → L (−1)
        | "mu-" → L 2 | "mu+" → L (−2)
        | "tau-" → L 3 | "tau+" → L (−3)
        | "nue" → N 1 | "nuebar" → N (−1)
        | "numu" → N 2 | "numubar" → N (−2)
        | "nutau" → N 3 | "nutaubar" → N (−3)
        | "se1-" → Slepton (M1, 1) | "se1+" → Slepton (M1, −1)
```

```
| "smu1-" → Slepton (M1, 2) | "smu1+" → Slepton (M1, −2)
| "stau1-" → Slepton (M1, 3) | "stau1+" → Slepton (M1, −3)
| "se2-" → Slepton (M2, 1) | "se2+" → Slepton (M2, −1)
| "smu2-" → Slepton (M2, 2) | "smu2+" → Slepton (M2, −2)
| "stau2-" → Slepton (M2, 3) | "stau2+" → Slepton (M2, −3)
| "snue" → Sneutrino 1 | "snue*" → Sneutrino (−1)
| "snumu" → Sneutrino 2 | "snumu*" → Sneutrino (−2)
| "snutau" → Sneutrino 3 | "snutau*" → Sneutrino (−3)
| "u" → U 1 | "ubar" → U (−1)
| "c" → U 2 | "cbar" → U (−2)
| "t" → U 3 | "tbar" → U (−3)
| "d" → D 1 | "dbar" → D (−1)
| "s" → D 2 | "sbar" → D (−2)
| "b" → D 3 | "bbar" → D (−3)
| "A" → Ga | "Z" | "Z0" → Z
| "W+" → Wp | "W-" → Wm
| "gl" | "g" → Gl
| "h01" → SHiggs S1 | "h02" → SHiggs S2 | "h03" →
SHiggs S3
| "A01" → PHiggs P1 | "A02" → PHiggs P2
| "h04" → SHiggs S4 | "h05" → SHiggs S5 | "h06" →
SHiggs S6
| "A03" → PHiggs P3 | "A04" → PHiggs P4
| "h07" → SHiggs S7 | "h08" → SHiggs S8 | "h09" →
SHiggs S9
| "A05" → PHiggs P5 | "A06" → PHiggs P6 | "A07" →
PHiggs P7
(* JR: Only the first charged Higgs. *)
| "H+" → CHiggs HC1 | "H-" → CHiggs HC1c
| "su1" → Sup (M1, 1) | "su1c" → Sup (M1, −1)
| "sc1" → Sup (M1, 2) | "sc1c" → Sup (M1, −2)
| "st1" → Sup (M1, 3) | "st1c" → Sup (M1, −3)
| "su2" → Sup (M2, 1) | "su2c" → Sup (M2, −1)
| "sc2" → Sup (M2, 2) | "sc2c" → Sup (M2, −2)
| "st2" → Sup (M2, 3) | "st2c" → Sup (M2, −3)
| "sgl" | "sg" → Gluino
| "sd1" → Sdown (M1, 1) | "sd1c" → Sdown (M1, −1)
| "ss1" → Sdown (M1, 2) | "ss1c" → Sdown (M1, −2)
| "sb1" → Sdown (M1, 3) | "sb1c" → Sdown (M1, −3)
| "sd2" → Sdown (M2, 1) | "sd2c" → Sdown (M2, −1)
| "ss2" → Sdown (M2, 2) | "ss2c" → Sdown (M2, −2)
| "sb2" → Sdown (M2, 3) | "sb2c" → Sdown (M2, −3)
| "neu1" → Neutralino N1 | "neu2" → Neutralino N2
| "neu3" → Neutralino N3 | "neu4" → Neutralino N4
| "neu5" → Neutralino N5 | "neu6" → Neutralino N6
| "neu7" → Neutralino N7 | "neu8" → Neutralino N8
| "neu9" → Neutralino N9 | "neu10" → Neutralino N10
| "neu11" → Neutralino N11
| "ch1+" → Chargino C1 | "ch2+" → Chargino C2
| "ch1-" → Chargino C1c | "ch2-" → Chargino C2c
```

```
        | "ch3+" → Chargino C3 | "ch4+" → Chargino C4
        | "ch3-" → Chargino C3c | "ch4-" → Chargino C4c
        | "lq11" → LQ (M1, 1) | "lq11c" → LQ (M1, −1)
        | "lq12" → LQ (M2, 1) | "lq12c" → LQ (M2, −1)
        | "lq21" → LQ (M1, 2) | "lq21c" → LQ (M1, −2)
        | "lq22" → LQ (M2, 2) | "lq22c" → LQ (M2, −2)
        | "lq31" → LQ (M1, 3) | "lq31c" → LQ (M1, −3)
        | "lq32" → LQ (M2, 3) | "lq32c" → LQ (M2, −3)
        | "lqino1" → LQino 1 | "lqino1b" → LQino (−1)
        | "lqino2" → LQino 2 | "lqino2b" → LQino (−2)
        | "lqino3" → LQino 3 | "lqino3b" → LQino (−3)
        | s → invalid_arg ("HUBABUBA:␣%s␣Modellib_PSSSM.ExtMSSM.flavor_of_string:" ^ s)

  let flavor_to_string = function
    | L 1 → "e-" | L (−1) → "e+"
    | L 2 → "mu-" | L (−2) → "mu+"
    | L 3 → "tau-" | L (−3) → "tau+"
    | N 1 → "nue" | N (−1) → "nuebar"
    | N 2 → "numu" | N (−2) → "numubar"
    | N 3 → "nutau" | N (−3) → "nutaubar"
    | U 1 → "u" | U (−1) → "ubar"
    | U 2 → "c" | U (−2) → "cbar"
    | U 3 → "t" | U (−3) → "tbar"
    | U _ → invalid_arg
        "Modellib_PSSSM.ExtMSSM.flavor_to_string:␣invalid␣up␣type␣quark"
    | D 1 → "d" | D (−1) → "dbar"
    | D 2 → "s" | D (−2) → "sbar"
    | D 3 → "b" | D (−3) → "bbar"
    | D _ → invalid_arg
        "Modellib_PSSSM.ExtMSSM.flavor_to_string:␣invalid␣down␣type␣quark"
    | Gl → "gl" | Gluino → "sgl"
    | Ga → "A" | Z → "Z"
    | Wp → "W+" | Wm → "W-"
    | SHiggs S1 → "h01" | SHiggs S2 → "h02" | SHiggs S3 → "h03"
    | PHiggs P1 → "A01" | PHiggs P2 → "A02"
    | SHiggs S4 → "h04" | SHiggs S5 → "h05" | SHiggs S6 → "h06"
    | PHiggs P3 → "A03" | PHiggs P4 → "A04"
    | SHiggs S7 → "h07" | SHiggs S8 → "h08" | SHiggs S9 → "h09"
    | PHiggs P5 → "A05" | PHiggs P6 → "A06" | PHiggs P7 → "A07"
(∗ JR: Only the first charged Higgs. ∗)
    | CHiggs HC1 → "H+" | CHiggs HC1c → "H-"
    | CHiggs HC2 → "HX_1+" | CHiggs HC2c → "HX_1-"
    | CHiggs HC3 → "HX_2+" | CHiggs HC3c → "HX_2-"
    | CHiggs HC4 → "HX_3+" | CHiggs HC4c → "HX_3-"
    | CHiggs HC5 → "HX_4+" | CHiggs HC5c → "HX_4-"
    | Slepton (M1, 1) → "se1-" | Slepton (M1, −1) → "se1+"
    | Slepton (M1, 2) → "smu1-" | Slepton (M1, −2) → "smu1+"
    | Slepton (M1, 3) → "stau1-" | Slepton (M1, −3) → "stau1+"
    | Slepton (M2, 1) → "se2-" | Slepton (M2, −1) → "se2+"
    | Slepton (M2, 2) → "smu2-" | Slepton (M2, −2) → "smu2+"
```

```
| Slepton (M2, 3) → "stau2-" | Slepton (M2, −3) → "stau2+"
| Sneutrino 1 → "snue" | Sneutrino (−1) → "snue*"
| Sneutrino 2 → "snumu" | Sneutrino (−2) → "snumu*"
| Sneutrino 3 → "snutau" | Sneutrino (−3) → "snutau*"
| Sup (M1, 1) → "su1" | Sup (M1, −1) → "su1c"
| Sup (M1, 2) → "sc1" | Sup (M1, −2) → "sc1c"
| Sup (M1, 3) → "st1" | Sup (M1, −3) → "st1c"
| Sup (M2, 1) → "su2" | Sup (M2, −1) → "su2c"
| Sup (M2, 2) → "sc2" | Sup (M2, −2) → "sc2c"
| Sup (M2, 3) → "st2" | Sup (M2, −3) → "st2c"
| Sdown (M1, 1) → "sd1" | Sdown (M1, −1) → "sd1c"
| Sdown (M1, 2) → "ss1" | Sdown (M1, −2) → "ss1c"
| Sdown (M1, 3) → "sb1" | Sdown (M1, −3) → "sb1c"
| Sdown (M2, 1) → "sd2" | Sdown (M2, −1) → "sd2c"
| Sdown (M2, 2) → "ss2" | Sdown (M2, −2) → "ss2c"
| Sdown (M2, 3) → "sb2" | Sdown (M2, −3) → "sb2c"
| Neutralino n → "neu" ^ string_of_neu n
| Chargino C1 → "ch1+" | Chargino C1c → "ch1-"
| Chargino C2 → "ch2+" | Chargino C2c → "ch2-"
| Chargino C3 → "ch3+" | Chargino C3c → "ch3-"
| Chargino C4 → "ch4+" | Chargino C4c → "ch4-"
| LQ (M1, 1) → "lq11" | LQ (M1, −1) → "lq11c"
| LQ (M2, 1) → "lq12" | LQ (M2, −1) → "lq12c"
| LQ (M1, 2) → "lq21" | LQ (M1, −2) → "lq21c"
| LQ (M2, 2) → "lq22" | LQ (M2, −2) → "lq22c"
| LQ (M1, 3) → "lq31" | LQ (M1, −3) → "lq31c"
| LQ (M2, 3) → "lq32" | LQ (M2, −3) → "lq32c"
| LQino 1 → "lqino1" | LQino (−1) → "lqino1b"
| LQino 2 → "lqino2" | LQino (−2) → "lqino2b"
| LQino 3 → "lqino3" | LQino (−3) → "lqino3b"
| _ → invalid_arg "Modellib_PSSSM.ExtMSSM.flavor_to_string"

let flavor_to_TeX = function
| L 1 → "e^-" | L (−1) → "e^+"
| L 2 → "\\mu^-" | L (−2) → "\\mu^+"
| L 3 → "\\tau^-" | L (−3) → "\\tau^+"
| N 1 → "\\nu_e" | N (−1) → "\\bar{\\nu}_e"
| N 2 → "\\nu_\\mu" | N (−2) → "\\bar{\\nu}_\\mu"
| N 3 → "\\nu_\\tau" | N (−3) → "\\bar{\\nu}_\\tau"
| U 1 → "u" | U (−1) → "\\bar{u}"
| U 2 → "c" | U (−2) → "\\bar{c}"
| U 3 → "t" | U (−3) → "\\bar{t}"
| D 1 → "d" | D (−1) → "\\bar{d}"
| D 2 → "s" | D (−2) → "\\bar{s}"
| D 3 → "b" | D (−3) → "\\bar{b}"
| L _ → invalid_arg
    "Modellib_PSSSM.ExtMSSM.flavor_to_TeX:␣invalid␣lepton"
| N _ → invalid_arg
    "Modellib_PSSSM.ExtMSSM.flavor_to_TeX:␣invalid␣neutrino"
| U _ → invalid_arg
```

```
      "Modellib_PSSSM.ExtMSSM.flavor_to_TeX:␣invalid␣up␣type␣quark"
  | D _ → invalid_arg
      "Modellib_PSSSM.ExtMSSM.flavor_to_TeX:␣invalid␣down␣type␣quark"
  | Gl → "g" | Gluino → "\\widetilde{g}"
  | Ga → "\\gamma" | Z → "Z" | Wp → "W^+" | Wm → "W^-"
  | SHiggs S1 → "S_1" | SHiggs S2 → "S_2" | SHiggs S3 → "S_3"
  | SHiggs S4 → "S_4" | SHiggs S5 → "S_5" | SHiggs S6 → "S_6"
  | SHiggs S7 → "S_7" | SHiggs S8 → "S_8" | SHiggs S9 → "S_9"
  | PHiggs P1 → "P_1" | PHiggs P2 → "P_2" | PHiggs P3 → "P_3"
  | PHiggs P4 → "P_4" | PHiggs P5 → "P_5" | PHiggs P6 → "P_6"
  | PHiggs P7 → "P_7"
  | CHiggs HC1 → "H^+" | CHiggs HC1c → "H^-"
  | CHiggs HC2 → "X_{H,1}^+" | CHiggs HC2c → "X_{H,1}^-"
  | CHiggs HC3 → "X_{H,2}^+" | CHiggs HC3c → "X_{H,2}^-"
  | CHiggs HC4 → "X_{H,3}^+" | CHiggs HC4c → "X_{H,3}^-"
  | CHiggs HC5 → "X_{H,4}^+" | CHiggs HC5c → "X_{H,4}^-"
  | Slepton (M1, 1) → "\\widetilde{e}_1^-"
  | Slepton (M1, −1) → "\\widetilde{e}_1^+"
  | Slepton (M1, 2) → "\\widetilde{\\mu}_1^-"
  | Slepton (M1, −2) → "\\widetilde{\\mu}_1^+"
  | Slepton (M1, 3) → "\\widetilde{\\tau}_1^-"
  | Slepton (M1, −3) → "\\widetilde{\\tau}_1^+"
  | Slepton (M2, 1) → "\\widetilde{e}_2^-"
  | Slepton (M2, −1) → "\\widetilde{e}_2^+"
  | Slepton (M2, 2) → "\\widetilde{\\mu}_2^-"
  | Slepton (M2, −2) → "\\widetilde{\\mu}_2^+"
  | Slepton (M2, 3) → "\\widetilde{\\tau}_2^-"
  | Slepton (M2, −3) → "\\widetilde{\\tau}_2^+"
  | Sneutrino 1 → "\\widetilde{\\nu}_e"
  | Sneutrino (−1) → "\\widetilde{\\nu}_e^*"
  | Sneutrino 2 → "\\widetilde{\\nu}_\\mu"
  | Sneutrino (−2) → "\\widetilde{\\nu}_\\mu^*"
  | Sneutrino 3 → "\\widetilde{\\nu}_\\tau"
  | Sneutrino (−3) → "\\widetilde{\\nu}_\\tau^*"
  | Sup (M1, 1) → "\\widetilde{u}_1"
  | Sup (M1, −1) → "\\widetilde{u}_1^*"
  | Sup (M1, 2) → "\\widetilde{c}_1"
  | Sup (M1, −2) → "\\widetilde{c}_1^*"
  | Sup (M1, 3) → "\\widetilde{t}_1"
  | Sup (M1, −3) → "\\widetilde{t}_1^*"
  | Sup (M2, 1) → "\\widetilde{u}_2"
  | Sup (M2, −1) → "\\widetilde{u}_2^*"
  | Sup (M2, 2) → "\\widetilde{c}_2"
  | Sup (M2, −2) → "\\widetilde{c}_2^*"
  | Sup (M2, 3) → "\\widetilde{t}_2"
  | Sup (M2, −3) → "\\widetilde{t}_2^*"
  | Sdown (M1, 1) → "\\widetilde{d}_1"
  | Sdown (M1, −1) → "\\widetilde{d}_1^*"
  | Sdown (M1, 2) → "\\widetilde{s}_1"
  | Sdown (M1, −2) → "\\widetilde{s}_1^*"
```

```
| Sdown (M1, 3) → "\\widetilde{b}_1"
| Sdown (M1, −3) → "\\widetilde{b}_1^*"
| Sdown (M2, 1) → "\\widetilde{d}_2"
| Sdown (M2, −1) → "\\widetilde{d}_2^*"
| Sdown (M2, 2) → "\\widetilde{s}_2"
| Sdown (M2, −2) → "\\widetilde{s}_2^*"
| Sdown (M2, 3) → "\\widetilde{b}_2"
| Sdown (M2, −3) → "\\widetilde{b}_2^*"
| Neutralino N1 → "\\widetilde{\\chi}^0_1"
| Neutralino N2 → "\\widetilde{\\chi}^0_2"
| Neutralino N3 → "\\widetilde{\\chi}^0_3"
| Neutralino N4 → "\\widetilde{\\chi}^0_4"
| Neutralino N5 → "\\widetilde{\\chi}^0_5"
| Neutralino N6 → "\\widetilde{\\chi}^0_6"
| Neutralino N7 → "\\widetilde{\\chi}^0_7"
| Neutralino N8 → "\\widetilde{\\chi}^0_8"
| Neutralino N9 → "\\widetilde{\\chi}^0_9"
| Neutralino N10 → "\\widetilde{\\chi}^0_{10}"
| Neutralino N11 → "\\widetilde{\\chi}^0_{11}"
| Slepton _ → invalid_arg
    "Modellib_PSSSM.ExtMSSM.flavor_to_TeX: invalid slepton"
| Sneutrino _ → invalid_arg
    "Modellib_PSSSM.ExtMSSM.flavor_to_TeX: invalid sneutrino"
| Sup _ → invalid_arg
    "Modellib_PSSSM.ExtMSSM.flavor_to_TeX: invalid up type squark"
| Sdown _ → invalid_arg
    "Modellib_PSSSM.ExtMSSM.flavor_to_TeX: invalid down type squark"
| Chargino C1 → "\\widetilde{\\chi}_1^+"
| Chargino C1c → "\\widetilde{\\chi}_1^-"
| Chargino C2 → "\\widetilde{\\chi}_2^+"
| Chargino C2c → "\\widetilde{\\chi}_2^-"
| Chargino C3 → "\\widetilde{\\chi}_3^+"
| Chargino C3c → "\\widetilde{\\chi}_3^-"
| Chargino C4 → "\\widetilde{\\chi}_4^+"
| Chargino C4c → "\\widetilde{\\chi}_4^-"
| LQ (M1, 1) → "D_{1,,1}" | LQ (M1, −1) → "D_{1,,1}^*"
| LQ (M2, 1) → "D_{1,,2}" | LQ (M2, −1) → "D_{1,,2}^*"
| LQ (M1, 2) → "D_{2,,1}" | LQ (M1, −2) → "D_{2,,1}^*"
| LQ (M2, 2) → "D_{2,,2}" | LQ (M2, −2) → "D_{2,,2}^*"
| LQ (M1, 3) → "D_{3,,1}" | LQ (M1, −3) → "D_{3,,1}^*"
| LQ (M2, 3) → "D_{3,,2}" | LQ (M2, −3) → "D_{3,,2}^*"
| LQino 1 → "\\widetilde{D}_1" | LQino (−1) → "\\bar\\widetilde{D}_1"
| LQino 2 → "\\widetilde{D}_2" | LQino (−2) → "\\bar\\widetilde{D}_2"
| LQino 3 → "\\widetilde{D}_3" | LQino (−3) → "\\bar\\widetilde{D}_3"
| LQ _ → invalid_arg
    "Modellib_PSSSM.ExtMSSM.flavor_to_TeX: invalid leptoquark type"
| LQino _ → invalid_arg
    "Modellib_PSSSM.ExtMSSM.flavor_to_TeX: invalid leptoquarkino type"

let flavor_symbol = function
```

```
| L g when g > 0 → "l" ^ string_of_int g
| L g → "l" ^ string_of_int (abs g) ^ "b"
| N g when g > 0 → "n" ^ string_of_int g
| N g → "n" ^ string_of_int (abs g) ^ "b"
| U g when g > 0 → "u" ^ string_of_int g
| U g → "u" ^ string_of_int (abs g) ^ "b"
| D g when g > 0 → "d" ^ string_of_int g
| D g → "d" ^ string_of_int (abs g) ^ "b"
| Gl → "gl"
| Ga → "a" | Z → "z"
| Wp → "wp" | Wm → "wm"
| Slepton (M1, g) when g > 0 → "sl1" ^ string_of_int g
| Slepton (M1, g) → "sl1c" ^ string_of_int (abs g)
| Slepton (M2, g) when g > 0 → "sl2" ^ string_of_int g
| Slepton (M2, g) → "sl2c" ^ string_of_int (abs g)
| Sneutrino g when g > 0 → "sn" ^ string_of_int g
| Sneutrino g → "snc" ^ string_of_int (abs g)
| Sup (M1, g) when g > 0 → "su1" ^ string_of_int g
| Sup (M1, g) → "su1c" ^ string_of_int (abs g)
| Sup (M2, g) when g > 0 → "su2" ^ string_of_int g
| Sup (M2, g) → "su2c" ^ string_of_int (abs g)
| Sdown (M1, g) when g > 0 → "sd1" ^ string_of_int g
| Sdown (M1, g) → "sd1c" ^ string_of_int (abs g)
| Sdown (M2, g) when g > 0 → "sd2" ^ string_of_int g
| Sdown (M2, g) → "sd2c" ^ string_of_int (abs g)
| Neutralino n → "neu" ^ (string_of_neu n)
| Chargino c when (int_of_char c) > 0 → "cp" ^ string_of_char c
| Chargino c → "cm" ^ string_of_int (abs (int_of_char c))
| Gluino → "sgl"
| SHiggs s → "h0" ^ (string_of_shiggs s)
| PHiggs p → "A0" ^ (string_of_phiggs p)
| CHiggs HC1 → "hp" | CHiggs HC1c → "hm"
| CHiggs _ → invalid_arg "charged␣Higgs␣not␣yet␣implemented"
| LQ (M1, g) when g > 0 → "lq" ^ string_of_int g ^ "1"
| LQ (M1, g) → "lq" ^ string_of_int (abs g) ^ "1c"
| LQ (M2, g) when g > 0 → "lq" ^ string_of_int g ^ "2"
| LQ (M2, g) → "lq" ^ string_of_int (abs g) ^ "2c"
| LQino g when g > 0 → "lqino" ^ string_of_int g
| LQino g → "lqino" ^ string_of_int (abs g) ^ "b"

let pdg = function
| L g when g > 0 → 9 + 2 × g
| L g → - 9 + 2 × g
| N g when g > 0 → 10 + 2 × g
| N g → - 10 + 2 × g
| U g when g > 0 → 2 × g
| U g → 2 × g
| D g when g > 0 → - 1 + 2 × g
| D g → 1 + 2 × g
| Gl → 21
```

$|$ Ga $\rightarrow$ 22 $|$ Z $\rightarrow$ 23
$|$ Wp $\rightarrow$ 24 $|$ Wm $\rightarrow$ (-24)
$|$ $SHiggs\ S1$ $\rightarrow$ 25 $|$ $SHiggs\ S2$ $\rightarrow$ 35 $|$ $PHiggs\ P1$ $\rightarrow$ 36
(* JR: Only the first charged Higgs. *)
$|$ $CHiggs\ HC1$ $\rightarrow$ 37 $|$ $CHiggs\ HC1c$ $\rightarrow$ (-37)
$|$ $CHiggs$ _ $\rightarrow$ $invalid\_arg$ `"charged␣Higgs␣not␣yet␣implemented"`
$|$ $Slepton\ (M1, g)$ when $g > 0 \rightarrow 1000009 + 2 \times g$
$|$ $Slepton\ (M1, g) \rightarrow -1000009 + 2 \times g$
$|$ $Slepton\ (M2, g)$ when $g > 0 \rightarrow 2000009 + 2 \times g$
$|$ $Slepton\ (M2, g) \rightarrow -2000009 + 2 \times g$
$|$ $Sneutrino\ g$ when $g > 0 \rightarrow 1000010 + 2 \times g$
$|$ $Sneutrino\ g \rightarrow -1000010 + 2 \times g$
$|$ $Sup\ (M1, g)$ when $g > 0 \rightarrow 1000000 + 2 \times g$
$|$ $Sup\ (M1, g) \rightarrow -1000000 + 2 \times g$
$|$ $Sup\ (M2, g)$ when $g > 0 \rightarrow 2000000 + 2 \times g$
$|$ $Sup\ (M2, g) \rightarrow -2000000 + 2 \times g$
$|$ $Sdown\ (M1, g)$ when $g > 0 \rightarrow 999999 + 2 \times g$
$|$ $Sdown\ (M1, g) \rightarrow -999999 + 2 \times g$
$|$ $Sdown\ (M2, g)$ when $g > 0 \rightarrow 1999999 + 2 \times g$
$|$ $Sdown\ (M2, g) \rightarrow -1999999 + 2 \times g$
$|$ $Gluino \rightarrow 1000021$
(* JR: only the first two charginos. *)
$|$ $Chargino\ C1 \rightarrow 1000024$ $|$ $Chargino\ C1c \rightarrow (-1000024)$
$|$ $Chargino\ C2 \rightarrow 1000037$ $|$ $Chargino\ C2c \rightarrow (-1000037)$
$|$ $Chargino\ C3 \rightarrow 1000039$ $|$ $Chargino\ C3c \rightarrow (-1000039)$
$|$ $Chargino\ C4 \rightarrow 1000041$ $|$ $Chargino\ C4c \rightarrow (-1000041)$
$|$ $Neutralino\ N1 \rightarrow 1000022$ $|$ $Neutralino\ N2 \rightarrow 1000023$
$|$ $Neutralino\ N3 \rightarrow 1000025$ $|$ $Neutralino\ N4 \rightarrow 1000035$
(* According to SLHA2 (not anymore ?!?)*)
$|$ $Neutralino\ N5 \rightarrow 1000045$ $|$ $Neutralino\ N6 \rightarrow 1000046$
$|$ $Neutralino\ N7 \rightarrow 1000047$ $|$ $Neutralino\ N8 \rightarrow 1000048$
$|$ $Neutralino\ N9 \rightarrow 1000049$ $|$ $Neutralino\ N10 \rightarrow 1000050$
$|$ $Neutralino\ N11 \rightarrow 1000051$
$|$ $PHiggs\ P2 \rightarrow 46$ $|$ $PHiggs\ P3 \rightarrow 47$ $|$ $PHiggs\ P4 \rightarrow 48$
$|$ $PHiggs\ P5 \rightarrow 49$ $|$ $PHiggs\ P6 \rightarrow 50$ $|$ $PHiggs\ P7 \rightarrow 51$
$|$ $SHiggs\ S3 \rightarrow 45$ $|$ $SHiggs\ S4 \rightarrow 52$ $|$ $SHiggs\ S5 \rightarrow 53$
$|$ $SHiggs\ S6 \rightarrow 54$ $|$ $SHiggs\ S7 \rightarrow 55$ $|$ $SHiggs\ S8 \rightarrow 56$
$|$ $SHiggs\ S9 \rightarrow 57$
$|$ $LQ\ (M1, g)$ when $g > 0 \rightarrow 1000059 + g$
$|$ $LQ\ (M1, g) \rightarrow -1000059 + g$
$|$ $LQ\ (M2, g)$ when $g > 0 \rightarrow 2000059 + g$
$|$ $LQ\ (M2, g) \rightarrow -2000059 + g$
$|$ $LQino\ g$ when $g > 0 \rightarrow 59 + g$
$|$ $LQino\ g \rightarrow -59 + g$

We must take care of the pdg numbers for the two different kinds of sfermions in the MSSM. The particle data group in its Monte Carlo particle numbering scheme takes only into account mixtures of the third generation squarks and the stau. For the other sfermions we will use the number of the lefthanded field for the lighter mixed state and the one for the righthanded for the heavier. Below are the official pdg numbers from the Particle Data Group. In order not to produce arrays with some million entries in the Fortran code for the masses and the widths we introduce our private pdg numbering scheme which only extends not too far beyond 42. Our private scheme then has the following pdf numbers (for the sparticles the subscripts L/R and $1/2$ are taken synonymously):

d	down-quark	1
u	up-quark	2
s	strange-quark	3
c	charm-quark	4
b	bottom-quark	5
t	top-quark	6
e^-	electron	11
ν_e	electron-neutrino	12
μ^-	muon	13
ν_μ	muon-neutrino	14
τ^-	tau	15
ν_τ	tau-neutrino	16
g	gluon	(9) 21
γ	photon	22
Z^0	Z-boson	23
W^+	W-boson	24
h^0	light Higgs boson	25
H^0	heavy Higgs boson	35
A^0	pseudoscalar Higgs	36
H^+	charged Higgs	37
$\tilde{d}_L$	down-squark 1	41
$\tilde{u}_L$	up-squark 1	42
$\tilde{s}_L$	strange-squark 1	43
$\tilde{c}_L$	charm-squark 1	44
$\tilde{b}_L$	bottom-squark 1	45
$\tilde{t}_L$	top-squark 1	46
$\tilde{d}_R$	down-squark 2	47
$\tilde{u}_R$	up-squark 2	48
$\tilde{s}_R$	strange-squark 2	49
$\tilde{c}_R$	charm-squark 2	50
$\tilde{b}_R$	bottom-squark 2	51
$\tilde{t}_R$	top-squark 2	52
$\tilde{e}_L$	selectron 1	53
$\tilde{\nu}_{e,L}$	electron-sneutrino	54
$\tilde{\mu}_L$	smuon 1	55
$\tilde{\nu}_{\mu,L}$	muon-sneutrino	56
$\tilde{\tau}_L$	stau 1	57
$\tilde{\nu}_{\tau,L}$	tau-sneutrino	58
$\tilde{e}_R$	selectron 2	59
$\tilde{\mu}_R$	smuon 2	61
$\tilde{\tau}_R$	stau 2	63
$\tilde{g}$	gluino	64
$\tilde{\chi}_1^0$	neutralino 1	65
$\tilde{\chi}_2^0$	neutralino 2	66
$\tilde{\chi}_3^0$	neutralino 3	67
$\tilde{\chi}_4^0$	neutralino 4	68
$\tilde{\chi}_4^0$	neutralino 5	69
$\tilde{\chi}_1^+$	chargino 1	70

```
let pdg_mw = function
    | L g when g > 0 → 9 + 2 × g
    | L g → − 9 + 2 × g
    | N g when g > 0 → 10 + 2 × g
    | N g → − 10 + 2 × g
    | U g when g > 0 → 2 × g
    | U g → 2 × g
    | D g when g > 0 → − 1 + 2 × g
    | D g → 1 + 2 × g
    | Gl → 21
    | Ga → 22 | Z → 23
    | Wp → 24 | Wm → (−24)
    | SHiggs S1 → 25 | SHiggs S2 → 35 | PHiggs P1 → 36
(* JR: Only the first charged Higgs. *)
    | CHiggs HC1 → 37 | CHiggs HC1c → (−37)
    | CHiggs _ → invalid_arg "charged␣Higgs␣not␣yet␣implemented"
    | Sup (M1, g) when g > 0 → 40 + 2 × g
    | Sup (M1, g) → − 40 + 2 × g
    | Sup (M2, g) when g > 0 → 46 + 2 × g
    | Sup (M2, g) → − 46 + 2 × g
    | Sdown (M1, g) when g > 0 → 39 + 2 × g
    | Sdown (M1, g) → − 39 + 2 × g
    | Sdown (M2, g) when g > 0 → 45 + 2 × g
    | Sdown (M2, g) → − 45 + 2 × g
    | Slepton (M1, g) when g > 0 → 51 + 2 × g
    | Slepton (M1, g) → − 51 + 2 × g
    | Slepton (M2, g) when g > 0 → 57 + 2 × g
    | Slepton (M2, g) → − 57 + 2 × g
    | Sneutrino g when g > 0 → 52 + 2 × g
    | Sneutrino g → − 52 + 2 × g
    | Gluino → 64
(* JR: Only the first two charginos. *)
    | Chargino C1 → 70 | Chargino C1c → (−70)
    | Chargino C2 → 71 | Chargino C2c → (−71)
    | Chargino C3 → 106 | Chargino C3c → (−106)
    | Chargino C4 → 107 | Chargino C4c → (−107)
    | Neutralino N1 → 65 | Neutralino N2 → 66
    | Neutralino N3 → 67 | Neutralino N4 → 68
    | Neutralino N5 → 69 | Neutralino N6 → 100
    | Neutralino N7 → 101 | Neutralino N8 → 102
    | Neutralino N9 → 103 | Neutralino N10 → 104
    | Neutralino N11 → 105
    | PHiggs P2 → 72 | PHiggs P3 → 89 | PHiggs P4 → 90
    | PHiggs P5 → 91 | PHiggs P6 → 92 | PHiggs P7 → 93
    | SHiggs S3 → 73 | SHiggs S4 → 94 | SHiggs S5 → 95
    | SHiggs S6 → 96 | SHiggs S7 → 97 | SHiggs S8 → 98
    | SHiggs S9 → 99
    | LQ (M1, g) when g > 0 → 78 + 2 × g
    | LQ (M1, g) → − 78 + 2 × g
    | LQ (M2, g) when g > 0 → 79 + 2 × g
```

```
  | LQ (M2, g) → − 79 + 2 × g
  | LQino g when g > 0 → 85 + g
  | LQino g → − 85 + g

let mass_symbol f =
  "mass(" ^ string_of_int (abs (pdg_mw f)) ^ ")"

let width_symbol f =
  "width(" ^ string_of_int (abs (pdg_mw f)) ^ ")"

let conj_symbol = function
  | false, str → str
  | true, str → str ^ "_c"

let constant_symbol = function
  | E → "e" | G → "g" | G_Z → "gz"
  | Q_lepton → "qlep" | Q_up → "qup" | Q_down → "qdwn"
  | Q_charg → "qchar"
  | G_NC_lepton → "gnclep" | G_NC_neutrino → "gncneu"
  | G_NC_up → "gncup" | G_NC_down → "gncdwn"
  | G_CC → "gcc"
  | G_CCQ (vc, g1, g2) → conj_symbol (vc, "g_ccq" ) ^ "("
      ^ string_of_int g1 ^ "," ^ string_of_int g2 ^ ")"
  | I_Q_W → "iqw" | I_G_ZWW → "igzww"
  | G_WWWW → "gw4" | G_ZZWW → "gzzww"
  | G_PZWW → "gpzww" | G_PPWW → "gppww"
  | G_GH4_ZZPP (p1, p2) → "g_ZZA0A0(" ^ string_of_phiggs p1 ^ "," ^
      string_of_phiggs p2 ^ ")"
  | G_GH4_ZZSS (s1, s2) → "g_ZZh0h0(" ^ string_of_shiggs s1 ^ "," ^
      string_of_shiggs s2 ^ ")"
  | G_GH4_ZZCC → "g_zzhphm"
  | G_GH4_GaGaCC → "g_AAhphm"
  | G_GH4_ZGaCC → "g_zAhphm"
  | G_GH4_WWCC → "g_wwhphm"
  | G_GH4_WWPP (p1, p2) → "g_WWA0A0(" ^ string_of_phiggs p1 ^ "," ^
      string_of_phiggs p2 ^ ")"
  | G_GH4_WWSS (s1, s2) → "g_WWh0h0(" ^ string_of_shiggs s1 ^ "," ^
      string_of_shiggs s2 ^ ")"
  | G_GH4_ZWSC s → "g_ZWhph0(" ^ string_of_shiggs s ^")"
  | G_GH4_GaWSC s → "g_AWhph0(" ^ string_of_shiggs s ^")"
  | G_GH4_ZWPC p → "g_ZWhpA0(" ^ string_of_phiggs p ^")"
  | G_GH4_GaWPC p → "g_AWhpA0(" ^ string_of_phiggs p ^")"
  | G_CICIS (n1, n2, s) → "g_neuneuh0(" ^ string_of_neu n1 ^ "," ^
      string_of_neu n2 ^ "," ^ string_of_shiggs s ^ ")"
  | G_CICIP (n1, n2, p) → "g_neuneuA0(" ^ string_of_neu n1 ^ "," ^
      string_of_neu n2 ^ "," ^ string_of_phiggs p ^ ")"
  | G_H3_SCC s → "g_h0hphm(" ^ string_of_shiggs s ^ ")"
  | G_H3_SPP (s, p1, p2) → "g_h0A0A0(" ^ string_of_shiggs s ^ "," ^
      string_of_phiggs p1 ^ "," ^ string_of_phiggs p2 ^ ")"
  | G_H3_SSS (s1, s2, s3) → "g_h0h0h0(" ^ string_of_shiggs s1 ^ "," ^
      string_of_shiggs s2 ^ "," ^ string_of_shiggs s3 ^ ")"
  | G_CSC (c1, c2, s) → "g_chchh0(" ^ string_of_char c1 ^ "," ^
```

```
      string_of_char c2 ^ "," ^ string_of_shiggs s ^ ")"
  | G_CPC (c1, c2, p) → "g_chchA0(" ^ string_of_char c1 ^ "," ^
      string_of_char c2 ^ "," ^ string_of_phiggs p ^")"
  | G_YUK_FFS (f1, f2, s) → "g_yuk_h0_" ^ string_of_fermion_type f1 ^
      string_of_fermion_type f2 ^ "(" ^ string_of_shiggs s ^ "," ^
      string_of_fermion_gen f1 ^ ")"
  | G_YUK_FFP (f1, f2, p) → "g_yuk_A0_" ^ string_of_fermion_type f1 ^
      string_of_fermion_type f2 ^ "(" ^ string_of_phiggs p ^ "," ^
      string_of_fermion_gen f1 ^ ")"
  | G_YUK_LCN g → "g_yuk_hp_ln(" ^ string_of_int g ^ ")"
  | G_NWC (n, c) → "g_nwc(" ^ string_of_char c ^ "," ^ string_of_neu n ^ ")"
  | G_CWN (c, n) → "g_cwn(" ^ string_of_char c ^ "," ^ string_of_neu n ^ ")"
  | G_SLSNW (vc, g, m) → conj_symbol (vc, "g_wslsn") ^ "(" ^ string_of_int g
      ^ "," ^ string_of_sfm m ^ ")"
  | G_NZN (n1, n2) → "g_zneuneu(" ^ string_of_neu n1 ^ ","
      ^ string_of_neu n2 ^ ")"
  | G_CZC (c1, c2) → "g_zchch(" ^ string_of_char c1 ^ ","
      ^ string_of_char c2 ^ ")"
  | Gs → "gs"
  | G_YUK_UCD (n, m) → "g_yuk_hp_ud(" ^ string_of_int n ^ "," ^
      string_of_int m ^ ")"
  | G_YUK_DCU (n, m) → "g_yuk_hm_du(" ^ string_of_int n ^ "," ^
      string_of_int m ^ ")"
  | G_YUK_N (vc, f, n, sf, m) → conj_symbol (vc, "g_yuk_neu_" ^
      string_of_fermion_type f ^ string_of_sff sf) ^ "(" ^
      string_of_fermion_gen f ^ "," ^ string_of_neu n ^ "," ^
      string_of_sfm m ^ ")"
  | G_YUK_G (vc, f, sf, m) → conj_symbol (vc, "g_yuk_gluino_" ^
      string_of_fermion_type f ^ string_of_sff sf) ^ "(" ^
      string_of_fermion_gen f ^ "," ^ string_of_sfm m ^ ")"
  | G_YUK_C (vc, f, c, sf, m) → conj_symbol (vc, "g_yuk_char_" ^
      string_of_fermion_type f ^ string_of_sff sf) ^ "(" ^
      string_of_fermion_gen f ^ "," ^ string_of_char c ^ "," ^
      string_of_sfm m ^ ")"
  | G_YUK_Q (vc, g1, f, c, sf, m) → conj_symbol (vc, "g_yuk_char_" ^
      string_of_fermion_type f ^ string_of_sff sf) ^"("^string_of_int g1 ^
      "," ^ string_of_fermion_gen f ^ "," ^ string_of_char c ^ "," ^
      string_of_sfm m ^ ")"
  | G_WPSUSD (vc, m1, m2, g1, g2) → conj_symbol (vc, "g_wA_susd") ^ "(" ^
      string_of_int g1 ^ "," ^ string_of_int g2 ^ "," ^ string_of_sfm m1 ^
      "," ^ string_of_sfm m2 ^ ")"
  | G_WZSUSD (vc, m1, m2, g1, g2) → conj_symbol (vc, "g_wz_susd") ^ "(" ^
      string_of_int g1 ^ "," ^ string_of_int g2 ^ "," ^ string_of_sfm m1
      ^ "," ^ string_of_sfm m2 ^ ")"
  (* 3vertex: Higgs-Gauge a la Franke-Fraas *)
```

Nomenclature consistent with *flavor_of_string*

```
  | G_GH_ZSP (s, p) → "g_zh0a0(" ^ string_of_shiggs s ^ "," ^
      string_of_phiggs p ^ ")"
  | G_GH_WSC s → "g_Whph0(" ^ string_of_shiggs s ^ ")"
```

```
| G_GH_WPC p → "g_WhpA0(" ^ string_of_phiggs p^ ")"
| G_GH_ZZS s → "g_ZZh0(" ^ string_of_shiggs s ^ ")"
| G_GH_WWS s → "g_WWh0(" ^ string_of_shiggs s ^ ")"
| G_GH_ZCC → "g_Zhmhp"
| G_GH_GaCC → "g_Ahmhp"
| G_ZSF (f, g, m1, m2) → "g_z" ^ string_of_sff f ^ string_of_sff f ^ "(" ^
    string_of_int g ^ "," ^ string_of_sfm m1 ^ "," ^ string_of_sfm m2 ^ ")"
| G_HSNSL (vc, g, m) → conj_symbol (vc, "g_hp_sl" ^ string_of_sfm m ^ "sn1" )
    ^ "(" ^ string_of_int g ^ ")"
| G_GlGlSQSQ → "g_gg_sqsq"
| G_PPSFSF f → "g_AA_" ^ string_of_sff f ^ string_of_sff f
| G_ZZSFSF (f, g, m1, m2) → "g_zz_" ^ string_of_sff f ^string_of_sff f ^ "(" ^
    string_of_int g ^ "," ^ string_of_sfm m1 ^ "," ^ string_of_sfm m2 ^ ")"
| G_ZPSFSF (f, g, m1, m2) → "g_zA_" ^ string_of_sff f ^string_of_sff f ^ "("
    ^ string_of_int g ^","^ string_of_sfm m1 ^ "," ^ string_of_sfm m2 ^ ")"
| G_GlPSQSQ → "g_gA_sqsq"
| G_GlZSFSF (f, g, m1, m2) → "g_gz_" ^ string_of_sff f ^ string_of_sff f ^
    "(" ^ string_of_int g ^ "," ^ string_of_sfm m1 ^ "," ^ string_of_sfm
    m2 ^ ")"
| G_GlWSUSD (vc, m1, m2, g1, g2) → conj_symbol (vc, "g_gw_susd") ^ "(" ^
    string_of_int g1 ^ "," ^string_of_int g2 ^ "," ^ string_of_sfm m1 ^ ","
    ^ string_of_sfm m2 ^ ")"
| G_strong → "gs" | G_SS → "gs**2"
| I_G_S → "igs"
| G_NHC (vc, n, c) → conj_symbol(vc,"g_neuhmchar") ^ "(" ^
    string_of_neu n ^ "," ^ string_of_char c ^ ")"
| G_WWSFSF (f, g, m1, m2) → "g_ww_" ^ string_of_sff f ^ string_of_sff f ^
    "(" ^ string_of_int g ^ "," ^ string_of_sfm m1 ^ "," ^ string_of_sfm m2
    ^ ")"
| G_WPSLSN (vc, g, m) → conj_symbol (vc, "g_wA_slsn") ^"("^ string_of_int g
    ^ "," ^ string_of_sfm m ^ ")"
| G_WZSLSN (vc, g, m) → conj_symbol (vc, "g_wz_slsn") ^ "(" ^ string_of_int
    g ^ "," ^ string_of_sfm m ^ ")"
| G_SFSFS (s, f, g, m1, m2) → "g_h0_"^ string_of_sff f ^ string_of_sfm m1
    ^ string_of_sff f ^ string_of_sfm m2 ^ "(" ^ string_of_shiggs s ^ "," ^
    string_of_int g ^ ")"
| G_SFSFP (p, f, g, m1, m2) → "g_A0_"^ string_of_sff f ^ string_of_sfm m1
    ^ string_of_sff f ^ string_of_sfm m2 ^ "(" ^ string_of_phiggs p ^ "," ^
    string_of_int g ^ ")"
| G_HSUSD (vc, m1, m2, g1, g2) → conj_symbol (vc, "g_hp_su" ^ string_of_sfm m1
    ^ "sd" ^ string_of_sfm m2 ) ^ "(" ^ string_of_int g1 ^ ","
    ^ string_of_int g2 ^ ")"
| G_WSQ (vc, g1, g2, m1, m2) → conj_symbol (vc, "g_wsusd") ^ "("
    ^ string_of_int g1 ^ "," ^ string_of_int g2 ^ "," ^ string_of_sfm m1 ^
    "," ^ string_of_sfm m2 ^ ")"
| G_YUK_LQ_S (g1, s, g3) → "g_yuk_lq_s(" ^ string_of_int g1 ^ "," ^
    string_of_shiggs s ^"," ^ string_of_int g3 ^")"
| G_YUK_LQ_P (g1, p, g3) → "g_yuk_lq_p(" ^ string_of_int g1 ^ "," ^
    string_of_phiggs p ^ "," ^ string_of_int g3 ^ ")"
| G_LQ_NEU (m, g1, g2, n) → "g_lq_neu(" ^ string_of_sfm m ^ "," ^
```

```
      string_of_int g1 ^ "," ^ string_of_int g2 ^ "," ^ string_of_neu n ^ ")"
  | G_LQ_GG (m, g1, g2) → "g_lq_gg(" ^ string_of_sfm m ^ "," ^
      string_of_int g1 ^ "," ^ string_of_int g2 ^ ")"
  | G_LQ_EC_UC (vc, m, g1, g2, g3) → conj_symbol(vc,"g_lq_ec_uc") ^ "(" ^
      string_of_sfm m ^ "," ^ string_of_int g1 ^ "," ^ string_of_int g2 ^ ","
      ^ string_of_int g3 ^ ")"
  | G_LQ_SSU (m1, m2, m3, g1, g2, g3) → "g_lq_sst(" ^ string_of_sfm m1 ^ "," ^
      string_of_sfm m2 ^ "," ^ string_of_sfm m3 ^ "," ^ string_of_int g1 ^ "," ^
      string_of_int g2 ^ "," ^ string_of_int g3 ^ ")"
  | G_LQ_SSD (m1, m2, g1, g2, g3) → "g_lq_ssta(" ^ string_of_sfm m1 ^ "," ^
      string_of_sfm m2 ^ "," ^ string_of_int g1 ^ "," ^ string_of_int g2 ^
      "," ^ string_of_int g3 ^ ")"
  | G_LQ_S (m1, m2, g1, s, g2) → "g_lq_s(" ^ string_of_sfm m1 ^ "," ^ string_of_sfm m2 ^ ","
      ^ string_of_int g1 ^ "," ^ string_of_shiggs s ^ "," ^ string_of_int g2 ^ ")"
  | G_LQ_P (m1, m2, g1, p, g2) → "g_lq_s(" ^ string_of_sfm m1 ^ "," ^ string_of_sfm m2 ^ ","
      ^ string_of_int g1 ^ "," ^ string_of_phiggs p ^ "," ^ string_of_int g2 ^ ")"
  | G_ZLQ (g, m1, m2) → "g_zlqlq(" ^ string_of_int g ^ "," ^
      string_of_sfm m1 ^ "," ^ string_of_sfm m2 ^ ")"
  | G_ZZLQLQ → "g_zz_lqlq"
  | G_ZPLQLQ → "g_zA_lqlq"
  | G_PPLQLQ → "g_AA_lqlq"
  | G_ZGlLQLQ → "g_zg_lqlq"
  | G_PGlLQLQ → "g_Ag_lqlq"
  | G_GlGlLQLQ → "g_gg_lqlq"
  | G_NLQC → "g_nlqc"

end
```

—14—
Comphep Models

14.1 Interface of Comphep_syntax

```
type raw =
  | I | Integer of int | Symbol of string
  | Application of string × raw
  | Dotproduct of raw × raw
  | Product of (raw × int) list
  | Sum of (raw × int) list

val symbol : string → raw
val integer : int → raw
val imag : raw

val apply : string → raw → raw
val dot : raw → raw → raw
val multiply : raw → raw → raw
val divide : raw → raw → raw
val power : raw → int → raw
val add : raw → raw → raw
val subtract : raw → raw → raw
val neg : raw → raw
```

14.2 Implementation of Comphep_syntax

```
type raw =
  | I | Integer of int | Symbol of string
  | Application of string × raw
  | Dotproduct of raw × raw
  | Product of (raw × int) list
  | Sum of (raw × int) list

let symbol name = Symbol name
let integer n = Integer n
let imag = I

let apply f x = Application (f, x)
let dot x y = Dotproduct (x, y)
```

```
let negate = List.map (fun (x, c) -> (x, - c))
let scale n = List.map (fun (x, c) -> (x, n × c))

let add1 (x, c) y =
  if c = 0 then
    y
  else
    try
      let c' = List.assoc x y + c in
      if c' = 0 then
        List.remove_assoc x y
      else
        (x, c') :: (List.remove_assoc x y)
    with
    | Not_found -> (x, c) :: y

let addn = List.fold_right add1

let multiply x y =
  match x, y with
  | Product x', Product y' -> Product (addn x' y')
  | Integer n, Product y' -> Product (scale n y')
  | Product x', Integer n -> Product (scale n x')
  | _, Product y' -> Product (add1 (x, 1) y')
  | Product x', _ -> Product (add1 (y, 1) x')
  | _ when x = y -> Product ([(x, 2)])
  | _ -> Product ([(x, 1); (y, 1)])

let divide x y =
  match y with
  | Product y' -> multiply x (Product (negate y'))
  | _ when x = y -> Product ([])
  | _ -> Product ([(x, 1); (y, - 1)])

let power x n =
  match x with
  | Product x' -> Product (scale n x')
  | x -> Product ([(x, n)])

let add x y =
  match x, y with
  | Sum x', Sum y' -> Sum (addn x' y')
  | _, Sum y' -> Sum (add1 (x, 1) y')
  | Sum x', _ -> Sum (add1 (y, 1) x')
  | _ when x = y -> Sum ([(x, 2)])
  | _ -> Sum ([(x, 1); (y, 1)])

let subtract x y =
  match y with
  | Sum y' -> add x (Sum (negate y'))
  | _ when x = y -> Sum ([])
  | _ -> Sum ([(x, 1); (y, - 1)])

let neg = function
```

```
  | Sum x → Sum (negate x)
  | x → Sum ([(x, −1)])

type vector =
  | Momentum of int
  | Index of int
  | Index′ of int

let vector_keyword = function
  | "p1" → Some (Momentum 1)
  | "p2" → Some (Momentum 2)
  | "p3" → Some (Momentum 3)
  | "p4" → Some (Momentum 4)
  | "m1" → Some (Index 1)
  | "m2" → Some (Index 2)
  | "m3" → Some (Index 3)
  | "m4" → Some (Index 4)
  | "M1" → Some (Index′ 1)
  | "M2" → Some (Index′ 2)
  | "M3" → Some (Index′ 3)
  | "M4" → Some (Index′ 4)
  | _ → None
```

14.3 Lexer

```
{
open Comphep_parser
}

let digit = ['0'−'9']
let upper = ['A'−'Z']
let lower = ['a'−'z']
let alpha = upper | lower
let alphanum = alpha | digit

let symbol = alpha alphanum*
let integer = digit+

rule token = parse
    [' ' '\t'] { token lexbuf } (* skip blanks *)
  | "(" { LPAREN }
  | ")" { RPAREN }
  | "i" { I }
  | "." { DOT }
  | "**" { POWER }
  | "*" { MULT }
  | "/" { DIV }
  | "+" { PLUS }
  | "-" { MINUS }
  | symbol { SYMBOL (Lexing.lexeme lexbuf) }
  | integer { INT (int_of_string (Lexing.lexeme lexbuf)) }
```

```
| _ { failwith ("lexer fails @" ^ Lexing.lexeme lexbuf) }
| eof { END }
```

14.4 Parser

Header

module S = *Comphep_syntax*

Token declarations

```
%token < string > SYMBOL
%token < int > INT
%token I
%token LPAREN RPAREN
%token DOT MULT DIV POWER PLUS MINUS
%token END

%left PLUS MINUS
%left MULT DIV
%nonassoc UNARY
%nonassoc POWER
%nonassoc DOT

%start expr
%type < Comphep_syntax.raw > expr
```

Grammar rules

```
expr ::=
    e END { $1 }

e ::=
    SYMBOL { S.symbol $1 }
  | INT { S.integer $1 }
  | I { S.imag }
  | SYMBOL LPAREN e RPAREN { S.apply $1 $3 }
  | LPAREN e RPAREN { $2 }
  | e DOT e { S.dot $1 $3 }
  | e MULT e { S.multiply $1 $3 }
  | e DIV e { S.divide $1 $3 }
  | e PLUS e { S.add $1 $3 }
  | e MINUS e { S.subtract $1 $3 }
```

```
  | PLUS e %prec UNARY { $2 }
  | MINUS e %prec UNARY { S.neg $2 }
  | e POWER INT { S.power $1 $3 }
```

14.5 Interface of Comphep

Wolfgang's idea: read Comphep's model files:

```
module Model : Model.T
```

14.6 Implementation of Comphep

```
let rcs_file = RCS.parse "Comphep" ["Plagiarizing␣CompHEP␣models␣..."]
    { RCS.revision = "$Revision:␣6465␣$";
      RCS.date = "$Date:␣2015-01-10␣16:22:31␣+0100␣(Sat,␣10␣Jan␣2015)␣$";
      RCS.author = "$Author:␣jr_reuter␣$";
      RCS.source
        = "$URL:␣svn+ssh://jr_reuter@login.hepforge.org/hepforge/svn/whizard/trunk/omega/sr
```

A friendlier *String.sub* that returns an empty string instead of raising an exception. Instead of the length, the second argument denotes the last position.

```
let substring buffer i1 i2 =
  let imax = String.length buffer − 1 in
  let i1 = max i1 0
  and i2 = min i2 imax in
  let len = i2 − i1 + 1 in
  if len > 0 then
    String.sub buffer i1 len
  else
    ""

let first_non_white buffer =
  let len = String.length buffer in
  let rec skip_white i =
    if i ≥ len then
      i
    else if buffer.[i] ≠ ' ' ∧ buffer.[i] ≠ '\t' then
      i
    else
      skip_white (succ i) in
  skip_white 0

let last_non_white buffer =
  let len = String.length buffer in
  let rec skip_white i =
    if i < 0 then
      i
    else if buffer.[i] ≠ ' ' ∧ buffer.[i] ≠ '\t' then
      i
```

```
    else
      skip_white (pred i) in
  skip_white (pred len)

let gobble_white buffer =
  substring buffer (first_non_white buffer) (last_non_white buffer)

let gobble_arrows buffer =
  let imax = String.length buffer − 1 in
  if imax ≥ 0 then
    gobble_white
      (substring buffer
         (if buffer.[0] = '>' then 1 else 0)
         (if buffer.[imax] = '<' then pred imax else imax))
  else
    ""

let fold_lines ic f init =
  let rec fold_lines' acc =
    let continue =
      try
        let acc' = f (input_line ic) acc in
        fun () → fold_lines' acc'
      with
      | End_of_file → fun () → acc in
    continue () in
  fold_lines' init

let column_tabs line =
  let len = String.length line in
  let rec tabs' acc i =
    if i ≥ len then
      List.rev acc
    else if line.[i] = '|' then
      tabs' (i :: acc) (succ i)
    else
      tabs' acc (succ i)
  in
  tabs' [] 0

let columns tabs line =
  let imax = String.length line − 1 in
  let rec columns' acc i = function
    | [] → List.rev_map gobble_white (substring line i imax :: acc)
    | tab :: rest →
        if tab < i then
          invalid_arg "columns: clash"
        else if (match rest with [] → false | _ → true)
            ∧ line.[tab] ≠ '|' then
          invalid_arg "columns: expecting '|'"
        else
          columns' (substring line i (pred tab) :: acc) (succ tab) rest
  in
```

```
  columns' [] 0 tabs

let input_table name =
  let ic = open_in name in
  let model = input_line ic in
  let table = input_line ic in
  let line = input_line ic in
  let tabs = column_tabs line in
  let titles = columns tabs line in
  let rows = fold_lines ic (fun line acc →
    if String.length line > 0 ∧ line.[0] = '=' then
      acc
    else
      columns tabs line :: acc) [] in
  close_in ic;
  (gobble_white model, gobble_white table, List.map gobble_arrows titles, rows)

let substitute_char (cold, cnew) s =
  for i = 0 to String.length s − 1 do
    if s.[i] = cold then
      s.[i] ← cnew
  done;
  s

let sanitize_symbol s =
  List.fold_right substitute_char [('+', 'p'); ('-', 'm')] (String.copy s)
```

Fodder for a future *Coupling* module ...

```
let rec fermion_of_lorentz = function
  | Coupling.Spinor → 1
  | Coupling.ConjSpinor → −1
  | Coupling.Majorana → 1
  | Coupling.Maj_Ghost → 1
  | Coupling.Vectorspinor → 1
  | Coupling.Vector | Coupling.Massive_Vector → 0
  | Coupling.Scalar | Coupling.Tensor_1 | Coupling.Tensor_2 → 0
  | Coupling.BRS f → fermion_of_lorentz f

let rec conjugate_lorentz = function
  | Coupling.Spinor → Coupling.ConjSpinor
  | Coupling.ConjSpinor → Coupling.Spinor
  | Coupling.BRS f → Coupling.BRS (conjugate_lorentz f)
  | f → f
```

Currently, this operates on the sanitized symbol names.

```
let pdg_heuristic name =
  match name with
  | "e1" → 11 | "E1" → −11 | "n1" → 12 | "N1" → −12
  | "e2" → 13 | "E2" → −13 | "n2" → 14 | "N2" → −14
```

```
  | "e3" → 15 | "E3" → − 15 | "n3" → 16 | "N3" → − 16
  | "u" → 2 | "U" → − 2 | "d" → 1 | "D" → − 1
  | "c" → 4 | "C" → − 4 | "s" → 3 | "S" → − 3
  | "t" → 6 | "T" → − 6 | "b" → 5 | "B" → − 5
  | "G" → 21 | "A" → 22 | "Z" → 23
  | "Wp" → 24 | "Wm" → − 24 | "H" → 25
  | _ → invalid_arg ("pdg_heuristic failed: " ^ name)

module Model =
  struct

    type flavor = int
    type constant = string
    type gauge = unit

    module M = Modeltools.Mutable
        (struct type f = flavor type g = gauge type c = constant end)

    let flavors = M.flavors
    let external_flavors = M.external_flavors
    let lorentz = M.lorentz
    let color = M.color
    let propagator = M.propagator
    let width = M.width
    let goldstone = M.goldstone
    let conjugate = M.conjugate
    let fermion = M.fermion
    let vertices = M.vertices
    let fuse2 = M.fuse2
    let fuse3 = M.fuse3
    let fuse = M.fuse
    let max_degree = M.max_degree
    let parameters = M.parameters
    let flavor_of_string = M.flavor_of_string
    let flavor_to_string = M.flavor_to_string
    let flavor_to_TeX = M.flavor_to_TeX
    let flavor_symbol = M.flavor_symbol
    let gauge_symbol = M.gauge_symbol
    let pdg = M.pdg
    let mass_symbol = M.mass_symbol
    let width_symbol = M.width_symbol
    let constant_symbol = M.constant_symbol
    module Ch = M.Ch
    let charges = M.charges

    let rcs = rcs_file

    type symbol =
      | Selfconjugate of string
      | Conjugates of string × string

    type particle =
        { p_name : string;
          p_symbol : symbol;
```

```
      p_spin : Coupling.lorentz;
      p_mass : Comphep_syntax.raw;
      p_width : Comphep_syntax.raw;
      p_color : Color.t;
      p_aux : string option }

let count_flavors particles =
  List.fold_left (fun n p → n +
    match p.p_symbol with
    | Selfconjugate _ → 1
    | Conjugates _ → 2) 0 particles

type particle_flavor =
    { f_name : string;
      f_conjugate : int;
      f_symbol : string;
      f_pdg : int;
      f_spin : Coupling.lorentz;
      f_propagator : gauge Coupling.propagator;
      f_fermion : int;
      f_mass : string;
      f_width : string;
      f_color : Color.t;
      f_aux : string option }

let real_variable = function
  | Comphep_syntax.Integer 0 → "zero"
  | Comphep_syntax.Symbol s → s
  | _ → invalid_arg "real_variable"

let dummy_flavor =
  { f_name = "";
    f_conjugate = − 1;
    f_symbol = "";
    f_pdg = 0;
    f_spin = Coupling.Scalar;
    f_propagator = Coupling.Prop_Scalar;
    f_fermion = 0;
    f_mass = real_variable (Comphep_syntax.integer 0);
    f_width = real_variable (Comphep_syntax.integer 0);
    f_color = Color.Singlet;
    f_aux = None }

let propagator_of_lorentz = function
  | Coupling.Scalar → Coupling.Prop_Scalar
  | Coupling.Spinor → Coupling.Prop_Spinor
  | Coupling.ConjSpinor → Coupling.Prop_ConjSpinor
  | Coupling.Majorana → Coupling.Prop_Majorana
  | Coupling.Maj_Ghost → invalid_arg
        "propagator_of_lorentz: SUSY ghosts do not propagate"
  | Coupling.Vector → Coupling.Prop_Feynman
  | Coupling.Massive_Vector → Coupling.Prop_Unitarity
  | Coupling.Vectorspinor →
```

```
    invalid_arg "propagator_of_lorentz: Vectorspinor"
  | Coupling.Tensor_1 → invalid_arg "propagator_of_lorentz: Tensor_1"
  | Coupling.Tensor_2 → invalid_arg "propagator_of_lorentz: Tensor_2"
  | Coupling.BRS _ → invalid_arg "propagator_of_lorentz: no BRST"

let flavor_of_particle symbol conjg particle =
  let spin = particle.p_spin in
  { f_name = particle.p_name;
    f_conjugate = conjg;
    f_symbol = symbol;
    f_pdg = pdg_heuristic symbol;
    f_spin = spin;
    f_propagator = propagator_of_lorentz spin;
    f_fermion = fermion_of_lorentz spin;
    f_mass = real_variable particle.p_mass;
    f_width = real_variable particle.p_width;
    f_color = particle.p_color;
    f_aux = particle.p_aux }

let flavor_of_antiparticle symbol conjg particle =
  let spin = conjugate_lorentz particle.p_spin in
  { f_name = "anti-" ^ particle.p_name;
    f_conjugate = conjg;
    f_symbol = symbol;
    f_pdg = pdg_heuristic symbol;
    f_spin = spin;
    f_propagator = propagator_of_lorentz spin;
    f_fermion = fermion_of_lorentz spin;
    f_mass = real_variable particle.p_mass;
    f_width = real_variable particle.p_width;
    f_color = Color.conjugate particle.p_color;
    f_aux = particle.p_aux }

let parse_expr text =
  try
    Comphep_parser.expr Comphep_lexer.token (Lexing.from_string text)
  with
  | Parsing.Parse_error → invalid_arg ("parse error: " ^ text)

let parse_function_row = function
  | name :: fct :: comment :: _ → (name, parse_expr fct, comment)
  | _ → invalid_arg "parse_function_row"

let parse_lagragian_row = function
  | p1 :: p2 :: p3 :: p4 :: c :: t :: _ →
      ((p1, p2, p3, p4), parse_expr c, parse_expr t)
  | _ → invalid_arg "parse_lagragian_row"

let parse_symbol s1 s2 =
  if s1 = s2 then
    Selfconjugate (sanitize_symbol s1)
  else
    Conjugates (sanitize_symbol s1, sanitize_symbol s2)
```

```
let parse_spin spin =
  match int_of_string spin with
  | 0 → Coupling.Scalar
  | 1 → Coupling.Spinor
  | 2 → Coupling.Vector
  | _ → invalid_arg ("parse_spin: spin = " ^ spin)

let parse_color color =
  match int_of_string color with
  | 1 → Color.Singlet
  | 3 → Color.SUN 3
  | 8 → Color.AdjSUN 3
  | _ → invalid_arg ("parse_color: color = " ^ color)

let parse_particle_row = function
  | name :: symbol :: symbol_cc :: spin :: mass :: width :: color ::
    aux :: _ →
      { p_name = name;
        p_symbol = parse_symbol symbol symbol_cc;
        p_spin = parse_spin spin;
        p_mass = parse_expr mass;
        p_width = parse_expr width;
        p_color = parse_color color;
        p_aux = match aux with "" → None | _ → Some aux }
  | _ → invalid_arg "parse_particle_row"

let parse_variable_row = function
  | name :: value :: comment :: _ →
      (name, float_of_string value, comment)
  | _ → invalid_arg "parse_variable_row"

let parse_table parse_row name =
  let model, table, titles, rows = input_table name in
  (model, table, titles, List.rev_map parse_row rows)

let input_functions = parse_table parse_function_row
let input_lagrangian = parse_table parse_lagragian_row
let input_particles = parse_table parse_particle_row
let input_variables = parse_table parse_variable_row

let input_model dir idx =
  let idx = string_of_int idx in
  (input_particles (dir ^ "/prtcls" ^ idx ^ ".mdl"),
   input_variables (dir ^ "/vars" ^ idx ^ ".mdl"),
   input_functions (dir ^ "/func" ^ idx ^ ".mdl"),
   input_lagrangian (dir ^ "/lgrng" ^ idx ^ ".mdl"))

let flavors_of_particles particles =
  let flavors = Array.make (count_flavors particles) dummy_flavor in
  ignore (List.fold_left (fun n p →
    match p.p_symbol with
    | Selfconjugate f →
        flavors.(n) ← flavor_of_particle f n p;
        n + 1
```

```
    | Conjugates (f1, f2) →
        flavors.(n) ← flavor_of_particle f1 (n + 1) p;
        flavors.(n + 1) ← flavor_of_antiparticle f2 n p;
        n + 2) 0 particles);
  flavors

module F = Modeltools.Fusions (struct
  type f = flavor
  type c = constant
  let compare = compare
  let conjugate = conjugate
end)

let translate_tensor3 _ = Coupling.Scalar_Scalar_Scalar 1
let translate_tensor4 _ = Coupling.Scalar4 1
let translate_constant _ = ""

let init flavors variables functions vertices =
  let fmax = Array.length flavors − 1 in
  let flist = ThoList.range 0 fmax in
  let clamp_flavor msg f =
    if f ≥ 0 ∨ f ≤ fmax then
      f
    else
      invalid_arg (msg ^ ": invalid flavor: " ^ string_of_int f) in
  let flavor_hash = Hashtbl.create 37 in
  let flavor_of_string s =
    try
      Hashtbl.find flavor_hash s
    with
    | Not_found → invalid_arg ("flavor_of_string: " ^ s) in
  for f = 0 to fmax do
    Hashtbl.add flavor_hash flavors.(f).f_symbol f
  done;
  let vertices3, vertices4 =
    List.fold_left (fun (v3, v4) ((p1, p2, p3, p4), c, t) →
      if p4 = "" then
        (((flavor_of_string p1, flavor_of_string p2, flavor_of_string p3),
          translate_tensor3 t, translate_constant c) :: v3, v4)
      else
        (v3, ((flavor_of_string p1, flavor_of_string p2,
                flavor_of_string p3, flavor_of_string p4),
              translate_tensor4 t, translate_constant c) :: v4))
      ([], []) vertices in
  let max_degree = match vertices4 with [] → 3 | _ → 4 in
  let all_vertices () = (vertices3, vertices4, []) in
  let table = F.of_vertices (all_vertices ()) in
  let input_parameters =
    (real_variable (Comphep_syntax.integer 0), 0.0) ::
    (List.map (fun (n, v, _) → (n, v)) variables) in
  let derived_parameters =
    List.map (fun (n, f, _) → (Coupling.Real n, Coupling.Const 0))
```

```
      functions in
  M.setup
    ~color : (fun f → flavors.(clamp_flavor "color" f).f_color)
    ~pdg : (fun f → flavors.(clamp_flavor "pdg" f).f_pdg)
    ~lorentz : (fun f → flavors.(clamp_flavor "spin" f).f_spin)
    ~propagator : (fun f →
      flavors.(clamp_flavor "propagator" f).f_propagator)
    ~width : (fun f → Coupling.Constant)
    ~goldstone : (fun f → None)
    ~conjugate : (fun f → flavors.(clamp_flavor "conjugate" f).f_conjugate)
    ~fermion : (fun f → flavors.(clamp_flavor "fermion" f).f_fermion)
    ~max_degree
    ~vertices : all_vertices
    ~fuse : (F.fuse2 table, F.fuse3 table, F.fuse table)
    ~flavors : ([("All␣Flavors", flist)])
    ~parameters : (fun () →
      { Coupling.input = input_parameters;
        Coupling.derived = derived_parameters;
        Coupling.derived_arrays = [] })
    ~flavor_of_string
    ~flavor_to_string : (fun f →
      flavors.(clamp_flavor "flavor_to_string" f).f_name)
    ~flavor_to_TeX : (fun f →
      flavors.(clamp_flavor "flavor_to_TeX" f).f_name)
    ~flavor_symbol : (fun f →
      flavors.(clamp_flavor "flavor_symbol" f).f_symbol)
    ~gauge_symbol : (fun () → "")
    ~mass_symbol : (fun f →
      flavors.(clamp_flavor "mass_symbol" f).f_mass)
    ~width_symbol : (fun f →
      flavors.(clamp_flavor "width_symbol" f).f_width)
    ~constant_symbol : (fun c → failwith "constant_symbol")

let particles_file = ref "prtcls1.mdl"
let variables_file = ref "vars1.mdl"
let functions_file = ref "func1.mdl"
let lagrangian_file = ref "lgrng1.mdl"

let load () =
  let (_, _, _, p), v, f, l =
    (input_particles !particles_file, input_variables !variables_file,
     input_functions !functions_file, input_lagrangian !lagrangian_file) in
  init (flavors_of_particles p) [] [] []

let options = Options.create
    [ ("p", Arg.String (fun name → particles_file := name),
        "CompHEP␣particles␣file␣(default:␣" ^ !particles_file ^ ")");
      ("v", Arg.String (fun name → variables_file := name),
        "CompHEP␣variables␣file␣(default:␣" ^ !variables_file ^ ")");
      ("f", Arg.String (fun name → functions_file := name),
        "CompHEP␣functions␣file␣(default:␣" ^ !functions_file ^ ")");
      ("l", Arg.String (fun name → lagrangian_file := name),
```

```
      "CompHEP␣lagrangian␣file␣(default:␣" ^ !lagrangian_file ^ ")");
     ("exec", Arg.Unit load,
      "load␣the␣model␣files␣(required␣_before_␣any␣particle)");
     ("help", Arg.Unit (fun () →
       print_endline
         ("[" ^ String.concat "|"
                 (List.map M.flavor_to_string (M.flavors ())) ^ "]")),
       "print␣information␣on␣the␣model")]

end
```

—15— Hardcoded Targets

15.1 Interface of Targets

```
module Dummy : Target.Maker
```

15.1.1 Supported Targets

```
module Fortran : Target.Maker
module Fortran_Majorana : Target.Maker
module VM : Target.Maker
```

15.1.2 Potential Targets

```
module Fortran77 : Target.Maker
module C : Target.Maker
module Cpp : Target.Maker
module Java : Target.Maker
module Ocaml : Target.Maker
module LaTeX : Target.Maker
```

15.2 Implementation of Targets

```
let rcs_file = RCS.parse "Targets" ["Code␣Generation"]
    { RCS.revision = "$Revision:␣6465␣$";
      RCS.date = "$Date:␣2015-01-10␣16:22:31␣+0100␣(Sat,␣10␣Jan␣2015)␣$";
      RCS.author = "$Author:␣jr_reuter␣$";
      RCS.source
        = "$URL:␣svn+ssh://jr_reuter@login.hepforge.org/hepforge/svn/whizard/trunk/omega/sr
module Dummy (F : Fusion.Maker) (P : Momentum.T) (M : Model.T) =
  struct
    let rcs_list = []
    type amplitudes = Fusion.Multi(F)(P)(M).amplitudes
    type diagnostic = All | Arguments | Momenta | Gauge
    let options = Options.empty
```

```
    let amplitudes_to_channel _ _ _ = failwith "Targets.Dummy"
    let parameters_to_channel _ = failwith "Targets.Dummy"
  end
```

15.2.1 O'Mega Virtual Machine with Fortran 90/95

Preliminaries

```
module VM (Fusion_Maker : Fusion.Maker) (P : Momentum.T) (M : Model.T) =
  struct
    let rcs_list =
      [RCS.rename rcs_file "Targets.VM"
                 ["Virtual␣Machine␣able␣to␣compute␣amplitudes␣from␣byte␣code"]]

    open Coupling
    open Format

    module CM = Colorize.It(M)
    module F = Fusion_Maker(P)(M)
    module CF = Fusion.Multi(Fusion_Maker)(P)(M)
    module CFlow = Color.Flow
    type amplitudes = CF.amplitudes
```

Options.

```
    type diagnostic = All | Arguments | Momenta | Gauge

    let wrapper_module = ref "ovm_wrapper"
    let parameter_module_external = ref "some_external_module_with_model_info"
    let bytecode_file = ref "bytecode.hbc"
    let md5sum = ref None
    let openmp = ref false
    let kind = ref "default"
    let whizard = ref false

    let options = Options.create
      [ "wrapper_module", Arg.String (fun s → wrapper_module := s),
          "name␣of␣wrapper␣module";
        "bytecode_file", Arg.String (fun s → bytecode_file := s),
          "bytecode␣file␣to␣be␣used␣in␣wrapper";
        "parameter_module_external", Arg.String (fun s →
                                     parameter_module_external := s),
          "external␣parameter␣module␣to␣be␣used␣in␣wrapper";
        "md5sum", Arg.String (fun s → md5sum := Some s),
          "transfer␣MD5␣checksum␣in␣wrapper";
        "whizard", Arg.Set whizard, "include␣WHIZARD␣interface␣in␣wrapper";
        "openmp", Arg.Set openmp,
          "activate␣parallel␣computation␣of␣amplitude␣with␣OpenMP"]
```

This is part of OCaml 4.01.

```
    let (| >) fn x = x fn
```

```
let (@@) fn x = fn x
```

Integers encode the opcodes (operation codes).

```
let ovm_ADD_MOMENTA = 1
let ovm_CALC_BRAKET = 2

let ovm_LOAD_SCALAR = 10
let ovm_LOAD_SPINOR_INC = 11
let ovm_LOAD_SPINOR_OUT = 12
let ovm_LOAD_CONJSPINOR_INC = 13
let ovm_LOAD_CONJSPINOR_OUT = 14
let ovm_LOAD_MAJORANA_INC = 15
let ovm_LOAD_MAJORANA_OUT = 16
let ovm_LOAD_VECTOR_INC = 17
let ovm_LOAD_VECTOR_OUT = 18
let ovm_LOAD_VECTORSPINOR_INC = 19
let ovm_LOAD_VECTORSPINOR_OUT = 20
let ovm_LOAD_TENSOR2_INC = 21
let ovm_LOAD_TENSOR2_OUT = 22
let ovm_LOAD_BRS_SCALAR = 30
let ovm_LOAD_BRS_SPINOR_INC = 31
let ovm_LOAD_BRS_SPINOR_OUT = 32
let ovm_LOAD_BRS_CONJSPINOR_INC = 33
let ovm_LOAD_BRS_CONJSPINOR_OUT = 34
let ovm_LOAD_BRS_VECTOR_INC = 37
let ovm_LOAD_BRS_VECTOR_OUT = 38
let ovm_LOAD_MAJORANA_GHOST_INC = 23
let ovm_LOAD_MAJORANA_GHOST_OUT = 24
let ovm_LOAD_BRS_MAJORANA_INC = 35
let ovm_LOAD_BRS_MAJORANA_OUT = 36

let ovm_PROPAGATE_SCALAR = 51
let ovm_PROPAGATE_COL_SCALAR = 52
let ovm_PROPAGATE_GHOST = 53
let ovm_PROPAGATE_SPINOR = 54
let ovm_PROPAGATE_CONJSPINOR = 55
let ovm_PROPAGATE_MAJORANA = 56
let ovm_PROPAGATE_COL_MAJORANA = 57
let ovm_PROPAGATE_UNITARITY = 58
let ovm_PROPAGATE_COL_UNITARITY = 59
let ovm_PROPAGATE_FEYNMAN = 60
let ovm_PROPAGATE_COL_FEYNMAN = 61
let ovm_PROPAGATE_VECTORSPINOR = 62
let ovm_PROPAGATE_TENSOR2 = 63
```

ovm_PROPAGATE_NONE has to be split up to different types to work in conjunction with color MC ...

```
let ovm_PROPAGATE_NONE = 64

let ovm_FUSE_V_FF = -1
```

let $ovm\_FUSE\_F\_VF = -2$
let $ovm\_FUSE\_F\_FV = -3$
let $ovm\_FUSE\_VA\_FF = -4$
let $ovm\_FUSE\_F\_VAF = -5$
let $ovm\_FUSE\_F\_FVA = -6$
let $ovm\_FUSE\_VA2\_FF = -7$
let $ovm\_FUSE\_F\_VA2F = -8$
let $ovm\_FUSE\_F\_FVA2 = -9$
let $ovm\_FUSE\_A\_FF = -10$
let $ovm\_FUSE\_F\_AF = -11$
let $ovm\_FUSE\_F\_FA = -12$
let $ovm\_FUSE\_VL\_FF = -13$
let $ovm\_FUSE\_F\_VLF = -14$
let $ovm\_FUSE\_F\_FVL = -15$
let $ovm\_FUSE\_VR\_FF = -16$
let $ovm\_FUSE\_F\_VRF = -17$
let $ovm\_FUSE\_F\_FVR = -18$
let $ovm\_FUSE\_VLR\_FF = -19$
let $ovm\_FUSE\_F\_VLRF = -20$
let $ovm\_FUSE\_F\_FVLR = -21$
let $ovm\_FUSE\_SP\_FF = -22$
let $ovm\_FUSE\_F\_SPF = -23$
let $ovm\_FUSE\_F\_FSP = -24$
let $ovm\_FUSE\_S\_FF = -25$
let $ovm\_FUSE\_F\_SF = -26$
let $ovm\_FUSE\_F\_FS = -27$
let $ovm\_FUSE\_P\_FF = -28$
let $ovm\_FUSE\_F\_PF = -29$
let $ovm\_FUSE\_F\_FP = -30$
let $ovm\_FUSE\_SL\_FF = -31$
let $ovm\_FUSE\_F\_SLF = -32$
let $ovm\_FUSE\_F\_FSL = -33$
let $ovm\_FUSE\_SR\_FF = -34$
let $ovm\_FUSE\_F\_SRF = -35$
let $ovm\_FUSE\_F\_FSR = -36$
let $ovm\_FUSE\_SLR\_FF = -37$
let $ovm\_FUSE\_F\_SLRF = -38$
let $ovm\_FUSE\_F\_FSLR = -39$

let $ovm\_FUSE\_G\_GG = -40$
let $ovm\_FUSE\_V\_SS = -41$
let $ovm\_FUSE\_S\_VV = -42$
let $ovm\_FUSE\_S\_VS = -43$
let $ovm\_FUSE\_V\_SV = -44$
let $ovm\_FUSE\_S\_SS = -45$
let $ovm\_FUSE\_S\_SVV = -46$
let $ovm\_FUSE\_V\_SSV = -47$
let $ovm\_FUSE\_S\_SSS = -48$
let $ovm\_FUSE\_V\_VVV = -49$

let $ovm\_FUSE\_S\_G2 = -50$

```
let ovm_FUSE_G_SG = -51
let ovm_FUSE_G_GS = -52
let ovm_FUSE_S_G2_SKEW = -53
let ovm_FUSE_G_SG_SKEW = -54
let ovm_FUSE_G_GS_SKEW = -55

let inst_length = 8
```

Some helper functions.

```
let printi ~lhs:l ~rhs1:r1 ?coupl:(cp = 0) ?coeff:(co = 0)
          ?rhs2:(r2 = 0) ?rhs3:(r3 = 0) ?rhs4:(r4 = 0) code =
  printf "@\n%d␣%d␣%d␣%d␣%d␣%d␣%d␣%d" code cp co l r1 r2 r3 r4

let nl () = printf "@\n"

let print_int_lst lst = nl (); lst |> List.iter (printf "%d␣␣␣")

let print_str_lst lst = nl (); lst |> List.iter (printf "%s␣")

let break () = printi ~lhs:0 ~rhs1:0 0
```

Copied from below. Needed for header.

Could be fused with *lorentz_ordering*.

```
type declarations =
  { scalars : F.wf list;
    spinors : F.wf list;
    conjspinors : F.wf list;
    realspinors : F.wf list;
    ghostspinors : F.wf list;
    vectorspinors : F.wf list;
    vectors : F.wf list;
    ward_vectors : F.wf list;
    massive_vectors : F.wf list;
    tensors_1 : F.wf list;
    tensors_2 : F.wf list;
    brs_scalars : F.wf list;
    brs_spinors : F.wf list;
    brs_conjspinors : F.wf list;
    brs_realspinors : F.wf list;
    brs_vectorspinors : F.wf list;
    brs_vectors : F.wf list;
    brs_massive_vectors : F.wf list }

let rec classify_wfs' acc = function
  | [] → acc
  | wf :: rest →
      classify_wfs'
        (match CM.lorentz (F.flavor wf) with
        | Scalar → {acc with scalars = wf :: acc.scalars}
        | Spinor → {acc with spinors = wf :: acc.spinors}
        | ConjSpinor → {acc with conjspinors = wf :: acc.conjspinors}
        | Majorana → {acc with realspinors = wf :: acc.realspinors}
```

```
          | Maj_Ghost → {acc with ghostspinors = wf :: acc.ghostspinors}
          | Vectorspinor →
              {acc with vectorspinors = wf :: acc.vectorspinors}
          | Vector → {acc with vectors = wf :: acc.vectors}
          | Massive_Vector →
              {acc with massive_vectors = wf :: acc.massive_vectors}
          | Tensor_1 → {acc with tensors_1 = wf :: acc.tensors_1}
          | Tensor_2 → {acc with tensors_2 = wf :: acc.tensors_2}
          | BRS Scalar → {acc with brs_scalars = wf :: acc.brs_scalars}
          | BRS Spinor → {acc with brs_spinors = wf :: acc.brs_spinors}
          | BRS ConjSpinor → {acc with brs_conjspinors =
                                wf :: acc.brs_conjspinors}
          | BRS Majorana → {acc with brs_realspinors =
                              wf :: acc.brs_realspinors}
          | BRS Vectorspinor → {acc with brs_vectorspinors =
                                  wf :: acc.brs_vectorspinors}
          | BRS Vector → {acc with brs_vectors = wf :: acc.brs_vectors}
          | BRS Massive_Vector → {acc with brs_massive_vectors =
                                    wf :: acc.brs_massive_vectors}
          | BRS _ → invalid_arg "Targets.classify_wfs':␣not␣needed␣here")
          rest

let classify_wfs wfs = classify_wfs'
  { scalars = [];
    spinors = [];
    conjspinors = [];
    realspinors = [];
    ghostspinors = [];
    vectorspinors = [];
    vectors = [];
    ward_vectors = [];
    massive_vectors = [];
    tensors_1 = [];
    tensors_2 = [];
    brs_scalars = [];
    brs_spinors = [];
    brs_conjspinors = [];
    brs_realspinors = [];
    brs_vectorspinors = [];
    brs_vectors = [];
    brs_massive_vectors = [] } wfs
```

Sets and maps

The OVM identifies all objects via integers. Therefore, we need maps which assign the abstract object a unique ID.

I want *int lists* with less elements to come first. Used in conjunction with the int list representation of momenta, this will set the outer particles at first position and allows the OVM to set them without further instructions.

Using the Momentum module might give better performance than integer lists?

```
let rec int_lst_compare (e1 : int list) (e2 : int list) =
  match e1, e2 with
  | [], [] → 0
  | _, [] → +1
  | [], _ → −1
  | [_; _], [_] → +1
  | [_], [_; _] → −1
  | hd1 :: tl1, hd2 :: tl2 →
      let c = compare hd1 hd2 in
      if (c ≢ 0 ∧ List.length tl1 = List.length tl2) then
        c
      else
        int_lst_compare tl1 tl2
```

We need a canonical ordering for the different types of wfs. Copied, and slightly modified to order *wf*s, from `fusion.ml`.

```
let lorentz_ordering wf =
  match CM.lorentz (F.flavor wf) with
  | Scalar → 0
  | Spinor → 1
  | ConjSpinor → 2
  | Majorana → 3
  | Vector → 4
  | Massive_Vector → 5
  | Tensor_2 → 6
  | Tensor_1 → 7
  | Vectorspinor → 8
  | BRS Scalar → 9
  | BRS Spinor → 10
  | BRS ConjSpinor → 11
  | BRS Majorana → 12
  | BRS Vector → 13
  | BRS Massive_Vector → 14
  | BRS Tensor_2 → 15
  | BRS Tensor_1 → 16
  | BRS Vectorspinor → 17
  | Maj_Ghost → invalid_arg "lorentz_ordering:␣not␣implemented"
  | BRS _ → invalid_arg "lorentz_ordering:␣not␣needed"

let wf_compare (wf1, mult1) (wf2, mult2) =
  let c1 = compare (lorentz_ordering wf1) (lorentz_ordering wf2) in
  if c1 ≠ 0 then
    c1
  else
    let c2 = compare wf1 wf2 in
    if c2 ≠ 0 then
      c2
```

```
      else
        compare mult1 mult2

  let amp_compare amp1 amp2 =
    let cflow a = CM.flow (F.incoming a) (F.outgoing a) in
    let c1 = compare (cflow amp1) (cflow amp2) in
    if c1 ≠ 0 then
      c1
    else
      let process_sans_color a =
        (List.map CM.flavor_sans_color (F.incoming a),
         List.map CM.flavor_sans_color (F.outgoing a)) in
      compare (process_sans_color amp1) (process_sans_color amp2)

  let level_compare (f1, amp1) (f2, amp2) =
    let p1 = F.momentum_list (F.lhs f1)
    and p2 = F.momentum_list (F.lhs f2) in
    let c1 = int_lst_compare p1 p2 in
    if c1 ≠ 0 then
      c1
    else
      let c2 = compare f1 f2 in
      if c2 ≠ 0 then
        c2
      else
        amp_compare amp1 amp2

  module ISet = Set.Make (struct type t = int list
                            let compare = int_lst_compare end)

  module WFSet = Set.Make (struct type t = CF.wf × int
                             let compare = wf_compare end)

  module CSet = Set.Make (struct type t = CM.constant
                            let compare = compare end)

  module FSet = Set.Make (struct type t = F.fusion × F.amplitude
                            let compare = level_compare end)
```

It might be preferable to use a *PMap* which maps mom to int, instead of this way. More standard functions like *mem* could be used. Also, *get_ID* would be faster, $\mathcal{O}(\log N)$ instead of $\mathcal{O}(N)$, and simpler. For 8 gluons: N=127 momenta. Minor performance issue.

```
  module IMap = Map.Make (struct type t = int let compare = compare end)
```

For *wf*s it is crucial for the performance to use a different type of *Map*s.

```
  module WFMap = Map.Make (struct type t = CF.wf × int
                             let compare = wf_compare end)

  type lookups = { pmap : int list IMap.t;
                   wfmap : int WFMap.t;
                   cmap : CM.constant IMap.t × CM.constant IMap.t;
```

```
                   amap : F.amplitude IMap.t;
                   n_wfs : int list;
                   amplitudes : CF.amplitudes;
                   dict : F.amplitude -> F.wf -> int }

let largest_key imap =
  if (IMap.is_empty imap) then
    failwith "largest_key: Map is empty!"
  else
    fst (IMap.max_binding imap)
```

OCaml's *compare* from pervasives cannot compare functional types, e.g. for type *amplitude*, if no specific equality function is given ("equal: functional value"). Therefore, we allow to specify the ordering.

```
let get_ID' comp map elt : int =
  let smallmap = IMap.filter (fun _ x -> (comp x elt) = 0 ) map in
  if IMap.is_empty smallmap then
    raise Not_found
  else
    fst (IMap.min_binding smallmap)
```

Trying to curry *map* here leads to type errors of the polymorphic function *get_ID*?

```
let get_ID map = match map with
  | map -> get_ID' compare map

let get_const_ID map x = match map with
  | (map1, map2) -> try get_ID' compare map1 x with
                    _ -> try get_ID' compare map2 x with
                    _ -> failwith "Impossible"
```

Creating an integer map of a list with an optional argument that indicates where the map should start counting.

```
let map_of_list ?start:(st=1) lst =
  let g (ind, map) wf = (succ ind, IMap.add ind wf map) in
  lst |> List.fold_left g (st, IMap.empty) |> snd

let wf_map_of_list ?start:(st=1) lst =
  let g (ind, map) wf = (succ ind, WFMap.add wf ind map) in
  lst |> List.fold_left g (st, WFMap.empty) |> snd
```

Header

It would be nice to safe the creation date as comment. However, the Unix module doesn't seem to be loaded on default.

```
let version_string x = RCS.name x ^ "-rev" ^ RCS.revision x

let print_description cmdline =
  rcs_list @ [M.rcs] |> List.map version_string
```

```
                              —> List.iter (printf "%s\n");
    printf "@\nBytecode␣file␣generated␣automatically␣by␣O'Mega␣for␣OVM";
    printf "@\nDo␣not␣delete␣any␣lines.␣You␣called␣O'Mega␣with";
    printf "@\n␣␣%s" cmdline;
          printf "@\n"

  let num_classified_wfs wfs =
    let wfs' = classify_wfs wfs in
    List.map List.length
      [ wfs'.scalars @ wfs'.brs_scalars;
        wfs'.spinors @ wfs'.brs_spinors;
        wfs'.conjspinors @ wfs'.brs_conjspinors;
        wfs'.realspinors @ wfs'.brs_realspinors @ wfs'.ghostspinors;
        wfs'.vectors @ wfs'.massive_vectors @ wfs'.brs_vectors
          @ wfs'.brs_massive_vectors @ wfs'.ward_vectors;
        wfs'.tensors_2;
        wfs'.tensors_1;
        wfs'.vectorspinors ]

  let description_classified_wfs =
    [ "N_scalars";
      "N_spinors";
      "N_conjspinors";
      "N_bispinors";
      "N_vectors";
      "N_tensors_2";
      "N_tensors_1";
      "N_vectorspinors" ]

  let num_particles_in amp =
    match CF.flavors amp with
    | [] → 0
    | (fin, _) :: _ → List.length fin

  let num_particles_out amp =
    match CF.flavors amp with
    | [] → 0
    | (_, fout) :: _ → List.length fout

  let num_particles amp =
    match CF.flavors amp with
    | [] → 0
    | (fin, fout) :: _ → List.length fin + List.length fout

  let num_color_indices_default = 2 (* Standard model and non-color-
exotica *)

  let num_color_indices amp =
    try CFlow.rank (List.hd (CF.color_flows amp)) with
    _ → num_color_indices_default

  let num_color_factors amp =
    let table = CF.color_factors amp in
    let n_cflow = Array.length table
```

```
    and n_cfactors = ref 0 in
    for c1 = 0 to pred n_cflow do
      for c2 = 0 to pred n_cflow do
        if c1 ≤ c2 then begin
          match table.(c1).(c2) with
          | [] → ()
          | _ → incr n_cfactors
        end
      done
    done;
    !n_cfactors
```

```
  let num_helicities amp = amp |> CF.helicities |> List.length
```

```
  let num_flavors amp = amp |> CF.flavors |> List.length
```

```
  let num_ks amp = amp |> CF.processes |> List.length
```

```
  let num_color_flows amp = amp |> CF.color_flows |> List.length
```

Use *fst* since $WFSet.t = F.wf \times int$.

```
  let num_wfs wfset = wfset |> WFSet.elements |> List.map fst
                                   —> num_classified_wfs
```

largest_key gives the number of momenta if applied to *pmap*.

```
  let num_lst lookups wfset =
    [ largest_key lookups.pmap;
      num_particles lookups.amplitudes;
      num_particles_in lookups.amplitudes;
      num_particles_out lookups.amplitudes;
      num_ks lookups.amplitudes;
      num_helicities lookups.amplitudes;
      num_color_flows lookups.amplitudes;
      num_color_indices lookups.amplitudes;
      num_flavors lookups.amplitudes;
      num_color_factors lookups.amplitudes ] @ num_wfs wfset
```

```
  let description_lst =
    [ "N_momenta";
      "N_particles";
      "N_prt_in";
      "N_prt_out";
      "N_amplitudes";
      "N_helicities";
      "N_col_flows";
      "N_col_indices";
      "N_flavors";
      "N_col_factors" ] @ description_classified_wfs
```

```
  let print_header' numbers =
    let chopped_num_lst = ThoList.chopn inst_length numbers
    and chopped_desc_lst = ThoList.chopn inst_length description_lst
    and printer a b = print_str_lst a; print_int_lst b in
```

```
  List.iter2 printer chopped_desc_lst chopped_num_lst

let print_header lookups wfset = print_header' (num_lst lookups wfset)

let print_zero_header () =
  let rec zero_list' j =
    if j < 1 then []
    else 0 :: zero_list' (j - 1) in
  let zero_list i = zero_list' (i + 1) in
  description_lst |> List.length |> zero_list |> print_header'
```

Tables

```
let print_spin_table' tuples =
  match tuples with
  | [] -> ()
  | _ -> tuples |> List.iter ( fun (tuple1, tuple2) ->
      tuple1 @ tuple2 |> List.map (Printf.sprintf "%d␣")
                    —> String.concat "" —> printf "@\n%s" )

let print_spin_table amplitudes =
  printf "@\nSpin␣states␣table";
  print_spin_table' @@ CF.helicities amplitudes

let print_flavor_table tuples =
  match tuples with
  | [] -> ()
  | _ -> List.iter ( fun tuple -> tuple
                    —> List.map (fun f -> Printf.sprintf "%d␣" @@ M.pdg f)
                    —> String.concat "" —> printf "@\n%s"
                   ) tuples

let print_flavor_tables amplitudes =
  printf "@\nFlavor␣states␣table";
  print_flavor_table @@ List.map (fun (fin, fout) -> fin @ fout)
                     @@ CF.flavors amplitudes

let print_color_flows_table' tuple =
    match CFlow.to_lists tuple with
    | [] -> ()
    | cfs -> printf "@\n%s" @@ String.concat "" @@ List.map
            ( fun cf -> cf |> List.map (Printf.sprintf "%d␣")
                          —> String.concat ""
            ) cfs

let print_color_flows_table tuples =
  match tuples with
  | [] -> ()
  | _ -> List.iter print_color_flows_table' tuples

let print_ghost_flags_table tuples =
  match tuples with
  | [] -> ()
```

```
| _ →
  List.iter (fun tuple →
  match CFlow.ghost_flags tuple with
      | [] → ()
      | gfs → printf "@\n"; List.iter (fun gf → printf "%s␣"
        (if gf then "1" else "0") ) gfs
  ) tuples

let format_power
  { CFlow.num = num; CFlow.den = den; CFlow.power = pwr } =
  match num, den, pwr with
  | _, 0, _ → invalid_arg "targets.format_power:␣zero␣denominator"
  | n, d, p → [n; d; p]

let format_powers = function
  | [] → [0]
  | powers → List.flatten (List.map format_power powers)
```

Straightforward iteration gives a great speedup compared to the fancier approach which only collects nonzero colorfactors.

```
let print_color_factor_table table =
  let n_cflow = Array.length table in
  if n_cflow > 0 then begin
    for c1 = 0 to pred n_cflow do
      for c2 = 0 to pred n_cflow do
        if c1 ≤ c2 then begin
          match table.(c1).(c2) with
          | [] → ()
          | cf → printf "@\n"; List.iter (printf "%9d")
            ([succ c1; succ c2] @ (format_powers cf));
        end
      done
    done
  end

let option_to_binary = function
  | Some _ → "1"
  | None → "0"

let print_flavor_color_table n_flv n_cflow table =
  if n_flv > 0 then begin
    for c = 0 to pred n_cflow do
      printf "@\n";
      for f = 0 to pred n_flv do
        printf "%s␣" (option_to_binary table.(f).(c))
      done;
    done;
  end

let print_color_tables amplitudes =
  let cflows = CF.color_flows amplitudes
  and cfactors = CF.color_factors amplitudes in
  printf "@\nColor␣flows␣table:␣[␣(i,␣j)␣(k,␣l)␣->␣(m,␣n)␣...]";
```

```
print_color_flows_table cflows;
printf "@\nColor␣ghost␣flags␣table:";
print_ghost_flags_table cflows;
printf "@\nColor␣factors␣table:␣[␣i,␣j:␣num␣den␣power],␣%s"
  "i,␣j␣are␣indexed␣color␣flows";
print_color_factor_table cfactors;
printf "@\nFlavor␣color␣combination␣is␣allowed:";
print_flavor_color_table (num_flavors amplitudes) (List.length
  (CF.color_flows amplitudes)) (CF.process_table amplitudes)
```

Momenta

Add the momenta of a WFSet to a Iset. For now, we are throwing away the information to which amplitude the momentum belongs. This could be optimized for random color flow computations.

```
let momenta_set wfset =
  let get_mom wf = wf |> fst |> F.momentum_list in
  let momenta = List.map get_mom (WFSet.elements wfset) in
  momenta |> List.fold_left (fun set x → set |> ISet.add x) ISet.empty

let chop_in_3 lst =
  let ceil_div i j = if (i mod j = 0) then i/j else i/j + 1 in
  ThoList.chopn (ceil_div (List.length lst) 3) lst
```

Assign momenta via instruction code. External momenta [_] are already set by the OVM. To avoid unnecessary look-ups of IDs we seperate two cases. If we have more, we split up in two or three parts.

```
let add_mom p pmap =
  let print_mom lhs rhs1 rhs2 rhs3 = if (rhs1 ≢ 0) then
    printi ~lhs : lhs ~rhs1 : rhs1 ~rhs2 : rhs2 ~rhs3 : rhs3 ovm_ADD_MOMENTA in
  let get_p_ID = get_ID pmap in
  match p with
  | [] | [_] → print_mom 0 0 0 0
  | [rhs1; rhs2] → print_mom (get_p_ID [rhs1; rhs2]) rhs1 rhs2 0
  | [rhs1; rhs2; rhs3] → print_mom (get_p_ID [rhs1; rhs2; rhs3]) rhs1 rhs2 rhs3
  | more →
      let ids = List.map get_p_ID (chop_in_3 more) in
      if (List.length ids = 3) then
        print_mom (get_p_ID more) (List.nth ids 0) (List.nth ids 1)
          (List.nth ids 2)
      else
        print_mom (get_p_ID more) (List.nth ids 0) (List.nth ids 1) 0
```

Hand through the current level and print level seperators if necessary.

```
let add_all_mom lookups pset =
  let add_all' level p =
    let level' = List.length p in
    if (level' > level ∧ level' > 3) then break ();
    add_mom p lookups.pmap; level'
  in
```

```
    ignore (pset |> ISet.elements |> List.fold_left add_all' 1)
```

Expand a set of momenta to contain all needed momenta for the computation in the OVM. For this, we create a list of sets which contains the chopped momenta and unify them afterwards. If the set has become larger, we expand again.

```
let rec expand_pset p =
  let momlst = ISet.elements p in
  let pset_of lst = List.fold_left (fun s x → ISet.add x s) ISet.empty
    lst in
  let sets = List.map (fun x → pset_of (chop_in_3 x) ) momlst in
  let bigset = List.fold_left ISet.union ISet.empty sets in
  let biggerset = ISet.union bigset p in
  if (List.length momlst < List.length (ISet.elements biggerset) ) then
    expand_pset biggerset
  else
    biggerset

let mom_ID pmap wf = get_ID pmap (F.momentum_list wf)
```

Wavefunctions and externals

mult_wf is needed because the *wf* with same combination of flavor and momentum can have different dependencies and content.

```
let mult_wf dict amplitude wf =
  try
    wf, dict amplitude wf
  with
    | Not_found → wf, 0
```

Build the union of all *wf*s of all amplitudes and a map of the amplitudes.

```
let wfset_amps amplitudes =
  let amap = amplitudes |> CF.processes |> List.sort amp_compare
                          —> map_of_list
  and dict = CF.dictionary amplitudes in
  let wfset_amp amp =
    let f = mult_wf dict amp in
    let lst = List.map f ((F.externals amp) @ (F.variables amp)) in
    lst |> List.fold_left (fun s x → WFSet.add x s) WFSet.empty in
  let list_of_sets = amplitudes |> CF.processes |> List.map wfset_amp in
    List.fold_left WFSet.union WFSet.empty list_of_sets, amap
```

To obtain the Fortran index, we substract the number of precedent wave functions.

```
let lorentz_ordering_reduced wf =
  match CM.lorentz (F.flavor wf) with
  | Scalar | BRS Scalar → 0
  | Spinor | BRS Spinor → 1
  | ConjSpinor | BRS ConjSpinor → 2
  | Majorana | BRS Majorana → 3
  | Vector | BRS Vector | Massive_Vector | BRS Massive_Vector → 4
```

```
    | Tensor_2 | BRS Tensor_2 → 5
    | Tensor_1 | BRS Tensor_1 → 6
    | Vectorspinor | BRS Vectorspinor → 7
    | Maj_Ghost → invalid_arg "lorentz_ordering: not implemented"
    | BRS _ → invalid_arg "lorentz_ordering: not needed"

let wf_index wfmap num_lst (wf, i) =
  let wf_ID = WFMap.find (wf, i) wfmap
  and sum lst = List.fold_left (fun x y → x + y) 0 lst in
    wf_ID − sum (ThoList.hdn (lorentz_ordering_reduced wf) num_lst)

let print_ext lookups amp_ID inc (wf, i) =
  let mom = (F.momentum_list wf) in
  let outer_index = if List.length mom = 1 then List.hd mom else
    failwith "targets.print_ext: called with non-external particle"
  and f = F.flavor wf in
  let pdg = CM.pdg f
  and wf_code =
    match CM.lorentz f with
    | Scalar → ovm_LOAD_SCALAR
    | BRS Scalar → ovm_LOAD_BRS_SCALAR
    | Spinor →
        if inc then ovm_LOAD_SPINOR_INC
        else ovm_LOAD_SPINOR_OUT
    | BRS Spinor →
        if inc then ovm_LOAD_BRS_SPINOR_INC
        else ovm_LOAD_BRS_SPINOR_OUT
    | ConjSpinor →
        if inc then ovm_LOAD_CONJSPINOR_INC
        else ovm_LOAD_CONJSPINOR_OUT
    | BRS ConjSpinor →
        if inc then ovm_LOAD_BRS_CONJSPINOR_INC
        else ovm_LOAD_BRS_CONJSPINOR_OUT
    | Vector | Massive_Vector →
        if inc then ovm_LOAD_VECTOR_INC
        else ovm_LOAD_VECTOR_OUT
    | BRS Vector | BRS Massive_Vector →
        if inc then ovm_LOAD_BRS_VECTOR_INC
        else ovm_LOAD_BRS_VECTOR_OUT
    | Tensor_2 →
        if inc then ovm_LOAD_TENSOR2_INC
        else ovm_LOAD_TENSOR2_OUT
    | Vectorspinor | BRS Vectorspinor →
        if inc then ovm_LOAD_VECTORSPINOR_INC
        else ovm_LOAD_VECTORSPINOR_OUT
    | Majorana →
        if inc then ovm_LOAD_MAJORANA_INC
        else ovm_LOAD_MAJORANA_OUT
    | BRS Majorana →
        if inc then ovm_LOAD_BRS_MAJORANA_INC
        else ovm_LOAD_BRS_MAJORANA_OUT
```

```
      | Maj_Ghost →
          if inc then ovm_LOAD_MAJORANA_GHOST_INC
          else ovm_LOAD_MAJORANA_GHOST_OUT
      | Tensor_1 →
          invalid_arg "targets.print_ext: Tensor_1 only internal"
      | BRS _ →
          failwith "targets.print_ext: Not implemented"
    and wf_ind = wf_index lookups.wfmap lookups.n_wfs (wf, i)
    in
      printi wf_code ~lhs : wf_ind ~coupl : (abs(pdg)) ~rhs1 : outer_index ~rhs4 :
amp_ID

  let print_ext_amp lookups amplitude =
    let incoming = (List.map (fun _ → true) (F.incoming amplitude) @
                    List.map (fun _ → false) (F.outgoing amplitude))
    and amp_ID = get_ID' amp_compare lookups.amap amplitude in
    let wf_tpl wf = mult_wf lookups.dict amplitude wf in
    let print_ext_wf inc wf = wf |> wf_tpl |> print_ext lookups amp_ID inc in
      List.iter2 print_ext_wf incoming (F.externals amplitude)

  let print_externals lookups seen_wfs amplitude =
    let externals =
      List.combine
        (F.externals amplitude)
        (List.map (fun _ → true) (F.incoming amplitude) @
         List.map (fun _ → false) (F.outgoing amplitude)) in
    List.fold_left (fun seen (wf, incoming) →
      let amp_ID = get_ID' amp_compare lookups.amap amplitude in
      let wf_tpl = mult_wf lookups.dict amplitude wf in
      if ¬ (WFSet.mem wf_tpl seen) then begin
        wf_tpl |> print_ext lookups amp_ID incoming
      end;
      WFSet.add wf_tpl seen) seen_wfs externals
```

print_externals and *print_ext_amp* do in principle the same thing but *print_externals* filters out dublicate external wave functions. Even with *print_externals* the same (numerically) external wave function will be loaded if it belongs to a different color flow, just as in the native Fortran code. For color MC, *print_ext_amp* has to be used (redundant instructions but only one flow is computed) and the filtering of duplicate fusions has to be disabled.

```
  let print_ext_amps lookups =
    let print_external_amp s x = print_externals lookups s x in
    ignore (
      List.fold_left print_external_amp WFSet.empty
        (CF.processes lookups.amplitudes)
      )
```

(*

Currents

*)

Parallelization issues: All fusions have to be completed before the propagation takes place. Preferably each fusion and propagation is done by one thread. Solution: All fusions are subinstructions, i.e. if they are read by the main loop they are skipped. If a propagation occurs, all fusions have to be computed first. The additional control bit is the sign of the first int of an instruction.

```
let print_fermion_current code_a code_b code_c coeff lhs c wf1 wf2 fusion =
  let printc code r1 r2 = printi code ~lhs : lhs ~coupl : c ~coeff : coeff
    ~rhs1 : r1 ~rhs2 : r2 in
  match fusion with
  | F13 → printc code_a wf1 wf2
  | F31 → printc code_a wf2 wf1
  | F23 → printc code_b wf1 wf2
  | F32 → printc code_b wf2 wf1
  | F12 → printc code_c wf1 wf2
  | F21 → printc code_c wf2 wf1

  let ferm_print_current = function
    | coeff, Psibar, V, Psi → print_fermion_current
      ovm_FUSE_V_FF ovm_FUSE_F_VF ovm_FUSE_F_FV coeff
    | coeff, Psibar, VA, Psi → print_fermion_current
      ovm_FUSE_VA_FF ovm_FUSE_F_VAF ovm_FUSE_F_FVA coeff
    | coeff, Psibar, VA2, Psi → print_fermion_current
      ovm_FUSE_VA2_FF ovm_FUSE_F_VA2F ovm_FUSE_F_FVA2 coeff
    | coeff, Psibar, A, Psi → print_fermion_current
      ovm_FUSE_A_FF ovm_FUSE_F_AF ovm_FUSE_F_FA coeff
    | coeff, Psibar, VL, Psi → print_fermion_current
      ovm_FUSE_VL_FF ovm_FUSE_F_VLF ovm_FUSE_F_FVL coeff
    | coeff, Psibar, VR, Psi → print_fermion_current
      ovm_FUSE_VR_FF ovm_FUSE_F_VRF ovm_FUSE_F_FVR coeff
    | coeff, Psibar, VLR, Psi → print_fermion_current
      ovm_FUSE_VLR_FF ovm_FUSE_F_VLRF ovm_FUSE_F_FVLR coeff
    | coeff, Psibar, SP, Psi → print_fermion_current
      ovm_FUSE_SP_FF ovm_FUSE_F_SPF ovm_FUSE_F_FSP coeff
    | coeff, Psibar, S, Psi → print_fermion_current
      ovm_FUSE_S_FF ovm_FUSE_F_SF ovm_FUSE_F_FS coeff
    | coeff, Psibar, P, Psi → print_fermion_current
      ovm_FUSE_P_FF ovm_FUSE_F_PF ovm_FUSE_F_FP coeff
    | coeff, Psibar, SL, Psi → print_fermion_current
      ovm_FUSE_SL_FF ovm_FUSE_F_SLF ovm_FUSE_F_FSL coeff
    | coeff, Psibar, SR, Psi → print_fermion_current
      ovm_FUSE_SR_FF ovm_FUSE_F_SRF ovm_FUSE_F_FSR coeff
    | coeff, Psibar, SLR, Psi → print_fermion_current
      ovm_FUSE_SLR_FF ovm_FUSE_F_SLRF ovm_FUSE_F_FSLR coeff
    | _, Psibar, _, Psi → invalid_arg
      "Targets.Fortran.VM:␣no␣superpotential␣here"
    | _, Chibar, _, _ | _, _, _, Chi → invalid_arg
      "Targets.Fortran.VM:␣Majorana␣spinors␣not␣handled"
    | _, Gravbar, _, _ | _, _, _, Grav → invalid_arg
      "Targets.Fortran.VM:␣Gravitinos␣not␣handled"
```

```
let children2 rhs =
  match F.children rhs with
  | [wf1; wf2] → (wf1, wf2)
  | _ → failwith "Targets.children2:␣can't␣happen"

let children3 rhs =
  match F.children rhs with
  | [wf1; wf2; wf3] → (wf1, wf2, wf3)
  | _ → invalid_arg "Targets.children3:␣can't␣happen"

let print_vector4 c lhs wf1 wf2 wf3 fusion (coeff, contraction) =
  let printc r1 r2 r3 = printi ovm_FUSE_V_VVV ~lhs : lhs ~coupl : c
    ~coeff : coeff ~rhs1 : r1 ~rhs2 : r2 ~rhs3 : r3 in
  match contraction, fusion with
  | C_12_34, (F341 | F431 | F342 | F432 | F123 | F213 | F124 | F214)
  | C_13_42, (F241 | F421 | F243 | F423 | F132 | F312 | F134 | F314)
  | C_14_23, (F231 | F321 | F234 | F324 | F142 | F412 | F143 |
F413) →
      printc wf1 wf2 wf3
  | C_12_34, (F134 | F143 | F234 | F243 | F312 | F321 | F412 | F421)
  | C_13_42, (F124 | F142 | F324 | F342 | F213 | F231 | F413 | F431)
  | C_14_23, (F123 | F132 | F423 | F432 | F214 | F241 | F314 |
F341) →
      printc wf2 wf3 wf1
  | C_12_34, (F314 | F413 | F324 | F423 | F132 | F231 | F142 | F241)
  | C_13_42, (F214 | F412 | F234 | F432 | F123 | F321 | F143 | F341)
  | C_14_23, (F213 | F312 | F243 | F342 | F124 | F421 | F134 |
F431) →
      printc wf1 wf3 wf2

let print_current lookups lhs amplitude rhs =
  let f = mult_wf lookups.dict amplitude in
  match F.coupling rhs with
  | V3 (vertex, fusion, constant) →
      let ch1, ch2 = children2 rhs in
      let wf1 = wf_index lookups.wfmap lookups.n_wfs (f ch1)
      and wf2 = wf_index lookups.wfmap lookups.n_wfs (f ch2)
      and p1 = mom_ID lookups.pmap ch1
      and p2 = mom_ID lookups.pmap ch2
      and const_ID = get_const_ID lookups.cmap constant in
      let c = if (F.sign rhs) < 0 then - const_ID else const_ID in
      begin match vertex with
      | FBF (coeff, fb, b, f) →
          begin match coeff, fb, b, f with
          | _, Psibar, VLRM, Psi | _, Psibar, SPM, Psi
          | _, Psibar, TVA, Psi | _, Psibar, TVAM, Psi
          | _, Psibar, TLR, Psi | _, Psibar, TLRM, Psi
          | _, Psibar, TRL, Psi | _, Psibar, TRLM, Psi → failwith
    "print_current:␣V3:␣Momentum␣dependent␣fermion␣couplings␣not␣implemented"

          | _, _, _, _ →
              ferm_print_current (coeff, fb, b, f) lhs c wf1 wf2 fusion
```

```
        end
    | PBP (_, _, _, _) →
        failwith "print_current: V3: PBP not implemented"
    | BBB (_, _, _, _) →
        failwith "print_current: V3: BBB not implemented"
    | GBG (_, _, _, _) →
        failwith "print_current: V3: GBG not implemented"

    | Gauge_Gauge_Gauge coeff →
        let printc r1 r2 r3 r4 = printi ovm_FUSE_G_GG
          ~lhs : lhs ~coupl : c ~coeff : coeff ~rhs1 : r1 ~rhs2 : r2 ~rhs3 :
r3
          ~rhs4 : r4 in
        begin match fusion with
        | (F23 | F31 | F12) → printc wf1 p1 wf2 p2
        | (F32 | F13 | F21) → printc wf2 p2 wf1 p1
        end

    | Aux_Gauge_Gauge _ →
        failwith "print_current: V3: not implemented"

    | Scalar_Vector_Vector coeff →
        let printc code r1 r2 = printi code
          ~lhs : lhs ~coupl : c ~coeff : coeff ~rhs1 : r1 ~rhs2 : r2 in
        begin match fusion with
        | (F23 | F32) → printc ovm_FUSE_S_VV wf1 wf2
        | (F12 | F13) → printc ovm_FUSE_V_SV wf1 wf2
        | (F21 | F31) → printc ovm_FUSE_V_SV wf2 wf1
        end

    | Scalar_Scalar_Scalar coeff →
        printi ovm_FUSE_S_SS ~lhs : lhs ~coupl : c ~coeff : coeff ~rhs1 :
wf1 ~rhs2 : wf2

    | Vector_Scalar_Scalar coeff →
        let printc code ?flip : (f = 1) r1 r2 r3 r4 = printi code
          ~lhs : lhs ~coupl : (c × f) ~coeff : coeff ~rhs1 : r1 ~rhs2 :
r2 ~rhs3 : r3
          ~rhs4 : r4 in
        begin match fusion with
        | F23 → printc ovm_FUSE_V_SS wf1 p1 wf2 p2
        | F32 → printc ovm_FUSE_V_SS wf2 p2 wf1 p1
        | F12 → printc ovm_FUSE_S_VS wf1 p1 wf2 p2
        | F21 → printc ovm_FUSE_S_VS wf2 p2 wf1 p1
        | F13 → printc ovm_FUSE_S_VS wf1 p1 wf2 p2 ~flip : (−1)
        | F31 → printc ovm_FUSE_S_VS wf2 p2 wf1 p1 ~flip : (−1)
        end

    | Aux_Vector_Vector _ →
        failwith "print_current: V3: not implemented"

    | Aux_Scalar_Scalar _ →
        failwith "print_current: V3: not implemented"
```

```
| Aux_Scalar_Vector _ →
    failwith "print_current:␣V3:␣not␣implemented"
| Graviton_Scalar_Scalar _ →
    failwith "print_current:␣V3:␣not␣implemented"
| Graviton_Vector_Vector _ →
    failwith "print_current:␣V3:␣not␣implemented"
| Graviton_Spinor_Spinor _ →
    failwith "print_current:␣V3:␣not␣implemented"
| Dim4_Vector_Vector_Vector_T _ →
    failwith "print_current:␣V3:␣not␣implemented"
| Dim4_Vector_Vector_Vector_L _ →
    failwith "print_current:␣V3:␣not␣implemented"
| Dim6_Gauge_Gauge_Gauge _ →
    failwith "print_current:␣V3:␣not␣implemented"
| Dim4_Vector_Vector_Vector_T5 _ →
    failwith "print_current:␣V3:␣not␣implemented"
| Dim4_Vector_Vector_Vector_L5 _ →
    failwith "print_current:␣V3:␣not␣implemented"
| Dim6_Gauge_Gauge_Gauge_5 _ →
    failwith "print_current:␣V3:␣not␣implemented"
| Aux_DScalar_DScalar _ →
    failwith "print_current:␣V3:␣not␣implemented"
| Aux_Vector_DScalar _ →
    failwith "print_current:␣V3:␣not␣implemented"
| Dim5_Scalar_Gauge2 coeff →
    let printc code r1 r2 r3 r4 = printi code
        ~lhs : lhs ~coupl : c ~coeff : coeff ~rhs1 : r1 ~rhs2 : r2 ~rhs3 :
r3
        ~rhs4 : r4 in
    begin match fusion with
    | (F23 | F32) → printc ovm_FUSE_S_G2 wf1 p1 wf2 p2
    | (F12 | F13) → printc ovm_FUSE_G_SG wf1 p1 wf2 p2
    | (F21 | F31) → printc ovm_FUSE_G_GS wf2 p2 wf1 p1
    end
| Dim5_Scalar_Gauge2_Skew coeff →
    let printc code ?flip : (f = 1) r1 r2 r3 r4 = printi code
        ~lhs : lhs ~coupl : (c × f) ~coeff : coeff ~rhs1 : r1 ~rhs2 :
r2 ~rhs3 : r3
        ~rhs4 : r4 in
    begin match fusion with
    | (F23 | F32) → printc ovm_FUSE_S_G2_SKEW wf1 p1 wf2 p2
    | (F12 | F13) → printc ovm_FUSE_G_SG_SKEW wf1 p1 wf2 p2
    | (F21 | F31) → printc ovm_FUSE_G_GS_SKEW wf2 p1 wf1 p2 ~flip :
(−1)
```

```
        end
    | Dim5_Scalar_Vector_Vector_T _ →
        failwith "print_current:␣V3:␣not␣implemented"
    | Dim5_Scalar_Vector_Vector_U _ →
        failwith "print_current:␣V3:␣not␣implemented"
    | Dim6_Vector_Vector_Vector_T _ →
        failwith "print_current:␣V3:␣not␣implemented"
    | Tensor_2_Vector_Vector _ →
        failwith "print_current:␣V3:␣not␣implemented"
    | Dim5_Tensor_2_Vector_Vector_1 _ →
        failwith "print_current:␣V3:␣not␣implemented"
    | Dim5_Tensor_2_Vector_Vector_2 _ →
        failwith "print_current:␣V3:␣not␣implemented"
    | Dim7_Tensor_2_Vector_Vector_T _ →
        failwith "print_current:␣V3:␣not␣implemented"
    | Dim5_Scalar_Vector_Vector_TU _ →
        failwith "print_current:␣V3:␣not␣implemented"
    | Scalar_Vector_Vector_t _ →
        failwith "print_current:␣V3:␣not␣implemented"
    | Tensor_2_Vector_Vector_1 _ →
        failwith "print_current:␣V3:␣not␣implemented"
    | Tensor_2_Vector_Vector_t _ →
        failwith "print_current:␣V3:␣not␣implemented"
    end
```

Flip the sign in c to account for the i^2 relative to diagrams with only cubic couplings.

```
| V4 (vertex, fusion, constant) →
    let ch1, ch2, ch3 = children3 rhs in
    let wf1 = wf_index lookups.wfmap lookups.n_wfs (f ch1)
    and wf2 = wf_index lookups.wfmap lookups.n_wfs (f ch2)
    and wf3 = wf_index lookups.wfmap lookups.n_wfs (f ch3)
              and const_ID = get_const_ID lookups.cmap constant in
    let c =
      if (F.sign rhs) < 0 then const_ID else - const_ID in
    begin match vertex with
    | Scalar4 coeff →
        printi ovm_FUSE_S_SSS ~lhs : lhs ~coupl : c ~coeff : coeff ~rhs1 :
wf1
          ~rhs2 : wf2 ~rhs3 : wf3
    | Scalar2_Vector2 coeff →
        let printc code r1 r2 r3 = printi code
          ~lhs : lhs ~coupl : c ~coeff : coeff ~rhs1 : r1 ~rhs2 : r2 ~rhs3 :
r3 in
```

```
          begin match fusion with
          | F134 | F143 | F234 | F243 →
              printc ovm_FUSE_S_SVV wf1 wf2 wf3
          | F314 | F413 | F324 | F423 →
              printc ovm_FUSE_S_SVV wf2 wf1 wf3
          | F341 | F431 | F342 | F432 →
              printc ovm_FUSE_S_SVV wf3 wf1 wf2
          | F312 | F321 | F412 | F421 →
              printc ovm_FUSE_V_SSV wf2 wf3 wf1
          | F231 | F132 | F241 | F142 →
              printc ovm_FUSE_V_SSV wf1 wf3 wf2
          | F123 | F213 | F124 | F214 →
              printc ovm_FUSE_V_SSV wf1 wf2 wf3
          end

      | Vector4 contractions →
          List.iter (print_vector4 c lhs wf1 wf2 wf3 fusion) contractions

      | Vector4_K_Matrix_tho _
      | Vector4_K_Matrix_jr _ →
          failwith "print_current: V4: K_Matrix not implemented"

      | GBBG _ →
          failwith "print_current: V4: GBBG not implemented"

      | DScalar4 _
      | DScalar2_Vector2 _ →
          failwith "print_current: V4: DScalars not implemented"
      end

  | Vn (_, _, _) → invalid_arg "Targets.print_current: n-ary fusion."
```

Fusions

```
let print_fusion lookups lhs_momID fusion amplitude =
  if F.on_shell amplitude (F.lhs fusion) then
    failwith "print_fusion: on_shell projectors not implemented!";
  if F.is_gauss amplitude (F.lhs fusion) then
    failwith "print_fusion: gauss amplitudes not implemented!";
  let lhs_wf = mult_wf lookups.dict amplitude (F.lhs fusion) in
  let lhs_wfID = wf_index lookups.wfmap lookups.n_wfs lhs_wf in
  let f = F.flavor (F.lhs fusion) in
  let pdg = CM.pdg f in
  let w =
    begin match CM.width f with
    | Vanishing | Fudged → 0
    | Constant → 1
    | Timelike → 2
    | Running → failwith "Targets.VM: running width not available"
    | Custom _ → failwith "Targets.VM: custom width not available"
    end
  in
```

```
let propagate code = printi code ~lhs : lhs_wfID ~rhs1 : lhs_momID
  ~coupl : (abs(pdg)) ~coeff : w ~rhs4 : (get_ID' amp_compare lookups.amap amplitude)
in
begin match CM.propagator f with
| Prop_Scalar →
    propagate ovm_PROPAGATE_SCALAR
| Prop_Col_Scalar →
    propagate ovm_PROPAGATE_COL_SCALAR
| Prop_Ghost →
    propagate ovm_PROPAGATE_GHOST
| Prop_Spinor →
    propagate ovm_PROPAGATE_SPINOR
| Prop_ConjSpinor →
    propagate ovm_PROPAGATE_CONJSPINOR
| Prop_Majorana →
    propagate ovm_PROPAGATE_MAJORANA
| Prop_Col_Majorana →
    propagate ovm_PROPAGATE_COL_MAJORANA
| Prop_Unitarity →
    propagate ovm_PROPAGATE_UNITARITY
| Prop_Col_Unitarity →
    propagate ovm_PROPAGATE_COL_UNITARITY
| Prop_Feynman →
    propagate ovm_PROPAGATE_FEYNMAN
| Prop_Col_Feynman →
    propagate ovm_PROPAGATE_COL_FEYNMAN
| Prop_Vectorspinor →
    propagate ovm_PROPAGATE_VECTORSPINOR
| Prop_Tensor_2 →
    propagate ovm_PROPAGATE_TENSOR2
| Aux_Col_Scalar | Aux_Col_Vector | Aux_Col_Tensor_1 →
    failwith "print_fusion: Aux_Col_* not implemented!"
| Aux_Vector | Aux_Tensor_1 | Aux_Scalar | Aux_Spinor |
Aux_ConjSpinor
| Aux_Majorana | Only_Insertion →
    propagate ovm_PROPAGATE_NONE
| Prop_Gauge _ →
    failwith "print_fusion: Prop_Gauge not implemented!"
| Prop_Rxi _ →
    failwith "print_fusion: Prop_Rxi not implemented!"
end;
```

Since the OVM knows that we want to propagate a wf, we can send the necessary fusions now.

```
List.iter (print_current lookups lhs_wfID amplitude) (F.rhs fusion)
```

```
let print_all_fusions lookups =
  let fusions = CF.fusions lookups.amplitudes in
  let fset = List.fold_left (fun s x → FSet.add x s) FSet.empty fusions in
  ignore (List.fold_left (fun level (f, amplitude) →
    let wf = F.lhs f in
```

```
      let lhs_momID = mom_ID lookups.pmap wf in
      let level' = List.length (F.momentum_list wf) in
      if (level' > level ∧ level' > 2) then break ();
      print_fusion lookups lhs_momID f amplitude;
      level')
    1 (FSet.elements fset) )
```

Brakets

```
let print_braket lookups amplitude braket =
  let bra = F.bra braket
  and ket = F.ket braket in
  let braID = wf_index lookups.wfmap lookups.n_wfs
    (mult_wf lookups.dict amplitude bra) in
  List.iter (print_current lookups braID amplitude) ket
```

$$\mathrm{i}T = \mathrm{i}^{\#\mathrm{vertices}}\mathrm{i}^{\#\mathrm{propagators}}\cdots = \mathrm{i}^{n-2}\mathrm{i}^{n-3}\cdots = -\mathrm{i}(-1)^n\cdots \tag{15.1}$$

All brakets for one cflow amplitude should be calculated by one thread to avoid multiple access on the same memory (amplitude).

```
let print_brakets lookups (amplitude, i) =
  let n = List.length (F.externals amplitude) in
  let sign = if n mod 2 = 0 then -1 else 1
  and sym = F.symmetry amplitude in
  printi ovm_CALC_BRAKET ~lhs : i ~rhs1 : sym ~coupl : sign;
  amplitude |> F.brakets |> List.iter (print_braket lookups amplitude)
```

Fortran arrays/OCaml lists start on 1/0. The amplitude list is sorted by *amp_compare* according to their color flows. In this way the amp array is sorted in the same way as *table_color_factors*.

```
let print_all_brakets lookups =
  let g i elt = print_brakets lookups (elt, i + 1) in
  lookups.amplitudes |> CF.processes |> List.sort amp_compare
                     —> ThoList.iteri g 0
```

Couplings

For now we only care to catch the arrays *gncneu*, *gnclep*, *gncup* and *gncdown* of the SM. This will need an overhaul when it is clear how we store the type information of coupling constants.

```
let strip_array_tag = function
  | Real_Array x → x
  | Complex_Array x → x

let array_constants_list =
  let params = M.parameters()
  and strip_to_constant (lhs, _) = strip_array_tag lhs in
    List.map strip_to_constant params.derived_arrays
```

```
let is_array x = List.mem x array_constants_list

let constants_map =
  let first = fun (x, _, _) → x in
  let second = fun (_, y, _) → y in
  let third = fun (_, _, z) → z in
  let v3 = List.map third (first (M.vertices () ))
  and v4 = List.map third (second (M.vertices () )) in
  let set = List.fold_left (fun s x → CSet.add x s) CSet.empty (v3 @ v4) in
  let (arrays, singles) = CSet.partition is_array set in
    (singles |> CSet.elements |> map_of_list,
     arrays |> CSet.elements |> map_of_list)
```

Output calls

```
let amplitudes_to_channel (cmdline : string) (oc : out_channel)
  (diagnostics : (diagnostic × bool) list ) (amplitudes : CF.amplitudes) =

  set_formatter_out_channel oc;
  if (num_particles amplitudes = 0) then begin
    print_description cmdline;
    print_zero_header (); nl ()
  end else begin
    let (wfset, amap) = wfset_amps amplitudes in
    let pset = expand_pset (momenta_set wfset)
    and n_wfs = num_wfs wfset in
    let wfmap = wf_map_of_list (WFSet.elements wfset)
    and pmap = map_of_list (ISet.elements pset)
    and cmap = constants_map in

    let lookups = {pmap = pmap; wfmap = wfmap; cmap = cmap; amap = amap;
      n_wfs = n_wfs; amplitudes = amplitudes;
      dict = CF.dictionary amplitudes} in

    print_description cmdline;
    print_header lookups wfset;
    print_spin_table amplitudes;
    print_flavor_tables amplitudes;
    print_color_tables amplitudes;
    printf "@\n%s" ("OVM␣instructions␣for␣momenta␣addition," ^
                    "␣fusions␣and␣brakets␣start␣here:␣");
    break ();
    add_all_mom lookups pset;
    print_ext_amps lookups;
    break ();
    print_all_fusions lookups;
    break ();
    print_all_brakets lookups;
    break (); nl ();
    print_flush ()
  end
```

```
let parameters_to_fortran oc _ =
      set_formatter_out_channel oc;
  let arrays_to_set = ¬ (IMap.is_empty (snd constants_map)) in
  let set_coupl ty dim cmap = IMap.iter (fun key elt →
    printf "    %s(%s%d) = %s" ty dim key (M.constant_symbol elt);
    nl () ) cmap in
  let declarations () =
    printf "  complex(%s), dimension(%d) :: ovm_coupl_cmplx"
      !kind (constants_map |> fst |> largest_key); nl ();
    if arrays_to_set then
      printf "  complex(%s), dimension(2, %d) :: ovm_coupl_cmplx2"
        !kind (constants_map |> snd |> largest_key); nl () in
  let print_line str = printf "%s" str; nl() in
  let print_md5sum = function
      | Some s →
        print_line "  function md5sum ()";
        print_line "    character(len=32) :: md5sum";
        print_line ("    bytecode_file = '" ^ !bytecode_file ^ "'");
        print_line "    call initialize_vm (vm, bytecode_file)";
        print_line "    ! DON'T EVEN THINK of modifying the following line!";
        print_line ("    md5sum = '" ^ s ^ "'");
        print_line "  end function md5sum";
      | None → ()
  in
  let print_inquiry_function_openmp () = begin
    print_line "  pure function openmp_supported () result (status)";
    print_line "    logical :: status";
    print_line ("    status = " ^ (if !openmp then ".true." else ".false."));
    print_line "  end function openmp_supported";
    nl ()
  end in
  let print_interface whizard =
  if whizard then begin
    print_line "  subroutine init (par)";
    print_line "    real(kind=default), dimension(*), intent(in) :: par";
    print_line ("     bytecode_file = '" ^ !bytecode_file ^ "'");
    print_line "    call import_from_whizard (par)";
    print_line "    call initialize_vm (vm, bytecode_file)";
    print_line "  end subroutine init";
    nl ();
    print_line "  subroutine final ()";
    print_line "    call vm%final ()";
    print_line "  end subroutine final";
    nl ();
    print_line "  subroutine update_alpha_s (alpha_s)";
    print_line ("    real(kind=" ^ !kind ^ "), intent(in) :: alpha_s");
    print_line "    call model_update_alpha_s (alpha_s)";
    print_line "  end subroutine update_alpha_s";
    nl ()
  end
```

```
else begin
  print_line "␣␣subroutine␣init␣()";
  print_line ("␣␣␣␣␣bytecode_file␣=␣'" ^ !bytecode_file ^ "'");
  print_line "␣␣␣␣␣call␣init_parameters␣()";
  print_line "␣␣␣␣␣call␣initialize_vm␣(vm,␣bytecode_file)";
  print_line "␣␣end␣subroutine"
end in
let print_lookup_functions () = begin
  print_line "␣␣pure␣function␣number_particles_in␣()␣result␣(n)";
  print_line "␣␣␣␣integer␣::␣n";
  print_line "␣␣␣␣n␣=␣vm%number_particles_in␣()";
  print_line "␣␣end␣function␣number_particles_in";
  nl();
  print_line "␣␣pure␣function␣number_particles_out␣()␣result␣(n)";
  print_line "␣␣␣␣integer␣::␣n";
  print_line "␣␣␣␣n␣=␣vm%number_particles_out␣()";
  print_line "␣␣end␣function␣number_particles_out";
  nl();
  print_line "␣␣pure␣function␣number_spin_states␣()␣result␣(n)";
  print_line "␣␣␣␣integer␣::␣n";
  print_line "␣␣␣␣n␣=␣vm%number_spin_states␣()";
  print_line "␣␣end␣function␣number_spin_states";
  nl();
  print_line "␣␣pure␣subroutine␣spin_states␣(a)";
  print_line "␣␣␣␣integer,␣dimension(:,:),␣intent(out)␣::␣a";
  print_line "␣␣␣␣call␣vm%spin_states␣(a)";
  print_line "␣␣end␣subroutine␣spin_states";
  nl();
  print_line "␣␣pure␣function␣number_flavor_states␣()␣result␣(n)";
  print_line "␣␣␣␣integer␣::␣n";
  print_line "␣␣␣␣n␣=␣vm%number_flavor_states␣()";
  print_line "␣␣end␣function␣number_flavor_states";
  nl();
  print_line "␣␣pure␣subroutine␣flavor_states␣(a)";
  print_line "␣␣␣␣integer,␣dimension(:,:),␣intent(out)␣::␣a";
  print_line "␣␣␣␣call␣vm%flavor_states␣(a)";
  print_line "␣␣end␣subroutine␣flavor_states";
  nl();
  print_line "␣␣pure␣function␣number_color_indices␣()␣result␣(n)";
  print_line "␣␣␣␣integer␣::␣n";
  print_line "␣␣␣␣n␣=␣vm%number_color_indices␣()";
  print_line "␣␣end␣function␣number_color_indices";
  nl();
  print_line "␣␣pure␣function␣number_color_flows␣()␣result␣(n)";
  print_line "␣␣␣␣integer␣::␣n";
  print_line "␣␣␣␣n␣=␣vm%number_color_flows␣()";
  print_line "␣␣end␣function␣number_color_flows";
  nl();
  print_line "␣␣pure␣subroutine␣color_flows␣(a,␣g)";
  print_line "␣␣␣␣integer,␣dimension(:,:,:),␣intent(out)␣::␣a";
```

```
print_line "␣␣␣␣logical,␣dimension(:,:),␣intent(out)␣::␣g";
print_line "␣␣␣␣call␣vm%color_flows␣(a,␣g)";
print_line "␣␣end␣subroutine␣color_flows";
nl();
print_line "␣␣pure␣function␣number_color_factors␣()␣result␣(n)";
print_line "␣␣␣␣integer␣::␣n";
print_line "␣␣␣␣n␣=␣vm%number_color_factors␣()";
print_line "␣␣end␣function␣number_color_factors";
nl();
print_line "␣␣pure␣subroutine␣color_factors␣(cf)";
print_line "␣␣␣␣use␣omega_color";
print_line "␣␣␣␣type(omega_color_factor),␣dimension(:),␣intent(out)␣::␣cf";
print_line "␣␣␣␣call␣vm%color_factors␣(cf)";
print_line "␣␣end␣subroutine␣color_factors";
nl();
print_line "␣␣!pure␣unless␣OpenMP";
print_line "␣␣!pure␣function␣color_sum␣(flv,␣hel)␣result␣(amp2)";
print_line "␣␣function␣color_sum␣(flv,␣hel)␣result␣(amp2)";
print_line "␣␣␣␣use␣kinds";
print_line "␣␣␣␣integer,␣intent(in)␣::␣flv,␣hel";
print_line "␣␣␣␣real(kind=default)␣::␣amp2";
print_line "␣␣␣␣amp2␣=␣vm%color_sum␣(flv,␣hel)";
print_line "␣␣end␣function␣color_sum";
nl();
print_line "␣␣subroutine␣new_event␣(p)";
print_line "␣␣␣␣use␣kinds";
print_line "␣␣␣␣real(kind=default),␣dimension(0:3,*),␣intent(in)␣::␣p";
print_line "␣␣␣␣call␣vm%new_event␣(p)";
print_line "␣␣end␣subroutine␣new_event";
nl();
print_line "␣␣subroutine␣reset_helicity_selection␣(threshold,␣cutoff)";
print_line "␣␣␣␣use␣kinds";
print_line "␣␣␣␣real(kind=default),␣intent(in)␣::␣threshold";
print_line "␣␣␣␣integer,␣intent(in)␣::␣cutoff";
print_line "␣␣␣␣call␣vm%reset_helicity_selection␣(threshold,␣cutoff)";
print_line "␣␣end␣subroutine␣reset_helicity_selection";
nl();
print_line "␣␣pure␣function␣is_allowed␣(flv,␣hel,␣col)␣result␣(yorn)";
print_line "␣␣␣␣logical␣::␣yorn";
print_line "␣␣␣␣integer,␣intent(in)␣::␣flv,␣hel,␣col";
print_line "␣␣␣␣yorn␣=␣vm%is_allowed␣(flv,␣hel,␣col)";
print_line "␣␣end␣function␣is_allowed";
nl();
print_line "␣␣pure␣function␣get_amplitude␣(flv,␣hel,␣col)␣result␣(amp_result)";
print_line "␣␣␣␣use␣kinds";
print_line "␣␣␣␣complex(kind=default)␣::␣amp_result";
print_line "␣␣␣␣integer,␣intent(in)␣::␣flv,␣hel,␣col";
print_line "␣␣␣␣amp_result␣=␣vm%get_amplitude(flv,␣hel,␣col)";
print_line "␣␣end␣function␣get_amplitude";
nl();
```

```
end in
let rcs_str s = String.sub s 0 (String.length s − 1) in
let rcs_tags = rcs_list @ [M.rcs] |> List.map version_string
                                  —> List.map rcs_str in

print_line ("module␣" ^ !wrapper_module);
print_line ("␣␣use␣" ^ !parameter_module_external);
print_line "␣␣use␣iso_varying_string,␣string_t␣=>␣varying_string";
print_line "␣␣use␣kinds";
print_line "␣␣use␣omegavm95";
print_line "␣␣implicit␣none";
print_line "␣␣private";
print_line "␣␣type(vm_t)␣::␣vm";
print_line "␣␣type(string_t)␣::␣bytecode_file";
print_line ("␣␣public␣::␣number_particles_in,␣number_particles_out," ^
    "␣number_spin_states,␣&");
print_line ("␣␣␣␣spin_states,␣number_flavor_states,␣flavor_states," ^
    "␣number_color_indices,␣&");
print_line ("␣␣␣␣number_color_flows,␣color_flows," ^
    "␣number_color_factors,␣color_factors,␣&");
print_line ("␣␣␣␣color_sum,␣new_event,␣reset_helicity_selection," ^
    "␣is_allowed,␣get_amplitude,␣&");
print_line ("␣␣␣␣init,␣" ^
    (match !md5sum with Some _ → "md5sum,␣"
                      | None → "") ^ "openmp_supported");
if !whizard then
  print_line ("␣␣public␣::␣final,␣update_alpha_s")
else
  print_line ("␣␣public␣::␣initialize_vm");
declarations ();
print_line "contains";

print_line "␣␣subroutine␣setup_couplings␣()";
set_coupl "ovm_coupl_cmplx" "" (fst constants_map);
if arrays_to_set then
  set_coupl "ovm_coupl_cmplx2" ":," (snd constants_map);
print_line "␣␣end␣subroutine␣setup_couplings";
print_line "␣␣subroutine␣initialize_vm␣(vm,␣bytecode_file)";
print_line "␣␣␣␣class(vm_t),␣intent(out)␣::␣vm";
print_line "␣␣␣␣type(string_t),␣intent(in)␣::␣bytecode_file";
print_line "␣␣␣␣type(string_t)␣::␣version";
print_line "␣␣␣␣type(string_t)␣::␣model";
print_line ("␣␣␣␣version␣=␣'" ^ List.nth rcs_tags 0 ^ "'");
print_line ("␣␣␣␣model␣=␣'" ^ List.nth rcs_tags 1 ^ "'");
print_line "␣␣␣␣call␣setup_couplings␣()";
print_line "␣␣␣␣call␣vm%init␣(bytecode_file,␣version,␣model,␣verbose=.False.,␣&";
print_line "␣␣␣␣␣␣coupl_cmplx=ovm_coupl_cmplx,␣&";
if arrays_to_set then
  print_line "␣␣␣␣␣␣coupl_cmplx2=ovm_coupl_cmplx2,␣&";
print_line ("␣␣␣␣␣␣mass=mass,␣width=width,␣openmp=" ^ (if !openmp then
  ".true." else ".false.") ^ ")");
```

```
      print_line "  end subroutine initialize_vm";
      nl ();
      print_md5sum !md5sum;
      print_inquiry_function_openmp ();
      print_interface !whizard;
      print_lookup_functions ();

      print_line ("end module " ^ !wrapper_module);
      print_line "! O'Mega revision control information:";
      ThoList.flatmap RCS.summary (M.rcs :: rcs_list) |>
        List.iter (fun s -> printf "!    %s" s; nl ())

    let parameters_to_channel oc =
      parameters_to_fortran oc (CM.parameters ())

  end
```

15.2.2 Fortran 90/95

Dirac Fermions

We factor out the code for fermions so that we can use the simpler implementation for Dirac fermions if the model contains no Majorana fermions.

```
module type Fermions =
  sig
    open Coupling
    val psi_type : string
    val psibar_type : string
    val chi_type : string
    val grav_type : string
    val psi_incoming : string
    val brs_psi_incoming : string
    val psibar_incoming : string
    val brs_psibar_incoming : string
    val chi_incoming : string
    val brs_chi_incoming : string
    val grav_incoming : string
    val psi_outgoing : string
    val brs_psi_outgoing : string
    val psibar_outgoing : string
    val brs_psibar_outgoing : string
    val chi_outgoing : string
    val brs_chi_outgoing : string
    val grav_outgoing : string
    val psi_propagator : string
    val psibar_propagator : string
    val chi_propagator : string
    val grav_propagator : string
    val psi_projector : string
    val psibar_projector : string
```

```
    val chi_projector : string
    val grav_projector : string
    val psi_gauss : string
    val psibar_gauss : string
    val chi_gauss : string
    val grav_gauss : string
    val print_current : int × fermionbar × boson × fermion →
      string → string → string → fuse2 → unit
    val print_current_mom : int × fermionbar × boson × fermion →
      string → string → string → string → string → string
      → fuse2 → unit
    val print_current_p : int × fermion × boson × fermion →
      string → string → string → fuse2 → unit
    val print_current_b : int × fermionbar × boson × fermionbar →
      string → string → string → fuse2 → unit
    val print_current_g : int × fermionbar × boson × fermion →
      string → string → string → string → string → string
      → fuse2 → unit
    val print_current_g4 : int × fermionbar × boson2 × fermion →
      string → string → string → string → fuse3 → unit
    val reverse_braket : lorentz → bool
    val use_module : string
    val require_library : string list
    val rcs : RCS.t
  end

module Fortran_Fermions : Fermions =
  struct
    let rcs = RCS.rename rcs_file "Targets.Fortran_Fermions()"
        [ "generates␣Fortran95␣code␣for␣Dirac␣fermions";
          "using␣revision␣2000_10_A␣of␣module␣omega95" ]

    open Coupling
    open Format

    let psi_type = "spinor"
    let psibar_type = "conjspinor"
    let chi_type = "???"
    let grav_type = "???"

    let psi_incoming = "u"
    let brs_psi_incoming = "brs_u"
    let psibar_incoming = "vbar"
    let brs_psibar_incoming = "brs_vbar"
    let chi_incoming = "???"
    let brs_chi_incoming = "???"
    let grav_incoming = "???"
    let psi_outgoing = "v"
    let brs_psi_outgoing = "brs_v"
    let psibar_outgoing = "ubar"
    let brs_psibar_outgoing = "brs_ubar"
    let chi_outgoing = "???"
```

```
let brs_chi_outgoing = "???"
let grav_outgoing = "???"

let psi_propagator = "pr_psi"
let psibar_propagator = "pr_psibar"
let chi_propagator = "???"
let grav_propagator = "???"

let psi_projector = "pj_psi"
let psibar_projector = "pj_psibar"
let chi_projector = "???"
let grav_projector = "???"

let psi_gauss = "pg_psi"
let psibar_gauss = "pg_psibar"
let chi_gauss = "???"
let grav_gauss = "???"

let format_coupling coeff c =
  match coeff with
  | 1 → c
  | −1 → "(-" ^ c ^")"
  | coeff → string_of_int coeff ^ "*" ^ c

let format_coupling_2 coeff c =
  match coeff with
  | 1 → c
  | −1 → "-" ^ c
  | coeff → string_of_int coeff ^ "*" ^ c
```

JR's coupling constant HACK, necessitated by tho's bad design descition.

```
let fastener s i ?p ?q () =
  try
    let offset = (String.index s '(') in
    if ((String.get s (String.length s − 1)) ≢ ')') then
      failwith "fastener: wrong usage of parentheses"
    else
      let func_name = (String.sub s 0 offset) and
          tail =
        (String.sub s (succ offset) (String.length s − offset − 2)) in
      if (String.contains func_name ')') ∨
         (String.contains tail '(') ∨
         (String.contains tail ')') then
        failwith "fastener: wrong usage of parentheses"
      else
        func_name ^ "(" ^ string_of_int i ^ "," ^ tail ^ ")"
  with
  | Not_found →
      if (String.contains s ')') then
        failwith "fastener: wrong usage of parentheses"
      else
```

```
        match p with
        | None → s ^ "(" ^ string_of_int i ^ ")"
        | Some p →
          match q with
          | None → s ^ "(" ^ p ^ "*" ^ p ^ "," ^ string_of_int i ^ ")"
          | Some q → s ^ "(" ^ p ^ "," ^ q ^ "," ^ string_of_int i ^ ")"
let print_fermion_current coeff f c wf1 wf2 fusion =
  let c = format_coupling coeff c in
  match fusion with
  | F13 → printf "%s_ff(%s,%s,%s)" f c wf1 wf2
  | F31 → printf "%s_ff(%s,%s,%s)" f c wf2 wf1
  | F23 → printf "f_%sf(%s,%s,%s)" f c wf1 wf2
  | F32 → printf "f_%sf(%s,%s,%s)" f c wf2 wf1
  | F12 → printf "f_f%s(%s,%s,%s)" f c wf1 wf2
  | F21 → printf "f_f%s(%s,%s,%s)" f c wf2 wf1
```

Using a two element array for the combined vector-axial and scalar-pseudo couplings helps to support HELAS as well. Since we will probably never support general boson couplings with HELAS, it might be retired in favor of two separate variables. For this *Model.constant_symbol* has to be generalized.

NB: passing the array instead of two separate constants would be a *bad* idea, because the support for Majorana spinors below will have to flip signs!

```
let print_fermion_current2 coeff f c wf1 wf2 fusion =
  let c = format_coupling_2 coeff c in
  let c1 = fastener c 1 ()
  and c2 = fastener c 2 () in
  match fusion with
  | F13 → printf "%s_ff(%s,%s,%s,%s)" f c1 c2 wf1 wf2
  | F31 → printf "%s_ff(%s,%s,%s,%s)" f c1 c2 wf2 wf1
  | F23 → printf "f_%sf(%s,%s,%s,%s)" f c1 c2 wf1 wf2
  | F32 → printf "f_%sf(%s,%s,%s,%s)" f c1 c2 wf2 wf1
  | F12 → printf "f_f%s(%s,%s,%s,%s)" f c1 c2 wf1 wf2
  | F21 → printf "f_f%s(%s,%s,%s,%s)" f c1 c2 wf2 wf1
let print_fermion_current_mom_v1 coeff f c wf1 wf2 p1 p2 p12 fusion =
  let c = format_coupling coeff c in
  let c1 = fastener c 1 and
      c2 = fastener c 2 in
  match fusion with
  | F13 → printf "%s_ff(%s,%s,%s,%s)" f (c1 ~p : p12 ()) (c2 ~p :
p12 ()) wf1 wf2
  | F31 → printf "%s_ff(%s,%s,%s,%s)" f (c1 ~p : p12 ()) (c2 ~p :
p12 ()) wf2 wf1
  | F23 → printf "f_%sf(%s,%s,%s,%s)" f (c1 ~p : p1 ()) (c2 ~p :
p1 ()) wf1 wf2
  | F32 → printf "f_%sf(%s,%s,%s,%s)" f (c1 ~p : p2 ()) (c2 ~p :
p2 ()) wf2 wf1
```

```
    | F12 → printf "f_f%s(%s,%s,%s,%s)" f (c1 ~p : p2 ()) (c2 ~p :
p2 ()) wf1 wf2
    | F21 → printf "f_f%s(%s,%s,%s,%s)" f (c1 ~p : p1 ()) (c2 ~p :
p1 ()) wf2 wf1
  let print_fermion_current_mom_v2 coeff f c wf1 wf2 p1 p2 p12 fusion =
    let c = format_coupling coeff c in
    let c1 = fastener c 1 and
        c2 = fastener c 2 in
    match fusion with
    | F13 → printf "%s_ff(%s,%s,@,%s,%s,%s)" f (c1 ~p : p12 ()) (c2 ~p :
p12 ()) wf1 wf2 p12
    | F31 → printf "%s_ff(%s,%s,@,%s,%s,%s)" f (c1 ~p : p12 ()) (c2 ~p :
p12 ()) wf2 wf1 p12
    | F23 → printf "f_%sf(%s,%s,@,%s,%s,%s)" f (c1 ~p : p1 ()) (c2 ~p :
p1 ()) wf1 wf2 p1
    | F32 → printf "f_%sf(%s,%s,@,%s,%s,%s)" f (c1 ~p : p2 ()) (c2 ~p :
p2 ()) wf2 wf1 p2
    | F12 → printf "f_f%s(%s,%s,@,%s,%s,%s)" f (c1 ~p : p2 ()) (c2 ~p :
p2 ()) wf1 wf2 p2
    | F21 → printf "f_f%s(%s,%s,@,%s,%s,%s)" f (c1 ~p : p1 ()) (c2 ~p :
p1 ()) wf2 wf1 p1
  let print_fermion_current_mom_ff coeff f c wf1 wf2 p1 p2 p12 fusion =
    let c = format_coupling coeff c in
    let c1 = fastener c 1 and
        c2 = fastener c 2 in
    match fusion with
    | F13 → printf "%s_ff(%s,%s,%s,%s)" f (c1 ~p : p1 ~q : p2 ()) (c2 ~p :
p1 ~q : p2 ()) wf1 wf2
    | F31 → printf "%s_ff(%s,%s,%s,%s)" f (c1 ~p : p1 ~q : p2 ()) (c2 ~p :
p1 ~q : p2 ()) wf2 wf1
    | F23 → printf "f_%sf(%s,%s,%s,%s)" f (c1 ~p : p12 ~q : p2 ()) (c2 ~p :
p12 ~q : p2 ()) wf1 wf2
    | F32 → printf "f_%sf(%s,%s,%s,%s)" f (c1 ~p : p12 ~q : p1 ()) (c2 ~p :
p12 ~q : p1 ()) wf2 wf1
    | F12 → printf "f_f%s(%s,%s,%s,%s)" f (c1 ~p : p12 ~q : p1 ()) (c2 ~p :
p12 ~q : p1 ()) wf1 wf2
    | F21 → printf "f_f%s(%s,%s,%s,%s)" f (c1 ~p : p12 ~q : p2 ()) (c2 ~p :
p12 ~q : p2 ()) wf2 wf1
  let print_current = function
    | coeff, Psibar, VA, Psi → print_fermion_current2 coeff "va"
    | coeff, Psibar, VA2, Psi → print_fermion_current coeff "va2"
    | coeff, Psibar, VA3, Psi → print_fermion_current coeff "va3"
    | coeff, Psibar, V, Psi → print_fermion_current coeff "v"
    | coeff, Psibar, A, Psi → print_fermion_current coeff "a"
    | coeff, Psibar, VL, Psi → print_fermion_current coeff "vl"
    | coeff, Psibar, VR, Psi → print_fermion_current coeff "vr"
    | coeff, Psibar, VLR, Psi → print_fermion_current2 coeff "vlr"
    | coeff, Psibar, SP, Psi → print_fermion_current2 coeff "sp"
    | coeff, Psibar, S, Psi → print_fermion_current coeff "s"
```

```
  | coeff, Psibar, P, Psi → print_fermion_current coeff "p"
  | coeff, Psibar, SL, Psi → print_fermion_current coeff "sl"
  | coeff, Psibar, SR, Psi → print_fermion_current coeff "sr"
  | coeff, Psibar, SLR, Psi → print_fermion_current2 coeff "slr"
  | _, Psibar, _, Psi → invalid_arg
      "Targets.Fortran_Fermions:␣no␣superpotential␣here"
  | _, Chibar, _, _ | _, _, _, Chi → invalid_arg
      "Targets.Fortran_Fermions:␣Majorana␣spinors␣not␣handled"
  | _, Gravbar, _, _ | _, _, _, Grav → invalid_arg
      "Targets.Fortran_Fermions:␣Gravitinos␣not␣handled"

let print_current_mom = function
  | coeff, Psibar, VLRM, Psi → print_fermion_current_mom_v1 coeff "vlr"
  | coeff, Psibar, VAM, Psi → print_fermion_current_mom_ff coeff "va"
  | coeff, Psibar, VA3M, Psi → print_fermion_current_mom_ff coeff "va3"
  | coeff, Psibar, SPM, Psi → print_fermion_current_mom_v1 coeff "sp"
  | coeff, Psibar, TVA, Psi → print_fermion_current_mom_v1 coeff "tva"
  | coeff, Psibar, TVAM, Psi → print_fermion_current_mom_v2 coeff "tvam"
  | coeff, Psibar, TLR, Psi → print_fermion_current_mom_v1 coeff "tlr"
  | coeff, Psibar, TLRM, Psi → print_fermion_current_mom_v2 coeff "tlrm"
  | coeff, Psibar, TRL, Psi → print_fermion_current_mom_v1 coeff "trl"
  | coeff, Psibar, TRLM, Psi → print_fermion_current_mom_v2 coeff "trlm"
  | _, Psibar, _, Psi → invalid_arg
      "Targets.Fortran_Fermions:␣only␣sigma␣tensor␣coupling␣here"
  | _, Chibar, _, _ | _, _, _, Chi → invalid_arg
      "Targets.Fortran_Fermions:␣Majorana␣spinors␣not␣handled"
  | _, Gravbar, _, _ | _, _, _, Grav → invalid_arg
      "Targets.Fortran_Fermions:␣Gravitinos␣not␣handled"

let print_current_p = function
  | _, _, _, _ → invalid_arg
      "Targets.Fortran_Fermions:␣No␣clashing␣arrows␣here"

let print_current_b = function
  | _, _, _, _ → invalid_arg
      "Targets.Fortran_Fermions:␣No␣clashing␣arrows␣here"

let print_current_g = function
  | _, _, _, _ → invalid_arg
      "Targets.Fortran_Fermions:␣No␣gravitinos␣here"

let print_current_g4 = function
  | _, _, _, _ → invalid_arg
      "Targets.Fortran_Fermions:␣No␣gravitinos␣here"

let reverse_braket = function
  | Spinor → true
  | _ → false

let use_module = "omega95"
let require_library =
  ["omega_spinors_2010_01_A"; "omega_spinor_cpls_2010_01_A"]
end
```

Main Functor

```
module Make_Fortran (Fermions : Fermions)
    (Fusion_Maker : Fusion.Maker) (P : Momentum.T) (M : Model.T) =
  struct
    let rcs_list =
      [ RCS.rename rcs_file "Targets.Make_Fortran()"
          [ "Interface for Whizard 2.X";
            "NB: non-gauge vector couplings are not available yet" ];
        Fermions.rcs ]

    let require_library =
      Fermions.require_library @
      [ "omega_vectors_2010_01_A"; "omega_polarizations_2010_01_A";
        "omega_couplings_2010_01_A"; "omega_color_2010_01_A";
        "omega_utils_2010_01_A" ]

    module CM = Colorize.It(M)
    module F = Fusion_Maker(P)(M)

    module CF = Fusion.Multi(Fusion_Maker)(P)(M)
    type amplitudes = CF.amplitudes

    open Coupling
    open Format

    type output_mode =
      | Single_Function
      | Single_Module of int
      | Single_File of int
      | Multi_File of int

    let line_length = ref 80
    let continuation_lines = ref (-1) (* 255 *)
    let kind = ref "default"
    let fortran95 = ref true
    let module_name = ref "omega_amplitude"
    let output_mode = ref (Single_Module 10)
    let use_modules = ref []
    let whizard = ref false
    let parameter_module = ref ""
    let md5sum = ref None
    let no_write = ref false
    let km_write = ref false
    let km_pure = ref false
    let openmp = ref false
    let pure_unless_openmp = false

    let options = Options.create
      [ "90", Arg.Clear fortran95,
        "don't use Fortran95 features that are not in Fortran90";
        "kind", Arg.String (fun s -> kind := s),
        "real and complex kind (default: " ^ !kind ^ ")";
```

```
"width", Arg.Int (fun w → line_length := w), "maximum␣line␣length";
"continuation", Arg.Int (fun l → continuation_lines := l),
"maximum␣#␣of␣continuation␣lines";
"module", Arg.String (fun s → module_name := s), "module␣name";
"single_function", Arg.Unit (fun () → output_mode := Single_Function),
"compute␣the␣matrix␣element(s)␣in␣a␣monolithic␣function";
"split_function", Arg.Int (fun n → output_mode := Single_Module n),
"split␣the␣matrix␣element(s)␣into␣small␣functions␣[default,␣size␣=␣10]";
"split_module", Arg.Int (fun n → output_mode := Single_File n),
"split␣the␣matrix␣element(s)␣into␣small␣modules";
"split_file", Arg.Int (fun n → output_mode := Multi_File n),
"split␣the␣matrix␣element(s)␣into␣small␣files";
"use", Arg.String (fun s → use_modules := s :: !use_modules),
"use␣module";
"parameter_module", Arg.String (fun s → parameter_module := s),
"parameter_module";
"md5sum", Arg.String (fun s → md5sum := Some s),
"transfer␣MD5␣checksum";
"whizard", Arg.Set whizard, "include␣WHIZARD␣interface";
"no_write", Arg.Set no_write, "no␣'write'␣statements";
"kmatrix_write", Arg.Set km_write, "write␣K␣matrix␣functions";
"kmatrix_write_pure", Arg.Set km_pure, "write␣K␣matrix␣pure␣functions";
"openmp", Arg.Set openmp, "activate␣OpenMP␣support␣in␣generated␣code"]
```

Fortran style line continuation:
Default function to output spaces (copied from `format.ml`).

```
let blank_line = String.make 80 ' '
let rec display_blanks oc n =
  if n > 0 then
    if n ≤ 80 then
      output oc blank_line 0 n
    else begin
      output oc blank_line 0 80;
      display_blanks oc (n − 80)
    end
```

Default function to output new lines (copied from `format.ml`).

```
let display_newline oc () =
  output oc "\n" 0 1
```

current_continuation_line

- ≤ 0: not continuing: print a straight newline,
- > 0: continuing: append "␣&" until we run up to !*continuation_lines*. NB: !*continuation_lines* < 0 means *unlimited* continuation lines.

```
let current_continuation_line = ref 1
exception Continuation_Lines of int

let fortran_newline oc () =
  if !current_continuation_line > 0 then begin
```

```
    if !continuation_lines ≥ 0 ∧ !current_continuation_line > !continuation_lines then
      raise (Continuation_Lines !current_continuation_line)
    else begin
      output oc "␣&" 0 2;
      incr current_continuation_line
    end
  end;
  display_newline oc ()

let nl () =
  current_continuation_line := 0;
  print_newline ();
  current_continuation_line := 1
```

Make a formatter with default functions to output spaces and new lines.

```
let setup_fortran_formatter width oc =
  set_all_formatter_output_functions
    ~out : (output oc)
    ~flush : (fun () → flush oc)
    ~newline : (fortran_newline oc)
    ~spaces : (display_blanks oc);
  set_margin (width − 2)

let print_list = function
  | [] → ()
  | a :: rest →
      print_string a;
      List.iter (fun s → printf ",@␣%s" s) rest
```

Variables and Declarations

"NC" is already used up in the module "constants":

```
let nc_parameter = "N_"
let omega_color_factor_abbrev = "OCF"
let openmp_tld_type = "thread_local_data"
let openmp_tld = "tld"

let flavors_symbol ?(decl = false) flavors =
  (if !openmp ∧ ¬ decl then openmp_tld ^ "%" else "" ) ^
  "oks_" ^ String.concat "" (List.map CM.flavor_symbol flavors)

let p2s p =
  if p ≥ 0 ∧ p ≤ 9 then
    string_of_int p
  else if p ≤ 36 then
    String.make 1 (Char.chr (Char.code 'A' + p − 10))
  else
    "_"

let format_momentum p =
  "p" ^ String.concat "" (List.map p2s p)
```

```
let format_p wf =
  String.concat "" (List.map p2s (F.momentum_list wf))

let ext_momentum wf =
  match F.momentum_list wf with
  | [n] → n
  | _ → invalid_arg "Targets.Fortran.ext_momentum"

module PSet = Set.Make (struct type t = int list let compare = compare end)
module WFSet = Set.Make (struct type t = F.wf let compare = compare end)

let add_tag wf name =
  match F.wf_tag wf with
  | None → name
  | Some tag → name ^ "_" ^ tag

let variable ?(decl = false) wf =
  (if !openmp ∧ ¬ decl then openmp_tld ^ "%" else "")
  ^ add_tag wf ("owf_" ^ CM.flavor_symbol (F.flavor wf) ^ "_" ^ format_p wf)

let momentum wf = "p" ^ format_p wf
let spin wf = "s(" ^ string_of_int (ext_momentum wf) ^ ")"

let format_multiple_variable ?(decl = false) wf i =
  variable ~decl : decl wf ^ "_X" ^ string_of_int i

let multiple_variable ?(decl = false) amplitude dictionary wf =
  try
    format_multiple_variable ~decl : decl wf (dictionary amplitude wf)
  with
  | Not_found → variable wf

let multiple_variables ?(decl = false) multiplicity wf =
  try
    List.map
      (format_multiple_variable ~decl : decl wf)
      (ThoList.range 1 (multiplicity wf))
  with
  | Not_found → [variable ~decl : decl wf]

let declaration_chunk_size = 64

let declare_list_chunk multiplicity t = function
  | [] → ()
  | wfs →
      printf "␣␣␣␣@[<2>%s␣::␣" t;
      print_list (ThoList.flatmap (multiple_variables ~decl :true multiplicity) wfs); nl ()

let declare_list multiplicity t = function
  | [] → ()
  | wfs →
      List.iter
        (declare_list_chunk multiplicity t)
        (ThoList.chopn declaration_chunk_size wfs)

type declarations =
```

```
  { scalars : F.wf list;
    spinors : F.wf list;
    conjspinors : F.wf list;
    realspinors : F.wf list;
    ghostspinors : F.wf list;
    vectorspinors : F.wf list;
    vectors : F.wf list;
    ward_vectors : F.wf list;
    massive_vectors : F.wf list;
    tensors_1 : F.wf list;
    tensors_2 : F.wf list;
    brs_scalars : F.wf list;
    brs_spinors : F.wf list;
    brs_conjspinors : F.wf list;
    brs_realspinors : F.wf list;
    brs_vectorspinors : F.wf list;
    brs_vectors : F.wf list;
    brs_massive_vectors : F.wf list }

let rec classify_wfs' acc = function
  | [] → acc
  | wf :: rest →
      classify_wfs'
        (match CM.lorentz (F.flavor wf) with
        | Scalar → {acc with scalars = wf :: acc.scalars}
        | Spinor → {acc with spinors = wf :: acc.spinors}
        | ConjSpinor → {acc with conjspinors = wf :: acc.conjspinors}
        | Majorana → {acc with realspinors = wf :: acc.realspinors}
        | Maj_Ghost → {acc with ghostspinors = wf :: acc.ghostspinors}
        | Vectorspinor →
            {acc with vectorspinors = wf :: acc.vectorspinors}
        | Vector → {acc with vectors = wf :: acc.vectors}
        | Massive_Vector →
            {acc with massive_vectors = wf :: acc.massive_vectors}
        | Tensor_1 → {acc with tensors_1 = wf :: acc.tensors_1}
        | Tensor_2 → {acc with tensors_2 = wf :: acc.tensors_2}
        | BRS Scalar → {acc with brs_scalars = wf :: acc.brs_scalars}
        | BRS Spinor → {acc with brs_spinors = wf :: acc.brs_spinors}
        | BRS ConjSpinor → {acc with brs_conjspinors =
                                wf :: acc.brs_conjspinors}
        | BRS Majorana → {acc with brs_realspinors =
                              wf :: acc.brs_realspinors}
        | BRS Vectorspinor → {acc with brs_vectorspinors =
                                  wf :: acc.brs_vectorspinors}
        | BRS Vector → {acc with brs_vectors = wf :: acc.brs_vectors}
        | BRS Massive_Vector → {acc with brs_massive_vectors =
                                    wf :: acc.brs_massive_vectors}
        | BRS _ → invalid_arg "Targets.wfs_classify':␣not␣needed␣here")
        rest

let classify_wfs wfs = classify_wfs'
```

```
{ scalars = []; spinors = []; conjspinors = []; realspinors = [];
  ghostspinors = []; vectorspinors = []; vectors = [];
  ward_vectors = [];
  massive_vectors = []; tensors_1 = []; tensors_2 = [];
  brs_scalars = [] ; brs_spinors = []; brs_conjspinors = [];
  brs_realspinors = []; brs_vectorspinors = [];
  brs_vectors = []; brs_massive_vectors = []}
wfs
```

Parameters

```
type α parameters =
    { real_singles : α list;
      real_arrays : (α × int) list;
      complex_singles : α list;
      complex_arrays : (α × int) list }

let rec classify_singles acc = function
  | [] → acc
  | Real p :: rest → classify_singles
        { acc with real_singles = p :: acc.real_singles } rest
  | Complex p :: rest → classify_singles
        { acc with complex_singles = p :: acc.complex_singles } rest

let rec classify_arrays acc = function
  | [] → acc
  | (Real_Array p, rhs) :: rest → classify_arrays
        { acc with real_arrays =
            (p, List.length rhs) :: acc.real_arrays } rest
  | (Complex_Array p, rhs) :: rest → classify_arrays
        { acc with complex_arrays =
            (p, List.length rhs) :: acc.complex_arrays } rest

let classify_parameters params =
  classify_arrays
    (classify_singles
       { real_singles = [];
         real_arrays = [];
         complex_singles = [];
         complex_arrays = [] }
       (List.map fst params.derived)) params.derived_arrays
```

Unify this with the other code using *ThoList.chopn*.

```
let rec schisma n l =
  if List.length l ≤ n then
    [l]
  else
    let a, b = ThoList.splitn n l in
    [a] @ (schisma n b)
```

```
let rec schisma_num i n l =
  if List.length l ≤ n then
    [(i, l)]
  else
    let a, b = ThoList.splitn n l in
    [(i, a)] @ (schisma_num (i + 1) n b)

let declare_parameters' t = function
  | [] → ()
  | plist →
      printf "␣␣@[<2>%s(kind=%s),␣public,␣save␣::␣" t !kind;
      print_list (List.map CM.constant_symbol plist); nl ()

let declare_parameters t plist =
  List.iter (declare_parameters' t) plist

let declare_parameter_array t (p, n) =
  printf "␣␣@[<2>%s(kind=%s),␣dimension(%d),␣public,␣save␣::␣%s"
    t !kind n (CM.constant_symbol p); nl ()

let default_parameter (x, v) =
  printf "@␣%s␣=␣%g_%s" (CM.constant_symbol x) v !kind

let declare_default_parameters t = function
  | [] → ()
  | p :: plist →
      printf "␣␣@[<2>%s(kind=%s),␣public,␣save␣::" t !kind;
      default_parameter p;
      List.iter (fun p' → printf ","; default_parameter p') plist;
      nl ()

let format_constant = function
  | I → sprintf "cmplx␣(0.0_%s,␣1.0_%s)" !kind !kind
  | Const c when c < 0 → sprintf "(%d.0_%s)" c !kind
  | Const c → sprintf "%d.0_%s" c !kind
  | _ → invalid_arg "format_constant"

let rec eval_parameter' = function
  | I → printf "cmplx␣(0.0_%s,␣1.0_%s)" !kind !kind
  | Const c when c < 0 → printf "(%d.0_%s)" c !kind
  | Const c → printf "%d.0_%s" c !kind
  | Atom x → printf "%s" (CM.constant_symbol x)
  | Sum [] → printf "0.0_%s" !kind
  | Sum [x] → eval_parameter' x
  | Sum (x :: xs) →
      printf "@,("; eval_parameter' x;
      List.iter (fun x → printf "@,␣+␣"; eval_parameter' x) xs;
      printf ")"
  | Diff (x, y) →
      printf "@,("; eval_parameter' x;
      printf "␣-␣"; eval_parameter' y; printf ")"
  | Neg x → printf "@,(␣-␣"; eval_parameter' x; printf ")"
  | Prod [] → printf "1.0_%s" !kind
  | Prod [x] → eval_parameter' x
```

```
  | Prod (x :: xs) →
      printf "@,("; eval_parameter′ x;
      List.iter (fun x → printf "␣*␣"; eval_parameter′ x) xs;
      printf ")"
  | Quot (x, y) →
      printf "@,("; eval_parameter′ x;
      printf "␣/␣"; eval_parameter′ y; printf ")"
  | Rec x →
      printf "@,␣(1.0_%s␣/␣" !kind; eval_parameter′ x; printf ")"
  | Pow (x, n) →
      printf "@,("; eval_parameter′ x; printf "**%d" n; printf ")"
  | Sqrt x → printf "@,sqrt␣("; eval_parameter′ x; printf ")"
  | Sin x → printf "@,sin␣("; eval_parameter′ x; printf ")"
  | Cos x → printf "@,cos␣("; eval_parameter′ x; printf ")"
  | Tan x → printf "@,tan␣("; eval_parameter′ x; printf ")"
  | Cot x → printf "@,cot␣("; eval_parameter′ x; printf ")"
  | Atan2 (y, x) → printf "@,atan2␣("; eval_parameter′ y;
      printf ",@␣"; eval_parameter′ x; printf ")"
  | Conj x → printf "@,conjg␣("; eval_parameter′ x; printf ")"

let strip_single_tag = function
  | Real x → x
  | Complex x → x

let strip_array_tag = function
  | Real_Array x → x
  | Complex_Array x → x

let eval_parameter (lhs, rhs) =
  let x = CM.constant_symbol (strip_single_tag lhs) in
  printf "␣␣␣␣@[<2>%s␣=␣" x; eval_parameter′ rhs; nl ()

let eval_para_list n l =
  printf "␣subroutine␣setup_parameters%s␣()" (string_of_int n); nl ();
  List.iter eval_parameter l;
  printf "␣end␣subroutine␣setup_parameters%s" (string_of_int n); nl ()

let eval_parameter_pair (lhs, rhs) =
  let x = CM.constant_symbol (strip_array_tag lhs) in
  let _ = List.fold_left (fun i rhs′ →
    printf "␣␣␣␣@[<2>%s(%d)␣=␣" x i; eval_parameter′ rhs′; nl ();
    succ i) 1 rhs in
  ()

let eval_para_pair_list n l =
  printf "␣subroutine␣setup_parameters%s␣()" (string_of_int n); nl ();
  List.iter eval_parameter_pair l;
  printf "␣end␣subroutine␣setup_parameters%s" (string_of_int n); nl ()

let print_echo fmt p =
  let s = CM.constant_symbol p in
  printf "␣␣␣␣write␣(unit␣=␣*,␣fmt␣=␣fmt_%s)␣\"%s\",␣%s"
    fmt s s; nl ()
```

```
let print_echo_array fmt (p, n) =
  let s = CM.constant_symbol p in
  for i = 1 to n do
    printf "␣␣␣␣write␣(unit␣=␣*,␣fmt␣=␣fmt_%s_array)␣" fmt ;
    printf "\"%s\",␣%d,␣%s(%d)" s i s i; nl ()
  done

let parameters_to_fortran oc params =
  setup_fortran_formatter !line_length oc;
  let declarations = classify_parameters params in
  printf "module␣%s" !parameter_module; nl ();
  printf "␣␣use␣kinds"; nl ();
  printf "␣␣use␣constants"; nl ();
  printf "␣␣implicit␣none"; nl ();
  printf "␣␣private"; nl ();
  printf "␣␣@[<2>public␣::␣setup_parameters";
  if !no_write then begin
    printf "!␣No␣print_parameters"; nl ();
  end else begin
    printf "@,,␣print_parameters"; nl ();
  end;
  declare_default_parameters "real" params.input;
  declare_parameters "real" (schisma 69 declarations.real_singles);
  List.iter (declare_parameter_array "real") declarations.real_arrays;
  declare_parameters "complex" (schisma 69 declarations.complex_singles);
  List.iter (declare_parameter_array "complex") declarations.complex_arrays;
  printf "contains"; nl ();
  printf "␣␣␣␣!␣derived␣parameters:"; nl ();
  let shredded = schisma_num 1 120 params.derived in
  let shredded_arrays = schisma_num 1 120 params.derived_arrays in
     let num_sub = List.length shredded in
     let num_sub_arrays = List.length shredded_arrays in
  printf "␣␣␣␣␣!length:␣%s" (string_of_int (List.length params.derived));
     nl ();
  printf "␣␣␣␣␣!Num_Sub:␣%s" (string_of_int num_sub); nl ();
  List.iter (fun (i, l) → eval_para_list i l) shredded;
  List.iter (fun (i, l) → eval_para_pair_list (num_sub + i) l)
    shredded_arrays;
  printf "␣␣subroutine␣setup_parameters␣()"; nl ();
  let sum_sub = num_sub + num_sub_arrays in
  for i = 1 to sum_sub do
    printf "␣␣␣␣call␣setup_parameters%s" (string_of_int i); nl ();
  done;
  printf "␣␣end␣subroutine␣setup_parameters"; nl ();
  if !no_write then begin
    printf "!␣No␣print_parameters"; nl ();
  end else begin
    printf "␣␣subroutine␣print_parameters␣()"; nl ();
    printf "␣␣␣␣@[<2>character(len=*),␣parameter␣::";
    printf "@␣fmt_real␣=␣\"(A12,4X,'␣=␣',E25.18)\",";
```

```
    printf "@␣fmt_complex␣=␣\"(A12,4X,'␣=␣',E25.18,'␣+␣i*',E25.18)\",";
    printf "@␣fmt_real_array␣=␣\"(A12,'(',I2.2,')','␣=␣',E25.18)\",";
    printf "@␣fmt_complex_array␣=␣";
    printf "\"(A12,'(',I2.2,')','␣=␣',E25.18,'␣+␣i*',E25.18)\""; nl ();
    printf "␣␣␣␣@[<2>write␣(unit␣=␣*,␣fmt␣=␣\"(A)\")␣@,";
    printf "\"default␣values␣for␣the␣input␣parameters:\""; nl ();
    List.iter (fun (p, _) → print_echo "real" p) params.input;
    printf "␣␣␣␣@[<2>write␣(unit␣=␣*,␣fmt␣=␣\"(A)\")␣@,";
    printf "\"derived␣parameters:\""; nl ();
    List.iter (print_echo "real") declarations.real_singles;
    List.iter (print_echo "complex") declarations.complex_singles;
    List.iter (print_echo_array "real") declarations.real_arrays;
    List.iter (print_echo_array "complex") declarations.complex_arrays;
    printf "␣␣end␣subroutine␣print_parameters"; nl ();
  end;
  printf "end␣module␣%s" !parameter_module; nl ();
  printf "!␣O'Mega␣revision␣control␣information:"; nl ();
  List.iter (fun s → printf "!␣␣␣␣%s" s; nl ())
    (ThoList.flatmap RCS.summary (M.rcs :: rcs_list));
  printf "!!!␣program␣test_parameters"; nl ();
  printf "!!!␣␣␣use␣%s" !parameter_module; nl ();
  printf "!!!␣␣␣call␣setup_parameters␣()"; nl ();
  printf "!!!␣␣␣call␣print_parameters␣()"; nl ();
  printf "!!!␣end␣program␣test_parameters"; nl ()
```

Run-Time Diagnostics

```
type diagnostic = All | Arguments | Momenta | Gauge

type diagnostic_mode = Off | Warn | Panic

let warn mode =
  match !mode with
  | Off → false
  | Warn → true
  | Panic → true

let panic mode =
  match !mode with
  | Off → false
  | Warn → false
  | Panic → true

let suffix mode =
  if panic mode then
    "panic"
  else
    "warn"

let diagnose_arguments = ref Off
let diagnose_momenta = ref Off
let diagnose_gauge = ref Off
```

```
let rec parse_diagnostic = function
  | All, panic →
      parse_diagnostic (Arguments, panic);
      parse_diagnostic (Momenta, panic);
      parse_diagnostic (Gauge, panic)
  | Arguments, panic →
      diagnose_arguments := if panic then Panic else Warn
  | Momenta, panic →
      diagnose_momenta := if panic then Panic else Warn
  | Gauge, panic →
      diagnose_gauge := if panic then Panic else Warn
```

If diagnostics are required, we have to switch off Fortran95 features like pure functions.

```
let parse_diagnostics = function
  | [] → ()
  | diagnostics →
      fortran95 := false;
      List.iter parse_diagnostic diagnostics
```

Amplitude

```
let declare_momenta_chunk = function
  | [] → ()
  | momenta →
      printf "␣␣␣␣@[<2>type(momentum)␣::␣";
      print_list (List.map format_momentum momenta); nl ()

let declare_momenta = function
  | [] → ()
  | momenta →
      List.iter
        declare_momenta_chunk
        (ThoList.chopn declaration_chunk_size momenta)

let declare_wavefunctions multiplicity wfs =
  let wfs' = classify_wfs wfs in
  declare_list multiplicity ("complex(kind=" ^ !kind ^ ")")
    (wfs'.scalars @ wfs'.brs_scalars);
  declare_list multiplicity ("type(" ^ Fermions.psi_type ^ ")")
    (wfs'.spinors @ wfs'.brs_spinors);
  declare_list multiplicity ("type(" ^ Fermions.psibar_type ^ ")")
    (wfs'.conjspinors @ wfs'.brs_conjspinors);
  declare_list multiplicity ("type(" ^ Fermions.chi_type ^ ")")
    (wfs'.realspinors @ wfs'.brs_realspinors @ wfs'.ghostspinors);
  declare_list multiplicity ("type(" ^ Fermions.grav_type ^ ")") wfs'.vectorspinors;
  declare_list multiplicity "type(vector)" (wfs'.vectors @ wfs'.massive_vectors @
      wfs'.brs_vectors @ wfs'.brs_massive_vectors @ wfs'.ward_vectors);
  declare_list multiplicity "type(tensor2odd)" wfs'.tensors_1;
  declare_list multiplicity "type(tensor)" wfs'.tensors_2
```

```
let flavors a = F.incoming a @ F.outgoing a

let declare_brakets_chunk = function
  | [] → ()
  | amplitudes →
      printf "␣␣␣␣@[<2>complex(kind=%s)␣::␣" !kind;
      print_list (List.map (fun a → flavors_symbol ~decl:true (flavors a)) amplitudes); nl ()

let declare_brakets = function
  | [] → ()
  | amplitudes →
      List.iter
        declare_brakets_chunk
        (ThoList.chopn declaration_chunk_size amplitudes)

let print_variable_declarations amplitudes =
  let multiplicity = CF.multiplicity amplitudes
  and processes = CF.processes amplitudes in
  declare_momenta
    (PSet.elements
       (List.fold_left
          (fun set a →
             PSet.union set (List.fold_right
                               (fun wf → PSet.add (F.momentum_list wf))
                               (F.externals a) PSet.empty))
          PSet.empty processes));
  declare_momenta
    (PSet.elements
       (List.fold_left
          (fun set a →
             PSet.union set (List.fold_right
                               (fun wf → PSet.add (F.momentum_list wf))
                               (F.variables a) PSet.empty))
          PSet.empty processes));
  if !openmp then begin
     printf "␣␣type␣%s@[<2>" openmp_tld_type;
     nl ();
  end ;
  declare_wavefunctions multiplicity
    (WFSet.elements
       (List.fold_left
          (fun set a →
             WFSet.union set (List.fold_right WFSet.add (F.externals a) WFSet.empty))
          WFSet.empty processes));
  declare_wavefunctions multiplicity
    (WFSet.elements
       (List.fold_left
          (fun set a →
             WFSet.union set (List.fold_right WFSet.add (F.variables a) WFSet.empty))
          WFSet.empty processes));
  declare_brakets processes;
  if !openmp then begin
```

```
    printf "@]␣␣end␣type␣%s\n" openmp_tld_type;
    printf "␣␣type(%s)␣::␣%s" openmp_tld_type openmp_tld;
    nl ();
  end
```

print_current is the most important function that has to match the functions in `omega95` (see appendix X). It offers plentiful opportunities for making mistakes, in particular those related to signs. We start with a few auxiliary functions:

```
let children2 rhs =
  match F.children rhs with
  | [wf1; wf2] → (wf1, wf2)
  | _ → failwith "Targets.children2:␣can't␣happen"

let children3 rhs =
  match F.children rhs with
  | [wf1; wf2; wf3] → (wf1, wf2, wf3)
  | _ → invalid_arg "Targets.children3:␣can't␣happen"
```

Note that it is (marginally) faster to multiply the two scalar products with the coupling constant than the four vector components.

This could be part of `omegalib` as well ...

```
let format_coeff = function
  | 1 → ""
  | −1 → "-"
  | coeff → "(" ^ string_of_int coeff ^ ")*"

let format_coupling coeff c =
  match coeff with
  | 1 → c
  | −1 → "(-" ^ c ^")"
  | coeff → string_of_int coeff ^ "*" ^ c
```

The following is error prone and should be generated automagically.

```
let print_vector4 c wf1 wf2 wf3 fusion (coeff, contraction) =
  match contraction, fusion with
  | C_12_34, (F341 | F431 | F342 | F432 | F123 | F213 | F124 | F214)
  | C_13_42, (F241 | F421 | F243 | F423 | F132 | F312 | F134 | F314)
  | C_14_23, (F231 | F321 | F234 | F324 | F142 | F412 | F143 |
F413) →
      printf "((%s%s)*(%s*%s))*%s" (format_coeff coeff) c wf1 wf2 wf3
  | C_12_34, (F134 | F143 | F234 | F243 | F312 | F321 | F412 | F421)
  | C_13_42, (F124 | F142 | F324 | F342 | F213 | F231 | F413 | F431)
  | C_14_23, (F123 | F132 | F423 | F432 | F214 | F241 | F314 |
F341) →
      printf "((%s%s)*(%s*%s))*%s" (format_coeff coeff) c wf2 wf3 wf1
  | C_12_34, (F314 | F413 | F324 | F423 | F132 | F231 | F142 | F241)
  | C_13_42, (F214 | F412 | F234 | F432 | F123 | F321 | F143 | F341)
  | C_14_23, (F213 | F312 | F243 | F342 | F124 | F421 | F134 |
F431) →
```

```
        printf "((%s%s)*(%s*%s))*%s" (format_coeff coeff) c wf1 wf3 wf2

  let print_add_vector4 c wf1 wf2 wf3 fusion (coeff, contraction) =
    printf "@␣+␣";
    print_vector4 c wf1 wf2 wf3 fusion (coeff, contraction)

  let print_vector4_km c pa pb wf1 wf2 wf3 fusion (coeff, contraction) =
    match contraction, fusion with
    | C_12_34, (F341 | F431 | F342 | F432 | F123 | F213 | F124 | F214)
    | C_13_42, (F241 | F421 | F243 | F423 | F132 | F312 | F134 | F314)
    | C_14_23, (F231 | F321 | F234 | F324 | F142 | F412 | F143 |
F413) →
        printf "((%s%s%s+%s))*(%s*%s))*%s"
          (format_coeff coeff) c pa pb wf1 wf2 wf3
    | C_12_34, (F134 | F143 | F234 | F243 | F312 | F321 | F412 | F421)
    | C_13_42, (F124 | F142 | F324 | F342 | F213 | F231 | F413 | F431)
    | C_14_23, (F123 | F132 | F423 | F432 | F214 | F241 | F314 |
F341) →
        printf "((%s%s%s+%s))*(%s*%s))*%s"
          (format_coeff coeff) c pa pb wf2 wf3 wf1
    | C_12_34, (F314 | F413 | F324 | F423 | F132 | F231 | F142 | F241)
    | C_13_42, (F214 | F412 | F234 | F432 | F123 | F321 | F143 | F341)
    | C_14_23, (F213 | F312 | F243 | F342 | F124 | F421 | F134 |
F431) →
        printf "((%s%s%s+%s))*(%s*%s))*%s"
          (format_coeff coeff) c pa pb wf1 wf3 wf2

  let print_add_vector4_km c pa pb wf1 wf2 wf3 fusion (coeff, contraction) =
    printf "@␣+␣";
    print_vector4_km c pa pb wf1 wf2 wf3 fusion (coeff, contraction)

  let print_dscalar4 c wf1 wf2 wf3 p1 p2 p3 p123
      fusion (coeff, contraction) =
    match contraction, fusion with
    | C_12_34, (F341 | F431 | F342 | F432 | F123 | F213 | F124 | F214)
    | C_13_42, (F241 | F421 | F243 | F423 | F132 | F312 | F134 | F314)
    | C_14_23, (F231 | F321 | F234 | F324 | F142 | F412 | F143 |
F413) →
        printf "((%s%s)*(%s*%s)*(%s*%s)*%s*%s*%s)"
          (format_coeff coeff) c p1 p2 p3 p123 wf1 wf2 wf3
    | C_12_34, (F134 | F143 | F234 | F243 | F312 | F321 | F412 | F421)
    | C_13_42, (F124 | F142 | F324 | F342 | F213 | F231 | F413 | F431)
    | C_14_23, (F123 | F132 | F423 | F432 | F214 | F241 | F314 |
F341) →
        printf "((%s%s)*(%s*%s)*(%s*%s)*%s*%s*%s)"
          (format_coeff coeff) c p2 p3 p1 p123 wf1 wf2 wf3
    | C_12_34, (F314 | F413 | F324 | F423 | F132 | F231 | F142 | F241)
    | C_13_42, (F214 | F412 | F234 | F432 | F123 | F321 | F143 | F341)
    | C_14_23, (F213 | F312 | F243 | F342 | F124 | F421 | F134 |
F431) →
        printf "((%s%s)*(%s*%s)*(%s*%s)*%s*%s*%s)"
          (format_coeff coeff) c p1 p3 p2 p123 wf1 wf2 wf3
```

```
let print_add_dscalar4 c wf1 wf2 wf3 p1 p2 p3 p123
    fusion (coeff, contraction) =
  printf "@␣+␣";
  print_dscalar4 c wf1 wf2 wf3 p1 p2 p3 p123 fusion (coeff, contraction)

let print_dscalar2_vector2 c wf1 wf2 wf3 p1 p2 p3 p123
    fusion (coeff, contraction) =
  failwith "Targets.Fortran.print_dscalar2_vector2:␣incomplete!";
  match contraction, fusion with
  | C_12_34, (F134 | F143 | F234 | F243) →
      printf "((%s%s)*(%s*%s)*(%s*%s)*%s)"
        (format_coeff coeff) c p123 p1 wf2 wf3 wf1
  | C_12_34, (F312 | F321 | F412 | F421) →
      printf "((%s%s)*((%s*%s)*%s*%s)*%s)"
        (format_coeff coeff) c p2 p3 wf2 wf3 wf1
  | C_12_34, (F341 | F431 | F342 | F432 | F123 | F213 | F124 | F214)
  | C_13_42, (F241 | F421 | F243 | F423 | F132 | F312 | F134 | F314)
  | C_14_23, (F231 | F321 | F234 | F324 | F142 | F412 | F143 |
F413) →
      printf "((%s%s)*(%s*%s)*(%s*%s)*%s*%s*%s)"
        (format_coeff coeff) c p1 p2 p3 p123 wf1 wf2 wf3
  | C_13_42, (F124 | F142 | F324 | F342 | F213 | F231 | F413 | F431)
  | C_14_23, (F123 | F132 | F423 | F432 | F214 | F241 | F314 |
F341) →
      printf "((%s%s)*(%s*%s)*(%s*%s)*%s*%s*%s)"
        (format_coeff coeff) c p2 p3 p1 p123 wf1 wf2 wf3
  | C_12_34, (F314 | F413 | F324 | F423 | F132 | F231 | F142 | F241)
  | C_13_42, (F214 | F412 | F234 | F432 | F123 | F321 | F143 | F341)
  | C_14_23, (F213 | F312 | F243 | F342 | F124 | F421 | F134 |
F431) →
      printf "((%s%s)*(%s*%s)*(%s*%s)*%s*%s*%s)"
        (format_coeff coeff) c p1 p3 p2 p123 wf1 wf2 wf3

let print_add_dscalar2_vector2 c wf1 wf2 wf3 p1 p2 p3 p123
    fusion (coeff, contraction) =
  printf "@␣+␣";
  print_dscalar2_vector2 c wf1 wf2 wf3 p1 p2 p3 p123
    fusion (coeff, contraction)

let print_current amplitude dictionary rhs =
  match F.coupling rhs with
  | V3 (vertex, fusion, constant) →
      let ch1, ch2 = children2 rhs in
      let wf1 = multiple_variable amplitude dictionary ch1
      and wf2 = multiple_variable amplitude dictionary ch2
      and p1 = momentum ch1
      and p2 = momentum ch2
      and m1 = CM.mass_symbol (F.flavor ch1)
      and m2 = CM.mass_symbol (F.flavor ch2) in
      let c = CM.constant_symbol constant in
      printf "@,␣%s␣" (if (F.sign rhs) < 0 then "-" else "+");
      begin match vertex with
```

Fermionic currents $\bar\psi \not{A}\psi$ and $\bar\psi\phi\psi$ are handled by the *Fermions* module, since they depend on the choice of Feynman rules: Dirac or Majorana.

```
| FBF (coeff, fb, b, f) →
    begin match coeff, fb, b, f with
    | _, Psibar, VLRM, Psi | _, Psibar, SPM, Psi
    | _, Psibar, VAM, Psi | _, Psibar, VA3M, Psi
    | _, Psibar, TVA, Psi | _, Psibar, TVAM, Psi
    | _, Psibar, TLR, Psi | _, Psibar, TLRM, Psi
    | _, Psibar, TRL, Psi | _, Psibar, TRLM, Psi →
        let p12 = Printf.sprintf "(-%s-%s)" p1 p2 in
        Fermions.print_current_mom (coeff, fb, b, f) c wf1 wf2 p1 p2
            p12 fusion
    | _, _, _, _ →
        Fermions.print_current (coeff, fb, b, f) c wf1 wf2 fusion
    end
| PBP (coeff, f1, b, f2) →
    Fermions.print_current_p (coeff, f1, b, f2) c wf1 wf2 fusion
| BBB (coeff, fb1, b, fb2) →
    Fermions.print_current_b (coeff, fb1, b, fb2) c wf1 wf2 fusion
| GBG (coeff, fb, b, f) → let p12 =
    Printf.sprintf "(-%s-%s)" p1 p2 in
    Fermions.print_current_g (coeff, fb, b, f) c wf1 wf2 p1 p2
        p12 fusion
```

Table 9.13 is a bit misleading, since if includes totally antisymmetric structure constants. The space-time part alone is also totally antisymmetric:

```
| Gauge_Gauge_Gauge coeff →
    let c = format_coupling coeff c in
    begin match fusion with
    | (F23 | F31 | F12) →
        printf "g_gg(%s,%s,%s,%s,%s)" c wf1 p1 wf2 p2
    | (F32 | F13 | F21) →
        printf "g_gg(%s,%s,%s,%s,%s)" c wf2 p2 wf1 p1
    end
```

In *Aux_Gauge_Gauge*, we can not rely on antisymmetry alone, because of the different Lorentz representations of the auxiliary and the gauge field. Instead we have to provide the sign in

$$(V_2 \wedge V_3) \cdot T_1 = \begin{cases} V_2 \cdot (T_1 \cdot V_3) = -V_2 \cdot (V_3 \cdot T_1) \\ V_3 \cdot (V_2 \cdot T_1) = -V_3 \cdot (T_1 \cdot V_2) \end{cases} \tag{15.2}$$

ourselves. Alternatively, one could provide `g_xg` mirroring `g_gx`.

```
| Aux_Gauge_Gauge coeff →
    let c = format_coupling coeff c in
    begin match fusion with
    | F23 → printf "x_gg(%s,%s,%s)" c wf1 wf2
    | F32 → printf "x_gg(%s,%s,%s)" c wf2 wf1
    | F12 → printf "g_gx(%s,%s,%s)" c wf2 wf1
    | F21 → printf "g_gx(%s,%s,%s)" c wf1 wf2
```

```
| F13 → printf "(-1)*g_gx(%s,%s,%s)" c wf2 wf1
| F31 → printf "(-1)*g_gx(%s,%s,%s)" c wf1 wf2
end
```

These cases are symmetric and we just have to juxtapose the correct fields and provide parentheses to minimize the number of multiplications.

```
| Scalar_Vector_Vector coeff →
    let c = format_coupling coeff c in
    begin match fusion with
    | (F23 | F32) → printf "%s*(%s*%s)" c wf1 wf2
    | (F12 | F13) → printf "(%s*%s)*%s" c wf1 wf2
    | (F21 | F31) → printf "(%s*%s)*%s" c wf2 wf1
    end

| Aux_Vector_Vector coeff →
    let c = format_coupling coeff c in
    begin match fusion with
    | (F23 | F32) → printf "%s*(%s*%s)" c wf1 wf2
    | (F12 | F13) → printf "(%s*%s)*%s" c wf1 wf2
    | (F21 | F31) → printf "(%s*%s)*%s" c wf2 wf1
    end
```

Even simpler:

```
| Scalar_Scalar_Scalar coeff →
    printf "(%s*%s*%s)" (format_coupling coeff c) wf1 wf2

| Aux_Scalar_Scalar coeff →
    printf "(%s*%s*%s)" (format_coupling coeff c) wf1 wf2

| Aux_Scalar_Vector coeff →
    let c = format_coupling coeff c in
    begin match fusion with
    | (F13 | F31) → printf "%s*(%s*%s)" c wf1 wf2
    | (F23 | F21) → printf "(%s*%s)*%s" c wf1 wf2
    | (F32 | F12) → printf "(%s*%s)*%s" c wf2 wf1
    end

| Vector_Scalar_Scalar coeff →
    let c = format_coupling coeff c in
    begin match fusion with
    | F23 → printf "v_ss(%s,%s,%s,%s,%s)" c wf1 p1 wf2 p2
    | F32 → printf "v_ss(%s,%s,%s,%s,%s)" c wf2 p2 wf1 p1
    | F12 → printf "s_vs(%s,%s,%s,%s,%s)" c wf1 p1 wf2 p2
    | F21 → printf "s_vs(%s,%s,%s,%s,%s)" c wf2 p2 wf1 p1
    | F13 → printf "(-1)*s_vs(%s,%s,%s,%s,%s)" c wf1 p1 wf2 p2
    | F31 → printf "(-1)*s_vs(%s,%s,%s,%s,%s)" c wf2 p2 wf1 p1
    end

| Graviton_Scalar_Scalar coeff →
    let c = format_coupling coeff c in
    begin match fusion with
    | F12 → printf "s_gravs(%s,%s,-(%s+%s),%s,%s,%s)" c m2 p1 p2 p2 wf1 wf2
    | F21 → printf "s_gravs(%s,%s,-(%s+%s),%s,%s,%s)" c m1 p1 p2 p1 wf2 wf1
```

```
| F13 → printf "s_gravs(%s,%s,%s,-(%s+%s),%s,%s)" c m2 p2 p1 p2 wf1 wf2
| F31 → printf "s_gravs(%s,%s,%s,-(%s+%s),%s,%s)" c m1 p1 p1 p2 wf2 wf1
| F23 → printf "grav_ss(%s,%s,%s,%s,%s,%s)" c m1 p1 p2 wf1 wf2
| F32 → printf "grav_ss(%s,%s,%s,%s,%s,%s)" c m1 p2 p1 wf2 wf1
end
```

In producing a vector in the fusion we always contract the rightmost index with the vector wavefunction from *rhs*. So the first momentum is always the one of the vector boson produced in the fusion, while the second one is that from the *rhs*. This makes the cases *F12* and *F13* as well as *F21* and *F31* equal. In principle, we could have already done this for the *Graviton_Scalar_Scalar* case.

```
| Graviton_Vector_Vector coeff →
    let c = format_coupling coeff c in
    begin match fusion with
    | (F12 | F13) → printf "v_gravv(%s,%s,-(%s+%s),%s,%s,%s)" c m2 p1 p2 p2 wf1 wf2
    | (F21 | F31) → printf "v_gravv(%s,%s,-(%s+%s),%s,%s,%s)" c m1 p1 p2 p1 wf2 wf1
    | F23 → printf "grav_vv(%s,%s,%s,%s,%s,%s)" c m1 p1 p2 wf1 wf2
    | F32 → printf "grav_vv(%s,%s,%s,%s,%s,%s)" c m1 p2 p1 wf2 wf1
    end

| Graviton_Spinor_Spinor coeff →
    let c = format_coupling coeff c in
    begin match fusion with
    | F23 → printf "f_gravf(%s,%s,-(%s+%s),(-%s),%s,%s)" c m2 p1 p2 p2 wf1 wf2
    | F32 → printf "f_gravf(%s,%s,-(%s+%s),(-%s),%s,%s)" c m1 p1 p2 p1 wf2 wf1
    | F12 → printf "f_fgrav(%s,%s,%s,%s+%s,%s,%s)" c m1 p1 p1 p2 wf1 wf2
    | F21 → printf "f_fgrav(%s,%s,%s,%s+%s,%s,%s)" c m2 p2 p1 p2 wf2 wf1
    | F13 → printf "grav_ff(%s,%s,%s,(-%s),%s,%s)" c m1 p1 p2 wf1 wf2
    | F31 → printf "grav_ff(%s,%s,%s,(-%s),%s,%s)" c m1 p2 p1 wf2 wf1
    end

| Dim4_Vector_Vector_Vector_T coeff →
    let c = format_coupling coeff c in
    begin match fusion with
    | F23 → printf "tkv_vv(%s,%s,%s,%s,%s)" c wf1 p1 wf2 p2
    | F32 → printf "tkv_vv(%s,%s,%s,%s,%s)" c wf2 p2 wf1 p1
    | F12 → printf "tv_kvv(%s,%s,%s,%s,%s)" c wf1 p1 wf2 p2
    | F21 → printf "tv_kvv(%s,%s,%s,%s,%s)" c wf2 p2 wf1 p1
    | F13 → printf "(-1)*tv_kvv(%s,%s,%s,%s,%s)" c wf1 p1 wf2 p2
    | F31 → printf "(-1)*tv_kvv(%s,%s,%s,%s,%s)" c wf2 p2 wf1 p1
    end

| Dim4_Vector_Vector_Vector_L coeff →
    let c = format_coupling coeff c in
    begin match fusion with
    | F23 → printf "lkv_vv(%s,%s,%s,%s,%s)" c wf1 p1 wf2 p2
    | F32 → printf "lkv_vv(%s,%s,%s,%s,%s)" c wf2 p2 wf1 p1
    | F12 | F13 → printf "lv_kvv(%s,%s,%s,%s)" c wf1 p1 wf2
    | F21 | F31 → printf "lv_kvv(%s,%s,%s,%s)" c wf2 p2 wf1
    end

| Dim6_Gauge_Gauge_Gauge coeff →
```

```
      let c = format_coupling coeff c in
      begin match fusion with
      | F23 | F31 | F12 →
          printf "kg_kgkg(%s,%s,%s,%s,%s)" c wf1 p1 wf2 p2
      | F32 | F13 | F21 →
          printf "kg_kgkg(%s,%s,%s,%s,%s)" c wf2 p2 wf1 p1
      end
  | Dim4_Vector_Vector_Vector_T5 coeff →
      let c = format_coupling coeff c in
      begin match fusion with
      | F23 → printf "t5kv_vv(%s,%s,%s,%s,%s)" c wf1 p1 wf2 p2
      | F32 → printf "t5kv_vv(%s,%s,%s,%s,%s)" c wf2 p2 wf1 p1
      | F12 | F13 → printf "t5v_kvv(%s,%s,%s,%s,%s)" c wf1 p1 wf2 p2
      | F21 | F31 → printf "t5v_kvv(%s,%s,%s,%s,%s)" c wf2 p2 wf1 p1
      end
  | Dim4_Vector_Vector_Vector_L5 coeff →
      let c = format_coupling coeff c in
      begin match fusion with
      | F23 → printf "l5kv_vv(%s,%s,%s,%s,%s)" c wf1 p1 wf2 p2
      | F32 → printf "l5kv_vv(%s,%s,%s,%s,%s)" c wf2 p2 wf1 p1
      | F12 → printf "l5v_kvv(%s,%s,%s,%s)" c wf1 p1 wf2
      | F21 → printf "l5v_kvv(%s,%s,%s,%s)" c wf2 p2 wf1
      | F13 → printf "(-1)*l5v_kvv(%s,%s,%s,%s)" c wf1 p1 wf2
      | F31 → printf "(-1)*l5v_kvv(%s,%s,%s,%s)" c wf2 p2 wf1
      end
  | Dim6_Gauge_Gauge_Gauge_5 coeff →
      let c = format_coupling coeff c in
      begin match fusion with
      | F23 → printf "kg5_kgkg(%s,%s,%s,%s,%s)" c wf1 p1 wf2 p2
      | F32 → printf "kg5_kgkg(%s,%s,%s,%s,%s)" c wf2 p2 wf1 p1
      | F12 → printf "kg_kg5kg(%s,%s,%s,%s,%s)" c wf1 p1 wf2 p2
      | F21 → printf "kg_kg5kg(%s,%s,%s,%s,%s)" c wf2 p2 wf1 p1
      | F13 → printf "(-1)*kg_kg5kg(%s,%s,%s,%s,%s)" c wf1 p1 wf2 p2
      | F31 → printf "(-1)*kg_kg5kg(%s,%s,%s,%s,%s)" c wf2 p2 wf1 p1
      end
  | Aux_DScalar_DScalar coeff →
      let c = format_coupling coeff c in
      begin match fusion with
      | (F23 | F32) →
          printf "%s*(%s*%s)*(%s*%s)" c p1 p2 wf1 wf2
      | (F12 | F13) →
          printf "%s*(-((%s+%s)*%s))*(%s*%s)" c p1 p2 p2 wf1 wf2
      | (F21 | F31) →
          printf "%s*(-((%s+%s)*%s))*(%s*%s)" c p1 p2 p1 wf1 wf2
      end
  | Aux_Vector_DScalar coeff →
      let c = format_coupling coeff c in
      begin match fusion with
```

```
| F23 → printf "%s*(%s*%s)*%s" c wf1 p2 wf2
| F32 → printf "%s*(%s*%s)*%s" c wf2 p1 wf1
| F12 → printf "%s*(-((%s+%s)*%s))*%s" c p1 p2 wf2 wf1
| F21 → printf "%s*(-((%s+%s)*%s))*%s" c p1 p2 wf1 wf2
| (F13 | F31) → printf "(-(%s+%s))*(%s*%s*%s)" p1 p2 c wf1 wf2
end
| Dim5_Scalar_Gauge2 coeff →
    let c = format_coupling coeff c in
    begin match fusion with
    | (F23 | F32) → printf "(%s)*((%s*%s)*(%s*%s) - (%s*%s)*(%s*%s))"
          c p1 wf2 p2 wf1 p1 p2 wf2 wf1
    | (F12 | F13) → printf "(%s)*%s*((-((%s+%s)*%s))*%s - ((-(%s+%s)*%s))*%s)"
          c wf1 p1 p2 wf2 p2 p1 p2 p2 wf2
    | (F21 | F31) → printf "(%s)*%s*((-((%s+%s)*%s))*%s - ((-(%s+%s)*%s))*%s)"
          c wf2 p2 p1 wf1 p1 p1 p2 p1 wf1
    end
| Dim5_Scalar_Gauge2_Skew coeff →
    let c = format_coupling coeff c in
    begin match fusion with
    | (F23 | F32) → printf "(- phi_vv (%s, %s, %s, %s, %s))" c p1 p2 wf1 wf2
    | (F12 | F13) → printf "(- v_phiv (%s, %s, %s, %s, %s))" c wf1 p1 p2 wf2
    | (F21 | F31) → printf "v_phiv (%s, %s, %s, %s, %s)" c wf2 p1 p2 wf1
    end
| Dim5_Scalar_Vector_Vector_T coeff →
    let c = format_coupling coeff c in
    begin match fusion with
    | (F23 | F32) → printf "(%s)*(%s*%s)*(%s*%s)" c p1 wf2 p2 wf1
    | (F12 | F13) → printf "(%s)*%s*(-((%s+%s)*%s))*%s" c wf1 p1 p2 wf2 p2
    | (F21 | F31) → printf "(%s)*%s*(-((%s+%s)*%s))*%s" c wf2 p2 p1 wf1 p1
    end
| Dim5_Scalar_Vector_Vector_U coeff →
    let c = format_coupling coeff c in
    begin match fusion with
    | (F23 | F32) → printf "phi_u_vv (%s, %s, %s, %s, %s)" c p1 p2 wf1 wf2
    | (F12 | F13) → printf "v_u_phiv (%s, %s, %s, %s, %s)" c wf1 p1 p2 wf2
    | (F21 | F31) → printf "v_u_phiv (%s, %s, %s, %s, %s)" c wf2 p2 p1 wf1
    end
| Dim5_Scalar_Vector_Vector_TU coeff →
    let c = format_coupling coeff c in
    begin match fusion with
    | F23 → printf "(%s)*((%s*%s)*(-(%s+%s)*%s) - (-(%s+%s)*%s)*(%s*%s))"
          c p1 wf2 p1 p2 wf1 p1 p2 p1 wf1 wf2
    | F32 → printf "(%s)*((%s*%s)*(-(%s+%s)*%s) - (-(%s+%s)*%s)*(%s*%s))"
          c p2 wf1 p1 p2 wf2 p1 p2 p2 wf1 wf2
    | F12 → printf "(%s)*%s*((%s*%s)*%s - (%s*%s)*%s)"
          c wf1 p1 wf2 p2 p1 p2 wf2
    | F21 → printf "(%s)*%s*((%s*%s)*%s - (%s*%s)*%s)"
          c wf2 p2 wf1 p1 p1 p2 wf1
```

```
| F13 → printf "(%s)*%s*((-(%s+%s)*%s)*%s␣-␣(-(%s+%s)*%s)*%s)"
      c wf1 p1 p2 wf2 p1 p1 p2 p1 wf2
| F31 → printf "(%s)*%s*((-(%s+%s)*%s)*%s␣-␣(-(%s+%s)*%s)*%s)"
      c wf2 p1 p2 wf1 p2 p1 p2 p2 wf1
end
| Scalar_Vector_Vector_t coeff →
    let c = format_coupling coeff c in
    begin match fusion with
    | (F23 | F32) → printf "s_vv_t(%s,%s,%s,%s,%s)" c wf1 p1 wf2 p2
    | (F12 | F13) → printf "v_sv_t(%s,%s,%s,%s,%s)" c wf1 p1 wf2 p2
    | (F21 | F31) → printf "v_sv_t(%s,%s,%s,%s,%s)" c wf2 p2 wf1 p1
    end
| Dim6_Vector_Vector_Vector_T coeff →
    let c = format_coupling coeff c in
    begin match fusion with
    | F23 → printf "(%s)*(%s*%s)*(%s*%s)*(%s-%s)" c p2 wf1 p1 wf2 p1 p2
    | F32 → printf "(%s)*(%s*%s)*(%s*%s)*(%s-%s)" c p1 wf2 p2 wf1 p2 p1
    | (F12 | F13) → printf "(%s)*((%s+2*%s)*%s)*(-((%s+%s)*%s))*%s"
          c p1 p2 wf1 p1 p2 wf2 p2
    | (F21 | F31) → printf "(%s)*((-((%s+%s)*%s))*(%s+2*%s)*%s)*%s"
          c p2 p1 wf1 p2 p1 wf2 p1
    end
| Tensor_2_Vector_Vector coeff →
    let c = format_coupling coeff c in
    begin match fusion with
    | (F23 | F32) → printf "t2_vv(%s,%s,%s)" c wf1 wf2
    | (F12 | F13) → printf "v_t2v(%s,%s,%s)" c wf1 wf2
    | (F21 | F31) → printf "v_t2v(%s,%s,%s)" c wf2 wf1
    end
| Tensor_2_Vector_Vector_1 coeff →
    let c = format_coupling coeff c in
    begin match fusion with
    | (F23 | F32) → printf "t2_vv_1(%s,%s,%s)" c wf1 wf2
    | (F12 | F13) → printf "v_t2v_1(%s,%s,%s)" c wf1 wf2
    | (F21 | F31) → printf "v_t2v_1(%s,%s,%s)" c wf2 wf1
    end
| Dim5_Tensor_2_Vector_Vector_1 coeff →
    let c = format_coupling coeff c in
    begin match fusion with
    | (F23 | F32) → printf "t2_vv_d5_1(%s,%s,%s,%s,%s)" c wf1 p1 wf2 p2
    | (F12 | F13) → printf "v_t2v_d5_1(%s,%s,%s,%s,%s)" c wf1 p1 wf2 p2
    | (F21 | F31) → printf "v_t2v_d5_1(%s,%s,%s,%s,%s)" c wf2 p2 wf1 p1
    end
| Tensor_2_Vector_Vector_t coeff →
    let c = format_coupling coeff c in
    begin match fusion with
    | (F23 | F32) → printf "t2_vv_t(%s,%s,%s,%s,%s)" c wf1 p1 wf2 p2
```

```
        | (F12 | F13) → printf "v_t2v_t(%s,%s,%s,%s,%s)" c wf1 p1 wf2 p2
        | (F21 | F31) → printf "v_t2v_t(%s,%s,%s,%s,%s)" c wf2 p2 wf1 p1
        end
    | Dim5_Tensor_2_Vector_Vector_2 coeff →
        let c = format_coupling coeff c in
        begin match fusion with
        | F23 → printf "t2_vv_d5_2(%s,%s,%s,%s,%s)" c wf1 p1 wf2 p2
        | F32 → printf "t2_vv_d5_2(%s,%s,%s,%s,%s)" c wf2 p2 wf1 p1
        | (F12 | F13) → printf "v_t2v_d5_2(%s,%s,%s,%s,%s)" c wf1 p1 wf2 p2
        | (F21 | F31) → printf "v_t2v_d5_2(%s,%s,%s,%s,%s)" c wf2 p2 wf1 p1
        end
    | Dim7_Tensor_2_Vector_Vector_T coeff →
        let c = format_coupling coeff c in
        begin match fusion with
        | F23 → printf "t2_vv_d7(%s,%s,%s,%s,%s)" c wf1 p1 wf2 p2
        | F32 → printf "t2_vv_d7(%s,%s,%s,%s,%s)" c wf2 p2 wf1 p1
        | (F12 | F13) → printf "v_t2v_d7(%s,%s,%s,%s,%s)" c wf1 p1 wf2 p2
        | (F21 | F31) → printf "v_t2v_d7(%s,%s,%s,%s,%s)" c wf2 p2 wf1 p1
        end
    end
```

Flip the sign to account for the i^2 relative to diagrams with only cubic couplings.

```
| V4 (vertex, fusion, constant) →
    let c = CM.constant_symbol constant
    and ch1, ch2, ch3 = children3 rhs in
    let wf1 = multiple_variable amplitude dictionary ch1
    and wf2 = multiple_variable amplitude dictionary ch2
    and wf3 = multiple_variable amplitude dictionary ch3
    and p1 = momentum ch1
    and p2 = momentum ch2
    and p3 = momentum ch3 in
    printf "@,␣%s␣" (if (F.sign rhs) < 0 then "+" else "-");
    begin match vertex with
    | Scalar4 coeff →
        printf "(%s*%s*%s*%s)" (format_coupling coeff c) wf1 wf2 wf3
    | Scalar2_Vector2 coeff →
        let c = format_coupling coeff c in
        begin match fusion with
        | F134 | F143 | F234 | F243 →
            printf "%s*%s*(%s*%s)" c wf1 wf2 wf3
        | F314 | F413 | F324 | F423 →
            printf "%s*%s*(%s*%s)" c wf2 wf1 wf3
        | F341 | F431 | F342 | F432 →
            printf "%s*%s*(%s*%s)" c wf3 wf1 wf2
        | F312 | F321 | F412 | F421 →
            printf "(%s*%s*%s)*%s" c wf2 wf3 wf1
        | F231 | F132 | F241 | F142 →
            printf "(%s*%s*%s)*%s" c wf1 wf3 wf2
        | F123 | F213 | F124 | F214 →
```

```
          printf "(%s*%s*%s)*%s" c wf1 wf2 wf3
        end
    | Vector4 contractions →
        begin match contractions with
        | [] → invalid_arg "Targets.print_current:␣Vector4␣[]"
        | head :: tail →
            printf "(";
            print_vector4 c wf1 wf2 wf3 fusion head;
            List.iter (print_add_vector4 c wf1 wf2 wf3 fusion) tail;
            printf ")"
        end
    | Vector4_K_Matrix_tho (_, poles) →
        let pa, pb =
          begin match fusion with
          | (F341 | F431 | F342 | F432 | F123 | F213 | F124 |
F214) → (p1, p2)
          | (F134 | F143 | F234 | F243 | F312 | F321 | F412 |
F421) → (p2, p3)
          | (F314 | F413 | F324 | F423 | F132 | F231 | F142 |
F241) → (p1, p3)
          end in
        printf "(%s*(%s*%s)*(%s*%s)*(%s*%s)@,*("
          c p1 wf1 p2 wf2 p3 wf3;
        List.iter (fun (coeff, pole) →
          printf "+%s/((%s+%s)*(%s+%s)-%s)"
            (CM.constant_symbol coeff) pa pb pa pb
            (CM.constant_symbol pole))
          poles;
        printf ")*(-%s-%s-%s))" p1 p2 p3
    | Vector4_K_Matrix_jr (disc, contractions) →
        let pa, pb =
          begin match disc, fusion with
          | 3, (F143 | F413 | F142 | F412 | F321 | F231 | F324 |
F234) → (p1, p2)
          | 3, (F314 | F341 | F214 | F241 | F132 | F123 | F432 |
F423) → (p2, p3)
          | 3, (F134 | F431 | F124 | F421 | F312 | F213 | F342 |
F243) → (p1, p3)
          | _, (F341 | F431 | F342 | F432 | F123 | F213 | F124 |
F214) → (p1, p2)
          | _, (F134 | F143 | F234 | F243 | F312 | F321 | F412 |
F421) → (p2, p3)
          | _, (F314 | F413 | F324 | F423 | F132 | F231 | F142 |
F241) → (p1, p3)
          end in
        begin match contractions with
        | [] → invalid_arg "Targets.print_current:␣Vector4_K_Matrix_jr␣[]"
        | head :: tail →
            printf "(";
            print_vector4_km c pa pb wf1 wf2 wf3 fusion head;
```

```
            List.iter (print_add_vector4_km c pa pb wf1 wf2 wf3 fusion)
              tail;
            printf ")"
          end
      | GBBG (coeff, fb, b, f) →
          Fermions.print_current_g4 (coeff, fb, b, f) c wf1 wf2 wf3
            fusion
```

In principle, *p4* could be obtained from the left hand side ...

```
      | DScalar4 contractions →
          let p123 = Printf.sprintf "(-%s-%s-%s)" p1 p2 p3 in
          begin match contractions with
          | [] → invalid_arg "Targets.print_current:␣DScalar4␣[]"
          | head :: tail →
              printf "(";
              print_dscalar4 c wf1 wf2 wf3 p1 p2 p3 p123 fusion head;
              List.iter (print_add_dscalar4
                           c wf1 wf2 wf3 p1 p2 p3 p123 fusion) tail;
              printf ")"
          end

      | DScalar2_Vector2 contractions →
          let p123 = Printf.sprintf "(-%s-%s-%s)" p1 p2 p3 in
          begin match contractions with
          | [] → invalid_arg "Targets.print_current:␣DScalar4␣[]"
          | head :: tail →
              printf "(";
              print_dscalar2_vector2
                c wf1 wf2 wf3 p1 p2 p3 p123 fusion head;
              List.iter (print_add_dscalar2_vector2
                           c wf1 wf2 wf3 p1 p2 p3 p123 fusion) tail;
              printf ")"
          end
      end

  | Vn (_, _, _) →
      invalid_arg "Targets.print_current:␣n-ary␣fusion"

let print_propagator f p m gamma =
  let minus_third = "(-1.0_" ^ !kind ^ "/3.0_" ^ !kind ^ ")" in
  let w =
    begin match CM.width f with
    | Vanishing | Fudged → "0.0_" ^ !kind
    | Constant → gamma
    | Timelike → "wd_tl(" ^ p ^ "," ^ gamma ^ ")"
    | Running →
        failwith "Targets.Fortran:␣running␣width␣not␣yet␣available"
    | Custom f → f ^ "(" ^ p ^ "," ^ gamma ^ ")"
    end in
  match CM.propagator f with
```

```
| Prop_Scalar →
    printf "pr_phi(%s,%s,%s," p m w
| Prop_Col_Scalar →
    printf "%s␣*␣pr_phi(%s,%s,%s," minus_third p m w
| Prop_Ghost → printf "(0,1)␣*␣pr_phi(%s,␣%s,␣%s," p m w
| Prop_Spinor →
    printf "%s(%s,%s,%s," Fermions.psi_propagator p m w
| Prop_ConjSpinor →
    printf "%s(%s,%s,%s," Fermions.psibar_propagator p m w
| Prop_Majorana →
    printf "%s(%s,%s,%s," Fermions.chi_propagator p m w
| Prop_Col_Majorana →
    printf "%s␣*␣%s(%s,%s,%s," minus_third Fermions.chi_propagator p m w
| Prop_Unitarity →
    printf "pr_unitarity(%s,%s,%s," p m w
| Prop_Col_Unitarity →
    printf "%s␣*␣pr_unitarity(%s,%s,%s," minus_third p m w
| Prop_Feynman →
    printf "pr_feynman(%s," p
| Prop_Col_Feynman →
    printf "%s␣*␣pr_feynman(%s," minus_third p
| Prop_Gauge xi →
    printf "pr_gauge(%s,%s," p (CM.gauge_symbol xi)
| Prop_Rxi xi →
    printf "pr_rxi(%s,%s,%s,%s," p m w (CM.gauge_symbol xi)
| Prop_Tensor_2 →
    printf "pr_tensor(%s,%s,%s," p m w
| Prop_Vectorspinor →
    printf "pr_grav(%s,%s,%s," p m w
| Aux_Scalar | Aux_Spinor | Aux_ConjSpinor | Aux_Majorana
| Aux_Vector | Aux_Tensor_1 → printf "("
| Aux_Col_Scalar | Aux_Col_Vector | Aux_Col_Tensor_1 →
printf "%s␣*␣(" minus_third
| Only_Insertion → printf "("

let print_projector f p m gamma =
  let minus_third = "(-1.0_" ^ !kind ^ "/3.0_" ^ !kind ^ ")" in
  match CM.propagator f with
  | Prop_Scalar →
      printf "pj_phi(%s,%s," m gamma
  | Prop_Col_Scalar →
      printf "%s␣*␣pj_phi(%s,%s," minus_third m gamma
  | Prop_Ghost →
      printf "(0,1)␣*␣pj_phi(%s,%s," m gamma
  | Prop_Spinor →
      printf "%s(%s,%s,%s," Fermions.psi_projector p m gamma
  | Prop_ConjSpinor →
      printf "%s(%s,%s,%s," Fermions.psibar_projector p m gamma
  | Prop_Majorana →
      printf "%s(%s,%s,%s," Fermions.chi_projector p m gamma
```

```
| Prop_Col_Majorana ->
    printf "%s * %s(%s,%s,%s," minus_third Fermions.chi_projector p m gamma
| Prop_Unitarity ->
    printf "pj_unitarity(%s,%s,%s," p m gamma
| Prop_Col_Unitarity ->
    printf "%s * pj_unitarity(%s,%s,%s," minus_third p m gamma
| Prop_Feynman | Prop_Col_Feynman ->
    invalid_arg "no on-shell Feynman propagator!"
| Prop_Gauge _ ->
    invalid_arg "no on-shell massless gauge propagator!"
| Prop_Rxi _ ->
    invalid_arg "no on-shell Rxi propagator!"
| Prop_Vectorspinor ->
    printf "pj_grav(%s,%s,%s," p m gamma
| Prop_Tensor_2 ->
    printf "pj_tensor(%s,%s,%s," p m gamma
| Aux_Scalar | Aux_Spinor | Aux_ConjSpinor | Aux_Majorana
| Aux_Vector | Aux_Tensor_1 -> printf "("
| Aux_Col_Scalar | Aux_Col_Vector | Aux_Col_Tensor_1 ->
printf "%s * (" minus_third
| Only_Insertion -> printf "("

let print_gauss f p m gamma =
  let minus_third = "(-1.0_" ^ !kind ^ "/3.0_" ^ !kind ^ ")" in
  match CM.propagator f with
  | Prop_Scalar ->
      printf "pg_phi(%s,%s,%s," p m gamma
  | Prop_Ghost ->
      printf "(0,1) * pg_phi(%s,%s,%s," p m gamma
  | Prop_Spinor ->
      printf "%s(%s,%s,%s," Fermions.psi_projector p m gamma
  | Prop_ConjSpinor ->
      printf "%s(%s,%s,%s," Fermions.psibar_projector p m gamma
  | Prop_Majorana ->
      printf "%s(%s,%s,%s," Fermions.chi_projector p m gamma
  | Prop_Col_Majorana ->
      printf "%s * %s(%s,%s,%s," minus_third Fermions.chi_projector p m gamma
  | Prop_Unitarity ->
      printf "pg_unitarity(%s,%s,%s," p m gamma
  | Prop_Feynman | Prop_Col_Feynman ->
      invalid_arg "no on-shell Feynman propagator!"
  | Prop_Gauge _ ->
      invalid_arg "no on-shell massless gauge propagator!"
  | Prop_Rxi _ ->
      invalid_arg "no on-shell Rxi propagator!"
  | Prop_Tensor_2 ->
      printf "pg_tensor(%s,%s,%s," p m gamma
  | Aux_Scalar | Aux_Spinor | Aux_ConjSpinor | Aux_Majorana
  | Aux_Vector | Aux_Tensor_1 -> printf "("
  | Only_Insertion -> printf "("
```

```
  | _ → invalid_arg "targets:print_gauss:␣not␣available"
let print_fusion_diagnostics amplitude dictionary fusion =
  if warn diagnose_gauge then begin
    let lhs = F.lhs fusion in
    let f = F.flavor lhs
    and v = variable lhs
    and p = momentum lhs in
    let mass = CM.mass_symbol f in
    match CM.propagator f with
    | Prop_Gauge _ | Prop_Feynman
    | Prop_Rxi _ | Prop_Unitarity →
        printf "␣␣␣␣␣␣@[<2>%s␣=" v;
        List.iter (print_current amplitude dictionary) (F.rhs fusion); nl ();
        begin match CM.goldstone f with
        | None →
            printf "␣␣␣␣␣␣call␣omega_ward_%s(\"%s\",%s,%s,%s)"
              (suffix diagnose_gauge) v mass p v; nl ()
        | Some (g, phase) →
            let gv = add_tag lhs (CM.flavor_symbol g ^ "_" ^ format_p lhs) in
            printf "␣␣␣␣␣␣call␣omega_slavnov_%s"
              (suffix diagnose_gauge);
            printf "(@[\"%s\",%s,%s,%s,@,%s*%s)"
              v mass p v (format_constant phase) gv; nl ()
        end
    | _ → ()
  end
let print_fusion amplitude dictionary fusion =
  let lhs = F.lhs fusion in
  let f = F.flavor lhs in
  printf "␣␣␣␣␣␣@[<2>%s␣=@,␣" (multiple_variable amplitude dictionary lhs);
  if F.on_shell amplitude lhs then
    print_projector f (momentum lhs)
      (CM.mass_symbol f) (CM.width_symbol f)
  else
    if F.is_gauss amplitude lhs then
      print_gauss f (momentum lhs)
        (CM.mass_symbol f) (CM.width_symbol f)
    else
      print_propagator f (momentum lhs)
        (CM.mass_symbol f) (CM.width_symbol f);
  List.iter (print_current amplitude dictionary) (F.rhs fusion);
  printf ")"; nl ()
let print_momenta seen_momenta amplitude =
  List.fold_left (fun seen f →
    let wf = F.lhs f in
    let p = F.momentum_list wf in
    if ¬ (PSet.mem p seen) then begin
      let rhs1 = List.hd (F.rhs f) in
      printf "␣␣␣␣%s␣=␣%s" (momentum wf)
```

```
        (String.concat " + "
           (List.map momentum (F.children rhs1))); nl ()
      end;
      PSet.add p seen)
      seen_momenta (F.fusions amplitude)

let print_fusions dictionary fusions =
  List.iter
    (fun (f, amplitude) →
      print_fusion_diagnostics amplitude dictionary f;
      print_fusion amplitude dictionary f)
    fusions

let print_braket amplitude dictionary name braket =
  let bra = F.bra braket
  and ket = F.ket braket in
  printf "      @[<2>%s = %s@, + " name name;
  begin match Fermions.reverse_braket (CM.lorentz (F.flavor bra)) with
  | false →
      printf "%s*(@," (multiple_variable amplitude dictionary bra);
      List.iter (print_current amplitude dictionary) ket;
      printf ")"
  | true →
      printf "(@,";
      List.iter (print_current amplitude dictionary) ket;
      printf ")*%s" (multiple_variable amplitude dictionary bra)
  end; nl ()
```

$$\mathrm{i}T = \mathrm{i}^{\#\text{vertices}}\mathrm{i}^{\#\text{propagators}}\cdots = \mathrm{i}^{n-2}\mathrm{i}^{n-3}\cdots = -\mathrm{i}(-1)^n\cdots \tag{15.3}$$

tho : we write some brakets twice using different names. Is it useful to cache them?

```
let print_brakets dictionary amplitude =
  let name = flavors_symbol (flavors amplitude) in
  printf "      %s = 0" name; nl ();
  List.iter (print_braket amplitude dictionary name) (F.brakets amplitude);
  let n = List.length (F.externals amplitude) in
  if n mod 2 = 0 then begin
    printf "      @[<2>%s =@, - %s ! %d vertices, %d propagators"
      name name (n − 2) (n − 3); nl ()
  end else begin
    printf "      ! %s = %s ! %d vertices, %d propagators"
      name name (n − 2) (n − 3); nl ()
  end;
  let s = F.symmetry amplitude in
  if s > 1 then
    printf "      @[<2>%s =@, %s@, / sqrt(%d.0_%s) ! symmetry factor" name name s !kind
  else
    printf "      ! unit symmetry factor";
```

```
  nl ()

let print_incoming wf =
  let p = momentum wf
  and s = spin wf
  and f = F.flavor wf in
  let m = CM.mass_symbol f in
  match CM.lorentz f with
  | Scalar → printf "1"
  | BRS Scalar → printf "(0,-1)␣*␣(%s␣*␣%s␣-␣%s**2)" p p m
  | Spinor →
      printf "%s␣(%s,␣-␣%s,␣%s)" Fermions.psi_incoming m p s
  | BRS Spinor →
      printf "%s␣(%s,␣-␣%s,␣%s)" Fermions.brs_psi_incoming m p s
  | ConjSpinor →
      printf "%s␣(%s,␣-␣%s,␣%s)" Fermions.psibar_incoming m p s
  | BRS ConjSpinor →
      printf "%s␣(%s,␣-␣%s,␣%s)" Fermions.brs_psibar_incoming m p s
  | Majorana →
      printf "%s␣(%s,␣-␣%s,␣%s)" Fermions.chi_incoming m p s
  | Maj_Ghost → printf "ghost␣(%s,␣-␣%s,␣%s)" m p s
  | BRS Majorana →
      printf "%s␣(%s,␣-␣%s,␣%s)" Fermions.brs_chi_incoming m p s
  | Vector | Massive_Vector →
      printf "eps␣(%s,␣-␣%s,␣%s)" m p s
  | BRS Vector | BRS Massive_Vector → printf
        "(0,1)␣*␣(%s␣*␣%s␣-␣%s**2)␣*␣eps␣(%s,␣-%s,␣%s)" p p m m p s
  | Vectorspinor | BRS Vectorspinor →
      printf "%s␣(%s,␣-␣%s,␣%s)" Fermions.grav_incoming m p s
  | Tensor_1 → invalid_arg "Tensor_1␣only␣internal"
  | Tensor_2 → printf "eps2␣(%s,␣-␣%s,␣%s)" m p s
  | _ → invalid_arg "no␣such␣BRST␣transformations"

let print_outgoing wf =
  let p = momentum wf
  and s = spin wf
  and f = F.flavor wf in
  let m = CM.mass_symbol f in
  match CM.lorentz f with
  | Scalar → printf "1"
  | BRS Scalar → printf "(0,-1)␣*␣(%s␣*␣%s␣-␣%s**2)" p p m
  | Spinor →
      printf "%s␣(%s,␣%s,␣%s)" Fermions.psi_outgoing m p s
  | BRS Spinor →
      printf "%s␣(%s,␣%s,␣%s)" Fermions.brs_psi_outgoing m p s
  | ConjSpinor →
      printf "%s␣(%s,␣%s,␣%s)" Fermions.psibar_outgoing m p s
  | BRS ConjSpinor →
      printf "%s␣(%s,␣%s,␣%s)" Fermions.brs_psibar_outgoing m p s
  | Majorana →
      printf "%s␣(%s,␣%s,␣%s)" Fermions.chi_outgoing m p s
```

```
  | BRS Majorana →
      printf "%s␣(%s,␣%s,␣%s)" Fermions.brs_chi_outgoing m p s
  | Maj_Ghost → printf "ghost␣(%s,␣%s,␣%s)" m p s
  | Vector | Massive_Vector →
      printf "conjg␣(eps␣(%s,␣%s,␣%s))" m p s
  | BRS Vector | BRS Massive_Vector → printf
        "(0,1)␣*␣(%s*%s-%s**2)␣*␣(conjg␣(eps␣(%s,␣%s,␣%s)))" p p m m p s
  | Vectorspinor | BRS Vectorspinor →
      printf "%s␣(%s,␣%s,␣%s)" Fermions.grav_incoming m p s
  | Tensor_1 → invalid_arg "Tensor_1␣only␣internal"
  | Tensor_2 → printf "conjg␣(eps2␣(%s,␣%s,␣%s))" m p s
  | BRS _ → invalid_arg "no␣such␣BRST␣transformations"

let print_external_momenta amplitude =
  let externals =
    List.combine
      (F.externals amplitude)
      (List.map (fun _ → true) (F.incoming amplitude) @
       List.map (fun _ → false) (F.outgoing amplitude)) in
  List.iter (fun (wf, incoming) →
    if incoming then
      printf "␣␣␣␣%s␣=␣-␣k(:,%d)␣!␣incoming"
        (momentum wf) (ext_momentum wf)
    else
      printf "␣␣␣␣%s␣=␣␣␣k(:,%d)␣!␣outgoing"
        (momentum wf) (ext_momentum wf); nl ()) externals

let print_externals seen_wfs amplitude =
  let externals =
    List.combine
      (F.externals amplitude)
      (List.map (fun _ → true) (F.incoming amplitude) @
       List.map (fun _ → false) (F.outgoing amplitude)) in
  List.fold_left (fun seen (wf, incoming) →
    if ¬ (WFSet.mem wf seen) then begin
      printf "␣␣␣␣␣␣@[<2>%s␣=@,␣" (variable wf);
      (if incoming then print_incoming else print_outgoing) wf; nl ()
    end;
    WFSet.add wf seen) seen_wfs externals

let flavors_sans_color_to_string flavors =
  String.concat "␣" (List.map M.flavor_to_string flavors)

let process_sans_color_to_string (fin, fout) =
  flavors_sans_color_to_string fin ^ "␣->␣" ^
  flavors_sans_color_to_string fout

let print_fudge_factor amplitude =
  let name = flavors_symbol (flavors amplitude) in
  List.iter (fun wf →
    let p = momentum wf
    and f = F.flavor wf in
    match CM.width f with
```

```
    | Fudged →
        let m = CM.mass_symbol f
        and w = CM.width_symbol f in
        printf "␣␣␣␣␣␣if␣(%s␣>␣0.0_%s)␣then" w !kind; nl ();
        printf "␣␣␣␣␣␣␣␣@[<2>%s␣=␣%s@␣*␣(%s*%s␣-␣%s**2)"
          name name p p m;
        printf "@␣/␣cmplx␣(%s*%s␣-␣%s**2,␣%s*%s,␣kind=%s)"
          p p m m w !kind; nl ();
        printf "␣␣␣␣␣␣end␣if"; nl ()
    | _ → ()) (F.s_channel amplitude)

let num_helicities amplitudes =
  List.length (CF.helicities amplitudes)
```

Spin, Flavor & Color Tables

The following abomination is required to keep the number of continuation lines as low as possible. FORTRAN77-style `DATA` statements are actually a bit nicer here, but they are nor available for *constant* arrays.

We used to have a more elegant design with a sentinel 0 added to each initializer, but some revisions of the Compaq/Digital Compiler have a bug that causes it to reject this variant.

The actual table writing code using `reshape` should be factored, since it's the same algorithm every time.

```
let print_integer_parameter name value =
  printf "␣␣@[<2>integer,␣parameter␣::␣%s␣=␣%d" name value; nl ()

let print_real_parameter name value =
  printf "␣␣@[<2>real(kind=%s),␣parameter␣::␣%s␣=␣%d"
    !kind name value; nl ()

let print_logical_parameter name value =
  printf "␣␣@[<2>logical,␣parameter␣::␣%s␣=␣.%s."
    name (if value then "true" else "false"); nl ()

let num_particles_in amplitudes =
  match CF.flavors amplitudes with
  | [] → 0
  | (fin, _) :: _ → List.length fin

let num_particles_out amplitudes =
  match CF.flavors amplitudes with
  | [] → 0
  | (_, fout) :: _ → List.length fout

let num_particles amplitudes =
  match CF.flavors amplitudes with
  | [] → 0
  | (fin, fout) :: _ → List.length fin + List.length fout
```

```
module CFlow = Color.Flow

let num_color_flows amplitudes =
  List.length (CF.color_flows amplitudes)

let num_color_indices_default = 2 (* Standard model *)

let num_color_indices amplitudes =
  try CFlow.rank (List.hd (CF.color_flows amplitudes)) with _ → num_color_indices_default

let color_to_string c =
  "(" ^ (String.concat "," (List.map (Printf.sprintf "%3d") c)) ^ ")"

let cflow_to_string cflow =
  String.concat "␣" (List.map color_to_string (CFlow.in_to_lists cflow)) ^ "␣->␣" ^
  String.concat "␣" (List.map color_to_string (CFlow.out_to_lists cflow))

let protected = ",␣protected" (* Fortran 2003! *)

let print_spin_table name tuples =
  printf "␣␣@[<2>integer,␣dimension(n_prt,n_hel),␣save%s␣::␣table_spin_%s"
    protected name; nl ();
  match tuples with
  | [] → ()
  | _ →
      ignore (List.fold_left (fun i (tuple1, tuple2) →
        printf "␣␣@[<2>data␣table_spin_%s(:,%4d)␣/␣%s␣/" name i
          (String.concat ",␣" (List.map (Printf.sprintf "%2d") (tuple1 @ tuple2)));
        nl (); succ i) 1 tuples)

let print_spin_tables amplitudes =
  (* print_spin_table_old "s" "states_old" (CF.helicities amplitudes);
*)
  print_spin_table "states" (CF.helicities amplitudes);
  nl ()

let print_flavor_table name tuples =
  printf "␣␣@[<2>integer,␣dimension(n_prt,n_flv),␣save%s␣::␣table_flavor_%s"
    protected name; nl ();
  match tuples with
  | [] → ()
  | _ →
      ignore (List.fold_left (fun i tuple →
        printf "␣␣@[<2>data␣table_flavor_%s(:,%4d)␣/␣%s␣/␣!␣%s" name i
          (String.concat ",␣"
             (List.map (fun f → Printf.sprintf "%3d" (M.pdg f)) tuple))
          (String.concat "␣" (List.map M.flavor_to_string tuple));
        nl (); succ i) 1 tuples)

let print_flavor_tables amplitudes =
  (* let n = num_particles amplitudes in *)
  (* print_flavor_table_old n "f" "states_old" (List.map (fun (fin, fout) →
fin @ fout) (CF.flavors amplitudes)); *)
  print_flavor_table "states"
    (List.map (fun (fin, fout) → fin @ fout) (CF.flavors amplitudes));
```

```
  nl ()

let num_flavors amplitudes =
  List.length (CF.flavors amplitudes)

let print_color_flows_table tuples =
  printf
    "␣␣@[<2>integer,␣dimension(n_cindex,n_prt,n_cflow),␣save%s␣::␣table_color_flows"
    protected; nl ();
  match tuples with
  | [] → ()
  | _ :: _ as tuples →
      ignore (List.fold_left (fun i tuple →
        begin match CFlow.to_lists tuple with
        | [] → ()
        | cf1 :: cfn →
            printf "␣␣@[<2>data␣table_color_flows(:,:,%4d)␣/" i;
            printf "@␣%s" (String.concat "," (List.map string_of_int cf1));
            List.iter (function cf →
              printf ",@␣␣%s" (String.concat "," (List.map string_of_int cf))) cfn;
            printf "@␣/"; nl ()
        end;
        succ i) 1 tuples)

let print_ghost_flags_table tuples =
  printf
    "␣␣@[<2>logical,␣dimension(n_prt,n_cflow),␣save%s␣::␣table_ghost_flags"
    protected; nl ();
  match tuples with
  | [] → ()
  | _ →
      ignore (List.fold_left (fun i tuple →
        begin match CFlow.ghost_flags tuple with
        | [] → ()
        | gf1 :: gfn →
            printf "␣␣@[<2>data␣table_ghost_flags(:,%4d)␣/" i;
            printf "@␣%s" (if gf1 then "T" else "F");
            List.iter (function gf → printf ",@␣␣%s" (if gf then "T" else "F")) gfn;
            printf "␣/";
            nl ()
        end;
        succ i) 1 tuples)

let format_power_of x
    { Color.Flow.num = num; Color.Flow.den = den; Color.Flow.power = pwr } =
  match num, den, pwr with
  | _, 0, _ → invalid_arg "format_power_of:␣zero␣denominator"
  | 0, _, _ → "+zero"
  | 1, 1, 0 | −1, −1, 0 → "+one"
  | −1, 1, 0 | 1, −1, 0 → "-one"
  | 1, 1, 1 | −1, −1, 1 → "+" ^ x
  | −1, 1, 1 | 1, −1, 1 → "-" ^ x
```

```
    | 1, 1, −1 | −1, −1, −1 → "+1/" ^ x
    | −1, 1, −1 | 1, −1, −1 → "-1/" ^ x
    | 1, 1, p | −1, −1, p →
        "+" ^ (if p > 0 then "" else "1/") ^ x ^ "**" ^ string_of_int (abs p)
    | −1, 1, p | 1, −1, p →
        "-" ^ (if p > 0 then "" else "1/") ^ x ^ "**" ^ string_of_int (abs p)
    | n, 1, 0 →
        (if n < 0 then "-" else "+") ^ string_of_int (abs n) ^ ".0_" ^ !kind
    | n, d, 0 →
        (if n × d < 0 then "-" else "+") ^
        string_of_int (abs n) ^ ".0_" ^ !kind ^ "/" ^
        string_of_int (abs d)
    | n, 1, 1 →
        (if n < 0 then "-" else "+") ^ string_of_int (abs n) ^ "*" ^ x
    | n, 1, −1 →
        (if n < 0 then "-" else "+") ^ string_of_int (abs n) ^ "/" ^ x
    | n, d, 1 →
        (if n × d < 0 then "-" else "+") ^
        string_of_int (abs n) ^ ".0_" ^ !kind ^ "/" ^
        string_of_int (abs d) ^ "*" ^ x
    | n, d, −1 →
        (if n × d < 0 then "-" else "+") ^
        string_of_int (abs n) ^ ".0_" ^ !kind ^ "/" ^
        string_of_int (abs d) ^ "/" ^ x
    | n, 1, p →
        (if n < 0 then "-" else "+") ^ string_of_int (abs n) ^
        (if p > 0 then "*" else "/") ^ x ^ "**" ^ string_of_int (abs p)
    | n, d, p →
        (if n × d < 0 then "-" else "+") ^
        string_of_int (abs n) ^ ".0_" ^ !kind ^ "/" ^
        string_of_int (abs d) ^
        (if p > 0 then "*" else "/") ^ x ^ "**" ^ string_of_int (abs p)

  let format_powers_of x = function
    | [] → "zero"
    | powers → String.concat "" (List.map (format_power_of x) powers)
```

We can optimize the following slightly by reusing common color factor *parameters*.

```
  let print_color_factor_table table =
    let n_cflow = Array.length table in
    let n_cfactors = ref 0 in
    for c1 = 0 to pred n_cflow do
      for c2 = 0 to pred n_cflow do
        match table.(c1).(c2) with
        | [] → ()
        | _ → incr n_cfactors
      done
    done;
```

```
    print_integer_parameter "n_cfactors" !n_cfactors;
    printf "␣␣@[<2>type(%s),␣dimension(n_cfactors),␣save%s␣::"
      omega_color_factor_abbrev protected;
    printf "@␣table_color_factors"; nl ();
    let i = ref 1 in
    if n_cflow > 0 then begin
      for c1 = 0 to pred n_cflow do
        for c2 = 0 to pred n_cflow do
          match table.(c1).(c2) with
          | [] → ()
          | cf →
              printf "␣␣@[<2>real(kind=%s),␣parameter,␣private␣::␣color_factor_%06d␣=␣%s"
                !kind !i (format_powers_of nc_parameter cf);
              nl ();
              printf "␣␣@[<2>data␣table_color_factors(%6d)␣/␣%s(%d,%d,color_factor_%06d)␣
                !i omega_color_factor_abbrev (succ c1) (succ c2) !i;
              incr i;
              nl ();
        done
      done
    end

  let print_color_tables amplitudes =
    let cflows = CF.color_flows amplitudes
    and cfactors = CF.color_factors amplitudes in
    (* print_color_flows_table_old "c" cflows; nl (); *)
    print_color_flows_table cflows; nl ();
    (* print_ghost_flags_table_old "g" cflows; nl (); *)
    print_ghost_flags_table cflows; nl ();
    (* print_color_factor_table_old cfactors; nl (); *)
    print_color_factor_table cfactors; nl ()

  let option_to_logical = function
    | Some _ → "T"
    | None → "F"

  let print_flavor_color_table n_flv n_cflow table =
    printf
      "␣␣@[<2>logical,␣dimension(n_flv,␣n_cflow),␣save%s␣::␣@␣flv_col_is_allowed"
      protected; nl ();
    if n_flv > 0 then begin
      for c = 0 to pred n_cflow do
        printf
          "␣␣@[<2>data␣flv_col_is_allowed(:,%4d)␣/" (succ c);
        printf "@␣%s" (option_to_logical table.(0).(c));
        for f = 1 to pred n_flv do
          printf ",@␣%s" (option_to_logical table.(f).(c))
        done;
        printf "@␣/"; nl ()
      done;
    end
```

```
let print_amplitude_table a =
  (* print_flavor_color_table_old "a" (num_flavors a) (List.length (CF.color_flows a)) (CF.process_ta
*)
  print_flavor_color_table
    (num_flavors a) (List.length (CF.color_flows a)) (CF.process_table a);
  nl ();
  printf
    "  @[<2>complex(kind=%s), dimension(n_flv, n_cflow, n_hel), save :: amp" !kind;
  nl ();
  nl ()

let print_helicity_selection_table () =
  printf "  @[<2>logical, dimension(n_hel), save :: ";
  printf "hel_is_allowed = T"; nl ();
  printf "  @[<2>real(kind=%s), dimension(n_hel), save :: " !kind;
  printf "hel_max_abs = 0"; nl ();
  printf "  @[<2>real(kind=%s), save :: " !kind;
  printf "hel_sum_abs = 0, ";
  printf "hel_threshold = 1E10"; nl ();
  printf "  @[<2>integer, save :: ";
  printf "hel_count = 0, ";
  printf "hel_cutoff = 100"; nl ();
  printf "  @[<2>integer :: ";
  printf "i"; nl ();
  printf "  @[<2>integer, save, dimension(n_hel) :: ";
  printf "hel_map = (/(i, i = 1, n_hel)/)"; nl ();
  printf "  @[<2>integer, save :: hel_finite = n_hel"; nl ();
  nl ()
```

Optional MD5 sum function

```
let print_md5sum_functions = function
  | Some s →
      printf "  @[<5>"; if !fortran95 then printf "pure ";
      printf "function md5sum ()"; nl ();
      printf "    character(len=32) :: md5sum"; nl ();
      printf "    ! DON'T EVEN THINK of modifying the following line!"; nl ();
      printf "    md5sum = \"%s\"" s; nl ();
      printf "  end function md5sum"; nl ();
      nl ()
  | None → ()
```

Maintenance & Inquiry Functions

```
let print_maintenance_functions () =
  if !whizard then begin
    printf "  subroutine init (par)"; nl ();
    printf "    real(kind=%s), dimension(*), intent(in) :: par" !kind; nl ();
    printf "    call import_from_whizard (par)"; nl ();
```

```
    printf "␣␣end␣subroutine␣init"; nl ();
    nl ();
    printf "␣␣subroutine␣final␣()"; nl ();
    printf "␣␣end␣subroutine␣final"; nl ();
    nl ();
    printf "␣␣subroutine␣update_alpha_s␣(alpha_s)"; nl ();
    printf "␣␣␣␣real(kind=%s),␣intent(in)␣::␣alpha_s" !kind; nl ();
    printf "␣␣␣␣call␣model_update_alpha_s␣(alpha_s)"; nl ();
    printf "␣␣end␣subroutine␣update_alpha_s"; nl ();
    nl ()
  end

let print_inquiry_function_openmp () = begin
  printf "␣␣pure␣function␣openmp_supported␣()␣result␣(status)"; nl ();
  printf "␣␣␣␣logical␣::␣status"; nl ();
  printf "␣␣␣␣status␣=␣%s" (if !openmp then ".true." else ".false."); nl ();
  printf "␣␣end␣function␣openmp_supported"; nl ();
  nl ()
end

let print_numeric_inquiry_functions (f, v) =
  printf "␣␣@[<5>"; if !fortran95 then printf "pure␣";
  printf "function␣%s␣()␣result␣(n)" f; nl ();
  printf "␣␣␣␣integer␣::␣n"; nl ();
  printf "␣␣␣␣n␣=␣%s" v; nl ();
  printf "␣␣end␣function␣%s" f; nl ();
  nl ()

let print_inquiry_functions name =
  printf "␣␣@[<5>"; if !fortran95 then printf "pure␣";
  printf "function␣number_%s␣()␣result␣(n)" name; nl ();
  printf "␣␣␣␣integer␣::␣n"; nl ();
  printf "␣␣␣␣n␣=␣size␣(table_%s,␣dim=2)" name; nl ();
  printf "␣␣end␣function␣number_%s" name; nl ();
  nl ();
  printf "␣␣@[<5>"; if !fortran95 then printf "pure␣";
  printf "subroutine␣%s␣(a)" name; nl ();
  printf "␣␣␣␣integer,␣dimension(:,:),␣intent(out)␣::␣a"; nl ();
  printf "␣␣␣␣a␣=␣table_%s" name; nl ();
  printf "␣␣end␣subroutine␣%s" name; nl ();
  nl ()

let print_color_flows () =
  printf "␣␣@[<5>"; if !fortran95 then printf "pure␣";
  printf "function␣number_color_indices␣()␣result␣(n)"; nl ();
  printf "␣␣␣␣integer␣::␣n"; nl ();
  printf "␣␣␣␣n␣=␣size␣(table_color_flows,␣dim=1)"; nl ();
  printf "␣␣end␣function␣number_color_indices"; nl ();
  nl ();
  printf "␣␣@[<5>"; if !fortran95 then printf "pure␣";
  printf "function␣number_color_flows␣()␣result␣(n)"; nl ();
  printf "␣␣␣␣integer␣::␣n"; nl ();
```

```
    printf "␣␣␣␣n␣=␣size␣(table_color_flows,␣dim=3)"; nl ();
    printf "␣␣end␣function␣number_color_flows"; nl ();
    nl ();
    printf "␣␣@[<5>"; if !fortran95 then printf "pure␣";
    printf "subroutine␣color_flows␣(a,␣g)"; nl ();
    printf "␣␣␣␣integer,␣dimension(:,:,:),␣intent(out)␣::␣a"; nl ();
    printf "␣␣␣␣logical,␣dimension(:,:),␣intent(out)␣::␣g"; nl ();
    printf "␣␣␣␣a␣=␣table_color_flows"; nl ();
    printf "␣␣␣␣g␣=␣table_ghost_flags"; nl ();
    printf "␣␣end␣subroutine␣color_flows"; nl ();
    nl ()

  let print_color_factors () =
    printf "␣␣@[<5>"; if !fortran95 then printf "pure␣";
    printf "function␣number_color_factors␣()␣result␣(n)"; nl ();
    printf "␣␣␣␣integer␣::␣n"; nl ();
    printf "␣␣␣␣n␣=␣size␣(table_color_factors)"; nl ();
    printf "␣␣end␣function␣number_color_factors"; nl ();
    nl ();
    printf "␣␣@[<5>"; if !fortran95 then printf "pure␣";
    printf "subroutine␣color_factors␣(cf)"; nl ();
    printf "␣␣␣␣type(%s),␣dimension(:),␣intent(out)␣::␣cf"
      omega_color_factor_abbrev; nl ();
    printf "␣␣␣␣cf␣=␣table_color_factors"; nl ();
    printf "␣␣end␣subroutine␣color_factors"; nl ();
    nl ();
    printf "␣␣@[<5>"; if !fortran95 ∧ pure_unless_openmp then printf "pure␣";
    printf "function␣color_sum␣(flv,␣hel)␣result␣(amp2)"; nl ();
    printf "␣␣␣␣integer,␣intent(in)␣::␣flv,␣hel"; nl ();
    printf "␣␣␣␣real(kind=%s)␣::␣amp2" !kind; nl ();
    printf "␣␣␣␣amp2␣=␣real␣(omega_color_sum␣(flv,␣hel,␣amp,␣table_color_factors))"; nl ();
    printf "␣␣end␣function␣color_sum"; nl ();
    nl ()

  let print_dispatch_functions () =
    printf "␣␣@[<5>";
    printf "subroutine␣new_event␣(p)"; nl ();
    printf "␣␣␣␣real(kind=%s),␣dimension(0:3,*),␣intent(in)␣::␣p" !kind; nl ();
    printf "␣␣␣␣logical␣::␣mask_dirty"; nl ();
    printf "␣␣␣␣integer␣::␣hel"; nl ();
    printf "␣␣␣␣call␣calculate_amplitudes␣(amp,␣p,␣hel_is_allowed)"; nl ();
    printf "␣␣␣␣if␣((hel_threshold␣.gt.␣0)␣.and.␣(hel_count␣.le.␣hel_cutoff))␣then"; nl ();
    printf "␣␣␣␣␣␣call␣@[<3>omega_update_helicity_selection@␣(hel_count,@␣amp,@␣";
    printf "hel_max_abs,@␣hel_sum_abs,@␣hel_is_allowed,@␣hel_threshold,@␣hel_cutoff,@␣mask
    printf "␣␣␣␣␣␣if␣(mask_dirty)␣then"; nl ();
    printf "␣␣␣␣␣␣␣␣hel_finite␣=␣0"; nl ();
    printf "␣␣␣␣␣␣␣␣do␣hel␣=␣1,␣n_hel"; nl ();
    printf "␣␣␣␣␣␣␣␣␣␣if␣(hel_is_allowed(hel))␣then"; nl ();
    printf "␣␣␣␣␣␣␣␣␣␣␣␣hel_finite␣=␣hel_finite␣+␣1"; nl ();
    printf "␣␣␣␣␣␣␣␣␣␣␣␣hel_map(hel_finite)␣=␣hel"; nl ();
    printf "␣␣␣␣␣␣␣␣␣␣end␣if"; nl ();
```

```
printf "        end do"; nl ();
printf "      end if"; nl ();
printf "    end if"; nl ();
printf "  end subroutine new_event"; nl ();
nl ();
printf "  @[<5>";
printf "subroutine reset_helicity_selection (threshold, cutoff)"; nl ();
printf "    real(kind=%s), intent(in) :: threshold" !kind; nl ();
printf "    integer, intent(in) :: cutoff"; nl ();
printf "    integer :: i"; nl ();
printf "    hel_is_allowed = T"; nl ();
printf "    hel_max_abs = 0"; nl ();
printf "    hel_sum_abs = 0"; nl ();
printf "    hel_count = 0"; nl ();
printf "    hel_threshold = threshold"; nl ();
printf "    hel_cutoff = cutoff"; nl ();
printf "    hel_map = (/(i, i = 1, n_hel)/)"; nl ();
printf "    hel_finite = n_hel"; nl ();
printf "  end subroutine reset_helicity_selection"; nl ();
nl ();
printf "  @[<5>"; if !fortran95 then printf "pure ";
printf "function is_allowed (flv, hel, col) result (yorn)"; nl ();
printf "    logical :: yorn"; nl ();
printf "    integer, intent(in) :: flv, hel, col"; nl ();
printf "    yorn = hel_is_allowed(hel) .and. ";
printf "flv_col_is_allowed(flv,col)"; nl ();
printf "  end function is_allowed"; nl ();
nl ();
printf "  @[<5>"; if !fortran95 then printf "pure ";
printf "function get_amplitude (flv, hel, col) result (amp_result)"; nl ();
printf "    complex(kind=%s) :: amp_result" !kind; nl ();
printf "    integer, intent(in) :: flv, hel, col"; nl ();
printf "    amp_result = amp(flv, col, hel)"; nl ();
printf "  end function get_amplitude"; nl ();
nl ()
```

Main Function

```
let format_power_of_nc
    { Color.Flow.num = num; Color.Flow.den = den; Color.Flow.power = pwr } =
  match num, den, pwr with
  | _, 0, _ → invalid_arg "format_power_of_nc: zero denominator"
  | 0, _, _ → ""
  | 1, 1, 0 | −1, −1, 0 → "+ 1"
  | −1, 1, 0 | 1, −1, 0 → "- 1"
  | 1, 1, 1 | −1, −1, 1 → "+ N"
  | −1, 1, 1 | 1, −1, 1 → "- N"
  | 1, 1, −1 | −1, −1, −1 → "+ 1/N"
  | −1, 1, −1 | 1, −1, −1 → "- 1/N"
```

```
  | 1, 1, p | −1, −1, p →
      "+ " ^ (if p > 0 then "" else "1/") ^ "N^" ^ string_of_int (abs p)
  | −1, 1, p | 1, −1, p →
      "- " ^ (if p > 0 then "" else "1/") ^ "N^" ^ string_of_int (abs p)
  | n, 1, 0 →
      (if n < 0 then "- " else "+ ") ^ string_of_int (abs n)
  | n, d, 0 →
      (if n × d < 0 then "- " else "+ ") ^
      string_of_int (abs n) ^ "/" ^ string_of_int (abs d)
  | n, 1, 1 →
      (if n < 0 then "- " else "+ ") ^ string_of_int (abs n) ^ "N"
  | n, 1, −1 →
      (if n < 0 then "- " else "+ ") ^ string_of_int (abs n) ^ "/N"
  | n, d, 1 →
      (if n × d < 0 then "- " else "+ ") ^
      string_of_int (abs n) ^ "/" ^ string_of_int (abs d) ^ "N"
  | n, d, −1 →
      (if n × d < 0 then "- " else "+ ") ^
      string_of_int (abs n) ^ "/" ^ string_of_int (abs d) ^ "/N"
  | n, 1, p →
      (if n < 0 then "- " else "+ ") ^ string_of_int (abs n) ^
      (if p > 0 then "*" else "/") ^ "N^" ^ string_of_int (abs p)
  | n, d, p →
      (if n × d < 0 then "- " else "+ ") ^ string_of_int (abs n) ^ "/" ^
      string_of_int (abs d) ^ (if p > 0 then "*" else "/") ^ "N^" ^ string_of_int (abs p)

let format_powers_of_nc = function
  | [] → "0"
  | powers → String.concat " " (List.map format_power_of_nc powers)

let print_description cmdline amplitudes () =
  printf "! File generated automatically by O'Mega"; nl ();
  printf "!"; nl ();
  printf "!   %s" cmdline; nl ();
  printf "!"; nl ();
  printf "! with all scattering amplitudes for the process(es)"; nl ();
  printf "!"; nl ();
  printf "!   flavor combinations:"; nl ();
  printf "!"; nl ();
  ThoList.iteri
    (fun i process →
      printf "!     %3d: %s" i (process_sans_color_to_string process); nl ())
    1 (CF.flavors amplitudes);
  printf "!"; nl ();
  printf "!   color flows:"; nl ();
  printf "!"; nl ();
  ThoList.iteri
    (fun i cflow →
      printf "!     %3d: %s" i (cflow_to_string cflow); nl ())
    1 (CF.color_flows amplitudes);
  printf "!"; nl ();
```

```
printf "!     NB: i.g. not all color flows contribute to all flavor"; nl ();
printf "!     combinations.  Consult the array FLV_COL_IS_ALLOWED"; nl ();
printf "!     below for the allowed combinations."; nl ();
printf "!"; nl ();
printf "!   Color Factors:"; nl ();
printf "!"; nl ();
let cfactors = CF.color_factors amplitudes in
for c1 = 0 to pred (Array.length cfactors) do
  for c2 = 0 to c1 do
    match cfactors.(c1).(c2) with
    | [] → ()
    | cfactor →
        printf "!     (%3d,%3d): %s"
          (succ c1) (succ c2) (format_powers_of_nc cfactor); nl ()
  done
done;
printf "!"; nl ();
printf "!   vanishing or redundant flavor combinations:"; nl ();
printf "!"; nl ();
List.iter (fun process →
  printf "!          %s" (process_sans_color_to_string process); nl ())
  (CF.vanishing_flavors amplitudes);
printf "!"; nl ();
begin
  match CF.constraints amplitudes with
  | None → ()
  | Some s →
      printf
        "!   diagram selection (MIGHT BREAK GAUGE INVARIANCE!!!):"; nl ();
      printf "!"; nl ();
      printf "!     %s" s; nl ();
      printf "!"; nl ()
end;
begin match RCS.description M.rcs with
| line1 :: lines →
    printf "! in %s" line1; nl ();
    List.iter (fun s → printf "!    %s" s; nl ()) lines
| [] → printf "! in %s" (RCS.name M.rcs); nl ()
end;
printf "!"; nl ()

let print_version () =
  printf "! O'Mega revision control information:"; nl ();
  List.iter (fun s → printf "!    %s" s; nl ())
    (ThoList.flatmap RCS.summary (M.rcs :: rcs_list @ F.rcs_list))
```

Printing Modules

```
type accessibility =
  | Public
```

```
  | Private
  | Protected (* Fortran 2003 *)

let accessibility_to_string = function
  | Public → "public"
  | Private → "private"
  | Protected → "protected"

type used_symbol =
  | As_Is of string
  | Aliased of string × string

let print_used_symbol = function
  | As_Is name → printf "%s" name
  | Aliased (orig, alias) → printf "%s => %s" alias orig

type used_module =
  | Full of string
  | Full_Aliased of string × (string × string) list
  | Subset of string × used_symbol list

let print_used_module = function
  | Full name
  | Full_Aliased (name, [])
  | Subset (name, []) →
      printf "  use %s" name;
      nl ()
  | Full_Aliased (name, aliases) →
      printf "  @[<5>use %s" name;
      List.iter
        (fun (orig, alias) → printf ", %s => %s" alias orig)
        aliases;
      nl ()
  | Subset (name, used_symbol :: used_symbols) →
      printf "  @[<5>use %s, only: " name;
      print_used_symbol used_symbol;
      List.iter (fun s → printf ", "; print_used_symbol s) used_symbols;
      nl ()

type fortran_module =
    { module_name : string;
      default_accessibility : accessibility;
      used_modules : used_module list;
      public_symbols : string list;
      print_declarations : (unit → unit) list;
      print_implementations : (unit → unit) list }

let print_public = function
  | name1 :: names →
      printf "  @[<2>public :: %s" name1;
      List.iter (fun n → printf ",@ %s" n) names; nl ()
  | [] → ()

let print_module m =
```

```
    printf "module␣%s" m.module_name; nl ();
    List.iter print_used_module m.used_modules;
    printf "␣␣implicit␣none"; nl ();
    printf "␣␣%s" (accessibility_to_string m.default_accessibility); nl ();
    print_public m.public_symbols; nl ();
    begin match m.print_declarations with
    | [] → ()
    | print_declarations →
        List.iter (fun f → f ()) print_declarations; nl ()
    end;
    begin match m.print_implementations with
    | [] → ()
    | print_implementations →
        printf "contains"; nl (); nl ();
        List.iter (fun f → f ()) print_implementations; nl ();
    end;
    printf "end␣module␣%s" m.module_name; nl ()

let print_modules modules =
    List.iter print_module modules;
    print_version ();
    print_flush ()

let module_to_file line_length oc prelude m =
    output_string oc (m.module_name ^ "\n");
    let filename = m.module_name ^ ".f90" in
    let channel = open_out filename in
    setup_fortran_formatter line_length channel;
    prelude ();
    print_modules [m];
    close_out channel

let modules_to_file line_length oc prelude = function
    | [] → ()
    | m :: mlist →
        module_to_file line_length oc prelude m;
        List.iter (module_to_file line_length oc (fun () → ())) mlist
```

Chopping Up Amplitudes

```
let num_fusions_brakets size amplitudes =
    let num_fusions =
      max 1 size in
    let count_brakets =
      List.fold_left
        (fun sum process → sum + List.length (F.brakets process))
        0 (CF.processes amplitudes)
    and count_processes =
      List.length (CF.processes amplitudes) in
    if count_brakets > 0 then
      let num_brakets =
```

```
        max 1 ((num_fusions × count_processes) / count_brakets) in
      (num_fusions, num_brakets)
    else
      (num_fusions, 1)

let chop_amplitudes size amplitudes =
  let num_fusions, num_brakets = num_fusions_brakets size amplitudes in
  (ThoList.enumerate 1 (ThoList.chopn num_fusions (CF.fusions amplitudes)),
   ThoList.enumerate 1 (ThoList.chopn num_brakets (CF.processes amplitudes)))

let print_compute_fusions1 dictionary (n, fusions) =
  if !openmp then begin
    printf "␣␣subroutine␣compute_fusions_%04d␣(%s)" n openmp_tld; nl ();
    printf "␣␣@[<5>type(%s),␣intent(inout)␣::␣%s" openmp_tld_type openmp_tld; nl ();
  end else begin
    printf "␣␣@[<5>subroutine␣compute_fusions_%04d␣()" n; nl ();
  end;
  print_fusions dictionary fusions;
  printf "␣␣end␣subroutine␣compute_fusions_%04d" n; nl ()

and print_compute_brakets1 dictionary (n, processes) =
  if !openmp then begin
    printf "␣␣subroutine␣compute_brakets_%04d␣(%s)" n openmp_tld; nl ();
    printf "␣␣@[<5>type(%s),␣intent(inout)␣::␣%s" openmp_tld_type openmp_tld; nl ();
  end else begin
    printf "␣␣@[<5>subroutine␣compute_brakets_%04d␣()" n; nl ();
  end;
  List.iter (print_brakets dictionary) processes;
  printf "␣␣end␣subroutine␣compute_brakets_%04d" n; nl ()
```

Common Stuff

```
let omega_public_symbols =
  ["number_particles_in"; "number_particles_out";
   "number_color_indices";
   "reset_helicity_selection"; "new_event";
   "is_allowed"; "get_amplitude"; "color_sum"; "openmp_supported"] @
  ThoList.flatmap
    (fun n → ["number_" ^ n; n])
    ["spin_states"; "flavor_states"; "color_flows"; "color_factors"]

let whizard_public_symbols md5sum =
  ["init"; "final"; "update_alpha_s"] @
  (match md5sum with Some _ → ["md5sum"] | None → [])

let used_modules () =
  [Full "kinds";
   Full Fermions.use_module;
   Full_Aliased ("omega_color", ["omega_color_factor", omega_color_factor_abbrev])] @
  List.map
    (fun m → Full m)
```

(match !*parameter_module* with "" → !*use_modules* | *pm* → *pm* :: !*use_modules*)

let *public_symbols* () =
  if !*whizard* then
    omega_public_symbols @ (*whizard_public_symbols* !*md5sum*)
  else
    omega_public_symbols

let *print_constants amplitudes* =

```
printf "␣␣!␣DON'T␣EVEN␣THINK␣of␣removing␣the␣following!"; nl ();
printf "␣␣!␣If␣the␣compiler␣complains␣about␣undeclared"; nl ();
printf "␣␣!␣or␣undefined␣variables,␣you␣are␣compiling"; nl ();
printf "␣␣!␣against␣an␣incompatible␣omega95␣module!"; nl ();
printf "␣␣@[<2>integer,␣dimension(%d),␣parameter,␣private␣::␣"
  (List.length require_library);
printf "require␣=@␣(/␣@[";
print_list require_library;
printf "␣/)"; nl (); nl ();
```

Using these parameters makes sense for documentation, but in practice, there is no need to ever change them.

```
List.iter
  (function name, value → print_integer_parameter name (value amplitudes))
  [ ("n_prt", num_particles);
    ("n_in", num_particles_in);
    ("n_out", num_particles_out);
    ("n_cflow", num_color_flows); (* Number of different color amplitudes. *)
    ("n_cindex", num_color_indices); (* Maximum rank of color tensors. *)
    ("n_flv", num_flavors); (* Number of different flavor amplitudes. *)
    ("n_hel", num_helicities) (* Number of different helicty amplitudes. *) ];
nl ();
```

Abbreviations.

```
printf "␣␣!␣NB:␣you␣MUST␣NOT␣change␣the␣value␣of␣%s␣here!!!" nc_parameter;
nl ();
printf "␣␣!␣␣␣␣␣It␣is␣defined␣here␣for␣convenience␣only␣and␣must␣be"; nl ();
printf "␣␣!␣␣␣␣␣compatible␣with␣hardcoded␣values␣in␣the␣amplitude!"; nl ();
print_real_parameter nc_parameter (CM.nc ()); (* N_C *)
List.iter
  (function name, value → print_logical_parameter name value)
  [ ("F", false); ("T", true) ]; nl ();

print_spin_tables amplitudes;
print_flavor_tables amplitudes;
print_color_tables amplitudes;
print_amplitude_table amplitudes;
```

```
    print_helicity_selection_table ()

  let print_interface () =
    print_md5sum_functions !md5sum;
    print_maintenance_functions ();
    List.iter print_numeric_inquiry_functions
      [("number_particles_in", "n_in");
       ("number_particles_out", "n_out")];
    List.iter print_inquiry_functions
      ["spin_states"; "flavor_states"];
    print_inquiry_function_openmp ();
    print_color_flows ();
    print_color_factors ();
    print_dispatch_functions ();
    nl ();
    current_continuation_line := 0;
    if !km_write ∨ !km_pure then (Targets_Kmatrix.Fortran.print !km_pure);
    current_continuation_line := 1;
    nl ()

  let print_calculate_amplitudes declarations computations amplitudes =
    printf "  @[<5>subroutine calculate_amplitudes (amp, k, mask)"; nl ();
    printf "    complex(kind=%s), dimension(:,:,:), intent(out) :: amp" !kind; nl ();
    printf "    real(kind=%s), dimension(0:3,*), intent(in) :: k" !kind; nl ();
    printf "    logical, dimension(:), intent(in) :: mask"; nl ();
    printf "    integer, dimension(n_prt) :: s"; nl ();
    printf "    integer :: h, hi"; nl ();
    declarations ();
    begin match CF.processes amplitudes with
    | p :: _ → print_external_momenta p
    | _ → ()
    end;
    ignore (List.fold_left print_momenta PSet.empty (CF.processes amplitudes));
    printf "    amp = 0"; nl ();
    if num_helicities amplitudes > 0 then begin
      printf "    if (hel_finite == 0) return"; nl ();
      if !openmp then begin
        printf "!$OMP PARALLEL DO DEFAULT(SHARED) PRIVATE(s, h, %s) SCHEDULE(STATIC)" openm
      end;
      printf "    do hi = 1, hel_finite"; nl ();
      printf "      h = hel_map(hi)"; nl ();
      printf "      s = table_spin_states(:,h)"; nl ();
      ignore (List.fold_left print_externals WFSet.empty (CF.processes amplitudes));
      computations ();
      List.iter print_fudge_factor (CF.processes amplitudes);
      (* This sorting should slightly improve cache locality. *)
      let triple_snd = fun (_, x, _) → x
      in let triple_fst = fun (x, _, _) → x
      in let rec builder1 flvi flowi flows = match flows with
        | (Some a) :: tl → (flvi, flowi, flavors_symbol (flavors a)) :: (builder1 flvi (flowi + 1) tl)
        | None :: tl → builder1 flvi (flowi + 1) tl
```

```
      | [] → []
    in let rec builder2 flvi flvs = match flvs with
      | flv :: tl → (builder1 flvi 1 flv) @ (builder2 (flvi + 1) tl)
      | [] → []
    in let unsorted = builder2 1 (List.map Array.to_list (Array.to_list (CF.process_table amplitudes)))
    in let sorted = List.sort (fun a b →
        if (triple_snd a ≢ triple_snd b) then triple_snd a − triple_snd b else (triple_fst a − triple_fst b)
      unsorted
    in List.iter (fun (flvi, flowi, flv) →
      (printf "      amp(%d,%d,h) = %s" flvi flowi flv; nl ();)) sorted;

    printf "    end do"; nl ();
    if !openmp then begin
      printf "!$OMP END PARALLEL DO"; nl ();
    end;
  end;
  printf "  end subroutine calculate_amplitudes"; nl ()

let print_compute_chops chopped_fusions chopped_brakets () =
  List.iter
    (fun (i, _) → printf "      call compute_fusions_%04d (%s)" i
        (if !openmp then openmp_tld else ""); nl ())
    chopped_fusions;
  List.iter
    (fun (i, _) → printf "      call compute_brakets_%04d (%s)" i
        (if !openmp then openmp_tld else ""); nl ())
    chopped_brakets
```

Single Function

```
let amplitudes_to_channel_single_function cmdline oc amplitudes =

  let print_declarations () =
    print_constants amplitudes

  and print_implementations () =
    print_interface ();
    print_calculate_amplitudes
      (fun () → print_variable_declarations amplitudes)
      (fun () →
        print_fusions (CF.dictionary amplitudes) (CF.fusions amplitudes);
        List.iter
          (print_brakets (CF.dictionary amplitudes))
          (CF.processes amplitudes))
      amplitudes in

  let fortran_module =
    { module_name = !module_name;
      used_modules = used_modules ();
      default_accessibility = Private;
      public_symbols = public_symbols ();
      print_declarations = [print_declarations];
```

```
      print_implementations = [print_implementations] } in
    setup_fortran_formatter !line_length oc;
    print_description cmdline amplitudes ();
    print_modules [fortran_module]
```

Single Module

```
let amplitudes_to_channel_single_module cmdline oc size amplitudes =
  let print_declarations () =
    print_constants amplitudes;
    print_variable_declarations amplitudes
  and print_implementations () =
    print_interface () in
  let chopped_fusions, chopped_brakets =
    chop_amplitudes size amplitudes in
  let dictionary = CF.dictionary amplitudes in
  let print_compute_amplitudes () =
    print_calculate_amplitudes
      (fun () → ())
      (print_compute_chops chopped_fusions chopped_brakets)
      amplitudes
  and print_compute_fusions () =
    List.iter (print_compute_fusions1 dictionary) chopped_fusions
  and print_compute_brakets () =
    List.iter (print_compute_brakets1 dictionary) chopped_brakets in
  let fortran_module =
    { module_name = !module_name;
      used_modules = used_modules ();
      default_accessibility = Private;
      public_symbols = public_symbols ();
      print_declarations = [print_declarations];
      print_implementations = [print_implementations;
                               print_compute_amplitudes;
                               print_compute_fusions;
                               print_compute_brakets] } in
  setup_fortran_formatter !line_length oc;
  print_description cmdline amplitudes ();
  print_modules [fortran_module]
```

Multiple Modules

```
let modules_of_amplitudes _ _ size amplitudes =
  let name = !module_name in
```

```
let print_declarations () =
  print_constants amplitudes
and print_variables () =
  print_variable_declarations amplitudes in

let constants_module =
  { module_name = name ^ "_constants";
    used_modules = used_modules ();
    default_accessibility = Public;
    public_symbols = [];
    print_declarations = [print_declarations];
    print_implementations = [] } in

let variables_module =
  { module_name = name ^ "_variables";
    used_modules = used_modules ();
    default_accessibility = Public;
    public_symbols = [];
    print_declarations = [print_variables];
    print_implementations = [] } in

let dictionary = CF.dictionary amplitudes in

let print_compute_fusions (n, fusions) () =
  if !openmp then begin
    printf "␣␣subroutine␣compute_fusions_%04d␣(%s)" n openmp_tld; nl ();
    printf "␣␣@[<5>type(%s),␣intent(inout)␣::␣%s" openmp_tld_type openmp_tld; nl ();
  end else begin
    printf "␣␣@[<5>subroutine␣compute_fusions_%04d␣()" n; nl ();
  end;
  print_fusions dictionary fusions;
  printf "␣␣end␣subroutine␣compute_fusions_%04d" n; nl () in

let print_compute_brakets (n, processes) () =
  if !openmp then begin
    printf "␣␣subroutine␣compute_brakets_%04d␣(%s)" n openmp_tld; nl ();
    printf "␣␣@[<5>type(%s),␣intent(inout)␣::␣%s" openmp_tld_type openmp_tld; nl ();
  end else begin
    printf "␣␣@[<5>subroutine␣compute_brakets_%04d␣()" n; nl ();
  end;
  List.iter (print_brakets dictionary) processes;
  printf "␣␣end␣subroutine␣compute_brakets_%04d" n; nl () in

let fusions_module (n, _ as fusions) =
  let tag = Printf.sprintf "_fusions_%04d" n in
  { module_name = name ^ tag;
    used_modules = (used_modules () @
                    [Full constants_module.module_name;
                     Full variables_module.module_name]);
    default_accessibility = Private;
    public_symbols = ["compute" ^ tag];
    print_declarations = [];
    print_implementations = [print_compute_fusions fusions] } in
```

```
  let brakets_module (n, _ as processes) =
    let tag = Printf.sprintf "_brakets_%04d" n in
    { module_name = name ^ tag;
      used_modules = (used_modules () @
                      [Full constants_module.module_name;
                       Full variables_module.module_name]);
      default_accessibility = Private;
      public_symbols = ["compute" ^ tag];
      print_declarations = [];
      print_implementations = [print_compute_brakets processes] } in

  let chopped_fusions, chopped_brakets =
    chop_amplitudes size amplitudes in

  let fusions_modules =
    List.map fusions_module chopped_fusions in

  let brakets_modules =
    List.map brakets_module chopped_brakets in

  let print_implementations () =
    print_interface ();
    print_calculate_amplitudes
      (fun () -> ())
      (print_compute_chops chopped_fusions chopped_brakets)
      amplitudes in

  let public_module =
    { module_name = name;
       used_modules = (used_modules () @
                       [Full constants_module.module_name;
                        Full variables_module.module_name ] @
                       List.map
                         (fun m -> Full m.module_name)
                         (fusions_modules @ brakets_modules));
      default_accessibility = Private;
      public_symbols = public_symbols ();
      print_declarations = [];
      print_implementations = [print_implementations] }
  and private_modules =
    [constants_module; variables_module] @ fusions_modules @ brakets_modules in

  (public_module, private_modules)

let amplitudes_to_channel_single_file cmdline oc size amplitudes =
  let public_module, private_modules =
    modules_of_amplitudes cmdline oc size amplitudes in
  setup_fortran_formatter !line_length oc;
  print_description cmdline amplitudes ();
  print_modules (private_modules @ [public_module])

let amplitudes_to_channel_multi_file cmdline oc size amplitudes =
  let public_module, private_modules =
    modules_of_amplitudes cmdline oc size amplitudes in
```

```
    modules_to_file !line_length oc
      (print_description cmdline amplitudes)
      (public_module :: private_modules)
```

Dispatch

```
  let amplitudes_to_channel cmdline oc diagnostics amplitudes =
    parse_diagnostics diagnostics;
    match !output_mode with
    | Single_Function →
        amplitudes_to_channel_single_function cmdline oc amplitudes
    | Single_Module size →
        amplitudes_to_channel_single_module cmdline oc size amplitudes
    | Single_File size →
        amplitudes_to_channel_single_file cmdline oc size amplitudes
    | Multi_File size →
        amplitudes_to_channel_multi_file cmdline oc size amplitudes

  let parameters_to_channel oc =
    parameters_to_fortran oc (CM.parameters ())

end

module Fortran = Make_Fortran(Fortran_Fermions)
```

Majorana Fermions

JR sez' (regarding the Majorana Feynman rules): For this function we need a different approach due to our aim of implementing the fermion vertices with the right line as ingoing (in a calculational sense) and the left line in a fusion as outgoing. In defining all external lines and the fermionic wavefunctions built out of them as ingoing we have to invert the left lines to make them outgoing. This happens by multiplying them with the inverse charge conjugation matrix in an appropriate representation and then transposing it. We must distinguish whether the direction of calculation and the physical direction of the fermion number flow are parallel or antiparallel. In the first case we can use the "normal" Feynman rules for Dirac particles, while in the second, according to the paper of Denner et al., we have to reverse the sign of the vector and antisymmetric bilinears of the Dirac spinors, cf. the *Coupling* module.

Note the subtlety for the left- and righthanded couplings: Only the vector part of these couplings changes in the appropriate cases its sign, changing the chirality to the negative of the opposite. *(JR's probably right, but I need to check myself ...)*

```
module Fortran_Majorana_Fermions : Fermions =
  struct
    let rcs = RCS.rename rcs_file "Targets.Fortran_Majorana_Fermions()"
        [ "generates Fortran95 code for Dirac and Majorana fermions";
```

```
        " using revision 2003_03_A of module omega95_bispinors" ]
open Coupling
open Format

let psi_type = "bispinor"
let psibar_type = "bispinor"
let chi_type = "bispinor"
let grav_type = "vectorspinor"
```

JR sez' (regarding the Majorana Feynman rules): Because of our rules for fermions we are going to give all incoming fermions a u spinor and all outgoing fermions a v spinor, no matter whether they are Dirac fermions, antifermions or Majorana fermions. *(JR's probably right, but I need to check myself ...)*

```
let psi_incoming = "u"
let brs_psi_incoming = "brs_u"
let psibar_incoming = "u"
let brs_psibar_incoming = "brs_u"
let chi_incoming = "u"
let brs_chi_incoming = "brs_u"
let grav_incoming = "ueps"

let psi_outgoing = "v"
let brs_psi_outgoing = "brs_v"
let psibar_outgoing = "v"
let brs_psibar_outgoing = "brs_v"
let chi_outgoing = "v"
let brs_chi_outgoing = "brs_v"
let grav_outgoing = "veps"

let psi_propagator = "pr_psi"
let psibar_propagator = "pr_psi"
let chi_propagator = "pr_psi"
let grav_propagator = "pr_grav"

let psi_projector = "pj_psi"
let psibar_projector = "pj_psi"
let chi_projector = "pj_psi"
let grav_projector = "pj_grav"

let psi_gauss = "pg_psi"
let psibar_gauss = "pg_psi"
let chi_gauss = "pg_psi"
let grav_gauss = "pg_grav"

let format_coupling coeff c =
  match coeff with
  | 1 -> c
  | -1 -> "(-" ^ c ^")"
  | coeff -> string_of_int coeff ^ "*" ^ c

let format_coupling_2 coeff c =
```

```
match coeff with
| 1 → c
| −1 → "-" ^ c
| coeff → string_of_int coeff ^ "*" ^ c
```

JR's coupling constant HACK, necessitated by tho's bad design descition.

```
let fastener s i =
  try
    let offset = (String.index s '(') in
    if ((String.get s (String.length s − 1)) ≢ ')') then
      failwith "fastener:␣wrong␣usage␣of␣parentheses"
    else
      let func_name = (String.sub s 0 offset) and
          tail =
            (String.sub s (succ offset) (String.length s − offset − 2)) in
      if (String.contains func_name ')') ∨
         (String.contains tail '(') ∨
         (String.contains tail ')') then
        failwith "fastener:␣wrong␣usage␣of␣parentheses"
      else
        func_name ^ "(" ^ string_of_int i ^ "," ^ tail ^ ")"
  with
  | Not_found →
      if (String.contains s ')') then
        failwith "fastener:␣wrong␣usage␣of␣parentheses"
      else
        s ^ "(" ^ string_of_int i ^ ")"

let print_fermion_current coeff f c wf1 wf2 fusion =
  let c = format_coupling coeff c in
  match fusion with
  | F13 | F31 → printf "%s_ff(%s,%s,%s)" f c wf1 wf2
  | F23 | F21 → printf "f_%sf(%s,%s,%s)" f c wf1 wf2
  | F32 | F12 → printf "f_%sf(%s,%s,%s)" f c wf2 wf1

let print_fermion_current2 coeff f c wf1 wf2 fusion =
  let c = format_coupling_2 coeff c in
  let c1 = fastener c 1 and
      c2 = fastener c 2 in
  match fusion with
  | F13 | F31 → printf "%s_ff(%s,%s,%s,%s)" f c1 c2 wf1 wf2
  | F23 | F21 → printf "f_%sf(%s,%s,%s,%s)" f c1 c2 wf1 wf2
  | F32 | F12 → printf "f_%sf(%s,%s,%s,%s)" f c1 c2 wf2 wf1

let print_fermion_current_vector coeff f c wf1 wf2 fusion =
  let c = format_coupling coeff c in
  match fusion with
  | F13 → printf "%s_ff(%s,%s,%s)" f c wf1 wf2
  | F31 → printf "%s_ff(-%s,%s,%s)" f c wf1 wf2
  | F23 → printf "f_%sf(%s,%s,%s)" f c wf1 wf2
```

```
| F32 → printf "f_%sf(%s,%s,%s)" f c wf2 wf1
| F12 → printf "f_%sf(-%s,%s,%s)" f c wf2 wf1
| F21 → printf "f_%sf(-%s,%s,%s)" f c wf1 wf2

let print_fermion_current2_vector coeff f c wf1 wf2 fusion =
  let c = format_coupling_2 coeff c in
  let c1 = fastener c 1 and
      c2 = fastener c 2 in
  match fusion with
  | F13 → printf "%s_ff(%s,%s,%s,%s)" f c1 c2 wf1 wf2
  | F31 → printf "%s_ff(-(%s),%s,%s,%s)" f c1 c2 wf1 wf2
  | F23 → printf "f_%sf(%s,%s,%s,%s)" f c1 c2 wf1 wf2
  | F32 → printf "f_%sf(%s,%s,%s,%s)" f c1 c2 wf2 wf1
  | F12 → printf "f_%sf(-(%s),%s,%s,%s)" f c1 c2 wf2 wf1
  | F21 → printf "f_%sf(-(%s),%s,%s,%s)" f c1 c2 wf1 wf2

let print_fermion_current_chiral coeff f1 f2 c wf1 wf2 fusion =
  let c = format_coupling coeff c in
  match fusion with
  | F13 → printf "%s_ff(%s,%s,%s)" f1 c wf1 wf2
  | F31 → printf "%s_ff(-%s,%s,%s)" f2 c wf1 wf2
  | F23 → printf "f_%sf(%s,%s,%s)" f1 c wf1 wf2
  | F32 → printf "f_%sf(%s,%s,%s)" f1 c wf2 wf1
  | F12 → printf "f_%sf(-%s,%s,%s)" f2 c wf2 wf1
  | F21 → printf "f_%sf(-%s,%s,%s)" f2 c wf1 wf2

let print_fermion_current2_chiral coeff f c wf1 wf2 fusion =
  let c = format_coupling_2 coeff c in
  let c1 = fastener c 1 and
      c2 = fastener c 2 in
  match fusion with
  | F13 → printf "%s_ff(%s,%s,%s,%s)" f c1 c2 wf1 wf2
  | F31 → printf "%s_ff(-(%s),-(%s),%s,%s)" f c2 c1 wf1 wf2
  | F23 → printf "f_%sf(%s,%s,%s,%s)" f c1 c2 wf1 wf2
  | F32 → printf "f_%sf(%s,%s,%s,%s)" f c1 c2 wf2 wf1
  | F12 → printf "f_%sf(-(%s),-(%s),%s,%s)" f c2 c1 wf2 wf1
  | F21 → printf "f_%sf(-(%s),-(%s),%s,%s)" f c2 c1 wf1 wf2

let print_current = function
  | coeff, _, VA, _ → print_fermion_current2_vector coeff "va"
  | coeff, _, V, _ → print_fermion_current_vector coeff "v"
  | coeff, _, A, _ → print_fermion_current coeff "a"
  | coeff, _, VL, _ → print_fermion_current_chiral coeff "vl" "vr"
  | coeff, _, VR, _ → print_fermion_current_chiral coeff "vr" "vl"
  | coeff, _, VLR, _ → print_fermion_current2_chiral coeff "vlr"
  | coeff, _, SP, _ → print_fermion_current2 coeff "sp"
  | coeff, _, S, _ → print_fermion_current coeff "s"
  | coeff, _, P, _ → print_fermion_current coeff "p"
  | coeff, _, SL, _ → print_fermion_current coeff "sl"
  | coeff, _, SR, _ → print_fermion_current coeff "sr"
  | coeff, _, SLR, _ → print_fermion_current2 coeff "slr"
  | coeff, _, POT, _ → print_fermion_current_vector coeff "pot"
```

```
| _, _, _, _ → invalid_arg
      "Targets.Fortran_Majorana_Fermions:␣Not␣needed␣in␣the␣models"
let print_current_p = function
  | coeff, Psi, SL, Psi → print_fermion_current coeff "sl"
  | coeff, Psi, SR, Psi → print_fermion_current coeff "sr"
  | coeff, Psi, SLR, Psi → print_fermion_current2 coeff "slr"
  | _, _, _, _ → invalid_arg
      "Targets.Fortran_Majorana_Fermions:␣Not␣needed␣in␣the␣used␣models"
let print_current_b = function
  | coeff, Psibar, SL, Psibar → print_fermion_current coeff "sl"
  | coeff, Psibar, SR, Psibar → print_fermion_current coeff "sr"
  | coeff, Psibar, SLR, Psibar → print_fermion_current2 coeff "slr"
  | _, _, _, _ → invalid_arg
      "Targets.Fortran_Majorana_Fermions:␣Not␣needed␣in␣the␣used␣models"
```

This function is for the vertices with three particles including two fermions but also a momentum, therefore with a dimensionful coupling constant, e.g. the gravitino vertices. One has to dinstinguish between the two kinds of canonical orders in the string of gamma matrices. Of course, the direction of the string of gamma matrices is reversed if one goes from the *Gravbar*, _, *Psi* to the *Psibar*, _, *Grav* vertices, and the same is true for the couplings of the gravitino to the Majorana fermions. For more details see the tables in the *coupling* implementation.
We now have to fix the directions of the momenta. For making the compiler happy and because we don't want to make constructions of infinite complexity we list the momentum including vertices without gravitinos here; the pattern matching says that's better. Perhaps we have to find a better name now.

For the cases of MOM, $MOM5$, $MOML$ and $MOMR$ which arise only in BRST transformations we take the mass as a coupling constant. For $VMOM$ we don't need a mass either. These vertices are like kinetic terms and so need not have a coupling constant. By this we avoid a strange and awful construction with a new variable. But be careful with a generalization if you want to use these vertices for other purposes.

```
let format_coupling_mom coeff c =
  match coeff with
  | 1 → c
  | -1 → "(-" ^ c ^")"
  | coeff → string_of_int coeff ^ "*" ^ c
let commute_proj f =
  match f with
  | "moml" → "lmom"
  | "momr" → "rmom"
  | "lmom" → "moml"
  | "rmom" → "momr"
  | "svl" → "svr"
  | "svr" → "svl"
  | "sl" → "sr"
  | "sr" → "sl"
  | "s" → "s"
```

```
  | "p" → "p"
  | _ → invalid_arg "Targets:Fortran_Majorana_Fermions:␣wrong␣case"

let print_fermion_current_mom coeff f c wf1 wf2 p1 p2 p12 fusion =
  let c = format_coupling_mom coeff c in
  let c1 = fastener c 1 and
      c2 = fastener c 2 in
  match fusion with
  | F13 → printf "%s_ff(%s,%s,%s,%s,%s)" f c1 c2 wf1 wf2 p12
  | F31 → printf "%s_ff(%s,%s,%s,%s,%s)" f c1 c2 wf1 wf2 p12
  | F23 → printf "f_%sf(%s,%s,%s,%s,%s)" f c1 c2 wf1 wf2 p1
  | F32 → printf "f_%sf(%s,%s,%s,%s,%s)" f c1 c2 wf2 wf1 p2
  | F12 → printf "f_%sf(%s,%s,%s,%s,%s)" f c1 c2 wf2 wf1 p2
  | F21 → printf "f_%sf(%s,%s,%s,%s,%s)" f c1 c2 wf1 wf2 p1

let print_fermion_current_mom_sign coeff f c wf1 wf2 p1 p2 p12 fusion =
  let c = format_coupling_mom coeff c in
  let c1 = fastener c 1 and
      c2 = fastener c 2 in
  match fusion with
  | F13 → printf "%s_ff(%s,%s,%s,%s,%s)" f c1 c2 wf1 wf2 p12
  | F31 → printf "%s_ff(%s,%s,%s,%s,-(%s))" f c1 c2 wf1 wf2 p12
  | F23 → printf "f_%sf(%s,%s,%s,%s,%s)" f c1 c2 wf1 wf2 p1
  | F32 → printf "f_%sf(%s,%s,%s,%s,%s)" f c1 c2 wf2 wf1 p2
  | F12 → printf "f_%sf(%s,%s,%s,%s,-(%s))" f c1 c2 wf2 wf1 p2
  | F21 → printf "f_%sf(%s,%s,%s,%s,-(%s))" f c1 c2 wf1 wf2 p1

let print_fermion_current_mom_sign_1 coeff f c wf1 wf2 p1 p2 p12 fusion =
  let c = format_coupling coeff c in
  match fusion with
  | F13 → printf "%s_ff(%s,%s,%s,%s)" f c wf1 wf2 p12
  | F31 → printf "%s_ff(%s,%s,%s,-(%s))" f c wf1 wf2 p12
  | F23 → printf "f_%sf(%s,%s,%s,%s)" f c wf1 wf2 p1
  | F32 → printf "f_%sf(%s,%s,%s,%s)" f c wf2 wf1 p2
  | F12 → printf "f_%sf(%s,%s,%s,-(%s))" f c wf2 wf1 p2
  | F21 → printf "f_%sf(%s,%s,%s,-(%s))" f c wf1 wf2 p1

let print_fermion_current_mom_chiral coeff f c wf1 wf2 p1 p2 p12 fusion =
  let c = format_coupling_mom coeff c and
      cf = commute_proj f in
  let c1 = fastener c 1 and
      c2 = fastener c 2 in
  match fusion with
  | F13 → printf "%s_ff(%s,%s,%s,%s,%s)" f c1 c2 wf1 wf2 p12
  | F31 → printf "%s_ff(%s,%s,%s,␣%s,-(%s))" cf c1 c2 wf1 wf2 p12
  | F23 → printf "f_%sf(%s,%s,%s,%s,%s)" f c1 c2 wf1 wf2 p1
  | F32 → printf "f_%sf(%s,%s,%s,%s,%s)" f c1 c2 wf2 wf1 p2
  | F12 → printf "f_%sf(%s,%s,%s,%s,-(%s))" cf c1 c2 wf2 wf1 p2
  | F21 → printf "f_%sf(%s,%s,%s,%s,-(%s))" cf c1 c2 wf1 wf2 p1

let print_fermion_g_current coeff f c wf1 wf2 p1 p2 p12 fusion =
  let c = format_coupling coeff c in
  match fusion with
```

```
  | F13 → printf "%s_grf(%s,%s,%s,%s)" f c wf1 wf2 p12
  | F31 → printf "%s_fgr(%s,%s,%s,%s)" f c wf1 wf2 p12
  | F23 → printf "gr_%sf(%s,%s,%s,%s)" f c wf1 wf2 p1
  | F32 → printf "gr_%sf(%s,%s,%s,%s)" f c wf2 wf1 p2
  | F12 → printf "f_%sgr(%s,%s,%s,%s)" f c wf2 wf1 p2
  | F21 → printf "f_%sgr(%s,%s,%s,%s)" f c wf1 wf2 p1
let print_fermion_g_2_current coeff f c wf1 wf2 p1 p2 p12 fusion =
  let c = format_coupling coeff c in
  match fusion with
  | F13 → printf "%s_grf(%s(1),%s(2),%s,%s,%s)" f c c wf1 wf2 p12
  | F31 → printf "%s_fgr(%s(1),%s(2),%s,%s,%s)" f c c wf1 wf2 p12
  | F23 → printf "gr_%sf(%s(1),%s(2),%s,%s,%s)" f c c wf1 wf2 p1
  | F32 → printf "gr_%sf(%s(1),%s(2),%s,%s,%s)" f c c wf2 wf1 p2
  | F12 → printf "f_%sgr(%s(1),%s(2),%s,%s,%s)" f c c wf2 wf1 p2
  | F21 → printf "f_%sgr(%s(1),%s(2),%s,%s,%s)" f c c wf1 wf2 p1
let print_fermion_g_current_rev coeff f c wf1 wf2 p1 p2 p12 fusion =
  let c = format_coupling coeff c in
  match fusion with
  | F13 → printf "%s_fgr(%s,%s,%s,%s)" f c wf1 wf2 p12
  | F31 → printf "%s_grf(%s,%s,%s,%s)" f c wf1 wf2 p12
  | F23 → printf "f_%sgr(%s,%s,%s,%s)" f c wf1 wf2 p1
  | F32 → printf "f_%sgr(%s,%s,%s,%s)" f c wf2 wf1 p2
  | F12 → printf "gr_%sf(%s,%s,%s,%s)" f c wf2 wf1 p2
  | F21 → printf "gr_%sf(%s,%s,%s,%s)" f c wf1 wf2 p1
let print_fermion_g_2_current_rev coeff f c wf1 wf2 p1 p2 p12 fusion =
  let c = format_coupling coeff c in
  match fusion with
  | F13 → printf "%s_fgr(%s(1),%s(2),%s,%s,%s)" f c c wf1 wf2 p12
  | F31 → printf "%s_grf(%s(1),%s(2),%s,%s,%s)" f c c wf1 wf2 p12
  | F23 → printf "f_%sgr(%s(1),%s(2),%s,%s,%s)" f c c wf1 wf2 p1
  | F32 → printf "f_%sgr(%s(1),%s(2),%s,%s,%s)" f c c wf2 wf1 p2
  | F12 → printf "gr_%sf(%s(1),%s(2),%s,%s,%s)" f c c wf2 wf1 p2
  | F21 → printf "gr_%sf(%s(1),%s(2),%s,%s,%s)" f c c wf1 wf2 p1
let print_fermion_g_current_vector coeff f c wf1 wf2 _ _ _ fusion =
  let c = format_coupling coeff c in
  match fusion with
  | F13 → printf "%s_grf(%s,%s,%s)" f c wf1 wf2
  | F31 → printf "%s_fgr(-%s,%s,%s)" f c wf1 wf2
  | F23 → printf "gr_%sf(%s,%s,%s)" f c wf1 wf2
  | F32 → printf "gr_%sf(%s,%s,%s)" f c wf2 wf1
  | F12 → printf "f_%sgr(-%s,%s,%s)" f c wf2 wf1
  | F21 → printf "f_%sgr(-%s,%s,%s)" f c wf1 wf2
let print_fermion_g_current_vector_rev coeff f c wf1 wf2 _ _ _ fusion =
  let c = format_coupling coeff c in
  match fusion with
  | F13 → printf "%s_fgr(%s,%s,%s)" f c wf1 wf2
  | F31 → printf "%s_grf(-%s,%s,%s)" f c wf1 wf2
  | F23 → printf "f_%sgr(%s,%s,%s)" f c wf1 wf2
```

```
| F32 → printf "f_%sgr(%s,%s,%s)" f c wf2 wf1
| F12 → printf "gr_%sf(-%s,%s,%s)" f c wf2 wf1
| F21 → printf "gr_%sf(-%s,%s,%s)" f c wf1 wf2

let print_current_g = function
  | coeff, _, MOM, _ → print_fermion_current_mom_sign coeff "mom"
  | coeff, _, MOM5, _ → print_fermion_current_mom coeff "mom5"
  | coeff, _, MOML, _ → print_fermion_current_mom_chiral coeff "moml"
  | coeff, _, MOMR, _ → print_fermion_current_mom_chiral coeff "momr"
  | coeff, _, LMOM, _ → print_fermion_current_mom_chiral coeff "lmom"
  | coeff, _, RMOM, _ → print_fermion_current_mom_chiral coeff "rmom"
  | coeff, _, VMOM, _ → print_fermion_current_mom_sign_1 coeff "vmom"
  | coeff, Gravbar, S, _ → print_fermion_g_current coeff "s"
  | coeff, Gravbar, SL, _ → print_fermion_g_current coeff "sl"
  | coeff, Gravbar, SR, _ → print_fermion_g_current coeff "sr"
  | coeff, Gravbar, SLR, _ → print_fermion_g_2_current coeff "slr"
  | coeff, Gravbar, P, _ → print_fermion_g_current coeff "p"
  | coeff, Gravbar, V, _ → print_fermion_g_current coeff "v"
  | coeff, Gravbar, VLR, _ → print_fermion_g_2_current coeff "vlr"
  | coeff, Gravbar, POT, _ → print_fermion_g_current_vector coeff "pot"
  | coeff, _, S, Grav → print_fermion_g_current_rev coeff "s"
  | coeff, _, SL, Grav → print_fermion_g_current_rev coeff "sl"
  | coeff, _, SR, Grav → print_fermion_g_current_rev coeff "sr"
  | coeff, _, SLR, Grav → print_fermion_g_2_current_rev coeff "slr"
  | coeff, _, P, Grav → print_fermion_g_current_rev (−coeff) "p"
  | coeff, _, V, Grav → print_fermion_g_current_rev coeff "v"
  | coeff, _, VLR, Grav → print_fermion_g_2_current_rev coeff "vlr"
  | coeff, _, POT, Grav → print_fermion_g_current_vector_rev coeff "pot"
  | _, _, _, _ → invalid_arg
      "Targets.Fortran_Majorana_Fermions:␣not␣used␣in␣the␣models"

let print_current_mom = function
  | _, _, _, _ → invalid_arg
      "Targets.Fortran_Majorana_Fermions:␣Not␣needed␣in␣the␣models"
```

We need support for dimension-5 vertices with two fermions and two bosons, appearing in theories of supergravity and also together with in insertions of the supersymmetric current. There is a canonical order *fermionbar*, *boson_1*, *boson_2*, *fermion*, so what one has to do is a mapping from the fusions *F123* etc. to the order of the three wave functions *wf1*, *wf2* and *wf3*.
The function *d_p* (for distinct the particle) distinguishes which particle (scalar or vector) must be fused to in the special functions.

```
let d_p = function
  | 1, ("sv"| "pv"| "svl"| "svr"| "slrv") → "1"
  | 1, _ → ""
  | 2, ("sv"| "pv"| "svl"| "svr"| "slrv") → "2"
  | 2, _ → ""
  | _, _ → invalid_arg "Targets.Fortran_Majorana_Fermions:␣not␣used"

let wf_of_f wf1 wf2 wf3 f =
  match f with
  | (F123 | F423) → [wf2; wf3; wf1]
```

```
    | (F213 | F243 | F143 | F142 | F413 | F412) → [wf1; wf3; wf2]
    | (F132 | F432) → [wf3; wf2; wf1]
    | (F231 | F234 | F134 | F124 | F431 | F421) → [wf1; wf2; wf3]
    | (F312 | F342) → [wf3; wf1; wf2]
    | (F321 | F324 | F314 | F214 | F341 | F241) → [wf2; wf1; wf3]

let print_fermion_g4_brs_vector_current coeff f c wf1 wf2 wf3 fusion =
  let cf = commute_proj f and
      cp = format_coupling coeff c and
      cm = if f = "pv" then
        format_coupling coeff c
      else
        format_coupling (−coeff) c
  and
      d1 = d_p (1,f) and
      d2 = d_p (2,f) and
      f1 = (List.nth (wf_of_f wf1 wf2 wf3 fusion) 0) and
      f2 = (List.nth (wf_of_f wf1 wf2 wf3 fusion) 1) and
      f3 = (List.nth (wf_of_f wf1 wf2 wf3 fusion) 2) in
  match fusion with
  | (F123 | F213 | F132 | F231 | F312 | F321) →
      printf "f_%sf(%s,%s,%s,%s)" cf cm f1 f2 f3
  | (F423 | F243 | F432 | F234 | F342 | F324) →
      printf "f_%sf(%s,%s,%s,%s)" f cp f1 f2 f3
  | (F134 | F143 | F314) → printf "%s%s_ff(%s,%s,%s,%s)" f d1 cp f1 f2 f3
  | (F124 | F142 | F214) → printf "%s%s_ff(%s,%s,%s,%s)" f d2 cp f1 f2 f3
  | (F413 | F431 | F341) → printf "%s%s_ff(%s,%s,%s,%s)" cf d1 cm f1 f2 f3
  | (F241 | F412 | F421) → printf "%s%s_ff(%s,%s,%s,%s)" cf d2 cm f1 f2 f3

let print_fermion_g4_svlr_current coeff _ c wf1 wf2 wf3 fusion =
  let c = format_coupling_2 coeff c and
      f1 = (List.nth (wf_of_f wf1 wf2 wf3 fusion) 0) and
      f2 = (List.nth (wf_of_f wf1 wf2 wf3 fusion) 1) and
      f3 = (List.nth (wf_of_f wf1 wf2 wf3 fusion) 2) in
  let c1 = fastener c 1 and
      c2 = fastener c 2 in
  match fusion with
  | (F123 | F213 | F132 | F231 | F312 | F321) →
      printf "f_svlrf(-(%s),-(%s),%s,%s,%s)" c2 c1 f1 f2 f3
  | (F423 | F243 | F432 | F234 | F342 | F324) →
      printf "f_svlrf(%s,%s,%s,%s,%s)" c1 c2 f1 f2 f3
  | (F134 | F143 | F314) →
      printf "svlr2_ff(%s,%s,%s,%s,%s)" c1 c2 f1 f2 f3
  | (F124 | F142 | F214) →
      printf "svlr1_ff(%s,%s,%s,%s,%s)" c1 c2 f1 f2 f3
  | (F413 | F431 | F341) →
      printf "svlr2_ff(-(%s),-(%s),%s,%s,%s)" c2 c1 f1 f2 f3
  | (F241 | F412 | F421) →
      printf "svlr1_ff(-(%s),-(%s),%s,%s,%s)" c2 c1 f1 f2 f3

let print_fermion_s2_current coeff f c wf1 wf2 wf3 fusion =
  let cp = format_coupling coeff c and
```

```
        cm = if f = "p" then
          format_coupling (−coeff) c
        else
          format_coupling coeff c
  and
        cf = commute_proj f and
        f1 = (List.nth (wf_of_f wf1 wf2 wf3 fusion) 0) and
        f2 = (List.nth (wf_of_f wf1 wf2 wf3 fusion) 1) and
        f3 = (List.nth (wf_of_f wf1 wf2 wf3 fusion) 2) in
  match fusion with
  | (F123 | F213 | F132 | F231 | F312 | F321) →
        printf "%s␣*␣f_%sf(%s,%s,%s)" f1 cf cm f2 f3
  | (F423 | F243 | F432 | F234 | F342 | F324) →
        printf "%s␣*␣f_%sf(%s,%s,%s)" f1 f cp f2 f3
  | (F134 | F143 | F314) →
        printf "%s␣*␣%s_ff(%s,%s,%s)" f2 f cp f1 f3
  | (F124 | F142 | F214) →
        printf "%s␣*␣%s_ff(%s,%s,%s)" f2 f cp f1 f3
  | (F413 | F431 | F341) →
        printf "%s␣*␣%s_ff(%s,%s,%s)" f2 cf cm f1 f3
  | (F241 | F412 | F421) →
        printf "%s␣*␣%s_ff(%s,%s,%s)" f2 cf cm f1 f3

let print_fermion_s2p_current coeff f c wf1 wf2 wf3 fusion =
  let c = format_coupling_2 coeff c and
        f1 = (List.nth (wf_of_f wf1 wf2 wf3 fusion) 0) and
        f2 = (List.nth (wf_of_f wf1 wf2 wf3 fusion) 1) and
        f3 = (List.nth (wf_of_f wf1 wf2 wf3 fusion) 2) in
  let c1 = fastener c 1 and
        c2 = fastener c 2 in
  match fusion with
  | (F123 | F213 | F132 | F231 | F312 | F321) →
        printf "%s␣*␣f_%sf(%s,-(%s),%s,%s)" f1 f c1 c2 f2 f3
  | (F423 | F243 | F432 | F234 | F342 | F324) →
        printf "%s␣*␣f_%sf(%s,%s,%s,%s)" f1 f c1 c2 f2 f3
  | (F134 | F143 | F314) →
        printf "%s␣*␣%s_ff(%s,%s,%s,%s)" f2 f c1 c2 f1 f3
  | (F124 | F142 | F214) →
        printf "%s␣*␣%s_ff(%s,%s,%s,%s)" f2 f c1 c2 f1 f3
  | (F413 | F431 | F341) →
        printf "%s␣*␣%s_ff(%s,-(%s),%s,%s)" f2 f c1 c2 f1 f3
  | (F241 | F412 | F421) →
        printf "%s␣*␣%s_ff(%s,-(%s),%s,%s)" f2 f c1 c2 f1 f3

let print_fermion_s2lr_current coeff f c wf1 wf2 wf3 fusion =
  let c = format_coupling_2 coeff c and
        f1 = (List.nth (wf_of_f wf1 wf2 wf3 fusion) 0) and
        f2 = (List.nth (wf_of_f wf1 wf2 wf3 fusion) 1) and
        f3 = (List.nth (wf_of_f wf1 wf2 wf3 fusion) 2) in
  let c1 = fastener c 1 and
        c2 = fastener c 2 in
```

```
match fusion with
| (F123 | F213 | F132 | F231 | F312 | F321) →
    printf "%s * f_%sf(%s,%s,%s,%s)" f1 f c2 c1 f2 f3
| (F423 | F243 | F432 | F234 | F342 | F324) →
    printf "%s * f_%sf(%s,%s,%s,%s)" f1 f c1 c2 f2 f3
| (F134 | F143 | F314) →
    printf "%s * %s_ff(%s,%s,%s,%s)" f2 f c1 c2 f1 f3
| (F124 | F142 | F214) →
    printf "%s * %s_ff(%s,%s,%s,%s)" f2 f c1 c2 f1 f3
| (F413 | F431 | F341) →
    printf "%s * %s_ff(%s,%s,%s,%s)" f2 f c2 c1 f1 f3
| (F241 | F412 | F421) →
    printf "%s * %s_ff(%s,%s,%s,%s)" f2 f c2 c1 f1 f3

let print_fermion_g4_current coeff f c wf1 wf2 wf3 fusion =
  let c = format_coupling coeff c and
      f1 = (List.nth (wf_of_f wf1 wf2 wf3 fusion) 0) and
      f2 = (List.nth (wf_of_f wf1 wf2 wf3 fusion) 1) and
      f3 = (List.nth (wf_of_f wf1 wf2 wf3 fusion) 2) in
  match fusion with
  | (F123 | F213 | F132 | F231 | F312 | F321) →
      printf "f_%sgr(-%s,%s,%s,%s)" f c f1 f2 f3
  | (F423 | F243 | F432 | F234 | F342 | F324) →
      printf "gr_%sf(%s,%s,%s,%s)" f c f1 f2 f3
  | (F134 | F143 | F314 | F124 | F142 | F214) →
      printf "%s_grf(%s,%s,%s,%s)" f c f1 f2 f3
  | (F413 | F431 | F341 | F241 | F412 | F421) →
      printf "%s_fgr(-%s,%s,%s,%s)" f c f1 f2 f3

let print_fermion_2_g4_current coeff f c wf1 wf2 wf3 fusion =
  let f1 = (List.nth (wf_of_f wf1 wf2 wf3 fusion) 0) and
      f2 = (List.nth (wf_of_f wf1 wf2 wf3 fusion) 1) and
      f3 = (List.nth (wf_of_f wf1 wf2 wf3 fusion) 2) in
  let c = format_coupling_2 coeff c in
  let c1 = fastener c 1 and
      c2 = fastener c 2 in
  match fusion with
  | (F123 | F213 | F132 | F231 | F312 | F321) →
      printf "f_%sgr(-(%s),-(%s),%s,%s,%s)" f c2 c1 f1 f2 f3
  | (F423 | F243 | F432 | F234 | F342 | F324) →
      printf "gr_%sf(%s,%s,%s,%s,%s)" f c1 c2 f1 f2 f3
  | (F134 | F143 | F314 | F124 | F142 | F214) →
      printf "%s_grf(%s,%s,%s,%s,%s)" f c1 c2 f1 f2 f3
  | (F413 | F431 | F341 | F241 | F412 | F421) →
      printf "%s_fgr(-(%s),-(%s),%s,%s,%s)" f c2 c1 f1 f2 f3

let print_fermion_g4_current_rev coeff f c wf1 wf2 wf3 fusion =
  let c = format_coupling coeff c and
      f1 = (List.nth (wf_of_f wf1 wf2 wf3 fusion) 0) and
      f2 = (List.nth (wf_of_f wf1 wf2 wf3 fusion) 1) and
      f3 = (List.nth (wf_of_f wf1 wf2 wf3 fusion) 2) in
  match fusion with
```

```
| (F123 | F213 | F132 | F231 | F312 | F321) →
    printf "f_%sgr(%s,%s,%s,%s)" f c f1 f2 f3
| (F423 | F243 | F432 | F234 | F342 | F324) →
    printf "gr_%sf(-%s,%s,%s,%s)" f c f1 f2 f3
| (F134 | F143 | F314 | F124 | F142 | F214) →
    printf "%s_grf(-%s,%s,%s,%s)" f c f1 f2 f3
| (F413 | F431 | F341 | F241 | F412 | F421) →
    printf "%s_fgr(%s,%s,%s,%s)" f c f1 f2 f3
```

Here we have to distinguish which of the two bosons is produced in the fusion of three particles which include both fermions.

```
let print_fermion_g4_vector_current coeff f c wf1 wf2 wf3 fusion =
  let c = format_coupling coeff c and
      d1 = d_p (1,f) and
      d2 = d_p (2,f) and
      f1 = (List.nth (wf_of_f wf1 wf2 wf3 fusion) 0) and
      f2 = (List.nth (wf_of_f wf1 wf2 wf3 fusion) 1) and
      f3 = (List.nth (wf_of_f wf1 wf2 wf3 fusion) 2) in
  match fusion with
  | (F123 | F213 | F132 | F231 | F312 | F321) →
      printf "f_%sgr(%s,%s,%s,%s)" f c f1 f2 f3
  | (F423 | F243 | F432 | F234 | F342 | F324) →
      printf "gr_%sf(%s,%s,%s,%s)" f c f1 f2 f3
  | (F134 | F143 | F314) → printf "%s%s_grf(%s,%s,%s,%s)" f d1 c f1 f2 f3
  | (F124 | F142 | F214) → printf "%s%s_grf(%s,%s,%s,%s)" f d2 c f1 f2 f3
  | (F413 | F431 | F341) → printf "%s%s_fgr(%s,%s,%s,%s)" f d1 c f1 f2 f3
  | (F241 | F412 | F421) → printf "%s%s_fgr(%s,%s,%s,%s)" f d2 c f1 f2 f3

let print_fermion_2_g4_vector_current coeff f c wf1 wf2 wf3 fusion =
  let d1 = d_p (1,f) and
      d2 = d_p (2,f) and
      f1 = (List.nth (wf_of_f wf1 wf2 wf3 fusion) 0) and
      f2 = (List.nth (wf_of_f wf1 wf2 wf3 fusion) 1) and
      f3 = (List.nth (wf_of_f wf1 wf2 wf3 fusion) 2) in
  let c = format_coupling_2 coeff c in
  let c1 = fastener c 1 and
      c2 = fastener c 2 in
  match fusion with
  | (F123 | F213 | F132 | F231 | F312 | F321) →
      printf "f_%sgr(%s,%s,%s,%s,%s)" f c1 c2 f1 f2 f3
  | (F423 | F243 | F432 | F234 | F342 | F324) →
      printf "gr_%sf(%s,%s,%s,%s,%s)" f c1 c2 f1 f2 f3
  | (F134 | F143 | F314) → printf "%s%s_grf(%s,%s,%s,%s,%s)" f d1 c1 c2 f1 f2 f3
  | (F124 | F142 | F214) → printf "%s%s_grf(%s,%s,%s,%s,%s)" f d2 c1 c2 f1 f2 f3
  | (F413 | F431 | F341) → printf "%s%s_fgr(%s,%s,%s,%s,%s)" f d1 c1 c2 f1 f2 f3
  | (F241 | F412 | F421) → printf "%s%s_fgr(%s,%s,%s,%s,%s)" f d2 c1 c2 f1 f2 f3

let print_fermion_g4_vector_current_rev coeff f c wf1 wf2 wf3 fusion =
  let c = format_coupling coeff c and
      d1 = d_p (1,f) and
      d2 = d_p (2,f) and
```

```
      f1 = (List.nth (wf_of_f wf1 wf2 wf3 fusion) 0) and
      f2 = (List.nth (wf_of_f wf1 wf2 wf3 fusion) 1) and
      f3 = (List.nth (wf_of_f wf1 wf2 wf3 fusion) 2) in
  match fusion with
  | (F123 | F213 | F132 | F231 | F312 | F321) →
      printf "gr_%sf(%s,%s,%s,%s)" f c f1 f2 f3
  | (F423 | F243 | F432 | F234 | F342 | F324) →
      printf "f_%sgr(%s,%s,%s,%s)" f c f1 f2 f3
  | (F134 | F143 | F314) → printf "%s%s_fgr(%s,%s,%s,%s)" f d1 c f1 f2 f3
  | (F124 | F142 | F214) → printf "%s%s_fgr(%s,%s,%s,%s)" f d2 c f1 f2 f3
  | (F413 | F431 | F341) → printf "%s%s_grf(%s,%s,%s,%s)" f d1 c f1 f2 f3
  | (F241 | F412 | F421) → printf "%s%s_grf(%s,%s,%s,%s)" f d2 c f1 f2 f3

let print_fermion_2_g4_current_rev coeff f c wf1 wf2 wf3 fusion =
  let c = format_coupling_2 coeff c in
  let c1 = fastener c 1 and
      c2 = fastener c 2 and
      d1 = d_p (1, f) and
      d2 = d_p (2, f) in
  let f1 = (List.nth (wf_of_f wf1 wf2 wf3 fusion) 0) and
      f2 = (List.nth (wf_of_f wf1 wf2 wf3 fusion) 1) and
      f3 = (List.nth (wf_of_f wf1 wf2 wf3 fusion) 2) in
  match fusion with
  | (F123 | F213 | F132 | F231 | F312 | F321) →
      printf "gr_%sf(%s,%s,%s,%s,%s)" f c1 c2 f1 f2 f3
  | (F423 | F243 | F432 | F234 | F342 | F324) →
      printf "f_%sgr(-(%s),-(%s),%s,%s,%s)" f c1 c2 f1 f2 f3
  | (F134 | F143 | F314) →
      printf "%s%s_fgr(-(%s),-(%s),%s,%s,%s)" f d1 c1 c2 f1 f2 f3
  | (F124 | F142 | F214) →
      printf "%s%s_fgr(-(%s),-(%s),%s,%s,%s)" f d2 c1 c2 f1 f2 f3
  | (F413 | F431 | F341) →
      printf "%s%s_grf(%s,%s,%s,%s,%s)" f d1 c1 c2 f1 f2 f3
  | (F241 | F412 | F421) →
      printf "%s%s_grf(%s,%s,%s,%s,%s)" f d2 c1 c2 f1 f2 f3

let print_fermion_2_g4_vector_current_rev coeff f c wf1 wf2 wf3 fusion =
  (* Here we put in the extra minus sign from the coeff. *)
  let c = format_coupling coeff c in
  let c1 = fastener c 1 and
      c2 = fastener c 2 in
  let d1 = d_p (1, f) and
      d2 = d_p (2, f) and
      f1 = (List.nth (wf_of_f wf1 wf2 wf3 fusion) 0) and
      f2 = (List.nth (wf_of_f wf1 wf2 wf3 fusion) 1) and
      f3 = (List.nth (wf_of_f wf1 wf2 wf3 fusion) 2) in
  match fusion with
  | (F123 | F213 | F132 | F231 | F312 | F321) →
      printf "gr_%sf(%s,%s,%s,%s,%s)" f c1 c2 f1 f2 f3
  | (F423 | F243 | F432 | F234 | F342 | F324) →
      printf "f_%sgr(%s,%s,%s,%s,%s)" f c1 c2 f1 f2 f3
```

```
      | (F134 | F143 | F314) → printf "%s%s_fgr(%s,%s,%s,%s,%s)" f d1 c1 c2 f1 f2 f3
      | (F124 | F142 | F214) → printf "%s%s_fgr(%s,%s,%s,%s,%s)" f d2 c1 c2 f1 f2 f3
      | (F413 | F431 | F341) → printf "%s%s_grf(%s,%s,%s,%s,%s)" f d1 c1 c2 f1 f2 f3
      | (F241 | F412 | F421) → printf "%s%s_grf(%s,%s,%s,%s,%s)" f d2 c1 c2 f1 f2 f3

  let print_current_g4 = function
    | coeff, Gravbar, S2, _ → print_fermion_g4_current coeff "s2"
    | coeff, Gravbar, SV, _ → print_fermion_g4_vector_current coeff "sv"
    | coeff, Gravbar, SLV, _ → print_fermion_g4_vector_current coeff "slv"
    | coeff, Gravbar, SRV, _ → print_fermion_g4_vector_current coeff "srv"
    | coeff, Gravbar, SLRV, _ → print_fermion_2_g4_vector_current coeff "slrv"
    | coeff, Gravbar, PV, _ → print_fermion_g4_vector_current coeff "pv"
    | coeff, Gravbar, V2, _ → print_fermion_g4_current coeff "v2"
    | coeff, Gravbar, V2LR, _ → print_fermion_2_g4_current coeff "v2lr"
    | _, Gravbar, _, _ → invalid_arg "print_current_g4:␣not␣implemented"
    | coeff, _, S2, Grav → print_fermion_g4_current_rev coeff "s2"
    | coeff, _, SV, Grav → print_fermion_g4_vector_current_rev (−coeff) "sv"
    | coeff, _, SLV, Grav → print_fermion_g4_vector_current_rev (−coeff) "slv"
    | coeff, _, SRV, Grav → print_fermion_g4_vector_current_rev (−coeff) "srv"
    | coeff, _, SLRV, Grav → print_fermion_2_g4_vector_current_rev coeff "slrv"
    | coeff, _, PV, Grav → print_fermion_g4_vector_current_rev coeff "pv"
    | coeff, _, V2, Grav → print_fermion_g4_vector_current_rev coeff "v2"
    | coeff, _, V2LR, Grav → print_fermion_2_g4_current_rev coeff "v2lr"
    | _, _, _, Grav → invalid_arg "print_current_g4:␣not␣implemented"
    | coeff, _, S2, _ → print_fermion_s2_current coeff "s"
    | coeff, _, P2, _ → print_fermion_s2_current coeff "p"
    | coeff, _, S2P, _ → print_fermion_s2p_current coeff "sp"
    | coeff, _, S2L, _ → print_fermion_s2_current coeff "sl"
    | coeff, _, S2R, _ → print_fermion_s2_current coeff "sr"
    | coeff, _, S2LR, _ → print_fermion_s2lr_current coeff "slr"
    | coeff, _, V2, _ → print_fermion_g4_brs_vector_current coeff "v2"
    | coeff, _, SV, _ → print_fermion_g4_brs_vector_current coeff "sv"
    | coeff, _, PV, _ → print_fermion_g4_brs_vector_current coeff "pv"
    | coeff, _, SLV, _ → print_fermion_g4_brs_vector_current coeff "svl"
    | coeff, _, SRV, _ → print_fermion_g4_brs_vector_current coeff "svr"
    | coeff, _, SLRV, _ → print_fermion_g4_svlr_current coeff "svlr"
    | _, _, V2LR, _ → invalid_arg "Targets.print_current:␣not␣available"

  let reverse_braket _ = false

  let use_module = "omega95_bispinors"
  let require_library =
    ["omega_bispinors_2010_01_A"; "omega_bispinor_cpls_2010_01_A"]
 end

module Fortran_Majorana = Make_Fortran(Fortran_Majorana_Fermions)
```

FORTRAN 77

```
module Fortran77 = Dummy
```

15.2.3 C

```
module C = Dummy
```

C++

```
module Cpp = Dummy
```

Java

```
module Java = Dummy
```

15.2.4 O'Caml

```
module Ocaml = Dummy
```

15.2.5 LaTeX

```
module LaTeX = Dummy
```

15.3 Interface of Targets_Kmatrix

```
module Fortran : sig val print : bool → unit end
```

15.4 Implementation of Targets_Kmatrix

```
let rcs_file = RCS.parse "Targets_Kmatrix" ["K-Matrix Support routines"]
    { RCS.revision = "$Revision: 6465 $";
      RCS.date = "$Date: 2015-01-10 16:22:31 +0100 (Sat, 10 Jan 2015) $";
      RCS.author = "$Author: jr_reuter $";
      RCS.source
        = "$URL: svn+ssh://jr_reuter@login.hepforge.org/hepforge/svn/whizard/trunk/omega/sr

module Fortran =
  struct

    open Format

    let nl = print_newline
```

Special functions for the K matrix approach. This might be generalized to other functions that have to have access to the parameters and coupling constants. At the moment, this is hardcoded.

```
    let print pure_functions =
      let pure =
        if pure_functions then
```

```
      "pure␣"
    else
      "" in
  printf "␣␣!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!"; nl ();
  printf "␣␣!!!␣Special␣K␣matrix␣functions"; nl ();
  printf "␣␣!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!"; nl ();
  nl();
  printf "␣␣%sfunction␣width_res␣(z,res,w_wkm,m,g)␣result␣(w)" pure; nl ();
  printf "␣␣␣␣␣␣real(kind=default),␣intent(in)␣::␣z,␣w_wkm,␣m,␣g"; nl ();
  printf "␣␣␣␣␣␣integer,␣intent(in)␣::␣res"; nl ();
  printf "␣␣␣␣␣␣real(kind=default)␣::␣w"; nl ();
  printf "␣␣␣␣␣␣if␣(z.eq.0␣.AND.␣w_wkm.eq.0␣)␣then"; nl ();
  printf "␣␣␣␣␣␣␣␣w␣=␣0"; nl ();
  printf "␣␣␣␣␣␣else"; nl ();
  printf "␣␣␣␣␣␣␣␣if␣(w_wkm.eq.0)␣then"; nl ();
  printf "␣␣␣␣␣␣␣␣␣␣select␣case␣(res)"; nl ();
  printf "␣␣␣␣␣␣␣␣␣␣␣␣case␣(1)␣!!!␣Scalar␣isosinglet"; nl ();
  printf "␣␣␣␣␣␣␣␣␣␣␣␣␣␣w␣=␣3.*g**2/32./Pi␣*␣m**3/vev**2"; nl ();
  printf "␣␣␣␣␣␣␣␣␣␣␣␣case␣(2)␣!!!␣Scalar␣isoquintet"; nl ();
  printf "␣␣␣␣␣␣␣␣␣␣␣␣␣␣w␣=␣g**2/64./Pi␣*␣m**3/vev**2"; nl ();
  printf "␣␣␣␣␣␣␣␣␣␣␣␣case␣(3)␣!!!␣Vector␣isotriplet"; nl ();
  printf "␣␣␣␣␣␣␣␣␣␣␣␣␣␣w␣=␣g**2/48./Pi␣*␣m"; nl ();
  printf "␣␣␣␣␣␣␣␣␣␣␣␣case␣(4)␣!!!␣Tensor␣isosinglet"; nl ();
  printf "␣␣␣␣␣␣␣␣␣␣␣␣␣␣w␣=␣g**2/320./Pi␣*␣m**3/vev**2"; nl ();
  printf "␣␣␣␣␣␣␣␣␣␣␣␣case␣(5)␣!!!␣Tensor␣isoquintet"; nl ();
  printf "␣␣␣␣␣␣␣␣␣␣␣␣␣␣w␣=␣g**2/1920./Pi␣*␣m**3/vev**2"; nl ();
  printf "␣␣␣␣␣␣␣␣␣␣␣␣case␣default"; nl ();
  printf "␣␣␣␣␣␣␣␣␣␣␣␣␣␣w␣=␣0"; nl ();
  printf "␣␣␣␣␣␣␣␣␣␣end␣select"; nl ();
  printf "␣␣␣␣␣␣␣␣else"; nl ();
  printf "␣␣␣␣␣␣␣␣␣␣w␣=␣w_wkm"; nl ();
  printf "␣␣␣␣␣␣␣␣end␣if"; nl ();
  printf "␣␣␣␣␣␣end␣if"; nl ();
  printf "␣␣end␣function␣width_res"; nl ();
  nl ();
  printf "␣␣%sfunction␣s0stu␣(s,␣m)␣result␣(s0)" pure; nl ();
  printf "␣␣␣␣␣␣real(kind=default),␣intent(in)␣::␣s,␣m"; nl ();
  printf "␣␣␣␣␣␣real(kind=default)␣::␣s0"; nl ();
  printf "␣␣␣␣␣␣if␣(m.ge.1.0e08)␣then"; nl ();
  printf "␣␣␣␣␣␣␣␣s0␣=␣0"; nl ();
  printf "␣␣␣␣␣␣else"; nl ();
  printf "␣␣␣␣␣␣␣␣s0␣=␣m**2␣-␣s/2␣+␣m**4/s␣*␣log(m**2/(s+m**2))"; nl ();
  printf "␣␣␣␣␣␣end␣if"; nl ();
  printf "␣␣end␣function␣s0stu"; nl();
  nl ();
  printf "␣␣%sfunction␣s1stu␣(s,␣m)␣result␣(s1)" pure; nl ();
  printf "␣␣␣␣␣␣real(kind=default),␣intent(in)␣::␣s,␣m"; nl ();
  printf "␣␣␣␣␣␣real(kind=default)␣::␣s1"; nl ();
  printf "␣␣␣␣␣␣if␣(m.ge.1.0e08)␣then"; nl ();
  printf "␣␣␣␣␣␣␣␣s1␣=␣0"; nl ();
```

```
printf "␣␣␣␣␣␣else"; nl ();
printf "␣␣␣␣␣␣␣␣s1␣=␣2*m**4/s␣+␣s/6␣+␣m**4/s**2*(2*m**2+s)␣&"; nl();
printf "␣␣␣␣␣␣␣␣␣␣␣␣*␣log(m**2/(s+m**2))"; nl ();
printf "␣␣␣␣␣␣end␣if"; nl ();
printf "␣␣end␣function␣s1stu"; nl();
nl ();
printf "␣␣%sfunction␣s2stu␣(s,␣m)␣result␣(s2)" pure; nl ();
printf "␣␣␣␣␣␣real(kind=default),␣intent(in)␣::␣s,␣m"; nl ();
printf "␣␣␣␣␣␣real(kind=default)␣::␣s2"; nl ();
printf "␣␣␣␣␣␣if␣(m.ge.1.0e08)␣then"; nl ();
printf "␣␣␣␣␣␣␣␣s2␣=␣0"; nl ();
printf "␣␣␣␣␣␣else"; nl ();
printf "␣␣␣␣␣␣␣␣s2␣=␣m**4/s**2␣*␣(6*m**2␣+␣3*s)␣+␣&"; nl();
printf "␣␣␣␣␣␣␣␣␣␣␣␣m**4/s**3␣*␣(6*m**4␣+␣6*m**2*s␣+␣s**2)␣&"; nl();
printf "␣␣␣␣␣␣␣␣␣␣␣␣*␣log(m**2/(s+m**2))"; nl ();
printf "␣␣␣␣␣␣end␣if"; nl ();
printf "␣␣end␣function␣s2stu"; nl();
nl ();
printf "␣!!␣%sfunction␣s3stu␣(s,␣m)␣result␣(s3)" pure; nl ();
printf "␣!!␣␣␣␣␣real(kind=default),␣intent(in)␣::␣s,␣m"; nl ();
printf "␣!!␣␣␣␣␣real(kind=default)␣::␣s3"; nl ();
printf "␣!!␣␣␣␣␣if␣(m.ge.1.0e08)␣then"; nl ();
printf "␣!!␣␣␣␣␣␣␣s3␣=␣0"; nl ();
printf "␣!!␣␣␣␣␣else"; nl ();
printf "␣!!␣␣␣␣␣␣␣s3␣=␣m**4/s**3␣*␣(60*m**4␣+␣60*m**2*s+11*s**2)␣+␣&"; nl();
printf "␣!!␣␣␣␣␣␣␣␣␣␣␣m**4/s**4␣*(2*m**2+s)␣(10*m**4␣+␣10*m**2*s␣+␣s**2)␣&"; nl();
printf "␣!!␣␣␣␣␣␣␣␣␣␣␣*␣log(m**2/(s+m**2))"; nl ();
printf "␣!!␣␣␣␣␣end␣if"; nl ();
printf "␣!!␣␣end␣function␣s3stu"; nl();
nl ();
printf "␣␣%sfunction␣p0stu␣(s,␣m)␣result␣(p0)" pure; nl ();
printf "␣␣␣␣␣␣real(kind=default),␣intent(in)␣::␣s,␣m"; nl ();
printf "␣␣␣␣␣␣real(kind=default)␣::␣p0"; nl ();
printf "␣␣␣␣␣␣if␣(m.ge.1.0e08)␣then"; nl ();
printf "␣␣␣␣␣␣␣␣p0␣=␣0"; nl ();
printf "␣␣␣␣␣␣else"; nl ();
printf "␣␣␣␣␣␣␣␣p0␣=␣1␣+␣(2*s+m**2)*log(m**2/(s+m**2))/s"; nl ();
printf "␣␣␣␣␣␣end␣if"; nl ();
printf "␣␣end␣function␣p0stu"; nl();
nl ();
printf "␣␣%sfunction␣p1stu␣(s,␣m)␣result␣(p1)" pure; nl ();
printf "␣␣␣␣␣␣real(kind=default),␣intent(in)␣::␣s,␣m"; nl ();
printf "␣␣␣␣␣␣real(kind=default)␣::␣p1"; nl ();
printf "␣␣␣␣␣␣if␣(m.ge.1.0e08)␣then"; nl ();
printf "␣␣␣␣␣␣␣␣p1␣=␣0"; nl ();
printf "␣␣␣␣␣␣else"; nl ();
printf "␣␣␣␣␣␣␣␣p1␣=␣(m**2␣+␣2*s)/s**2␣*␣(2*s+(2*m**2+s)␣&"; nl();
printf "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣*␣log(m**2/(s+m**2)))"; nl ();
printf "␣␣␣␣␣␣end␣if"; nl ();
printf "␣␣end␣function␣p1stu"; nl();
```

```
nl ();
printf "␣␣%sfunction␣d0stu␣(s,␣m)␣result␣(d0)" pure; nl ();
printf "␣␣␣␣␣␣real(kind=default),␣intent(in)␣::␣s,␣m"; nl ();
printf "␣␣␣␣␣␣real(kind=default)␣::␣d0"; nl ();
printf "␣␣␣␣␣␣if␣(m.ge.1.0e08)␣then"; nl ();
printf "␣␣␣␣␣␣␣␣d0␣=␣0"; nl ();
printf "␣␣␣␣␣␣else"; nl ();
printf "␣␣␣␣␣␣␣␣d0␣=␣(2*m**2+11*s)/2␣+␣(m**4+6*m**2*s+6*s**2)␣&"; nl();
printf "␣␣␣␣␣␣␣␣␣␣␣␣␣␣/s␣*␣log(m**2/(s+m**2))"; nl ();
printf "␣␣␣␣␣␣end␣if"; nl ();
printf "␣␣end␣function␣d0stu"; nl();
nl ();
printf "␣␣%sfunction␣d1stu␣(s,␣m)␣result␣(d1)" pure; nl ();
printf "␣␣␣␣␣␣real(kind=default),␣intent(in)␣::␣s,␣m"; nl ();
printf "␣␣␣␣␣␣real(kind=default)␣::␣d1"; nl ();
printf "␣␣␣␣␣␣if␣(m.ge.1.0e08)␣then"; nl ();
printf "␣␣␣␣␣␣␣␣d1␣=␣0"; nl ();
printf "␣␣␣␣␣␣else"; nl ();
printf "␣␣␣␣␣␣␣␣d1␣=␣(s*(12*m**4␣+␣72*m**2*s␣+␣73*s**2)␣&"; nl();
printf "␣␣␣␣␣␣␣␣␣␣␣␣+␣6*(2*m**2␣+␣s)*(m**4␣+␣6*m**2*s␣+␣6*s**2)␣&"; nl();
printf "␣␣␣␣␣␣␣␣␣␣␣␣*␣log(m**2/(s+m**2)))/6/s**2"; nl ();
printf "␣␣␣␣␣␣end␣if"; nl ();
printf "␣␣end␣function␣d1stu"; nl();
nl ();
printf "␣␣%sfunction␣da00␣(cc,␣s,␣m)␣result␣(amp_00)" pure; nl ();
printf "␣␣␣␣␣␣real(kind=default),␣intent(in)␣::␣s"; nl ();
printf "␣␣␣␣␣␣real(kind=default),␣dimension(1:12),␣intent(in)␣::␣cc"; nl ();
printf "␣␣␣␣␣␣real(kind=default),␣dimension(1:5),␣intent(in)␣::␣m"; nl ();
printf "␣␣␣␣␣␣complex(kind=default)␣::␣a00_0,␣a00_1,␣a00_a,␣a00_f"; nl ();
printf "␣␣␣␣␣␣complex(kind=default),␣dimension(1:7)␣::␣a00"; nl ();
printf "␣␣␣␣␣␣complex(kind=default)␣::␣ii,␣jj,␣amp_00"; nl ();
printf "␣␣␣␣␣␣real(kind=default)␣::␣kappal,␣kappam,␣kappat"; nl ();
printf "␣␣␣␣␣␣ii␣=␣cmplx(0.0,1.0/32.0/Pi,default)"; nl ();
printf "␣␣␣␣␣␣jj␣=␣s**2/vev**4*ii"; nl ();
printf "␣␣␣␣␣␣kappal␣=␣cc(12)*((mass(23)**2+mass(24)**2)/m(4)**2-2*mass(23)**2*mass(24
printf "␣␣␣␣␣␣kappam␣=␣cc(12)*((mass(23)**4+mass(24)**4)/m(4)**2/(mass(23)**2+mass(24)
printf "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣-␣2*mass(23)**2*mass(24)**2/m(4)**4)"; nl ();
printf "␣␣␣␣␣␣kappat␣=␣cc(12)*mass(23)**2*mass(24)**2/m(4)**4"; nl ();
printf "␣␣␣␣␣␣!!!␣Longitudinal"; nl ();
printf "␣␣␣␣␣␣!!!␣Scalar␣isosinglet"; nl ();
printf "␣␣␣␣␣␣a00(1)␣=␣-2.0␣*␣cc(1)**2/vev**2␣*␣s0stu(s,m(1))␣"; nl ();
printf "␣␣␣␣␣␣if␣(cc(1)␣/=␣0)␣then"; nl ();
printf "␣␣␣␣␣␣␣␣a00(1)␣=␣a00(1)␣-␣3.0*cc(1)**2/vev**2␣*␣&"; nl ();
printf "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣s**2/cmplx(s-m(1)**2,m(1)*wkm(1),default)␣"; nl ();
printf "␣␣␣␣␣␣end␣if"; nl ();
printf "␣␣␣␣␣␣!!!␣Scalar␣isoquintet"; nl ();
printf "␣␣␣␣␣␣a00(2)␣=␣-5.0*cc(2)**2/vev**2␣*␣s0stu(s,m(2))␣/␣3.0"; nl ();
printf "␣␣␣␣␣␣!!!␣Vector␣isotriplet"; nl ();
printf "␣␣␣␣␣␣a00(3)␣=␣-cc(3)**2*(4.0*p0stu(s,m(3))␣+␣6.0*s/m(3)**2)"; nl ();
printf "␣␣␣␣␣␣!!!␣Tensor␣isosinglet"; nl ();
```

```
printf "␣␣␣␣␣␣a00(4)␣=␣-cc(4)**2/vev**2/3␣*␣(d0stu(s,m(4))␣&"; nl ();
printf "␣␣␣␣␣␣␣␣␣␣␣␣␣␣-␣2*kappal*s0stu(s,m(4)))"; nl ();
printf "␣␣␣␣␣␣if␣(␣(cc(4)␣/=␣0).and.(kappal␣/=␣0))␣then"; nl ();
printf "␣␣␣␣␣␣␣␣a00(4)␣=␣a00(4)␣-␣cc(4)**2/vev**2*kappal␣*␣&"; nl ();
printf "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣s**2/cmplx(s-m(4)**2,m(4)␣*␣wkm(4),default)"; nl ();
printf "␣␣␣␣␣␣end␣if"; nl ();
printf "␣␣␣␣␣␣!!!␣Tensor␣isoquintet"; nl ();
printf "␣␣␣␣␣␣a00(5)␣=␣-5.0*cc(5)**2/vev**2*(d0stu(s,m(5))␣&"; nl ();
printf "␣␣␣␣␣␣␣␣␣␣␣␣␣␣/3.0)/6.0"; nl ();
printf "␣␣␣␣␣␣!!!␣Transversal"; nl ();
printf "␣␣␣␣␣␣!!!␣Tensor␣isosinglet"; nl ();
printf "␣␣␣␣␣␣a00(6)␣=␣-␣cc(9)**2/Pi/vev**6*mass(23)**2*mass(24)**2/4*␣s**2␣&"; nl ();
printf "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣*␣((2-2*s/m(4)**2+s**2/m(4)**4)+kappat/2␣)"; nl ();
printf "␣␣␣␣␣␣if␣(a00(6)␣/=␣0)␣then"; nl ();
printf "␣␣␣␣␣␣␣␣a00(6)␣=␣a00(6)/cmplx(s-m(4)**2,␣-␣w_res/32/Pi␣*␣real(a00(6),default),
printf "␣␣␣␣␣␣end␣if"; nl ();
printf "␣␣␣␣␣␣a00(6)␣=␣a00(6)␣-␣cc(9)**2/Pi/vev**6*mass(23)**2*mass(24)**2/12␣*␣(s0stu
printf "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣*␣(3*(1+2*s/m(4)**2+2*s**2/m(4)**4)+kappat␣))"; nl ();
printf "␣␣␣␣␣␣!!!␣Mixed"; nl ();
printf "␣␣␣␣␣␣!!!␣Tensor␣isosinglet"; nl ();
printf "␣␣␣␣␣␣a00(7)␣=␣-␣cc(11)*cc(9)*cc(4)/Pi/vev**4*(mass(23)**2+mass(24)**2)/4␣*␣s*
printf "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣*␣␣((1-4*s/m(4)**2+2*s**2/m(4)**4)+kappam␣)"; nl ();
printf "␣␣␣␣␣␣if␣(a00(7)␣/=␣0)␣then"; nl ();
printf "␣␣␣␣␣␣␣␣a00(7)␣=␣a00(7)/cmplx(s-m(4)**2,␣-␣w_res/32/Pi␣*␣real(a00(7),default),
printf "␣␣␣␣␣␣end␣if"; nl ();
printf "␣␣␣␣␣␣a00(7)␣=␣a00(7)␣-␣cc(11)*cc(9)*cc(4)/Pi/vev**4*(mass(23)**2+mass(24)**2)
printf "␣␣␣␣␣␣␣␣␣␣␣␣␣␣*␣(12*s/m(4)**2+12*s**2/m(4)**4+2*kappam␣))"; nl ();
printf "␣␣␣␣␣␣!!!␣Fudge-Higgs"; nl ();
printf "␣␣␣␣␣␣a00_f␣=␣2.*fudge_higgs*s/vev**2"; nl ();
printf "␣␣␣␣␣␣a00_f␣=␣a00_f␣!!!␣-␣0*5.*(1-ghvva)**2/vev**2*mass(25)**2"; nl ();
printf "␣␣␣␣␣␣!!!␣Low␣energy␣theory␣alphas"; nl ();
printf "␣␣␣␣␣␣a00_0␣=␣8.*(7.*a4␣+␣11.*a5)/3.*s**2/vev**4"; nl ();
printf "␣␣␣␣␣␣a00_1␣=␣(25.*log(lam_reg**2/s)/9␣+␣11./54.0_default)*s**2/vev**4"; nl ();
printf "␣␣␣␣␣␣a00_a␣=␣␣a00_0␣!!!␣+␣a00_1/16./Pi**2"; nl ();
printf "␣␣␣␣␣␣!!!␣Unitarize"; nl ();
printf "␣␣␣␣␣␣if␣(fudge_km␣/=␣0)␣then"; nl ();
printf "␣␣␣␣␣␣␣␣amp_00␣=␣sum(a00)+a00_f+a00_a"; nl();
printf "␣␣␣␣␣␣␣␣if␣(amp_00␣/=␣0)␣then"; nl ();
printf "␣␣␣␣␣␣␣␣␣␣amp_00␣=␣-␣a00_a␣-␣a00_f␣-␣part_r␣*␣(sum(a00)␣-␣a00(3))␣+␣1/(real(1/
printf "␣␣␣␣␣␣␣␣end␣if"; nl ();
printf "␣␣␣␣␣␣else"; nl ();
printf "␣␣␣␣␣␣␣␣amp_00␣=␣(1-part_r)␣*␣sum(a00)␣+␣part_r␣*␣a00(3)"; nl ();
printf "␣␣␣␣␣␣end␣if"; nl ();
printf "␣␣␣␣␣␣amp_00␣=␣vev**4/s**2␣*␣amp_00"; nl ();
printf "␣␣end␣function␣da00"; nl();
nl ();
printf "␣␣%sfunction␣da02␣(cc,␣s,␣m)␣result␣(amp_02)" pure; nl ();
printf "␣␣␣␣␣␣real(kind=default),␣intent(in)␣::␣s"; nl ();
printf "␣␣␣␣␣␣real(kind=default),␣dimension(1:12),␣intent(in)␣::␣cc"; nl ();
printf "␣␣␣␣␣␣real(kind=default),␣dimension(1:5),␣intent(in)␣::␣m"; nl ();
```

```
printf "␣␣␣␣␣␣complex(kind=default)␣::␣a02_0,␣a02_1,␣a02_a"; nl ();
printf "␣␣␣␣␣␣complex(kind=default),␣dimension(1:7)␣::␣a02"; nl ();
printf "␣␣␣␣␣␣complex(kind=default)␣::␣ii,␣jj,␣amp_02"; nl ();
printf "␣␣␣␣␣␣real(kind=default)␣::␣kappal,␣kappam,␣kappat"; nl ();
printf "␣␣␣␣␣␣ii␣=␣cmplx(0.0,1.0/32.0/Pi,default)"; nl ();
printf "␣␣␣␣␣␣jj␣=␣s**2/vev**4*ii"; nl ();
printf "␣␣␣␣␣␣kappal␣=␣cc(12)*((mass(23)**2+mass(24)**2)/m(4)**2-2*mass(23)**2*mass(24
printf "␣␣␣␣␣␣kappam␣=␣cc(12)*((mass(23)**4+mass(24)**4)/m(4)**2/(mass(23)**2+mass(24)
printf "␣␣␣␣␣␣␣␣␣␣␣-␣2*mass(23)**2*mass(24)**2/m(4)**4)"; nl ();
printf "␣␣␣␣␣␣kappat␣=␣cc(12)*mass(23)**2*mass(24)**2/m(4)**4"; nl ();
printf "␣␣␣␣␣␣!!!␣Longitudinal"; nl ();
printf "␣␣␣␣␣␣!!!␣Scalar␣␣isosinglet"; nl ();
printf "␣␣␣␣␣␣a02(1)␣=␣-2.0*cc(1)**2/vev**2␣*␣s2stu(s,m(1))"; nl ();
printf "␣␣␣␣␣␣!!!␣Scalar␣isoquintet"; nl ();
printf "␣␣␣␣␣␣a02(2)␣=␣-5.0*cc(2)**2/vev**2␣*␣s2stu(s,m(2))␣/␣3.0"; nl ();
printf "␣␣␣␣␣␣!!!␣Vector␣isotriplet"; nl ();
printf "␣␣␣␣␣␣a02(3)␣=␣-4.0*cc(3)**2*(2*s+m(3)**2)*s2stu(s,m(3))/m(3)**4"; nl ();
printf "␣␣␣␣␣␣!!!␣Tensor␣isosinglet"; nl ();
printf "␣␣␣␣␣␣a02(4)␣=␣-␣cc(4)**2/vev**2/3␣*␣␣&"; nl ();
printf "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣((1.+6.*s/m(4)**2+6.*s**2/m(4)**4)-2*kappal)␣*␣s2stu(s,m(4))
printf "␣␣␣␣␣␣if␣(cc(4)␣/=␣0)␣then"; nl ();
printf "␣␣␣␣␣␣␣␣a02(4)␣=␣a02(4)␣-␣cc(4)**2/vev**2/10.␣&"; nl ();
printf "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣*␣s**2/cmplx(s-m(4)**2,m(4)*wkm(4),default)"; nl ();
printf "␣␣␣␣␣␣end␣if"; nl ();
printf "␣␣␣␣␣␣!!!␣Tensor␣isoquintet"; nl ();
printf "␣␣␣␣␣␣a02(5)␣=␣-cc(5)**2/vev**2*(5.0*(1.0+6.0*␣&"; nl ();
printf "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣s/m(5)**2+6.0*s**2/m(5)**4)*s2stu(s,m(5))/3.0␣&"; nl ();
printf "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣)/6.0"; nl ();
printf "␣␣␣␣␣␣!!!␣Transversal"; nl ();
printf "␣␣␣␣␣␣!!!␣Tensor␣isosinglet"; nl ();
printf "␣␣␣␣␣␣a02(6)␣=␣-␣cc(9)**2/Pi/vev**6*mass(23)**2*mass(24)**2/40*␣s**2"; nl ();
printf "␣␣␣␣␣␣if␣(a02(6)␣/=␣0)␣then"; nl ();
printf "␣␣␣␣␣␣␣␣a02(6)␣=␣a02(6)/cmplx(s-m(4)**2,␣-␣w_res/32/Pi␣*␣real(a02(6),default),
printf "␣␣␣␣␣␣end␣if"; nl ();
printf "␣␣␣␣␣␣a02(6)␣=␣a02(6)␣-␣cc(9)**2/Pi/vev**6*mass(23)**2*mass(24)**2/12␣*␣(s2stu
printf "␣␣␣␣␣␣␣␣␣␣␣␣␣␣*␣(3*(1+2*s/m(4)**2+2*s**2/m(4)**4)+kappat␣))"; nl ();
printf "␣␣␣␣␣␣!!!␣Mixed"; nl ();
printf "␣␣␣␣␣␣!!!␣Tensor␣isosinglet"; nl ();
printf "␣␣␣␣␣␣a02(7)␣=␣-␣cc(11)*cc(9)*cc(4)/Pi/vev**4*(mass(23)**2+mass(24)**2)/20␣&";
printf "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣*␣s**2"; nl ();
printf "␣␣␣␣␣␣if␣(a02(7)␣/=␣0)␣then"; nl ();
printf "␣␣␣␣␣␣␣␣a02(7)␣=␣a02(7)/cmplx(s-m(4)**2,␣-␣w_res/32/Pi␣*␣real(a02(7),default),
printf "␣␣␣␣␣␣end␣if"; nl ();
printf "␣␣␣␣␣␣a02(7)␣=␣a02(7)␣-␣cc(11)*cc(9)*cc(4)/Pi/vev**4*(mass(23)**2+mass(24)**2)
printf "␣␣␣␣␣␣␣␣␣␣␣␣␣␣*␣(12*s/m(4)**2+12*s**2/m(4)**4+2*kappam␣))"; nl ();
printf "␣␣␣␣␣␣!!!␣Low␣energy␣theory␣alphas"; nl ();
printf "␣␣␣␣␣␣a02_0␣=␣(8.*(2.*a4␣+␣a5)/15.)␣*␣␣s**2/vev**4"; nl ();
printf "␣␣␣␣␣␣a02_1␣=␣(log(lam_reg**2/s)/9.␣-␣7./135.0_default)␣*␣␣s**2/vev**4"; nl ();
printf "␣␣␣␣␣␣a02_a␣=␣a02_0␣!!!␣+␣a02_1/16/Pi**2"; nl ();
printf "␣␣␣␣␣␣!!!␣Unitarize"; nl ();
```

```
printf "␣␣␣␣␣␣if␣(fudge_km␣/=␣0)␣then"; nl ();
printf "␣␣␣␣␣␣␣␣amp_02␣=␣sum(a02)+a02_a"; nl();
printf "␣␣␣␣␣␣␣␣if␣(amp_02␣/=␣0)␣then"; nl ();
printf "␣␣␣␣␣␣␣␣␣␣amp_02␣=␣-␣a02_a␣-␣part_r␣*␣(sum(a02)␣-␣a02(3))␣+␣1/(real(1/amp_02,d
printf "␣␣␣␣␣␣␣␣end␣if"; nl ();
printf "␣␣␣␣␣␣else"; nl ();
printf "␣␣␣␣␣␣␣␣amp_02␣=␣(1-part_r)␣*␣sum(a02)␣+␣part_r␣*␣a02(3)"; nl ();
printf "␣␣␣␣␣␣end␣if"; nl ();
printf "␣␣␣␣␣amp_02␣=␣vev**4/s**2␣*␣amp_02"; nl ();
printf "␣␣end␣function␣da02"; nl();
nl ();
printf "␣␣%sfunction␣da11␣(cc,␣s,␣m)␣result␣(amp_11)" pure; nl ();
printf "␣␣␣␣␣␣real(kind=default),␣intent(in)␣::␣s"; nl ();
printf "␣␣␣␣␣␣real(kind=default),␣dimension(1:12),␣intent(in)␣::␣cc"; nl ();
printf "␣␣␣␣␣␣real(kind=default),␣dimension(1:5),␣intent(in)␣::␣m"; nl ();
printf "␣␣␣␣␣␣complex(kind=default)␣::␣a11_0,␣a11_1,␣a11_a,␣a11_f"; nl ();
printf "␣␣␣␣␣␣complex(kind=default),␣dimension(1:7)␣::␣a11"; nl ();
printf "␣␣␣␣␣␣complex(kind=default)␣::␣ii,␣jj,␣amp_11"; nl ();
printf "␣␣␣␣␣␣real(kind=default)␣::␣kappal,␣kappam,␣kappat"; nl ();
printf "␣␣␣␣␣␣ii␣=␣cmplx(0.0,1.0/32.0/Pi,default)"; nl ();
printf "␣␣␣␣␣␣jj␣=␣s**2/vev**4*ii"; nl ();
printf "␣␣␣␣␣␣kappal␣=␣cc(12)*((mass(23)**2+mass(24)**2)/m(4)**2-2*mass(23)**2*mass(24
printf "␣␣␣␣␣␣kappam␣=␣cc(12)*((mass(23)**4+mass(24)**4)/m(4)**2/(mass(23)**2+mass(24)
printf "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣-␣2*mass(23)**2*mass(24)**2/m(4)**4)"; nl ();
printf "␣␣␣␣␣␣kappat␣=␣cc(12)*mass(23)**2*mass(24)**2/m(4)**4"; nl ();
printf "␣␣␣␣␣␣!!!␣Longitudinal"; nl ();
printf "␣␣␣␣␣␣!!!␣Scalar␣isosinglet"; nl ();
printf "␣␣␣␣␣␣a11(1)␣=␣-␣2.0*cc(1)**2/vev**2␣*␣s1stu(s,m(1))"; nl ();
printf "␣␣␣␣␣␣!!!␣Scalar␣isoquintet"; nl ();
printf "␣␣␣␣␣␣a11(2)␣=␣5.0*cc(2)**2/vev**2␣*␣s1stu(s,m(2))␣/␣6.0"; nl ();
printf "␣␣␣␣␣␣!!!␣Vector␣isotriplet"; nl ();
printf "␣␣␣␣␣␣a11(3)␣=␣-␣cc(3)**2␣*␣&"; nl ();
printf "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣(s/m(3)**2␣+␣2.␣*␣p1stu(s,m(3)))"; nl ();
printf "␣␣␣␣␣␣if␣(cc(3)␣/=␣0)␣then"; nl ();
printf "␣␣␣␣␣␣␣␣a11(3)␣=␣a11(3)␣-2./3.␣*␣cc(3)**2␣*␣&"; nl ();
printf "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣s/cmplx(s-m(3)**2,m(3)*wkm(3),default)␣"; nl ();
printf "␣␣␣␣␣␣end␣if"; nl ();
printf "␣␣␣␣␣␣!!!␣Tensor␣isosinglet"; nl ();
printf "␣␣␣␣␣␣a11(4)␣=␣-␣cc(4)**2/vev**2*(d1stu(s,m(4)-2*kappal*s1stu(s,m(4)))␣&"; nl ()
printf "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣/3.0)"; nl ();
printf "␣␣␣␣␣␣!!!␣Tensor␣isoquintet"; nl ();
printf "␣␣␣␣␣␣a11(5)␣=␣␣5.0*cc(5)**2/vev**2*(d1stu(s,m(5))␣&"; nl ();
printf "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣)/36.0"; nl ();
printf "␣␣␣␣␣␣!!!␣Transversal"; nl ();
printf "␣␣␣␣␣␣!!!␣Tensor␣isosinglet"; nl ();
printf "␣␣␣␣␣␣a11(6)␣=␣-cc(9)**2/Pi/vev**6*mass(23)**2*mass(24)**2/12␣*␣(s1stu(s,m(4))
printf "␣␣␣␣␣␣␣␣␣␣␣␣␣(3*(1+2*s/m(4)**2+2*s**2/m(4)**4)+kappat␣)␣-␣(s/m(4)**2+s**2/m(4)
printf "␣␣␣␣␣␣!!!␣Mixed"; nl ();
printf "␣␣␣␣␣␣!!!␣Tensor␣isosinglet"; nl ();
printf "␣␣␣␣␣␣a11(7)␣=␣-cc(11)*cc(9)*cc(4)/Pi/vev**4*(mass(23)**2+mass(24)**2)/12␣*␣(s
```

```
printf "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣*␣(12*s/m(4)**2+12*s**2/m(4)**4+2*kappam␣)␣-␣2*(s/m(4)**2+s**2/
printf "␣␣␣␣␣␣!!!␣Fudge-Higgs"; nl ();
printf "␣␣␣␣␣␣a11_f␣=␣fudge_higgs*s/3./vev**2"; nl ();
printf "␣␣␣␣␣␣!!!␣Low␣energy␣theory␣alphas"; nl ();
printf "␣␣␣␣␣␣a11_0␣=␣4.*(a4␣-␣2*a5)/3.␣*␣s**2/vev**4␣"; nl ();
printf "␣␣␣␣␣␣a11_1␣=␣-␣1.0/54.0_default␣*␣s**2/vev**4"; nl ();
printf "␣␣␣␣␣␣a11_a␣=␣a11_0␣!!!␣+␣a11_1/16/Pi**2"; nl ();
printf "␣␣␣␣␣␣!!!␣Unitarize"; nl ();
printf "␣␣␣␣␣␣if␣(fudge_km␣/=␣0)␣then"; nl ();
printf "␣␣␣␣␣␣␣␣amp_11␣=␣sum(a11)+a11_f+a11_a"; nl();
printf "␣␣␣␣␣␣␣␣if␣(amp_11␣/=␣0)␣then"; nl ();
printf "␣␣␣␣␣␣␣␣␣␣amp_11␣=␣-␣a11_a␣-␣part_r␣*␣(sum(a11)␣-␣a11(3))␣+␣1/(real(1/amp_11,d
printf "␣␣␣␣␣␣␣␣end␣if"; nl ();
printf "␣␣␣␣␣␣else"; nl ();
printf "␣␣␣␣␣␣␣␣amp_11␣=␣(1-part_r)␣*␣sum(a11)␣+␣part_r␣*␣a11(3)"; nl ();
printf "␣␣␣␣␣␣end␣if"; nl ();
printf "␣␣␣␣␣amp_11␣=␣vev**4/s**2␣*␣amp_11"; nl ();
printf "␣␣end␣function␣da11"; nl();
nl ();
printf "␣␣%sfunction␣da20␣(cc,␣s,␣m)␣result␣(amp_20)" pure; nl ();
printf "␣␣␣␣␣␣real(kind=default),␣intent(in)␣::␣s"; nl ();
printf "␣␣␣␣␣␣real(kind=default),␣dimension(1:12),␣intent(in)␣::␣cc"; nl ();
printf "␣␣␣␣␣␣real(kind=default),␣dimension(1:5),␣intent(in)␣::␣m"; nl ();
printf "␣␣␣␣␣␣complex(kind=default)␣::␣a20_0,␣a20_1,␣a20_a,␣a20_f"; nl ();
printf "␣␣␣␣␣␣complex(kind=default),␣dimension(1:7)␣::␣a20"; nl ();
printf "␣␣␣␣␣␣complex(kind=default)␣::␣ii,␣jj,␣amp_20"; nl ();
printf "␣␣␣␣␣␣real(kind=default)␣::␣kappal,␣kappam,␣kappat"; nl ();
printf "␣␣␣␣␣␣ii␣=␣cmplx(0.0,1.0/32.0/Pi,default)"; nl ();
printf "␣␣␣␣␣␣jj␣=␣s**2/vev**4*ii"; nl ();
printf "␣␣␣␣␣␣!!!␣Scalar␣isosinglet"; nl ();
printf "␣␣␣␣␣␣kappal␣=␣cc(12)*((mass(23)**2+mass(24)**2)/m(4)**2-2*mass(23)**2*mass(24
printf "␣␣␣␣␣␣kappam␣=␣cc(12)*((mass(23)**4+mass(24)**4)/m(4)**2/(mass(23)**2+mass(24)
printf "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣-␣2*mass(23)**2*mass(24)**2/m(4)**4)"; nl ();
printf "␣␣␣␣␣␣kappat␣=␣cc(12)*mass(23)**2*mass(24)**2/m(4)**4"; nl ();
printf "␣␣␣␣␣␣!!!␣Longitudinal"; nl ();
printf "␣␣␣␣␣␣a20(1)␣=␣-2.0*cc(1)**2/vev**2␣*␣s0stu(s,m(1))"; nl ();
printf "␣␣␣␣␣␣!!!␣Scalar␣isoquintet"; nl ();
printf "␣␣␣␣␣␣a20(2)␣=␣-␣cc(2)**2/vev**2/6.␣*␣s0stu(s,m(2))"; nl ();
printf "␣␣␣␣␣␣if␣(cc(2)␣/=␣0)␣then"; nl ();
printf "␣␣␣␣␣␣␣␣a20(2)␣=␣a20(2)␣-␣cc(2)**2/vev**2/2.␣*&"; nl ();
printf "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣s**2/cmplx(s-m(2)**2,m(2)*wkm(2),default)"; nl ();
printf "␣␣␣␣␣␣end␣if"; nl ();
printf "␣␣␣␣␣␣!!!␣Vector␣isotriplet"; nl ();
printf "␣␣␣␣␣␣a20(3)␣=␣cc(3)**2*(2.0*p0stu(s,m(3))␣+␣3.0*s/m(3)**2)"; nl ();
printf "␣␣␣␣␣␣!!!␣Tensor␣isosinglet"; nl ();
printf "␣␣␣␣␣␣a20(4)␣=␣-␣cc(4)**2/vev**2*(d0stu(s,m(4)-2*kappal*s0stu(s,m(4)))␣&"; nl ()
printf "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣/3.0)"; nl ();
printf "␣␣␣␣␣␣!!!␣Tensor␣isoquintet"; nl ();
printf "␣␣␣␣␣␣a20(5)␣=␣-␣cc(5)**2/vev**2*(d0stu(s,m(5))␣&"; nl ();
printf "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣)/36.0"; nl ();
```

```
printf "␣␣␣␣␣␣!!!␣Transversal"; nl ();
printf "␣␣␣␣␣␣!!!␣Tensor␣isosinglet"; nl ();
printf "␣␣␣␣␣␣a20(6)␣=␣-cc(9)**2/Pi/vev**6*mass(23)**2*mass(24)**2/12␣*␣(s0stu(s,m(4))
printf "␣␣␣␣␣␣␣␣␣␣␣␣␣*␣(3*(1+2*s/m(4)**2+2*s**2/m(4)**4)+kappat␣)␣-␣3*(s/m(4)**2-s**2/
printf "␣␣␣␣␣␣!!!␣Mixed"; nl ();
printf "␣␣␣␣␣␣!!!␣Tensor␣isosinglet"; nl ();
printf "␣␣␣␣␣␣a20(7)␣=␣-cc(11)*cc(9)*cc(4)/Pi/vev**4*(mass(23)**2+mass(24)**2)/12␣*␣(s
printf "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣*␣(12*s/m(4)**2+12*s**2/m(4)**4+2*kappam␣)␣-␣6*(s/m(4)**2-s**2/
printf "␣␣␣␣␣␣!!!␣Fudge-Higgs"; nl ();
printf "␣␣␣␣␣␣a20_f␣=␣-␣fudge_higgs*s/vev**2"; nl ();
printf "␣␣␣␣␣␣a20_f␣=␣a20_f␣-␣0*2*(1-ghvva)**2/vev**2*mass(25)**2"; nl ();
printf "␣␣␣␣␣␣!!!␣Low␣energy␣theory␣alphas"; nl ();
printf "␣␣␣␣␣␣a20_0␣=␣␣16*(2*a4␣+␣a5)/3*s**2/vev**4"; nl ();
printf "␣␣␣␣␣␣a20_1␣=␣(10*log(lam_reg**2/s)/9␣+␣25/108.0_default)␣*␣s**2/vev**4"; nl ();
printf "␣␣␣␣␣␣a20_a␣=␣a20_0␣!!!␣+␣a20_1/16/Pi**2"; nl ();
printf "␣␣␣␣␣␣!!!␣Unitarize"; nl ();
printf "␣␣␣␣␣␣if␣(fudge_km␣/=␣0)␣then"; nl ();
printf "␣␣␣␣␣␣␣␣amp_20␣=␣sum(a20)+a20_f+a20_a"; nl();
printf "␣␣␣␣␣␣␣␣if␣(amp_20␣/=␣0)␣then"; nl ();
printf "␣␣␣␣␣␣␣␣␣␣amp_20␣=␣-␣a20_a␣-␣a20_f␣-␣part_r␣*␣(sum(a20)␣-␣a20(3))␣+␣1/(real(1/
printf "␣␣␣␣␣␣␣␣end␣if"; nl ();
printf "␣␣␣␣␣␣else"; nl ();
printf "␣␣␣␣␣␣␣␣amp_20␣=␣(1-part_r)␣*␣sum(a20)␣+␣part_r␣*␣a20(3)"; nl ();
printf "␣␣␣␣␣␣end␣if"; nl ();
printf "␣␣␣␣␣amp_20␣=␣vev**4/s**2␣*␣amp_20"; nl ();
printf "␣␣end␣function␣da20"; nl();
nl ();
printf "␣␣%sfunction␣da22␣(cc,␣s,␣m)␣result␣(amp_22)" pure; nl ();
printf "␣␣␣␣␣␣real(kind=default),␣intent(in)␣::␣s"; nl ();
printf "␣␣␣␣␣␣real(kind=default),␣dimension(1:12),␣intent(in)␣::␣cc"; nl ();
printf "␣␣␣␣␣␣real(kind=default),␣dimension(1:5),␣intent(in)␣::␣m"; nl ();
printf "␣␣␣␣␣␣complex(kind=default)␣::␣a22_0,␣a22_1,␣a22_a,␣a22_r"; nl ();
printf "␣␣␣␣␣␣complex(kind=default),␣dimension(1:7)␣::␣a22"; nl ();
printf "␣␣␣␣␣␣complex(kind=default)␣::␣ii,␣jj,␣amp_22"; nl ();
printf "␣␣␣␣␣␣real(kind=default)␣::␣kappal,␣kappam,␣kappat"; nl ();
printf "␣␣␣␣␣␣ii␣=␣cmplx(0.0,1.0/32.0/Pi,default)"; nl ();
printf "␣␣␣␣␣␣jj␣=␣s**2/vev**4*ii"; nl ();
printf "␣␣␣␣␣␣kappal␣=␣cc(12)*((mass(23)**2+mass(24)**2)/m(4)**2-2*mass(23)**2*mass(24
printf "␣␣␣␣␣␣kappam␣=␣cc(12)*((mass(23)**4+mass(24)**4)/m(4)**2/(mass(23)**2+mass(24)
printf "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣-␣2*mass(23)**2*mass(24)**2/m(4)**4)"; nl ();
printf "␣␣␣␣␣␣kappat␣=␣cc(12)*mass(23)**2*mass(24)**2/m(4)**4"; nl ();
printf "␣␣␣␣␣␣!!!␣Longitudinal"; nl ();
printf "␣␣␣␣␣␣!!!␣Scalar␣isosinglet"; nl ();
printf "␣␣␣␣␣␣a22(1)␣=␣-␣2.0*cc(1)**2/vev**2␣*␣s2stu(s,m(1))"; nl ();
printf "␣␣␣␣␣␣!!!␣Scalar␣isoquintet"; nl ();
printf "␣␣␣␣␣␣a22(2)␣=␣-␣cc(2)**2/vev**2␣*␣s2stu(s,m(2))␣/␣6.0"; nl ();
printf "␣␣␣␣␣␣!!!␣Vector␣triplet"; nl ();
printf "␣␣␣␣␣␣a22(3)␣=␣2.0*cc(3)**2*(2*s+m(3)**2)*s2stu(s,m(3))/m(3)**4"; nl ();
printf "␣␣␣␣␣␣!!!␣Tensor␣isosinglet"; nl ();
printf "␣␣␣␣␣␣a22(4)␣=␣-␣cc(4)**2/vev**2*((1.0␣+␣6.0*s/m(4)**2␣&"; nl ();
```

```
printf "␣␣␣␣␣␣␣␣␣␣␣␣+6.0*s**2/m(4)**4-2*kappal)*s2stu(s,m(4))/3.0)"; nl ();
printf "␣␣␣␣␣␣!!!␣Tensor␣isoquintet"; nl ();
printf "␣␣␣␣␣␣a22(5)␣=␣-␣cc(5)**2/vev**2/36.␣*␣&"; nl ();
printf "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣((1.+6.*s/m(5)**2+6.*s**2/m(5)**4␣)␣&"; nl ();
printf "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣*␣s2stu(s,m(5)))"; nl ();
printf "␣␣␣␣␣␣if␣(cc(5)␣/=␣0)␣then"; nl ();
printf "␣␣␣␣␣␣␣␣a22(5)␣=␣a22(5)␣-␣cc(5)**2/vev**2/60␣*␣&"; nl ();
printf "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣s**2/cmplx(s-m(5)**2,m(5)*wkm(5),default)"; nl ();
printf "␣␣␣␣␣␣end␣if"; nl ();
printf "␣␣␣␣␣␣!!!␣Transversal"; nl ();
printf "␣␣␣␣␣␣!!!␣Tensor␣isosinglet"; nl ();
printf "␣␣␣␣␣␣a22(6)␣=␣-cc(9)**2/Pi/vev**6*mass(23)**2*mass(24)**2/12␣*␣(s2stu(s,m(4))
printf "␣␣␣␣␣␣␣␣␣␣␣␣␣␣*␣(3*(1+2*s/m(4)**2+2*s**2/m(4)**4)+kappat␣))"; nl ();
printf "␣␣␣␣␣␣!!!␣Mixed"; nl ();
printf "␣␣␣␣␣␣!!!␣Tensor␣isosinglet"; nl ();
printf "␣␣␣␣␣␣a22(7)␣=␣-cc(11)*cc(9)*cc(4)/Pi/vev**4*(mass(23)**2+mass(24)**2)/12␣*␣(s
printf "␣␣␣␣␣␣␣␣␣␣␣␣␣␣*␣(12*s/m(4)**2+12*s**2/m(4)**4+2*kappam␣))"; nl ();
printf "␣␣␣␣␣␣!!!␣Low␣energy␣theory␣alphas"; nl ();
printf "␣␣␣␣␣␣a22_0␣=␣4*(a4␣+␣2*a5)/15*s**2/vev**4␣"; nl ();
printf "␣␣␣␣␣␣a22_1␣=␣(2*log(lam_reg**2/s)/45␣-␣247/5400.0_default)*s**2/vev**4"; nl ();
printf "␣␣␣␣␣␣a22_a␣=␣a22_0␣!!!␣+␣a22_1/16/Pi**2"; nl ();
printf "␣␣␣␣␣␣!!!␣Unitarize"; nl ();
printf "␣␣␣␣␣␣if␣(fudge_km␣/=␣0)␣then"; nl ();
printf "␣␣␣␣␣␣␣␣amp_22␣=␣sum(a22)+a22_a"; nl();
printf "␣␣␣␣␣␣␣␣if␣(amp_22␣/=␣0)␣then"; nl ();
printf "␣␣␣␣␣␣␣␣␣␣amp_22␣=␣-␣a22_a␣-␣part_r␣*␣(sum(a22)␣-␣a22(3))␣+␣1/(real(1/amp_22,d
printf "␣␣␣␣␣␣␣␣end␣if"; nl ();
printf "␣␣␣␣␣␣else"; nl ();
printf "␣␣␣␣␣␣␣␣amp_22␣=␣(1-part_r)␣*␣sum(a22)␣+␣part_r␣*␣a22(3)"; nl ();
printf "␣␣␣␣␣␣end␣if"; nl ();
printf "␣␣␣␣␣amp_22␣=␣vev**4/s**2␣*␣amp_22"; nl ();
printf "␣␣end␣function␣da22"; nl();
nl ();
printf "␣␣%sfunction␣dalzz0_s␣(cc,m,k)␣result␣(alzz0_s)" pure; nl ();
printf "␣␣␣␣␣␣type(momentum),␣intent(in)␣::␣k"; nl ();
printf "␣␣␣␣␣␣real(kind=default),␣dimension(1:12),␣intent(in)␣::␣cc"; nl ();
printf "␣␣␣␣␣␣real(kind=default),␣dimension(1:5),␣intent(in)␣::␣m"; nl ();
printf "␣␣␣␣␣␣complex(kind=default)␣::␣alzz0_s"; nl ();
printf "␣␣␣␣␣␣real(kind=default)␣::␣s"; nl ();
printf "␣␣␣␣␣␣s␣=␣k*k"; nl ();
printf "␣␣␣␣␣␣alzz0_s␣=␣2*g**4/costhw**2*((da00(cc,s,m)␣&"; nl ();
printf "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣-␣da20(cc,s,m))/24␣&"; nl ();
printf "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣-␣(5.)*(da02(cc,s,m)␣-␣da22(cc,s,m))/12)"; nl ();
printf "␣␣end␣function␣dalzz0_s"; nl ();
nl ();
printf "␣␣%sfunction␣dalzz0_t␣(cc,m,k)␣result␣(alzz0_t)" pure; nl ();
printf "␣␣␣␣␣␣type(momentum),␣intent(in)␣::␣k"; nl ();
printf "␣␣␣␣␣␣real(kind=default),␣dimension(1:12),␣intent(in)␣::␣cc"; nl ();
printf "␣␣␣␣␣␣real(kind=default),␣dimension(1:5),␣intent(in)␣::␣m"; nl ();
printf "␣␣␣␣␣␣complex(kind=default)␣::␣alzz0_t"; nl ();
```

```
printf "␣␣␣␣␣␣real(kind=default)␣::␣s"; nl ();
printf "␣␣␣␣␣␣s␣=␣k*k"; nl ();
printf "␣␣␣␣␣␣alzz0_t␣=␣(5.)*g**4/costhw**2*(da02(cc,s,m)␣-␣&"; nl ();
printf "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣da22(cc,s,m))/4"; nl ();
printf "␣␣end␣function␣dalzz0_t"; nl ();
nl ();
printf "␣␣%sfunction␣dalzz1_s␣(cc,m,k)␣result␣(alzz1_s)" pure; nl ();
printf "␣␣␣␣␣␣type(momentum),␣intent(in)␣::␣k"; nl ();
printf "␣␣␣␣␣␣real(kind=default),␣dimension(1:12),␣intent(in)␣::␣cc"; nl ();
printf "␣␣␣␣␣␣real(kind=default),␣dimension(1:5),␣intent(in)␣::␣m"; nl ();
printf "␣␣␣␣␣␣complex(kind=default)␣::␣alzz1_s"; nl ();
printf "␣␣␣␣␣␣real(kind=default)␣::␣s"; nl ();
printf "␣␣␣␣␣␣s␣=␣k*k"; nl ();
printf "␣␣␣␣␣␣alzz1_s␣=␣g**4/costhw**2*(da20(cc,s,m)/8␣&"; nl ();
printf "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣-␣(5.)*da22(cc,s,m)/4)"; nl ();
printf "␣␣end␣function␣dalzz1_s"; nl ();
nl ();
printf "␣␣%sfunction␣dalzz1_t␣(cc,m,k)␣result␣(alzz1_t)" pure; nl ();
printf "␣␣␣␣␣␣type(momentum),␣intent(in)␣::␣k"; nl ();
printf "␣␣␣␣␣␣real(kind=default),␣dimension(1:12),␣intent(in)␣::␣cc"; nl ();
printf "␣␣␣␣␣␣real(kind=default),␣dimension(1:5),␣intent(in)␣::␣m"; nl ();
printf "␣␣␣␣␣␣complex(kind=default)␣::␣alzz1_t"; nl ();
printf "␣␣␣␣␣␣real(kind=default)␣::␣s"; nl ();
printf "␣␣␣␣␣␣s␣=␣k*k"; nl ();
printf "␣␣␣␣␣␣alzz1_t␣=␣g**4/costhw**2*(-␣(3.)*da11(cc,s,m)/8␣&"; nl ();
printf "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣+␣3*(5.)*da22(cc,s,m)/8)"; nl ();
printf "␣␣end␣function␣dalzz1_t"; nl ();
nl ();
printf "␣␣%sfunction␣dalzz1_u␣(cc,m,k)␣result␣(alzz1_u)" pure; nl ();
printf "␣␣␣␣␣␣type(momentum),␣intent(in)␣::␣k"; nl ();
printf "␣␣␣␣␣␣real(kind=default),␣dimension(1:12),␣intent(in)␣::␣cc"; nl ();
printf "␣␣␣␣␣␣real(kind=default),␣dimension(1:5),␣intent(in)␣::␣m"; nl ();
printf "␣␣␣␣␣␣complex(kind=default)␣::␣alzz1_u"; nl ();
printf "␣␣␣␣␣␣real(kind=default)␣::␣s"; nl ();
printf "␣␣␣␣␣␣s␣=␣k*k"; nl ();
printf "␣␣␣␣␣␣alzz1_u␣=␣g**4/costhw**2*((3.)*da11(cc,s,m)/8␣&"; nl ();
printf "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣+␣3*(5.)*da22(cc,s,m)/8)"; nl ();
printf "␣␣end␣function␣dalzz1_u"; nl ();
nl ();
printf "␣␣%sfunction␣dalww0_s␣(cc,m,k)␣result␣(alww0_s)" pure; nl ();
printf "␣␣␣␣␣␣type(momentum),␣intent(in)␣::␣k"; nl ();
printf "␣␣␣␣␣␣real(kind=default),␣dimension(1:12),␣intent(in)␣::␣cc"; nl ();
printf "␣␣␣␣␣␣real(kind=default),␣dimension(1:5),␣intent(in)␣::␣m"; nl ();
printf "␣␣␣␣␣␣complex(kind=default)␣::␣alww0_s"; nl ();
printf "␣␣␣␣␣␣real(kind=default)␣::␣s"; nl ();
printf "␣␣␣␣␣␣s␣=␣k*k"; nl ();
printf "␣␣␣␣␣␣alww0_s␣=␣g**4*((2*da00(cc,s,m)␣+␣da20(cc,s,m))/24␣&"; nl ();
printf "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣-␣(5.)*(2*da02(cc,s,m)␣+␣da22(cc,s,m))/12)"; nl ();
printf "␣␣end␣function␣dalww0_s"; nl ();
nl ();
```

```
printf "␣␣%sfunction␣dalww0_t␣(cc,m,k)␣result␣(alww0_t)" pure; nl ();
printf "␣␣␣␣␣␣type(momentum),␣intent(in)␣::␣k"; nl ();
printf "␣␣␣␣␣␣real(kind=default),␣dimension(1:12),␣intent(in)␣::␣cc"; nl ();
printf "␣␣␣␣␣␣real(kind=default),␣dimension(1:5),␣intent(in)␣::␣m"; nl ();
printf "␣␣␣␣␣␣complex(kind=default)␣::␣alww0_t"; nl ();
printf "␣␣␣␣␣␣real(kind=default)␣::␣s"; nl ();
printf "␣␣␣␣␣␣s␣=␣k*k"; nl ();
printf "␣␣␣␣␣␣alww0_t␣=␣g**4*(2*(5.)*da02(cc,s,m)␣-␣(3.)*da11(cc,s,m)␣&"; nl ();
printf "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣+␣(5.)*da22(cc,s,m))/8"; nl ();
printf "␣␣end␣function␣dalww0_t"; nl ();
nl ();
printf "␣␣%sfunction␣dalww0_u␣(cc,m,k)␣result␣(alww0_u)" pure; nl ();
printf "␣␣␣␣␣␣type(momentum),␣intent(in)␣::␣k"; nl ();
printf "␣␣␣␣␣␣real(kind=default),␣dimension(1:12),␣intent(in)␣::␣cc"; nl ();
printf "␣␣␣␣␣␣real(kind=default),␣dimension(1:5),␣intent(in)␣::␣m"; nl ();
printf "␣␣␣␣␣␣complex(kind=default)␣::␣alww0_u"; nl ();
printf "␣␣␣␣␣␣real(kind=default)␣::␣s"; nl ();
printf "␣␣␣␣␣␣s␣=␣k*k"; nl ();
printf "␣␣␣␣␣␣alww0_u␣=␣g**4*(2*(5.)*da02(cc,s,m)␣+␣(3.)*da11(cc,s,m)␣&"; nl ();
printf "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣+␣(5.)*da22(cc,s,m))/8"; nl ();
printf "␣␣end␣function␣dalww0_u"; nl ();
nl ();
printf "␣␣%sfunction␣dalww2_s␣(cc,m,k)␣result␣(alww2_s)" pure; nl ();
printf "␣␣␣␣␣␣type(momentum),␣intent(in)␣::␣k"; nl ();
printf "␣␣␣␣␣␣real(kind=default),␣dimension(1:12),␣intent(in)␣::␣cc"; nl ();
printf "␣␣␣␣␣␣real(kind=default),␣dimension(1:5),␣intent(in)␣::␣m"; nl ();
printf "␣␣␣␣␣␣complex(kind=default)␣::␣alww2_s"; nl ();
printf "␣␣␣␣␣␣real(kind=default)␣::␣s"; nl ();
printf "␣␣␣␣␣␣s␣=␣k*k"; nl ();
printf "␣␣␣␣␣␣alww2_s␣=␣g**4*(da20(cc,s,m)␣-␣2*(5.)*da22(cc,s,m))/4␣"; nl ();
printf "␣␣end␣function␣dalww2_s"; nl ();
nl ();
printf "␣␣%sfunction␣dalww2_t␣(cc,m,k)␣result␣(alww2_t)" pure; nl ();
printf "␣␣␣␣␣␣type(momentum),␣intent(in)␣::␣k"; nl ();
printf "␣␣␣␣␣␣real(kind=default),␣dimension(1:12),␣intent(in)␣::␣cc"; nl ();
printf "␣␣␣␣␣␣real(kind=default),␣dimension(1:5),␣intent(in)␣::␣m"; nl ();
printf "␣␣␣␣␣␣complex(kind=default)␣::␣alww2_t"; nl ();
printf "␣␣␣␣␣␣real(kind=default)␣::␣s"; nl ();
printf "␣␣␣␣␣␣s␣=␣k*k"; nl ();
printf "␣␣␣␣␣␣alww2_t␣=␣3*(5.)*g**4*da22(cc,s,m)/4"; nl ();
printf "␣␣end␣function␣dalww2_t"; nl ();
nl ();
printf "␣␣%sfunction␣dalz4_s␣(cc,m,k)␣result␣(alz4_s)" pure; nl ();
printf "␣␣␣␣␣␣type(momentum),␣intent(in)␣::␣k"; nl ();
printf "␣␣␣␣␣␣real(kind=default),␣dimension(1:12),␣intent(in)␣::␣cc"; nl ();
printf "␣␣␣␣␣␣real(kind=default),␣dimension(1:5),␣intent(in)␣::␣m"; nl ();
printf "␣␣␣␣␣␣complex(kind=default)␣::␣alz4_s"; nl ();
printf "␣␣␣␣␣␣real(kind=default)␣::␣s"; nl ();
printf "␣␣␣␣␣␣s␣=␣k*k"; nl ();
printf "␣␣␣␣␣␣alz4_s␣=␣g**4/costhw**4*((da00(cc,s,m)␣&"; nl ();
```

```
printf "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣+␣2*da20(cc,s,m))/12␣&"; nl ();
printf "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣-␣(5.)*(da02(cc,s,m)+2*da22(cc,s,m))/6)"; nl ();
printf "␣␣end␣function␣dalz4_s"; nl ();
nl ();
printf "␣␣@[<5>";
printf "␣␣%sfunction␣dalz4_t␣(cc,m,k)␣result␣(alz4_t)" pure; nl ();
printf "␣␣␣␣␣␣type(momentum),␣intent(in)␣::␣k"; nl ();
printf "␣␣␣␣␣␣real(kind=default),␣dimension(1:12),␣intent(in)␣::␣cc"; nl ();
printf "␣␣␣␣␣␣real(kind=default),␣dimension(1:5),␣intent(in)␣::␣m"; nl ();
printf "␣␣␣␣␣␣complex(kind=default)␣::␣alz4_t"; nl ();
printf "␣␣␣␣␣␣real(kind=default)␣::␣s"; nl ();
printf "␣␣␣␣␣␣s␣=␣k*k"; nl ();
printf "␣␣␣␣␣␣alz4_t␣=␣g**4/costhw**4*(5.)*(da02(cc,s,m)␣&"; nl ();
printf "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣+␣2*da22(cc,s,m))/4"; nl ();
printf "␣␣end␣function␣dalz4_t"; nl ();
nl ();
end
```

—16—
Phase Space

16.1 Interface of Phasespace

module type T =
  sig
    type *momentum*

    type α *t*
    type α *decay*

Sort individual decays and complete phasespaces in a canonical order to determine topological equivalence classes.

    val *sort* : $(\alpha \to \alpha \to int) \to \alpha\ t \to \alpha\ t$
    val *sort_decay* : $(\alpha \to \alpha \to int) \to \alpha\ decay \to \alpha\ decay$

Functionals:

    val *map* : $(\alpha \to \beta) \to \alpha\ t \to \beta\ t$
    val *map_decay* : $(\alpha \to \beta) \to \alpha\ decay \to \beta\ decay$

    val *eval* : $(\alpha \to \beta) \to (\alpha \to \beta) \to (\alpha \to \beta \to \beta \to \beta) \to \alpha\ t \to \beta\ t$
    val *eval_decay* : $(\alpha \to \beta) \to (\alpha \to \beta \to \beta \to \beta) \to \alpha\ decay \to \beta\ decay$

of_momenta f1 f2 plist constructs the phasespace parameterization for a process $f_1 f_2 \to X$ with flavor decoration from pairs of outgoing momenta and flavors *plist* and initial flavors $f1$ and $f2$

    val *of_momenta* : $\alpha \to \alpha \to (momentum \times \alpha)\ list \to (momentum \times \alpha)\ t$
    val *decay_of_momenta* : $(momentum \times \alpha)\ list \to (momentum \times \alpha)\ decay$

    exception *Duplicate* of *momentum*
    exception *Unordered* of *momentum*
    exception *Incomplete* of *momentum*

  end

module *Make* (M : *Momentum*.T) : T with type *momentum* = $M.t$

16.2 Implementation of Phasespace

16.2.1 Tools

These are candidates for *ThoList* and not specific to phase space.

```
let rec first_match' mismatch f = function
  | [] → None
  | x :: rest →
      if f x then
        Some (x, List.rev_append mismatch rest)
      else
        first_match' (x :: mismatch) f rest
```

Returns $(x, X \setminus \{x\})$ if $\exists x \in X : f(x)$.

```
let first_match f l = first_match' [] f l
```

```
let rec first_pair' mismatch1 f l1 l2 =
  match l1 with
  | [] → None
  | x1 :: rest1 →
      begin match first_match (f x1) l2 with
      | None → first_pair' (x1 :: mismatch1) f rest1 l2
      | Some (x2, rest2) →
          Some ((x1, x2), (List.rev_append mismatch1 rest1, rest2))
      end
```

Returns $((x, y), (X \setminus \{x\}, Y \setminus \{y\}))$ if $\exists x \in X : \exists y \in Y : f(x, y)$.

```
let first_pair f l1 l2 = first_pair' [] f l1 l2
```

16.2.2 Phase Space Parameterization Trees

```
module type T =
  sig
    type momentum
    type α t
    type α decay
    val sort : (α → α → int) → α t → α t
    val sort_decay : (α → α → int) → α decay → α decay
    val map : (α → β) → α t → β t
    val map_decay : (α → β) → α decay → β decay
    val eval : (α → β) → (α → β) → (α → β → β → β) → α t → β t
    val eval_decay : (α → β) → (α → β → β → β) → α decay →
β decay
    val of_momenta : α → α → (momentum × α) list → (momentum ×
α) t
    val decay_of_momenta : (momentum × α) list → (momentum × α) decay
    exception Duplicate of momentum
    exception Unordered of momentum
    exception Incomplete of momentum
  end
```

```
module Make (M : Momentum.T) =
  struct

    type momentum = M.t
```

Finally, we came back to binary trees ...

Cascade Decays

```
    type α decay =
      | Leaf of α
      | Branch of α × α decay × α decay
```

Trees of type (*momentum* × α *option*) *decay* can be build easily and mapped to (*momentum* × α) *decay* later, once all the α slots are filled. A more elegant functor operating on β *decay* directly (with *Momentum* style functions defined for β) would not allow holes in the β *decay* during the construction.

```
    let label = function
      | Leaf p → p
      | Branch (p, _, _) → p

    let rec sort_decay cmp = function
      | Leaf _ as l → l
      | Branch (p, d1, d2) →
          let d1′ = sort_decay cmp d1
          and d2′ = sort_decay cmp d2 in
          if cmp (label d1′) (label d2′) ≤ 0 then
            Branch (p, d1′, d2′)
          else
            Branch (p, d2′, d1′)

    let rec map_decay f = function
      | Leaf p → Leaf (f p)
      | Branch (p, d1, d2) → Branch (f p, map_decay f d1, map_decay f d2)

    let rec eval_decay fl fb = function
      | Leaf p → Leaf (fl p)
      | Branch (p, d1, d2) →
          let d1′ = eval_decay fl fb d1
          and d2′ = eval_decay fl fb d2 in
          Branch (fb p (label d1′) (label d2′), d1′, d2′)
```

Assuming that $p > p_D \lor p = p_D \lor p < p_D$, where p_D is the overall momentum of a decay tree D, we can add p to D at the top or somewhere in the middle. Note that '<' is not a total ordering and the operation can fail (raise exceptions) if the set of momenta does not correspond to a tree. Also note that a momentum can already be present without flavor as a complement in a branching entered earlier.

```
    exception Duplicate of momentum
    exception Unordered of momentum
```

```
let rec embed_in_decay (p, f as pf) = function
  | Leaf (p', f' as pf') as d' ->
      if M.less p' p then
        Branch ((p, Some f), d', Leaf (M.sub p p', None))
      else if M.less p p' then
        Branch (pf', Leaf (p, Some f), Leaf (M.sub p' p, None))
      else if p = p' then
        begin match f' with
        | None -> Leaf (p, Some f)
        | Some _ -> raise (Duplicate p)
        end
      else
        raise (Unordered p)
  | Branch ((p', f' as pf'), d1, d2) as d' ->
      let p1, _ = label d1
      and p2, _ = label d2 in
      if M.less p' p then
        Branch ((p, Some f), d', Leaf (M.sub p p', None))
      else if M.lesseq p p1 then
        Branch (pf', embed_in_decay pf d1, d2)
      else if M.lesseq p p2 then
        Branch (pf', d1, embed_in_decay pf d2)
      else if p = p' then
        begin match f' with
        | None -> Branch ((p, Some f), d1, d2)
        | Some _ -> raise (Duplicate p)
        end
      else
        raise (Unordered p)
```

Note that both *embed_in_decay* and *embed_in_decays* below do *not* commute, and should process 'bigger' momenta first, because disjoint sub-momenta will create disjoint subtrees in the latter and raise exceptions in the former.

```
exception Incomplete of momentum

let finalize1 = function
  | p, Some f -> (p, f)
  | p, None -> raise (Incomplete p)

let finalize_decay t = map_decay finalize1 t
```

Process the momenta starting in with the highest *M.rank*:

```
let sort_momenta plist =
  List.sort (fun (p1, _) (p2, _) -> M.compare p1 p2) plist

let decay_of_momenta plist =
  match sort_momenta plist with
  | (p, f) :: rest ->
      finalize_decay (List.fold_right embed_in_decay rest (Leaf (p, Some f)))
  | [] -> invalid_arg "Phasespace.decay_of_momenta: empty"
```

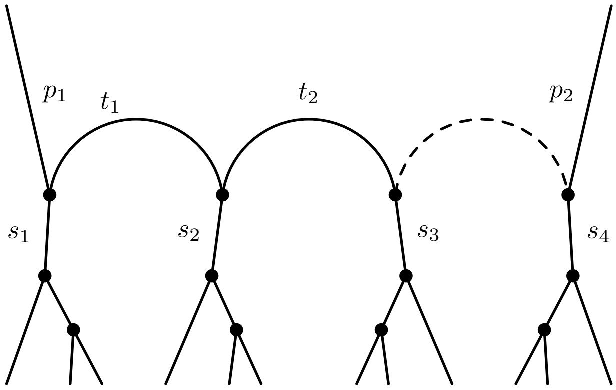


Figure 16.1: Phasespace parameterization for $2 \to n$ scattering by a sequence of cascade decays.

$2 \to n$ Scattering

A general $2 \to n$ scattering process can be parameterized by a sequence of cascade decays. The most symmetric representation is a little bit redundant and enters each t-channel momentum twice.

```
type α t = (α × α decay × α) list
```

let *topology* = *map snd* has type $(momentum \times \alpha)\ t \to \alpha\ t$ and can be used to define topological equivalence classes "up to permutations of momenta," which are useful for calculating Whizard "groves"[1] [11].

```
let sort cmp = List.map (fun (l, d, r) → (l, sort_decay cmp d, r))
let map f = List.map (fun (l, d, r) → (f l, map_decay f d, f r))
let eval ft fl fb = List.map (fun (l, d, r) → (ft l, eval_decay fl fb d, ft r))
```

Find a tree with a defined ordering relation with respect to p or create a new one at the end of the list.

```
let rec embed_in_decays (p, f as pf) = function
  | [] → [Leaf (p, Some f)]
  | d′ :: rest →
      let p′, _ = label d′ in
      if M.lesseq p′ p ∨ M.less p p′ then
        embed_in_decay pf d′ :: rest
      else
        d′ :: embed_in_decays pf rest
```

Collecting Ingredients

```
type α unfinished_decays =
```

[1]Not to be confused with gauge invariant classes of Feynman diagrams [12].

```
  { n : int;
    t_channel : (momentum × α option) list;
    decays : (momentum × α option) decay list }

let empty n = { n = n; t_channel = []; decays = [] }

let insert_in_unfinished_decays (p, f as pf) d =
  if M.Scattering.spacelike p then
    { d with t_channel = (p, Some f) :: d.t_channel }
  else
    { d with decays = embed_in_decays pf d.decays }

let flip_incoming plist =
  List.map (fun (p', f') → (M.Scattering.flip_s_channel_in p', f')) plist

let unfinished_decays_of_momenta n f2 p =
  List.fold_right insert_in_unfinished_decays
    (sort_momenta (flip_incoming ((M.of_ints n [2], f2) :: p))) (empty n)
```

Assembling Ingredients

```
let sort3 compare x y z =
  let a = [| x; y; z |] in
  Array.sort compare a;
  (a.(0), a.(1), a.(2))
```

Take advantage of the fact that sorting with *M.compare* sorts with *rising* values of *M.rank*:

```
let allows_momentum_fusion (p, _) (p1, _) (p2, _) =
  let p2', p1', p' = sort3 M.compare p p1 p2 in
  match M.try_fusion p' p1' p2' with
  | Some _ → true
  | None → false

let allows_fusion p1 p2 d = allows_momentum_fusion (label d) p1 p2

let rec thread_unfinished_decays' p acc tlist dlist =
  match first_pair (allows_fusion p) tlist dlist with
  | None → (p, acc, tlist, dlist)
  | Some ((t, _ as td), (tlist', dlist')) →
      thread_unfinished_decays' t (td :: acc) tlist' dlist'

let thread_unfinished_decays p c =
  match thread_unfinished_decays' p [] c.t_channel c.decays with
  | _, pairs, [], [] → pairs
  | _ → failwith "thread_unfinished_decays"

let rec combine_decays = function
  | [] → []
  | ((t, f as tf), d) :: rest →
      let p, _ = label d in
      begin match M.try_sub t p with
      | Some p' → (tf, d, (p', f)) :: combine_decays rest
```

```
        | None → (tf, d, (M.sub (M.neg t) p, f)) :: combine_decays rest
        end

  let finalize t = map finalize1 t

  let of_momenta f1 f2 = function
    | (p, _) :: _ as l →
        let n = M.dim p in
        finalize (combine_decays
                    (thread_unfinished_decays (M.of_ints n [1], Some f1)
                       (unfinished_decays_of_momenta n f2 l)))
    | [] → []
```

Diagnostics

```
  let p_to_string p =
    String.concat "" (List.map string_of_int (M.to_ints (M.abs p)))

  let rec to_string1 = function
    | Leaf p → "(" ^ p_to_string p ^ ")"
    | Branch (_, d1, d2) → "(" ^ to_string1 d1 ^ to_string1 d2 ^ ")"

  let to_string ps =
    String.concat "/"
      (List.map (fun (p1, d, p2) →
        p_to_string p1 ^ to_string1 d ^ p_to_string p2) ps)
```

Examples

```
  let try_thread_unfinished_decays p c =
    thread_unfinished_decays' p [] c.t_channel c.decays

  let try_of_momenta f = function
    | (p, _) :: _ as l →
        let n = M.dim p in
        try_thread_unfinished_decays
          (M.of_ints n [1], None) (unfinished_decays_of_momenta n f l)
    | [] → invalid_arg "try_of_momenta"

end
```

—17— Whizard

Talk to [11].

17.1 Interface of Whizard

```
module type T =
  sig
    type t
    type amplitude
    val trees : amplitude → t
    val merge : t → t
    val write : out_channel → string → t → unit
  end

module Make (FM : Fusion.Maker) (P : Momentum.T)
    (PW : Momentum.Whizard with type t = P.t) (M : Model.T) :
    T with type amplitude = FM(P)(M).amplitude

val write_interface : out_channel → string list → unit
val write_makefile : out_channel → α → unit
val write_makefile_processes : out_channel → string list → unit
```

17.2 Implementation of Whizard

```
let rcs = RCS.parse "Whizard" ["Whizard␣Interface"]
    { RCS.revision = "$Revision:␣6465␣$";
      RCS.date = "$Date:␣2015-01-10␣16:22:31␣+0100␣(Sat,␣10␣Jan␣2015)␣$";
      RCS.author = "$Author:␣jr_reuter␣$";
      RCS.source
        = "$URL:␣svn+ssh://jr_reuter@login.hepforge.org/hepforge/svn/whizard/trunk/omega/sr

open Printf

module type T =
  sig
    type t
    type amplitude
```

```
    val trees : amplitude → t
    val merge : t → t
    val write : out_channel → string → t → unit
  end

module Make (FM : Fusion.Maker) (P : Momentum.T)
    (PW : Momentum.Whizard with type t = P.t) (M : Model.T) =
  struct
    module F = FM(P)(M)

    type tree = (P.t × F.flavor list) list

    module Poles = Map.Make
        (struct
          type t = int × int
          let compare (s1, t1) (s2, t2) =
            let c = compare s2 s1 in
            if c ≠ 0 then
              c
            else
              compare t1 t2
        end)

    let add_tree maps tree trees =
      Poles.add maps
        (try tree :: (Poles.find maps trees) with Not_found → [tree]) trees

    type t =
        { in1 : F.flavor;
          in2 : F.flavor;
          out : F.flavor list;
          trees : tree list Poles.t }

    type amplitude = F.amplitude
```

17.2.1 Building Trees

A singularity is to be mapped if it is timelike and not the overall s-channel.

```
    let timelike_map c = P.Scattering.timelike c ∧ ¬ (P.Scattering.s_channel c)

    let count_maps n clist =
      List.fold_left (fun (s, t as cnt) (c, _) →
        if timelike_map c then
          (succ s, t)
        else if P.Scattering.spacelike c then
          (s, succ t)
        else
          cnt) (0, 0) clist

    let poles_to_whizard n trees poles =
      let tree = List.map (fun wf →
        (P.Scattering.flip_s_channel_in (F.momentum wf), [F.flavor wf])) poles in
      add_tree (count_maps n tree) tree trees
```

I must reinstate the *conjugate* eventually!

```
let trees a =
  match F.externals a with
  | in1 :: in2 :: out ->
      let n = List.length out + 2 in
      { in1 = F.flavor in1;
        in2 = F.flavor in2;
        out = List.map (fun f -> (* M.conjugate *) (F.flavor f)) out;
        trees = List.fold_left
          (poles_to_whizard n) Poles.empty (F.poles a) }
  | _ -> invalid_arg "Whizard().trees"
```

17.2.2 Merging Homomorphic Trees

```
module Pole_Map =
  Map.Make (struct type t = P.t list let compare = compare end)
module Flavor_Set =
  Set.Make (struct type t = F.flavor let compare = compare end)

let add_flavors flist fset =
  List.fold_right Flavor_Set.add flist fset

let set_of_flavors flist =
  List.fold_right Flavor_Set.add flist Flavor_Set.empty

let pack_tree map t =
  let c, f =
    List.split (List.sort (fun (c1, _) (c2, _) ->
      compare (PW.of_momentum c2) (PW.of_momentum c1)) t) in
  let f' =
    try
      List.map2 add_flavors f (Pole_Map.find c map)
    with
    | Not_found -> List.map set_of_flavors f in
  Pole_Map.add c f' map

let pack_map trees = List.fold_left pack_tree Pole_Map.empty trees

let merge_sets clist flist =
  List.map2 (fun c f -> (c, Flavor_Set.elements f)) clist flist

let unpack_map map =
  Pole_Map.fold (fun c f l -> (merge_sets c f) :: l) map []
```

If a singularity is to be mapped (i. e. if it is timelike and not the overall s-channel), expand merged particles again:

```
let unfold1 (c, f) =
  if timelike_map c then
    List.map (fun f' -> (c, [f'])) f
  else
    [(c,f)]
```

```
let unfold_tree tree = Product.list (fun x → x) (List.map unfold1 tree)

let unfold trees = ThoList.flatmap unfold_tree trees

let merge t =
  { t with trees = Poles.map
      (fun t′ → unfold (unpack_map (pack_map t′))) t.trees }
```

17.2.3 Printing Trees

```
let flavors_to_string f =
  String.concat "/" (List.map M.flavor_to_string f)

let whizard_tree t =
  "tree␣" ^
  (String.concat "␣" (List.rev_map (fun (c, _) →
    (string_of_int (PW.of_momentum c))) t)) ^
  "␣!␣" ^
  (String.concat ",␣" (List.rev_map (fun (_, f) → flavors_to_string f) t))

let whizard_tree_debug t =
  "tree␣" ^
  (String.concat "␣" (List.rev_map (fun (c, _) →
    ("[" ^ (String.concat "+" (List.map string_of_int (P.to_ints c))) ^ "]"))
                              (List.sort (fun (t1, _) (t2, _) →
                                let c =
                                  compare
                                    (List.length (P.to_ints t2))
                                    (List.length (P.to_ints t1)) in
                                if c ≠ 0 then
                                  c
                                else
                                  compare t1 t2) t))) ^
  "␣!␣" ^
  (String.concat ",␣" (List.rev_map (fun (_, f) → flavors_to_string f) t))

let format_maps = function
  | (0, 0) → "neither␣mapped␣timelike␣nor␣spacelike␣poles"
  | (0, 1) → "no␣mapped␣timelike␣poles,␣one␣spacelike␣pole"
  | (0, n) → "no␣mapped␣timelike␣poles,␣" ^
      string_of_int n ^ "␣spacelike␣poles"
  | (1, 0) → "one␣mapped␣timelike␣pole,␣no␣spacelike␣pole"
  | (1, 1) → "one␣mapped␣timelike␣and␣spacelike␣pole␣each"
  | (1, n) → "one␣mapped␣timelike␣and␣" ^
      string_of_int n ^ "␣spacelike␣poles"
  | (n, 0) → string_of_int n ^
      "␣mapped␣timelike␣poles␣and␣no␣spacelike␣pole"
  | (n, 1) → string_of_int n ^
      "␣mapped␣timelike␣poles␣and␣one␣spacelike␣pole"
  | (n, n′) → string_of_int n ^ "␣mapped␣timelike␣and␣" ^
      string_of_int n′ ^ "␣spacelike␣poles"
```

```
let format_flavor f =
  match flavors_to_string f with
  | "d" → "d" | "dbar" → "D"
  | "u" → "u" | "ubar" → "U"
  | "s" → "s" | "sbar" → "S"
  | "c" → "c" | "cbar" → "C"
  | "b" → "b" | "bbar" → "B"
  | "t" → "t" | "tbar" → "T"
  | "e-" → "e1" | "e+" → "E1"
  | "nue" → "n1" | "nuebar" → "N1"
  | "mu-" → "e2" | "mu+" → "E2"
  | "numu" → "n2" | "numubar" → "N2"
  | "tau-" → "e3" | "tau+" → "E3"
  | "nutau" → "n3" | "nutaubar" → "N3"
  | "g" → "G" | "A" → "A" | "Z" → "Z"
  | "W+" → "W+" | "W-" → "W-"
  | "H" → "H"
  | s → s ^ " (not translated)"

module Mappable =
  Set.Make (struct type t = string let compare = compare end)
let mappable =
  List.fold_right Mappable.add
    [ "T"; "Z"; "W+"; "W-"; "H" ] Mappable.empty

let analyze_tree ch t =
  List.iter (fun (c, f) →
    let f′ = format_flavor f
    and c′ = PW.of_momentum c in
    if P.Scattering.timelike c then begin
      if P.Scattering.s_channel c then
        fprintf ch "      ! overall s-channel %d %s not mapped\n" c′ f′
      else if Mappable.mem f′ mappable then
        fprintf ch "      map %d s-channel %s\n" c′ f′
      else
        fprintf ch
          "      ! %d s-channel %s can't be mapped by whizard\n"
          c′ f′
    end else
      fprintf ch "      ! t-channel %d %s not mapped\n" c′ f′) t

let write ch pid t =
  failwith "Whizard.Make().write: incomplete"
  fprintf ch "process %s\n" pid;
  Poles.iter (fun maps ds →
    fprintf ch "\n    ! %d times %s:\n"
      (List.length ds) (format_maps maps);
    List.iter (fun d →
      fprintf ch "\n    grove\n";
      fprintf ch "    %s\n" (whizard_tree d);
      analyze_tree ch d) ds) t.trees;
  fprintf ch "\n! O'Mega revision control information:\n";
```

```
    List.iter (fun s → fprintf ch "!␣␣␣␣%s\n" s)
      (ThoList.flatmap RCS.summary (rcs :: M.rcs :: F.rcs_list));
    fprintf ch "\n"
i × )
  end
```

17.2.4 Process Dispatcher

```
let arguments = function
  | [] → ("", "")
  | args →
      let arg_list = String.concat ",␣" (List.map snd args) in
      (arg_list, ",␣" ^ arg_list)

let import_prefixed ch pid name =
  fprintf ch "␣␣␣␣use␣%s,␣only:␣%s_%s␣=>␣%s␣!NODEP!\n"
    pid pid name name

let declare_argument ch (arg_type, arg) =
  fprintf ch "␣␣␣␣%s,␣intent(in)␣::␣%s\n" arg_type arg

let call_function ch pid result name args =
  fprintf ch "␣␣␣␣␣␣␣case␣(pr_%s)\n" pid;
  fprintf ch "␣␣␣␣␣␣␣␣␣␣%s␣=␣%s_%s␣(%s)\n" result pid name args

let default_function ch result default =
  fprintf ch "␣␣␣␣␣␣␣case␣default\n";
  fprintf ch "␣␣␣␣␣␣␣␣␣␣call␣invalid_process␣(pid)\n";
  fprintf ch "␣␣␣␣␣␣␣␣␣␣%s␣=␣%s\n" result default

let call_subroutine ch pid name args =
  fprintf ch "␣␣␣␣␣␣␣case␣(pr_%s)\n" pid;
  fprintf ch "␣␣␣␣␣␣␣␣␣␣call␣%s_%s␣(%s)\n" pid name args

let default_subroutine ch =
  fprintf ch "␣␣␣␣␣␣␣case␣default\n";
  fprintf ch "␣␣␣␣␣␣␣␣␣␣call␣invalid_process␣(pid)\n"

let write_interface_subroutine ch wrapper name args processes =
  let arg_list, arg_list′ = arguments args in
  fprintf ch "␣␣subroutine␣%s␣(pid%s)\n" wrapper arg_list′;
  List.iter (fun p → import_prefixed ch p name) processes;
  List.iter (declare_argument ch) (("character(len=*)", "pid") :: args);
  fprintf ch "␣␣␣␣select␣case␣(pid)\n";
  List.iter (fun p → call_subroutine ch p name arg_list) processes;
  default_subroutine ch;
  fprintf ch "␣␣␣␣end␣select\n";
  fprintf ch "␣␣end␣subroutine␣%s\n" wrapper

let write_interface_function ch wrapper name
    (result_type, result, default) args processes =
  let arg_list, arg_list′ = arguments args in
  fprintf ch "␣␣function␣%s␣(pid%s)␣result␣(%s)\n" wrapper arg_list′ result;
```

```
  List.iter (fun p → import_prefixed ch p name) processes;
  List.iter (declare_argument ch) (("character(len=*)", "pid") :: args);
  fprintf ch "␣␣␣␣%s␣::␣%s\n" result_type result;
  fprintf ch "␣␣␣␣select␣case␣(pid)\n";
  List.iter (fun p → call_function ch p result name arg_list) processes;
  default_function ch result default;
  fprintf ch "␣␣␣␣end␣select\n";
  fprintf ch "␣␣end␣function␣%s\n" wrapper
let write_other_interface_functions ch =
  fprintf ch "␣␣subroutine␣invalid_process␣(pid)\n";
  fprintf ch "␣␣␣␣character(len=*),␣intent(in)␣::␣pid\n";
  fprintf ch "␣␣␣␣print␣*,␣\"PANIC:";
  fprintf ch "␣process␣‘\"//trim(pid)//\"’␣not␣available!\"\n";
  fprintf ch "␣␣end␣subroutine␣invalid_process\n";
  fprintf ch "␣␣function␣n_tot␣(pid)␣result␣(n)\n";
  fprintf ch "␣␣␣␣character(len=*),␣intent(in)␣::␣pid\n";
  fprintf ch "␣␣␣␣integer␣::␣n\n";
  fprintf ch "␣␣␣␣n␣=␣n_in(pid)␣+␣n_out(pid)\n";
  fprintf ch "␣␣end␣function␣n_tot\n"
let write_other_declarations ch =
  fprintf ch "␣␣public␣::␣n_in,␣n_out,␣n_tot,␣pdg_code\n";
  fprintf ch "␣␣public␣::␣allow_helicities\n";
  fprintf ch "␣␣public␣::␣create,␣destroy\n";
  fprintf ch "␣␣public␣::␣set_const,␣sqme\n";
  fprintf ch "␣␣interface␣create\n";
  fprintf ch "␣␣␣␣␣module␣procedure␣process_create\n";
  fprintf ch "␣␣end␣interface\n";
  fprintf ch "␣␣interface␣destroy\n";
  fprintf ch "␣␣␣␣␣module␣procedure␣process_destroy\n";
  fprintf ch "␣␣end␣interface\n";
  fprintf ch "␣␣interface␣set_const\n";
  fprintf ch "␣␣␣␣␣module␣procedure␣process_set_const\n";
  fprintf ch "␣␣end␣interface\n";
  fprintf ch "␣␣interface␣sqme\n";
  fprintf ch "␣␣␣␣␣module␣procedure␣process_sqme\n";
  fprintf ch "␣␣end␣interface\n"
let write_interface ch names =
  fprintf ch "module␣process_interface\n";
  fprintf ch "␣␣use␣kinds,␣only:␣default␣␣!NODEP!\n";
  fprintf ch "␣␣use␣parameters,␣only:␣parameter_set\n";
  fprintf ch "␣␣implicit␣none\n";
  fprintf ch "␣␣private\n";
  List.iter (fun p →
    fprintf ch
      "␣␣character(len=*),␣parameter,␣public␣::␣pr_%s␣=␣\"%s\"\n" p p)
    names;
  write_other_declarations ch;
  fprintf ch "contains\n";
  write_interface_function ch "n_in" "n_in" ("integer", "n", "0") [] names;
```

```
  write_interface_function ch "n_out" "n_out" ("integer", "n", "0") [] names;
  write_interface_function ch "pdg_code" "pdg_code"
    ("integer", "n", "0") [ "integer", "i" ] names;
  write_interface_function ch "allow_helicities" "allow_helicities"
    ("logical", "yorn", ".false.") [] names;
  write_interface_subroutine ch "process_create" "create" [] names;
  write_interface_subroutine ch "process_destroy" "destroy" [] names;
  write_interface_subroutine ch "process_set_const" "set_const"
    [ "type(parameter_set)", "par"] names;
  write_interface_function ch "process_sqme" "sqme"
    ("real(kind=default)", "sqme", "0")
    [ "real(kind=default),␣dimension(0:,:)", "p";
      "integer,␣dimension(:),␣optional", "h" ] names;
  write_other_interface_functions ch;
  fprintf ch "end␣module␣process_interface\n"
```

17.2.5 Makefile

```
let write_makefile ch names =
  fprintf ch "KINDS␣=␣../@KINDS@\n";
  fprintf ch "HELAS␣=␣../@HELAS@\n";
  fprintf ch "F90␣=␣@F90@\n";
  fprintf ch "F90FLAGS␣=␣@F90FLAGS@\n";
  fprintf ch "F90INCL␣=␣-I$(KINDS)␣-I$(HELAS)\n";
  fprintf ch "F90COMMON␣=␣omega_bundle_whizard.f90";
  fprintf ch "␣file_utils.f90␣process_interface.f90\n";
  fprintf ch "include␣Makefile.processes\n";
  fprintf ch "F90SRC␣=␣$(F90COMMON)␣$(F90PROCESSES)\n";
  fprintf ch "OBJ␣=␣$(F90SRC:.f90=.o)\n";
  fprintf ch "MOD␣=␣$(F90SRC:.f90=.mod)\n";
  fprintf ch "archive:␣processes.a\n";
  fprintf ch "processes.a:␣$(OBJ)\n";
  fprintf ch "\t$(AR)␣r␣$@␣$(OBJ)\n";
  fprintf ch "\t@RANLIB@␣$@\n";
  fprintf ch "clean:\n";
  fprintf ch "\trm␣-f␣$(OBJ)\n";
  fprintf ch "realclean:\n";
  fprintf ch "\trm␣-f␣processes.a\n";
  fprintf ch "parameters.o:␣file_utils.o\n";
  fprintf ch "omega_bundle_whizard.o:␣parameters.o\n";
  fprintf ch "process_interface.o:␣parameters.o\n";
  fprintf ch "%%.o:␣%%.f90␣$(KINDS)/kinds.f90\n";
  fprintf ch "\t$(F90)␣$(F90FLAGS)␣$(F90INCL)␣-c␣$<\n"

let write_makefile_processes ch names =
  fprintf ch "F90PROCESSES␣=";
  List.iter (fun f → fprintf ch "␣\\\n␣␣%s.f90" f) names;
  fprintf ch "\n";
  List.iter (fun f →
    fprintf ch "%s.o:␣omega_bundle_whizard.o␣parameters.o\n" f;
```

fprintf ch `"process_interface.o:␣%s.o\n"` *f*) *names*

—18— Applications

18.1 Sample

18.2 Interface of Omega

```
module type T =
  sig
    val main : unit → unit
```

This used to be only intended for debugging O'Giga, but might live longer ...

```
    type flavor
    val diagrams : flavor → flavor → flavor list →
      ((flavor × Momentum.Default.t) ×
          (flavor × Momentum.Default.t,
           flavor × Momentum.Default.t) Tree.t) list
  end
```

```
module Make (FM : Fusion.Maker) (TM : Target.Maker) (M : Model.T) :
    T with type flavor = M.flavor
```

18.3 Implementation of Omega

```
let (<<) f g x = f (g x)
let (>>) f g x = g (f x)
```

```
module P = Momentum.Default
module P_Whizard = Momentum.DefaultW
```

```
module type T =
  sig
    val main : unit → unit
    type flavor
    val diagrams : flavor → flavor → flavor list →
      ((flavor × Momentum.Default.t) ×
          (flavor × Momentum.Default.t,
```

```
          flavor × Momentum.Default.t) Tree.t) list
  end

module Make (Fusion_Maker : Fusion.Maker) (Target_Maker : Target.Maker) (M : Model.T) =
  struct

    module CM = Colorize.It(M)

    type flavor = M.flavor

    module Proc = Process.Make(M)
```

NB: this causes the constant initializers in *Fusion_Maker* more than once. Such side effects must be avoided if the initializers involve expensive computations. *Relying on the fact that the functor will be called only once is not a good idea!*

```
    module F = Fusion_Maker(P)(M)
    module CF = Fusion.Multi(Fusion_Maker)(P)(M)
    module T = Target_Maker(Fusion_Maker)(P)(M)
    module W = Whizard.Make(Fusion_Maker)(P)(P_Whizard)(M)
    module C = Cascade.Make(M)(P)
```

Form a α *list* from a α *option array*, containing the elements that are not *None* in order.

```
    let opt_array_to_list a =
      let rec opt_array_to_list' acc i a =
        if i < 0 then
          acc
        else
          begin match a.(i) with
          | None → opt_array_to_list' acc (pred i) a
          | Some x → opt_array_to_list' (x :: acc) (pred i) a
          end in
      opt_array_to_list' [] (Array.length a − 1) a
```

Return a list of *CF.amplitude list*s, corresponig to the diagrams for a specific color flow for each flavor combination.

```
    let amplitudes_by_flavor amplitudes =
      List.map opt_array_to_list (Array.to_list (CF.process_table amplitudes))
```

If we plan to distiguish different couplings later on, we can no long map all instances of *coupling option* in the tree to *None*. In this case, we will need to normalize different fusion orders *Coupling.fuse2*, *Coupling.fuse3* or *Coupling.fusen*, because they would otherwise lead to inequivalent diagrams. Unfortunately, this stuff packaged deep in *Fusion.Tagged_Coupling*.

The *Tree.canonicalize* below should be necessary to remove topologically equivalent duplicates.

Take a *CF.amplitude list* assumed to correspond to the same external states after stripping the color and return a pair of the list of external particles and the corresponding Feynman diagrams without color.

```
let wf1 amplitude =
  match F.externals amplitude with
  | wf :: _ → wf
  | [] → failwith "Omega.forest_sans_color:␣no␣external␣particles"

let uniq l =
  ThoList.uniq (List.sort compare l)

let forest_sans_color = function
  | amplitude :: _ as amplitudes →
    let externals = F.externals amplitude in
    let prune_color wf =
      (F.flavor_sans_color wf, F.momentum_list wf) in
    let prune_color_and_couplings (wf, c) =
      (prune_color wf, None) in
    (List.map prune_color externals,
     uniq
       (List.map
          (fun t →
            Tree.canonicalize
              (Tree.map prune_color_and_couplings prune_color t))
          (ThoList.flatmap (fun a → F.forest (wf1 a) a) amplitudes)))
  | [] → ([], [])

let p2s p =
  if p ≥ 0 ∧ p ≤ 9 then
    string_of_int p
  else if p ≤ 36 then
    String.make 1 (Char.chr (Char.code 'A' + p − 10))
  else
    "_"

let format_p wf =
  String.concat "" (List.map p2s (F.momentum_list wf))

let variable wf =
  M.flavor_to_string (F.flavor_sans_color wf) ^ "[" ^ format_p wf ^ "]"

let variable' wf =
  CM.flavor_to_TeX (F.flavor wf) ^ "(" ^ format_p wf ^ ")"

let feynmf_style propagator color =
  { Tree.style =
      begin match propagator with
      | Coupling.Prop_Feynman
      | Coupling.Prop_Gauge _ →
        begin match color with
        | Color.AdjSUN _ → Some ("gluon", "")
        | _ → Some ("boson", "")
        end
```

```
      | Coupling.Prop_Col_Feynman → Some ("gluon", "")
      | Coupling.Prop_Unitarity
      | Coupling.Prop_Rxi _ → Some ("dbl_wiggly", "")
      | Coupling.Prop_Spinor
      | Coupling.Prop_ConjSpinor → Some ("fermion", "")
      | _ → None
      end;
    Tree.rev =
      begin match propagator with
      | Coupling.Prop_Spinor → true
      | Coupling.Prop_ConjSpinor → false
      | _ → false
      end;
    Tree.label = None;
    Tree.tension = None }

let header incoming outgoing =
  "$␣" ^
  String.concat "␣"
    (List.map (CM.flavor_to_TeX << F.flavor) incoming) ^
  "␣\\to␣" ^
  String.concat "␣"
    (List.map (CM.flavor_to_TeX << CM.conjugate << F.flavor) outgoing) ^
  "␣$"

let header_sans_color incoming outgoing =
  "$␣" ^
  String.concat "␣"
    (List.map (M.flavor_to_TeX << fst) incoming) ^
  "␣\\to␣" ^
  String.concat "␣"
    (List.map (M.flavor_to_TeX << M.conjugate << fst) outgoing) ^
  "␣$"

let diagram incoming tree =
  let fmf wf =
    let f = F.flavor wf in
    feynmf_style (CM.propagator f) (CM.color f) in
  Tree.map
    (fun (n, _) →
      let n' = fmf n in
      if List.mem n incoming then
        { n' with Tree.rev = ¬ n'.Tree.rev }
      else
        n')
    (fun l →
      if List.mem l incoming then
        l
      else
        F.conjugate l)
    tree
```

```
let diagram_sans_color incoming (tree) =
  let fmf (f, p) =
    feynmf_style (M.propagator f) (M.color f) in
  Tree.map
    (fun (n, c) →
      let n' = fmf n in
      if List.mem n incoming then
        { n' with Tree.rev = ¬ n'.Tree.rev }
      else
        n')
    (fun (f, p) →
      if List.mem (f, p) incoming then
        (f, p)
      else
        (M.conjugate f, p))
    tree

let feynmf_set amplitude =
  match F.externals amplitude with
  | wf1 :: wf2 :: wfs →
    let incoming = [wf1; wf2] in
    { Tree.header = header incoming wfs;
      Tree.incoming = incoming;
      Tree.diagrams =
        List.map (diagram incoming) (F.forest wf1 amplitude) }
  | _ → failwith "less␣than␣two␣external␣particles"

let feynmf_set_sans_color (externals, trees) =
  match externals with
  | wf1 :: wf2 :: wfs →
    let incoming = [wf1; wf2] in
    { Tree.header = header_sans_color incoming wfs;
      Tree.incoming = incoming;
      Tree.diagrams =
        List.map (diagram_sans_color incoming) trees }
  | _ → failwith "less␣than␣two␣external␣particles"

let feynmf_set_sans_color_empty (externals, trees) =
  match externals with
  | wf1 :: wf2 :: wfs →
    let incoming = [wf1; wf2] in
    { Tree.header = header_sans_color incoming wfs;
      Tree.incoming = incoming;
      Tree.diagrams = [] }
  | _ → failwith "less␣than␣two␣external␣particles"

let uncolored_colored amplitudes =
  { Tree.outer = feynmf_set_sans_color (forest_sans_color amplitudes);
    Tree.inner = List.map feynmf_set amplitudes }

let uncolored_only amplitudes =
  { Tree.outer = feynmf_set_sans_color (forest_sans_color amplitudes);
    Tree.inner = [] }
```

```
let colored_only amplitudes =
  { Tree.outer = feynmf_set_sans_color_empty (forest_sans_color amplitudes);
    Tree.inner = List.map feynmf_set amplitudes }

let momentum_to_TeX (_, p) =
  String.concat "" (List.map p2s p)

let wf_to_TeX (f, _ as wf) =
  M.flavor_to_TeX f ^ "(" ^ momentum_to_TeX wf ^ ")"

let amplitudes_to_feynmf latex name amplitudes =
    Tree.feynmf_sets_wrapped latex name
      wf_to_TeX momentum_to_TeX variable' format_p
      (List.map uncolored_colored (amplitudes_by_flavor amplitudes))

let amplitudes_to_feynmf_sans_color latex name amplitudes =
    Tree.feynmf_sets_wrapped latex name
      wf_to_TeX momentum_to_TeX variable' format_p
      (List.map uncolored_only (amplitudes_by_flavor amplitudes))

let amplitudes_to_feynmf_color_only latex name amplitudes =
    Tree.feynmf_sets_wrapped latex name
      wf_to_TeX momentum_to_TeX variable' format_p
      (List.map colored_only (amplitudes_by_flavor amplitudes))

let version () =
  List.iter (fun s → prerr_endline ("RCS: " ^ s))
    (ThoList.flatmap RCS.summary (CM.rcs :: T.rcs_list @ F.rcs_list))

let debug (str, descr, opt, var) =
  [ "-warning:" ^ str, Arg.Unit (fun () → var := (opt, false) :: !var),
    "          check " ^ descr ^ " and print warning on error";
    "-error:" ^ str, Arg.Unit (fun () → var := (opt, true) :: !var),
    "            check " ^ descr ^ " and terminate on error" ]

let rec include_goldstones = function
  | [] → false
  | (T.Gauge, _) :: _ → true
  | _ :: rest → include_goldstones rest

let read_lines_rev file =
  let ic = open_in file in
  let rev_lines = ref [] in
  let rec slurp () =
    rev_lines := input_line ic :: !rev_lines;
    slurp () in
  try
    slurp ()
  with
  | End_of_file →
      close_in ic;
      !rev_lines

let read_lines file =
  List.rev (read_lines_rev file)
```

```
type cache_mode =
  | Cache_Default
  | Cache_Initialize of string

let cache_option =
  ref Cache_Default

let unphysical_polarization = ref None
```

18.3.1 Main Program

```
let main () =
  let usage =
    "usage: " ^ Sys.argv.(0) ^
    " [options] [" ^
      String.concat "|" (List.map M.flavor_to_string
                           (ThoList.flatmap snd
                              (M.external_flavors ()))) ^ "]"
  and rev_scatterings = ref []
  and rev_decays = ref []
  and cascades = ref []
  and checks = ref []
  and output_file = ref None
  and print_forest = ref false
  and template = ref false
  and diagrams_all = ref None
  and diagrams_sans_color = ref None
  and diagrams_color_only = ref None
  and diagrams_LaTeX = ref false
  and quiet = ref false
  and write = ref true
  and params = ref false
  and poles = ref false
  and dag_out = ref None
  and dag0_out = ref None in
  Arg.parse
    (Options.cmdline "-target:" T.options @
     Options.cmdline "-model:" M.options @
     Options.cmdline "-fusion:" CF.options @
     ThoList.flatmap debug
       ["a", "arguments", T.All, checks;
        "n", "# of input arguments", T.Arguments, checks;
        "m", "input momenta", T.Momenta, checks;
        "g", "internal Ward identities", T.Gauge, checks] @
    [("-o", Arg.String (fun s -> output_file := Some s),
      "file            write to given file instead of /dev/stdout");
     ("-scatter",
      Arg.String (fun s -> rev_scatterings := s :: !rev_scatterings),
      "expr       in1 in2 -> out1 out2 ...");
     ("-scatter_file",
```

```
      Arg.String (fun s → rev_scatterings := read_lines_rev s @ !rev_scatterings),
      "name␣␣each␣line:␣in1␣in2␣->␣out1␣out2␣...");
    ("-decay", Arg.String (fun s → rev_decays := s :: !rev_decays),
      "expr␣␣␣␣␣␣␣␣␣in␣->␣out1␣out2␣...");
    ("-decay_file",
      Arg.String (fun s → rev_decays := read_lines_rev s @ !rev_decays),
      "name␣␣␣␣each␣line:␣in␣->␣out1␣out2␣...");
    ("-cascade", Arg.String (fun s → cascades := s :: !cascades),
      "expr␣␣␣␣␣␣␣select␣diagrams");
    ("-initialize",
      Arg.String (fun s → cache_option := Cache_Initialize s),
      "dir␣␣␣␣␣precompute␣lookup␣tables␣and␣store␣them␣in␣directory");
    ("-unphysical", Arg.Int (fun i → unphysical_polarization := Some i),
      "n␣␣␣␣␣␣␣use␣unphysical␣polarization␣for␣n-th␣particle␣/␣test␣WIs");
    ("-template", Arg.Set template,
      "␣␣␣␣␣␣␣␣␣␣write␣a␣template␣for␣handwritten␣amplitudes");
    ("-forest", Arg.Set print_forest,
      "␣␣␣␣␣␣␣␣␣␣␣␣Diagrammatic␣expansion");
    ("-diagrams", Arg.String (fun s → diagrams_sans_color := Some s),
      "file␣␣␣␣␣␣produce␣FeynMP␣output␣for␣Feynman␣diagrams");
    ("-diagrams:c", Arg.String (fun s → diagrams_color_only := Some s),
      "file␣␣␣␣produce␣FeynMP␣output␣for␣color␣flow␣diagrams");
    ("-diagrams:C", Arg.String (fun s → diagrams_all := Some s),
      "file␣␣␣␣produce␣FeynMP␣output␣for␣Feynman␣and␣color␣flow␣diagrams");
    ("-diagrams_LaTeX", Arg.Set diagrams_LaTeX,
      "␣␣␣␣enclose␣FeynMP␣output␣in␣LaTeX␣wrapper");
    ("-revision", Arg.Unit version,
      "␣␣␣␣␣␣␣␣␣␣print␣revision␣control␣information");
    ("-quiet", Arg.Set quiet,
      "␣␣␣␣␣␣␣␣␣␣␣␣␣don't␣print␣a␣summary");
    ("-summary", Arg.Clear write,
      "␣␣␣␣␣␣␣␣␣␣␣print␣only␣a␣summary");
    ("-params", Arg.Set params,
      "␣␣␣␣␣␣␣␣␣␣␣␣print␣the␣model␣parameters");
    ("-poles", Arg.Set poles,
      "␣␣␣␣␣␣␣␣␣␣␣␣␣print␣the␣Monte␣Carlo␣poles");
    ("-dag", Arg.String (fun s → dag_out := Some s),
      "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣print␣minimal␣DAG");
    ("-full_dag", Arg.String (fun s → dag0_out := Some s),
      "␣␣␣␣␣␣␣␣␣␣print␣complete␣DAG")])
  (fun _ → prerr_endline usage; exit 1)
  usage;

let cmdline =
  String.concat "␣" (List.map ThoString.quote (Array.to_list Sys.argv)) in

let output_channel =
  match !output_file with
  | None → stdout
  | Some name → open_out name in
```

```
let processes =
  try
    ThoList.uniq
      (List.sort compare
         (match List.rev !rev_scatterings, List.rev !rev_decays with
         | [], [] → []
         | scatterings, [] →
             Proc.expand_scatterings (List.map Proc.parse_scattering scatterings)
         | [], decays →
             Proc.expand_decays (List.map Proc.parse_decay decays)
         | scatterings, decays →
             invalid_arg "mixed scattering and decay!"))
  with
  | Invalid_argument s →
      begin
        Printf.eprintf "O'Mega: invalid process specification: %s!\n" s;
        flush stderr;
        []
      end in
```

This is still crude. Eventually, we want to catch *all* exceptions and write an empty (but compilable) amplitude unless one of the special options is selected.

```
begin match processes, !cache_option, !params with
| [], Cache_Initialize dir, false →
    F.initialize_cache dir;
    exit 0
| _, _, true →
    T.parameters_to_channel output_channel;
    exit 0
| [], _, false →
    T.amplitudes_to_channel cmdline output_channel !checks CF.empty;
    exit 0
| _, _, false →

  let selectors =
    let fin, fout = List.hd processes in
    C.to_selectors (C.of_string_list (List.length fin + List.length fout) !cascades) in

  let amplitudes =
    try
      begin match F.check_charges () with
      | [] → ()
      | violators →
          let violator_strings =
            String.concat ", "
              (List.map
                 (fun flist →
                   "(" ^ String.concat "," (List.map M.flavor_to_string flist) ^ ")")
                 violators) in
```

```
          failwith ("charge violating vertices: " ^ violator_strings)
      end;
      CF.amplitudes (include_goldstones !checks) !unphysical_polarization selectors processes
    with
    | exc →
        begin
          Printf.eprintf
            "O'Mega: exception %s in amplitude construction!\n"
            (Printexc.to_string exc);
          flush stderr;
          CF.empty;
        end in

  if !write then
    T.amplitudes_to_channel cmdline output_channel !checks amplitudes;

  if ¬ !quiet then begin
    List.iter
      (fun amplitude →
        Printf.eprintf "SUMMARY: %d fusions, %d propagators"
          (F.count_fusions amplitude) (F.count_propagators amplitude);
        flush stderr;
        Printf.eprintf ", %d diagrams" (F.count_diagrams amplitude);
        Printf.eprintf "\n")
      (CF.processes amplitudes);
  end;

  if !poles then begin
    List.iter
      (fun amplitude →
        W.write output_channel "omega" (W.merge (W.trees amplitude)))
      (CF.processes amplitudes)
  end;

  begin match !dag0_out with
  | Some name →
      let ch = open_out name in
      List.iter (F.tower_to_dot ch) (CF.processes amplitudes);
      close_out ch
  | None → ()
  end;

  begin match !dag_out with
  | Some name →
      let ch = open_out name in
      List.iter (F.amplitude_to_dot ch) (CF.processes amplitudes);
      close_out ch
  | None → ()
  end;

  if !print_forest then
    List.iter
      (fun amplitude →
```

```
              List.iter (fun t -> Printf.eprintf "%s\n"
                  (Tree.to_string
                     (Tree.map (fun (wf, _) -> variable wf) (fun _ ->
"") t)))
                (F.forest (List.hd (F.externals amplitude)) amplitude))
            (CF.processes amplitudes);

        begin match !diagrams_all with
        | Some name ->
          amplitudes_to_feynmf !diagrams_LaTeX name amplitudes
        | None -> ()
        end;

        begin match !diagrams_sans_color with
        | Some name ->
          amplitudes_to_feynmf_sans_color !diagrams_LaTeX name amplitudes
        | None -> ()
        end;

        begin match !diagrams_color_only with
        | Some name ->
          amplitudes_to_feynmf_color_only !diagrams_LaTeX name amplitudes
        | None -> ()
        end;

        begin match !output_file with
        | None -> ()
        | Some name -> close_out output_channel
        end;

        exit 0

      end
```

This was only intended for debugging O'Giga ...

```
    let decode wf =
      (F.flavor wf, (F.momentum wf : Momentum.Default.t))

    let diagrams in1 in2 out =
      match F.amplitudes false C.no_cascades [in1; in2] out with
      | a :: _ ->
          let wf1 = List.hd (F.externals a)
          and wf2 = List.hd (List.tl (F.externals a)) in
          let wf2 = decode wf2 in
          List.map (fun t ->
            (wf2,
             Tree.map (fun (wf, _) -> decode wf) decode t))
            (F.forest wf1 a)
      | [] -> []

    let diagrams in1 in2 out =
      failwith "Omega().diagrams: disabled"

  end
```

18.4 Implementation of Omega_QED

```
module O = Omega.Make(Fusion.Binary)(Targets.Fortran)(Modellib_SM.QED)
let _ = O.main ()
```

18.5 Implementation of Omega_SM

```
module O = Omega.Make(Fusion.Mixed23)(Targets.Fortran)
                     (Modellib_SM.SM(Modellib_SM.SM_no_anomalous))
let _ = O.main ()
```

18.6 Implementation of Omega_SYM

```
let rcs_file = RCS.parse "omega_SYM"
    ["Super␣Yang-Mills␣(incomplete,␣just␣for␣stress-testing␣Colorize.It())"]
    { RCS.revision = "$Revision:␣6465␣$";
      RCS.date = "$Date:␣2015-01-10␣16:22:31␣+0100␣(Sat,␣10␣Jan␣2015)␣$";
      RCS.author = "$Author:␣jr_reuter␣$";
      RCS.source
        = "$URL:␣svn+ssh://jr_reuter@login.hepforge.org/hepforge/svn/whizard/trunk/omega/s

module SYM =
  struct

    let rcs = rcs_file

    open Coupling

    let options = Options.empty

    let nc = 3

    type flavor =
      | Q of int | SQ of int
      | G of int | SG of int
      | Phi

    let generations = ThoList.range 1 1

    let generations_pairs =
      List.map
        (function [a; b] → (a, b)
          | _ → failwith "omega_SYM.generations_pairs")
        (Product.power 2 generations)

    let generations_triples =
      List.map
        (function [a; b; c] → (a, b, c)
          | _ → failwith "omega_SYM.generations_triples")
        (Product.power 3 generations)

    let generations_quadruples =
```

```
  List.map
    (function [a; b; c; d] → (a, b, c, d)
      | _ → failwith "omega_SYM.generations_quadruples")
    (Product.power 4 generations)

let external_flavors () =
  [ "Quarks", List.map (fun i → Q i) generations;
    "Anti-Quarks", List.map (fun i → Q (−i)) generations;
    "SQuarks", List.map (fun i → SQ i) generations;
    "Anti-SQuarks", List.map (fun i → SQ (−i)) generations;
    "Gluons", List.map (fun i → G i) generations;
    "SGluons", List.map (fun i → SG i) generations;
    "Other", [Phi]]

let flavors () =
  ThoList.flatmap snd (external_flavors ())

type gauge = unit
type constant =
  | G_saa of int × int
  | G_saaa of int × int × int
  | G3 of int × int × int
  | I_G3 of int × int × int
  | G4 of int × int × int × int

type orders = unit
let orders = function
  | _ → ()

let lorentz = function
  | Q i →
      if i > 0 then
        Spinor
      else if i < 0 then
        ConjSpinor
      else
        invalid_arg "SYM.lorentz␣(Q␣0)"
  | SQ _ | Phi → Scalar
  | G _ → Vector
  | SG _ → Majorana

let color = function
  | Q i | SQ i →
      Color.SUN (if i > 0 then nc else if i < 0 then -nc else invalid_arg "SYM.color␣(Q␣0)")
  | G _ | SG _ → Color.AdjSUN nc
  | Phi → Color.Singlet

let propagator = function
  | Q i →
      if i > 0 then
        Prop_Spinor
      else if i < 0 then
        Prop_ConjSpinor
      else
```

```
        invalid_arg "SYM.lorentz␣(Q␣0)"
  | SQ _ | Phi → Prop_Scalar
  | G _ → Prop_Feynman
  | SG _ → Prop_Majorana

let width _ = Timelike
let goldstone _ = None

let conjugate = function
  | Q i → Q (−i)
  | SQ i → SQ (−i)
  | (G _ | SG _ | Phi) as p → p

let fermion = function
  | Q i →
      if i > 0 then
        1
      else if i < 0 then
        -1
      else
        invalid_arg "SYM.fermion␣(Q␣0)"
  | SQ _ | G _ | Phi → 0
  | SG _ → 2

module Ch = Charges.Null
let charges _ = ()

module F = Modeltools.Fusions (struct
  type f = flavor
  type c = constant
  let compare = compare
  let conjugate = conjugate
end)

let quark_current =
  List.map
    (fun (i, j, k) →
      ((Q (−i), G j, Q k), FBF (−1, Psibar, V, Psi), G3 (i, j, k)))
    generations_triples

let squark_current =
  List.map
    (fun (i, j, k) →
      ((G j, SQ i, SQ (−k)), Vector_Scalar_Scalar 1, G3 (i, j, k)))
    generations_triples

let three_gluon =
  List.map
    (fun (i, j, k) →
      ((G i, G j, G k), Gauge_Gauge_Gauge 1, I_G3 (i, j, k)))
    generations_triples

let gluon2_phi =
  List.map
    (fun (i, j) →
```

```
      ((Phi, G i, G j), Dim5_Scalar_Gauge2 1, G_saa (i, j)))
    generations_pairs

let vertices3 =
  quark_current @ squark_current @ three_gluon @ gluon2_phi

let gauge4 = Vector4 [(2, C_13_42); (-1, C_12_34); (-1, C_14_23)]

let squark_seagull =
  List.map
    (fun (i, j, k, l) ->
      ((SQ i, SQ (-j), G k, G l), Scalar2_Vector2 1, G4 (i, j, k, l)))
    generations_quadruples

let four_gluon =
  List.map
    (fun (i, j, k, l) ->
      ((G i, G j, G k, G l), gauge4, G4 (i, j, k, l)))
    generations_quadruples
```

We need at least a *Dim6_Scalar_Gauge3* vertex to support this.

```
let gluon3_phi =
  []

let vertices4 =
  squark_seagull @ four_gluon @ gluon3_phi

let vertices () =
  (vertices3, vertices4, [])

let table = F.of_vertices (vertices ())
let fuse2 = F.fuse2 table
let fuse3 = F.fuse3 table
let fuse = F.fuse table
let max_degree () = 4

let parameters () = { input = []; derived = []; derived_arrays = [] }

let invalid_flavor s =
  invalid_arg ("omega_SYM.flavor_of_string: " ^ s)

let flavor_of_string s =
  let l = String.length s in
  if l < 2 then
    invalid_flavor s
  else if l = 2 then
    if String.sub s 0 1 = "q" then
      Q (int_of_string (String.sub s 1 1))
    else if String.sub s 0 1 = "Q" then
      Q (- (int_of_string (String.sub s 1 1)))
    else if String.sub s 0 1 = "g" then
      G (int_of_string (String.sub s 1 1))
    else
      invalid_flavor s
```

```
    else if l = 3 then
      if s = "phi" then
        Phi
      else if String.sub s 0 2 = "sq" then
        SQ (int_of_string (String.sub s 2 1))
      else if String.sub s 0 2 = "sQ" then
        SQ (- (int_of_string (String.sub s 2 1)))
      else if String.sub s 0 2 = "sg" then
        SG (int_of_string (String.sub s 2 1))
      else
        invalid_flavor s
    else
      invalid_flavor s

let flavor_to_string = function
  | Q i ->
      if i > 0 then
        "q" ^ string_of_int i
      else if i < 0 then
        "Q" ^ string_of_int (-i)
      else
        invalid_arg "SYM.flavor_to_string (Q 0)"
  | SQ i ->
      if i > 0 then
        "sq" ^ string_of_int i
      else if i < 0 then
        "sQ" ^ string_of_int (-i)
      else
        invalid_arg "SYM.flavor_to_string (SQ 0)"
  | G i -> "g" ^ string_of_int i
  | SG i -> "sg" ^ string_of_int i
  | Phi -> "phi"

let flavor_to_TeX = function
  | Q i ->
      if i > 0 then
        "q_{" ^ string_of_int i ^ "}"
      else if i < 0 then
        "{\bar q}_{" ^ string_of_int (-i) ^ "}"
      else
        invalid_arg "SYM.flavor_to_string (Q 0)"
  | SQ i ->
      if i > 0 then
        "{\tilde q}_{" ^ string_of_int i ^ "}"
      else if i < 0 then
        "{\bar{\tilde q}}_{" ^ string_of_int (-i) ^ "}"
      else
        invalid_arg "SYM.flavor_to_string (SQ 0)"
  | G i -> "g_{" ^ string_of_int i ^ "}"
  | SG i -> "{\tilde g}_{" ^ string_of_int i ^ "}"
  | Phi -> "phi"
```

```
let flavor_symbol = function
  | Q i ->
      if i > 0 then
        "q" ^ string_of_int i
      else if i < 0 then
        "qbar" ^ string_of_int (-i)
      else
        invalid_arg "SYM.flavor_to_string (Q 0)"
  | SQ i ->
      if i > 0 then
        "sq" ^ string_of_int i
      else if i < 0 then
        "sqbar" ^ string_of_int (-i)
      else
        invalid_arg "SYM.flavor_to_string (SQ 0)"
  | G i -> "g" ^ string_of_int i
  | SG i -> "sg" ^ string_of_int i
  | Phi -> "phi"

let gauge_symbol () =
  failwith "omega_SYM.gauge_symbol: internal error"

let pdg _ = 0
let mass_symbol _ = "0.0_default"
let width_symbol _ = "0.0_default"

let string_of_int_list int_list =
  "(" ^ String.concat "," (List.map string_of_int int_list) ^ ")"

let constant_symbol = function
  | G_saa (i, j) -> "g_saa" ^ string_of_int_list [i;j]
  | G_saaa (i, j, k) -> "g_saaa" ^ string_of_int_list [i;j;k]
  | G3 (i, j, k) -> "g3" ^ string_of_int_list [i;j;k]
  | I_G3 (i, j, k) -> "ig3" ^ string_of_int_list [i;j;k]
  | G4 (i, j, k, l) -> "g4" ^ string_of_int_list [i;j;k;l]

end

module O = Omega.Make(Fusion.Mixed23)(Targets.Fortran_Majorana)(SYM)
let _ = O.main ()
```

Acknowledgements

We thank Mauro Moretti for fruitful discussions of the ALPHA algorithm [1], that inspired our solution of the double counting problem.

We thank Wolfgang Kilian for providing the WHIZARD environment that turns our numbers into real events with unit weight. Thanks to the ECFA/DESY workshops and their participants for providing a showcase. Thanks to Edward Boos for discussions in Kaluza-Klein gravitons.

This research is supported by Bundesministerium für Bildung und Forschung, Germany, (05 HT9RDA) and Deutsche Forschungsgemeinschaft (MA 676/6-1).

Thanks to the Caml and Objective Caml teams from INRIA for the development and the lean and mean implementation of a programming language that does not insult the programmer's intelligence.

Bibliography

[1] F. Caravaglios, M. Moretti, Z. Phys. **C74** (1997) 291.

[2] A. Kanaki, C. Papadopoulos, DEMO-HEP-2000/01, hep-ph/0002082, February 2000.

[3] Xavier Leroy, *The Objective Caml system, documentation and user's guide*, Technical Report, INRIA, 1997.

[4] Chris Okasaki, *Purely Functional Data Structures*, Cambridge University Press, 1998.

[5] H. Murayama, I. Watanabe, K. Hagiwara, KEK Report 91-11, January 1992.

[6] T. Stelzer, W.F. Long, Comput. Phys. Commun. **81** (1994) 357.

[7] A. Denner, H. Eck, O. Hahn and J. Küblbeck, Phys. Lett. **B291** (1992) 278; Nucl. Phys. **B387** (1992) 467.

[8] V. Barger, A. L. Stange, R. J. N. Phillips, Phys. Rev. **D45**, (1992) 1751.

[9] T. Ohl, *Lord of the Rings*, (Computer algebra library for O'Caml, unpublished).

[10] T. Ohl, *Bocages*, (Feynman diagram library for O'Caml, unpublished).

[11] W. Kilian, *WHIZARD*, University of Karlsruhe, 2000.

[12] E. E. Boos, T. Ohl, Phys. Rev. Lett. **83** (1999) 480.

[13] T. Han, J. D. Lykken and R. Zhang, Phys. Rev. **D59** (1999) 105006 [hep-ph/9811350].

[14] William H. Press, Saul A. Teukolsky, William T. Vetterling, Brian P. Flannery, *Numerical Recipes: The Art of Scientific Computing*, Second Edition, Cambridge University Press, 1992.

[15] P. Cvitanović, Phys. Rev. **D14** (1976) 1536.

—A— Revision Control

A.1 Interface of RCS

This is a very simple library for exporting and accessing `RCS` and `CVS` revision control information. In addition, module names and short descriptions are supported as well.

If multiple applications are constructed by functors, the functions in this module can be used to identify the concrete implementations. In the context of O'Mega, this is particularly important for physics models and target languages. One structure of type *raw* has to be initialized in each file by the raw RCS keyword strings. It can remain private to the module, because it is only used as argument to the function *parse*.

type $raw = \{ revision : string; date : string; author : string; source : string \}$

Parsed revision control info:

type t

parse name description keywords initializes revision control info:

val $parse : string \rightarrow string\ list \rightarrow raw \rightarrow t$

rename rcs name description changes the name and description. This is useful if more than one module is defined in a file.

val $rename : t \rightarrow string \rightarrow string\ list \rightarrow t$

Access individual parts of the revision control information:

val $name : t \rightarrow string$
val $description : t \rightarrow string\ list$
val $revision : t \rightarrow string$
val $date : t \rightarrow string$
val $author : t \rightarrow string$

This one tries `URL` (svn), `Source` (CVS) and `Id`, in that order, for the filename.

val $source : t \rightarrow string$

Return the formatted revision control info as a list of strings suitable for printing to the terminal or embedding in the output:

val $summary : t \rightarrow string\ list$

A.2 Implementation of RCS

```
type raw = { revision : string; date : string; author : string; source : string }

type t =
    { name : string;
      description : string list;
      rcs_revision : string;
      rcs_date : string;
      rcs_author : string;
      rcs_source : string }

let name r = r.name
let description r = r.description
let revision r = r.rcs_revision
let date r = r.rcs_date
let author r = r.rcs_author
let source r = r.rcs_source

module TS = ThoString

let strip_dollars s =
  TS.strip_from_last '$' (TS.strip_prefix "$" s)

let strip_keyword k s =
  TS.strip_prefix_star ' ' (TS.strip_prefix ":" (TS.strip_required_prefix k s))

let parse1 k s =
  strip_keyword k (strip_dollars s)

let strip_before_keyword k s =
  try
    let i = TS.index_string k s in
    String.sub s i (String.length s - i)
  with
  | Not_found -> s

let strip_before_a_keyword k_list s =
  let rec strip_before_a_keyword' = function
    | k :: k_rest ->
        begin try
          let i = TS.index_string k s in
          String.sub s i (String.length s - i)
        with
        | Not_found -> strip_before_a_keyword' k_rest
        end
    | [] -> s in
  strip_before_a_keyword' k_list
```

Required for the transition from CVS to Subversion, because the latter doesn't support the `Source` keyword. `URL` is probably the way to go, but we leave in `Id` as a fallback option.

```
let parse_source s =
  let s = strip_dollars s in
```

```
  try strip_keyword "URL" s with Invalid_argument _ →
    try strip_keyword "Source" s with Invalid_argument _ →
      TS.strip_from_first ' ' (strip_keyword "Id" s)
```

Assume that the SVN repository follows the recommended layout and that all files can be found beneath "/trunk/", "/branches/" or "/tags/". Strip everything before that.

```
let strip_svn_repos s =
  strip_before_a_keyword ["/trunk/"; "/branches/"; "/tags/"] s

let parse name description r =
  { name = name;
    description = description;
    rcs_revision = parse1 "Revision" r.revision;
    rcs_date = parse1 "Date" r.date;
    rcs_author = parse1 "Author" r.author;
    rcs_source = strip_svn_repos (parse_source r.source) }

let rename rcs name description =
  { rcs with name = name; description = description }

let summary rcs =
  [ name rcs ^ ":"] @
  List.map (fun s → "␣␣" ^ s) (description rcs) @
  [ "␣␣Source:␣" ^ source rcs;
    "␣␣revision:␣" ^ revision rcs ^ "␣checked␣in␣by␣" ^
    author rcs ^ "␣at␣" ^ date rcs ]
```

—B—
Autotools

B.1 Interface of Config

Cache writing is attempted in the order [*system_cache_dir*], [*user_cache_dir*], ["."] and cache reading in the opposite order.

```
val system_cache_dir : string
val user_cache_dir : string
```

```
val cache_prefix : string
val cache_suffix : string
```

```
val openmp : bool
```

B.2 Implementation of Config

```
let system_cache_dir = "/Users/reuter/local/var/cache"
let user_cache_dir = "/Users/reuter/.whizard/var/cache"
```

This relies on the fast that the executable names should be unique.

```
let cache_prefix =
  let basename = Filename.basename Sys.executable_name in
  try Filename.chop_extension basename with | _ → basename
```

```
let cache_suffix = "vertices"
```

```
let openmp = false
```

—C— Textual Options

C.1 Interface of Options

```
type t
val empty : t
val merge : t → t → t
val extend : t → (string × Arg.spec × string) list → t
val create : (string × Arg.spec × string) list → t
val parse : t → string × string → unit
val list : t → (string × string) list
val cmdline : string → t → (string × Arg.spec × string) list

exception Invalid of string × string
```

C.2 Implementation of Options

```
module A = Map.Make (struct type t = string let compare = compare end)

type t =
    { actions : Arg.spec A.t;
      raw : (string × Arg.spec × string) list }

let empty = { actions = A.empty; raw = [] }

let extend old options =
  { actions = List.fold_left
      (fun a (s, f, _) → A.add s f a) old.actions options;
    raw = options @ old.raw }

let create = extend empty

let merge o1 o2 =
  extend o1 o2.raw

exception Invalid of string × string

let parse options (name, value) =
  try
    match A.find name options.actions with
    | Arg.Unit f → f ()
```

```
    | Arg.Set b  →  b  := true
    | Arg.Clear b  →  b  := false
    | Arg.String f  →  f value
    | Arg.Int f  →  f (int_of_string value)
    | Arg.Float f  →  f (float_of_string value)
    | _  →  invalid_arg "Options.parse"
  with
  | Not_found  →  raise (Invalid (name, value))

let list options =
  List.map (fun (o, _, d) → (o, d)) options.raw

let cmdline prefix options =
  List.map (fun (o, f, d) → (prefix ^ o, f, d)) options.raw
```

—D—
Progress Reports

D.1 Interface of Progress

```
type t

val dummy : t
val channel : out_channel → int → t
val file : string → int → t
val open_file : string → int → t
val reset : t → int → string → unit
val begin_step : t → string → unit
val end_step : t → string → unit
val summary : t → string → unit
```

D.2 Implementation of Progress

```
type channel =
  | Channel of out_channel
  | File of string
  | Open_File of string × out_channel

type state =
    { channel : channel;
      mutable steps : int;
      mutable digits : int;
      mutable step : int;
      created : float;
      mutable last_reset : float;
      mutable last_begin : float; }

type t = state option

let digits n =
  if n > 0 then
    succ (truncate (log10 (float n)))
  else
    invalid_arg "Progress.digits: non-positive argument"

let mod_float2 a b =
```

```
  let modulus = mod_float a b in
  ((a -. modulus) /. b, modulus)

let time_to_string seconds =
  let minutes, seconds = mod_float2 seconds 60. in
  if minutes > 0.0 then
    let hours, minutes = mod_float2 minutes 60. in
    if hours > 0.0 then
      let days, hours = mod_float2 hours 24. in
      if days > 0.0 then
        Printf.sprintf "%.0f:%02.0f days" days hours
      else
        Printf.sprintf "%.0f:%02.0f hrs" hours minutes
    else
      Printf.sprintf "%.0f:%02.0f mins" minutes seconds
  else
    Printf.sprintf "%.2f secs" seconds

let create channel steps =
  let now = Sys.time () in
  Some { channel = channel;
         steps = steps;
         digits = digits steps;
         step = 0;
         created = now;
         last_reset = now;
         last_begin = now }

let dummy =
  None

let channel oc =
  create (Channel oc)

let file name =
  let oc = open_out name in
  close_out oc;
  create (File name)

let open_file name =
  let oc = open_out name in
  create (Open_File (name, oc))

let close_channel state =
  match state.channel with
  | Channel oc ->
      flush oc
  | File _ -> ()
  | Open_File (_, oc) ->
      flush oc;
      close_out oc

let use_channel state f =
  match state.channel with
```

```
  | Channel oc | Open_File (_, oc) →
      f oc;
      flush oc
  | File name →
      let oc = open_out_gen [Open_append; Open_creat] 644₈ name in
      f oc;
      flush oc;
      close_out oc

let reset state steps msg =
  match state with
  | None → ()
  | Some state →
      let now = Sys.time () in
      state.steps ← steps;
      state.digits ← digits steps;
      state.step ← 0;
      state.last_reset ← now;
      state.last_begin ← now

let begin_step state msg =
  match state with
  | None → ()
  | Some state →
      let now = Sys.time () in
      state.step ← succ state.step;
      state.last_begin ← now;
      use_channel state (fun oc →
        Printf.fprintf oc "[%0*d/%0*d]␣%s␣..." state.digits state.step state.digits state.steps msg)

let end_step state msg =
  match state with
  | None → ()
  | Some state →
      let now = Sys.time () in
      let last = now −. state.last_begin in
      let elapsed = now −. state.last_reset in
      let estimated = float state.steps ∗. elapsed /. float state.step in
      let remaining = estimated −. elapsed in
      use_channel state (fun oc →
        Printf.fprintf oc "␣%s.␣[time:␣%s,␣total:␣%s,␣remaining:␣%s]\n" msg
          (time_to_string last) (time_to_string estimated) (time_to_string remaining))

let summary state msg =
  match state with
  | None → ()
  | Some state →
      let now = Sys.time () in
      use_channel state (fun oc →
        Printf.fprintf oc "%s.␣[total␣time:␣%s]\n" msg
          (time_to_string (now −. state.created)));
      close_channel state
```

—E—
More on Filenames

E.1 Interface of ThoFilename

```
val split : string → string list
val join : string list → string

val expand_home : string → string
```

E.2 Implementation of ThoFilename

```
let rec split' acc path =
  match Filename.dirname path, Filename.basename path with
  | "/", basename → "/" :: basename :: acc
  | ".", basename → basename :: acc
  | dirname, basename → split' (basename :: acc) dirname

let split path =
  split' [] path

let join = function
  | [] → "."
  | [basename] → basename
  | dirname :: rest → List.fold_left Filename.concat dirname rest

let expand_home path =
  match split path with
  | ("~" | "$HOME" | "${HOME}") :: rest →
      join ((try Sys.getenv "HOME" with Not_found → "/tmp") :: rest)
  | _ → path
```

—F—
Cache Files

F.1 Interface of Cache

module type T =
  sig

    type key
    type $hash$ = $string$
    type $value$

    type α $result$ =
      | Hit of α
      | $Miss$
      | $Stale$ of $string$

    exception $Mismatch$ of $string \times string \times string$

    val $hash$: $key \rightarrow hash$
    val $exists$: $hash \rightarrow string \rightarrow bool$
    val $find$: $hash \rightarrow string \rightarrow string\ option$
    val $write$: $hash \rightarrow string \rightarrow value \rightarrow unit$
    val $write\_dir$: $hash \rightarrow string \rightarrow string \rightarrow value \rightarrow unit$
    val $read$: $hash \rightarrow string \rightarrow value$
    val $maybe\_read$: $hash \rightarrow string \rightarrow value\ result$

  end

module type Key =
  sig
    type t
  end

module type $Value$ =
  sig
    type t
  end

module $Make$ (Key : Key) ($Value$: $Value$) :
    T with type key = $Key.t$ and type $value$ = $Value.t$

F.2 Implementation of Cache

```
let search_path =
  [ Filename.current_dir_name;
    ThoFilename.expand_home Config.user_cache_dir;
    Config.system_cache_dir ]

module type T =
  sig

    type key
    type hash = string
    type value

    type α result =
      | Hit of α
      | Miss
      | Stale of string

    exception Mismatch of string × string × string

    val hash : key → hash
    val exists : hash → string → bool
    val find : hash → string → string option
    val write : hash → string → value → unit
    val write_dir : hash → string → string → value → unit
    val read : hash → string → value
    val maybe_read : hash → string → value result

  end

module type Key =
  sig
    type t
  end

module type Value =
  sig
    type t
  end

module Make (Key : Key) (Value : Value) =
  struct

    type key = Key.t
    type hash = string
    type value = Value.t

    type tagged =
        { tag : hash;
          value : value; }

    let hash value =
      Digest.string (Marshal.to_string value [])

    let find_first path name =
```

```
    let rec find_first' = function
      | [] -> raise Not_found
      | dir :: path ->
          let f = Filename.concat dir name in
          if Sys.file_exists f then
            f
          else
            find_first' path
    in
    find_first' path

let find hash name =
    try Some (find_first search_path name) with Not_found -> None

let exists hash name =
    match find hash name with
    | None -> false
    | Some _ -> true

let try_first f path name =
    let rec try_first' = function
      | [] -> raise Not_found
      | dir :: path ->
          try (f (Filename.concat dir name), dir) with _ -> try_first' path
    in
    try_first' path

let open_in_bin_first = try_first open_in_bin
let open_out_bin_last path = try_first open_out_bin (List.rev path)

let write hash name value =
    let oc, _ = open_out_bin_last search_path name in
    Marshal.to_channel oc { tag = hash; value = value } [];
    close_out oc

let write_dir hash dir name value =
    let oc = open_out_bin (Filename.concat dir name) in
    Marshal.to_channel oc { tag = hash; value = value } [];
    close_out oc

type α result =
    | Hit of α
    | Miss
    | Stale of string

exception Mismatch of string × string × string

let read hash name =
    let ic, dir = open_in_bin_first search_path name in
    let { tag = tag; value = value } = Marshal.from_channel ic in
    close_in ic;
    if tag = hash then
      value
    else
      raise (Mismatch (Filename.concat dir name, hash, tag))
```

```
  let maybe_read hash name =
    try
      Hit (read hash name)
    with
    | Not_found -> Miss
    | Mismatch (file, _, _) -> Stale file
end
```

—G—
More On Lists

G.1 Interface of ThoList

splitn n l $=$ $(hdn\ l,\ tln\ l)$, but more efficient.

val hdn : $int \rightarrow \alpha\ list \rightarrow \alpha\ list$
val tln : $int \rightarrow \alpha\ list \rightarrow \alpha\ list$
val $splitn$: $int \rightarrow \alpha\ list \rightarrow \alpha\ list \times \alpha\ list$

chop n l chops l into pieces of size n (except for the last one, which contains th remainder).

val $chopn$: $int \rightarrow \alpha\ list \rightarrow \alpha\ list\ list$

of_subarray n m a is $[a.(n); a.(n+1); \ldots; a.(m)]$. Values of n and m out of bounds are silently shifted towards these bounds.

val $of\_subarray$: $int \rightarrow int \rightarrow \alpha\ array \rightarrow \alpha\ list$

range s n m is $[n; n+s; n+2s; \ldots; m-((m-n) \mod s)]$

val $range$: $?stride : int \rightarrow int \rightarrow int \rightarrow int\ list$

enumerate s n $[a1; a2; ...]$ *is* $[(n, a1);\ (n+s, a2);\ ...]$

val $enumerate$: $?stride : int \rightarrow int \rightarrow \alpha\ list \rightarrow (int \times \alpha)\ list$

Compress identical elements in a sorted list. Identity is determined using the polymorphic equality function $Pervasives.(=)$.

val $uniq$: $\alpha\ list \rightarrow \alpha\ list$

Test if all members of a list are structurally identical (actually *homogeneous* l and $List.length\ (uniq\ l) \leq 1$ are equivalent, but the former is more efficient if a mismatch comes early).

val $homogeneous$: $\alpha\ list \rightarrow bool$

compare cmp $l1$ $l2$ compare two lists $l1$ and $l2$ according to cmp. cmp defaults to the polymorphic $Pervasives.compare$.

val $compare$: $?cmp : (\alpha \rightarrow \alpha \rightarrow int) \rightarrow \alpha\ list \rightarrow \alpha\ list \rightarrow int$

Collect and count identical elements in a list. Identity is determined using the polymorphic equality function $Pervasives.(=)$. *classify* does not assume that the list is sorted. However, it is $O(n)$ for sorted lists and $O(n^2)$ in the worst case.

val $classify$: $\alpha\ list \rightarrow (int \times \alpha)\ list$

Collect the second factors with a common first factor in lists.

val $factorize$: $(\alpha \times \beta)\ list \rightarrow (\alpha \times \beta\ list)\ list$

flatmap f is equivalent to *flatten* $\circ$ (*map* f), but more efficient, because no intermediate lists are built. Unfortunately, it is not tail recursive.

val $flatmap$: $(\alpha \rightarrow \beta\ list) \rightarrow \alpha\ list \rightarrow \beta\ list$

rev_flatmap f is equivalent to *flatten* $\circ$ (*rev_map* (*rev* $\circ f$)) = *rev* $\circ$ (*flatmap* f), but more efficient, because no intermediate lists are built. It is tail recursive.

val $rev\_flatmap$: $(\alpha \rightarrow \beta\ list) \rightarrow \alpha\ list \rightarrow \beta\ list$

val $clone$: $int \rightarrow \alpha \rightarrow \alpha\ list$
val $multiply$: $int \rightarrow \alpha\ list \rightarrow \alpha\ list$

Invent other names to avoid confusions with *List.fold_left2* and *List.fold_right2*.

val $fold\_right2$: $(\alpha \rightarrow \beta \rightarrow \beta) \rightarrow \alpha\ list\ list \rightarrow \beta \rightarrow \beta$
val $fold\_left2$: $(\beta \rightarrow \alpha \rightarrow \beta) \rightarrow \beta \rightarrow \alpha\ list\ list \rightarrow \beta$

iteri $f\ n\ [a; b; c]$ evaluates $f\ n\ a$, $f\ (n+1)\ b$ and $f\ (n+2)\ c$.

val $iteri$: $(int \rightarrow \alpha \rightarrow unit) \rightarrow int \rightarrow \alpha\ list \rightarrow unit$
val $mapi$: $(int \rightarrow \alpha \rightarrow \beta) \rightarrow int \rightarrow \alpha\ list \rightarrow \beta\ list$

iteri2 $f\ n\ m\ [[aa; ab]; [ba; bb]]$ evaluates $f\ n\ m\ aa$, $f\ n\ (m+1)\ ab$, $f\ (n+1)\ m\ ba$ and $f\ (n+1)\ (m+1)\ bb$. NB: the nested lists need not be rectangular.

val $iteri2$: $(int \rightarrow int \rightarrow \alpha \rightarrow unit) \rightarrow int \rightarrow int \rightarrow \alpha\ list\ list \rightarrow unit$

Transpose a *rectangular* list of lists like a matrix.

val $transpose$: $\alpha\ list\ list \rightarrow \alpha\ list\ list$

interleave f *list* walks through *list* and inserts the result of f applied to the reversed list of elements before and the list of elements after. The empty lists at the beginning and end are included!

val $interleave$: $(\alpha\ list \rightarrow \alpha\ list \rightarrow \alpha\ list) \rightarrow \alpha\ list \rightarrow \alpha\ list$

interleave_nearest f *list* is like *interleave* f *list*, but f looks only at the nearest neighbors.

val $interleave\_nearest$: $(\alpha \rightarrow \alpha \rightarrow \alpha\ list) \rightarrow \alpha\ list \rightarrow \alpha\ list$

partitioned_sort cmp index_sets list sorts the sublists of *list* specified by the *index_sets* and the complement of their union. **NB:** the sorting follows to order in the lists in *index_sets*. **NB:** the indices are 0-based.

val $partitioned\_sort$: $(\alpha \rightarrow \alpha \rightarrow int) \rightarrow int\ list\ list \rightarrow \alpha\ list \rightarrow \alpha\ list$
exception *Overlapping_indices*
exception *Out_of_bounds*

ariadne_sort cmp list sorts *list* according to *cmp* (default *Pervasives.compare*) keeping track of the original order by a 0-based list of indices.

val *ariadne_sort* : $?cmp : (\alpha \rightarrow \alpha \rightarrow int) \rightarrow \alpha\ list \rightarrow \alpha\ list \times int\ list$

ariadne_unsort (*ariadne_sort cmp list*) returns *list*.

val *ariadne_unsort* : $\alpha\ list \times int\ list \rightarrow \alpha\ list$

G.2 Implementation of ThoList

```
let rec hdn n l =
  if n ≤ 0 then
    []
  else
    match l with
    | x :: rest → x :: hdn (pred n) rest
    | [] → invalid_arg "ThoList.hdn"

let rec tln n l =
  if n ≤ 0 then
    l
  else
    match l with
    | _ :: rest → tln (pred n) rest
    | [] → invalid_arg "ThoList.tln"

let rec splitn' n l1_rev l2 =
  if n ≤ 0 then
    (List.rev l1_rev, l2)
  else
    match l2 with
    | x :: l2' → splitn' (pred n) (x :: l1_rev) l2'
    | [] → invalid_arg "ThoList.splitn n > len"

let splitn n l =
  if n < 0 then
    invalid_arg "ThoList.splitn n < 0"
  else
    splitn' n [] l
```

This is *splitn'* all over again, but without the exception.

```
let rec chopn'' n l1_rev l2 =
  if n ≤ 0 then
    (List.rev l1_rev, l2)
  else
    match l2 with
    | x :: l2' → chopn'' (pred n) (x :: l1_rev) l2'
    | [] → (List.rev l1_rev, [])

let rec chopn' n ll_rev = function
  | [] → List.rev ll_rev
```

```
  | l ->
      begin match chopn'' n [] l with
      | [], [] -> List.rev ll_rev
      | l1, [] -> List.rev (l1 :: ll_rev)
      | l1, l2 -> chopn' n (l1 :: ll_rev) l2
      end

let chopn n l =
  if n <= 0 then
    invalid_arg "ThoList.chopn␣n␣<=␣0"
  else
    chopn' n [] l

let of_subarray n1 n2 a =
  let rec of_subarray' n1 n2 =
    if n1 > n2 then
      []
    else
      a.(n1) :: of_subarray' (succ n1) n2 in
  of_subarray' (max 0 n1) (min n2 (pred (Array.length a)))

let range ?(stride = 1) n1 n2 =
  if stride <= 0 then
    invalid_arg "ThoList.range:␣stride␣<=␣0"
  else
    let rec range' n =
      if n > n2 then
        []
      else
        n :: range' (n + stride) in
    range' n1
```

Tail recursive:

```
let enumerate ?(stride = 1) n l =
  let _, l_rev =
    List.fold_left
      (fun (i, acc) a -> (i + stride, (i, a) :: acc))
      (n, []) l in
  List.rev l_rev
```

This is *not* tail recursive!

```
let rec flatmap f = function
  | [] -> []
  | x :: rest -> f x @ flatmap f rest
```

This is!

```
let rev_flatmap f l =
  let rec rev_flatmap' acc f = function
    | [] -> acc
    | x :: rest -> rev_flatmap' (List.rev_append (f x) acc) f rest in
  rev_flatmap' [] f l
```

```
let fold_left2 f acc lists =
  List.fold_left (List.fold_left f) acc lists

let fold_right2 f lists acc =
  List.fold_right (List.fold_right f) lists acc

let iteri f start list =
  ignore (List.fold_left (fun i a → f i a; succ i) start list)

let iteri2 f start_outer star_inner lists =
  iteri (fun j → iteri (f j) star_inner) start_outer lists

let mapi f start list =
  let next, list' =
    List.fold_left (fun (i, acc) a → (succ i, f i a :: acc)) (start, []) list in
  List.rev list'
```

Is there a more efficient implementation?

```
let transpose lists =
  let rec transpose' rest =
    if List.for_all ((=) []) rest then
      []
    else
      List.map List.hd rest :: transpose' (List.map List.tl rest) in
  try
    transpose' lists
  with
  | Failure "tl" → invalid_arg "ThoList.transpose: not rectangular"

let compare ?(cmp = Pervasives.compare) l1 l2 =
  let rec compare' l1' l2' =
    match l1', l2' with
    | [], [] → 0
    | [], _ → -1
    | _, [] → 1
    | n1 :: r1, n2 :: r2 →
        let c = cmp n1 n2 in
        if c ≠ 0 then
          c
        else
          compare' r1 r2
  in
  compare' l1 l2

let rec uniq' x = function
  | [] → []
  | x' :: rest →
      if x' = x then
        uniq' x rest
      else
        x' :: uniq' x' rest

let uniq = function
  | [] → []
```

```
  | x :: rest -> x :: uniq' x rest

let rec homogeneous = function
  | [] | [_] -> true
  | a1 :: (a2 :: _ as rest) ->
      if a1 <> a2 then
        false
      else
        homogeneous rest
```

If we needed it, we could use a polymorphic version of *Set* to speed things up from $O(n^2)$ to $O(n \ln n)$. But not before it matters somewhere ...

```
let classify l =
  let rec add_to_class a = function
    | [] -> [1, a]
    | (n, a') :: rest ->
        if a = a' then
          (succ n, a) :: rest
        else
          (n, a') :: add_to_class a rest
  in
  let rec classify' cl = function
    | [] -> cl
    | a :: rest -> classify' (add_to_class a cl) rest
  in
  classify' [] l

let rec factorize l =
  let rec add_to_class x y = function
    | [] -> [(x, [y])]
    | (x', ys) :: rest ->
        if x = x' then
          (x, y :: ys) :: rest
        else
          (x', ys) :: add_to_class x y rest
  in
  let rec factorize' fl = function
    | [] -> fl
    | (x, y) :: rest -> factorize' (add_to_class x y fl) rest
  in
  List.map (fun (x, ys) -> (x, List.rev ys)) (factorize' [] l)

let rec clone n x =
  if n < 0 then
    invalid_arg "ThoList.clone"
  else if n = 0 then
    []
  else
    x :: clone (pred n) x

let interleave f list =
  let rec interleave' rev_head tail =
```

```
    let rev_head' = List.rev_append (f rev_head tail) rev_head in
    match tail with
    | [] -> List.rev rev_head'
    | x :: tail' -> interleave' (x :: rev_head') tail'
  in
  interleave' [] list

let interleave_nearest f list =
  interleave
    (fun head tail ->
      match head, tail with
      | h :: _, t :: _ -> f h t
      | _ -> [])
    list

let rec rev_multiply n rl l =
  if n < 0 then
    invalid_arg "ThoList.multiply"
  else if n = 0 then
    []
  else
    List.rev_append rl (rev_multiply (pred n) rl l)

let multiply n l = rev_multiply n (List.rev l) l

module ISet = Set.Make (struct type t = int let compare = Pervasives.compare end)

exception Overlapping_indices
exception Out_of_bounds

let iset_of_list list =
  List.fold_right ISet.add list ISet.empty

let iset_list_union list =
  List.fold_right ISet.union list ISet.empty

let complement_index_sets n index_set_lists =
  let index_sets = List.map iset_of_list index_set_lists in
  let index_set = iset_list_union index_sets in
  let size_index_sets =
    List.fold_left (fun acc s -> ISet.cardinal s + acc) 0 index_sets in
  if size_index_sets <> ISet.cardinal index_set then
    raise Overlapping_indices
  else if ISet.exists (fun i -> i < 0 || i >= n) index_set then
    raise Overlapping_indices
  else
    match ISet.elements (ISet.diff (iset_of_list (range 0 (pred n))) index_set) with
    | [] -> index_set_lists
    | complement -> complement :: index_set_lists

let sort_section cmp array index_set =
  List.iter2
    (Array.set array)
    index_set (List.sort cmp (List.map (Array.get array) index_set))
```

```
let partitioned_sort cmp index_sets list =
  let array = Array.of_list list in
  List.fold_left
    (fun () → sort_section cmp array)
    () (complement_index_sets (List.length list) index_sets);
  Array.to_list array

let ariadne_sort ?(cmp = Pervasives.compare) list =
  let sorted =
    List.sort (fun (n1, a1) (n2, a2) → cmp a1 a2) (enumerate 0 list) in
  (List.map snd sorted, List.map fst sorted)

let ariadne_unsort (sorted, indices) =
  List.map snd
    (List.sort
       (fun (n1, a1) (n2, a2) → Pervasives.compare n1 n2)
       (List.map2 (fun n a → (n, a)) indices sorted))
```

—H— More On Arrays

H.1 Interface of ThoArray

Compressed arrays, i.e. arrays with only unique elements and an embedding that allows to recover the original array. NB: in the current implementation, compressing saves space, if *and only if* objects of type α require more storage than integers. The main use of α *compressed* is *not* for saving space, anyway, but for avoiding the repetition of hard calculations.

type α *compressed*
val *uniq* : α *compressed* $\rightarrow$ α *array*
val *embedding* : α *compressed* $\rightarrow$ *int array*

These two are inverses of each other:

val *compress* : α *array* $\rightarrow$ α *compressed*
val *uncompress* : α *compressed* $\rightarrow$ α *array*

One can play the same game for matrices.

type α *compressed2*
val *uniq2* : α *compressed2* $\rightarrow$ α *array array*
val *embedding1* : α *compressed2* $\rightarrow$ *int array*
val *embedding2* : α *compressed2* $\rightarrow$ *int array*

Again, these two are inverses of each other:

val *compress2* : α *array array* $\rightarrow$ α *compressed2*
val *uncompress2* : α *compressed2* $\rightarrow$ α *array array*

H.2 Implementation of ThoArray

type α *compressed* =
  { *uniq* : α *array*;
    embedding : *int array* }

let *uniq* a = $a.uniq$
let *embedding* a = $a.embedding$

type α *compressed2* =
  { *uniq2* : α *array array*;

```
      embedding1 : int array;
      embedding2 : int array }

let uniq2 a = a.uniq2
let embedding1 a = a.embedding1
let embedding2 a = a.embedding2

module PMap = Pmap.Tree

let compress a =
  let last = Array.length a - 1 in
  let embedding = Array.make (succ last) (-1) in
  let rec scan num_uniq uniq elements n =
    if n > last then
      { uniq = Array.of_list (List.rev elements);
        embedding = embedding }
    else
      match PMap.find_opt compare a.(n) uniq with
      | Some n' ->
          embedding.(n) <- n';
          scan num_uniq uniq elements (succ n)
      | None ->
          embedding.(n) <- num_uniq;
          scan
            (succ num_uniq)
            (PMap.add compare a.(n) num_uniq uniq)
            (a.(n) :: elements)
            (succ n) in
  scan 0 PMap.empty [] 0

let uncompress a =
  Array.map (Array.get a.uniq) a.embedding
```

Using *transpose* simplifies the algorithms, but can be inefficient. If this turns out to be the case, we should add special treatments for symmetric matrices.

```
let transpose a =
  let dim1 = Array.length a
  and dim2 = Array.length a.(0) in
  let a' = Array.make_matrix dim2 dim1 a.(0).(0) in
  for i1 = 0 to pred dim1 do
    for i2 = 0 to pred dim2 do
      a'.(i2).(i1) <- a.(i1).(i2)
    done
  done;
  a'

let compress2 a =
  let c2 = compress a in
  let c12_transposed = compress (transpose c2.uniq) in
  { uniq2 = transpose c12_transposed.uniq;
    embedding1 = c12_transposed.embedding;
```

$$embedding2 \;=\; c2.embedding \;\}$$

let $uncompress2 \; a \;=$
  let $a2 \;=\; uncompress \; \{ \; uniq \;=\; a.uniq2; \; embedding \;=\; a.embedding2 \; \}$ in
  $transpose \, (uncompress \, \{ \; uniq \;=\; transpose \; a2; \; embedding \;=\; a.embedding1 \; \})$

—I—
More On Strings

I.1 Interface of ThoString

This is a very simple library if stroing manipulation functions missing in O'Caml's standard library.
strip_prefix prefix string returns *string* with 0 or 1 occurences of a leading *prefix* removed.

```
val strip_prefix : string → string → string
```

strip_prefix_star prefix string returns *string* with any number of leading occurences of *prefix* removed.

```
val strip_prefix_star : char → string → string
```

strip_prefix prefix string returns *string* with a leading *prefix* removed, raises *Invalid_argument* if there's no match.

```
val strip_required_prefix : string → string → string
```

strip_from_first c s returns *s* with everything starting from the first *c* removed. *strip_from_last c s* returns *s* with everything starting from the last *c* removed.

```
val strip_from_first : char → string → string
val strip_from_last : char → string → string
```

index_string pattern string returns the index of the first occurence of *pattern* in *string*, if any. Raises *Not_found*, if *pattern* is not in *string*.

```
val index_string : string → string → int
```

This silently fails if the argument contains both single and double quotes!

```
val quote : string → string
```

I.2 Implementation of ThoString

```
let strip_prefix p s =
  let lp = String.length p
  and ls = String.length s in
  if lp > ls then
    s
```

```
  else
    let rec strip_prefix' i =
      if i ≥ lp then
        String.sub s i (ls − i)
      else if p.[i] ≠ s.[i] then
        s
      else
        strip_prefix' (succ i)
    in
    strip_prefix' 0

let strip_prefix_star p s =
  let ls = String.length s in
  if ls < 1 then
    s
  else
    let rec strip_prefix_star' i =
      if i < ls then begin
        if p ≠ s.[i] then
          String.sub s i (ls − i)
        else
          strip_prefix_star' (succ i)
      end else
        ""
    in
    strip_prefix_star' 0

let strip_required_prefix p s =
  let lp = String.length p
  and ls = String.length s in
  if lp > ls then
    invalid_arg ("strip_required_prefix: expected ‘" ^ p ^ "’ got ‘" ^ s ^ "’")
  else
    let rec strip_prefix' i =
      if i ≥ lp then
        String.sub s i (ls − i)
      else if p.[i] ≠ s.[i] then
        invalid_arg ("strip_required_prefix: expected ‘" ^ p ^ "’ got ‘" ^ s ^ "’")
      else
        strip_prefix' (succ i)
    in
    strip_prefix' 0

let strip_from_first c s =
  try
    String.sub s 0 (String.index s c)
  with
  | Not_found → s

let strip_from_last c s =
  try
    String.sub s 0 (String.rindex s c)
```

```
  with
  | Not_found → s

let index_string pat s =
  let lpat = String.length pat
  and ls = String.length s in
  if lpat = 0 then
    0
  else
    let rec index_string' n =
      let i = String.index_from s n pat.[0] in
      if i + lpat > ls then
        raise Not_found
      else
        if String.compare pat (String.sub s i lpat) = 0 then
          i
        else
          index_string' (succ i)
    in
    index_string' 0

let quote s =
  if String.contains s ' ' ∨ String.contains s '\n' then begin
    if String.contains s '"' then
      "'" ^ s ^ "'"
    else
      "\"" ^ s ^ "\""
  end else
    s
```

—J—
Polymorphic Maps

From [9].

J.1 Interface of Pmap

Module *Pmap*: association tables over a polymorphic type[1].

```
module type T =
  sig
    type ('key, α) t
    val empty : ('key, α) t
    val is_empty : ('key, α) t → bool
    val singleton : 'key → α → ('key, α) t
    val add : ('key → 'key → int) → 'key → α → ('key, α) t →
 ('key, α) t
    val update : ('key → 'key → int) → (α → α → α) →
      'key → α → ('key, α) t → ('key, α) t
    val cons : ('key → 'key → int) → (α → α → α option) →
      'key → α → ('key, α) t → ('key, α) t
    val find : ('key → 'key → int) → 'key → ('key, α) t → α
    val find_opt : ('key → 'key → int) → 'key → ('key, α) t → α option
    val choose : ('key, α) t → 'key × α
    val choose_opt : ('key, α) t → ('key × α) option
    val uncons : ('key, α) t → 'key × α × ('key, α) t
    val uncons_opt : ('key, α) t → ('key × α × ('key, α) t) option
    val elements : ('key, α) t → ('key × α) list
    val mem : ('key → 'key → int) → 'key → ('key, α) t → bool
    val remove : ('key → 'key → int) → 'key → ('key, α) t → ('key, α) t
    val union : ('key → 'key → int) → (α → α → α) →
      ('key, α) t → ('key, α) t → ('key, α) t
    val compose : ('key → 'key → int) → (α → α → α option) →
      ('key, α) t → ('key, α) t → ('key, α) t
    val iter : ('key → α → unit) → ('key, α) t → unit
    val map : (α → β) → ('key, α) t → ('key, β) t
    val mapi : ('key → α → β) → ('key, α) t → ('key, β) t
    val fold : ('key → α → β → β) → ('key, α) t → β → β
```

[1]Extension of code © 1996 by Xavier Leroy

```
    val compare : ('key → 'key → int) → (α → α → int) →
      ('key, α) t → ('key, α) t → int
    val canonicalize : ('key → 'key → int) → ('key, α) t → ('key, α) t
  end
```

Balanced trees: logarithmic access, but representation not unique.

```
module Tree : T
```

Sorted lists: representation unique, but linear access.

```
module List : T
```

J.2 Implementation of Pmap

```
module type T =
  sig
    type ('key, α) t
    val empty : ('key, α) t
    val is_empty : ('key, α) t → bool
    val singleton : 'key → α → ('key, α) t
    val add : ('key → 'key → int) → 'key → α → ('key, α) t →
('key, α) t
    val update : ('key → 'key → int) → (α → α → α) →
      'key → α → ('key, α) t → ('key, α) t
    val cons : ('key → 'key → int) → (α → α → α option) →
      'key → α → ('key, α) t → ('key, α) t
    val find : ('key → 'key → int) → 'key → ('key, α) t → α
    val find_opt : ('key → 'key → int) → 'key → ('key, α) t → α option
    val choose : ('key, α) t → 'key × α
    val choose_opt : ('key, α) t → ('key × α) option
    val uncons : ('key, α) t → 'key × α × ('key, α) t
    val uncons_opt : ('key, α) t → ('key × α × ('key, α) t) option
    val elements : ('key, α) t → ('key × α) list
    val mem : ('key → 'key → int) → 'key → ('key, α) t → bool
    val remove : ('key → 'key → int) → 'key → ('key, α) t → ('key, α) t
    val union : ('key → 'key → int) → (α → α → α) →
      ('key, α) t → ('key, α) t → ('key, α) t
    val compose : ('key → 'key → int) → (α → α → α option) →
      ('key, α) t → ('key, α) t → ('key, α) t
    val iter : ('key → α → unit) → ('key, α) t → unit
    val map : (α → β) → ('key, α) t → ('key, β) t
    val mapi : ('key → α → β) → ('key, α) t → ('key, β) t
    val fold : ('key → α → β → β) → ('key, α) t → β → β
    val compare : ('key → 'key → int) → (α → α → int) →
      ('key, α) t → ('key, α) t → int
    val canonicalize : ('key → 'key → int) → ('key, α) t → ('key, α) t
  end

module Tree =
  struct
```

```
type ('key, α) t =
  | Empty
  | Node of ('key, α) t × 'key × α × ('key, α) t × int

let empty = Empty

let is_empty = function
  | Empty → true
  | _ → false

let singleton k d =
  Node (Empty, k, d, Empty, 1)

let height = function
  | Empty → 0
  | Node (_, _, _, _, h) → h

let create l x d r =
  let hl = height l and hr = height r in
  Node (l, x, d, r, (if hl ≥ hr then hl + 1 else hr + 1))

let bal l x d r =
  let hl = match l with Empty → 0 | Node (_, _, _, _, h) → h in
  let hr = match r with Empty → 0 | Node (_, _, _, _, h) → h in
  if hl > hr + 2 then begin
    match l with
    | Empty → invalid_arg "Map.bal"
    | Node (ll, lv, ld, lr, _) →
        if height ll ≥ height lr then
          create ll lv ld (create lr x d r)
        else begin
          match lr with
          | Empty → invalid_arg "Map.bal"
          | Node (lrl, lrv, lrd, lrr, _) →
              create (create ll lv ld lrl) lrv lrd (create lrr x d r)
        end
  end else if hr > hl + 2 then begin
    match r with
    | Empty → invalid_arg "Map.bal"
    | Node (rl, rv, rd, rr, _) →
        if height rr ≥ height rl then
          create (create l x d rl) rv rd rr
        else begin
          match rl with
          | Empty → invalid_arg "Map.bal"
          | Node (rll, rlv, rld, rlr, _) →
              create (create l x d rll) rlv rld (create rlr rv rd rr)
        end
  end else
    Node (l, x, d, r, (if hl ≥ hr then hl + 1 else hr + 1))

let rec join l x d r =
  match bal l x d r with
  | Empty → invalid_arg "Pmap.join"
```

```
    | Node (l', x', d', r', _) as t' ->
        let d = height l' - height r' in
        if d < -2 ∨ d > 2 then
          join l' x' d' r'
        else
          t'
```

Merge two trees *t1* and *t2* into one. All elements of *t1* must precede the elements of *t2*. Assumes $height\ t1 - height\ t2 \leq 2$.

```
let rec merge t1 t2 =
  match t1, t2 with
  | Empty, t -> t
  | t, Empty -> t
  | Node (l1, v1, d1, r1, h1), Node (l2, v2, d2, r2, h2) ->
      bal l1 v1 d1 (bal (merge r1 l2) v2 d2 r2)
```

Same as merge, but does not assume anything about *t1* and *t2*.

```
let rec concat t1 t2 =
  match t1, t2 with
  | Empty, t -> t
  | t, Empty -> t
  | Node (l1, v1, d1, r1, h1), Node (l2, v2, d2, r2, h2) ->
      join l1 v1 d1 (join (concat r1 l2) v2 d2 r2)
```

Splitting

```
let rec split cmp x = function
  | Empty -> (Empty, None, Empty)
  | Node (l, v, d, r, _) ->
      let c = cmp x v in
      if c = 0 then
        (l, Some d, r)
      else if c < 0 then
        let ll, vl, rl = split cmp x l in
        (ll, vl, join rl v d r)
      else (* if c > 0 then *)
        let lr, vr, rr = split cmp x r in
        (join l v d lr, vr, rr)

let rec find cmp x = function
  | Empty -> raise Not_found
  | Node (l, v, d, r, _) ->
      let c = cmp x v in
      if c = 0 then
        d
      else if c < 0 then
        find cmp x l
      else (* if c > 0 *)
        find cmp x r

let rec find_opt cmp x = function
  | Empty -> None
```

```
  | Node (l, v, d, r, _) ->
      let c = cmp x v in
      if c = 0 then
        Some d
      else if c < 0 then
        find_opt cmp x l
      else (* if c > 0 *)
        find_opt cmp x r

let rec mem cmp x = function
  | Empty -> false
  | Node (l, v, d, r, _) ->
      let c = cmp x v in
      if c = 0 then
        true
      else if c < 0 then
        mem cmp x l
      else (* if c > 0 *)
        mem cmp x r

let choose = function
  | Empty -> raise Not_found
  | Node (l, v, d, r, _) -> (v, d)

let choose_opt = function
  | Empty -> None
  | Node (l, v, d, r, _) -> Some (v, d)

let uncons = function
  | Empty -> raise Not_found
  | Node (l, v, d, r, h) -> (v, d, merge l r)

let uncons_opt = function
  | Empty -> None
  | Node (l, v, d, r, h) -> Some (v, d, merge l r)

let rec remove cmp x = function
  | Empty -> Empty
  | Node (l, v, d, r, h) ->
      let c = cmp x v in
      if c = 0 then
        merge l r
      else if c < 0 then
        bal (remove cmp x l) v d r
      else (* if c > 0 *)
        bal l v d (remove cmp x r)

let rec cons cmp resolve x data' = function
  | Empty -> Node (Empty, x, data', Empty, 1)
  | Node (l, v, data, r, h) ->
      let c = cmp x v in
      if c = 0 then
        match resolve data' data with
        | Some data'' -> Node (l, x, data'', r, h)
```

```
          | None -> merge l r
      else if c < 0 then
        bal (cons cmp resolve x data' l) v data r
      else (* if c > 0 *)
        bal l v data (cons cmp resolve x data' r)

let rec update cmp resolve x data' = function
  | Empty -> Node (Empty, x, data', Empty, 1)
  | Node (l, v, data, r, h) ->
      let c = cmp x v in
      if c = 0 then
        Node (l, x, resolve data' data, r, h)
      else if c < 0 then
        bal (update cmp resolve x data' l) v data r
      else (* if c > 0 *)
        bal l v data (update cmp resolve x data' r)

let add cmp x data = update cmp (fun n o -> n) x data

let rec compose cmp resolve s1 s2 =
  match s1, s2 with
  | Empty, t2 -> t2
  | t1, Empty -> t1
  | Node (l1, v1, d1, r1, h1), Node (l2, v2, d2, r2, h2) ->
      if h1 >= h2 then
        if h2 = 1 then
          cons cmp (fun o n -> resolve n o) v2 d2 s1
        else begin
          match split cmp v1 s2 with
          | l2', None, r2' ->
              join (compose cmp resolve l1 l2') v1 d1
                (compose cmp resolve r1 r2')
          | l2', Some d, r2' ->
              begin match resolve d1 d with
              | None ->
                  concat (compose cmp resolve l1 l2')
                    (compose cmp resolve r1 r2')
              | Some d ->
                  join (compose cmp resolve l1 l2') v1 d
                    (compose cmp resolve r1 r2')
              end
        end
      else
        if h1 = 1 then
          cons cmp resolve v1 d1 s2
        else begin
          match split cmp v2 s1 with
          | l1', None, r1' ->
              join (compose cmp resolve l1' l2) v2 d2
                (compose cmp resolve r1' r2)
          | l1', Some d, r1' ->
              begin match resolve d d2 with
```

```
          | None ->
              concat (compose cmp resolve l1' l2)
                (compose cmp resolve r1' r2)
          | Some d ->
              join (compose cmp resolve l1' l2) v2 d
                (compose cmp resolve r1' r2)
          end
      end

let rec union cmp resolve s1 s2 =
  match s1, s2 with
  | Empty, t2 -> t2
  | t1, Empty -> t1
  | Node (l1, v1, d1, r1, h1), Node (l2, v2, d2, r2, h2) ->

      if h1 >= h2 then
        if h2 = 1 then
          update cmp (fun o n -> resolve n o) v2 d2 s1
        else begin
          match split cmp v1 s2 with
          | l2', None, r2' ->
              join (union cmp resolve l1 l2') v1 d1
                (union cmp resolve r1 r2')
          | l2', Some d, r2' ->
              join (union cmp resolve l1 l2') v1 (resolve d1 d)
                (union cmp resolve r1 r2')
        end
      else
        if h1 = 1 then
          update cmp resolve v1 d1 s2
        else begin
          match split cmp v2 s1 with
          | l1', None, r1' ->
              join (union cmp resolve l1' l2) v2 d2
                (union cmp resolve r1' r2)
          | l1', Some d, r1' ->
              join (union cmp resolve l1' l2) v2 (resolve d d2)
                (union cmp resolve r1' r2)
        end

let rec iter f = function
  | Empty -> ()
  | Node (l, v, d, r, _) -> iter f l; f v d; iter f r

let rec map f = function
  | Empty -> Empty
  | Node (l, v, d, r, h) -> Node (map f l, v, f d, map f r, h)

let rec mapi f = function
  | Empty -> Empty
  | Node(l, v, d, r, h) -> Node (mapi f l, v, f v d, mapi f r, h)

let rec fold f m accu =
```

```
      match m with
      | Empty → accu
      | Node (l, v, d, r, _) → fold f l (f v d (fold f r accu))

    let rec compare' cmp_k cmp_d l1 l2 =
      match l1, l2 with
      | [], [] → 0
      | [], _ → −1
      | _, [] → 1
      | Empty :: t1, Empty :: t2 → compare' cmp_k cmp_d t1 t2
      | Node (Empty, v1, d1, r1, _) :: t1,
          Node (Empty, v2, d2, r2, _) :: t2 →
          let cv = cmp_k v1 v2 in
          if cv ≠ 0 then begin
            cv
          end else begin
            let cd = cmp_d d1 d2 in
            if cd ≠ 0 then
              cd
            else
              compare' cmp_k cmp_d (r1 :: t1) (r2 :: t2)
          end
      | Node (l1, v1, d1, r1, _) :: t1, t2 →
          compare' cmp_k cmp_d (l1 :: Node (Empty, v1, d1, r1, 0) :: t1) t2
      | t1, Node (l2, v2, d2, r2, _) :: t2 →
          compare' cmp_k cmp_d t1 (l2 :: Node (Empty, v2, d2, r2, 0) :: t2)

    let compare cmp_k cmp_d m1 m2 = compare' cmp_k cmp_d [m1] [m2]

    let rec elements' accu = function
      | Empty → accu
      | Node (l, v, d, r, _) → elements' ((v, d) :: elements' accu r) l

    let elements s =
      elements' [] s

    let canonicalize cmp m =
      fold (add cmp) m empty

  end

module List =
  struct
    type ('key, α) t = ('key × α) list

    let empty = []

    let is_empty = function
      | [] → true
      | _ → false

    let singleton k d = [(k, d)]

    let rec cons cmp resolve k' d' = function
      | [] → [(k', d')]
      | ((k, d) as kd :: rest) as list →
```

```
        let c = cmp k′ k in
        if c = 0 then
          match resolve d′ d with
          | None → rest
          | Some d″ → (k′, d″) :: rest
        else if c < 0 then (* k′ < k *)
          (k′, d′) :: list
        else (* if c > 0, i. e. k < k′ *)
          kd :: cons cmp resolve k′ d′ rest

let rec update cmp resolve k′ d′ = function
  | [] → [(k′, d′)]
  | ((k, d) as kd :: rest) as list →
        let c = cmp k′ k in
        if c = 0 then
          (k′, resolve d′ d) :: rest
        else if c < 0 then (* k′ < k *)
          (k′, d′) :: list
        else (* if c > 0, i. e. k < k′ *)
          kd :: update cmp resolve k′ d′ rest

let add cmp k′ d′ list =
  update cmp (fun n o → n) k′ d′ list

let rec find cmp k′ = function
  | [] → raise Not_found
  | (k, d) :: rest →
        let c = cmp k′ k in
        if c = 0 then
          d
        else if c < 0 then (* k′ < k *)
          raise Not_found
        else (* if c > 0, i. e. k < k′ *)
          find cmp k′ rest

let rec find_opt cmp k′ = function
  | [] → None
  | (k, d) :: rest →
        let c = cmp k′ k in
        if c = 0 then
          Some d
        else if c < 0 then (* k′ < k *)
          None
        else (* if c > 0, i. e. k < k′ *)
          find_opt cmp k′ rest

let choose = function
  | [] → raise Not_found
  | kd :: _ → kd

let rec choose_opt = function
  | [] → None
  | kd :: _ → Some kd
```

```
let uncons = function
  | [] → raise Not_found
  | (k, d) :: rest → (k, d, rest)

let uncons_opt = function
  | [] → None
  | (k, d) :: rest → Some (k, d, rest)

let elements list = list

let rec mem cmp k' = function
  | [] → false
  | (k, d) :: rest →
      let c = cmp k' k in
      if c = 0 then
        true
      else if c < 0 then (* k' < k *)
        false
      else (* if c > 0, i. e. k < k' *)
        mem cmp k' rest

let rec remove cmp k' = function
  | [] → []
  | ((k, d) as kd :: rest) as list →
      let c = cmp k' k in
      if c = 0 then
        rest
      else if c < 0 then (* k' < k *)
        list
      else (* if c > 0, i. e. k < k' *)
        kd :: remove cmp k' rest

let rec compare cmp_k cmp_d m1 m2 =
  match m1, m2 with
  | [], [] → 0
  | [], _ → −1
  | _, [] → 1
  | (k1, d1) :: rest1, (k2, d2) :: rest2 →
      let c = cmp_k k1 k2 in
      if c = 0 then begin
        let c' = cmp_d d1 d2 in
        if c' = 0 then
          compare cmp_k cmp_d rest1 rest2
        else
          c'
      end else
        c

let rec iter f = function
  | [] → ()
  | (k, d) :: rest → f k d; iter f rest

let rec map f = function
  | [] → []
```

```
    | (k, d) :: rest → (k, f d) :: map f rest

  let rec mapi f = function
    | [] → []
    | (k, d) :: rest → (k, f k d) :: mapi f rest

  let rec fold f m accu =
    match m with
    | [] → accu
    | (k, d) :: rest → fold f rest (f k d accu)

  let rec compose cmp resolve m1 m2 =
    match m1, m2 with
    | [], [] → []
    | [], m → m
    | m, [] → m
    | ((k1, d1) as kd1 :: rest1), ((k2, d2) as kd2 :: rest2) →
        let c = cmp k1 k2 in
        if c = 0 then
          match resolve d1 d2 with
          | None → compose cmp resolve rest1 rest2
          | Some d → (k1, d) :: compose cmp resolve rest1 rest2
        else if c < 0 then (* k1 < k2 *)
          kd1 :: compose cmp resolve rest1 m2
        else (* if c > 0, i.e. k2 < k1 *)
          kd2 :: compose cmp resolve m1 rest2

  let rec union cmp resolve m1 m2 =
    match m1, m2 with
    | [], [] → []
    | [], m → m
    | m, [] → m
    | ((k1, d1) as kd1 :: rest1), ((k2, d2) as kd2 :: rest2) →
        let c = cmp k1 k2 in
        if c = 0 then
          (k1, resolve d1 d2) :: union cmp resolve rest1 rest2
        else if c < 0 then (* k1 < k2 *)
          kd1 :: union cmp resolve rest1 m2
        else (* if c > 0, i.e. k2 < k1 *)
          kd2 :: union cmp resolve m1 rest2

  let canonicalize cmp x = x

end
```

J.3 Interface of Partial

Partial maps that are constructed from assoc lists.

```
module type T =
  sig
    type domain
    type α t
```

```
    val of_list : (domain × α) list → α t
    val of_lists : domain list → α list → α t
    val apply : α t → domain → α
  end

module Make : functor (D : Map.OrderedType) → T with type domain = D.t
module Test : sig val suite : OUnit.test end
```

J.4 Implementation of Partial

```
module type T =
  sig
    type domain
    type α t
    val of_list : (domain × α) list → α t
    val of_lists : domain list → α list → α t
    val apply : α t → domain → α
  end

module Make (D : Map.OrderedType) : T with type domain = D.t =
  struct

    module M = Map.Make (D)

    type domain = D.t
    type α t = α M.t

    let of_list l =
      List.fold_left (fun m (d, v) → M.add d v m) M.empty l

    let of_lists domain values =
      of_list
        (try
           List.rev_map2 (fun d v → (d, v)) domain values
         with
           | Invalid_argument "List.rev_map2" →
             invalid_arg "Partial.of_lists:␣length␣mismatch")

    let apply partial d = M.find d partial

  end

module Test : sig val suite : OUnit.test end =
  struct

    open OUnit

    module P = Make (struct type t = int let compare = compare end)

    let apply_ok =
      "apply/ok" >::
        (fun () →
          let p = P.of_list [ (0,"a"); (1,"b"); (2,"c") ]
          and l = [ 0; 1; 2 ] in
          assert_equal [ "a"; "b"; "c" ] (List.map (P.apply p) l))
```

```
    let suite_apply =
      "apply" >:::
        [apply_ok]

    let suite =
      "Partial" >:::
        [suite_apply]

    let time () =
      ()

  end
```

—K— TRIES

From [4], extended for [9].

K.1 Interface of Trie

K.1.1 Monomorphically

module type T =
  sig

    type key
    type $(+\alpha)\ t$
    val $empty$: $\alpha\ t$
    val is_empty : $\alpha\ t \to bool$

Standard trie interface:

    val add : $key \to \alpha \to \alpha\ t \to \alpha\ t$
    val $find$: $key \to \alpha\ t \to \alpha$

Functionals:

    val $remove$: $key \to \alpha\ t \to \alpha\ t$
    val mem : $key \to \alpha\ t \to bool$
    val map : $(\alpha \to \beta) \to \alpha\ t \to \beta\ t$
    val $mapi$: $(key \to \alpha \to \beta) \to \alpha\ t \to \beta\ t$
    val $iter$: $(key \to \alpha \to unit) \to \alpha\ t \to unit$
    val $fold$: $(key \to \alpha \to \beta \to \beta) \to \alpha\ t \to \beta \to \beta$

Try to match a longest prefix and return the unmatched rest.

    val $longest$: $key \to \alpha\ t \to \alpha\ option \times key$

Try to match a shortest prefix and return the unmatched rest.

    val $shortest$: $key \to \alpha\ t \to \alpha\ option \times key$

K.1.2 New in O'Caml 3.08

    val $compare$: $(\alpha \to \alpha \to int) \to \alpha\ t \to \alpha\ t \to int$
    val $equal$: $(\alpha \to \alpha \to bool) \to \alpha\ t \to \alpha\ t \to bool$

K.1.3 O'Mega customization

export f_open f_close f_descend f_match trie allows us to export the trie *trie* as source code to another programming language.

$$\text{val } export : (int \to unit) \to (int \to unit) \to (int \to key \to unit) \to (int \to key \to \alpha \to unit) \to \alpha\ t \to unit$$

end

O'Caml's *Map.S* prior to Version 3.12:

module type Map_S =
sig
type key
type $(+\alpha)\ t$
val $empty : \alpha\ t$
val $is\_empty : \alpha\ t \to bool$
val $add : key \to \alpha \to \alpha\ t \to \alpha\ t$
val $find : key \to \alpha\ t \to \alpha$
val $remove : key \to \alpha\ t \to \alpha\ t$
val $mem : key \to \alpha\ t \to bool$
val $iter : (key \to \alpha \to unit) \to \alpha\ t \to unit$
val $map : (\alpha \to \beta) \to \alpha\ t \to \beta\ t$
val $mapi : (key \to \alpha \to \beta) \to \alpha\ t \to \beta\ t$
val $fold : (key \to \alpha \to \beta \to \beta) \to \alpha\ t \to \beta \to \beta$
val $compare : (\alpha \to \alpha \to int) \to \alpha\ t \to \alpha\ t \to int$
val $equal : (\alpha \to \alpha \to bool) \to \alpha\ t \to \alpha\ t \to bool$
end

module $Make\ (M : Map\_S) : T$ with type $key = M.key\ list$
module $MakeMap\ (M : Map\_S) : Map\_S$ with type $key = M.key\ list$

K.1.4 Polymorphically

module type $Poly$ =
sig

type $(\alpha, \beta)\ t$
val $empty : (\alpha, \beta)\ t$

Standard trie interface:

val $add : (\alpha \to \alpha \to int) \to \alpha\ list \to \beta \to (\alpha, \beta)\ t \to (\alpha, \beta)\ t$
val $find : (\alpha \to \alpha \to int) \to \alpha\ list \to (\alpha, \beta)\ t \to \beta$

Functionals:

val $remove : (\alpha \to \alpha \to int) \to \alpha\ list \to (\alpha, \beta)\ t \to (\alpha, \beta)\ t$
val $mem : (\alpha \to \alpha \to int) \to \alpha\ list \to (\alpha, \beta)\ t \to bool$
val $map : (\beta \to \gamma) \to (\alpha, \beta)\ t \to (\alpha, \gamma)\ t$
val $mapi : (\alpha\ list \to \beta \to \gamma) \to (\alpha, \beta)\ t \to (\alpha, \gamma)\ t$
val $iter : (\alpha\ list \to \beta \to unit) \to (\alpha, \beta)\ t \to unit$
val $fold : (\alpha\ list \to \beta \to \gamma \to \gamma) \to (\alpha, \beta)\ t \to \gamma \to \gamma$

Try to match a longest prefix and return the unmatched rest.

val *longest* : $(\alpha \to \alpha \to int) \to \alpha\ list \to (\alpha, \beta)\ t \to \beta\ option \times \alpha\ list$

Try to match a shortest prefix and return the unmatched rest.

val *shortest* : $(\alpha \to \alpha \to int) \to \alpha\ list \to (\alpha, \beta)\ t \to \beta\ option \times \alpha\ list$

K.1.5 O'Mega customization

export f_open f_close f_descend f_match trie allows us to export the trie *trie* as source code to another programming language.

val *export* : $(int \to unit) \to (int \to unit) \to$
$(int \to \alpha\ list \to unit) \to (int \to \alpha\ list \to \beta \to unit) \to (\alpha, \beta)\ t \to$
$unit$

end

module *MakePoly* $(M : Pmap.T)$: *Poly*

K.2 Implementation of Trie

K.2.1 Monomorphically

```
module type T =
  sig
    type key
    type (+α) t
    val empty : α t
    val is_empty : α t → bool
    val add : key → α → α t → α t
    val find : key → α t → α
    val remove : key → α t → α t
    val mem : key → α t → bool
    val map : (α → β) → α t → β t
    val mapi : (key → α → β) → α t → β t
    val iter : (key → α → unit) → α t → unit
    val fold : (key → α → β → β) → α t → β → β
    val longest : key → α t → α option × key
    val shortest : key → α t → α option × key
    val compare : (α → α → int) → α t → α t → int
    val equal : (α → α → bool) → α t → α t → bool
    val export : (int → unit) → (int → unit) →
      (int → key → unit) → (int → key → α → unit) → α t → unit
  end
```

O'Caml's *Map.S* prior to Version 3.12:

```
module type Map_S =
  sig
    type key
```

```
    type (+α) t
    val empty : α t
    val is_empty : α t → bool
    val add : key → α → α t → α t
    val find : key → α t → α
    val remove : key → α t → α t
    val mem : key → α t → bool
    val iter : (key → α → unit) → α t → unit
    val map : (α → β) → α t → β t
    val mapi : (key → α → β) → α t → β t
    val fold : (key → α → β → β) → α t → β → β
    val compare : (α → α → int) → α t → α t → int
    val equal : (α → α → bool) → α t → α t → bool
  end

module Make (M : Map_S) : (T with type key = M.key list) =
  struct
```

Derived from SML code by Chris Okasaki [4].

```
    type key = M.key list

    type α t = Trie of α option × α t M.t

    let empty = Trie (None, M.empty)

    let is_empty = function
      | Trie (None, m) →
          m = M.empty (* after O'Caml 3.08: M.is_empty m *)
      | _ → false

    let rec add key data trie =
      match key, trie with
      | [], Trie (_, children) → Trie (Some data, children)
      | k :: rest, Trie (node, children) →
          let t = try M.find k children with Not_found → empty in
          Trie (node, M.add k (add rest data t) children)

    let rec find key trie =
      match key, trie with
      | [], Trie (None, _) → raise Not_found
      | [], Trie (Some data, _) → data
      | k :: rest, Trie (_, children) → find rest (M.find k children)
```

The rest is my own fault ...

```
    let find1 k children =
      try Some (M.find k children) with Not_found → None

    let add_non_empty k t children =
      if t = empty then
        M.remove k children
      else
        M.add k t children

    let rec remove key trie =
```

```
  match key, trie with
  | [], Trie (_, children) → Trie (None, children)
  | k :: rest, (Trie (node, children) as orig) →
      match find1 k children with
      | None → orig
      | Some t → Trie (node, add_non_empty k (remove rest t) children)

let rec mem key trie =
  match key, trie with
  | [], Trie (None, _) → false
  | [], Trie (Some data, _) → true
  | k :: rest, Trie (_, children) →
      match find1 k children with
      | None → false
      | Some t → mem rest t

let rec map f = function
  | Trie (Some data, children) →
      Trie (Some (f data), M.map (map f) children)
  | Trie (None, children) → Trie (None, M.map (map f) children)

let rec mapi' key f = function
  | Trie (Some data, children) →
      Trie (Some (f key data), descend key f children)
  | Trie (None, children) → Trie (None, descend key f children)
and descend key f = M.mapi (fun k → mapi' (key @ [k]) f)
let mapi f = mapi' [] f

let rec iter' key f = function
  | Trie (Some data, children) → f key data; descend key f children
  | Trie (None, children) → descend key f children
and descend key f = M.iter (fun k → iter' (key @ [k]) f)
let iter f = iter' [] f

let rec fold' key f t acc =
  match t with
  | Trie (Some data, children) → descend key f children (f key data acc)
  | Trie (None, children) → descend key f children acc
and descend key f = M.fold (fun k → fold' (key @ [k]) f)
let fold f t acc = fold' [] f t acc

let rec longest' partial partial_rest key trie =
  match key, trie with
  | [], Trie (data, _) → (data, [])
  | k :: rest, Trie (data, children) →
      match data, find1 k children with
      | None, None → (partial, partial_rest)
      | Some _, None → (data, key)
      | _, Some t → longest' partial partial_rest rest t
let longest key = longest' None key key

let rec shortest' partial partial_rest key trie =
  match key, trie with
  | [], Trie (data, _) → (data, [])
```

```
  | k :: rest, Trie (Some _ as data, children) → (data, key)
  | k :: rest, Trie (None, children) →
      match find1 k children with
      | None → (partial, partial_rest)
      | Some t → shortest' partial partial_rest rest t
let shortest key = shortest' None key key
```

K.2.2 O'Mega customization

```
let rec export' n key f_open f_close f_descend f_match = function
  | Trie (Some data, children) →
      f_match n key data;
      if children ≠ M.empty then
        descend n key f_open f_close f_descend f_match children
  | Trie (None, children) →
      if children ≠ M.empty then begin
        f_descend n key;
        descend n key f_open f_close f_descend f_match children
      end
and descend n key f_open f_close f_descend f_match children =
  f_open n;
  M.iter (fun k →
    export' (succ n) (k :: key) f_open f_close f_descend f_match) children;
  f_close n

let export f_open f_close f_descend f_match =
  export' 0 [] f_open f_close f_descend f_match

let compare _ _ _ =
  failwith "incomplete"

let equal _ _ _ =
  failwith "incomplete"

end

module MakeMap (M : Map_S) : (Map_S with type key = M.key list) = Make(M)
```

K.2.3 Polymorphically

```
module type Poly =
  sig
    type (α, β) t
    val empty : (α, β) t
    val add : (α → α → int) → α list → β → (α, β) t → (α, β) t
    val find : (α → α → int) → α list → (α, β) t → β
    val remove : (α → α → int) → α list → (α, β) t → (α, β) t
    val mem : (α → α → int) → α list → (α, β) t → bool
    val map : (β → γ) → (α, β) t → (α, γ) t
    val mapi : (α list → β → γ) → (α, β) t → (α, γ) t
    val iter : (α list → β → unit) → (α, β) t → unit
```

```
  val fold : (α list → β → γ → γ) → (α, β) t → γ → γ
  val longest : (α → α → int) → α list → (α, β) t → β option × α list
  val shortest : (α → α → int) → α list → (α, β) t → β option × α list
  val export : (int → unit) → (int → unit) →
    (int → α list → unit) → (int → α list → β → unit) → (α, β) t →
unit
 end

module MakePoly (M : Pmap.T) : Poly =
 struct
```

Derived from SML code by Chris Okasaki [4].

```
  type (α, β) t = Trie of β option × (α, (α, β) t) M.t

  let empty = Trie (None, M.empty)

  let rec add cmp key data trie =
    match key, trie with
    | [], Trie (_, children) → Trie (Some data, children)
    | k :: rest, Trie (node, children) →
        let t = try M.find cmp k children with Not_found → empty in
        Trie (node, M.add cmp k (add cmp rest data t) children)

  let rec find cmp key trie =
    match key, trie with
    | [], Trie (None, _) → raise Not_found
    | [], Trie (Some data, _) → data
    | k :: rest, Trie (_, children) → find cmp rest (M.find cmp k children)
```

The rest is my own fault . . .

```
  let find1 cmp k children =
    try Some (M.find cmp k children) with Not_found → None

  let add_non_empty cmp k t children =
    if t = empty then
      M.remove cmp k children
    else
      M.add cmp k t children

  let rec remove cmp key trie =
    match key, trie with
    | [], Trie (_, children) → Trie (None, children)
    | k :: rest, (Trie (node, children) as orig) →
        match find1 cmp k children with
        | None → orig
        | Some t → Trie (node, add_non_empty cmp k (remove cmp rest t) children)

  let rec mem cmp key trie =
    match key, trie with
    | [], Trie (None, _) → false
    | [], Trie (Some data, _) → true
    | k :: rest, Trie (_, children) →
        match find1 cmp k children with
        | None → false
```

```
    | Some t → mem cmp rest t

let rec map f = function
  | Trie (Some data, children) →
      Trie (Some (f data), M.map (map f) children)
  | Trie (None, children) → Trie (None, M.map (map f) children)

let rec mapi' key f = function
  | Trie (Some data, children) →
      Trie (Some (f key data), descend key f children)
  | Trie (None, children) → Trie (None, descend key f children)
and descend key f = M.mapi (fun k → mapi' (key @ [k]) f)
let mapi f = mapi' [] f

let rec iter' key f = function
  | Trie (Some data, children) → f key data; descend key f children
  | Trie (None, children) → descend key f children
and descend key f = M.iter (fun k → iter' (key @ [k]) f)
let iter f = iter' [] f

let rec fold' key f t acc =
  match t with
  | Trie (Some data, children) → descend key f children (f key data acc)
  | Trie (None, children) → descend key f children acc
and descend key f = M.fold (fun k → fold' (key @ [k]) f)
let fold f t acc = fold' [] f t acc

let rec longest' cmp partial partial_rest key trie =
  match key, trie with
  | [], Trie (data, _) → (data, [])
  | k :: rest, Trie (data, children) →
      match data, find1 cmp k children with
      | None, None → (partial, partial_rest)
      | Some _, None → (data, key)
      | _, Some t → longest' cmp partial partial_rest rest t
let longest cmp key = longest' cmp None key key

let rec shortest' cmp partial partial_rest key trie =
  match key, trie with
  | [], Trie (data, _) → (data, [])
  | k :: rest, Trie (Some _ as data, children) → (data, key)
  | k :: rest, Trie (None, children) →
      match find1 cmp k children with
      | None → (partial, partial_rest)
      | Some t → shortest' cmp partial partial_rest rest t
let shortest cmp key = shortest' cmp None key key
```

K.2.4 O'Mega customization

```
let rec export' n key f_open f_close f_descend f_match = function
  | Trie (Some data, children) →
      f_match n key data;
```

```
        if children ≠ M.empty then
          descend n key f_open f_close f_descend f_match children
    | Trie (None, children) →
        if children ≠ M.empty then begin
          f_descend n key;
          descend n key f_open f_close f_descend f_match children
        end
  and descend n key f_open f_close f_descend f_match children =
    f_open n;
    M.iter (fun k →
      export′ (succ n) (k :: key) f_open f_close f_descend f_match) children;
    f_close n

  let export f_open f_close f_descend f_match =
    export′ 0 [] f_open f_close f_descend f_match

end
```

—L— Tensor Products

From [9].

L.1 Interface of Product

L.1.1 Lists

Since April 2001, we preserve lexicographic ordering.

val $\mathit{fold2}$: $(\alpha \rightarrow \beta \rightarrow \gamma \rightarrow \gamma) \rightarrow \alpha\ \mathit{list} \rightarrow \beta\ \mathit{list} \rightarrow \gamma \rightarrow \gamma$
val $\mathit{fold3}$: $(\alpha \rightarrow \beta \rightarrow \gamma \rightarrow \delta \rightarrow \delta) \rightarrow \alpha\ \mathit{list} \rightarrow \beta\ \mathit{list} \rightarrow \gamma\ \mathit{list} \rightarrow \delta \rightarrow \delta$
val fold : $(\alpha\ \mathit{list} \rightarrow \beta \rightarrow \beta) \rightarrow \alpha\ \mathit{list}\ \mathit{list} \rightarrow \beta \rightarrow \beta$

val $\mathit{list2}$: $(\alpha \rightarrow \beta \rightarrow \gamma) \rightarrow \alpha\ \mathit{list} \rightarrow \beta\ \mathit{list} \rightarrow \gamma\ \mathit{list}$
val $\mathit{list3}$: $(\alpha \rightarrow \beta \rightarrow \gamma \rightarrow \delta) \rightarrow \alpha\ \mathit{list} \rightarrow \beta\ \mathit{list} \rightarrow \gamma\ \mathit{list} \rightarrow \delta\ \mathit{list}$
val list : $(\alpha\ \mathit{list} \rightarrow \beta) \rightarrow \alpha\ \mathit{list}\ \mathit{list} \rightarrow \beta\ \mathit{list}$

val power : $\mathit{int} \rightarrow \alpha\ \mathit{list} \rightarrow \alpha\ \mathit{list}\ \mathit{list}$

val thread : $\alpha\ \mathit{list}\ \mathit{list} \rightarrow \alpha\ \mathit{list}\ \mathit{list}$

L.1.2 Sets

'a_set is actually α *set* for a suitable *set*, but this relation can not be expressed polymorphically (in *set*) in O'Caml. The two sets can be of different type, but we provide a symmetric version as syntactic sugar.

type $\alpha\ \mathit{set}$

type $(\alpha,\ \text{'}a\_set,\ \beta)\ \mathit{fold} = (\alpha \rightarrow \beta \rightarrow \beta) \rightarrow \text{'}a\_set \rightarrow \beta \rightarrow \beta$
type $(\alpha,\ \text{'}a\_set,\ \beta,\ \text{'}b\_set,\ \gamma)\ \mathit{fold2} =$
$(\alpha \rightarrow \beta \rightarrow \gamma \rightarrow \gamma) \rightarrow \text{'}a\_set \rightarrow \text{'}b\_set \rightarrow \gamma \rightarrow \gamma$

val outer : $(\alpha,\ \text{'}a\_set,\ \gamma)\ \mathit{fold} \rightarrow (\beta,\ \text{'}b\_set,\ \gamma)\ \mathit{fold} \rightarrow$
$(\alpha,\ \text{'}a\_set,\ \beta,\ \text{'}b\_set,\ \gamma)\ \mathit{fold2}$
val $\mathit{outer\_self}$: $(\alpha,\ \text{'}a\_set,\ \beta)\ \mathit{fold} \rightarrow (\alpha,\ \text{'}a\_set,\ \alpha,\ \text{'}a\_set,\ \beta)\ \mathit{fold2}$

L.2 Implementation of Product

L.2.1 Lists

We use the tail recursive *List.fold_left* over *List.fold_right* for efficiency, but revert the argument lists in order to preserve lexicographic ordering. The argument lists are much shorter than the results, so the cost of the *List.rev* is negligible.

```
let fold2_rev f l1 l2 acc =
  List.fold_left (fun acc1 x1 ->
    List.fold_left (fun acc2 x2 -> f x1 x2 acc2) acc1 l2) acc l1
```

```
let fold2 f l1 l2 acc =
  fold2_rev f (List.rev l1) (List.rev l2) acc
```

```
let fold3_rev f l1 l2 l3 acc =
  List.fold_left (fun acc1 x1 -> fold2 (f x1) l2 l3 acc1) acc l1
```

```
let fold3 f l1 l2 l3 acc =
  fold3_rev f (List.rev l1) (List.rev l2) (List.rev l3) acc
```

If all lists have the same type, there's also

```
let rec fold_rev f ll acc =
  match ll with
  | [] -> acc
  | [l] -> List.fold_left (fun acc' x -> f [x] acc') acc l
  | l :: rest ->
      List.fold_left (fun acc' x -> fold_rev (fun xr -> f (x :: xr)) rest acc') acc l
```

```
let fold f ll acc = fold_rev f (List.map List.rev ll) acc
```

```
let list2 op l1 l2 =
  fold2 (fun x1 x2 c -> op x1 x2 :: c) l1 l2 []
```

```
let list3 op l1 l2 l3 =
  fold3 (fun x1 x2 x3 c -> op x1 x2 x3 :: c) l1 l2 l3 []
```

```
let list op ll =
  fold (fun l c -> op l :: c) ll []
```

```
let power n l =
  list (fun x -> x) (ThoList.clone n l)
```

Reshuffling lists:

$$[[a_1;\ldots;a_k];[b_1;\ldots;b_k];[c_1;\ldots;c_k];\ldots] \to [[a_1;b_1;c_1;\ldots];[a_2;b_2;c_2;\ldots];\ldots] \tag{L.1}$$

tho : Is this really an optimal implementation?

```
let thread = function
  | head :: tail ->
      List.map List.rev
        (List.fold_left (fun i acc -> List.map2 (fun a b -> b :: a) i acc)
           (List.map (fun i -> [i]) head) tail)
  | [] -> []
```

L.2.2 Sets

The implementation is amazingly simple:

type α *set*

type (α, *'a_set*, β) *fold* = ($\alpha \rightarrow \beta \rightarrow \beta$) $\rightarrow$ *'a_set* $\rightarrow \beta \rightarrow \beta$
type (α, *'a_set*, β, *'b_set*, γ) *fold2* =
    ($\alpha \rightarrow \beta \rightarrow \gamma \rightarrow \gamma$) $\rightarrow$ *'a_set* $\rightarrow$ *'b_set* $\rightarrow \gamma \rightarrow \gamma$

let *outer fold1 fold2 f l1 l2* = *fold1* (fun *x1* $\rightarrow$ *fold2* (*f x1*) *l2*) *l1*
let *outer_self fold f l1 l2* = *fold* (fun *x1* $\rightarrow$ *fold* (*f x1*) *l2*) *l1*

—M—

(Fiber) Bundles

M.1 Interface of Bundle

See figure M.1 for the geometric intuition behind the bundle structure.

Does the current implementation support faithful projections with a forgetful comparison in the base?

```
module type Elt_Base =
  sig
    type elt
    type base
    val compare_elt : elt → elt → int
```

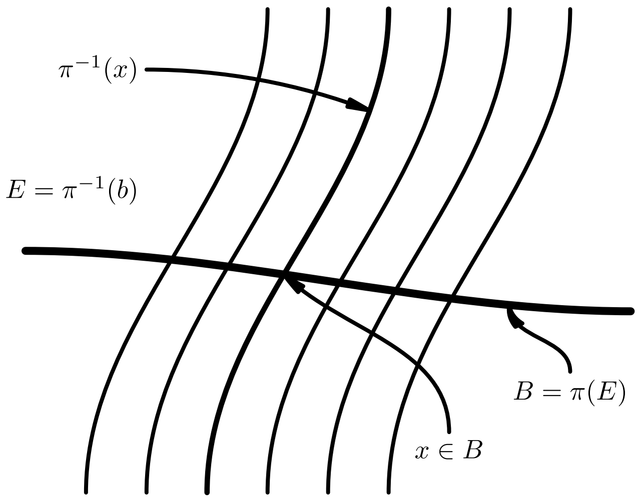


Figure M.1: The bundle structure implemented by *Bundle.T*

```
    val compare_base : base → base → int
  end
module type Projection =
  sig
    include Elt_Base
```

$\pi : E \to B$

```
    val pi : elt → base
  end
module type T =
  sig
    type t
    type elt
    type fiber = elt list
    type base
    val add : elt → t → t
    val of_list : elt list → t
```

$\pi : E \to B$

```
    val pi : elt → base
```

$\pi^{-1} : B \to E$

```
    val inv_pi : base → t → fiber
    val base : t → base list
```

$\pi^{-1} \circ \pi$

```
    val fiber : elt → t → fiber
    val fibers : t → (base × fiber) list
  end
module Make (P : Projection) : T with type elt = P.elt and type base = P.base
```

The same thing again, but with a projection that is not hardcoded, but passed as an argument at runtime.

```
module type Dyn =
  sig
    type t
    type elt
    type fiber = elt list
    type base
    val add : (elt → base) → elt → t → t
    val of_list : (elt → base) → elt list → t
    val inv_pi : base → t → fiber
    val base : t → base list
    val fiber : (elt → base) → elt → t → fiber
    val fibers : t → (base × fiber) list
  end
module Dyn (P : Elt_Base) : Dyn with type elt = P.elt and type base = P.base
```

M.2 Implementation of Bundle

```
module type Elt_Base =
  sig
    type elt
    type base
    val compare_elt : elt → elt → int
    val compare_base : base → base → int
  end

module type Dyn =
  sig
    type t
    type elt
    type fiber = elt list
    type base
    val add : (elt → base) → elt → t → t
    val of_list : (elt → base) → elt list → t
    val inv_pi : base → t → fiber
    val base : t → base list
    val fiber : (elt → base) → elt → t → fiber
    val fibers : t → (base × fiber) list
  end

module Dyn (P : Elt_Base) =
  struct

    type elt = P.elt
    type base = P.base

    type fiber = elt list

    module InvPi = Map.Make (struct type t = P.base let compare = P.compare_base end)
    module Fiber = Set.Make (struct type t = P.elt let compare = P.compare_elt end)

    type t = Fiber.t InvPi.t

    let add pi element fibers =
      let base = pi element in
      let fiber =
        try InvPi.find base fibers with Not_found → Fiber.empty in
      InvPi.add base (Fiber.add element fiber) fibers

    let of_list pi list =
      List.fold_right (add pi) list InvPi.empty

    let fibers bundle =
      InvPi.fold
        (fun base fiber acc → (base, Fiber.elements fiber) :: acc) bundle []

    let base bundle =
      InvPi.fold
        (fun base fiber acc → base :: acc) bundle []

    let inv_pi base bundle =
```

```
      try
        Fiber.elements (InvPi.find base bundle)
      with
      | Not_found → []

    let fiber pi elt bundle =
      inv_pi (pi elt) bundle

  end

module type Projection =
  sig
    include Elt_Base
    val pi : elt → base
  end

module type T =
  sig
    type t
    type elt
    type fiber = elt list
    type base
    val add : elt → t → t
    val of_list : elt list → t
    val pi : elt → base
    val inv_pi : base → t → fiber
    val base : t → base list
    val fiber : elt → t → fiber
    val fibers : t → (base × fiber) list
  end

module Make (P : Projection) =
  struct

    module D = Dyn (P)

    type elt = D.elt
    type base = D.base
    type fiber = D.fiber
    type t = D.t

    let pi = P.pi

    let add = D.add pi
    let of_list = D.of_list pi
    let base = D.base
    let inv_pi = D.inv_pi
    let fibers = D.fibers

    let fiber elt bundle =
      inv_pi (pi elt) bundle

  end
```

—N— Power Sets

N.1 Interface of PowSet

Manipulate the power set, i. e. the set of all subsets, of an set *Ordered_Type*. The concrete order is actually irrelevant, we just need it to construct *Set.S*s in the implementation. In fact, what we are implementating is the *free semilattice* generated from the set of subsets of *Ordered_Type*, where the join operation is the set union.

The non trivial operation is *basis*, which takes a set of subsets and returns the smallest set of disjoint subsets from which the argument can be reconstructed by forming unions. It is used in O'Mega for finding coarsest partitions of sets of partiticles.

Eventually, this could be generalized from *power set* or *semi lattice* to *lattice* with a notion of subtraction.

```
module type Ordered_Type =
  sig
    type t
    val compare : t → t → int
```

Debugging . . .

```
    val to_string : t → string
  end
```

```
module type T =
  sig
    type elt
    type t

    val empty : t
    val is_empty : t → bool
```

Set union (a. k. a. join).

```
    val union : t list → t
```

Construct the abstract type from a list of subsets represented as lists and the inverse operation.

```
    val of_lists : elt list list → t
```

```
    val to_lists : t -> elt list list
```

The smallest set of disjoint subsets that generates the given subset.

```
    val basis : t -> t
```

Debugging ...

```
    val to_string : t -> string
  end

module Make (E : Ordered_Type) : T with type elt = E.t
```

N.2 Implementation of PowSet

```
module type Ordered_Type =
  sig
    type t
    val compare : t -> t -> int
    val to_string : t -> string
  end

module type T =
  sig
    type elt
    type t
    val empty : t
    val is_empty : t -> bool
    val union : t list -> t
    val of_lists : elt list list -> t
    val to_lists : t -> elt list list
    val basis : t -> t
    val to_string : t -> string
  end

module Make (E : Ordered_Type) =
  struct

    type elt = E.t

    module ESet = Set.Make (E)
    type set = ESet.t

    module EPowSet = Set.Make (ESet)
    type t = EPowSet.t

    let empty = EPowSet.empty
    let is_empty = EPowSet.is_empty

    let union s_list =
      List.fold_right EPowSet.union s_list EPowSet.empty

    let set_to_string set =
      "{" ^ String.concat "," (List.map E.to_string (ESet.elements set)) ^ "}"
```

```
let to_string powset =
  "{" ^ String.concat "," (List.map set_to_string (EPowSet.elements powset)) ^ "}"

let set_of_list list =
  List.fold_right ESet.add list ESet.empty

let of_lists lists =
  List.fold_right
    (fun list acc -> EPowSet.add (set_of_list list) acc)
    lists EPowSet.empty

let to_lists ps =
  List.map ESet.elements (EPowSet.elements ps)
```

product $(s_1, s_2) = s_1 \circ s_2 = \{s_1 \setminus s_2, s_1 \cap s_2, s_2 \setminus s_1\} \setminus \{\emptyset\}$

```
let product s1 s2 =
  List.fold_left
    (fun pset set -> if ESet.is_empty set then pset else EPowSet.add set pset)
    EPowSet.empty [ESet.diff s1 s2; ESet.inter s1 s2; ESet.diff s2 s1]

let disjoint s1 s2 =
  ESet.is_empty (ESet.inter s1 s2)
```

In *augment_basis_overlapping* $(s, \{s_i\}_i)$, we are guaranteed that

$$\forall_i : s \cap s_i \neq \emptyset \tag{N.1a}$$
$$\forall_{i \neq j} : s_i \cap s_j = \emptyset\,. \tag{N.1b}$$

Therefore from (N.1b)

$$\forall_{i \neq j} : (s \cap s_i) \cap (s \cap s_j) = s \cap (s_i \cap s_j) = s \cap \emptyset = \emptyset \tag{N.2a}$$
$$\forall_{i \neq j} : (s_i \setminus s) \cap (s_j \setminus s) \subset s_i \cap s_j = \emptyset \tag{N.2b}$$
$$\forall_{i \neq j} : (s \setminus s_i) \cap (s_j \setminus s) \subset s \cap \bar{s} = \emptyset \tag{N.2c}$$
$$\forall_{i \neq j} : (s \cap s_i) \cap (s_j \setminus s) \subset s \cap \bar{s} = \emptyset\,, \tag{N.2d}$$

but in general

$$\exists_{i \neq j} : (s \setminus s_i) \cap (s \setminus s_j) \neq \emptyset \tag{N.3a}$$
$$\exists_{i \neq j} : (s \setminus s_i) \cap (s \cap s_j) \neq \emptyset\,, \tag{N.3b}$$

because, e. g., for $s_i = \{i\}$ and $s = \{1, 2, 3\}$

$$(s \setminus s_1) \cap (s \setminus s_2) = \{2, 3\} \cap \{1, 3\} = \{3\} \tag{N.4a}$$
$$(s \setminus s_1) \cap (s \cap s_2) = \{2, 3\} \cap \{2\} = \{2\}\,. \tag{N.4b}$$

Summarizing:

$\forall_{i \neq j} :\ A_i \cap A_j$	$s_j \setminus s$	$s \cap s_j$	$s \setminus s_j$
$s_i \setminus s$	$\emptyset$	$\emptyset$	$\emptyset$
$s \cap s_i$	$\emptyset$	$\emptyset$	$\neq \emptyset$
$s \setminus s_i$	$\emptyset$	$\neq \emptyset$	$\neq \emptyset$

Fortunately, we also know from (N.1a) that

$$\forall_i : |s \setminus s_i| < |s| \tag{N.5a}$$
$$\forall_i : |s \cap s_i| < \min(|s|, |s_i|) \tag{N.5b}$$
$$\forall_i : |s_i \setminus s| < |s_i| \tag{N.5c}$$

and can call *basis* recursively without risking non-termination.

```
  let rec basis ps =
    EPowSet.fold augment_basis ps EPowSet.empty

  and augment_basis s ps =
    if EPowSet.mem s ps then
      ps
    else
      let no_overlaps, overlaps = EPowSet.partition (disjoint s) ps in
      if EPowSet.is_empty overlaps then
        EPowSet.add s ps
      else
        EPowSet.union no_overlaps (augment_basis_overlapping s overlaps)

  and augment_basis_overlapping s ps =
    basis (EPowSet.fold (fun s' → EPowSet.union (product s s')) ps EPowSet.empty)

end
```

—O—
Combinatorics

O.1 Interface of Combinatorics

This type is defined just for documentation. Below, most functions will construct a (possibly nested) *list* of partitions or permutations of a α *seq*.

type α *seq* $=$ α *list*

O.1.1 Simple Combinatorial Functions

The functions

$$\textit{factorial} : n \to n! \tag{O.1a}$$

$$\textit{binomial} : (n,k) \to \binom{n}{k} = \frac{n!}{k!(n-k)!} \tag{O.1b}$$

$$\textit{multinomial} : [n_1; n_2; \ldots; n_k] \to \binom{n_1 + n_2 + \ldots + n_k}{n_1, n_2, \ldots, n_k} = \frac{(n_1 + n_2 + \ldots + n_k)!}{n_1! n_2! \cdots n_k!} \tag{O.1c}$$

have not been optimized. They can quickly run out of the range of native integers.

val *factorial* : *int* $\to$ *int*
val *binomial* : *int* $\to$ *int* $\to$ *int*
val *multinomial* : *int list* $\to$ *int*

symmetry l returns the size of the symmetric group on l, i. e. the product of the factorials of the numbers of identical elements.

val *symmetry* : α *list* $\to$ *int*

O.1.2 Partitions

partitions $[n_1; n_2; \ldots; n_k]$ $[x_1; x_2; \ldots; x_n]$, where $n = n_1 + n_2 + \ldots + n_k$, returns all inequivalent partitions of $[x_1; x_2; \ldots; x_n]$ into parts of size n_1, n_2, $\ldots$, n_k. The order of the n_i is not respected. There are

$$\frac{1}{S(n_1, n_2, \ldots, n_k)} \binom{n_1 + n_2 + \ldots + n_k}{n_1, n_2, \ldots, n_k} \tag{O.2}$$

such partitions, where the symmetry factor $S(n_1, n_2, \ldots, n_k)$ is the size of the permutation group of $[n_1; n_2; \ldots; n_k]$ as determined by the function *symmetry*.

val *partitions* : $int\ list \rightarrow \alpha\ seq \rightarrow \alpha\ seq\ list\ list$

ordered_partitions is identical to *partitions*, except that the order of the n_i is respected. There are

$$\binom{n_1 + n_2 + \ldots + n_k}{n_1, n_2, \ldots, n_k} \tag{O.3}$$

such partitions.

val *ordered_partitions* : $int\ list \rightarrow \alpha\ seq \rightarrow \alpha\ seq\ list\ list$

keystones m l is equivalent to *partitions m l*, except for the special case when the length of l is even and m contains a part that has exactly half the length of l. In this case only the half of the partitions is created that has the head of l in the longest part.

val *keystones* : $int\ list \rightarrow \alpha\ seq \rightarrow \alpha\ seq\ list\ list$

It can be beneficial to factorize a common part in the partitions and keystones:

val *factorized_partitions* : $int\ list \rightarrow \alpha\ seq \rightarrow (\alpha\ seq \times \alpha\ seq\ list\ list)\ list$
val *factorized_keystones* : $int\ list \rightarrow \alpha\ seq \rightarrow (\alpha\ seq \times \alpha\ seq\ list\ list)\ list$

Special Cases

partitions is built from components that can be convenient by themselves, even thepugh they are just special cases of *partitions*.

split k l returns the list of all inequivalent splits of the list l into one part of length k and the rest. There are

$$\frac{1}{S(|l| - k, k)} \binom{|l|}{k} \tag{O.4}$$

such splits. After replacing the pairs by two-element lists, *split k l* is equivalent to *partitions* $[k;\ length\ l\ -\ k]\ l$.

val *split* : $int \rightarrow \alpha\ seq \rightarrow (\alpha\ seq \times \alpha\ seq)\ list$

Create both equipartitions of lists of even length. There are

$$\binom{|l|}{k} \tag{O.5}$$

such splits. After replacing the pairs by two-element lists, the result of *ordered_split k l* is equivalent to *ordered_partitions* $[k;\ length\ l\ -\ k]\ l$.

val *ordered_split* : $int \rightarrow \alpha\ seq \rightarrow (\alpha\ seq \times \alpha\ seq)\ list$

multi_split n k l returns the list of all inequivalent splits of the list l into n parts of length k and the rest.

val *multi_split* : $int \rightarrow int \rightarrow \alpha\ seq \rightarrow (\alpha\ seq\ list \times \alpha\ seq)\ list$
val *ordered_multi_split* : $int \rightarrow int \rightarrow \alpha\ seq \rightarrow (\alpha\ seq\ list \times \alpha\ seq)\ list$

O.1.3 Choices

choose $n\,[x_1; x_2; \ldots; x_n]$ returns the list of all n-element subsets of $[x_1; x_2; \ldots; x_n]$. *choose* n is equivalent to $(map\,fst) \circ (ordered\_split\,n)$.

val $choose\ :\ int \rightarrow\ \alpha\ seq\ \rightarrow\ \alpha\ seq\ list$

$multi\_choose\ n\ k$ is equivalent to $(map\,fst) \circ (multi\_split\,n\,k)$.

val $multi\_choose\ :\ int \rightarrow\ int \rightarrow\ \alpha\ seq\ \rightarrow\ \alpha\ seq\ list\ list$
val $ordered\_multi\_choose\ :\ int \rightarrow\ int \rightarrow\ \alpha\ seq\ \rightarrow\ \alpha\ seq\ list\ list$

O.1.4 Permutations

val $permute\ :\ \alpha\ seq\ \rightarrow\ \alpha\ seq\ list$

Graded Permutations

val $permute\_signed\ :\ \alpha\ seq\ \rightarrow\ (int \times \alpha\ seq)\ list$
val $permute\_even\ :\ \alpha\ seq\ \rightarrow\ \alpha\ seq\ list$
val $permute\_odd\ :\ \alpha\ seq\ \rightarrow\ \alpha\ seq\ list$

Tensor Products of Permutations

In other words: permutations which respect compartmentalization.

val $permute\_tensor\ :\ \alpha\ seq\ list \rightarrow\ \alpha\ seq\ list\ list$
val $permute\_tensor\_signed\ :\ \alpha\ seq\ list \rightarrow\ (int \times \alpha\ seq\ list)\ list$
val $permute\_tensor\_even\ :\ \alpha\ seq\ list \rightarrow\ \alpha\ seq\ list\ list$
val $permute\_tensor\_odd\ :\ \alpha\ seq\ list \rightarrow\ \alpha\ seq\ list\ list$

Sorting

val $sort\_signed\ :\ (\alpha\ \rightarrow\ \alpha\ \rightarrow\ int)\ \rightarrow\ \alpha\ list \rightarrow\ int \times \alpha\ list$

O.2 Implementation of Combinatorics

type $\alpha\ seq\ =\ \alpha\ list$

O.2.1 Simple Combinatorial Functions

let rec $factorial'\ fn\ n\ =$
  if $n\ <\ 1$ then
    fn
  else
    $factorial'\ (n\ \times\ fn)\ (pred\ n)$

let $factorial\ n\ =$
  let $result\ =\ factorial'\ 1\ n$ in

```
  if result < 0 then
    invalid_arg "Combinatorics.factorial␣overflow"
  else
    result
```

$$\begin{aligned}\binom{n}{k} &= \frac{n!}{k!(n-k)!} = \frac{n(n-1)\cdots(n-k+1)}{k(k-1)\cdots 1} \\ &= \frac{n(n-1)\cdots(k+1)}{(n-k)(n-k-1)\cdots 1} = \begin{cases} B_{n-k+1}(n,k) & \text{for } k \le \lfloor n/2 \rfloor \\ B_{k+1}(n,n-k) & \text{for } k > \lfloor n/2 \rfloor \end{cases}\end{aligned} \tag{O.6}$$

where

$$B_{n_\text{min}}(n,k) = \begin{cases} nB_{n_\text{min}}(n-1,k) & \text{for } n \ge n_\text{min} \\ \frac{1}{k}B_{n_\text{min}}(n,k-1) & \text{for } k > 1 \\ 1 & \text{otherwise} \end{cases} \tag{O.7}$$

```
let rec binomial' n_min n k acc =
  if n ≥ n_min then
    binomial' n_min (pred n) k (n × acc)
  else if k > 1 then
    binomial' n_min n (pred k) (acc / k)
  else
    acc

let binomial n k =
  if k > n / 2 then
    binomial' (k + 1) n (n − k) 1
  else
    binomial' (n − k + 1) n k 1
```

Overflows later, but takes much more time:

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1} \tag{O.8}$$

```
let rec slow_binomial n k =
  if n < 0 ∨ k < 0 then
    invalid_arg "Combinatorics.binomial"
  else if k = 0 ∨ k = n then
    1
  else
    slow_binomial (pred n) k + slow_binomial (pred n) (pred k)

let multinomial n_list =
  List.fold_left (fun acc n → acc / (factorial n))
    (factorial (List.fold_left (+) 0 n_list)) n_list

let symmetry l =
  List.fold_left (fun s (n, _) → s × factorial n) 1 (ThoList.classify l)
```

O.2.2 Partitions

The inner steps of the recursion (i. e. $n = 1$) are expanded as follows

$$\begin{aligned}
&\mathit{split}'(1, [p_k; p_{k-1}; \ldots; p_1], [x_l; x_{l-1}; \ldots; x_1], [x_{l+1}; x_{l+2}; \ldots; x_m]) = \\
&\qquad [([p_1; \ldots; p_k; x_{l+1}], [x_1; \ldots; x_l; x_{l+2}; \ldots; x_m]); \\
&\qquad\quad ([p_1; \ldots; p_k; x_{l+2}], [x_1; \ldots; x_l; x_{l+1}; x_{l+3} \ldots; x_m]); \ldots; \\
&\qquad\qquad ([p_1; \ldots; p_k; x_m], [x_1; \ldots; x_l; x_{l+1}; \ldots; x_{m-1}])]
\end{aligned} \tag{O.9}$$

while the outer steps (i. e. $n > 1$) perform the same with one element moved from the last argument to the first argument. At the nth level we have

$$\begin{aligned}
&\mathit{split}'(n, [p_k; p_{k-1}; \ldots; p_1], [x_l; x_{l-1}; \ldots; x_1], [x_{l+1}; x_{l+2}; \ldots; x_m]) = \\
&\quad [([p_1; \ldots; p_k; x_{l+1}; x_{l+2}; \ldots; x_{l+n}], [x_1; \ldots; x_l; x_{l+n+1}; \ldots; x_m]); \ldots; \\
&([p_1; \ldots; p_k; x_{m-n+1}; x_{m-n+2}; \ldots; x_m], [x_1; \ldots; x_l; x_{l+1}; \ldots; x_{m-n}])]
\end{aligned} \tag{O.10}$$

where the order of the $[x_1; x_2; \ldots; x_m]$ is maintained in the partitions. Variations on this multiple recursion idiom are used many times below.

```
let rec split' n rev_part rev_head = function
  | [] → []
  | x :: tail →
      let rev_part' = x :: rev_part
      and parts = split' n rev_part (x :: rev_head) tail in
      if n < 1 then
        failwith "Combinatorics.split':␣can't␣happen"
      else if n = 1 then
        (List.rev rev_part', List.rev_append rev_head tail) :: parts
      else
        split' (pred n) rev_part' rev_head tail @ parts
```

Kick off the recursion for $0 < n < |l|$ and handle the cases $n \in \{0, |l|\}$ explicitely. Use reflection symmetry for a small optimization.

```
let ordered_split_unsafe n abs_l l =
  let abs_l = List.length l in
  if n = 0 then
    [[], l]
  else if n = abs_l then
    [l, []]
  else if n ≤ abs_l / 2 then
    split' n [] [] l
  else
    List.rev_map (fun (a, b) → (b, a)) (split' (abs_l − n) [] [] l)
```

Check the arguments and call the workhorse:

```
let ordered_split n l =
  let abs_l = List.length l in
  if n < 0 ∨ n > abs_l then
    invalid_arg "Combinatorics.ordered_split"
  else
```

```
    ordered_split_unsafe n abs_l l
```

Handle equipartitions specially:

```
let split n l =
  let abs_l = List.length l in
  if n < 0 ∨ n > abs_l then
    invalid_arg "Combinatorics.split"
  else begin
    if 2 × n = abs_l then
      match l with
      | [] → failwith "Combinatorics.split: can't happen"
      | x :: tail →
          List.map (fun (p1, p2) → (x :: p1, p2)) (split' (pred n) [] [] tail)
    else
      ordered_split_unsafe n abs_l l
  end
```

If we chop off parts repeatedly, we can either keep permutations or suppress them. Generically, *attach_to_fst* has type

$$(\alpha \times \beta)\ list \to \alpha\ list \to (\alpha\ list \times \beta)\ list \to (\alpha\ list \times \beta)\ list$$

and semantics

$$\begin{aligned}attach\_to\_fst([(a_1,b_1),(a_2,b_2),\ldots,(a_m,b_m)],[a'_1,a'_2,\ldots]) = \\ [([a_1,a'_1,\ldots],b_1),([a_2,a'_1,\ldots],b_2),\ldots,([a_m,a'_1,\ldots],b_m)]\end{aligned} \tag{O.11}$$

(where some of the result can be filtered out), assumed to be prepended to the final argument.

```
let rec multi_split' attach_to_fst n size splits =
  if n ≤ 0 then
    splits
  else
    multi_split' attach_to_fst (pred n) size
      (List.fold_left (fun acc (parts, tail) →
        attach_to_fst (ordered_split size tail) parts acc) [] splits)
```

```
let attach_to_fst_unsorted splits parts acc =
  List.fold_left (fun acc' (p, rest) → (p :: parts, rest) :: acc') acc splits
```

Similarly, if the secod argument is a list of lists:

```
let prepend_to_fst_unsorted splits parts acc =
  List.fold_left (fun acc' (p, rest) → (p @ parts, rest) :: acc') acc splits
```

```
let attach_to_fst_sorted splits parts acc =
  match parts with
  | [] → List.fold_left (fun acc' (p, rest) → ([p], rest) :: acc') acc splits
  | p :: _ as parts →
      List.fold_left (fun acc' (p', rest) →
        if p' > p then
          (p' :: parts, rest) :: acc'
        else
```

```
        acc') acc splits

let multi_split n size l =
  multi_split' attach_to_fst_sorted n size [([], l)]

let ordered_multi_split n size l =
  multi_split' attach_to_fst_unsorted n size [([], l)]

let rec partitions' splits = function
  | [] → List.map (fun (h, r) → (List.rev h, r)) splits
  | (1, size) :: more →
      partitions'
        (List.fold_left (fun acc (parts, rest) →
          attach_to_fst_unsorted (split size rest) parts acc)
           [] splits) more
  | (n, size) :: more →
      partitions'
        (List.fold_left (fun acc (parts, rest) →
          prepend_to_fst_unsorted (multi_split n size rest) parts acc)
           [] splits) more

let partitions multiplicities l =
  if List.fold_left (+) 0 multiplicities ≠ List.length l then
    invalid_arg "Combinatorics.partitions"
  else
    List.map fst (partitions' [([], l)]
                        (ThoList.classify (List.sort compare multiplicities)))

let rec ordered_partitions' splits = function
  | [] → List.map (fun (h, r) → (List.rev h, r)) splits
  | size :: more →
      ordered_partitions'
        (List.fold_left (fun acc (parts, rest) →
          attach_to_fst_unsorted (ordered_split size rest) parts acc)
           [] splits) more

let ordered_partitions multiplicities l =
  if List.fold_left (+) 0 multiplicities ≠ List.length l then
    invalid_arg "Combinatorics.ordered_partitions"
  else
    List.map fst (ordered_partitions' [([], l)] multiplicities)

let hdtl = function
  | [] → invalid_arg "Combinatorics.hdtl"
  | h :: t → (h, t)

let factorized_partitions multiplicities l =
  ThoList.factorize (List.map hdtl (partitions multiplicities l))
```

In order to construct keystones (cf. chapter 3), we must eliminate reflectionsc consistently. For this to work, the lengths of the parts *must not* be reordered arbitrarily. Ordering with monotonously fallings lengths would be incorrect however, because then some remainders could fake a reflection symmetry and partitions would be dropped erroneously. Therefore we put the longest first and order the remaining with rising lengths:

```
let longest_first l =
  match ThoList.classify (List.sort (fun n1 n2 → compare n2 n1) l) with
  | [] → []
  | longest :: rest → longest :: List.rev rest

let keystones multiplicities l =
  if List.fold_left (+) 0 multiplicities ≠ List.length l then
    invalid_arg "Combinatorics.keystones"
  else
    List.map fst (partitions' [([], l)] (longest_first multiplicities))

let factorized_keystones multiplicities l =
  ThoList.factorize (List.map hdtl (keystones multiplicities l))
```

O.2.3 Choices

The implementation is very similar to *split'*, but here we don't have to keep track of the complements of the chosen sets.

```
let rec choose' n rev_choice = function
  | [] → []
  | x :: tail →
      let rev_choice' = x :: rev_choice
      and choices = choose' n rev_choice tail in
      if n < 1 then
        failwith "Combinatorics.choose':␣can't␣happen"
      else if n = 1 then
        List.rev rev_choice' :: choices
      else
        choose' (pred n) rev_choice' tail @ choices
```

choose n is equivalent to $(List.map\,fst) \circ (split\_ordered\,n)$, but more efficient.

```
let choose n l =
  let abs_l = List.length l in
  if n < 0 then
    invalid_arg "Combinatorics.choose"
  else if n > abs_l then
    []
  else if n = 0 then
    [[]]
  else if n = abs_l then
    [l]
  else
    choose' n [] l

let multi_choose n size l =
  List.map fst (multi_split n size l)

let ordered_multi_choose n size l =
  List.map fst (ordered_multi_split n size l)
```

O.2.4 Permutations

```
let rec insert x = function
  | [] → [[x]]
  | h :: t as l →
      (x :: l) :: List.rev_map (fun l' → h :: l') (insert x t)

let permute l =
  List.fold_left (fun acc x → ThoList.rev_flatmap (insert x) acc) [[]] l
```

Graded Permutations

```
let rec insert_signed x = function
  | (eps, []) → [(eps, [x])]
  | (eps, h :: t) → (eps, x :: h :: t) ::
      (List.map (fun (eps', l') → (−eps', h :: l')) (insert_signed x (eps, t)))

let rec permute_signed' = function
  | (eps, []) → [(eps, [])]
  | (eps, h :: t) → ThoList.flatmap (insert_signed h) (permute_signed' (eps, t))

let permute_signed l =
  permute_signed' (1, l)
```

The following are wasting at most a factor of two and there's probably no point in improving on this ...

```
let filter_sign s l =
  List.map snd (List.filter (fun (eps, _) → eps = s) l)

let permute_even l =
  filter_sign 1 (permute_signed l)

let permute_odd l =
  filter_sign (−1) (permute_signed l)
```

Tensor Products of Permutations

```
let permute_tensor ll =
  Product.list (fun l → l) (List.map permute ll)

let join_signs l =
  let el, pl = List.split l in
  (List.fold_left (fun acc x → x × acc) 1 el, pl)

let permute_tensor_signed ll =
  Product.list join_signs (List.map permute_signed ll)

let permute_tensor_even l =
  filter_sign 1 (permute_tensor_signed l)

let permute_tensor_odd l =
  filter_sign (−1) (permute_tensor_signed l)

let insert_inorder_signed order x (eps, l) =
```

```
let rec insert eps' accu = function
  | [] → (eps × eps', List.rev_append accu [x])
  | h :: t →
      if order x h = 0 then
        invalid_arg
          "Combinatorics.insert_inorder_signed: identical elements"
      else if order x h < 0 then
        (eps × eps', List.rev_append accu (x :: h :: t))
      else
        insert (−eps') (h :: accu) t
in
insert 1 [] l
```

Sorting

```
let sort_signed order l =
  List.fold_left (fun acc x → insert_inorder_signed order x acc) (1, []) l
```

O.3 Interface of Permutation

```
module type T =
  sig
    type t
    val of_list : int list → t
    val of_array : int array → t
    val inverse : t → t
    val compose : t → t → t
    val list : t → α list → α list
    val array : t → α array → α array
  end

module Using_Lists : T
module Using_Arrays : T

module Default : T

module Test : functor (P : T) →
  sig val suite : OUnit.test val time : unit → unit end
```

O.4 Implementation of Permutation

```
module type T =
  sig
    type t
    val of_list : int list → t
    val of_array : int array → t
    val inverse : t → t
    val compose : t → t → t
    val list : t → α list → α list
```

```
    val array : t → α array → α array
  end

module Using_Lists : T =
  struct

    type t = int list

    let of_list p =
      if List.sort compare p ≠ (ThoList.range 0 (List.length p − 1)) then
        invalid_arg "Permutation.of_list"
      else
        p

    let of_array p =
      try
        of_list (Array.to_list p)
      with
      | Invalid_argument "Permutation.of_list" →
        invalid_arg "Permutation.of_array"

    let inverse p = snd (ThoList.ariadne_sort p)

    let list p l =
      List.map snd
        (List.sort compare
           (try
              List.rev_map2 (fun i x → (i, x)) p l
            with
            | Invalid_argument "List.rev_map2" →
              invalid_arg "Permutation.list: length mismatch"))

    let array p a =
      try
        Array.of_list (list p (Array.to_list a))
      with
      | Invalid_argument "Permutation.list: length mismatch" →
        invalid_arg "Permutation.array: length mismatch"
```

Probably not optimal (or really inefficient), but correct by associativity.

```
    let compose p q =
      list (inverse q) p

  end

module Using_Arrays : T =
  struct

    type t = int array

    let of_list p =
      if List.sort compare p ≠ (ThoList.range 0 (List.length p − 1)) then
        invalid_arg "Permutation.of_list"
      else
        Array.of_list p
```

```
    let of_array p =
      try
        of_list (Array.to_list p)
      with
      | Invalid_argument "Permutation.of_list" →
        invalid_arg "Permutation.of_array"

      let inverse p =
      let len_p = Array.length p in
      let p′ = Array.make len_p p.(0) in
      for i = 0 to pred len_p do
        p′.(p.(i)) ← i
      done;
      p′

    let array p a =
      let len_a = Array.length a
      and len_p = Array.length p in
      if len_a ≠ len_p then
        invalid_arg "Permutation.array: length mismatch";
      let a′ = Array.make len_a a.(0) in
      for i = 0 to pred len_a do
        a′.(p.(i)) ← a.(i)
      done;
      a′

    let list p l =
      try
        Array.to_list (array p (Array.of_list l))
      with
      | Invalid_argument "Permutation.array: length mismatch" →
        invalid_arg "Permutation.list: length mismatch"

    let compose p q =
      array (inverse q) p

  end

module Default = Using_Arrays
```

To shuffle an array a of n elements (indices 0..n-1):

for i from n 1 downto 1 do j random integer with 0 j i exchange aj and ai

To initialize an array a of n elements to a randomly shuffled copy of source, both 0-based:

a0 source0 for i from 1 to n 1 do j random integer with 0 j i ai aj aj source$i$

```
let shuffle l =
  let a = Array.of_list l in
  for n = Array.length a − 1 downto 1 do
    let k = Random.int (succ n) in
    if k ≠ n then
      let tmp = Array.get a n in
      Array.set a n (Array.get a k);
```

```
      Array.set a k tmp
  done;
  Array.to_list a

let time f x =
  let start = Sys.time () in
  let f_x = f x in
  let stop = Sys.time () in
  (f_x, stop -. start)

let print_time msg f x =
  let f_x, seconds = time f x in
  Printf.printf "%s took %10.2f ms\n" msg (seconds *. 1000.);
  f_x

module Test (P : T) : sig val suite : OUnit.test val time : unit → unit end =
  struct

    open OUnit
    open P

    let of_list_overlap =
      "overlap" >::
        (fun () →
          assert_raises (Invalid_argument "Permutation.of_list")
            (fun () →
              of_list [0; 1; 2; 2]))

    let of_list_gap =
      "gap" >::
        (fun () →
          assert_raises (Invalid_argument "Permutation.of_list")
            (fun () →
              of_list [0; 1; 2; 4; 5]))

    let of_list_ok =
      "ok" >::
        (fun () →
          let l = ThoList.range 0 10 in
          assert_equal (of_list l) (of_list l))

    let suite_of_list =
      "of_list" >:::
        [of_list_overlap;
         of_list_gap;
         of_list_ok]

    let apply_invalid_lengths =
      "invalid/lengths" >::
        (fun () →
          assert_raises
            (Invalid_argument "Permutation.list: length mismatch")
            (fun () →
              list (of_list [0; 1; 2; 3; 4]) [0; 1; 2; 3]))
```

```
let apply_ok =
  "ok" >::
    (fun () ->
      assert_equal [2; 0; 1; 3; 5; 4]
        (list (of_list [1; 2; 0; 3; 5; 4]) [0; 1; 2; 3; 4; 5]))

let suite_apply =
  "apply" >:::
    [apply_invalid_lengths;
     apply_ok]

let inverse_ok =
  "ok" >::
    (fun () ->
      let l = shuffle (ThoList.range 0 1000) in
      let p = of_list (shuffle l) in
      assert_equal l (list (inverse p) (list p l)))

let suite_inverse =
  "inverse" >:::
    [inverse_ok]

let compose_ok =
  "ok" >::
    (fun () ->
      let id = ThoList.range 0 1000 in
      let p = of_list (shuffle id)
      and q = of_list (shuffle id)
      and l = id in
      assert_equal (list p (list q l)) (list (compose p q) l))

let compose_inverse_ok =
  "inverse/ok" >::
    (fun () ->
      let id = ThoList.range 0 1000 in
      let p = of_list (shuffle id)
      and q = of_list (shuffle id) in
      assert_equal
        (compose (inverse p) (inverse q))
        (inverse (compose q p)))

let suite_compose =
  "compose" >:::
    [compose_ok;
     compose_inverse_ok]

let suite =
  "Permutations" >:::
    [suite_of_list;
     suite_apply;
     suite_inverse;
     suite_compose]

let repeat repetitions size =
```

```
      let id = ThoList.range 0 size in
      let p = of_list (shuffle id)
      and l = shuffle (List.map string_of_int id) in
      print_time (Printf.sprintf "reps=%d, len=%d" repetitions size)
        (fun () →
          for i = 1 to repetitions do
            ignore (P.list p l)
          done)
        ()

    let time () =
      repeat 100000 10;
      repeat 10000 100;
      repeat 1000 1000;
      repeat 100 10000;
      repeat 10 100000;
      ()

  end
```

—P—
Partitions

P.1 Interface of Partition

pairs n n1 n2 returns all (unordered) pairs of integers with the sum n in the range from *n1* to *n2*.

```
val pairs : int → int → int → (int × int) list
val triples : int → int → int → (int × int × int) list
```

tuples d n n_min n_max returns all $[n_1; n_2; \ldots; n_d]$ with $n_{\min} \leq n_1 \leq n_2 \leq \ldots \leq n_d \leq n_{\max}$ and

$$\sum_{i=1}^{d} n_i = n \tag{P.1}$$

```
val tuples : int → int → int → int → int list list
```

```
val rcs : RCS.t
```

P.2 Implementation of Partition

```
let rcs = RCS.parse "Partition" ["Partitions"]
  { RCS.revision = "$Revision:␣6465␣$";
    RCS.date = "$Date:␣2015-01-10␣16:22:31␣+0100␣(Sat,␣10␣Jan␣2015)␣$";
    RCS.author = "$Author:␣jr_reuter␣$";
    RCS.source
      = "$URL:␣svn+ssh://jr_reuter@login.hepforge.org/hepforge/svn/whizard/trunk/omega/sr
```

All unordered pairs of integers with the same sum n in a given range $\{n_1, \ldots, n_2\}$:

$$pairs : (n, n_1, n_2) \to \left\{(i,j) \,|\, i + j = n \wedge n_1 \leq i \leq j \leq n_2\right\} \tag{P.2}$$

```
let rec pairs' acc n1 n2 =
  if n1 > n2 then
    List.rev acc
  else
    pairs' ((n1, n2) :: acc) (succ n1) (pred n2)
```

```
let pairs sum min_n1 max_n2 =
  let n1 = max min_n1 (sum − max_n2) in
```

```
    let n2 = sum − n1 in
    if n2 ≤ max_n2 then
      pairs' [] n1 n2
    else
      []

let rec tuples d sum n_min n_max =
  if d ≤ 0 then
    invalid_arg "tuples"
  else if d > 1 then
    tuples' d sum n_min n_max n_min
  else if sum ≥ n_min ∧ sum ≤ n_max then
    [[sum]]
  else
    []

and tuples' d sum n_min n_max n =
  if n > n_max then
    []
  else
    List.fold_right (fun l ll → (n :: l) :: ll)
      (tuples (pred d) (sum − n) (max n_min n) n_max)
      (tuples' d sum n_min n_max (succ n))
```

When I find a little spare time, I can provide a dedicated implementation, but we *know* that *Impossible* is *never* raised and the present approach is just as good (except for a possible tiny inefficiency).

```
exception Impossible of string
let impossible name = raise (Impossible name)

let triples sum n_min n_max =
  List.map (function [n1; n2; n3] → (n1, n2, n3) | _ → impossible "triples")
    (tuples 3 sum n_min n_max)
```

—Q—
Trees

From [10]: Trees with one root admit a straightforward recursive definition

$$T(N, L) = L \cup N \times T(N, L) \times T(N, L) \tag{Q.1}$$

that is very well adapted to mathematical reasoning. Such recursive definitions are useful because they allow us to prove properties of elements by induction

$$\forall l \in L : p(l) \land (\forall n \in N : \forall t_1, t_2 \in T(N, L) : p(t_1) \land p(t_2) \Rightarrow p(n \times t_1 \times t_2)) \\ \Longrightarrow \forall t \in T(N, L) : p(t) \tag{Q.2}$$

i. e. establishing a property for all leaves and showing that a node automatically satisfies the property if it is true for all children proves the property for *all* trees. This induction is of course modelled after standard mathematical induction

$$p(1) \land (\forall n \in \mathbf{N} : p(n) \Rightarrow p(n+1)) \Longrightarrow \forall n \in \mathbf{N} : p(n) \tag{Q.3}$$

The recursive definition (Q.1) is mirrored by the two tree construction functions[1]

$$\mathit{leaf} : \nu \times \lambda \to (\nu, \lambda)T \tag{Q.4a}$$
$$\mathit{node} : \nu \times (\nu, \lambda)T \times (\nu, \lambda)T \to (\nu, \lambda)T \tag{Q.4b}$$

Renaming leaves and nodes leaves the structure of the tree invariant. Therefore, morphisms $L \to L'$ and $N \to N'$ of the sets of leaves and nodes induce natural homomorphisms $T(N, L) \to T(N', L')$ of trees

$$\mathit{map} :\ (\nu \to \nu') \times (\lambda \to \lambda') \times (\nu, \lambda)T \to (\nu', \lambda')T \tag{Q.5}$$

The homomorphisms constructed by *map* are trivial, but ubiquitous. More interesting are the morphisms

$$\begin{aligned} \mathit{fold} : (\nu \times \lambda \to \alpha) \times (\nu \times \alpha \times \alpha \to \alpha) \times (\nu, \lambda)T \to \alpha \\ (f_1, f_2, l \in L) \mapsto f_1(l) \\ (f_1, f_2, (n, t_1, t_2)) \mapsto f_2(n, \mathit{fold}(f_1, f_2, t_1), \mathit{fold}(f_1, f_2, t_2)) \end{aligned} \tag{Q.6}$$

[1] To make the introduction more accessible to non-experts, I avoid the 'curried' notation for functions with multiple arguments and use tuples instead. The actual implementation takes advantage of curried functions, however. Experts can read $\alpha \to \beta \to \gamma$ for $\alpha \times \beta \to \gamma$.

and

$$
\begin{aligned}
\mathit{fan} : (\nu \times \lambda \to \{\alpha\}) \times (\nu \times \alpha \times \alpha \to \{\alpha\}) \times (\nu, \lambda)T \to \{\alpha\} \\
(f_1, f_2, l \in L) \mapsto f_1(l) \\
(f_1, f_2, (n, t_1, t_2)) \mapsto f_2(n, \mathit{fold}(f_1, f_2, t_1) \otimes \mathit{fold}(f_1, f_2, t_2))
\end{aligned} \tag{Q.7}
$$

where the tensor product notation means that f_2 is applied to all combinations of list members in the argument:

$$\phi(\{x\} \otimes \{y\}) = \{\phi(x, y) | x \in \{x\} \wedge y \in \{y\}\} \tag{Q.8}$$

But note that due to the recursive nature of trees, *fan* is *not* a morphism from $T(N, L)$ to $T(N \otimes N, L)$.

If we identify singleton sets with their members, *fold* could be viewed as a special case of *fan*, but that is probably more confusing than helpful. Also, using the special case $\alpha = (\nu', \lambda')T$, the homomorphism *map* can be expressed in terms of *fold* and the constructors

$$
\begin{aligned}
\mathit{map} : (\nu \to \nu') \times (\lambda \to \lambda') \times (\nu, \lambda)T \to (\nu', \lambda')T \\
(f, g, t) \mapsto \mathit{fold}(\mathit{leaf} \circ (f \times g), \mathit{node} \circ (f \times \mathit{id} \times \mathit{id}), t)
\end{aligned} \tag{Q.9}
$$

fold is much more versatile than *map*, because it can be used with constructors for other tree representations to translate among different representations. The target type can also be a mathematical expression. This is used extensively below for evaluating Feynman diagrams.

Using *fan* with $\alpha = (\nu', \lambda')T$ can be used to construct a multitude of homomorphic trees. In fact, below it will be used extensively to construct all Feynman diagrams $\{(\nu, \{p_1, \ldots, p_n\})T\}$ of a given topology $t \in (\emptyset, \{1, \ldots, n\})T$.

The physicist in me guesses that there is another morphism of trees that is related to *fan* like a Lie-algebra is related to the it's Lie-group. I have not been able to pin it down, but I guess that it is a generalization of *grow* below.

Q.1 Interface of Tree

This module provides utilities for generic decorated trees, such as FeynMF output.

Q.1.1 Abstract Data Type

type $(\nu,\ \lambda)\ t$

leaf n l returns a tree consisting of a single leaf node of type n with a label l.

val *leaf* : $\nu \to \lambda \to (\nu,\ \lambda)\ t$

cons n *ch* returns a tree node.

val *cons* : $\nu \to (\nu,\ \lambda)\ t$ *list* $\to (\nu,\ \lambda)\ t$

Note that *cons node* [] constructs a terminal node, but *not* a leaf, since the latter *must* have a label!

This approach was probably tailored to Feynman diagrams, where we have external propagators as nodes with additional labels (cf. the function *to_feynmf* on page 807 below). I'm not so sure anymore that this was a good choice.

node t returns the top node of the tree *t*.

val *node* : $(\nu,\ \lambda)\ t\ \rightarrow\ \nu$

leafs t returns a list of all leaf labels *in order*.

val *leafs* : $(\nu,\ \lambda)\ t\ \rightarrow\ \lambda\ list$

nodes t returns a list of all nodes that are not leafs in post-order. This guarantees that the root node can be stripped from the result by *List.tl*.

val *nodes* : $(\nu,\ \lambda)\ t\ \rightarrow\ \nu\ list$

fuse conjg root contains_root trees joins the *trees*, using the leaf *root* in one of the trees as root of the new tree. *contains_root* guides the search for the subtree containing *root* as a leaf. fun $t\ \rightarrow$ *List.mem root* (*leafs t*) is acceptable, but more efficient solutions could be available in special circumstances.

val *fuse* : $(\nu\ \rightarrow\ \nu)\ \rightarrow\ \lambda\ \rightarrow\ ((\nu,\ \lambda)\ t\ \rightarrow\ bool)\ \rightarrow\ (\nu,\ \lambda)\ t\ list \rightarrow\ (\nu,\ \lambda)\ t$

sort lesseq t return a sorted copy of the tree *t*: node labels are ignored and nodes are according to the supremum of the leaf labels in the corresponding subtree.

val *sort* : $(\lambda\ \rightarrow\ \lambda\ \rightarrow\ bool)\ \rightarrow\ (\nu,\ \lambda)\ t\ \rightarrow\ (\nu,\ \lambda)\ t$
val *canonicalize* : $(\nu,\ \lambda)\ t\ \rightarrow\ (\nu,\ \lambda)\ t$

Q.1.2 Homomorphisms

val *map* : $('n1\ \rightarrow\ 'n2)\ \rightarrow\ ('l1\ \rightarrow\ 'l2)\ \rightarrow\ ('n1,\ 'l1)\ t\ \rightarrow\ ('n2,\ 'l2)\ t$
val *fold* : $(\nu\ \rightarrow\ \lambda\ \rightarrow\ \alpha)\ \rightarrow\ (\nu\ \rightarrow\ \alpha\ list \rightarrow\ \alpha)\ \rightarrow\ (\nu,\ \lambda)\ t\ \rightarrow\ \alpha$
val *fan* : $(\nu\ \rightarrow\ \lambda\ \rightarrow\ \alpha\ list)\ \rightarrow\ (\nu\ \rightarrow\ \alpha\ list \rightarrow\ \alpha\ list)\ \rightarrow$
  $(\nu,\ \lambda)\ t\ \rightarrow\ \alpha\ list$

Q.1.3 Output

val *to_string* : $(string,\ string)\ t\ \rightarrow\ string$

Feynmf

style : (*string* × *string*) *option* should be replaced by *style* : *string option*; *tex_label* : *string option*

type *feynmf* =
  { *style* : (*string* × *string*) *option*;
   rev : *bool*;
   label : *string option*;
   tension : *float option* }

val *vanilla* : *feynmf*
val *sty* : $(string \times string) \times bool \times string \rightarrow feynmf$

to_feynmf file to_string incoming t write the trees in the list *t* to the file named *file*. The leaves *incoming* are used as incoming particles and *to_string* is use to convert leaf labels to LaTeX-strings.

type λ *feynmf_set* =
 { *header* : *string*;
 incoming : λ *list*;
 diagrams : $(feynmf, \lambda)$ *t list* }

type (λ, μ) *feynmf_sets* =
 { *outer* : λ *feynmf_set*;
 inner : μ *feynmf_set list* }

val *feynmf_sets_plain* : $bool \rightarrow int \rightarrow string \rightarrow$
 $(\lambda \rightarrow string) \rightarrow (\lambda \rightarrow string) \rightarrow$
 $(\mu \rightarrow string) \rightarrow (\mu \rightarrow string) \rightarrow (\lambda, \mu)$ *feynmf_sets list* $\rightarrow$ *unit*

val *feynmf_sets_wrapped* : $bool \rightarrow string \rightarrow$
 $(\lambda \rightarrow string) \rightarrow (\lambda \rightarrow string) \rightarrow$
 $(\mu \rightarrow string) \rightarrow (\mu \rightarrow string) \rightarrow (\lambda, \mu)$ *feynmf_sets list* $\rightarrow$ *unit*

If the diagrams at all levels are of the same type, we can recurse to arbitrary depth.

type λ *feynmf_levels* =
 { *this* : λ *feynmf_set*;
 lower : λ *feynmf_levels list* }

to_feynmf_levels_plain sections level file wf_to_TeX p_to_TeX levels ...

val *feynmf_levels_plain* : $bool \rightarrow int \rightarrow string \rightarrow$
 $(\lambda \rightarrow string) \rightarrow (\lambda \rightarrow string) \rightarrow \lambda$ *feynmf_levels list* $\rightarrow$ *unit*

to_feynmf_levels_wrapped file wf_to_TeX p_to_TeX levels ...

val *feynmf_levels_wrapped* : $string \rightarrow$
 $(\lambda \rightarrow string) \rightarrow (\lambda \rightarrow string) \rightarrow \lambda$ *feynmf_levels list* $\rightarrow$ *unit*

Least Squares Layout

A general graph with edges of type ε, internal nodes of type ν, and external nodes of type *'ext*.

type $(\varepsilon, \nu, 'ext)$ *graph*
val *graph_of_tree* : $(\nu \rightarrow \nu \rightarrow \varepsilon) \rightarrow (\nu \rightarrow \nu) \rightarrow$
 $\nu \rightarrow (\nu, \nu)\ t \rightarrow (\varepsilon, \nu, \nu)$ *graph*

A general graph with the layout of the external nodes fixed.

type $(\varepsilon, \nu, 'ext)$ *ext_layout*
val *left_to_right* : $int \rightarrow (\varepsilon, \nu, 'ext)$ *graph* $\rightarrow (\varepsilon, \nu, 'ext)$ *ext_layout*

A general graph with the layout of all nodes fixed.

```
type (ε, ν, 'ext) layout
val layout : (ε, ν, 'ext) ext_layout → (ε, ν, 'ext) layout

val dump : (ε, ν, 'ext) layout → unit
val iter_edges : (ε → float × float → float × float → unit) →
  (ε, ν, 'ext) layout → unit
val iter_internal : (float × float → unit) →
  (ε, ν, 'ext) layout → unit
val iter_incoming : ('ext × float × float → unit) →
  (ε, ν, 'ext) layout → unit
val iter_outgoing : ('ext × float × float → unit) →
  (ε, ν, 'ext) layout → unit
```

Q.2 Implementation of Tree

Q.2.1 Abstract Data Type

```
type (ν, λ) t =
  | Leaf of ν × λ
  | Node of ν × (ν, λ) t list

let leaf n l = Leaf (n, l)

let cons n children = Node (n, children)
```

Presenting the leafs *in order* comes naturally, but will be useful below.

```
let rec leafs = function
  | Leaf (_, l) → [l]
  | Node (_, ch) → ThoList.flatmap leafs ch

let node = function
  | Leaf (n, _) → n
  | Node (n, _) → n
```

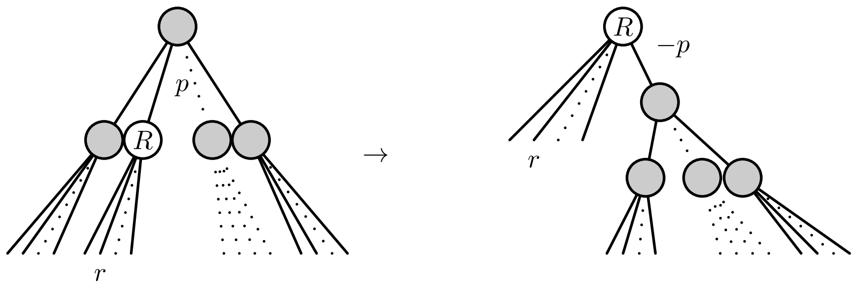
This guarantees that the root node can be stripped from the result by *List.tl*.

```
let rec nodes = function
  | Leaf _ → []
  | Node (n, ch) → n :: ThoList.flatmap nodes ch
```

first_match p list returns $(x, list')$, where x is the first element of *list* for which $p\ x$ = true and $list'$ is *list* sans x.

```
let first_match p list =
  let rec first_match' no_match = function
    | [] → invalid_arg "Tree.fuse: prospective root not found"
    | t :: rest when p t → (t, List.rev_append no_match rest)
    | t :: rest → first_match' (t :: no_match) rest in
  first_match' [] list
```

One recursion step in *fuse′* rotates the topmost tree node, moving the prospective root up:



(Q.10)

```
let fuse conjg root contains_root trees =
  let rec fuse' subtrees =
    match first_match contains_root subtrees with
```

If the prospective root is contained in a leaf, we have either found the root—in which case we're done—or have failed catastrophically:

```
    | Leaf (n, l), children →
        if l = root then
          Node (conjg n, children)
        else
          invalid_arg "Tree.fuse:␣root␣predicate␣inconsistent"
```

Otherwise, we perform a rotation as in (Q.10) and connect all nodes that do not contain the root to a new node. For efficiency, we append the new node at the end and prevent *first_match* from searching for the root in it in vain again. Since *root_children* is probably rather short, this should be a good strategy.

```
    | Node (n, root_children), other_children →
        fuse' (root_children @ [Node (conjg n, other_children)]) in
  fuse' trees
```

Sorting is also straightforward, we only have to keep track of the suprema of the subtrees:

```
type (α, β) with_supremum = { sup : α; data : β }
```

Since the lists are rather short, *Sort.list* could be replaced by an optimized version, but we're not (yet) dealing with the most important speed bottleneck here:

```
let rec sort' lesseq = function
  | Leaf (_, l) as e → { sup = l; data = e }
  | Node (n, ch) →
      let ch' = Sort.list
          (fun x y → lesseq x.sup y.sup) (List.map (sort' lesseq) ch) in
      { sup = (List.hd (List.rev ch')).sup;
        data = Node (n, List.map (fun x → x.data) ch') }
```

finally, throw away the overall supremum:

```
let sort lesseq t = (sort' lesseq t).data
```

```
let rec canonicalize = function
  | Leaf (_, _) as l → l
  | Node (n, ch) →
    Node (n, List.sort compare (List.map canonicalize ch))
```

Q.2.2 Homomorphisms

Isomophisms are simple:

```
let rec map fn fl = function
  | Leaf (n, l) → Leaf (fn n, fl l)
  | Node (n, ch) → Node (fn n, List.map (map fn fl) ch)
```

homomorphisms are not more complicated:

```
let rec fold leaf node = function
  | Leaf (n, l) → leaf n l
  | Node (n, ch) → node n (List.map (fold leaf node) ch)
```

and tensor products are fun:

```
let rec fan leaf node = function
  | Leaf (n, l) → leaf n l
  | Node (n, ch) → Product.fold
        (fun ch' t → node n ch' @ t) (List.map (fan leaf node) ch) []
```

Q.2.3 Output

```
let leaf_to_string n l =
  if n = "" then
    l
  else if l = "" then
    n
  else
    n ^ "(" ^ l ^ ")"

let node_to_string n ch =
  "(" ^ (if n = "" then "" else n ^ ":") ^ (String.concat "," ch) ^ ")"

let to_string t =
  fold leaf_to_string node_to_string t
```

Feynmf

Add a value that is greater than all suprema

```
type α supremum_or_infinity = Infinity | Sup of α

type (α, β) with_supremum_or_infinity =
    { sup : α supremum_or_infinity; data : β }

let with_infinity lesseq x y =
  match x.sup, y.sup with
```

```
  | Infinity, _ → false
  | _, Infinity → true
  | Sup x′, Sup y′ → lesseq x′ y′
```

Using this, we can sort the tree in another way that guarantees that a particular leaf (*i2*) is moved as far to the end as possible. We can then flip this leaf from outgoing to incoming without introducing a crossing:

```
let rec sort_2i′ lesseq i2 = function
  | Leaf (_, l) as e →
      { sup = if l = i2 then Infinity else Sup l; data = e }
  | Node (n, ch) →
      let ch′ = Sort.list (with_infinity lesseq)
          (List.map (sort_2i′ lesseq i2) ch) in
      { sup = (List.hd (List.rev ch′)).sup;
        data = Node (n, List.map (fun x → x.data) ch′) }
```

again, throw away the overall supremum:

```
let sort_2i lesseq i2 t = (sort_2i′ lesseq i2 t).data
```

```
type feynmf =
     { style : (string × string) option;
       rev : bool;
       label : string option;
       tension : float option }
```

```
open Printf
```

```
let style prop =
  match prop.style with
  | None → ("plain","")
  | Some s → s
```

```
let species prop = fst (style prop)
let tex_lbl prop = snd (style prop)
```

```
let leaf_label tex io leaf lab = function
  | None → fprintf tex "    \\fmflabel{${%s}$}{%s%s}\n" lab io leaf
  | Some s →
      fprintf tex "    \\fmflabel{${%s{}^{(%s)}}$}{%s%s}\n" s lab io leaf
```

```
let leaf_label tex io leaf lab label =
  ()
```

We try to draw diagrams more symmetrically by reducing the tension on the outgoing external lines.

This is insufficient for asymmetrical cascade decays.

```
let rec leaf_node tex to_label i2 n prop leaf =
  let io, tension, rev =
    if leaf = i2 then
      ("i", "", ¬ prop.rev)
    else
      ("o", ",tension=0.5", prop.rev) in
```

```
    leaf_label tex io (to_label leaf) (tex_lbl prop) prop.label ;
    fprintf tex "␣␣␣␣\\fmfdot{v%d}\n" n;
    if rev then
      fprintf tex "␣␣␣␣\\fmf{%s%s}{%s%s,v%d}\n"
        (species prop) tension io (to_label leaf) n
    else
      fprintf tex "␣␣␣␣\\fmf{%s%s}{v%d,%s%s}\n"
        (species prop) tension n io (to_label leaf)

and int_node tex to_label i2 n n' prop t =
  if prop.rev then
    fprintf tex
      "␣␣␣␣\\fmf{%s,label=\\begin{scriptsize}${%s}$\\end{scriptsize}}{v%d,v%d}\n"
      (species prop) (tex_lbl prop) n' n
  else
    fprintf tex
      "␣␣␣␣\\fmf{%s,label=\\begin{scriptsize}${%s}$\\end{scriptsize}}{v%d,v%d}\n"
      (species prop) (tex_lbl prop) n n';
  fprintf tex "␣␣␣␣\\fmfdot{v%d,v%d}\n" n n';
  edges_feynmf' tex to_label i2 n' t

and leaf_or_int_node tex to_label i2 n n' = function
  | Leaf (prop, l) → leaf_node tex to_label i2 n prop l
  | Node (prop, _) as t → int_node tex to_label i2 n n' prop t

and edges_feynmf' tex to_label i2 n = function
  | Leaf (prop, l) → leaf_node tex to_label i2 n prop l
  | Node (_, ch) →
      ignore (List.fold_right
                (fun t' n' →
                  leaf_or_int_node tex to_label i2 n n' t';
                  succ n') ch (4 × n))

let edges_feynmf tex to_label i1 i2 t =
  let n = 1 in
  begin match t with
  | Leaf _ → ()
  | Node (prop, _) →
      leaf_label tex "i" "1" (tex_lbl prop) prop.label;
      if prop.rev then
        fprintf tex "␣␣␣␣\\fmf{%s}{v%d,i%s}\n" (species prop) n (to_label i1)
      else
        fprintf tex "␣␣␣␣\\fmf{%s}{i%s,v%d}\n" (species prop) (to_label i1) n
  end;
  fprintf tex "␣␣␣␣\\fmfdot{v%d}\n" n;
  edges_feynmf' tex to_label i2 n t

let to_feynmf_channel tex to_TeX to_label incoming t =
  match incoming with
  | i1 :: i2 :: _ →
      let t' = sort_2i (≤) i2 t in
      let out = List.filter (fun a → i2 ≠ a) (leafs t') in
      fprintf tex "\\fmfframe(8,7)(8,6){%%\n";
```

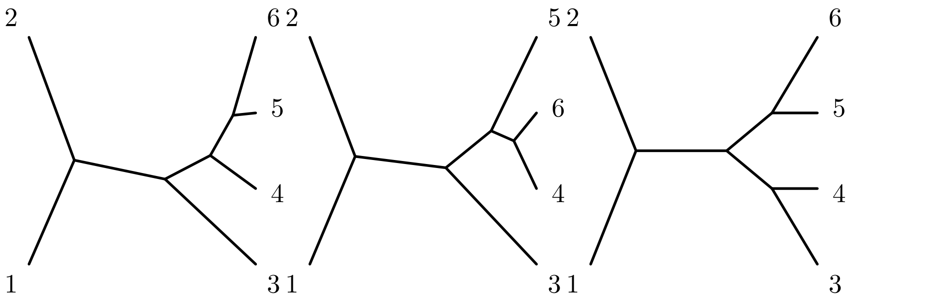


Figure Q.1: Note that this is subtly different . . .

```
      fprintf tex "␣␣\\begin{fmfgraph*}(35,30)\n";
      fprintf tex "␣␣␣\\fmfpen{thin}\n";
      fprintf tex "␣␣␣\\fmfset{arrow_len}{2mm}\n";
      fprintf tex "␣␣␣␣\\fmfleft{i%s,i%s}\n" (to_label i1) (to_label i2);
      fprintf tex "␣␣␣␣\\fmfright{o%s}\n"
        (String.concat ",o" (List.map to_label out));
      List.iter
        (fun s →
          fprintf tex "␣␣␣␣\\fmflabel{${%s}$}{i%s}\n"
            (to_TeX s) (to_label s))
        [i1; i2];
      List.iter
        (fun s →
          fprintf tex "␣␣␣␣\\fmflabel{${%s}$}{o%s}\n"
            (to_TeX s) (to_label s))
        out;
      edges_feynmf tex to_label i1 i2 t';
      fprintf tex "␣␣\\end{fmfgraph*}}\\hfil\\allowbreak\n"
  | _ → ()
```

```
let vanilla = { style = None; rev = false; label = None; tension = None }

let sty (s, r, l) = { vanilla with style = Some s; rev = r; label = Some l }

type λ feynmf_set =
  { header : string;
    incoming : λ list;
    diagrams : (feynmf, λ) t list }

type (λ, μ) feynmf_sets =
  { outer : λ feynmf_set;
    inner : μ feynmf_set list }

type λ feynmf_levels =
  { this : λ feynmf_set;
    lower : λ feynmf_levels list }

let latex_section = function
```

```
  | level when level < 0 → "part"
  | 0 → "chapter"
  | 1 → "section"
  | 2 → "subsection"
  | 3 → "subsubsection"
  | 4 → "paragraph"
  | _ → "subparagraph"

let rec feynmf_set tex sections level to_TeX to_label set =
  fprintf tex "%s\\%s{%s}\n"
    (if sections then "" else "%%␣")
    (latex_section level)
    set.header;
  List.iter
    (to_feynmf_channel tex to_TeX to_label set.incoming)
    set.diagrams

let feynmf_sets tex sections level
    to_TeX_outer to_label_outer to_TeX_inner to_label_inner set =
  feynmf_set tex sections level to_TeX_outer to_label_outer set.outer;
  List.iter
    (feynmf_set tex sections (succ level) to_TeX_inner to_label_inner)
    set.inner

let feynmf_sets_plain sections level file
    to_TeX_outer to_label_outer to_TeX_inner to_label_inner sets =
  let tex = open_out (file ^ ".tex") in
  List.iter
    (feynmf_sets tex sections level
       to_TeX_outer to_label_outer to_TeX_inner to_label_inner)
    sets;
  close_out tex

let feynmf_header tex file =
  fprintf tex "\\documentclass[10pt]{article}\n";
  fprintf tex "\\usepackage{ifpdf}\n";
  fprintf tex "\\usepackage[colorlinks]{hyperref}\n";
  fprintf tex "\\usepackage[a4paper,margin=1cm]{geometry}\n";
  fprintf tex "\\usepackage{feynmp}\n";
  fprintf tex "\\ifpdf\n";
  fprintf tex "␣␣␣\\DeclareGraphicsRule{*}{mps}{*}{}\n";
  fprintf tex "\\else\n";
  fprintf tex "␣␣␣\\DeclareGraphicsRule{*}{eps}{*}{}\n";
  fprintf tex "\\fi\n";
  fprintf tex "\\setlength{\\unitlength}{1mm}\n";
  fprintf tex "\\setlength{\\parindent}{0pt}\n";
  fprintf tex
    "\\renewcommand{\\mathstrut}{\\protect\\vphantom{\\hat{0123456789}}}\n";
  fprintf tex "\\begin{document}\n";
  fprintf tex "\\tableofcontents\n";
  fprintf tex "\\begin{fmffile}{%s-fmf}\n\n" file
```

```
let feynmf_footer tex =
  fprintf tex "\n";
  fprintf tex "\\end{fmffile}␣\n";
  fprintf tex "\\end{document}␣\n"

let feynmf_sets_wrapped latex file
    to_TeX_outer to_label_outer to_TeX_inner to_label_inner sets =
  let tex = open_out (file ^ ".tex") in
  if latex then feynmf_header tex file;
  List.iter
    (feynmf_sets tex latex 1
       to_TeX_outer to_label_outer to_TeX_inner to_label_inner)
    sets;
  if latex then feynmf_footer tex;
  close_out tex

let rec feynmf_levels tex sections level to_TeX to_label set =
  fprintf tex "%s\\%s{%s}\n"
    (if sections then "" else "%%␣")
    (latex_section level)
    set.this.header;
  List.iter
    (to_feynmf_channel tex to_TeX to_label set.this.incoming)
    set.this.diagrams;
  List.iter (feynmf_levels tex sections (succ level) to_TeX to_label) set.lower

let feynmf_levels_plain sections level file to_TeX to_label sets =
  let tex = open_out (file ^ ".tex") in
  List.iter (feynmf_levels tex sections level to_TeX to_label) sets;
  close_out tex

let feynmf_levels_wrapped file to_TeX to_label sets =
  let tex = open_out (file ^ ".tex") in
  feynmf_header tex file;
  List.iter (feynmf_levels tex true 1 to_TeX to_label) sets;
  feynmf_footer tex;
  close_out tex
```

Q.2.4 Least Squares Layout

$$L = \frac{1}{2}\sum_{i\neq i'} T_{ii'}\left(x_i - x_{i'}\right)^2 + \frac{1}{2}\sum_{i,j} T'_{ij}\left(x_i - e_j\right)^2 \tag{Q.11}$$

and thus

$$0 = \frac{\partial L}{\partial x_i} = \sum_{i'\neq i} T_{ii'}\left(x_i - x_{i'}\right) + \sum_j T'_{ij}\left(x_i - e_j\right) \tag{Q.12}$$

or

$$\left(\sum_{i'\neq i} T_{ii'} + \sum_j T'_{ij}\right) x_i - \sum_{i'\neq i} T_{ii'} x_{i'} = \sum_j T'_{ij} e_j \tag{Q.13}$$

where we can assume that

$$T_{ii'} = T_{i'i} \tag{Q.14a}$$
$$T_{ii} = 0 \tag{Q.14b}$$

```
type α node_with_tension = { node : α; tension : float }

let unit_tension t =
  map (fun n → { node = n; tension = 1.0 }) (fun l → l) t

let leafs_and_nodes i2 t =
  let t′ = sort_2i (≤ ) i2 t in
  match nodes t′ with
  | [] → failwith "Tree.nodes_and_leafs:␣impossible"
  | i1 :: _ as n → (i1, i2, List.filter (fun l → l ≠ i2) (leafs t′), n)
```

Not tail recursive, but they're unlikely to meet any deep trees:

```
let rec internal_edges_from n = function
  | Leaf _ → []
  | Node (n′, ch) → (n′, n) :: (ThoList.flatmap (internal_edges_from n′) ch)
```

The root node of the tree represents a vertex (node) and an external line (leaf) of the Feynman diagram simultaneously. Thus it requires special treatment:

```
let internal_edges = function
  | Leaf _ → []
  | Node (n, ch) → ThoList.flatmap (internal_edges_from n) ch

let rec external_edges_from n = function
  | Leaf (n′, _) → [(n′, n)]
  | Node (n′, ch) → ThoList.flatmap (external_edges_from n′) ch

let external_edges = function
  | Leaf (n, _) → [(n, n)]
  | Node (n, ch) → (n, n) :: ThoList.flatmap (external_edges_from n) ch

type ('edge, 'node, 'ext) graph =
    { int_nodes : 'node array;
      ext_nodes : 'ext array;
      int_edges : ('edge × int × int) list;
      ext_edges : ('edge × int × int) list }

module M = Pmap.Tree
```

Invert an array, viewed as a map from non-negative integers into a set. The result is a map from the set to the integers: val *invert_array* : α *array* → (α, *int*) *M.t*

```
let invert_array_unsafe a =
  fst (Array.fold_left (fun (m, i) a_i →
    (M.add compare a_i i m, succ i)) (M.empty, 0) a)

exception Not_invertible

let add_unique key data map =
  if M.mem compare key map then
```

```
    raise Not_invertible
  else
    M.add compare key data map

let invert_array a =
  fst (Array.fold_left (fun (m, i) a_i →
    (add_unique a_i i m, succ i)) (M.empty, 0) a)

let graph_of_tree nodes2edge conjugate i2 t =
  let i1, i2, out, vertices = leafs_and_nodes i2 t in
  let int_nodes = Array.of_list vertices
  and ext_nodes = Array.of_list (conjugate i1 :: i2 :: out) in
  let int_nodes_index_table = invert_array int_nodes
  and ext_nodes_index_table = invert_array ext_nodes in
  let int_nodes_index n = M.find compare n int_nodes_index_table
  and ext_nodes_index n = M.find compare n ext_nodes_index_table in
  { int_nodes = int_nodes;
    ext_nodes = ext_nodes;
    int_edges = List.map
      (fun (n1, n2) →
        (nodes2edge n1 n2, int_nodes_index n1, int_nodes_index n2))
      (internal_edges t);
    ext_edges = List.map
      (fun (e, n) →
        let e' =
          if e = i1 then
            conjugate e
          else
            e in
        (nodes2edge e' n, ext_nodes_index e', int_nodes_index n))
      (external_edges t) }

let int_incidence f null g =
  let n = Array.length g.int_nodes in
  let incidence = Array.make_matrix n n null in
  List.iter (fun (edge, n1, n2) →
    if n1 ≠ n2 then begin
      let edge' = f edge g.int_nodes.(n1) g.int_nodes.(n2) in
      incidence.(n1).(n2) ← edge';
      incidence.(n2).(n1) ← edge'
    end)
    g.int_edges;
  incidence

let ext_incidence f null g =
  let n_int = Array.length g.int_nodes
  and n_ext = Array.length g.ext_nodes in
  let incidence = Array.make_matrix n_int n_ext null in
  List.iter (fun (edge, e, n) →
    incidence.(n).(e) ← f edge g.ext_nodes.(e) g.int_nodes.(n))
    g.ext_edges;
  incidence
```

```
let division n =
  if n < 0 then
    []
  else if n = 1 then
    [0.5]
  else
    let n' = pred n in
    let d = 1.0 /. (float n') in
    let rec division' i acc =
      if i < 0 then
        acc
      else
        division' (pred i) (float i *. d :: acc) in
    division' n' []

type (ε, ν, 'ext) ext_layout = (ε, ν, 'ext × float × float) graph
type (ε, ν, 'ext) layout = (ε, ν × float × float, 'ext) ext_layout

let left_to_right num_in g =
  if num_in < 1 then
    invalid_arg "left_to_right"
  else
    let num_out = Array.length g.ext_nodes − num_in in
    if num_out < 1 then
      invalid_arg "left_to_right"
    else
      let incoming =
        List.map2 (fun e y → (e, 0.0, y))
          (Array.to_list (Array.sub g.ext_nodes 0 num_in))
          (division num_in)
      and outgoing =
        List.map2 (fun e y → (e, 1.0, y))
          (Array.to_list (Array.sub g.ext_nodes num_in num_out))
          (division num_out) in
      { g with ext_nodes = Array.of_list (incoming @ outgoing) }
```

Reformulating (Q.13)

$$Ax = b_x \tag{Q.15a}$$
$$Ay = b_y \tag{Q.15b}$$

with

$$A_{ii'} = \left(\sum_{i''\neq i} T_{ii''} + \sum_j T'_{ij}\right)\delta_{ii'} - T_{ii'} \tag{Q.16a}$$
$$(b_{x/y})_i = \sum_j T'_{ij}(e_{x/y})_j \tag{Q.16b}$$

```
let sum a = Array.fold_left (+.) 0.0 a

let tension_to_equation t t' e =
```

```
  let xe, ye = List.split e in
  let bx = Linalg.matmulv t' (Array.of_list xe)
  and by = Linalg.matmulv t' (Array.of_list ye)
  and a = Array.init (Array.length t)
      (fun i ->
        let a_i = Array.map (~-.) t.(i) in
        a_i.(i) <- a_i.(i) +. sum t.(i) +. sum t'.(i);
        a_i) in
  (a, bx, by)

let layout g =
  let ext_nodes =
    List.map (fun (_, x, y) -> (x, y)) (Array.to_list g.ext_nodes) in
  let a, bx, by =
    tension_to_equation
      (int_incidence (fun _ _ _ -> 1.0) 0.0 g)
      (ext_incidence (fun _ _ _ -> 1.0) 0.0 g) ext_nodes in
  match Linalg.solve_many a [bx; by] with
  | [x; y] -> { g with int_nodes = Array.mapi
                     (fun i n -> (n, x.(i), y.(i))) g.int_nodes }
  | _ -> failwith "impossible"

let iter_edges f g =
  List.iter (fun (edge, n1, n2) ->
    let _, x1, y1 = g.int_nodes.(n1)
    and _, x2, y2 = g.int_nodes.(n2) in
    f edge (x1, y1) (x2, y2)) g.int_edges;
  List.iter (fun (edge, e, n) ->
    let _, x1, y1 = g.ext_nodes.(e)
    and _, x2, y2 = g.int_nodes.(n) in
    f edge (x1, y1) (x2, y2)) g.ext_edges

let iter_internal f g =
  Array.iter (fun (node, x, y) -> f (x, y)) g.int_nodes

let iter_incoming f g =
  f g.ext_nodes.(0);
  f g.ext_nodes.(1)

let iter_outgoing f g =
  for i = 2 to pred (Array.length g.ext_nodes) do
    f g.ext_nodes.(i)
  done

let dump g =
  Array.iter (fun (_, x, y) -> Printf.eprintf "(%g,%g) " x y) g.ext_nodes;
  Printf.eprintf "\n => ";
  Array.iter (fun (_, x, y) -> Printf.eprintf "(%g,%g) " x y) g.int_nodes;
  Printf.eprintf "\n"
```

—R—

Dependency Trees

R.1 Interface of Tree2

Dependency trees for wavefunctions.

```
type (ν, ε) t
val cons : (ε × ν × (ν, ε) t list) list → (ν, ε) t
val leaf : ν → (ν, ε) t

val to_string : (ν → string) → (ε → string) → (ν, ε) t → string
```

R.2 Implementation of Tree2

Dependency trees for wavefunctions.

```
type (ν, ε) t =
  | Node of (ε × ν × (ν, ε) t list) list
  | Leaf of ν

let leaf node = Leaf node

let sort_children (edge, node, children) =
  (edge, node, List.sort compare children)

let cons fusions = Node (List.sort compare (List.map sort_children fusions))

let rec to_string n2s e2s = function
  | Leaf n → n2s n
  | Node children →
      "{" ^
      String.concat ","
        (List.map
           (fun (e, n, ch_list) →
             e2s e ^ ":" ^ n2s n ^
             "<(" ^ (String.concat ";" (List.map (to_string n2s e2s) ch_list)) ^ ")")
           children) ^
      "}"
```

—S—
Consistency Checks

Application `count.ml` *unavailable!*

—T—
Complex Numbers

Interface `complex.mli` *unavailable!*

Implementation `complex.ml` *unavailable!*

—U—

Algebra

U.1 Interface of Algebra

U.1.1 Coefficients

For our algebra, we need coefficient rings.

```
module type CRing =
  sig
    type t
    val null : t
    val unit : t
    val mul : t → t → t
    val add : t → t → t
    val sub : t → t → t
    val neg : t → t
    val to_string : t → string
  end
```

And rational numbers provide a particularly important example:

```
module type Rational =
  sig
    include CRing
    val is_null : t → bool
    val is_unit : t → bool
    val make : int → int → t
    val to_ratio : t → int × int
    val to_float : t → float
  end
```

U.1.2 Naive Rational Arithmetic

This *is* dangerous and will overflow even for simple applications. The production code will have to be linked to a library for large integer arithmetic.

```
module Small_Rational : Rational
```

U.1.3 Expressions: Terms, Rings and Linear Combinations

The tensor algebra will be spanned by an abelian monoid:

```
module type Term =
  sig
    type α t
    val unit : unit → α t
    val is_unit : α t → bool
    val atom : α → α t
    val power : int → α t → α t
    val mul : α t → α t → α t
    val map : (α → β) → α t → β t
    val to_string : (α → string) → α t → string
```

The derivative of a term is *not* a term, but a sum of terms instead:

$$D(f_1^{p_1} f_2^{p_2} \cdots f_n^{p_n}) = \sum_i (Df_i) p_i f_1^{p_1} f_2^{p_2} \cdots f_i^{p_i - 1} \cdots f_n^{p_n} \tag{U.1}$$

The function returns the sum as a list of triples $(Df_i, p_i, f_1^{p_1} f_2^{p_2} \cdots f_i^{p_i-1} \cdots f_n^{p_n})$. Summing the terms is left to the calling module and the Df_i are *not* guaranteed to be different. NB: The function implementating the inner derivative, is supposed to return *Some* Df_i and *None*, iff Df_i vanishes.

```
    val derive : (α → β option) → α t → (β × int × α t) list
```

convenience function

```
    val product : α t list → α t
    val atoms : α t → α list
  end

module type Ring =
  sig
    module C : Rational
    type α t
    val null : unit → α t
    val unit : unit → α t
    val is_null : α t → bool
    val is_unit : α t → bool
    val atom : α → α t
    val scale : C.t → α t → α t
    val add : α t → α t → α t
    val sub : α t → α t → α t
    val mul : α t → α t → α t
    val neg : α t → α t
```

Again

$$D(f_1^{p_1} f_2^{p_2} \cdots f_n^{p_n}) = \sum_i (Df_i) p_i f_1^{p_1} f_2^{p_2} \cdots f_i^{p_i - 1} \cdots f_n^{p_n} \tag{U.2}$$

but, iff Df_i can be identified with a f', we know how to perform the sum.

```
    val derive_inner : (α → α t) → α t → α t (* this? *)
```

```
val derive_inner' : (α → α t option) → α t → α t (* or that? *)
```

Below, we will need partial derivatives that lead out of the ring: *derive_outer derive_atom term* returns a list of partial derivatives β with non-zero coefficients $\alpha\ t$:

```
val derive_outer : (α → β option) → α t → (β × α t) list
```

convenience functions

```
val sum : α t list → α t
val product : α t list → α t
```

The list of all generators appearing in an expression:

```
val atoms : α t → α list

val to_string : (α → string) → α t → string

end
```

```
module type Linear =
  sig
    module C : Ring
    type (α, γ) t
    val null : unit → (α, γ) t
    val atom : α → (α, γ) t
    val singleton : γ C.t → α → (α, γ) t
    val scale : γ C.t → (α, γ) t → (α, γ) t
    val add : (α, γ) t → (α, γ) t → (α, γ) t
    val sub : (α, γ) t → (α, γ) t → (α, γ) t
```

A partial derivative w. r. t. a vector maps from a coefficient ring to the dual vector space.

```
val partial : (γ → (α, γ) t) → γ C.t → (α, γ) t
```

A linear combination of vectors

$$linear[(v_1, c_1); (v_2, c_2); \ldots; (v_n, c_n)] = \sum_{i=1}^{n} c_i \cdot v_i \tag{U.3}$$

```
val linear : ((α, γ) t × γ C.t) list → (α, γ) t
```

Some convenience functions

```
val map : (α → γ C.t → (β, δ) t) → (α, γ) t → (β, δ) t
val sum : (α, γ) t list → (α, γ) t
```

The list of all generators and the list of all generators of coefficients appearing in an expression:

```
val atoms : (α, γ) t → α list × γ list

val to_string : (α → string) → (γ → string) → (α, γ) t → string

end
```

```
module Term : Term

module Make_Ring (C : Rational) (T : Term) : Ring
module Make_Linear (C : Ring) : Linear with module C = C
```

U.2 Implementation of Algebra

The terms will be small and there's no need to be fancy and/or efficient. It's more important to have a unique representation.

```
module PM = Pmap.List
```

U.2.1 Coefficients

For our algebra, we need coefficient rings.

```
module type CRing =
  sig
    type t
    val null : t
    val unit : t
    val mul : t → t → t
    val add : t → t → t
    val sub : t → t → t
    val neg : t → t
    val to_string : t → string
  end
```

And rational numbers provide a particularly important example:

```
module type Rational =
  sig
    include CRing
    val is_null : t → bool
    val is_unit : t → bool
    val make : int → int → t
    val to_ratio : t → int × int
    val to_float : t → float
  end
```

U.2.2 Naive Rational Arithmetic

This *is* dangerous and will overflow even for simple applications. The production code will have to be linked to a library for large integer arithmetic.

Anyway, here's Euclid's algorithm:

```
let rec gcd i1 i2 =
  if i2 = 0 then
    abs i1
  else
    gcd i2 (i1 mod i2)

let lcm i1 i2 = (i1 / gcd i1 i2) × i2

module Small_Rational : Rational =
  struct
    type t = int × int
```

```
    let is_null (n, _) = (n = 0)
    let is_unit (n, d) = (n ≠ 0) ∧ (n = d)
    let null = (0, 1)
    let unit = (1, 1)
    let make n d =
      let c = gcd n d in
      (n / c, d / c)
    let mul (n1, d1) (n2, d2) = make (n1 × n2) (d1 × d2)
    let add (n1, d1) (n2, d2) = make (n1 × d2 + n2 × d1) (d1 × d2)
    let sub (n1, d1) (n2, d2) = make (n1 × d2 − n2 × d1) (d1 × d2)
    let neg (n, d) = (− n, d)
    let to_ratio (n, d) =
      if d < 0 then
        (−n, − d)
      else
        (n, d)
    let to_float (n, d) = float n /. float d
    let to_string (n, d) =
      if d = 1 then
        Printf.sprintf "%d" n
      else
        Printf.sprintf "(%d/%d)" n d
  end
```

U.2.3 Expressions: Terms, Rings and Linear Combinations

The tensor algebra will be spanned by an abelian monoid:

```
module type Term =
  sig
    type α t
    val unit : unit → α t
    val is_unit : α t → bool
    val atom : α → α t
    val power : int → α t → α t
    val mul : α t → α t → α t
    val map : (α → β) → α t → β t
    val to_string : (α → string) → α t → string
    val derive : (α → β option) → α t → (β × int × α t) list
    val product : α t list → α t
    val atoms : α t → α list
  end
```

```
module type Ring =
  sig
    module C : Rational
    type α t
    val null : unit → α t
    val unit : unit → α t
    val is_null : α t → bool
    val is_unit : α t → bool
```

```
    val atom : α → α t
    val scale : C.t → α t → α t
    val add : α t → α t → α t
    val sub : α t → α t → α t
    val mul : α t → α t → α t
    val neg : α t → α t
    val derive_inner : (α → α t) → α t → α t (* this? *)
    val derive_inner' : (α → α t option) → α t → α t (* or that? *)
    val derive_outer : (α → β option) → α t → (β × α t) list
    val sum : α t list → α t
    val product : α t list → α t
    val atoms : α t → α list
    val to_string : (α → string) → α t → string
  end

module type Linear =
  sig
    module C : Ring
    type (α, γ) t
    val null : unit → (α, γ) t
    val atom : α → (α, γ) t
    val singleton : γ C.t → α → (α, γ) t
    val scale : γ C.t → (α, γ) t → (α, γ) t
    val add : (α, γ) t → (α, γ) t → (α, γ) t
    val sub : (α, γ) t → (α, γ) t → (α, γ) t
    val partial : (γ → (α, γ) t) → γ C.t → (α, γ) t
    val linear : ((α, γ) t × γ C.t) list → (α, γ) t
    val map : (α → γ C.t → (β, δ) t) → (α, γ) t → (β, δ) t
    val sum : (α, γ) t list → (α, γ) t
    val atoms : (α, γ) t → α list × γ list
    val to_string : (α → string) → (γ → string) → (α, γ) t → string
  end

module Term : Term =
  struct

    module M = PM

    type α t = (α, int) M.t

    let unit () = M.empty
    let is_unit = M.is_empty

    let atom f = M.singleton f 1

    let power p x = M.map (( × ) p) x

    let insert1 binop f p term =
      let p' = binop (try M.find compare f term with Not_found → 0) p in
      if p' = 0 then
        M.remove compare f term
      else
        M.add compare f p' term
```

```
    let mul1 f p term = insert1 (+) f p term
    let mul x y = M.fold mul1 x y

    let map f term = M.fold (fun t → mul1 (f t)) term M.empty

    let to_string fmt term =
      String.concat "*"
        (M.fold (fun f p acc →
          (if p = 0 then
            "1"
          else if p = 1 then
            fmt f
          else
            "[" ^ fmt f ^ "]^" ^ string_of_int p) :: acc) term [])

    let derive derive1 x =
      M.fold (fun f p dx →
        if p ≠ 0 then
          match derive1 f with
          | Some df → (df, p, mul1 f (pred p) (M.remove compare f x)) :: dx
          | None → dx
        else
          dx) x []

    let product factors =
      List.fold_left mul (unit ()) factors

    let atoms t =
      List.map fst (PM.elements t)

  end

module Make_Ring (C : Rational) (T : Term) : Ring =
  struct

    module C = C
    let one = C.unit

    module M = PM

    type α t = (α T.t, C.t) M.t

    let null () = M.empty
    let is_null = M.is_empty

    let power t p = M.singleton t p
    let unit () = power (T.unit ()) one

    let is_unit t = unit () = t
```

The following should be correct too, but produces to many false positives instead! What's going on?

```
    let broken__is_unit t =
      match M.elements t with
      | [(t, p)] → T.is_unit t ∨ C.is_null p
```

```
  | _ → false
let atom t = power (T.atom t) one
let scale c x = M.map (C.mul c) x
let insert1 binop t c sum =
  let c' = binop (try M.find compare t sum with Not_found → C.null) c in
  if C.is_null c' then
    M.remove compare t sum
  else
    M.add compare t c' sum
let add x y = M.fold (insert1 C.add) x y
let sub x y = M.fold (insert1 C.sub) y x
```

One might be tempted to use *Product.outer_self M.fold* instead, but this would require us to combine tx and cx to $(tx,\ cx)$.

```
let fold2 f x y =
  M.fold (fun tx cx → M.fold (f tx cx) y) x
let mul x y =
  fold2 (fun tx cx ty cy → insert1 C.add (T.mul tx ty) (C.mul cx cy))
    x y (null ())
let neg x =
  sub (null ()) x
let neg x =
  scale (C.neg C.unit) x
```

Multiply the *derivatives* by c and add the result to dx.

```
let add_derivatives derivatives c dx =
  List.fold_left (fun acc (df, dt_c, dt_t) →
    add (mul df (power dt_t (C.mul c (C.make dt_c 1)))) acc) dx derivatives
let derive_inner derive1 x =
  M.fold (fun t →
    add_derivatives (T.derive (fun f → Some (derive1 f)) t)) x (null ())
let derive_inner' derive1 x =
  M.fold (fun t → add_derivatives (T.derive derive1 t)) x (null ())
let collect_derivatives derivatives c dx =
  List.fold_left (fun acc (df, dt_c, dt_t) →
    (df, power dt_t (C.mul c (C.make dt_c 1))) :: acc) dx derivatives
let derive_outer derive1 x =
  M.fold (fun t → collect_derivatives (T.derive derive1 t)) x []
let sum terms =
  List.fold_left add (null ()) terms
let product factors =
  List.fold_left mul (unit ()) factors
let atoms t =
```

```
    ThoList.uniq (List.sort compare
                    (ThoList.flatmap (fun (t, _) -> T.atoms t) (PM.elements t)))

  let to_string fmt sum =
    "(" ^ String.concat " + "
            (M.fold (fun t c acc ->
              if C.is_null c then
                acc
              else if C.is_unit c then
                T.to_string fmt t :: acc
              else if C.is_unit (C.neg c) then
                ("(-" ^ T.to_string fmt t ^ ")") :: acc
              else
                (C.to_string c ^ "*[" ^ T.to_string fmt t ^ "]") :: acc) sum []) ^ ")"

end

module Make_Linear (C : Ring) : Linear with module C = C =
  struct

    module C = C

    module M = PM

    type (α, γ) t = (α, γ C.t) M.t

    let null () = M.empty
    let is_null = M.is_empty
    let atom a = M.singleton a (C.unit ())
    let singleton c a = M.singleton a c

    let scale c x = M.map (C.mul c) x

    let insert1 binop t c sum =
      let c' = binop (try M.find compare t sum with Not_found -> C.null ()) c in
      if C.is_null c' then
        M.remove compare t sum
      else
        M.add compare t c' sum

    let add x y = M.fold (insert1 C.add) x y
    let sub x y = M.fold (insert1 C.sub) y x

    let map f t =
      M.fold (fun a c -> add (f a c)) t M.empty

    let sum terms =
      List.fold_left add (null ()) terms

    let linear terms =
      List.fold_left (fun acc (a, c) -> add (scale c a) acc) (null ()) terms

    let partial derive t =
      let d t' =
        let dt' = derive t' in
        if is_null dt' then
          None
```

```
      else
        Some dt' in
    linear (C.derive_outer d t)

  let atoms t =
    let a, c = List.split (PM.elements t) in
    (a, ThoList.uniq (List.sort compare (ThoList.flatmap C.atoms c)))

  let to_string fmt cfmt sum =
    "(" ^ String.concat " + "
            (M.fold (fun t c acc ->
              if C.is_null c then
                acc
              else if C.is_unit c then
                fmt t :: acc
              else if C.is_unit (C.neg c) then
                ("(-" ^ fmt t ^ ")") :: acc
              else
                (C.to_string cfmt c ^ "*" ^ fmt t) :: acc)
              sum []) ^ ")"

end
```

—V—
Simple Linear Algebra

V.1 Interface of Linalg

```
exception Singular
exception Not_Square

val copy_matrix : float array array → float array array

val matmul : float array array → float array array → float array array
val matmulv : float array array → float array → float array

val lu_decompose : float array array → float array array × float array array
val solve : float array array → float array → float array
val solve_many : float array array → float array list → float array list
```

V.2 Implementation of Linalg

This is not a functional implementations, but uses imperative array in Fotran style for maximimum speed.

```
exception Singular
exception Not_Square

let copy_matrix a =
  Array.init (Array.length a)
    (fun i → Array.copy a.(i))

let matmul a b =
  let ni = Array.length a
  and nj = Array.length b.(0)
  and n = Array.length b in
  let ab = Array.make_matrix ni nj 0.0 in
  for i = 0 to pred ni do
    for j = 0 to pred nj do
      for k = 0 to pred n do
        ab.(i).(j) ← ab.(i).(j) +. a.(i).(k) *. b.(k).(j)
      done
    done
  done;
  ab
```

```
let matmulv a v =
  let na = Array.length a in
  let nv = Array.length v in
  let v′ = Array.make na 0.0 in
  for i = 0 to pred na do
    for j = 0 to pred nv do
      v′.(i) ← v′.(i) +. a.(i).(j) *. v.(j)
    done
  done;
  v′

let maxabsval a : float =
  let x = ref (abs_float a.(0)) in
  for i = 1 to Array.length a − 1 do
    x := max !x (abs_float a.(i))
  done;
  !x
```

V.2.1 LU Decomposition

$$A = LU \tag{V.1a}$$

In more detail

$$\begin{pmatrix} a_{00} & a_{01} & \dots & a_{0(n-1)} \\ a_{10} & a_{11} & \dots & a_{1(n-1)} \\ \vdots & \vdots & \vdots & \vdots \\ a_{(n-1)0} & a_{(n-1)1} & \dots & a_{(n-1)(n-1)} \end{pmatrix} = \begin{pmatrix} 1 & 0 & \dots & 0 \\ l_{10} & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ l_{(n-1)0} & l_{(n-1)1} & \dots & 1 \end{pmatrix} \begin{pmatrix} u_{00} & u_{01} & \dots & u_{0(n-1)} \\ 0 & u_{11} & \dots & u_{1(n-1)} \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & u_{(n-1)(n-1)} \end{pmatrix} \tag{V.1b}$$

Rewriting (V.1) in block matrix notation

$$\begin{pmatrix} a_{00} & a_{0\cdot} \\ a_{\cdot 0} & A \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ l_{\cdot 0} & L \end{pmatrix} \begin{pmatrix} u_{00} & u_{0\cdot} \\ 0 & U \end{pmatrix} = \begin{pmatrix} u_{00} & u_{0\cdot} \\ l_{\cdot 0}u_{00} & l_{\cdot 0} \otimes u_{0\cdot} + LU \end{pmatrix} \tag{V.2}$$

we can solve it easily

$$u_{00} = a_{00} \tag{V.3a}$$
$$u_{0\cdot} = a_{0\cdot} \tag{V.3b}$$
$$l_{\cdot 0} = \frac{a_{\cdot 0}}{a_{00}} \tag{V.3c}$$
$$LU = A - \frac{a_{\cdot 0} \otimes a_{0\cdot}}{a_{00}} \tag{V.3d}$$

and (V.3c) and (V.3d) define a simple iterative algorithm if we work from the outside in. It just remains to add pivoting.

```
let swap a i j =
```

```
    let a_i = a.(i) in
    a.(i) ← a.(j);
    a.(j) ← a_i

let pivot_column v a n =
  let n′ = ref n
  and max_va = ref (v.(n) *. (abs_float a.(n).(n))) in
  for i = succ n to Array.length v − 1 do
    let va_i = v.(i) *. (abs_float a.(i).(n)) in
    if va_i > !max_va then begin
      n′ := i;
      max_va := va_i
    end
  done;
  !n′

let lu_decompose_in_place a =
  let n = Array.length a in
  let eps = ref 1
  and pivots = Array.make n 0
  and v =
    try
      Array.init n (fun i →
        let a_i = a.(i) in
        if Array.length a_i ≠ n then
          raise Not_Square;
        1.0 /. (maxabsval a_i))
    with
    | Division_by_zero → raise Singular in
  for i = 0 to pred n do
    let pivot = pivot_column v a i in
    if pivot ≠ i then begin
      swap a pivot i;
      eps := − !eps;
      v.(pivot) ← v.(i)
    end;
    pivots.(i) ← pivot;
    let inv_a_ii =
      try 1.0 /. a.(i).(i) with Division_by_zero → raise Singular in
    for j = succ i to pred n do
      a.(j).(i) ← inv_a_ii *. a.(j).(i)
    done;
    for j = succ i to pred n do
      for k = succ i to pred n do
        a.(j).(k) ← a.(j).(k) −. a.(j).(i) *. a.(i).(k)
      done
    done
  done;
  (pivots, !eps)

let lu_decompose_split a pivots =
  let n = Array.length pivots in
```

```
  let l = Array.make_matrix n n 0.0 in
  let u = Array.make_matrix n n 0.0 in
  for i = 0 to pred n do
    l.(i).(i) ← 1.0;
    for j = succ i to pred n do
      l.(j).(i) ← a.(j).(i)
    done
  done;
  for i = pred n downto 0 do
    swap l i pivots.(i)
  done;
  for i = 0 to pred n do
    for j = 0 to i do
      u.(j).(i) ← a.(j).(i)
    done
  done;
  (l, u)

let lu_decompose a =
  let a = copy_matrix a in
  let pivots, _ = lu_decompose_in_place a in
  lu_decompose_split a pivots

let lu_backsubstitute a pivots b =
  let n = Array.length a in
  let nonzero = ref (−1) in
  let b = Array.copy b in
  for i = 0 to pred n do
    let ll = pivots.(i) in
    let b_i = ref (b.(ll)) in
    b.(ll) ← b.(i);
    if !nonzero ≥ 0 then
      for j = !nonzero to pred i do
        b_i := !b_i −. a.(i).(j) ∗. b.(j)
      done
    else if !b_i ≠ 0.0 then
      nonzero := i;
    b.(i) ← !b_i
  done;
  for i = pred n downto 0 do
    let b_i = ref (b.(i)) in
    for j = succ i to pred n do
      b_i := !b_i −. a.(i).(j) ∗. b.(j)
    done;
    b.(i) ← !b_i /. a.(i).(i)
  done;
  b

let solve_destructive a b =
  let pivot, _ = lu_decompose_in_place a in
  lu_backsubstitute a pivot b
```

```
let solve_many_destructive a bs =
  let pivot, _ = lu_decompose_in_place a in
  List.map (lu_backsubstitute a pivot) bs

let solve a b =
  solve_destructive (copy_matrix a) b

let solve_many a bs =
  solve_many_destructive (copy_matrix a) bs
```

—W—
Talk To The WHiZard …

Talk to [11].

Temporarily disabled, until, we implement some conditional weaving…

—X— Fortran Libraries

X.1 Trivia

⟨omega_spinors.f90⟩≡
```
  ⟨Copyleft⟩
  module omega_spinors
    use kinds
    use constants
    implicit none
    private
    public :: operator (*), operator (+), operator (-)
    public :: abs
    ⟨intrinsic :: abs⟩
    type, public :: conjspinor
       ! private (omegalib needs access, but DON'T TOUCH IT!)
       complex(kind=default), dimension(4) :: a
    end type conjspinor
    type, public :: spinor
       ! private (omegalib needs access, but DON'T TOUCH IT!)
       complex(kind=default), dimension(4) :: a
    end type spinor
    ⟨Declaration of operations for spinors⟩
    integer, parameter, public :: omega_spinors_2010_01_A = 0
  contains
    ⟨Implementation of operations for spinors⟩
  end module omega_spinors
```

⟨intrinsic :: abs *(if working)*⟩≡
```
  intrinsic :: abs
```

⟨intrinsic :: conjg *(if working)*⟩≡
```
  intrinsic :: conjg
```

well, the Intel Fortran Compiler chokes on these with an internal error:

⟨intrinsic :: abs⟩≡

⟨intrinsic :: conjg⟩≡

To reenable the pure functions that have been removed for OpenMP, one should set this chunk to `pure &`

⟨`pure` *unless OpenMP*⟩≡

X.1.1 Inner Product

⟨*Declaration of operations for spinors*⟩≡
```
interface operator (*)
   module procedure conjspinor_spinor
end interface
private :: conjspinor_spinor
```

$$\bar\psi\psi' \tag{X.1}$$

NB: `dot_product` conjugates its first argument, we can either cancel this or inline `dot_product`:

⟨*Implementation of operations for spinors*⟩≡
```
pure function conjspinor_spinor (psibar, psi) result (psibarpsi)
  complex(kind=default) :: psibarpsi
  type(conjspinor), intent(in) :: psibar
  type(spinor), intent(in) :: psi
  psibarpsi = psibar%a(1)*psi%a(1) + psibar%a(2)*psi%a(2) &
            + psibar%a(3)*psi%a(3) + psibar%a(4)*psi%a(4)
end function conjspinor_spinor
```

X.1.2 Spinor Vector Space

Scalar Multiplication

⟨*Declaration of operations for spinors*⟩+≡
```
interface operator (*)
   module procedure integer_spinor, spinor_integer, &
        real_spinor, double_spinor, &
        complex_spinor, dcomplex_spinor, &
        spinor_real, spinor_double, &
        spinor_complex, spinor_dcomplex
end interface
private :: integer_spinor, spinor_integer, real_spinor, &
     double_spinor, complex_spinor, dcomplex_spinor, &
     spinor_real, spinor_double, spinor_complex, spinor_dcomplex
```

⟨*Implementation of operations for spinors*⟩+≡
```
pure function integer_spinor (x, y) result (xy)
  integer, intent(in) :: x
  type(spinor), intent(in) :: y
  type(spinor) :: xy
  xy%a = x * y%a
end function integer_spinor
```

⟨*Implementation of operations for spinors*⟩+≡
```
pure function real_spinor (x, y) result (xy)
  real(kind=single), intent(in) :: x
  type(spinor), intent(in) :: y
  type(spinor) :: xy
  xy%a = x * y%a
end function real_spinor
pure function double_spinor (x, y) result (xy)
  real(kind=default), intent(in) :: x
  type(spinor), intent(in) :: y
```

```
    type(spinor) :: xy
    xy%a = x * y%a
  end function double_spinor
  pure function complex_spinor (x, y) result (xy)
    complex(kind=single), intent(in) :: x
    type(spinor), intent(in) :: y
    type(spinor) :: xy
    xy%a = x * y%a
  end function complex_spinor
  pure function dcomplex_spinor (x, y) result (xy)
    complex(kind=default), intent(in) :: x
    type(spinor), intent(in) :: y
    type(spinor) :: xy
    xy%a = x * y%a
  end function dcomplex_spinor
  pure function spinor_integer (y, x) result (xy)
    integer, intent(in) :: x
    type(spinor), intent(in) :: y
    type(spinor) :: xy
    xy%a = x * y%a
  end function spinor_integer
  pure function spinor_real (y, x) result (xy)
    real(kind=single), intent(in) :: x
    type(spinor), intent(in) :: y
    type(spinor) :: xy
    xy%a = x * y%a
  end function spinor_real
  pure function spinor_double (y, x) result (xy)
    real(kind=default), intent(in) :: x
    type(spinor), intent(in) :: y
    type(spinor) :: xy
    xy%a = x * y%a
  end function spinor_double
  pure function spinor_complex (y, x) result (xy)
    complex(kind=single), intent(in) :: x
    type(spinor), intent(in) :: y
    type(spinor) :: xy
    xy%a = x * y%a
  end function spinor_complex
  pure function spinor_dcomplex (y, x) result (xy)
    complex(kind=default), intent(in) :: x
    type(spinor), intent(in) :: y
    type(spinor) :: xy
    xy%a = x * y%a
  end function spinor_dcomplex
```

⟨*Declaration of operations for spinors*⟩+≡

```
  interface operator (*)
     module procedure integer_conjspinor, conjspinor_integer, &
          real_conjspinor, double_conjspinor, &
          complex_conjspinor, dcomplex_conjspinor, &
          conjspinor_real, conjspinor_double, &
          conjspinor_complex, conjspinor_dcomplex
  end interface
  private :: integer_conjspinor, conjspinor_integer, real_conjspinor, &
```

```
      double_conjspinor, complex_conjspinor, dcomplex_conjspinor, &
      conjspinor_real, conjspinor_double, conjspinor_complex, &
      conjspinor_dcomplex
```

⟨*Implementation of operations for spinors*⟩+≡

```
  pure function integer_conjspinor (x, y) result (xy)
    integer, intent(in) :: x
    type(conjspinor), intent(in) :: y
    type(conjspinor) :: xy
    xy%a = x * y%a
  end function integer_conjspinor
  pure function real_conjspinor (x, y) result (xy)
    real(kind=single), intent(in) :: x
    type(conjspinor), intent(in) :: y
    type(conjspinor) :: xy
    xy%a = x * y%a
  end function real_conjspinor
  pure function double_conjspinor (x, y) result (xy)
    real(kind=default), intent(in) :: x
    type(conjspinor), intent(in) :: y
    type(conjspinor) :: xy
    xy%a = x * y%a
  end function double_conjspinor
  pure function complex_conjspinor (x, y) result (xy)
    complex(kind=single), intent(in) :: x
    type(conjspinor), intent(in) :: y
    type(conjspinor) :: xy
    xy%a = x * y%a
  end function complex_conjspinor
  pure function dcomplex_conjspinor (x, y) result (xy)
    complex(kind=default), intent(in) :: x
    type(conjspinor), intent(in) :: y
    type(conjspinor) :: xy
    xy%a = x * y%a
  end function dcomplex_conjspinor
  pure function conjspinor_integer (y, x) result (xy)
    integer, intent(in) :: x
    type(conjspinor), intent(in) :: y
    type(conjspinor) :: xy
    xy%a = x * y%a
  end function conjspinor_integer
  pure function conjspinor_real (y, x) result (xy)
    real(kind=single), intent(in) :: x
    type(conjspinor), intent(in) :: y
    type(conjspinor) :: xy
    xy%a = x * y%a
  end function conjspinor_real
  pure function conjspinor_double (y, x) result (xy)
    real(kind=default), intent(in) :: x
    type(conjspinor), intent(in) :: y
    type(conjspinor) :: xy
    xy%a = x * y%a
  end function conjspinor_double
  pure function conjspinor_complex (y, x) result (xy)
    complex(kind=single), intent(in) :: x
```

```
    type(conjspinor), intent(in) :: y
    type(conjspinor) :: xy
    xy%a = x * y%a
  end function conjspinor_complex
  pure function conjspinor_dcomplex (y, x) result (xy)
    complex(kind=default), intent(in) :: x
    type(conjspinor), intent(in) :: y
    type(conjspinor) :: xy
    xy%a = x * y%a
  end function conjspinor_dcomplex
```

Unary Plus and Minus

⟨*Declaration of operations for spinors*⟩+≡

```
  interface operator (+)
     module procedure plus_spinor, plus_conjspinor
  end interface
  private :: plus_spinor, plus_conjspinor
  interface operator (-)
     module procedure neg_spinor, neg_conjspinor
  end interface
  private :: neg_spinor, neg_conjspinor
```

⟨*Implementation of operations for spinors*⟩+≡

```
  pure function plus_spinor (x) result (plus_x)
    type(spinor), intent(in) :: x
    type(spinor) :: plus_x
    plus_x%a = x%a
  end function plus_spinor
  pure function neg_spinor (x) result (neg_x)
    type(spinor), intent(in) :: x
    type(spinor) :: neg_x
    neg_x%a = - x%a
  end function neg_spinor
```

⟨*Implementation of operations for spinors*⟩+≡

```
  pure function plus_conjspinor (x) result (plus_x)
    type(conjspinor), intent(in) :: x
    type(conjspinor) :: plus_x
    plus_x%a = x%a
  end function plus_conjspinor
  pure function neg_conjspinor (x) result (neg_x)
    type(conjspinor), intent(in) :: x
    type(conjspinor) :: neg_x
    neg_x%a = - x%a
  end function neg_conjspinor
```

Addition and Subtraction

⟨*Declaration of operations for spinors*⟩+≡

```
  interface operator (+)
     module procedure add_spinor, add_conjspinor
  end interface
  private :: add_spinor, add_conjspinor
  interface operator (-)
```

```
    module procedure sub_spinor, sub_conjspinor
  end interface
  private :: sub_spinor, sub_conjspinor
```

⟨*Implementation of operations for spinors*⟩+≡
```
  pure function add_spinor (x, y) result (xy)
    type(spinor), intent(in) :: x, y
    type(spinor) :: xy
    xy%a = x%a + y%a
  end function add_spinor
  pure function sub_spinor (x, y) result (xy)
    type(spinor), intent(in) :: x, y
    type(spinor) :: xy
    xy%a = x%a - y%a
  end function sub_spinor
```

⟨*Implementation of operations for spinors*⟩+≡
```
  pure function add_conjspinor (x, y) result (xy)
    type(conjspinor), intent(in) :: x, y
    type(conjspinor) :: xy
    xy%a = x%a + y%a
  end function add_conjspinor
  pure function sub_conjspinor (x, y) result (xy)
    type(conjspinor), intent(in) :: x, y
    type(conjspinor) :: xy
    xy%a = x%a - y%a
  end function sub_conjspinor
```

X.1.3 Norm

⟨*Declaration of operations for spinors*⟩+≡
```
  interface abs
    module procedure abs_spinor, abs_conjspinor
  end interface
  private :: abs_spinor, abs_conjspinor
```

⟨*Implementation of operations for spinors*⟩+≡
```
  pure function abs_spinor (psi) result (x)
    type(spinor), intent(in) :: psi
    real(kind=default) :: x
    x = sqrt (real (dot_product (psi%a, psi%a)))
  end function abs_spinor
```

⟨*Implementation of operations for spinors*⟩+≡
```
  pure function abs_conjspinor (psibar) result (x)
    real(kind=default) :: x
    type(conjspinor), intent(in) :: psibar
    x = sqrt (real (dot_product (psibar%a, psibar%a)))
  end function abs_conjspinor
```

X.2 Spinors Revisited

⟨`omega_bispinors.f90`⟩≡
  ⟨*Copyleft*⟩

```
module omega_bispinors
  use kinds
  use constants
  implicit none
  private
  public :: operator (*), operator (+), operator (-)
  public :: abs
  type, public :: bispinor
     ! private (omegalib needs access, but DON'T TOUCH IT!)
     complex(kind=default), dimension(4) :: a
  end type bispinor
  ⟨Declaration of operations for bispinors⟩
  integer, parameter, public :: omega_bispinors_2010_01_A = 0
contains
  ⟨Implementation of operations for bispinors⟩
end module omega_bispinors
```

⟨*Declaration of operations for bispinors*⟩≡

```
interface operator (*)
  module procedure spinor_product
end interface
private :: spinor_product
```

$$\bar{\psi}\psi' \tag{X.2}$$

NB: `dot_product` conjugates its first argument, we have to cancel this.

⟨*Implementation of operations for bispinors*⟩≡

```
pure function spinor_product (psil, psir) result (psilpsir)
  complex(kind=default) :: psilpsir
  type(bispinor), intent(in) :: psil, psir
  type(bispinor) :: psidum
  psidum%a(1) = psir%a(2)
  psidum%a(2) = - psir%a(1)
  psidum%a(3) = - psir%a(4)
  psidum%a(4) = psir%a(3)
  psilpsir = dot_product (conjg (psil%a), psidum%a)
end function spinor_product
```

X.2.1 Spinor Vector Space

Scalar Multiplication

⟨*Declaration of operations for bispinors*⟩+≡

```
interface operator (*)
   module procedure integer_bispinor, bispinor_integer, &
          real_bispinor, double_bispinor, &
          complex_bispinor, dcomplex_bispinor, &
          bispinor_real, bispinor_double, &
          bispinor_complex, bispinor_dcomplex
end interface
private :: integer_bispinor, bispinor_integer, real_bispinor, &
     double_bispinor, complex_bispinor, dcomplex_bispinor, &
     bispinor_real, bispinor_double, bispinor_complex, bispinor_dcomplex
```

⟨*Implementation of operations for bispinors*⟩+≡
```
pure function integer_bispinor (x, y) result (xy)
  type(bispinor) :: xy
  integer, intent(in) :: x
  type(bispinor), intent(in) :: y
  xy%a = x * y%a
end function integer_bispinor
```

⟨*Implementation of operations for bispinors*⟩+≡
```
pure function real_bispinor (x, y) result (xy)
  type(bispinor) :: xy
  real(kind=single), intent(in) :: x
  type(bispinor), intent(in) :: y
  xy%a = x * y%a
end function real_bispinor
```

⟨*Implementation of operations for bispinors*⟩+≡
```
pure function double_bispinor (x, y) result (xy)
  type(bispinor) :: xy
  real(kind=default), intent(in) :: x
  type(bispinor), intent(in) :: y
  xy%a = x * y%a
end function double_bispinor
```

⟨*Implementation of operations for bispinors*⟩+≡
```
pure function complex_bispinor (x, y) result (xy)
  type(bispinor) :: xy
  complex(kind=single), intent(in) :: x
  type(bispinor), intent(in) :: y
  xy%a = x * y%a
end function complex_bispinor
```

⟨*Implementation of operations for bispinors*⟩+≡
```
pure function dcomplex_bispinor (x, y) result (xy)
  type(bispinor) :: xy
  complex(kind=default), intent(in) :: x
  type(bispinor), intent(in) :: y
  xy%a = x * y%a
end function dcomplex_bispinor
```

⟨*Implementation of operations for bispinors*⟩+≡
```
pure function bispinor_integer (y, x) result (xy)
  type(bispinor) :: xy
  integer, intent(in) :: x
  type(bispinor), intent(in) :: y
  xy%a = x * y%a
end function bispinor_integer
```

⟨*Implementation of operations for bispinors*⟩+≡
```
pure function bispinor_real (y, x) result (xy)
  type(bispinor) :: xy
  real(kind=single), intent(in) :: x
  type(bispinor), intent(in) :: y
  xy%a = x * y%a
end function bispinor_real
```

⟨*Implementation of operations for bispinors*⟩+≡
```
pure function bispinor_double (y, x) result (xy)
```

```
    type(bispinor) :: xy
    real(kind=default), intent(in) :: x
    type(bispinor), intent(in) :: y
    xy%a = x * y%a
  end function bispinor_double
```

⟨*Implementation of operations for bispinors*⟩+≡

```
  pure function bispinor_complex (y, x) result (xy)
    type(bispinor) :: xy
    complex(kind=single), intent(in) :: x
    type(bispinor), intent(in) :: y
    xy%a = x * y%a
  end function bispinor_complex
```

⟨*Implementation of operations for bispinors*⟩+≡

```
  pure function bispinor_dcomplex (y, x) result (xy)
    type(bispinor) :: xy
    complex(kind=default), intent(in) :: x
    type(bispinor), intent(in) :: y
    xy%a = x * y%a
  end function bispinor_dcomplex
```

Unary Plus and Minus

⟨*Declaration of operations for bispinors*⟩+≡

```
  interface operator (+)
     module procedure plus_bispinor
  end interface
  private :: plus_bispinor
  interface operator (-)
     module procedure neg_bispinor
  end interface
  private :: neg_bispinor
```

⟨*Implementation of operations for bispinors*⟩+≡

```
  pure function plus_bispinor (x) result (plus_x)
    type(bispinor) :: plus_x
    type(bispinor), intent(in) :: x
    plus_x%a = x%a
  end function plus_bispinor
```

⟨*Implementation of operations for bispinors*⟩+≡

```
  pure function neg_bispinor (x) result (neg_x)
    type(bispinor) :: neg_x
    type(bispinor), intent(in) :: x
    neg_x%a = - x%a
  end function neg_bispinor
```

Addition and Subtraction

⟨*Declaration of operations for bispinors*⟩+≡

```
  interface operator (+)
     module procedure add_bispinor
  end interface
  private :: add_bispinor
  interface operator (-)
```

```
    module procedure sub_bispinor
  end interface
  private :: sub_bispinor
```

⟨*Implementation of operations for bispinors*⟩+≡
```
  pure function add_bispinor (x, y) result (xy)
    type(bispinor) :: xy
    type(bispinor), intent(in) :: x, y
    xy%a = x%a + y%a
  end function add_bispinor
```

⟨*Implementation of operations for bispinors*⟩+≡
```
  pure function sub_bispinor (x, y) result (xy)
    type(bispinor) :: xy
    type(bispinor), intent(in) :: x, y
    xy%a = x%a - y%a
  end function sub_bispinor
```

X.2.2 Norm

⟨*Declaration of operations for bispinors*⟩+≡
```
  interface abs
    module procedure abs_bispinor
  end interface
  private :: abs_bispinor
```

⟨*Implementation of operations for bispinors*⟩+≡
```
  pure function abs_bispinor (psi) result (x)
    real(kind=default) :: x
    type(bispinor), intent(in) :: psi
    x = sqrt (real (dot_product (psi%a, psi%a)))
  end function abs_bispinor
```

X.3 Vectorspinors

⟨omega_vectorspinors.f90⟩≡
```
  ⟨Copyleft⟩
  module omega_vectorspinors
    use kinds
    use constants
    use omega_bispinors
    use omega_vectors
    implicit none
    private
    public :: operator (*), operator (+), operator (-)
    public :: abs
    type, public :: vectorspinor
       ! private (omegalib needs access, but DON'T TOUCH IT!)
       type(bispinor), dimension(4) :: psi
    end type vectorspinor
    ⟨Declaration of operations for vectorspinors⟩
    integer, parameter, public :: omega_vectorspinors_2010_01_A = 0
  contains
    ⟨Implementation of operations for vectorspinors⟩
```

```
  end module omega_vectorspinors
```

⟨*Declaration of operations for vectorspinors*⟩≡
```
  interface operator (*)
    module procedure vspinor_product
  end interface
  private :: vspinor_product
```

$$\bar\psi^\mu\psi'_\mu \tag{X.3}$$

⟨*Implementation of operations for vectorspinors*⟩≡
```
  pure function vspinor_product (psil, psir) result (psilpsir)
    complex(kind=default) :: psilpsir
    type(vectorspinor), intent(in) :: psil, psir
    psilpsir = psil%psi(1) * psir%psi(1) &
             - psil%psi(2) * psir%psi(2) &
             - psil%psi(3) * psir%psi(3) &
             - psil%psi(4) * psir%psi(4)
  end function vspinor_product
```

X.3.1 Vectorspinor Vector Space

Scalar Multiplication

⟨*Declaration of operations for vectorspinors*⟩+≡
```
  interface operator (*)
     module procedure integer_vectorspinor, vectorspinor_integer, &
            real_vectorspinor, double_vectorspinor, &
            complex_vectorspinor, dcomplex_vectorspinor, &
            vectorspinor_real, vectorspinor_double, &
            vectorspinor_complex, vectorspinor_dcomplex, &
            momentum_vectorspinor, vectorspinor_momentum
  end interface
  private :: integer_vectorspinor, vectorspinor_integer, real_vectorspinor, &
       double_vectorspinor, complex_vectorspinor, dcomplex_vectorspinor, &
       vectorspinor_real, vectorspinor_double, vectorspinor_complex, &
       vectorspinor_dcomplex
```

⟨*Implementation of operations for vectorspinors*⟩+≡
```
  pure function integer_vectorspinor (x, y) result (xy)
    type(vectorspinor) :: xy
    integer, intent(in) :: x
    type(vectorspinor), intent(in) :: y
    integer :: k
    do k = 1,4
      xy%psi(k) = x * y%psi(k)
    end do
  end function integer_vectorspinor
```

⟨*Implementation of operations for vectorspinors*⟩+≡
```
  pure function real_vectorspinor (x, y) result (xy)
    type(vectorspinor) :: xy
    real(kind=single), intent(in) :: x
    type(vectorspinor), intent(in) :: y
    integer :: k
```

```
    do k = 1,4
    xy%psi(k) = x * y%psi(k)
    end do
  end function real_vectorspinor
```

⟨*Implementation of operations for vectorspinors*⟩+≡

```
  pure function double_vectorspinor (x, y) result (xy)
    type(vectorspinor) :: xy
    real(kind=default), intent(in) :: x
    type(vectorspinor), intent(in) :: y
    integer :: k
    do k = 1,4
    xy%psi(k) = x * y%psi(k)
    end do
  end function double_vectorspinor
```

⟨*Implementation of operations for vectorspinors*⟩+≡

```
  pure function complex_vectorspinor (x, y) result (xy)
    type(vectorspinor) :: xy
    complex(kind=single), intent(in) :: x
    type(vectorspinor), intent(in) :: y
    integer :: k
    do k = 1,4
    xy%psi(k) = x * y%psi(k)
    end do
  end function complex_vectorspinor
```

⟨*Implementation of operations for vectorspinors*⟩+≡

```
  pure function dcomplex_vectorspinor (x, y) result (xy)
    type(vectorspinor) :: xy
    complex(kind=default), intent(in) :: x
    type(vectorspinor), intent(in) :: y
    integer :: k
    do k = 1,4
    xy%psi(k) = x * y%psi(k)
    end do
  end function dcomplex_vectorspinor
```

⟨*Implementation of operations for vectorspinors*⟩+≡

```
  pure function vectorspinor_integer (y, x) result (xy)
    type(vectorspinor) :: xy
    integer, intent(in) :: x
    type(vectorspinor), intent(in) :: y
    integer :: k
    do k = 1,4
    xy%psi(k) = y%psi(k) * x
    end do
  end function vectorspinor_integer
```

⟨*Implementation of operations for vectorspinors*⟩+≡

```
  pure function vectorspinor_real (y, x) result (xy)
    type(vectorspinor) :: xy
    real(kind=single), intent(in) :: x
    type(vectorspinor), intent(in) :: y
    integer :: k
    do k = 1,4
    xy%psi(k) = y%psi(k) * x
```

```
    end do
  end function vectorspinor_real
```

⟨*Implementation of operations for vectorspinors*⟩+≡

```
  pure function vectorspinor_double (y, x) result (xy)
    type(vectorspinor) :: xy
    real(kind=default), intent(in) :: x
    type(vectorspinor), intent(in) :: y
    integer :: k
    do k = 1,4
    xy%psi(k) = y%psi(k) * x
    end do
  end function vectorspinor_double
```

⟨*Implementation of operations for vectorspinors*⟩+≡

```
  pure function vectorspinor_complex (y, x) result (xy)
    type(vectorspinor) :: xy
    complex(kind=single), intent(in) :: x
    type(vectorspinor), intent(in) :: y
    integer :: k
    do k = 1,4
    xy%psi(k) = y%psi(k) * x
    end do
  end function vectorspinor_complex
```

⟨*Implementation of operations for vectorspinors*⟩+≡

```
  pure function vectorspinor_dcomplex (y, x) result (xy)
    type(vectorspinor) :: xy
    complex(kind=default), intent(in) :: x
    type(vectorspinor), intent(in) :: y
    integer :: k
    do k = 1,4
    xy%psi(k) = y%psi(k) * x
    end do
  end function vectorspinor_dcomplex
```

⟨*Implementation of operations for vectorspinors*⟩+≡

```
  pure function momentum_vectorspinor (y, x) result (xy)
    type(bispinor) :: xy
    type(momentum), intent(in) :: y
    type(vectorspinor), intent(in) :: x
    integer :: k
    do k = 1,4
    xy%a(k) = y%t    * x%psi(1)%a(k) - y%x(1) * x%psi(2)%a(k) - &
            y%x(2) * x%psi(3)%a(k) - y%x(3) * x%psi(4)%a(k)
    end do
  end function momentum_vectorspinor
```

⟨*Implementation of operations for vectorspinors*⟩+≡

```
  pure function vectorspinor_momentum (y, x) result (xy)
    type(bispinor) :: xy
    type(momentum), intent(in) :: x
    type(vectorspinor), intent(in) :: y
    integer :: k
    do k = 1,4
    xy%a(k) = x%t    * y%psi(1)%a(k) - x%x(1) * y%psi(2)%a(k) - &
            x%x(2) * y%psi(3)%a(k) - x%x(3) * y%psi(4)%a(k)
```

```
    end do
  end function vectorspinor_momentum
```

Unary Plus and Minus

⟨*Declaration of operations for vectorspinors*⟩+≡

```
  interface operator (+)
     module procedure plus_vectorspinor
  end interface
  private :: plus_vectorspinor
  interface operator (-)
     module procedure neg_vectorspinor
  end interface
  private :: neg_vectorspinor
```

⟨*Implementation of operations for vectorspinors*⟩+≡

```
  pure function plus_vectorspinor (x) result (plus_x)
    type(vectorspinor) :: plus_x
    type(vectorspinor), intent(in) :: x
    integer :: k
    do k = 1,4
    plus_x%psi(k) = + x%psi(k)
    end do
  end function plus_vectorspinor
```

⟨*Implementation of operations for vectorspinors*⟩+≡

```
  pure function neg_vectorspinor (x) result (neg_x)
    type(vectorspinor) :: neg_x
    type(vectorspinor), intent(in) :: x
    integer :: k
    do k = 1,4
    neg_x%psi(k) = - x%psi(k)
    end do
  end function neg_vectorspinor
```

Addition and Subtraction

⟨*Declaration of operations for vectorspinors*⟩+≡

```
  interface operator (+)
     module procedure add_vectorspinor
  end interface
  private :: add_vectorspinor
  interface operator (-)
     module procedure sub_vectorspinor
  end interface
  private :: sub_vectorspinor
```

⟨*Implementation of operations for vectorspinors*⟩+≡

```
  pure function add_vectorspinor (x, y) result (xy)
    type(vectorspinor) :: xy
    type(vectorspinor), intent(in) :: x, y
    integer :: k
    do k = 1,4
    xy%psi(k) = x%psi(k) + y%psi(k)
    end do
  end function add_vectorspinor
```

⟨*Implementation of operations for vectorspinors*⟩+≡
```
pure function sub_vectorspinor (x, y) result (xy)
  type(vectorspinor) :: xy
  type(vectorspinor), intent(in) :: x, y
  integer :: k
  do k = 1,4
  xy%psi(k) = x%psi(k) - y%psi(k)
  end do
end function sub_vectorspinor
```

X.3.2 Norm

⟨*Declaration of operations for vectorspinors*⟩+≡
```
interface abs
   module procedure abs_vectorspinor
end interface
private :: abs_vectorspinor
```

⟨*Implementation of operations for vectorspinors*⟩+≡
```
pure function abs_vectorspinor (psi) result (x)
  real(kind=default) :: x
  type(vectorspinor), intent(in) :: psi
  x = sqrt (real (dot_product (psi%psi(1)%a, psi%psi(1)%a) &
          - dot_product (psi%psi(2)%a, psi%psi(2)%a)   &
          - dot_product (psi%psi(3)%a, psi%psi(3)%a)   &
          - dot_product (psi%psi(4)%a, psi%psi(4)%a)))
end function abs_vectorspinor
```

X.4 Vectors and Tensors

Condensed representation of antisymmetric rank-2 tensors:

$$\begin{pmatrix} T^{00} & T^{01} & T^{02} & T^{03} \\ T^{10} & T^{11} & T^{12} & T^{13} \\ T^{20} & T^{21} & T^{22} & T^{23} \\ T^{30} & T^{31} & T^{32} & T^{33} \end{pmatrix} = \begin{pmatrix} 0 & T_e^1 & T_e^2 & T_e^3 \\ -T_e^1 & 0 & T_b^3 & -T_b^2 \\ -T_e^2 & -T_b^3 & 0 & T_b^1 \\ -T_e^3 & T_b^2 & -T_b^1 & 0 \end{pmatrix} \tag{X.4}$$

⟨omega_vectors.f90⟩≡
```
⟨Copyleft⟩
module omega_vectors
  use kinds
  use constants
  implicit none
  private
  public :: assignment (=)
  public :: operator (*), operator (+), operator (-), operator (.wedge.)
  public :: abs, conjg
  public :: random_momentum
  ⟨intrinsic :: abs⟩
  ⟨intrinsic :: conjg⟩
  type, public :: momentum
     ! private (omegalib needs access, but DON'T TOUCH IT!)
     real(kind=default) :: t
```

```
      real(kind=default), dimension(3) :: x
   end type momentum
   type, public :: vector
      ! private (omegalib needs access, but DON'T TOUCH IT!)
      complex(kind=default) :: t
      complex(kind=default), dimension(3) :: x
   end type vector
   type, public :: tensor2odd
      ! private (omegalib needs access, but DON'T TOUCH IT!)
      complex(kind=default), dimension(3) :: e
      complex(kind=default), dimension(3) :: b
   end type tensor2odd
   ⟨Declaration of operations for vectors⟩
   integer, parameter, public :: omega_vectors_2010_01_A = 0
 contains
   ⟨Implementation of operations for vectors⟩
 end module omega_vectors
```

X.4.1 Constructors

⟨*Declaration of operations for vectors*⟩≡

```
 interface assignment (=)
    module procedure momentum_of_array, vector_of_momentum, &
         vector_of_array, vector_of_double_array, &
         array_of_momentum, array_of_vector
 end interface
 private :: momentum_of_array, vector_of_momentum, vector_of_array, &
      vector_of_double_array, array_of_momentum, array_of_vector
```

⟨*Implementation of operations for vectors*⟩≡

```
 pure subroutine momentum_of_array (m, p)
   type(momentum), intent(out) :: m
   real(kind=default), dimension(0:), intent(in) :: p
   m%t = p(0)
   m%x = p(1:3)
 end subroutine momentum_of_array
 pure subroutine array_of_momentum (p, v)
   real(kind=default), dimension(0:), intent(out) :: p
   type(momentum), intent(in) :: v
   p(0) = v%t
   p(1:3) = v%x
 end subroutine array_of_momentum
```

⟨*Implementation of operations for vectors*⟩+≡

```
 pure subroutine vector_of_array (v, p)
   type(vector), intent(out) :: v
   complex(kind=default), dimension(0:), intent(in) :: p
   v%t = p(0)
   v%x = p(1:3)
 end subroutine vector_of_array
 pure subroutine vector_of_double_array (v, p)
   type(vector), intent(out) :: v
   real(kind=default), dimension(0:), intent(in) :: p
   v%t = p(0)
   v%x = p(1:3)
```

```
  end subroutine vector_of_double_array
  pure subroutine array_of_vector (p, v)
    complex(kind=default), dimension(0:), intent(out) :: p
    type(vector), intent(in) :: v
    p(0) = v%t
    p(1:3) = v%x
  end subroutine array_of_vector
```

⟨*Implementation of operations for vectors*⟩+≡

```
  pure subroutine vector_of_momentum (v, p)
    type(vector), intent(out) :: v
    type(momentum), intent(in) :: p
    v%t = p%t
    v%x = p%x
  end subroutine vector_of_momentum
```

X.4.2 Inner Products

⟨*Declaration of operations for vectors*⟩+≡

```
  interface operator (*)
     module procedure momentum_momentum, vector_vector, &
          vector_momentum, momentum_vector, tensor2odd_tensor2odd
  end interface
  private :: momentum_momentum, vector_vector, vector_momentum, &
       momentum_vector, tensor2odd_tensor2odd
```

⟨*Implementation of operations for vectors*⟩+≡

```
  pure function momentum_momentum (x, y) result (xy)
    type(momentum), intent(in) :: x
    type(momentum), intent(in) :: y
    real(kind=default) :: xy
    xy = x%t*y%t - x%x(1)*y%x(1) - x%x(2)*y%x(2) - x%x(3)*y%x(3)
  end function momentum_momentum
  pure function momentum_vector (x, y) result (xy)
    type(momentum), intent(in) :: x
    type(vector), intent(in) :: y
    complex(kind=default) :: xy
    xy = x%t*y%t - x%x(1)*y%x(1) - x%x(2)*y%x(2) - x%x(3)*y%x(3)
  end function momentum_vector
  pure function vector_momentum (x, y) result (xy)
    type(vector), intent(in) :: x
    type(momentum), intent(in) :: y
    complex(kind=default) :: xy
    xy = x%t*y%t - x%x(1)*y%x(1) - x%x(2)*y%x(2) - x%x(3)*y%x(3)
  end function vector_momentum
  pure function vector_vector (x, y) result (xy)
    type(vector), intent(in) :: x
    type(vector), intent(in) :: y
    complex(kind=default) :: xy
    xy = x%t*y%t - x%x(1)*y%x(1) - x%x(2)*y%x(2) - x%x(3)*y%x(3)
  end function vector_vector
```

Just like classical electrodynamics:

$$\frac{1}{2}T_{\mu\nu}U^{\mu\nu} = \frac{1}{2}\left(-T^{0i}U^{0i} - T^{i0}U^{i0} + T^{ij}U^{ij}\right) = T_b^k U_b^k - T_e^k U_e^k \tag{X.5}$$

⟨*Implementation of operations for vectors*⟩+≡
```
pure function tensor2odd_tensor2odd (x, y) result (xy)
  type(tensor2odd), intent(in) :: x
  type(tensor2odd), intent(in) :: y
  complex(kind=default) :: xy
  xy = x%b(1)*y%b(1) + x%b(2)*y%b(2) + x%b(3)*y%b(3) &
     - x%e(1)*y%e(1) - x%e(2)*y%e(2) - x%e(3)*y%e(3)
end function tensor2odd_tensor2odd
```

X.4.3 Not Entirely Inner Products

⟨*Declaration of operations for vectors*⟩+≡
```
interface operator (*)
   module procedure momentum_tensor2odd, tensor2odd_momentum, &
        vector_tensor2odd, tensor2odd_vector
end interface
private :: momentum_tensor2odd, tensor2odd_momentum, vector_tensor2odd, &
     tensor2odd_vector
```

$$y^\nu = x_\mu T^{\mu\nu} : y^0 = -x^i T^{i0} = x^i T^{0i} \tag{X.6a}$$
$$y^1 = x^0 T^{01} - x^2 T^{21} - x^3 T^{31} \tag{X.6b}$$
$$y^2 = x^0 T^{02} - x^1 T^{12} - x^3 T^{32} \tag{X.6c}$$
$$y^3 = x^0 T^{03} - x^1 T^{13} - x^2 T^{23} \tag{X.6d}$$

⟨*Implementation of operations for vectors*⟩+≡
```
pure function vector_tensor2odd (x, t2) result (xt2)
  type(vector), intent(in) :: x
  type(tensor2odd), intent(in) :: t2
  type(vector) :: xt2
  xt2%t = x%x(1)*t2%e(1) + x%x(2)*t2%e(2) + x%x(3)*t2%e(3)
  xt2%x(1) = x%t*t2%e(1) + x%x(2)*t2%b(3) - x%x(3)*t2%b(2)
  xt2%x(2) = x%t*t2%e(2) + x%x(3)*t2%b(1) - x%x(1)*t2%b(3)
  xt2%x(3) = x%t*t2%e(3) + x%x(1)*t2%b(2) - x%x(2)*t2%b(1)
end function vector_tensor2odd
pure function momentum_tensor2odd (x, t2) result (xt2)
  type(momentum), intent(in) :: x
  type(tensor2odd), intent(in) :: t2
  type(vector) :: xt2
  xt2%t = x%x(1)*t2%e(1) + x%x(2)*t2%e(2) + x%x(3)*t2%e(3)
  xt2%x(1) = x%t*t2%e(1) + x%x(2)*t2%b(3) - x%x(3)*t2%b(2)
  xt2%x(2) = x%t*t2%e(2) + x%x(3)*t2%b(1) - x%x(1)*t2%b(3)
  xt2%x(3) = x%t*t2%e(3) + x%x(1)*t2%b(2) - x%x(2)*t2%b(1)
end function momentum_tensor2odd
```

$$y^\mu = T^{\mu\nu} x_\nu : y^0 = -T^{0i} x^i \tag{X.7a}$$
$$y^1 = T^{10} x^0 - T^{12} x^2 - T^{13} x^3 \tag{X.7b}$$
$$y^2 = T^{20} x^0 - T^{21} x^1 - T^{23} x^3 \tag{X.7c}$$
$$y^3 = T^{30} x^0 - T^{31} x^1 - T^{32} x^2 \tag{X.7d}$$

⟨*Implementation of operations for vectors*⟩+≡
```
  pure function tensor2odd_vector (t2, x) result (t2x)
    type(tensor2odd), intent(in) :: t2
    type(vector), intent(in) :: x
    type(vector) :: t2x
    t2x%t = - t2%e(1)*x%x(1) - t2%e(2)*x%x(2) - t2%e(3)*x%x(3)
    t2x%x(1) = - t2%e(1)*x%t + t2%b(2)*x%x(3) - t2%b(3)*x%x(2)
    t2x%x(2) = - t2%e(2)*x%t + t2%b(3)*x%x(1) - t2%b(1)*x%x(3)
    t2x%x(3) = - t2%e(3)*x%t + t2%b(1)*x%x(2) - t2%b(2)*x%x(1)
  end function tensor2odd_vector
  pure function tensor2odd_momentum (t2, x) result (t2x)
    type(tensor2odd), intent(in) :: t2
    type(momentum), intent(in) :: x
    type(vector) :: t2x
    t2x%t = - t2%e(1)*x%x(1) - t2%e(2)*x%x(2) - t2%e(3)*x%x(3)
    t2x%x(1) = - t2%e(1)*x%t + t2%b(2)*x%x(3) - t2%b(3)*x%x(2)
    t2x%x(2) = - t2%e(2)*x%t + t2%b(3)*x%x(1) - t2%b(1)*x%x(3)
    t2x%x(3) = - t2%e(3)*x%t + t2%b(1)*x%x(2) - t2%b(2)*x%x(1)
  end function tensor2odd_momentum
```

X.4.4 Outer Products

⟨*Declaration of operations for vectors*⟩+≡
```
  interface operator (.wedge.)
     module procedure momentum_wedge_momentum, &
          momentum_wedge_vector, vector_wedge_momentum, vector_wedge_vector
  end interface
  private :: momentum_wedge_momentum, momentum_wedge_vector, &
       vector_wedge_momentum, vector_wedge_vector
```

⟨*Implementation of operations for vectors*⟩+≡
```
  pure function momentum_wedge_momentum (x, y) result (t2)
    type(momentum), intent(in) :: x
    type(momentum), intent(in) :: y
    type(tensor2odd) :: t2
    t2%e = x%t * y%x - x%x * y%t
    t2%b(1) = x%x(2) * y%x(3) - x%x(3) * y%x(2)
    t2%b(2) = x%x(3) * y%x(1) - x%x(1) * y%x(3)
    t2%b(3) = x%x(1) * y%x(2) - x%x(2) * y%x(1)
  end function momentum_wedge_momentum
  pure function momentum_wedge_vector (x, y) result (t2)
    type(momentum), intent(in) :: x
    type(vector), intent(in) :: y
    type(tensor2odd) :: t2
    t2%e = x%t * y%x - x%x * y%t
    t2%b(1) = x%x(2) * y%x(3) - x%x(3) * y%x(2)
    t2%b(2) = x%x(3) * y%x(1) - x%x(1) * y%x(3)
    t2%b(3) = x%x(1) * y%x(2) - x%x(2) * y%x(1)
  end function momentum_wedge_vector
  pure function vector_wedge_momentum (x, y) result (t2)
    type(vector), intent(in) :: x
    type(momentum), intent(in) :: y
    type(tensor2odd) :: t2
    t2%e = x%t * y%x - x%x * y%t
```

```
    t2%b(1) = x%x(2) * y%x(3) - x%x(3) * y%x(2)
    t2%b(2) = x%x(3) * y%x(1) - x%x(1) * y%x(3)
    t2%b(3) = x%x(1) * y%x(2) - x%x(2) * y%x(1)
  end function vector_wedge_momentum
  pure function vector_wedge_vector (x, y) result (t2)
    type(vector), intent(in) :: x
    type(vector), intent(in) :: y
    type(tensor2odd) :: t2
    t2%e = x%t * y%x - x%x * y%t
    t2%b(1) = x%x(2) * y%x(3) - x%x(3) * y%x(2)
    t2%b(2) = x%x(3) * y%x(1) - x%x(1) * y%x(3)
    t2%b(3) = x%x(1) * y%x(2) - x%x(2) * y%x(1)
  end function vector_wedge_vector
```

X.4.5 Vector Space

Scalar Multiplication

⟨*Declaration of operations for vectors*⟩+≡

```
  interface operator (*)
     module procedure integer_momentum, real_momentum, double_momentum, &
          complex_momentum, dcomplex_momentum, &
          integer_vector, real_vector, double_vector, &
          complex_vector, dcomplex_vector, &
          integer_tensor2odd, real_tensor2odd, double_tensor2odd, &
          complex_tensor2odd, dcomplex_tensor2odd, &
          momentum_integer, momentum_real, momentum_double, &
          momentum_complex, momentum_dcomplex, &
          vector_integer, vector_real, vector_double, &
          vector_complex, vector_dcomplex, &
          tensor2odd_integer, tensor2odd_real, tensor2odd_double, &
          tensor2odd_complex, tensor2odd_dcomplex
  end interface
  private :: integer_momentum, real_momentum, double_momentum, &
       complex_momentum, dcomplex_momentum, integer_vector, real_vector, &
       double_vector, complex_vector, dcomplex_vector, &
       integer_tensor2odd, real_tensor2odd, double_tensor2odd, &
       complex_tensor2odd, dcomplex_tensor2odd, momentum_integer, &
       momentum_real, momentum_double, momentum_complex, &
       momentum_dcomplex, vector_integer, vector_real, vector_double, &
       vector_complex, vector_dcomplex, tensor2odd_integer, &
       tensor2odd_real, tensor2odd_double, tensor2odd_complex, &
       tensor2odd_dcomplex
```

⟨*Implementation of operations for vectors*⟩+≡

```
  pure function integer_momentum (x, y) result (xy)
    integer, intent(in) :: x
    type(momentum), intent(in) :: y
    type(momentum) :: xy
    xy%t = x * y%t
    xy%x = x * y%x
  end function integer_momentum
  pure function real_momentum (x, y) result (xy)
    real(kind=single), intent(in) :: x
```

```
    type(momentum), intent(in) :: y
    type(momentum) :: xy
    xy%t = x * y%t
    xy%x = x * y%x
  end function real_momentum
  pure function double_momentum (x, y) result (xy)
    real(kind=default), intent(in) :: x
    type(momentum), intent(in) :: y
    type(momentum) :: xy
    xy%t = x * y%t
    xy%x = x * y%x
  end function double_momentum
  pure function complex_momentum (x, y) result (xy)
    complex(kind=single), intent(in) :: x
    type(momentum), intent(in) :: y
    type(vector) :: xy
    xy%t = x * y%t
    xy%x = x * y%x
  end function complex_momentum
  pure function dcomplex_momentum (x, y) result (xy)
    complex(kind=default), intent(in) :: x
    type(momentum), intent(in) :: y
    type(vector) :: xy
    xy%t = x * y%t
    xy%x = x * y%x
  end function dcomplex_momentum
```

⟨*Implementation of operations for vectors*⟩+≡

```
  pure function integer_vector (x, y) result (xy)
    integer, intent(in) :: x
    type(vector), intent(in) :: y
    type(vector) :: xy
    xy%t = x * y%t
    xy%x = x * y%x
  end function integer_vector
  pure function real_vector (x, y) result (xy)
    real(kind=single), intent(in) :: x
    type(vector), intent(in) :: y
    type(vector) :: xy
    xy%t = x * y%t
    xy%x = x * y%x
  end function real_vector
  pure function double_vector (x, y) result (xy)
    real(kind=default), intent(in) :: x
    type(vector), intent(in) :: y
    type(vector) :: xy
    xy%t = x * y%t
    xy%x = x * y%x
  end function double_vector
  pure function complex_vector (x, y) result (xy)
    complex(kind=single), intent(in) :: x
    type(vector), intent(in) :: y
    type(vector) :: xy
    xy%t = x * y%t
    xy%x = x * y%x
```

```
  end function complex_vector
  pure function dcomplex_vector (x, y) result (xy)
    complex(kind=default), intent(in) :: x
    type(vector), intent(in) :: y
    type(vector) :: xy
    xy%t = x * y%t
    xy%x = x * y%x
  end function dcomplex_vector
```

⟨*Implementation of operations for vectors*⟩+≡

```
  pure function integer_tensor2odd (x, t2) result (xt2)
    integer, intent(in) :: x
    type(tensor2odd), intent(in) :: t2
    type(tensor2odd) :: xt2
    xt2%e = x * t2%e
    xt2%b = x * t2%b
  end function integer_tensor2odd
  pure function real_tensor2odd (x, t2) result (xt2)
    real(kind=single), intent(in) :: x
    type(tensor2odd), intent(in) :: t2
    type(tensor2odd) :: xt2
    xt2%e = x * t2%e
    xt2%b = x * t2%b
  end function real_tensor2odd
  pure function double_tensor2odd (x, t2) result (xt2)
    real(kind=default), intent(in) :: x
    type(tensor2odd), intent(in) :: t2
    type(tensor2odd) :: xt2
    xt2%e = x * t2%e
    xt2%b = x * t2%b
  end function double_tensor2odd
  pure function complex_tensor2odd (x, t2) result (xt2)
    complex(kind=single), intent(in) :: x
    type(tensor2odd), intent(in) :: t2
    type(tensor2odd) :: xt2
    xt2%e = x * t2%e
    xt2%b = x * t2%b
  end function complex_tensor2odd
  pure function dcomplex_tensor2odd (x, t2) result (xt2)
    complex(kind=default), intent(in) :: x
    type(tensor2odd), intent(in) :: t2
    type(tensor2odd) :: xt2
    xt2%e = x * t2%e
    xt2%b = x * t2%b
  end function dcomplex_tensor2odd
```

⟨*Implementation of operations for vectors*⟩+≡

```
  pure function momentum_integer (y, x) result (xy)
    integer, intent(in) :: x
    type(momentum), intent(in) :: y
    type(momentum) :: xy
    xy%t = x * y%t
    xy%x = x * y%x
  end function momentum_integer
  pure function momentum_real (y, x) result (xy)
```

```
    real(kind=single), intent(in) :: x
    type(momentum), intent(in) :: y
    type(momentum) :: xy
    xy%t = x * y%t
    xy%x = x * y%x
  end function momentum_real
  pure function momentum_double (y, x) result (xy)
    real(kind=default), intent(in) :: x
    type(momentum), intent(in) :: y
    type(momentum) :: xy
    xy%t = x * y%t
    xy%x = x * y%x
  end function momentum_double
  pure function momentum_complex (y, x) result (xy)
    complex(kind=single), intent(in) :: x
    type(momentum), intent(in) :: y
    type(vector) :: xy
    xy%t = x * y%t
    xy%x = x * y%x
  end function momentum_complex
  pure function momentum_dcomplex (y, x) result (xy)
    complex(kind=default), intent(in) :: x
    type(momentum), intent(in) :: y
    type(vector) :: xy
    xy%t = x * y%t
    xy%x = x * y%x
  end function momentum_dcomplex
```

⟨*Implementation of operations for vectors*⟩+≡

```
  pure function vector_integer (y, x) result (xy)
    integer, intent(in) :: x
    type(vector), intent(in) :: y
    type(vector) :: xy
    xy%t = x * y%t
    xy%x = x * y%x
  end function vector_integer
  pure function vector_real (y, x) result (xy)
    real(kind=single), intent(in) :: x
    type(vector), intent(in) :: y
    type(vector) :: xy
    xy%t = x * y%t
    xy%x = x * y%x
  end function vector_real
  pure function vector_double (y, x) result (xy)
    real(kind=default), intent(in) :: x
    type(vector), intent(in) :: y
    type(vector) :: xy
    xy%t = x * y%t
    xy%x = x * y%x
  end function vector_double
  pure function vector_complex (y, x) result (xy)
    complex(kind=single), intent(in) :: x
    type(vector), intent(in) :: y
    type(vector) :: xy
    xy%t = x * y%t
```

```
    xy%x = x * y%x
  end function vector_complex
  pure function vector_dcomplex (y, x) result (xy)
    complex(kind=default), intent(in) :: x
    type(vector), intent(in) :: y
    type(vector) :: xy
    xy%t = x * y%t
    xy%x = x * y%x
  end function vector_dcomplex
```

⟨*Implementation of operations for vectors*⟩+≡

```
  pure function tensor2odd_integer (t2, x) result (t2x)
    type(tensor2odd), intent(in) :: t2
    integer, intent(in) :: x
    type(tensor2odd) :: t2x
    t2x%e = x * t2%e
    t2x%b = x * t2%b
  end function tensor2odd_integer
  pure function tensor2odd_real (t2, x) result (t2x)
    type(tensor2odd), intent(in) :: t2
    real(kind=single), intent(in) :: x
    type(tensor2odd) :: t2x
    t2x%e = x * t2%e
    t2x%b = x * t2%b
  end function tensor2odd_real
  pure function tensor2odd_double (t2, x) result (t2x)
    type(tensor2odd), intent(in) :: t2
    real(kind=default), intent(in) :: x
    type(tensor2odd) :: t2x
    t2x%e = x * t2%e
    t2x%b = x * t2%b
  end function tensor2odd_double
  pure function tensor2odd_complex (t2, x) result (t2x)
    type(tensor2odd), intent(in) :: t2
    complex(kind=single), intent(in) :: x
    type(tensor2odd) :: t2x
    t2x%e = x * t2%e
    t2x%b = x * t2%b
  end function tensor2odd_complex
  pure function tensor2odd_dcomplex (t2, x) result (t2x)
    type(tensor2odd), intent(in) :: t2
    complex(kind=default), intent(in) :: x
    type(tensor2odd) :: t2x
    t2x%e = x * t2%e
    t2x%b = x * t2%b
  end function tensor2odd_dcomplex
```

Unary Plus and Minus

⟨*Declaration of operations for vectors*⟩+≡

```
  interface operator (+)
     module procedure plus_momentum, plus_vector, plus_tensor2odd
  end interface
  private :: plus_momentum, plus_vector, plus_tensor2odd
```

```
  interface operator (-)
     module procedure neg_momentum, neg_vector, neg_tensor2odd
  end interface
  private :: neg_momentum, neg_vector, neg_tensor2odd
```

⟨*Implementation of operations for vectors*⟩+≡

```
  pure function plus_momentum (x) result (plus_x)
    type(momentum), intent(in) :: x
    type(momentum) :: plus_x
    plus_x = x
  end function plus_momentum
  pure function neg_momentum (x) result (neg_x)
    type(momentum), intent(in) :: x
    type(momentum) :: neg_x
    neg_x%t = - x%t
    neg_x%x = - x%x
  end function neg_momentum
```

⟨*Implementation of operations for vectors*⟩+≡

```
  pure function plus_vector (x) result (plus_x)
    type(vector), intent(in) :: x
    type(vector) :: plus_x
    plus_x = x
  end function plus_vector
  pure function neg_vector (x) result (neg_x)
    type(vector), intent(in) :: x
    type(vector) :: neg_x
    neg_x%t = - x%t
    neg_x%x = - x%x
  end function neg_vector
```

⟨*Implementation of operations for vectors*⟩+≡

```
  pure function plus_tensor2odd (x) result (plus_x)
    type(tensor2odd), intent(in) :: x
    type(tensor2odd) :: plus_x
    plus_x = x
  end function plus_tensor2odd
  pure function neg_tensor2odd (x) result (neg_x)
    type(tensor2odd), intent(in) :: x
    type(tensor2odd) :: neg_x
    neg_x%e = - x%e
    neg_x%b = - x%b
  end function neg_tensor2odd
```

Addition and Subtraction

⟨*Declaration of operations for vectors*⟩+≡

```
  interface operator (+)
     module procedure add_momentum, add_vector, &
          add_vector_momentum, add_momentum_vector, add_tensor2odd
  end interface
  private :: add_momentum, add_vector, add_vector_momentum, &
       add_momentum_vector, add_tensor2odd
  interface operator (-)
     module procedure sub_momentum, sub_vector, &
```

```
       sub_vector_momentum, sub_momentum_vector, sub_tensor2odd
  end interface
  private :: sub_momentum, sub_vector, sub_vector_momentum, &
       sub_momentum_vector, sub_tensor2odd
```

⟨*Implementation of operations for vectors*⟩+≡

```
  pure function add_momentum (x, y) result (xy)
    type(momentum), intent(in) :: x, y
    type(momentum) :: xy
    xy%t = x%t + y%t
    xy%x = x%x + y%x
  end function add_momentum
  pure function add_vector (x, y) result (xy)
    type(vector), intent(in) :: x, y
    type(vector) :: xy
    xy%t = x%t + y%t
    xy%x = x%x + y%x
  end function add_vector
  pure function add_momentum_vector (x, y) result (xy)
    type(momentum), intent(in) :: x
    type(vector), intent(in) :: y
    type(vector) :: xy
    xy%t = x%t + y%t
    xy%x = x%x + y%x
  end function add_momentum_vector
  pure function add_vector_momentum (x, y) result (xy)
    type(vector), intent(in) :: x
    type(momentum), intent(in) :: y
    type(vector) :: xy
    xy%t = x%t + y%t
    xy%x = x%x + y%x
  end function add_vector_momentum
  pure function add_tensor2odd (x, y) result (xy)
    type(tensor2odd), intent(in) :: x, y
    type(tensor2odd) :: xy
    xy%e = x%e + y%e
    xy%b = x%b + y%b
  end function add_tensor2odd
```

⟨*Implementation of operations for vectors*⟩+≡

```
  pure function sub_momentum (x, y) result (xy)
    type(momentum), intent(in) :: x, y
    type(momentum) :: xy
    xy%t = x%t - y%t
    xy%x = x%x - y%x
  end function sub_momentum
  pure function sub_vector (x, y) result (xy)
    type(vector), intent(in) :: x, y
    type(vector) :: xy
    xy%t = x%t - y%t
    xy%x = x%x - y%x
  end function sub_vector
  pure function sub_momentum_vector (x, y) result (xy)
    type(momentum), intent(in) :: x
    type(vector), intent(in) :: y
```

```
    type(vector) :: xy
    xy%t = x%t - y%t
    xy%x = x%x - y%x
  end function sub_momentum_vector
  pure function sub_vector_momentum (x, y) result (xy)
    type(vector), intent(in) :: x
    type(momentum), intent(in) :: y
    type(vector) :: xy
    xy%t = x%t - y%t
    xy%x = x%x - y%x
  end function sub_vector_momentum
  pure function sub_tensor2odd (x, y) result (xy)
    type(tensor2odd), intent(in) :: x, y
    type(tensor2odd) :: xy
    xy%e = x%e - y%e
    xy%b = x%b - y%b
  end function sub_tensor2odd
```

X.4.6 Norm

Not the covariant length!

⟨*Declaration of operations for vectors*⟩+≡

```
  interface abs
     module procedure abs_momentum, abs_vector, abs_tensor2odd
  end interface
  private :: abs_momentum, abs_vector, abs_tensor2odd
```

⟨*Implementation of operations for vectors*⟩+≡

```
  pure function abs_momentum (x) result (absx)
    type(momentum), intent(in) :: x
    real(kind=default) :: absx
    absx = sqrt (real (x%t*x%t + dot_product (x%x, x%x)))
  end function abs_momentum
  pure function abs_vector (x) result (absx)
    type(vector), intent(in) :: x
    real(kind=default) :: absx
    absx = sqrt (real (conjg(x%t)*x%t + dot_product (x%x, x%x)))
  end function abs_vector
  pure function abs_tensor2odd (x) result (absx)
    type(tensor2odd), intent(in) :: x
    real(kind=default) :: absx
    absx = sqrt (real (dot_product (x%e, x%e) + dot_product (x%b, x%b)))
  end function abs_tensor2odd
```

X.4.7 Conjugation

⟨*Declaration of operations for vectors*⟩+≡

```
  interface conjg
     module procedure conjg_momentum, conjg_vector, conjg_tensor2odd
  end interface
  private :: conjg_momentum, conjg_vector, conjg_tensor2odd
```

⟨*Implementation of operations for vectors*⟩+≡

```
  pure function conjg_momentum (x) result (conjg_x)
```

```
    type(momentum), intent(in) :: x
    type(momentum) :: conjg_x
    conjg_x = x
  end function conjg_momentum
  pure function conjg_vector (x) result (conjg_x)
    type(vector), intent(in) :: x
    type(vector) :: conjg_x
    conjg_x%t = conjg (x%t)
    conjg_x%x = conjg (x%x)
  end function conjg_vector
  pure function conjg_tensor2odd (t2) result (conjg_t2)
    type(tensor2odd), intent(in) :: t2
    type(tensor2odd) :: conjg_t2
    conjg_t2%e = conjg (t2%e)
    conjg_t2%b = conjg (t2%b)
  end function conjg_tensor2odd
```

X.4.8 ϵ-Tensors

$$\epsilon_{0123} = 1 = -\epsilon^{0123} \tag{X.8}$$

in particular

$$\epsilon(p_1, p_2, p_3, p_4) = \epsilon_{\mu_1\mu_2\mu_3\mu_4} p_1^{\mu_1} p_2^{\mu_2} p_3^{\mu_3} p_4^{\mu_4} = p_1^0 p_2^1 p_3^2 p_4^3 \pm \ldots \tag{X.9}$$

⟨*Declaration of operations for vectors*⟩+≡

```
  interface pseudo_scalar
     module procedure pseudo_scalar_momentum, pseudo_scalar_vector, &
          pseudo_scalar_vec_mom
  end interface
  public :: pseudo_scalar
  private :: pseudo_scalar_momentum, pseudo_scalar_vector
```

⟨*Implementation of operations for vectors*⟩+≡

```
  pure function pseudo_scalar_momentum (p1, p2, p3, p4) result (eps1234)
    type(momentum), intent(in) :: p1, p2, p3, p4
    real(kind=default) :: eps1234
    eps1234 = &
         p1%t    * p2%x(1) * (p3%x(2) * p4%x(3) - p3%x(3) * p4%x(2)) &
       + p1%t    * p2%x(2) * (p3%x(3) * p4%x(1) - p3%x(1) * p4%x(3)) &
       + p1%t    * p2%x(3) * (p3%x(1) * p4%x(2) - p3%x(2) * p4%x(1)) &
       - p1%x(1) * p2%x(2) * (p3%x(3) * p4%t    - p3%t    * p4%x(3)) &
       - p1%x(1) * p2%x(3) * (p3%t    * p4%x(2) - p3%x(2) * p4%t   ) &
       - p1%x(1) * p2%t    * (p3%x(2) * p4%x(3) - p3%x(3) * p4%x(2)) &
       + p1%x(2) * p2%x(3) * (p3%t    * p4%x(1) - p3%x(1) * p4%t   ) &
       + p1%x(2) * p2%t    * (p3%x(1) * p4%x(3) - p3%x(3) * p4%x(1)) &
       + p1%x(2) * p2%x(1) * (p3%x(3) * p4%t    - p3%t    * p4%x(3)) &
       - p1%x(3) * p2%t    * (p3%x(1) * p4%x(2) - p3%x(2) * p4%x(1)) &
       - p1%x(3) * p2%x(1) * (p3%x(2) * p4%t    - p3%t    * p4%x(2)) &
       - p1%x(3) * p2%x(2) * (p3%t    * p4%x(1) - p3%x(1) * p4%t   )
  end function pseudo_scalar_momentum
```

⟨*Implementation of operations for vectors*⟩+≡

```
  pure function pseudo_scalar_vector (p1, p2, p3, p4) result (eps1234)
    type(vector), intent(in) :: p1, p2, p3, p4
```

```
    complex(kind=default) :: eps1234
    eps1234 = &
         p1%t    * p2%x(1) * (p3%x(2) * p4%x(3) - p3%x(3) * p4%x(2)) &
       + p1%t    * p2%x(2) * (p3%x(3) * p4%x(1) - p3%x(1) * p4%x(3)) &
       + p1%t    * p2%x(3) * (p3%x(1) * p4%x(2) - p3%x(2) * p4%x(1)) &
       - p1%x(1) * p2%x(2) * (p3%x(3) * p4%t    - p3%t    * p4%x(3)) &
       - p1%x(1) * p2%x(3) * (p3%t    * p4%x(2) - p3%x(2) * p4%t   ) &
       - p1%x(1) * p2%t    * (p3%x(2) * p4%x(3) - p3%x(3) * p4%x(2)) &
       + p1%x(2) * p2%x(3) * (p3%t    * p4%x(1) - p3%x(1) * p4%t   ) &
       + p1%x(2) * p2%t    * (p3%x(1) * p4%x(3) - p3%x(3) * p4%x(1)) &
       + p1%x(2) * p2%x(1) * (p3%x(3) * p4%t    - p3%t    * p4%x(3)) &
       - p1%x(3) * p2%t    * (p3%x(1) * p4%x(2) - p3%x(2) * p4%x(1)) &
       - p1%x(3) * p2%x(1) * (p3%x(2) * p4%t    - p3%t    * p4%x(2)) &
       - p1%x(3) * p2%x(2) * (p3%t    * p4%x(1) - p3%x(1) * p4%t   )
  end function pseudo_scalar_vector
```

⟨*Implementation of operations for vectors*⟩+≡

```
  pure function pseudo_scalar_vec_mom (p1, v1, p2, v2) result (eps1234)
    type(momentum), intent(in)   :: p1, p2
    type(vector), intent(in) :: v1, v2
    complex(kind=default) :: eps1234
    eps1234 = &
         p1%t    * v1%x(1) * (p2%x(2) * v2%x(3) - p2%x(3) * v2%x(2)) &
       + p1%t    * v1%x(2) * (p2%x(3) * v2%x(1) - p2%x(1) * v2%x(3)) &
       + p1%t    * v1%x(3) * (p2%x(1) * v2%x(2) - p2%x(2) * v2%x(1)) &
       - p1%x(1) * v1%x(2) * (p2%x(3) * v2%t    - p2%t    * v2%x(3)) &
       - p1%x(1) * v1%x(3) * (p2%t    * v2%x(2) - p2%x(2) * v2%t   ) &
       - p1%x(1) * v1%t    * (p2%x(2) * v2%x(3) - p2%x(3) * v2%x(2)) &
       + p1%x(2) * v1%x(3) * (p2%t    * v2%x(1) - p2%x(1) * v2%t   ) &
       + p1%x(2) * v1%t    * (p2%x(1) * v2%x(3) - p2%x(3) * v2%x(1)) &
       + p1%x(2) * v1%x(1) * (p2%x(3) * v2%t    - p2%t    * v2%x(3)) &
       - p1%x(3) * v1%t    * (p2%x(1) * v2%x(2) - p2%x(2) * v2%x(1)) &
       - p1%x(3) * v1%x(1) * (p2%x(2) * v2%t    - p2%t    * v2%x(2)) &
       - p1%x(3) * v1%x(2) * (p2%t    * v2%x(1) - p2%x(1) * v2%t   )
  end function pseudo_scalar_vec_mom
```

$$\epsilon_\mu(p_1,p_2,p_3) = \epsilon_{\mu\mu_1\mu_2\mu_3} p_1^{\mu_1} p_2^{\mu_2} p_3^{\mu_3} \tag{X.10}$$

i. e.

$$\epsilon_0(p_1,p_2,p_3) = p_1^1 p_2^2 p_3^3 \pm \ldots \tag{X.11a}$$
$$\epsilon_1(p_1,p_2,p_3) = p_1^2 p_2^3 p_3^0 \pm \ldots \tag{X.11b}$$
$$\epsilon_2(p_1,p_2,p_3) = -p_1^3 p_2^0 p_3^1 \pm \ldots \tag{X.11c}$$
$$\epsilon_3(p_1,p_2,p_3) = p_1^0 p_2^1 p_3^2 \pm \ldots \tag{X.11d}$$

⟨*Declaration of operations for vectors*⟩+≡

```
  interface pseudo_vector
     module procedure pseudo_vector_momentum, pseudo_vector_vector, &
          pseudo_vector_vec_mom
  end interface
  public :: pseudo_vector
  private :: pseudo_vector_momentum, pseudo_vector_vector
```

⟨*Implementation of operations for vectors*⟩+≡

```
  pure function pseudo_vector_momentum (p1, p2, p3) result (eps123)
    type(momentum), intent(in) :: p1, p2, p3
    type(momentum) :: eps123
    eps123%t = &
      + p1%x(1) * (p2%x(2) * p3%x(3) - p2%x(3) * p3%x(2)) &
      + p1%x(2) * (p2%x(3) * p3%x(1) - p2%x(1) * p3%x(3)) &
      + p1%x(3) * (p2%x(1) * p3%x(2) - p2%x(2) * p3%x(1))
    eps123%x(1) = &
      + p1%x(2) * (p2%x(3) * p3%t    - p2%t    * p3%x(3)) &
      + p1%x(3) * (p2%t    * p3%x(2) - p2%x(2) * p3%t   ) &
      + p1%t    * (p2%x(2) * p3%x(3) - p2%x(3) * p3%x(2))
    eps123%x(2) = &
      - p1%x(3) * (p2%t    * p3%x(1) - p2%x(1) * p3%t   ) &
      - p1%t    * (p2%x(1) * p3%x(3) - p2%x(3) * p3%x(1)) &
      - p1%x(1) * (p2%x(3) * p3%t    - p2%t    * p3%x(3))
    eps123%x(3) =  &
      + p1%t    * (p2%x(1) * p3%x(2) - p2%x(2) * p3%x(1)) &
      + p1%x(1) * (p2%x(2) * p3%t    - p2%t    * p3%x(2)) &
      + p1%x(2) * (p2%t    * p3%x(1) - p2%x(1) * p3%t   )
  end function pseudo_vector_momentum
```

⟨*Implementation of operations for vectors*⟩+≡

```
  pure function pseudo_vector_vector (p1, p2, p3) result (eps123)
    type(vector), intent(in) :: p1, p2, p3
    type(vector) :: eps123
    eps123%t = &
      + p1%x(1) * (p2%x(2) * p3%x(3) - p2%x(3) * p3%x(2)) &
      + p1%x(2) * (p2%x(3) * p3%x(1) - p2%x(1) * p3%x(3)) &
      + p1%x(3) * (p2%x(1) * p3%x(2) - p2%x(2) * p3%x(1))
    eps123%x(1) = &
      + p1%x(2) * (p2%x(3) * p3%t    - p2%t    * p3%x(3)) &
      + p1%x(3) * (p2%t    * p3%x(2) - p2%x(2) * p3%t   ) &
      + p1%t    * (p2%x(2) * p3%x(3) - p2%x(3) * p3%x(2))
    eps123%x(2) = &
      - p1%x(3) * (p2%t    * p3%x(1) - p2%x(1) * p3%t   ) &
      - p1%t    * (p2%x(1) * p3%x(3) - p2%x(3) * p3%x(1)) &
      - p1%x(1) * (p2%x(3) * p3%t    - p2%t    * p3%x(3))
    eps123%x(3) =  &
      + p1%t    * (p2%x(1) * p3%x(2) - p2%x(2) * p3%x(1)) &
      + p1%x(1) * (p2%x(2) * p3%t    - p2%t    * p3%x(2)) &
      + p1%x(2) * (p2%t    * p3%x(1) - p2%x(1) * p3%t   )
  end function pseudo_vector_vector
```

⟨*Implementation of operations for vectors*⟩+≡

```
  pure function pseudo_vector_vec_mom (p1, p2, v) result (eps123)
    type(momentum), intent(in) :: p1, p2
    type(vector), intent(in)   :: v
    type(vector) :: eps123
    eps123%t = &
      + p1%x(1) * (p2%x(2) * v%x(3) - p2%x(3) * v%x(2)) &
      + p1%x(2) * (p2%x(3) * v%x(1) - p2%x(1) * v%x(3)) &
      + p1%x(3) * (p2%x(1) * v%x(2) - p2%x(2) * v%x(1))
    eps123%x(1) = &
      + p1%x(2) * (p2%x(3) * v%t    - p2%t    * v%x(3)) &
```

```
      + p1%x(3) * (p2%t    * v%x(2) - p2%x(2) * v%t   ) &
      + p1%t    * (p2%x(2) * v%x(3) - p2%x(3) * v%x(2))
    eps123%x(2) = &
      - p1%x(3) * (p2%t    * v%x(1) - p2%x(1) * v%t   ) &
      - p1%t    * (p2%x(1) * v%x(3) - p2%x(3) * v%x(1)) &
      - p1%x(1) * (p2%x(3) * v%t    - p2%t    * v%x(3))
    eps123%x(3) =  &
      + p1%t    * (p2%x(1) * v%x(2) - p2%x(2) * v%x(1)) &
      + p1%x(1) * (p2%x(2) * v%t    - p2%t    * v%x(2)) &
      + p1%x(2) * (p2%t    * v%x(1) - p2%x(1) * v%t   )
  end function pseudo_vector_vec_mom
```

X.4.9 Utilities

⟨*Declaration of operations for vectors*⟩+≡

⟨*Implementation of operations for vectors*⟩+≡

```
  subroutine random_momentum (p, pabs, m)
    type(momentum), intent(out) :: p
    real(kind=default), intent(in) :: pabs, m
    real(kind=default), dimension(2) :: r
    real(kind=default) :: phi, cos_th
    call random_number (r)
    phi = 2*PI * r(1)
    cos_th = 2 * r(2) - 1
    p%t = sqrt (pabs**2 + m**2)
    p%x = pabs * (/ cos_th * cos(phi), cos_th * sin(phi), sqrt (1 - cos_th**2) /)
  end subroutine random_momentum
```

X.5 Polarization vectors

⟨omega_polarizations.f90⟩≡

```
  ⟨Copyleft⟩
  module omega_polarizations
    use kinds
    use constants
    use omega_vectors
    implicit none
    private
    ⟨Declaration of polarization vectors⟩
    integer, parameter, public :: omega_polarizations_2010_01_A = 0
  contains
    ⟨Implementation of polarization vectors⟩
  end module omega_polarizations
```

Here we use a phase convention for the polarization vectors compatible with the angular momentum coupling to spin 3/2 and spin 2.

$$\epsilon_1^\mu(k) = \frac{1}{|\vec k|\sqrt{k_x^2+k_y^2}}\left(0; k_zk_x, k_yk_z, -k_x^2-k_y^2\right) \tag{X.12a}$$

$$\epsilon_2^\mu(k) = \frac{1}{\sqrt{k_x^2+k_y^2}}\left(0; -k_y, k_x, 0\right) \tag{X.12b}$$

$$\epsilon_3^\mu(k) = \frac{k_0}{m|\vec{k}|}\left(\vec{k}^2/k_0; k_x, k_y, k_z\right) \tag{X.12c}$$

and

$$\epsilon_\pm^\mu(k) = \frac{1}{\sqrt{2}}(\epsilon_1^\mu(k) \pm \mathrm{i}\epsilon_2^\mu(k)) \tag{X.13a}$$

$$\epsilon_0^\mu(k) = \epsilon_3^\mu(k) \tag{X.13b}$$

i. e.

$$\epsilon_+^\mu(k) = \frac{1}{\sqrt{2}\sqrt{k_x^2+k_y^2}}\left(0; \frac{k_z k_x}{|\vec{k}|} - \mathrm{i}k_y, \frac{k_y k_z}{|\vec{k}|} + \mathrm{i}k_x, -\frac{k_x^2+k_y^2}{|\vec{k}|}\right) \tag{X.14a}$$

$$\epsilon_-^\mu(k) = \frac{1}{\sqrt{2}\sqrt{k_x^2+k_y^2}}\left(0; \frac{k_z k_x}{|\vec{k}|} + \mathrm{i}k_y, \frac{k_y k_z}{|\vec{k}|} - \mathrm{i}k_x, -\frac{k_x^2+k_y^2}{|\vec{k}|}\right) \tag{X.14b}$$

$$\epsilon_0^\mu(k) = \frac{k_0}{m|\vec{k}|}\left(\vec{k}^2/k_0; k_x, k_y, k_z\right) \tag{X.14c}$$

Determining the mass from the momenta is a numerically haphazardous for light particles. Therefore, we accept some redundancy and pass the mass explicitely.

⟨*Declaration of polarization vectors*⟩≡
```
public :: eps
```

⟨*Implementation of polarization vectors*⟩≡
```
pure function eps (m, k, s) result (e)
  type(vector) :: e
  real(kind=default), intent(in) :: m
  type(momentum), intent(in) :: k
  integer, intent(in) :: s
  real(kind=default) :: kt, kabs, kabs2, sqrt2
  sqrt2 = sqrt (2.0_default)
  kabs2 = dot_product (k%x, k%x)
  e%t = 0
  e%x = 0
  if (kabs2 > 0) then
     kabs = sqrt (kabs2)
     select case (s)
     case (1)
        kt = sqrt (k%x(1)**2 + k%x(2)**2)
        if (abs(kt) <= epsilon(kt) * kabs) then
           if (k%x(3) > 0) then
              e%x(1) = cmplx (   1,   0, kind=default) / sqrt2
              e%x(2) = cmplx (   0,   1, kind=default) / sqrt2
           else
              e%x(1) = cmplx ( - 1,   0, kind=default) / sqrt2
              e%x(2) = cmplx (   0,   1, kind=default) / sqrt2
           end if
        else
           e%x(1) = cmplx (   k%x(3)*k%x(1)/kabs, &
                - k%x(2), kind=default) / kt / sqrt2
           e%x(2) = cmplx (   k%x(2)*k%x(3)/kabs, &
                k%x(1), kind=default) / kt / sqrt2
```

```
         e%x(3) = - kt / kabs / sqrt2
      end if
   case (-1)
      kt = sqrt (k%x(1)**2 + k%x(2)**2)
      if (abs(kt) <= epsilon(kt) * kabs) then
         if (k%x(3) > 0) then
            e%x(1) = cmplx (   1,   0, kind=default) / sqrt2
            e%x(2) = cmplx (   0, - 1, kind=default) / sqrt2
         else
            e%x(1) = cmplx (  -1,   0, kind=default) / sqrt2
            e%x(2) = cmplx (   0, - 1, kind=default) / sqrt2
         end if
      else
         e%x(1) = cmplx (   k%x(3)*k%x(1)/kabs, &
              k%x(2), kind=default) / kt / sqrt2
         e%x(2) = cmplx (   k%x(2)*k%x(3)/kabs, &
              - k%x(1), kind=default) / kt / sqrt2
         e%x(3) = - kt / kabs / sqrt2
      end if
   case (0)
      if (m > 0) then
         e%t = kabs / m
         e%x = k%t / (m*kabs) * k%x
      end if
   case (3)
      e = (0,1) * k
   case (4)
      if (m > 0) then
         e = (1 / m) * k
      else
         e = (1 / k%t) * k
      end if
   end select
else   !!! for particles in their rest frame defined to be
       !!! polarized along the 3-direction
   select case (s)
   case (1)
      e%x(1) = cmplx (   1,   0, kind=default) / sqrt2
      e%x(2) = cmplx (   0,   1, kind=default) / sqrt2
   case (-1)
      e%x(1) = cmplx (   1,   0, kind=default) / sqrt2
      e%x(2) = cmplx (   0, - 1, kind=default) / sqrt2
   case (0)
      if (m > 0) then
         e%x(3) = 1
      end if
   case (4)
      if (m > 0) then
         e = (1 / m) * k
      else
         e = (1 / k%t) * k
      end if
   end select
end if
```

```
end function eps
```

X.6 Polarization vectors revisited

⟨omega_polarizations_madgraph.f90⟩≡
```
⟨Copyleft⟩
module omega_polarizations_madgraph
  use kinds
  use constants
  use omega_vectors
  implicit none
  private
  ⟨Declaration of polarization vectors for madgraph⟩
  integer, parameter, public :: omega_pols_madgraph_2010_01_A = 0
contains
  ⟨Implementation of polarization vectors for madgraph⟩
end module omega_polarizations_madgraph
```

This set of polarization vectors is compatible with HELAS [5]:

$$\epsilon_1^\mu(k) = \frac{1}{|\vec{k}|\sqrt{k_x^2+k_y^2}}\left(0; k_z k_x, k_y k_z, -k_x^2-k_y^2\right) \tag{X.15a}$$

$$\epsilon_2^\mu(k) = \frac{1}{\sqrt{k_x^2+k_y^2}}\left(0; -k_y, k_x, 0\right) \tag{X.15b}$$

$$\epsilon_3^\mu(k) = \frac{k_0}{m|\vec{k}|}\left(\vec{k}^2/k_0; k_x, k_y, k_z\right) \tag{X.15c}$$

and

$$\epsilon_\pm^\mu(k) = \frac{1}{\sqrt{2}}(\mp\epsilon_1^\mu(k) - \mathrm{i}\epsilon_2^\mu(k)) \tag{X.16a}$$

$$\epsilon_0^\mu(k) = \epsilon_3^\mu(k) \tag{X.16b}$$

i. e.

$$\epsilon_+^\mu(k) = \frac{1}{\sqrt{2}\sqrt{k_x^2+k_y^2}}\left(0; -\frac{k_z k_x}{|\vec{k}|} + \mathrm{i}k_y, -\frac{k_y k_z}{|\vec{k}|} - \mathrm{i}k_x, \frac{k_x^2+k_y^2}{|\vec{k}|}\right) \tag{X.17a}$$

$$\epsilon_-^\mu(k) = \frac{1}{\sqrt{2}\sqrt{k_x^2+k_y^2}}\left(0; \frac{k_z k_x}{|\vec{k}|} + \mathrm{i}k_y, \frac{k_y k_z}{|\vec{k}|} - \mathrm{i}k_x, -\frac{k_x^2+k_y^2}{|\vec{k}|}\right) \tag{X.17b}$$

$$\epsilon_0^\mu(k) = \frac{k_0}{m|\vec{k}|}\left(\vec{k}^2/k_0; k_x, k_y, k_z\right) \tag{X.17c}$$

Fortunately, for comparing with squared matrix generated by Madgraph we can also use the modified version, since the difference is only a phase and does *not* mix helicity states. Determining the mass from the momenta is a numerically haphazardous for light particles. Therefore, we accept some redundancy and pass the mass explicitely.

⟨*Declaration of polarization vectors for madgraph*⟩≡
```
public :: eps
```

⟨*Implementation of polarization vectors for madgraph*⟩≡

```
  pure function eps (m, k, s) result (e)
    type(vector) :: e
    real(kind=default), intent(in) :: m
    type(momentum), intent(in) :: k
    integer, intent(in) :: s
    real(kind=default) :: kt, kabs, kabs2, sqrt2
    sqrt2 = sqrt (2.0_default)
    kabs2 = dot_product (k%x, k%x)
    e%t = 0
    e%x = 0
    if (kabs2 > 0) then
       kabs = sqrt (kabs2)
       select case (s)
       case (1)
          kt = sqrt (k%x(1)**2 + k%x(2)**2)
          if (abs(kt) <= epsilon(kt) * kabs) then
             if (k%x(3) > 0) then
                e%x(1) = cmplx ( - 1,   0, kind=default) / sqrt2
                e%x(2) = cmplx (   0, - 1, kind=default) / sqrt2
             else
                e%x(1) = cmplx (   1,   0, kind=default) / sqrt2
                e%x(2) = cmplx (   0, - 1, kind=default) / sqrt2
             end if
          else
             e%x(1) = cmplx ( - k%x(3)*k%x(1)/kabs, &
                  k%x(2), kind=default) / kt / sqrt2
             e%x(2) = cmplx ( - k%x(2)*k%x(3)/kabs, &
                  - k%x(1), kind=default) / kt / sqrt2
             e%x(3) = kt / kabs / sqrt2
          end if
       case (-1)
          kt = sqrt (k%x(1)**2 + k%x(2)**2)
          if (abs(kt) <= epsilon(kt) * kabs) then
             if (k%x(3) > 0) then
                e%x(1) = cmplx (   1,   0, kind=default) / sqrt2
                e%x(2) = cmplx (   0, - 1, kind=default) / sqrt2
             else
                e%x(1) = cmplx (  -1,   0, kind=default) / sqrt2
                e%x(2) = cmplx (   0, - 1, kind=default) / sqrt2
             end if
          else
             e%x(1) = cmplx (   k%x(3)*k%x(1)/kabs, &
                  k%x(2), kind=default) / kt / sqrt2
             e%x(2) = cmplx (   k%x(2)*k%x(3)/kabs, &
                  - k%x(1), kind=default) / kt / sqrt2
             e%x(3) = - kt / kabs / sqrt2
          end if
       case (0)
          if (m > 0) then
             e%t = kabs / m
             e%x = k%t / (m*kabs) * k%x
          end if
       case (3)
```

```
        e = (0,1) * k
      case (4)
        if (m > 0) then
          e = (1 / m) * k
        else
          e = (1 / k%t) * k
        end if
      end select
    else   !!! for particles in their rest frame defined to be
           !!! polarized along the 3-direction
      select case (s)
      case (1)
        e%x(1) = cmplx ( - 1,   0, kind=default) / sqrt2
        e%x(2) = cmplx (   0, - 1, kind=default) / sqrt2
      case (-1)
        e%x(1) = cmplx (   1,   0, kind=default) / sqrt2
        e%x(2) = cmplx (   0, - 1, kind=default) / sqrt2
      case (0)
        if (m > 0) then
          e%x(3) = 1
        end if
      case (4)
        if (m > 0) then
          e = (1 / m) * k
        else
          e = (1 / k%t) * k
        end if
      end select
    end if
  end function eps
```

X.7 Symmetric Tensors

Spin-2 polarization tensors are symmetric, transversal and traceless

$$\epsilon_m^{\mu\nu}(k) = \epsilon_m^{\nu\mu}(k) \tag{X.18a}$$

$$k_\mu \epsilon_m^{\mu\nu}(k) = k_\nu \epsilon_m^{\mu\nu}(k) = 0 \tag{X.18b}$$

$$\epsilon_{m,\mu}^{\mu}(k) = 0 \tag{X.18c}$$

with $m = 1, 2, 3, 4, 5$. Our current representation is redundant and does *not* enforce symmetry or tracelessness.

```
⟨omega_tensors.f90⟩≡
  ⟨Copyleft⟩
  module omega_tensors
    use kinds
    use constants
    use omega_vectors
    implicit none
    private
    public :: operator (*), operator (+), operator (-), &
         operator (.tprod.)
    public :: abs, conjg
```

```
  ⟨intrinsic :: abs⟩
  ⟨intrinsic :: conjg⟩
  type, public :: tensor
     ! private (omegalib needs access, but DON'T TOUCH IT!)
     complex(kind=default), dimension(0:3,0:3) :: t
  end type tensor
  ⟨Declaration of operations for tensors⟩
  integer, parameter, public :: omega_tensors_2010_01_A = 0
contains
  ⟨Implementation of operations for tensors⟩
end module omega_tensors
```

X.7.1 Vector Space

Scalar Multliplication

⟨*Declaration of operations for tensors*⟩≡

```
interface operator (*)
   module procedure integer_tensor, real_tensor, double_tensor, &
        complex_tensor, dcomplex_tensor
end interface
private :: integer_tensor, real_tensor, double_tensor
private :: complex_tensor, dcomplex_tensor
```

⟨*Implementation of operations for tensors*⟩≡

```
pure function integer_tensor (x, y) result (xy)
  integer, intent(in) :: x
  type(tensor), intent(in) :: y
  type(tensor) :: xy
  xy%t = x * y%t
end function integer_tensor
pure function real_tensor (x, y) result (xy)
  real(kind=single), intent(in) :: x
  type(tensor), intent(in) :: y
  type(tensor) :: xy
  xy%t = x * y%t
end function real_tensor
pure function double_tensor (x, y) result (xy)
  real(kind=default), intent(in) :: x
  type(tensor), intent(in) :: y
  type(tensor) :: xy
  xy%t = x * y%t
end function double_tensor
pure function complex_tensor (x, y) result (xy)
  complex(kind=single), intent(in) :: x
  type(tensor), intent(in) :: y
  type(tensor) :: xy
  xy%t = x * y%t
end function complex_tensor
pure function dcomplex_tensor (x, y) result (xy)
  complex(kind=default), intent(in) :: x
  type(tensor), intent(in) :: y
  type(tensor) :: xy
  xy%t = x * y%t
end function dcomplex_tensor
```

Addition and Subtraction

⟨*Declaration of operations for tensors*⟩+≡
```
interface operator (+)
   module procedure plus_tensor
end interface
private :: plus_tensor
interface operator (-)
  module procedure neg_tensor
end interface
private :: neg_tensor
```

⟨*Implementation of operations for tensors*⟩+≡
```
pure function plus_tensor (t1) result (t2)
  type(tensor), intent(in) :: t1
  type(tensor) :: t2
  t2 = t1
end function plus_tensor
pure function neg_tensor (t1) result (t2)
  type(tensor), intent(in) :: t1
  type(tensor) :: t2
  t2%t = - t1%t
end function neg_tensor
```

⟨*Declaration of operations for tensors*⟩+≡
```
interface operator (+)
   module procedure add_tensor
end interface
private :: add_tensor
interface operator (-)
   module procedure sub_tensor
end interface
private :: sub_tensor
```

⟨*Implementation of operations for tensors*⟩+≡
```
pure function add_tensor (x, y) result (xy)
  type(tensor), intent(in) :: x, y
  type(tensor) :: xy
  xy%t = x%t + y%t
end function add_tensor
pure function sub_tensor (x, y) result (xy)
  type(tensor), intent(in) :: x, y
  type(tensor) :: xy
  xy%t = x%t - y%t
end function sub_tensor
```

⟨*Declaration of operations for tensors*⟩+≡
```
interface operator (.tprod.)
   module procedure out_prod_vv, out_prod_vm, &
        out_prod_mv, out_prod_mm
end interface
private :: out_prod_vv, out_prod_vm, &
     out_prod_mv, out_prod_mm
```

⟨*Implementation of operations for tensors*⟩+≡
```
pure function out_prod_vv (v, w) result (t)
  type(tensor) :: t
```

```
    type(vector), intent(in) :: v, w
    integer :: i, j
    t%t(0,0) = v%t * w%t
    t%t(0,1:3) = v%t * w%x
    t%t(1:3,0) = v%x * w%t
    do i = 1, 3
       do j = 1, 3
          t%t(i,j) = v%x(i) * w%x(j)
       end do
    end do
  end function out_prod_vv
```

⟨*Implementation of operations for tensors*⟩+≡

```
  pure function out_prod_vm (v, m) result (t)
    type(tensor) :: t
    type(vector), intent(in) :: v
    type(momentum), intent(in) :: m
    integer :: i, j
    t%t(0,0) = v%t * m%t
    t%t(0,1:3) = v%t * m%x
    t%t(1:3,0) = v%x * m%t
    do i = 1, 3
       do j = 1, 3
          t%t(i,j) = v%x(i) * m%x(j)
       end do
    end do
  end function out_prod_vm
```

⟨*Implementation of operations for tensors*⟩+≡

```
  pure function out_prod_mv (m, v) result (t)
    type(tensor) :: t
    type(vector), intent(in) :: v
    type(momentum), intent(in) :: m
    integer :: i, j
    t%t(0,0) = m%t * v%t
    t%t(0,1:3) = m%t * v%x
    t%t(1:3,0) = m%x * v%t
    do i = 1, 3
       do j = 1, 3
          t%t(i,j) = m%x(i) * v%x(j)
       end do
    end do
  end function out_prod_mv
```

⟨*Implementation of operations for tensors*⟩+≡

```
  pure function out_prod_mm (m, n) result (t)
    type(tensor) :: t
    type(momentum), intent(in) :: m, n
    integer :: i, j
    t%t(0,0) = m%t * n%t
    t%t(0,1:3) = m%t * n%x
    t%t(1:3,0) = m%x * n%t
    do i = 1, 3
       do j = 1, 3
          t%t(i,j) = m%x(i) * n%x(j)
       end do
```

```
    end do
  end function out_prod_mm
```

⟨*Declaration of operations for tensors*⟩+≡

```
  interface abs
     module procedure abs_tensor
  end interface
  private :: abs_tensor
```

⟨*Implementation of operations for tensors*⟩+≡

```
  pure function abs_tensor (t) result (abs_t)
    type(tensor), intent(in) :: t
    real(kind=default) :: abs_t
    abs_t = sqrt (sum ((abs (t%t))**2))
  end function abs_tensor
```

⟨*Declaration of operations for tensors*⟩+≡

```
  interface conjg
     module procedure conjg_tensor
  end interface
  private :: conjg_tensor
```

⟨*Implementation of operations for tensors*⟩+≡

```
  pure function conjg_tensor (t) result (conjg_t)
    type(tensor), intent(in) :: t
    type(tensor) :: conjg_t
    conjg_t%t = conjg (t%t)
  end function conjg_tensor
```

⟨*Declaration of operations for tensors*⟩+≡

```
  interface operator (*)
     module procedure tensor_tensor, vector_tensor, tensor_vector, &
          momentum_tensor, tensor_momentum
  end interface
  private :: tensor_tensor, vector_tensor, tensor_vector, &
       momentum_tensor, tensor_momentum
```

⟨*Implementation of operations for tensors*⟩+≡

```
  pure function tensor_tensor (t1, t2) result (t1t2)
    type(tensor), intent(in) :: t1
    type(tensor), intent(in) :: t2
    complex(kind=default) :: t1t2
    integer :: i1, i2
    t1t2 = t1%t(0,0)*t2%t(0,0) &
         - dot_product (conjg (t1%t(0,1:)), t2%t(0,1:)) &
         - dot_product (conjg (t1%t(1:,0)), t2%t(1:,0))
    do i1 = 1, 3
       do i2 = 1, 3
          t1t2 = t1t2 + t1%t(i1,i2)*t2%t(i1,i2)
       end do
    end do
  end function tensor_tensor
```

⟨*Implementation of operations for tensors*⟩+≡

```
  pure function tensor_vector (t, v) result (tv)
    type(tensor), intent(in) :: t
    type(vector), intent(in) :: v
    type(vector) :: tv
```

```
    tv%t =    t%t(0,0) * v%t - dot_product (conjg (t%t(0,1:)), v%x)
    tv%x(1) = t%t(0,1) * v%t - dot_product (conjg (t%t(1,1:)), v%x)
    tv%x(2) = t%t(0,2) * v%t - dot_product (conjg (t%t(2,1:)), v%x)
    tv%x(3) = t%t(0,3) * v%t - dot_product (conjg (t%t(3,1:)), v%x)
  end function tensor_vector
```

⟨*Implementation of operations for tensors*⟩+≡

```
  pure function vector_tensor (v, t) result (vt)
    type(vector), intent(in) :: v
    type(tensor), intent(in) :: t
    type(vector) :: vt
    vt%t =    v%t * t%t(0,0) - dot_product (conjg (v%x), t%t(1:,0))
    vt%x(1) = v%t * t%t(0,1) - dot_product (conjg (v%x), t%t(1:,1))
    vt%x(2) = v%t * t%t(0,2) - dot_product (conjg (v%x), t%t(1:,2))
    vt%x(3) = v%t * t%t(0,3) - dot_product (conjg (v%x), t%t(1:,3))
  end function vector_tensor
```

⟨*Implementation of operations for tensors*⟩+≡

```
  pure function tensor_momentum (t, p) result (tp)
    type(tensor), intent(in) :: t
    type(momentum), intent(in) :: p
    type(vector) :: tp
    tp%t =    t%t(0,0) * p%t - dot_product (conjg (t%t(0,1:)), p%x)
    tp%x(1) = t%t(0,1) * p%t - dot_product (conjg (t%t(1,1:)), p%x)
    tp%x(2) = t%t(0,2) * p%t - dot_product (conjg (t%t(2,1:)), p%x)
    tp%x(3) = t%t(0,3) * p%t - dot_product (conjg (t%t(3,1:)), p%x)
  end function tensor_momentum
```

⟨*Implementation of operations for tensors*⟩+≡

```
  pure function momentum_tensor (p, t) result (pt)
    type(momentum), intent(in) :: p
    type(tensor), intent(in) :: t
    type(vector) :: pt
    pt%t =    p%t * t%t(0,0) - dot_product (p%x, t%t(1:,0))
    pt%x(1) = p%t * t%t(0,1) - dot_product (p%x, t%t(1:,1))
    pt%x(2) = p%t * t%t(0,2) - dot_product (p%x, t%t(1:,2))
    pt%x(3) = p%t * t%t(0,3) - dot_product (p%x, t%t(1:,3))
  end function momentum_tensor
```

X.8 Symmetric Polarization Tensors

$$\epsilon^{\mu\nu}_{+2}(k) = \epsilon^{\mu}_{+}(k)\epsilon^{\nu}_{+}(k) \tag{X.19a}$$

$$\epsilon^{\mu\nu}_{+1}(k) = \frac{1}{\sqrt{2}}\left(\epsilon^{\mu}_{+}(k)\epsilon^{\nu}_{0}(k) + \epsilon^{\mu}_{0}(k)\epsilon^{\nu}_{+}(k)\right) \tag{X.19b}$$

$$\epsilon^{\mu\nu}_{0}(k) = \frac{1}{\sqrt{6}}\left(\epsilon^{\mu}_{+}(k)\epsilon^{\nu}_{-}(k) + \epsilon^{\mu}_{-}(k)\epsilon^{\nu}_{+}(k) - 2\epsilon^{\mu}_{0}(k)\epsilon^{\nu}_{0}(k)\right) \tag{X.19c}$$

$$\epsilon^{\mu\nu}_{-1}(k) = \frac{1}{\sqrt{2}}\left(\epsilon^{\mu}_{-}(k)\epsilon^{\nu}_{0}(k) + \epsilon^{\mu}_{0}(k)\epsilon^{\nu}_{-}(k)\right) \tag{X.19d}$$

$$\epsilon^{\mu\nu}_{-2}(k) = \epsilon^{\mu}_{-}(k)\epsilon^{\nu}_{-}(k) \tag{X.19e}$$

Note that $\epsilon^{\mu}_{\pm 2,\mu}(k) = \epsilon^{\mu}_{\pm}(k)\epsilon_{\pm,\mu}(k) \propto \epsilon^{\mu}_{\pm}(k)\epsilon^{*}_{\mp,\mu}(k) = 0$ and that the sign in $\epsilon^{\mu\nu}_{0}(k)$ insures its tracelessness[1].

```
⟨omega_tensor_polarizations.f90⟩≡
  ⟨Copyleft⟩
  module omega_tensor_polarizations
    use kinds
    use constants
    use omega_vectors
    use omega_tensors
    use omega_polarizations
    implicit none
    private
    ⟨Declaration of polarization tensors⟩
    integer, parameter, public :: omega_tensor_pols_2010_01_A = 0
  contains
    ⟨Implementation of polarization tensors⟩
  end module omega_tensor_polarizations
```

```
⟨Declaration of polarization tensors⟩≡
  public :: eps2
```

```
⟨Implementation of polarization tensors⟩≡
  pure function eps2 (m, k, s) result (t)
    type(tensor) :: t
    real(kind=default), intent(in) :: m
    type(momentum), intent(in) :: k
    integer, intent(in) :: s
    type(vector) :: ep, em, e0
    t%t = 0
    select case (s)
    case (2)
       ep = eps (m, k, 1)
       t = ep.tprod.ep
    case (1)
       ep = eps (m, k, 1)
       e0 = eps (m, k, 0)
       t = (1 / sqrt (2.0_default)) &
            * ((ep.tprod.e0) + (e0.tprod.ep))
    case (0)
       ep = eps (m, k, 1)
       e0 = eps (m, k, 0)
       em = eps (m, k, -1)
       t = (1 / sqrt (6.0_default)) &
             * ((ep.tprod.em) + (em.tprod.ep) - 2*(e0.tprod.e0))
    case (-1)
       e0 = eps (m, k, 0)
       em = eps (m, k, -1)
       t = (1 / sqrt (2.0_default)) &
             * ((em.tprod.e0) + (e0.tprod.em))
    case (-2)
```

[1] On the other hand, with the shift operator $L_- |+\rangle = e^{i\phi} |0\rangle$ and $L_- |0\rangle = e^{i\chi} |-\rangle$, we find

$$L_-^2 |{+}{+}\rangle = 2e^{2i\phi} |00\rangle + e^{i(\phi+\chi)}(|{+}{-}\rangle + |{-}{+}\rangle)$$

i. e. $\chi - \phi = \pi$, if we want to identify $\epsilon^{\mu}_{-,0,+}$ with $|-,0,+\rangle$.

```
      em = eps (m, k, -1)
      t = em.tprod.em
    end select
  end function eps2
```

X.9 Couplings

⟨omega_couplings.f90⟩≡
```
  ⟨Copyleft⟩
  module omega_couplings
    use kinds
    use constants
    use omega_vectors
    use omega_tensors
    implicit none
    private
    ⟨Declaration of couplings⟩
    ⟨Declaration of propagators⟩
    integer, parameter, public :: omega_couplings_2010_01_A = 0
  contains
    ⟨Implementation of couplings⟩
    ⟨Implementation of propagators⟩
  end module omega_couplings
```

⟨*Declaration of propagators*⟩≡
```
  public :: wd_tl
```

⟨*Declaration of propagators*⟩+≡
```
  public :: gauss
```

$$\Theta(p^2)\Gamma \tag{X.20}$$

⟨*Implementation of propagators*⟩≡
```
  pure function wd_tl (p, w) result (width)
    real(kind=default) :: width
    type(momentum), intent(in) :: p
    real(kind=default), intent(in) :: w
    if (p*p > 0) then
       width = w
    else
       width = 0
    end if
  end function wd_tl
```

⟨*Implementation of propagators*⟩+≡
```
  pure function gauss (x, mu, w) result (gg)
    real(kind=default) :: gg
    real(kind=default), intent(in) :: x, mu, w
    if (w > 0) then
      gg = exp(-(x - mu**2)**2/4.0_default/mu**2/w**2) * &
           sqrt(sqrt(PI/2)) / w / mu
      else
      gg = 1.0_default
    end if
  end function gauss
```

⟨*Declaration of propagators*⟩+≡
```
public :: pr_phi, pr_unitarity, pr_feynman, pr_gauge, pr_rxi
public :: pj_phi, pj_unitarity
public :: pg_phi, pg_unitarity
```

$$\frac{\mathrm{i}}{p^2 - m^2 + \mathrm{i}m\Gamma}\phi \tag{X.21}$$

⟨*Implementation of propagators*⟩+≡
```
pure function pr_phi (p, m, w, phi) result (pphi)
  complex(kind=default) :: pphi
  type(momentum), intent(in) :: p
  real(kind=default), intent(in) :: m, w
  complex(kind=default), intent(in) :: phi
  pphi = (1 / cmplx (p*p - m**2, m*w, kind=default)) * phi
end function pr_phi
```

$$\sqrt{\frac{\pi}{M\Gamma}}\phi \tag{X.22}$$

⟨*Implementation of propagators*⟩+≡
```
pure function pj_phi (m, w, phi) result (pphi)
  complex(kind=default) :: pphi
  real(kind=default), intent(in) :: m, w
  complex(kind=default), intent(in) :: phi
  pphi = (0, -1) * sqrt (PI / m / w) * phi
end function pj_phi
```

⟨*Implementation of propagators*⟩+≡
```
pure function pg_phi (p, m, w, phi) result (pphi)
  complex(kind=default) :: pphi
  type(momentum), intent(in) :: p
  real(kind=default), intent(in) :: m, w
  complex(kind=default), intent(in) :: phi
  pphi = ((0, 1) * gauss (p*p, m, w)) * phi
end function pg_phi
```

$$\frac{\mathrm{i}}{p^2 - m^2 + \mathrm{i}m\Gamma}\left(-g_{\mu\nu} + \frac{p_\mu p_\nu}{m^2}\right)\epsilon^\nu(p) \tag{X.23}$$

NB: the explicit cast to `vector` is required here, because a specific `complex_momentum` procedure for `operator (*)` would introduce ambiguities. NB: we used to use the constructor `vector (p%t, p%x)` instead of the temporary variable, but the Intel Fortran Compiler choked on it.

⟨*Implementation of propagators*⟩+≡
```
pure function pr_unitarity (p, m, w, e) result (pe)
  type(vector) :: pe
  type(momentum), intent(in) :: p
  real(kind=default), intent(in) :: m, w
  type(vector), intent(in) :: e
  type(vector) :: pv
  pv = p
  pe = - (1 / cmplx (p*p - m**2, m*w, kind=default)) &
       * (e - (p*e / m**2) * pv)
end function pr_unitarity
```

$$\sqrt{\frac{\pi}{M\Gamma}}\left(-g_{\mu\nu}+\frac{p_\mu p_\nu}{m^2}\right)\epsilon^\nu(p) \tag{X.24}$$

⟨*Implementation of propagators*⟩+≡
```
  pure function pj_unitarity (p, m, w, e) result (pe)
    type(vector) :: pe
    type(momentum), intent(in) :: p
    real(kind=default), intent(in) :: m, w
    type(vector), intent(in) :: e
    type(vector) :: pv
    pv = p
    pe = (0, 1) * sqrt (PI / m / w) * (e - (p*e / m**2) * pv)
  end function pj_unitarity
```

⟨*Implementation of propagators*⟩+≡
```
  pure function pg_unitarity (p, m, w, e) result (pe)
    type(vector) :: pe
    type(momentum), intent(in) :: p
    real(kind=default), intent(in) :: m, w
    type(vector), intent(in) :: e
    type(vector) :: pv
    pv = p
    pe = - gauss (p*p, m, w) &
         * (e - (p*e / m**2) * pv)
  end function pg_unitarity
```

$$\frac{-i}{p^2}\epsilon^\nu(p) \tag{X.25}$$

⟨*Implementation of propagators*⟩+≡
```
  pure function pr_feynman (p, e) result (pe)
    type(vector) :: pe
    type(momentum), intent(in) :: p
    type(vector), intent(in) :: e
    pe = - (1 / (p*p)) * e
  end function pr_feynman
```

$$\frac{\mathrm{i}}{p^2}\left(-g_{\mu\nu}+(1-\xi)\frac{p_\mu p_\nu}{p^2}\right)\epsilon^\nu(p) \tag{X.26}$$

⟨*Implementation of propagators*⟩+≡
```
  pure function pr_gauge (p, xi, e) result (pe)
    type(vector) :: pe
    type(momentum), intent(in) :: p
    real(kind=default), intent(in) :: xi
    type(vector), intent(in) :: e
    real(kind=default) :: p2
    type(vector) :: pv
    p2 = p*p
    pv = p
    pe = - (1 / p2) * (e - ((1 - xi) * (p*e) / p2) * pv)
  end function pr_gauge
```

$$\frac{\mathrm{i}}{p^2 - m^2 + \mathrm{i}m\Gamma}\left(-g_{\mu\nu} + (1-\xi)\frac{p_\mu p_\nu}{p^2 - \xi m^2}\right)\epsilon^\nu(p) \tag{X.27}$$

⟨*Implementation of propagators*⟩+≡

```
pure function pr_rxi (p, m, w, xi, e) result (pe)
  type(vector) :: pe
  type(momentum), intent(in) :: p
  real(kind=default), intent(in) :: m, w, xi
  type(vector), intent(in) :: e
  real(kind=default) :: p2
  type(vector) :: pv
  p2 = p*p
  pv = p
  pe = - (1 / cmplx (p2 - m**2, m*w, kind=default)) &
       * (e - ((1 - xi) * (p*e) / (p2 - xi * m**2)) * pv)
end function pr_rxi
```

⟨*Declaration of propagators*⟩+≡

```
public :: pr_tensor
```

$$\frac{\mathrm{i}P^{\mu\nu,\rho\sigma}(p,m)}{p^2 - m^2 + \mathrm{i}m\Gamma}T_{\rho\sigma} \tag{X.28a}$$

with

$$P^{\mu\nu,\rho\sigma}(p,m) = \frac{1}{2}\left(g^{\mu\rho} - \frac{p^\mu p^\nu}{m^2}\right)\left(g^{\nu\sigma} - \frac{p^\nu p^\sigma}{m^2}\right) + \frac{1}{2}\left(g^{\mu\sigma} - \frac{p^\mu p^\sigma}{m^2}\right)\left(g^{\nu\rho} - \frac{p^\nu p^\rho}{m^2}\right) \\ - \frac{1}{3}\left(g^{\mu\nu} - \frac{p^\mu p^\nu}{m^2}\right)\left(g^{\rho\sigma} - \frac{p^\rho p^\sigma}{m^2}\right) \tag{X.28b}$$

Be careful with raising and lowering of indices:

$$g^{\mu\nu} - \frac{k^\mu k^\nu}{m^2} = \begin{pmatrix} 1 - k^0k^0/m^2 & -k^0\vec{k}/m^2 \\ -\vec{k}k^0/m^2 & -\mathbf{1} - \vec{k}\otimes\vec{k}/m^2 \end{pmatrix} \tag{X.29a}$$

$$g^\mu{}_\nu - \frac{k^\mu k_\nu}{m^2} = \begin{pmatrix} 1 - k^0k^0/m^2 & k^0\vec{k}/m^2 \\ -\vec{k}k^0/m^2 & \mathbf{1} + \vec{k}\otimes\vec{k}/m^2 \end{pmatrix} \tag{X.29b}$$

⟨*Implementation of propagators*⟩+≡

```
pure function pr_tensor (p, m, w, t) result (pt)
  type(tensor) :: pt
  type(momentum), intent(in) :: p
  real(kind=default), intent(in) :: m, w
  type(tensor), intent(in) :: t
  complex(kind=default) :: p_dd_t
  real(kind=default), dimension(0:3,0:3) :: p_uu, p_ud, p_du, p_dd
  integer :: i, j
  p_uu(0,0) = 1 - p%t * p%t / m**2
  p_uu(0,1:3) = - p%t * p%x / m**2
  p_uu(1:3,0) = p_uu(0,1:3)
  do i = 1, 3
     do j = 1, 3
        p_uu(i,j) = - p%x(i) * p%x(j) / m**2
```

```
      end do
   end do
   do i = 1, 3
      p_uu(i,i) = - 1 + p_uu(i,i)
   end do
   p_ud(:,0) = p_uu(:,0)
   p_ud(:,1:3) = - p_uu(:,1:3)
   p_du = transpose (p_ud)
   p_dd(:,0) = p_du(:,0)
   p_dd(:,1:3) = - p_du(:,1:3)
   p_dd_t = 0
   do i = 0, 3
      do j = 0, 3
         p_dd_t = p_dd_t + p_dd(i,j) * t%t(i,j)
      end do
   end do
   pt%t = matmul (p_ud, matmul (0.5_default * (t%t + transpose (t%t)), p_du)) &
        - (p_dd_t / 3.0_default) * p_uu
   pt%t = pt%t / cmplx (p*p - m**2, m*w, kind=default)
 end function pr_tensor
```

X.9.1 Triple Gauge Couplings

⟨*Declaration of couplings*⟩≡

```
 public :: g_gg
```

According to (9.6c)

$$A^{a,\mu}(k_1+k_2) = -\mathrm{i}g\big((k_1^\mu - k_2^\mu)A^{a_1}(k_1)\cdot A^{a_2}(k_2) \\ + (2k_2+k_1)\cdot A^{a_1}(k_1)A^{a_2,\mu}(k_2) - A^{a_1,\mu}(k_1)A^{a_2}(k_2)\cdot(2k_1+k_2)\big) \quad \text{(X.30)}$$

⟨*Implementation of couplings*⟩≡

```
 pure function g_gg (g, a1, k1, a2, k2) result (a)
   complex(kind=default), intent(in) :: g
   type(vector), intent(in) :: a1, a2
   type(momentum), intent(in) :: k1, k2
   type(vector) :: a
   a = (0, -1) * g * ((k1 - k2) * (a1 * a2) &
                    + ((2*k2 + k1) * a1) * a2 - a1 * ((2*k1 + k2) * a2))
 end function g_gg
```

X.9.2 Quadruple Gauge Couplings

⟨*Declaration of couplings*⟩+≡

```
 public :: x_gg, g_gx
```

$$T^{a,\mu\nu}(k_1+k_2) = g\big(A^{a_1,\mu}(k_1)A^{a_2,\nu}(k_2) - A^{a_1,\nu}(k_1)A^{a_2,\mu}(k_2)\big) \quad \text{(X.31)}$$

⟨*Implementation of couplings*⟩+≡

```
 pure function x_gg (g, a1, a2) result (x)
   complex(kind=default), intent(in) :: g
   type(vector), intent(in) :: a1, a2
   type(tensor2odd) :: x
   x = g * (a1 .wedge. a2)
 end function x_gg
```

$$A^{a,\mu}(k_1+k_2) = gA^{a_1}_\nu(k_1)T^{a_2,\nu\mu}(k_2) \tag{X.32}$$

⟨*Implementation of couplings*⟩+≡
```
pure function g_gx (g, a1, x) result (a)
  complex(kind=default), intent(in) :: g
  type(vector), intent(in) :: a1
  type(tensor2odd), intent(in) :: x
  type(vector) :: a
  a = g * (a1 * x)
end function g_gx
```

X.9.3 Scalar Current

⟨*Declaration of couplings*⟩+≡
```
public :: v_ss, s_vs
```

$$V^\mu(k_1+k_2) = g(k_1^\mu - k_2^\mu)\phi_1(k_1)\phi_2(k_2) \tag{X.33}$$

⟨*Implementation of couplings*⟩+≡
```
pure function v_ss (g, phi1, k1, phi2, k2) result (v)
  complex(kind=default), intent(in) :: g, phi1, phi2
  type(momentum), intent(in) :: k1, k2
  type(vector) :: v
  v = (k1 - k2) * (g * phi1 * phi2)
end function v_ss
```

$$\phi(k_1+k_2) = g(k_1^\mu + 2k_2^\mu)V_\mu(k_1)\phi(k_2) \tag{X.34}$$

⟨*Implementation of couplings*⟩+≡
```
pure function s_vs (g, v1, k1, phi2, k2) result (phi)
  complex(kind=default), intent(in) :: g, phi2
  type(vector), intent(in) :: v1
  type(momentum), intent(in) :: k1, k2
  complex(kind=default) :: phi
  phi = g * ((k1 + 2*k2) * v1) * phi2
end function s_vs
```

X.9.4 Transversal Scalar-Vector Coupling

⟨*Declaration of couplings*⟩+≡
```
public :: s_vv_t, v_sv_t
```

$$phi(k_1+k_2) = g((V_1(k_1)V_2(k_2))(k_1k_2) - (V_1(k_1)k_2)(V_2(k_2)k_1)) \tag{X.35}$$

⟨*Implementation of couplings*⟩+≡
```
pure function s_vv_t (g, v1, k1, v2, k2) result (phi)
  complex(kind=default), intent(in) :: g
  type(vector), intent(in) :: v1, v2
  type(momentum), intent(in) :: k1, k2
  complex(kind=default) :: phi
  phi = g * ((v1*v2) * (k1*k2) - (v1*k2) * (v2*k1))
end function s_vv_t
```

$$V_1^\mu(k_\phi + k_V) = gphi(((k_\phi + k_V)k_V)V_2^\mu - (k_\phi + k_V)V_2)k_V^\mu) \tag{X.36}$$

⟨*Implementation of couplings*⟩+≡

```
  pure function v_sv_t (g, phi, kphi,v, kv) result (vout)
    complex(kind=default), intent(in) :: g, phi
    type(vector), intent(in) :: v
    type(momentum), intent(in) :: kv, kphi
    type(momentum) :: kout
    type(vector)  :: vout
    kout = - (kv + kphi)
    vout = g * phi * ((kout*kv) * v - (v * kout) * kv)
  end function v_sv_t
```

X.9.5 Triple Vector Couplings

⟨*Declaration of couplings*⟩+≡

```
  public :: tkv_vv, lkv_vv, tv_kvv, lv_kvv, kg_kgkg
  public :: t5kv_vv, l5kv_vv, t5v_kvv, l5v_kvv, kg5_kgkg, kg_kg5kg
```

$$V^\mu(k_1 + k_2) = \mathrm{i}g(k_1 - k_2)^\mu V_1^\nu(k_1)V_{2,\nu}(k_2) \tag{X.37}$$

⟨*Implementation of couplings*⟩+≡

```
  pure function tkv_vv (g, v1, k1, v2, k2) result (v)
    complex(kind=default), intent(in) :: g
    type(vector), intent(in) :: v1, v2
    type(momentum), intent(in) :: k1, k2
    type(vector) :: v
    v = (k1 - k2) * ((0, 1) * g * (v1*v2))
  end function tkv_vv
```

$$V^\mu(k_1 + k_2) = \mathrm{i}g\epsilon^{\mu\nu\rho\sigma}(k_1 - k_2)_\nu V_{1,\rho}(k_1)V_{2,\sigma}(k_2) \tag{X.38}$$

⟨*Implementation of couplings*⟩+≡

```
  pure function t5kv_vv (g, v1, k1, v2, k2) result (v)
    complex(kind=default), intent(in) :: g
    type(vector), intent(in) :: v1, v2
    type(momentum), intent(in) :: k1, k2
    type(vector) :: v
    type(vector) :: k
    k = k1 - k2
    v = (0, 1) * g * pseudo_vector (k, v1, v2)
  end function t5kv_vv
```

$$V^\mu(k_1 + k_2) = \mathrm{i}g(k_1 + k_2)^\mu V_1^\nu(k_1)V_{2,\nu}(k_2) \tag{X.39}$$

⟨*Implementation of couplings*⟩+≡

```
  pure function lkv_vv (g, v1, k1, v2, k2) result (v)
    complex(kind=default), intent(in) :: g
    type(vector), intent(in) :: v1, v2
    type(momentum), intent(in) :: k1, k2
    type(vector) :: v
    v = (k1 + k2) * ((0, 1) * g * (v1*v2))
  end function lkv_vv
```

$$V^\mu(k_1+k_2) = \mathrm{i}g\epsilon^{\mu\nu\rho\sigma}(k_1+k_2)_\nu V_{1,\rho}(k_1)V_{2,\sigma}(k_2) \tag{X.40}$$

⟨*Implementation of couplings*⟩+≡
```
  pure function l5kv_vv (g, v1, k1, v2, k2) result (v)
    complex(kind=default), intent(in) :: g
    type(vector), intent(in) :: v1, v2
    type(momentum), intent(in) :: k1, k2
    type(vector) :: v
    type(vector) :: k
    k = k1 + k2
    v = (0, 1) * g * pseudo_vector (k, v1, v2)
  end function l5kv_vv
```

$$V^\mu(k_1+k_2) = \mathrm{i}g(k_2-k)^\nu V_{1,\nu}(k_1)V_2^\mu(k_2) = \mathrm{i}g(2k_2+k_1)^\nu V_{1,\nu}(k_1)V_2^\mu(k_2) \tag{X.41}$$

using $k = -k_1 - k_2$

⟨*Implementation of couplings*⟩+≡
```
  pure function tv_kvv (g, v1, k1, v2, k2) result (v)
    complex(kind=default), intent(in) :: g
    type(vector), intent(in) :: v1, v2
    type(momentum), intent(in) :: k1, k2
    type(vector) :: v
    v = v2 * ((0, 1) * g * ((2*k2 + k1)*v1))
  end function tv_kvv
```

$$V^\mu(k_1+k_2) = \mathrm{i}g\epsilon^{\mu\nu\rho\sigma}(2k_2+k_1)_\nu V_{1,\rho}(k_1)V_{2,\sigma}(k_2) \tag{X.42}$$

⟨*Implementation of couplings*⟩+≡
```
  pure function t5v_kvv (g, v1, k1, v2, k2) result (v)
    complex(kind=default), intent(in) :: g
    type(vector), intent(in) :: v1, v2
    type(momentum), intent(in) :: k1, k2
    type(vector) :: v
    type(vector) :: k
    k = k1 + 2*k2
    v = (0, 1) * g * pseudo_vector (k, v1, v2)
  end function t5v_kvv
```

$$V^\mu(k_1+k_2) = -\mathrm{i}gk_1^\nu V_{1,\nu}(k_1)V_2^\mu(k_2) \tag{X.43}$$

using $k = -k_1 - k_2$

⟨*Implementation of couplings*⟩+≡
```
  pure function lv_kvv (g, v1, k1, v2) result (v)
    complex(kind=default), intent(in) :: g
    type(vector), intent(in) :: v1, v2
    type(momentum), intent(in) :: k1
    type(vector) :: v
    v = v2 * ((0, -1) * g * (k1*v1))
  end function lv_kvv
```

$$V^\mu(k_1+k_2) = -\mathrm{i}g\epsilon^{\mu\nu\rho\sigma}k_{1,\nu}V_{1,\rho}(k_1)V_{2,\sigma}(k_2) \tag{X.44}$$

⟨*Implementation of couplings*⟩+≡

```
  pure function l5v_kvv (g, v1, k1, v2) result (v)
    complex(kind=default), intent(in) :: g
    type(vector), intent(in) :: v1, v2
    type(momentum), intent(in) :: k1
    type(vector) :: v
    type(vector) :: k
    k = k1
    v = (0, -1) * g * pseudo_vector (k, v1, v2)
  end function l5v_kvv
```

$$A^\mu(k_1+k_2) = \mathrm{i}gk^\nu\Big(F_{1,\nu}{}^\rho(k_1)F_{2,\rho\mu}(k_2) - F_{1,\mu}{}^\rho(k_1)F_{2,\rho\nu}(k_2)\Big) \tag{X.45}$$

with $k=-k_1-k_2$, i. e.

$$\begin{aligned}A^\mu(k_1+k_2) = -\mathrm{i}g\Big([(kk_2)(k_1A_2) - (k_1k_2)(kA_2)]A_1^\mu \\ + [(k_1k_2)(kA_1) - (kk_1)(k_2A_1)]A_2^\mu \\ + [(k_2A_1)(kA_2) - (kk_2)(A_1A_2)]k_1^\mu \\ + [(kk_1)(A_1A_2) - (kA_1)(k_1A_2)]k_2^\mu\Big)\end{aligned} \tag{X.46}$$

⟨*Implementation of couplings*⟩+≡

```
  pure function kg_kgkg (g, a1, k1, a2, k2) result (a)
    complex(kind=default), intent(in) :: g
    type(vector), intent(in) :: a1, a2
    type(momentum), intent(in) :: k1, k2
    type(vector) :: a
    real(kind=default) :: k1k1, k2k2, k1k2, kk1, kk2
    complex(kind=default) :: a1a2, k2a1, ka1, k1a2, ka2
    k1k1 = k1 * k1
    k1k2 = k1 * k2
    k2k2 = k2 * k2
    kk1 = k1k1 + k1k2
    kk2 = k1k2 + k2k2
    k2a1 = k2 * a1
    ka1 = k2a1 + k1 * a1
    k1a2 = k1 * a2
    ka2 = k1a2 + k2 * a2
    a1a2 = a1 * a2
    a = (0, -1) * g * (   (kk2  * k1a2 - k1k2 * ka2 ) * a1 &
                        + (k1k2 * ka1  - kk1  * k2a1) * a2 &
                        + (ka2  * k2a1 - kk2  * a1a2) * k1 &
                        + (kk1  * a1a2 - ka1  * k1a2) * k2 )
  end function kg_kgkg
```

$$A^\mu(k_1+k_2) = \mathrm{i}g\epsilon^{\mu\nu\rho\sigma}k_\nu F_{1,\rho}{}^\lambda(k_1)F_{2,\lambda\sigma}(k_2) \tag{X.47}$$

with $k=-k_1-k_2$, i. e.

$$A^\mu(k_1+k_2) = -2\mathrm{i}g\epsilon^{\mu\nu\rho\sigma}k_\nu\Big((k_2A_1)k_{1,\rho}A_{2,\sigma} + (k_1A_2)A_{1,\rho}k_{2,\sigma}$$

$$- (A_1A_2)k_{1,\rho}k_{2,\sigma} - (k_1k_2)A_{1,\rho}A_{2,\sigma}\Big) \quad \text{(X.48)}$$

⟨*Implementation of couplings*⟩+≡
```
  pure function kg5_kgkg (g, a1, k1, a2, k2) result (a)
    complex(kind=default), intent(in) :: g
    type(vector), intent(in) :: a1, a2
    type(momentum), intent(in) :: k1, k2
    type(vector) :: a
    type(vector) :: kv, k1v, k2v
    kv = - k1 - k2
    k1v = k1
    k2v = k2
    a = (0, -2) * g * (   (k2*A1) * pseudo_vector (kv, k1v, a2 ) &
                        + (k1*A2) * pseudo_vector (kv, A1 , k2v) &
                        - (A1*A2) * pseudo_vector (kv, k1v, k2v) &
                        - (k1*k2) * pseudo_vector (kv, a1 , a2 ) )
  end function kg5_kgkg
```

$$A^\mu(k_1+k_2) = \mathrm{i}gk_\nu\Big(\epsilon^{\mu\rho\lambda\sigma}F_{1,}{}^{\nu}{}_{\rho} - \epsilon^{\nu\rho\lambda\sigma}F_{1,}{}^{\mu}{}_{\rho}\Big)\frac{1}{2}F_{1,\lambda\sigma} \quad \text{(X.49)}$$

with $k = -k_1 - k_2$, i. e.

$$A^\mu(k_1+k_2) = -\mathrm{i}g\Big(\epsilon^{\mu\rho\lambda\sigma}(kk_2)A_{2,\rho} - \epsilon^{\mu\rho\lambda\sigma}(kA_2)k_{2,\rho} - k_2^\mu\epsilon^{\nu\rho\lambda\sigma}k_nuA_{2,\rho} + A_2^\mu\epsilon^{\nu\rho\lambda\sigma}k_nuk_{2,\rho}\Big)k_{1,\lambda}A_{1,\sigma} \quad \text{(X.50)}$$

This is not the most efficienct way of doing it: $\epsilon^{\mu\nu\rho\sigma}F_{1,\rho\sigma}$ should be cached!

⟨*Implementation of couplings*⟩+≡
```
  pure function kg_kg5kg (g, a1, k1, a2, k2) result (a)
    complex(kind=default), intent(in) :: g
    type(vector), intent(in) :: a1, a2
    type(momentum), intent(in) :: k1, k2
    type(vector) :: a
    type(vector) :: kv, k1v, k2v
    kv = - k1 - k2
    k1v = k1
    k2v = k2
    a = (0, -1) * g * (   (kv*k2v) * pseudo_vector (a2 , k1v, a1) &
                        - (kv*a2 ) * pseudo_vector (k2v, k1v, a1) &
                        -  k2v * pseudo_scalar (kv, a2,  k1v, a1) &
                        +  a2  * pseudo_scalar (kv, k2v, k1v, a1) )
  end function kg_kg5kg
```

X.10 Graviton Couplings

⟨*Declaration of couplings*⟩+≡
```
  public :: s_gravs, v_gravv, grav_ss, grav_vv
```

⟨*Implementation of couplings*⟩+≡
```
  pure function s_gravs (g, m, k1, k2, t, s) result (phi)
    complex(kind=default), intent(in) :: g, s
    real(kind=default), intent(in) :: m
```

```
    type(momentum), intent(in) :: k1, k2
    type(tensor), intent(in) :: t
    complex(kind=default) :: phi, t_tr
    t_tr = t%t(0,0) - t%t(1,1) - t%t(2,2) - t%t(3,3)
    phi = g * s * (((t*k1)*k2) + ((t*k2)*k1) &
        - g * (m**2 + (k1*k2))*t_tr)/2.0_default
  end function s_gravs
```

⟨*Implementation of couplings*⟩+≡

```
  pure function grav_ss (g, m, k1, k2, s1, s2) result (t)
    complex(kind=default), intent(in) :: g, s1, s2
    real(kind=default), intent(in) :: m
    type(momentum), intent(in) :: k1, k2
    type(tensor) :: t_metric, t
    t_metric%t = 0
    t_metric%t(0,0) = 1.0_default
    t_metric%t(1,1) = - 1.0_default
    t_metric%t(2,2) = - 1.0_default
    t_metric%t(3,3) = - 1.0_default
    t = g*s1*s2/2.0_default * (-(m**2 + (k1*k2)) * t_metric &
      + (k1.tprod.k2) + (k2.tprod.k1))
  end function grav_ss
```

⟨*Implementation of couplings*⟩+≡

```
  pure function v_gravv (g, m, k1, k2, t, v) result (vec)
    complex(kind=default), intent(in) :: g
    real(kind=default), intent(in) :: m
    type(momentum), intent(in) :: k1, k2
    type(vector), intent(in) :: v
    type(tensor), intent(in) :: t
    complex(kind=default) :: t_tr
    real(kind=default) :: xi
    type(vector) :: vec
    xi = 1.0_default
    t_tr = t%t(0,0) - t%t(1,1) - t%t(2,2) - t%t(3,3)
    vec = (-g)/ 2.0_default * (((k1*k2) + m**2) * &
         (t*v + v*t - t_tr * v) + t_tr * (k1*v) * k2 &
         - (k1*v) * ((k2*t) + (t*k2)) &
         - ((k1*(t*v)) + (v*(t*k1))) * k2 &
         + ((k1*(t*k2)) + (k2*(t*k1))) * v)
  !!!       Unitarity gauge: xi -> Infinity
  !!!       + (1.0_default/xi) * (t_tr * ((k1*v)*k2) + &
  !!!       (k2*v)*k2 + (k2*v)*k1 - (k1*(t*v))*k1 + &
  !!!       (k2*v)*(k2*t) - (v*(t*k1))*k1 - (k2*v)*(t*k2)))
  end function v_gravv
```

⟨*Implementation of couplings*⟩+≡

```
  pure function grav_vv (g, m, k1, k2, v1, v2) result (t)
    complex(kind=default), intent(in) :: g
    type(momentum), intent(in) :: k1, k2
    real(kind=default), intent(in) :: m
    real(kind=default) :: xi
    type(vector), intent (in) :: v1, v2
    type(tensor) :: t_metric, t
    xi = 0.00001_default
    t_metric%t = 0
```

```
    t_metric%t(0,0) = 1.0_default
    t_metric%t(1,1) = - 1.0_default
    t_metric%t(2,2) = - 1.0_default
    t_metric%t(3,3) = - 1.0_default
    t = (-g)/2.0_default * ( &
         ((k1*k2) + m**2) * ( &
         (v1.tprod.v2) +  (v2.tprod.v1) - (v1*v2) * t_metric) &
         + (v1*k2)*(v2*k1)*t_metric &
         - (k2*v1)*((v2.tprod.k1) + (k1.tprod.v2)) &
         - (k1*v2)*((v1.tprod.k2) + (k2.tprod.v1)) &
         + (v1*v2)*((k1.tprod.k2) + (k2.tprod.k1)))
  !!!       Unitarity gauge: xi -> Infinity
  !!!       + (1.0_default/xi) * ( &
  !!!       ((k1*v1)*(k1*v2) + (k2*v1)*(k2*v2) + (k1*v1)*(k2*v2))* &
  !!!       t_metric) - (k1*v1) * ((k1.tprod.v2) + (v2.tprod.k1)) &
  !!!       - (k2*v2) * ((k2.tprod.v1) + (v1.tprod.k2)))
  end function grav_vv
```

X.11 Tensor Couplings

⟨*Declaration of couplings*⟩+≡

```
  public :: t2_vv, v_t2v, t2_vv_1, v_t2v_1, t2_vv_t, v_t2v_t
```

X.12 Scalar-Vector Dim-5 Couplings

⟨*Declaration of couplings*⟩+≡

```
  public :: phi_vv, v_phiv, phi_u_vv, v_u_phiv
```

⟨*Implementation of couplings*⟩+≡

```
  pure function phi_vv (g, k1, k2, v1, v2) result (phi)
    complex(kind=default), intent(in) :: g
    type(momentum), intent(in) :: k1, k2
    type(vector), intent(in) :: v1, v2
    complex(kind=default) :: phi
    phi = g * pseudo_scalar (k1, v1, k2, v2)
  end function phi_vv
```

⟨*Implementation of couplings*⟩+≡

```
  pure function v_phiv (g, phi, k1, k2, v) result (w)
    complex(kind=default), intent(in) :: g, phi
    type(vector), intent(in) :: v
    type(momentum), intent(in) :: k1, k2
    type(vector) :: w
    w = g * phi * pseudo_vector (k1, k2, v)
  end function v_phiv
```

⟨*Implementation of couplings*⟩+≡

```
  pure function phi_u_vv (g, k1, k2, v1, v2) result (phi)
    complex(kind=default), intent(in) :: g
    type(momentum), intent(in) :: k1, k2
    type(vector), intent(in) :: v1, v2
    complex(kind=default) :: phi
    phi = g * ((k1*v2)*((-(k1+k2))*v1) + &
```

```
             (k2*v1)*((-(k1+k2))*v2) + &
             (((k1+k2)*(k1+k2)) * (v1*v2)))
  end function phi_u_vv
```

⟨*Implementation of couplings*⟩+≡

```
  pure function v_u_phiv (g, phi, k1, k2, v) result (w)
    complex(kind=default), intent(in) :: g, phi
    type(vector), intent(in) :: v
    type(momentum), intent(in) :: k1, k2
    type(vector) :: w
    w = g * phi * ((k1*v)*k2 + &
         ((-(k1+k2))*v)*k1 + &
         ((k1*k1)*v))
  end function v_u_phiv
```

⟨*Implementation of couplings*⟩+≡

```
  pure function t2_vv (g, v1, v2) result (t)
    complex(kind=default), intent(in) :: g
    type(vector), intent(in) :: v1, v2
    type(tensor) :: t
    type(tensor) :: tmp
    tmp = v1.tprod.v2
    t%t = g * (tmp%t + transpose (tmp%t))
  end function t2_vv
```

⟨*Implementation of couplings*⟩+≡

```
  pure function v_t2v (g, t, v) result (tv)
    complex(kind=default), intent(in) :: g
    type(tensor), intent(in) :: t
    type(vector), intent(in) :: v
    type(vector) :: tv
    type(tensor) :: tmp
    tmp%t = t%t + transpose (t%t)
    tv = g * (tmp * v)
  end function v_t2v
```

⟨*Implementation of couplings*⟩+≡

```
  pure function t2_vv_1 (g, v1, v2) result (t)
    complex(kind=default), intent(in) :: g
    complex(kind=default) :: tmp_s
    type(vector), intent(in) :: v1, v2
    type(tensor) :: tmp
    type(tensor) :: t_metric, t
    t_metric%t = 0
    t_metric%t(0,0) =   1.0_default
    t_metric%t(1,1) = - 1.0_default
    t_metric%t(2,2) = - 1.0_default
    t_metric%t(3,3) = - 1.0_default
    tmp = v1.tprod.v2
    tmp_s = v1 * v2
    t%t = g * (tmp%t + transpose (tmp%t) - tmp_s * t_metric%t )
  end function t2_vv_1
```

⟨*Implementation of couplings*⟩+≡

```
  pure function v_t2v_1 (g, t, v) result (tv)
    complex(kind=default), intent(in) :: g
    type(tensor), intent(in) :: t
```

```
    type(vector), intent(in) :: v
    type(vector) :: tv, tmp_tv
    type(tensor) :: tmp
    tmp_tv =  ( t%t(0,0)-t%t(1,1)-t%t(2,2)-t%t(3,3) ) * v
    tmp%t = t%t + transpose (t%t)
    tv = g * (tmp * v - tmp_tv)
  end function v_t2v_1
```

⟨*Implementation of couplings*⟩+≡

```
  pure function t2_vv_t (g, v1, k1, v2, k2) result (t)
    complex(kind=default), intent(in) :: g
    complex(kind=default) :: tmp_s
    type(vector), intent(in) :: v1, v2
    type(momentum), intent(in) :: k1, k2
    type(tensor) :: tmp, tmp_v1k2, tmp_v2k1, tmp_k1k2, tmp2
    type(tensor) :: t_metric, t
    t_metric%t = 0
    t_metric%t(0,0) =   1.0_default
    t_metric%t(1,1) = - 1.0_default
    t_metric%t(2,2) = - 1.0_default
    t_metric%t(3,3) = - 1.0_default
    tmp = v1.tprod.v2
    tmp_s = v1 * v2
    tmp_v1k2 = (v2 * k1) * (v1.tprod.k2)
    tmp_v2k1 = (v1 * k2) * (v2.tprod.k1)
    tmp_k1k2 = tmp_s * (k1.tprod.k2)
    tmp2%t = tmp_v1k2%t + tmp_v2k1%t - tmp_k1k2%t
    t%t = g * ( (k1*k2) * (tmp%t + transpose (tmp%t) - tmp_s * t_metric%t ) &
         + ((v1 * k2) * (v2 * k1)) * t_metric%t &
         - tmp2%t - transpose(tmp2%t))
  end function t2_vv_t
```

⟨*Implementation of couplings*⟩+≡

```
  pure function v_t2v_t (g, t, kt, v, kv) result (tv)
    complex(kind=default), intent(in) :: g
    type(tensor), intent(in) :: t
    type(vector), intent(in) :: v
    type(momentum), intent(in) :: kt, kv
    type(momentum) :: kout
    type(vector) :: tv, tmp_tv
    type(tensor) :: tmp
    kout = - (kt + kv)
    tmp_tv =  ( t%t(0,0)-t%t(1,1)-t%t(2,2)-t%t(3,3) ) * v
    tmp%t = t%t + transpose (t%t)
    tv = g * ( (tmp * v - tmp_tv) * (kv * kout )&
         + ( t%t(0,0)-t%t(1,1)-t%t(2,2)-t%t(3,3) ) * (kout * v ) * kv &
         - (kout * v) * ( tmp * kv) &
         - (v* (t * kout) + kout * (t * v)) * kv &
         + (kout* (t * kv) + kv * (t * kout)) * v)
  end function v_t2v_t
```

⟨*Declaration of couplings*⟩+≡

```
  public :: t2_vv_d5_1, v_t2v_d5_1
```

⟨*Implementation of couplings*⟩+≡

```
  pure function t2_vv_d5_1 (g, v1, k1, v2, k2) result (t)
```

```
    complex(kind=default), intent(in) :: g
    type(vector), intent(in) :: v1, v2
    type(momentum), intent(in) :: k1, k2
    type(tensor) :: t
    t = (g * (v1 * v2)) * (k1-k2).tprod.(k1-k2)
  end function t2_vv_d5_1
```

⟨*Implementation of couplings*⟩+≡

```
  pure function v_t2v_d5_1 (g, t1, k1, v2, k2) result (tv)
    complex(kind=default), intent(in) :: g
    type(tensor), intent(in) :: t1
    type(vector), intent(in) :: v2
    type(momentum), intent(in) :: k1, k2
    type(vector) :: tv
    tv = (g * ((k1+2*k2).tprod.(k1+2*k2) * t1)) * v2
  end function v_t2v_d5_1
```

⟨*Declaration of couplings*⟩+≡

```
  public :: t2_vv_d5_2, v_t2v_d5_2
```

⟨*Implementation of couplings*⟩+≡

```
  pure function t2_vv_d5_2 (g, v1, k1, v2, k2) result (t)
    complex(kind=default), intent(in) :: g
    type(vector), intent(in) :: v1, v2
    type(momentum), intent(in) :: k1, k2
    type(tensor) :: t
    t = (g * (k2 * v1)) * (k2-k1).tprod.v2
    t%t = t%t + transpose (t%t)
  end function t2_vv_d5_2
```

⟨*Implementation of couplings*⟩+≡

```
  pure function v_t2v_d5_2 (g, t1, k1, v2, k2) result (tv)
    complex(kind=default), intent(in) :: g
    type(tensor), intent(in) :: t1
    type(vector), intent(in) :: v2
    type(momentum), intent(in) :: k1, k2
    type(vector) :: tv
    type(tensor) :: tmp
    type(momentum) :: k1_k2, k1_2k2
    k1_k2 = k1 + k2
    k1_2k2 = k1_k2 + k2
    tmp%t = t1%t + transpose (t1%t)
    tv = (g * (k1_k2 * v2)) * (k1_2k2 * tmp)
  end function v_t2v_d5_2
```

⟨*Declaration of couplings*⟩+≡

```
  public :: t2_vv_d7, v_t2v_d7
```

⟨*Implementation of couplings*⟩+≡

```
  pure function t2_vv_d7 (g, v1, k1, v2, k2) result (t)
    complex(kind=default), intent(in) :: g
    type(vector), intent(in) :: v1, v2
    type(momentum), intent(in) :: k1, k2
    type(tensor) :: t
    t = (g * (k2 * v1) * (k1 * v2)) * (k1-k2).tprod.(k1-k2)
  end function t2_vv_d7
```

⟨*Implementation of couplings*⟩+≡
```
pure function v_t2v_d7 (g, t1, k1, v2, k2) result (tv)
  complex(kind=default), intent(in) :: g
  type(tensor), intent(in) :: t1
  type(vector), intent(in) :: v2
  type(momentum), intent(in) :: k1, k2
  type(vector) :: tv
  type(vector) :: k1_k2, k1_2k2
  k1_k2 = k1 + k2
  k1_2k2 = k1_k2 + k2
  tv = (- g * (k1_k2 * v2) * (k1_2k2.tprod.k1_2k2 * t1)) * k2
end function v_t2v_d7
```

X.13 Spinor Couplings

⟨omega_spinor_couplings.f90⟩≡
```
⟨Copyleft⟩
module omega_spinor_couplings
  use kinds
  use constants
  use omega_spinors
  use omega_vectors
  use omega_tensors
  use omega_couplings
  implicit none
  private
  ⟨Declaration of spinor on shell wave functions⟩
  ⟨Declaration of spinor off shell wave functions⟩
  ⟨Declaration of spinor currents⟩
  ⟨Declaration of spinor propagators⟩
  integer, parameter, public :: omega_spinor_cpls_2010_01_A = 0
contains
  ⟨Implementation of spinor on shell wave functions⟩
  ⟨Implementation of spinor off shell wave functions⟩
  ⟨Implementation of spinor currents⟩
  ⟨Implementation of spinor propagators⟩
end module omega_spinor_couplings
```

See table X.1 for the names of Fortran functions. We could have used long names instead, but this would increase the chance of running past continuation line limits without adding much to the legibility.

X.13.1 Fermionic Vector and Axial Couplings

There's more than one chiral representation. This one is compatible with HELAS [5].

$$\gamma^0 = \begin{pmatrix} 0 & \mathbf{1} \\ \mathbf{1} & 0 \end{pmatrix},\ \gamma^i = \begin{pmatrix} 0 & \sigma^i \\ -\sigma^i & 0 \end{pmatrix},\ \gamma_5 = i\gamma^0\gamma^1\gamma^2\gamma^3 = \begin{pmatrix} -\mathbf{1} & 0 \\ 0 & \mathbf{1} \end{pmatrix} \tag{X.51}$$

$\bar{\psi}(g_V\gamma^\mu - g_A\gamma^\mu\gamma_5)\psi$	`va_ff`$(g_V, g_A, \bar{\psi}, \psi)$
$g_V\bar{\psi}\gamma^\mu\psi$	`v_ff`$(g_V, \bar{\psi}, \psi)$
$g_A\bar{\psi}\gamma_5\gamma^\mu\psi$	`a_ff`$(g_A, \bar{\psi}, \psi)$
$g_L\bar{\psi}\gamma^\mu(1-\gamma_5)\psi$	`vl_ff`$(g_L, \bar{\psi}, \psi)$
$g_R\bar{\psi}\gamma^\mu(1+\gamma_5)\psi$	`vr_ff`$(g_R, \bar{\psi}, \psi)$
$\not{V}(g_V - g_A\gamma_5)\psi$	`f_vaf`(g_V, g_A, V, ψ)
$g_V\not{V}\psi$	`f_vf`(g_V, V, ψ)
$g_A\gamma_5\not{V}\psi$	`f_af`(g_A, V, ψ)
$g_L\not{V}(1-\gamma_5)\psi$	`f_vlf`(g_L, V, ψ)
$g_R\not{V}(1+\gamma_5)\psi$	`f_vrf`(g_R, V, ψ)
$\psi\not{V}(g_V - g_A\gamma_5)$	`f_fva`(g_V, g_A, ψ, V)
$g_V\bar{\psi}\not{V}$	`f_fv`$(g_V, \bar{\psi}, V)$
$g_A\bar{\psi}\gamma_5\not{V}$	`f_fa`$(g_A, \bar{\psi}, V)$
$g_L\bar{\psi}\not{V}(1-\gamma_5)$	`f_fvl`$(g_L, \bar{\psi}, V)$
$g_R\bar{\psi}\not{V}(1+\gamma_5)$	`f_fvr`$(g_R, \bar{\psi}, V)$

Table X.1: Mnemonically abbreviated names of Fortran functions implementing fermionic vector and axial currents.

Therefore

$$g_S + g_P\gamma_5 = \begin{pmatrix} g_S - g_P & 0 & 0 & 0 \\ 0 & g_S - g_P & 0 & 0 \\ 0 & 0 & g_S + g_P & 0 \\ 0 & 0 & 0 & g_S + g_P \end{pmatrix} \tag{X.52a}$$

$$g_V\gamma^0 - g_A\gamma^0\gamma_5 = \begin{pmatrix} 0 & 0 & g_V - g_A & 0 \\ 0 & 0 & 0 & g_V - g_A \\ g_V + g_A & 0 & 0 & 0 \\ 0 & g_V + g_A & 0 & 0 \end{pmatrix} \tag{X.52b}$$

$$g_V\gamma^1 - g_A\gamma^1\gamma_5 = \begin{pmatrix} 0 & 0 & 0 & g_V - g_A \\ 0 & 0 & g_V - g_A & 0 \\ 0 & -g_V - g_A & 0 & 0 \\ -g_V - g_A & 0 & 0 & 0 \end{pmatrix} \tag{X.52c}$$

$$g_V\gamma^2 - g_A\gamma^2\gamma_5 = \begin{pmatrix} 0 & 0 & 0 & -\mathrm{i}(g_V - g_A) \\ 0 & 0 & \mathrm{i}(g_V - g_A) & 0 \\ 0 & \mathrm{i}(g_V + g_A) & 0 & 0 \\ -\mathrm{i}(g_V + g_A) & 0 & 0 & 0 \end{pmatrix} \tag{X.52d}$$

$$g_V\gamma^3 - g_A\gamma^3\gamma_5 = \begin{pmatrix} 0 & 0 & g_V - g_A & 0 \\ 0 & 0 & 0 & -g_V + g_A \\ -g_V - g_A & 0 & 0 & 0 \\ 0 & g_V + g_A & 0 & 0 \end{pmatrix} \tag{X.52e}$$

⟨*Declaration of spinor currents*⟩≡

```
public :: va_ff, v_ff, a_ff, vl_ff, vr_ff, vlr_ff, grav_ff, va2_ff, &
          tva_ff, tlr_ff, trl_ff, tvam_ff, tlrm_ff, trlm_ff, va3_ff
```

⟨*Implementation of spinor currents*⟩≡

$\bar{\psi}(g_S + g_P\gamma_5)\psi$	sp_ff$(g_S, g_P, \bar{\psi}, \psi)$
$g_S\bar{\psi}\psi$	s_ff$(g_S, \bar{\psi}, \psi)$
$g_P\bar{\psi}\gamma_5\psi$	p_ff$(g_P, \bar{\psi}, \psi)$
$g_L\bar{\psi}(1-\gamma_5)\psi$	sl_ff$(g_L, \bar{\psi}, \psi)$
$g_R\bar{\psi}(1+\gamma_5)\psi$	sr_ff$(g_R, \bar{\psi}, \psi)$
$\phi(g_S + g_P\gamma_5)\psi$	f_spf(g_S, g_P, ϕ, ψ)
$g_S\phi\psi$	f_sf(g_S, ϕ, ψ)
$g_P\phi\gamma_5\psi$	f_pf(g_P, ϕ, ψ)
$g_L\phi(1-\gamma_5)\psi$	f_slf(g_L, ϕ, ψ)
$g_R\phi(1+\gamma_5)\psi$	f_srf(g_R, ϕ, ψ)
$\bar{\psi}\phi(g_S + g_P\gamma_5)$	f_fsp$(g_S, g_P, \bar{\psi}, \phi)$
$g_S\bar{\psi}\phi$	f_fs$(g_S, \bar{\psi}, \phi)$
$g_P\bar{\psi}\phi\gamma_5$	f_fp$(g_P, \bar{\psi}, \phi)$
$g_L\bar{\psi}\phi(1-\gamma_5)$	f_fsl$(g_L, \bar{\psi}, \phi)$
$g_R\bar{\psi}\phi(1+\gamma_5)$	f_fsr$(g_R, \bar{\psi}, \phi)$

Table X.2: Mnemonically abbreviated names of Fortran functions implementing fermionic scalar and pseudo scalar "currents".

```
pure function va_ff (gv, ga, psibar, psi) result (j)
  type(vector) :: j
  complex(kind=default), intent(in) :: gv, ga
  type(conjspinor), intent(in) :: psibar
  type(spinor), intent(in) :: psi
  complex(kind=default) :: gl, gr
  complex(kind=default) :: g13, g14, g23, g24, g31, g32, g41, g42
  gl = gv + ga
  gr = gv - ga
  g13 = psibar%a(1)*psi%a(3)
  g14 = psibar%a(1)*psi%a(4)
  g23 = psibar%a(2)*psi%a(3)
  g24 = psibar%a(2)*psi%a(4)
  g31 = psibar%a(3)*psi%a(1)
  g32 = psibar%a(3)*psi%a(2)
  g41 = psibar%a(4)*psi%a(1)
  g42 = psibar%a(4)*psi%a(2)
  j%t    =  gr * (   g13 + g24) + gl * (   g31 + g42)
  j%x(1) =  gr * (   g14 + g23) - gl * (   g32 + g41)
  j%x(2) = (gr * ( - g14 + g23) + gl * (   g32 - g41)) * (0, 1)
  j%x(3) =  gr * (   g13 - g24) + gl * ( - g31 + g42)
end function va_ff
```

⟨*Implementation of spinor currents*⟩+≡

```
pure function va2_ff (gva, psibar, psi) result (j)
  type(vector) :: j
  complex(kind=default), intent(in), dimension(2) :: gva
  type(conjspinor), intent(in) :: psibar
  type(spinor), intent(in) :: psi
  complex(kind=default) :: gl, gr
  complex(kind=default) :: g13, g14, g23, g24, g31, g32, g41, g42
  gl = gva(1) + gva(2)
```

```
    gr = gva(1) - gva(2)
    g13 = psibar%a(1)*psi%a(3)
    g14 = psibar%a(1)*psi%a(4)
    g23 = psibar%a(2)*psi%a(3)
    g24 = psibar%a(2)*psi%a(4)
    g31 = psibar%a(3)*psi%a(1)
    g32 = psibar%a(3)*psi%a(2)
    g41 = psibar%a(4)*psi%a(1)
    g42 = psibar%a(4)*psi%a(2)
    j%t    =  gr * (   g13 + g24) + gl * (   g31 + g42)
    j%x(1) =  gr * (   g14 + g23) - gl * (   g32 + g41)
    j%x(2) = (gr * ( - g14 + g23) + gl * (   g32 - g41)) * (0, 1)
    j%x(3) =  gr * (   g13 - g24) + gl * ( - g31 + g42)
  end function va2_ff
```

⟨*Implementation of spinor currents*⟩+≡

```
  pure function va3_ff (gv, ga, psibar, psi) result (j)
    type(vector) :: j
    complex(kind=default), intent(in) :: gv, ga
    type(conjspinor), intent(in) :: psibar
    type(spinor), intent(in) :: psi
    j   = va_ff (gv, ga, psibar, psi)
    j%t = 0.0_default
  end function va3_ff
```

⟨*Implementation of spinor currents*⟩+≡

```
  pure function tva_ff (gv, ga, psibar, psi) result (t)
    type(tensor2odd) :: t
    complex(kind=default), intent(in) :: gv, ga
    type(conjspinor), intent(in) :: psibar
    type(spinor), intent(in) :: psi
    complex(kind=default) :: gl, gr
    complex(kind=default) :: g12, g21, g1m2, g34, g43, g3m4
    gr     = gv + ga
    gl     = gv - ga
    g12    = psibar%a(1)*psi%a(2)
    g21    = psibar%a(2)*psi%a(1)
    g1m2   = psibar%a(1)*psi%a(1) - psibar%a(2)*psi%a(2)
    g34    = psibar%a(3)*psi%a(4)
    g43    = psibar%a(4)*psi%a(3)
    g3m4   = psibar%a(3)*psi%a(3) - psibar%a(4)*psi%a(4)
    t%e(1) = (gl * ( - g12 - g21) + gr * (   g34 + g43)) * (0, 1)
    t%e(2) =  gl * ( - g12 + g21) + gr * (   g34 - g43)
    t%e(3) = (gl * ( - g1m2     ) + gr * (   g3m4     )) * (0, 1)
    t%b(1) =  gl * (   g12 + g21) + gr * (   g34 + g43)
    t%b(2) = (gl * ( - g12 + g21) + gr * ( - g34 + g43)) * (0, 1)
    t%b(3) =  gl * (   g1m2     ) + gr * (   g3m4     )
  end function tva_ff
```

⟨*Implementation of spinor currents*⟩+≡

```
  pure function tlr_ff (gl, gr, psibar, psi) result (t)
    type(tensor2odd) :: t
    complex(kind=default), intent(in) :: gl, gr
    type(conjspinor), intent(in) :: psibar
    type(spinor), intent(in) :: psi
    t = tva_ff (gr+gl, gr-gl, psibar, psi)
```

```
  end function tlr_ff
```

⟨*Implementation of spinor currents*⟩+≡

```
  pure function trl_ff (gr, gl, psibar, psi) result (t)
    type(tensor2odd) :: t
    complex(kind=default), intent(in) :: gl, gr
    type(conjspinor), intent(in) :: psibar
    type(spinor), intent(in) :: psi
    t = tva_ff (gr+gl, gr-gl, psibar, psi)
  end function trl_ff
```

⟨*Implementation of spinor currents*⟩+≡

```
  pure function tvam_ff (gv, ga, psibar, psi, p) result (j)
    type(vector) :: j
    complex(kind=default), intent(in) :: gv, ga
    type(conjspinor), intent(in) :: psibar
    type(spinor), intent(in) :: psi
    type(momentum), intent(in) :: p
    j = (tva_ff(gv, ga, psibar, psi) * p) * (0,1)
  end function tvam_ff
```

⟨*Implementation of spinor currents*⟩+≡

```
  pure function tlrm_ff (gl, gr, psibar, psi, p) result (j)
    type(vector) :: j
    complex(kind=default), intent(in) :: gl, gr
    type(conjspinor), intent(in) :: psibar
    type(spinor), intent(in) :: psi
    type(momentum), intent(in) :: p
    j = tvam_ff (gr+gl, gr-gl, psibar, psi, p)
  end function tlrm_ff
```

⟨*Implementation of spinor currents*⟩+≡

```
  pure function trlm_ff (gr, gl, psibar, psi, p) result (j)
    type(vector) :: j
    complex(kind=default), intent(in) :: gl, gr
    type(conjspinor), intent(in) :: psibar
    type(spinor), intent(in) :: psi
    type(momentum), intent(in) :: p
    j = tvam_ff (gr+gl, gr-gl, psibar, psi, p)
  end function trlm_ff
```

Special cases that avoid some multiplications

⟨*Implementation of spinor currents*⟩+≡

```
  pure function v_ff (gv, psibar, psi) result (j)
    type(vector) :: j
    complex(kind=default), intent(in) :: gv
    type(conjspinor), intent(in) :: psibar
    type(spinor), intent(in) :: psi
    complex(kind=default) :: g13, g14, g23, g24, g31, g32, g41, g42
    g13 = psibar%a(1)*psi%a(3)
    g14 = psibar%a(1)*psi%a(4)
    g23 = psibar%a(2)*psi%a(3)
    g24 = psibar%a(2)*psi%a(4)
    g31 = psibar%a(3)*psi%a(1)
    g32 = psibar%a(3)*psi%a(2)
    g41 = psibar%a(4)*psi%a(1)
    g42 = psibar%a(4)*psi%a(2)
```

```
    j%t    =   gv * (   g13 + g24 + g31 + g42)
    j%x(1) =   gv * (   g14 + g23 - g32 - g41)
    j%x(2) =   gv * ( - g14 + g23 + g32 - g41) * (0, 1)
    j%x(3) =   gv * (   g13 - g24 - g31 + g42)
  end function v_ff
```

⟨*Implementation of spinor currents*⟩+≡

```
  pure function a_ff (ga, psibar, psi) result (j)
    type(vector) :: j
    complex(kind=default), intent(in) :: ga
    type(conjspinor), intent(in) :: psibar
    type(spinor), intent(in) :: psi
    complex(kind=default) :: g13, g14, g23, g24, g31, g32, g41, g42
    g13 = psibar%a(1)*psi%a(3)
    g14 = psibar%a(1)*psi%a(4)
    g23 = psibar%a(2)*psi%a(3)
    g24 = psibar%a(2)*psi%a(4)
    g31 = psibar%a(3)*psi%a(1)
    g32 = psibar%a(3)*psi%a(2)
    g41 = psibar%a(4)*psi%a(1)
    g42 = psibar%a(4)*psi%a(2)
    j%t    =   ga * ( - g13 - g24 + g31 + g42)
    j%x(1) = - ga * (   g14 + g23 + g32 + g41)
    j%x(2) =   ga * (   g14 - g23 + g32 - g41) * (0, 1)
    j%x(3) =   ga * ( - g13 + g24 - g31 + g42)
  end function a_ff
```

⟨*Implementation of spinor currents*⟩+≡

```
  pure function vl_ff (gl, psibar, psi) result (j)
    type(vector) :: j
    complex(kind=default), intent(in) :: gl
    type(conjspinor), intent(in) :: psibar
    type(spinor), intent(in) :: psi
    complex(kind=default) :: gl2
    complex(kind=default) :: g31, g32, g41, g42
    gl2 = 2 * gl
    g31 = psibar%a(3)*psi%a(1)
    g32 = psibar%a(3)*psi%a(2)
    g41 = psibar%a(4)*psi%a(1)
    g42 = psibar%a(4)*psi%a(2)
    j%t    =   gl2 * (   g31 + g42)
    j%x(1) = - gl2 * (   g32 + g41)
    j%x(2) =   gl2 * (   g32 - g41) * (0, 1)
    j%x(3) =   gl2 * ( - g31 + g42)
  end function vl_ff
```

⟨*Implementation of spinor currents*⟩+≡

```
  pure function vr_ff (gr, psibar, psi) result (j)
    type(vector) :: j
    complex(kind=default), intent(in) :: gr
    type(conjspinor), intent(in) :: psibar
    type(spinor), intent(in) :: psi
    complex(kind=default) :: gr2
    complex(kind=default) :: g13, g14, g23, g24
    gr2 = 2 * gr
    g13 = psibar%a(1)*psi%a(3)
```

```
    g14 = psibar%a(1)*psi%a(4)
    g23 = psibar%a(2)*psi%a(3)
    g24 = psibar%a(2)*psi%a(4)
    j%t    = gr2 * (   g13 + g24)
    j%x(1) = gr2 * (   g14 + g23)
    j%x(2) = gr2 * ( - g14 + g23) * (0, 1)
    j%x(3) = gr2 * (   g13 - g24)
  end function vr_ff
```

⟨*Implementation of spinor currents*⟩+≡

```
  pure function grav_ff (g, m, kb, k, psibar, psi) result (j)
    type(tensor) :: j
    complex(kind=default), intent(in) :: g
    real(kind=default), intent(in) :: m
    type(conjspinor), intent(in) :: psibar
    type(spinor), intent(in) :: psi
    type(momentum), intent(in) :: kb, k
    complex(kind=default) :: g2, g8, c_dum
    type(vector) :: v_dum
    type(tensor) :: t_metric
    t_metric%t = 0
    t_metric%t(0,0) = 1.0_default
    t_metric%t(1,1) = - 1.0_default
    t_metric%t(2,2) = - 1.0_default
    t_metric%t(3,3) = - 1.0_default
    g2 = g/2.0_default
    g8 = g/8.0_default
    v_dum = v_ff(g8, psibar, psi)
    c_dum = (- m) * s_ff (g2, psibar, psi) - (kb+k)*v_dum
    j = c_dum*t_metric - (((kb+k).tprod.v_dum) + &
         (v_dum.tprod.(kb+k)))
  end function grav_ff
```

$$g_L\gamma_\mu(1-\gamma_5)+g_R\gamma_\mu(1+\gamma_5)=(g_L+g_R)\gamma_\mu-(g_L-g_R)\gamma_\mu\gamma_5=g_V\gamma_\mu-g_A\gamma_\mu\gamma_5 \tag{X.53}$$

... give the compiler the benefit of the doubt that it will optimize the function all. If not, we could inline it ...

⟨*Implementation of spinor currents*⟩+≡

```
  pure function vlr_ff (gl, gr, psibar, psi) result (j)
    type(vector) :: j
    complex(kind=default), intent(in) :: gl, gr
    type(conjspinor), intent(in) :: psibar
    type(spinor), intent(in) :: psi
    j = va_ff (gl+gr, gl-gr, psibar, psi)
  end function vlr_ff
```

and

$$\not v - \not a\gamma_5 = \begin{pmatrix} 0 & 0 & v_- - a_- & -v^* + a^* \\ 0 & 0 & -v + a & v_+ - a_+ \\ v_+ + a_+ & v^* + a^* & 0 & 0 \\ v + a & v_- + a_- & 0 & 0 \end{pmatrix} \tag{X.54}$$

with $v_\pm = v_0 \pm v_3$, $a_\pm = a_0 \pm a_3$, $v = v_1 + \mathrm{i}v_2$, $v^* = v_1 - \mathrm{i}v_2$, $a = a_1 + \mathrm{i}a_2$, and $a^* = a_1 - \mathrm{i}a_2$. But note that $\cdot^*$ is *not* complex conjugation for complex v_μ or a_μ.

⟨*Declaration of spinor currents*⟩+≡
```
public :: f_vaf, f_vf, f_af, f_vlf, f_vrf, f_vlrf, f_va2f, &
          f_tvaf, f_tlrf, f_trlf, f_tvamf, f_tlrmf, f_trlmf, f_va3f
```

⟨*Implementation of spinor currents*⟩+≡
```
pure function f_vaf (gv, ga, v, psi) result (vpsi)
  type(spinor) :: vpsi
  complex(kind=default), intent(in) :: gv, ga
  type(vector), intent(in) :: v
  type(spinor), intent(in) :: psi
  complex(kind=default) :: gl, gr
  complex(kind=default) :: vp, vm, v12, v12s
  gl = gv + ga
  gr = gv - ga
  vp = v%t + v%x(3)
  vm = v%t - v%x(3)
  v12  =  v%x(1) + (0,1)*v%x(2)
  v12s =  v%x(1) - (0,1)*v%x(2)
  vpsi%a(1) = gr * (   vm  * psi%a(3) - v12s * psi%a(4))
  vpsi%a(2) = gr * ( - v12 * psi%a(3) + vp   * psi%a(4))
  vpsi%a(3) = gl * (   vp  * psi%a(1) + v12s * psi%a(2))
  vpsi%a(4) = gl * (   v12 * psi%a(1) + vm   * psi%a(2))
end function f_vaf
```

⟨*Implementation of spinor currents*⟩+≡
```
pure function f_va2f (gva, v, psi) result (vpsi)
  type(spinor) :: vpsi
  complex(kind=default), intent(in), dimension(2) :: gva
  type(vector), intent(in) :: v
  type(spinor), intent(in) :: psi
  complex(kind=default) :: gl, gr
  complex(kind=default) :: vp, vm, v12, v12s
  gl = gva(1) + gva(2)
  gr = gva(1) - gva(2)
  vp = v%t + v%x(3)
  vm = v%t - v%x(3)
  v12  =  v%x(1) + (0,1)*v%x(2)
  v12s =  v%x(1) - (0,1)*v%x(2)
  vpsi%a(1) = gr * (   vm  * psi%a(3) - v12s * psi%a(4))
  vpsi%a(2) = gr * ( - v12 * psi%a(3) + vp   * psi%a(4))
  vpsi%a(3) = gl * (   vp  * psi%a(1) + v12s * psi%a(2))
  vpsi%a(4) = gl * (   v12 * psi%a(1) + vm   * psi%a(2))
end function f_va2f
```

⟨*Implementation of spinor currents*⟩+≡
```
pure function f_va3f (gv, ga, v, psi) result (vpsi)
  type(spinor) :: vpsi
  complex(kind=default), intent(in) :: gv, ga
  type(vector), intent(in) :: v
  type(spinor), intent(in) :: psi
  complex(kind=default) :: gl, gr
  complex(kind=default) :: vp, vm, v12, v12s
```

```
    gl = gv + ga
    gr = gv - ga
    vp =   v%x(3) !+ v%t
    vm = - v%x(3) !+ v%t
    v12  =  v%x(1) + (0,1)*v%x(2)
    v12s =  v%x(1) - (0,1)*v%x(2)
    vpsi%a(1) = gr * (   vm  * psi%a(3) - v12s * psi%a(4))
    vpsi%a(2) = gr * ( - v12 * psi%a(3) + vp   * psi%a(4))
    vpsi%a(3) = gl * (   vp  * psi%a(1) + v12s * psi%a(2))
    vpsi%a(4) = gl * (   v12 * psi%a(1) + vm   * psi%a(2))
  end function f_va3f
```

⟨*Implementation of spinor currents*⟩+≡

```
  pure function f_tvaf (gv, ga, t, psi) result (tpsi)
    type(spinor) :: tpsi
    complex(kind=default), intent(in) :: gv, ga
    type(tensor2odd), intent(in) :: t
    type(spinor), intent(in) :: psi
    complex(kind=default) :: gl, gr
    complex(kind=default) :: e21, e21s, b12, b12s, be3, be3s
    gr   = gv + ga
    gl   = gv - ga
    e21  = t%e(2) + t%e(1)*(0,1)
    e21s = t%e(2) - t%e(1)*(0,1)
    b12  = t%b(1) + t%b(2)*(0,1)
    b12s = t%b(1) - t%b(2)*(0,1)
    be3  = t%b(3) + t%e(3)*(0,1)
    be3s = t%b(3) - t%e(3)*(0,1)
    tpsi%a(1) =   2*gl * (   psi%a(1) * be3  + psi%a(2) * ( e21 +b12s))
    tpsi%a(2) =   2*gl * ( - psi%a(2) * be3  + psi%a(1) * (-e21s+b12 ))
    tpsi%a(3) =   2*gr * (   psi%a(3) * be3s + psi%a(4) * (-e21 +b12s))
    tpsi%a(4) =   2*gr * ( - psi%a(4) * be3s + psi%a(3) * ( e21s+b12 ))
  end function f_tvaf
```

⟨*Implementation of spinor currents*⟩+≡

```
  pure function f_tlrf (gl, gr, t, psi) result (tpsi)
    type(spinor) :: tpsi
    complex(kind=default), intent(in) :: gl, gr
    type(tensor2odd), intent(in) :: t
    type(spinor), intent(in) :: psi
    tpsi = f_tvaf (gr+gl, gr-gl, t, psi)
  end function f_tlrf
```

⟨*Implementation of spinor currents*⟩+≡

```
  pure function f_trlf (gr, gl, t, psi) result (tpsi)
    type(spinor) :: tpsi
    complex(kind=default), intent(in) :: gl, gr
    type(tensor2odd), intent(in) :: t
    type(spinor), intent(in) :: psi
    tpsi = f_tvaf (gr+gl, gr-gl, t, psi)
  end function f_trlf
```

⟨*Implementation of spinor currents*⟩+≡

```
  pure function f_tvamf (gv, ga, v, psi, k) result (vpsi)
    type(spinor) :: vpsi
    complex(kind=default), intent(in) :: gv, ga
```

```
    type(vector), intent(in) :: v
    type(spinor), intent(in) :: psi
    type(momentum), intent(in) :: k
    type(tensor2odd) :: t
    t = (v.wedge.k) * (0, 0.5)
    vpsi = f_tvaf(gv, ga, t, psi)
  end function f_tvamf
```

⟨*Implementation of spinor currents*⟩+≡

```
  pure function f_tlrmf (gl, gr, v, psi, k) result (vpsi)
    type(spinor) :: vpsi
    complex(kind=default), intent(in) :: gl, gr
    type(vector), intent(in) :: v
    type(spinor), intent(in) :: psi
    type(momentum), intent(in) :: k
    vpsi = f_tvamf (gr+gl, gr-gl, v, psi, k)
  end function f_tlrmf
```

⟨*Implementation of spinor currents*⟩+≡

```
  pure function f_trlmf (gr, gl, v, psi, k) result (vpsi)
    type(spinor) :: vpsi
    complex(kind=default), intent(in) :: gl, gr
    type(vector), intent(in) :: v
    type(spinor), intent(in) :: psi
    type(momentum), intent(in) :: k
    vpsi = f_tvamf (gr+gl, gr-gl, v, psi, k)
  end function f_trlmf
```

⟨*Implementation of spinor currents*⟩+≡

```
  pure function f_vf (gv, v, psi) result (vpsi)
    type(spinor) :: vpsi
    complex(kind=default), intent(in) :: gv
    type(vector), intent(in) :: v
    type(spinor), intent(in) :: psi
    complex(kind=default) :: vp, vm, v12, v12s
    vp = v%t + v%x(3)
    vm = v%t - v%x(3)
    v12  =  v%x(1) + (0,1)*v%x(2)
    v12s =  v%x(1) - (0,1)*v%x(2)
    vpsi%a(1) = gv * (   vm  * psi%a(3) - v12s * psi%a(4))
    vpsi%a(2) = gv * ( - v12 * psi%a(3) + vp   * psi%a(4))
    vpsi%a(3) = gv * (   vp  * psi%a(1) + v12s * psi%a(2))
    vpsi%a(4) = gv * (   v12 * psi%a(1) + vm   * psi%a(2))
  end function f_vf
```

⟨*Implementation of spinor currents*⟩+≡

```
  pure function f_af (ga, v, psi) result (vpsi)
    type(spinor) :: vpsi
    complex(kind=default), intent(in) :: ga
    type(vector), intent(in) :: v
    type(spinor), intent(in) :: psi
    complex(kind=default) :: vp, vm, v12, v12s
    vp = v%t + v%x(3)
    vm = v%t - v%x(3)
    v12  =  v%x(1) + (0,1)*v%x(2)
    v12s =  v%x(1) - (0,1)*v%x(2)
```

```
    vpsi%a(1) = ga * ( - vm  * psi%a(3) + v12s * psi%a(4))
    vpsi%a(2) = ga * (   v12 * psi%a(3) - vp   * psi%a(4))
    vpsi%a(3) = ga * (   vp  * psi%a(1) + v12s * psi%a(2))
    vpsi%a(4) = ga * (   v12 * psi%a(1) + vm   * psi%a(2))
  end function f_af
```

⟨*Implementation of spinor currents*⟩+≡

```
  pure function f_vlf (gl, v, psi) result (vpsi)
    type(spinor) :: vpsi
    complex(kind=default), intent(in) :: gl
    type(vector), intent(in) :: v
    type(spinor), intent(in) :: psi
    complex(kind=default) :: gl2
    complex(kind=default) :: vp, vm, v12, v12s
    gl2 = 2 * gl
    vp = v%t + v%x(3)
    vm = v%t - v%x(3)
    v12  =  v%x(1) + (0,1)*v%x(2)
    v12s =  v%x(1) - (0,1)*v%x(2)
    vpsi%a(1) = 0
    vpsi%a(2) = 0
    vpsi%a(3) = gl2 * (   vp  * psi%a(1) + v12s * psi%a(2))
    vpsi%a(4) = gl2 * (   v12 * psi%a(1) + vm   * psi%a(2))
  end function f_vlf
```

⟨*Implementation of spinor currents*⟩+≡

```
  pure function f_vrf (gr, v, psi) result (vpsi)
    type(spinor) :: vpsi
    complex(kind=default), intent(in) :: gr
    type(vector), intent(in) :: v
    type(spinor), intent(in) :: psi
    complex(kind=default) :: gr2
    complex(kind=default) :: vp, vm, v12, v12s
    gr2 = 2 * gr
    vp = v%t + v%x(3)
    vm = v%t - v%x(3)
    v12  =  v%x(1) + (0,1)*v%x(2)
    v12s =  v%x(1) - (0,1)*v%x(2)
    vpsi%a(1) = gr2 * (   vm  * psi%a(3) - v12s * psi%a(4))
    vpsi%a(2) = gr2 * ( - v12 * psi%a(3) + vp   * psi%a(4))
    vpsi%a(3) = 0
    vpsi%a(4) = 0
  end function f_vrf
```

⟨*Implementation of spinor currents*⟩+≡

```
  pure function f_vlrf (gl, gr, v, psi) result (vpsi)
    type(spinor) :: vpsi
    complex(kind=default), intent(in) :: gl, gr
    type(vector), intent(in) :: v
    type(spinor), intent(in) :: psi
    vpsi = f_vaf (gl+gr, gl-gr, v, psi)
  end function f_vlrf
```

⟨*Declaration of spinor currents*⟩+≡

```
  public :: f_fva, f_fv, f_fa, f_fvl, f_fvr, f_fvlr, f_fva2, &
            f_ftva, f_ftlr, f_ftrl, f_ftvam, f_ftlrm, f_ftrlm, f_fva3
```

⟨*Implementation of spinor currents*⟩+≡
```
  pure function f_fva (gv, ga, psibar, v) result (psibarv)
    type(conjspinor) :: psibarv
    complex(kind=default), intent(in) :: gv, ga
    type(conjspinor), intent(in) :: psibar
    type(vector), intent(in) :: v
    complex(kind=default) :: gl, gr
    complex(kind=default) :: vp, vm, v12, v12s
    gl = gv + ga
    gr = gv - ga
    vp = v%t + v%x(3)
    vm = v%t - v%x(3)
    v12  =  v%x(1) + (0,1)*v%x(2)
    v12s =  v%x(1) - (0,1)*v%x(2)
    psibarv%a(1) = gl * (   psibar%a(3) * vp   + psibar%a(4) * v12)
    psibarv%a(2) = gl * (   psibar%a(3) * v12s + psibar%a(4) * vm )
    psibarv%a(3) = gr * (   psibar%a(1) * vm   - psibar%a(2) * v12)
    psibarv%a(4) = gr * ( - psibar%a(1) * v12s + psibar%a(2) * vp )
  end function f_fva
```

⟨*Implementation of spinor currents*⟩+≡
```
  pure function f_fva2 (gva, psibar, v) result (psibarv)
    type(conjspinor) :: psibarv
    complex(kind=default), intent(in), dimension(2) :: gva
    type(conjspinor), intent(in) :: psibar
    type(vector), intent(in) :: v
    complex(kind=default) :: gl, gr
    complex(kind=default) :: vp, vm, v12, v12s
    gl = gva(1) + gva(2)
    gr = gva(1) - gva(2)
    vp = v%t + v%x(3)
    vm = v%t - v%x(3)
    v12  =  v%x(1) + (0,1)*v%x(2)
    v12s =  v%x(1) - (0,1)*v%x(2)
    psibarv%a(1) = gl * (   psibar%a(3) * vp   + psibar%a(4) * v12)
    psibarv%a(2) = gl * (   psibar%a(3) * v12s + psibar%a(4) * vm )
    psibarv%a(3) = gr * (   psibar%a(1) * vm   - psibar%a(2) * v12)
    psibarv%a(4) = gr * ( - psibar%a(1) * v12s + psibar%a(2) * vp )
  end function f_fva2
```

⟨*Implementation of spinor currents*⟩+≡
```
  pure function f_fva3 (gv, ga, psibar, v) result (psibarv)
    type(conjspinor) :: psibarv
    complex(kind=default), intent(in) :: gv, ga
    type(conjspinor), intent(in) :: psibar
    type(vector), intent(in) :: v
    complex(kind=default) :: gl, gr
    complex(kind=default) :: vp, vm, v12, v12s
    gl = gv + ga
    gr = gv - ga
    vp =   v%x(3) !+ v%t
    vm = - v%x(3) !+ v%t
    v12  =  v%x(1) + (0,1)*v%x(2)
    v12s =  v%x(1) - (0,1)*v%x(2)
    psibarv%a(1) = gl * (   psibar%a(3) * vp   + psibar%a(4) * v12)
```

```
    psibarv%a(2) = gl * (   psibar%a(3) * v12s + psibar%a(4) * vm )
    psibarv%a(3) = gr * (   psibar%a(1) * vm   - psibar%a(2) * v12)
    psibarv%a(4) = gr * ( - psibar%a(1) * v12s + psibar%a(2) * vp )
  end function f_fva3
```

⟨*Implementation of spinor currents*⟩+≡

```
  pure function f_ftva (gv, ga, psibar, t) result (psibart)
    type(conjspinor) :: psibart
    complex(kind=default), intent(in) :: gv, ga
    type(conjspinor), intent(in) :: psibar
    type(tensor2odd), intent(in) :: t
    complex(kind=default) :: gl, gr
    complex(kind=default) :: e21, e21s, b12, b12s, be3, be3s
    gr   = gv + ga
    gl   = gv - ga
    e21  = t%e(2) + t%e(1)*(0,1)
    e21s = t%e(2) - t%e(1)*(0,1)
    b12  = t%b(1) + t%b(2)*(0,1)
    b12s = t%b(1) - t%b(2)*(0,1)
    be3  = t%b(3) + t%e(3)*(0,1)
    be3s = t%b(3) - t%e(3)*(0,1)
    psibart%a(1) = 2*gl * (   psibar%a(1) * be3  + psibar%a(2) * (-e21s+b12 ))
    psibart%a(2) = 2*gl * ( - psibar%a(2) * be3  + psibar%a(1) * ( e21 +b12s))
    psibart%a(3) = 2*gr * (   psibar%a(3) * be3s + psibar%a(4) * ( e21s+b12 ))
    psibart%a(4) = 2*gr * ( - psibar%a(4) * be3s + psibar%a(3) * (-e21 +b12s))
  end function f_ftva
```

⟨*Implementation of spinor currents*⟩+≡

```
  pure function f_ftlr (gl, gr, psibar, t) result (psibart)
    type(conjspinor) :: psibart
    complex(kind=default), intent(in) :: gl, gr
    type(conjspinor), intent(in) :: psibar
    type(tensor2odd), intent(in) :: t
    psibart = f_ftva (gr+gl, gr-gl, psibar, t)
  end function f_ftlr
```

⟨*Implementation of spinor currents*⟩+≡

```
  pure function f_ftrl (gr, gl, psibar, t) result (psibart)
    type(conjspinor) :: psibart
    complex(kind=default), intent(in) :: gl, gr
    type(conjspinor), intent(in) :: psibar
    type(tensor2odd), intent(in) :: t
    psibart = f_ftva (gr+gl, gr-gl, psibar, t)
  end function f_ftrl
```

⟨*Implementation of spinor currents*⟩+≡

```
  pure function f_ftvam (gv, ga, psibar, v, k) result (psibarv)
    type(conjspinor) :: psibarv
    complex(kind=default), intent(in) :: gv, ga
    type(conjspinor), intent(in) :: psibar
    type(vector), intent(in) :: v
    type(momentum), intent(in) :: k
    type(tensor2odd) :: t
    t = (v.wedge.k) * (0, 0.5)
    psibarv = f_ftva(gv, ga, psibar, t)
  end function f_ftvam
```

⟨*Implementation of spinor currents*⟩+≡
```
pure function f_ftlrm (gl, gr, psibar, v, k) result (psibarv)
  type(conjspinor) :: psibarv
  complex(kind=default), intent(in) :: gl, gr
  type(conjspinor), intent(in) :: psibar
  type(vector), intent(in) :: v
  type(momentum), intent(in) :: k
  psibarv = f_ftvam (gr+gl, gr-gl, psibar, v, k)
end function f_ftlrm
```

⟨*Implementation of spinor currents*⟩+≡
```
pure function f_ftrlm (gr, gl, psibar, v, k) result (psibarv)
  type(conjspinor) :: psibarv
  complex(kind=default), intent(in) :: gl, gr
  type(conjspinor), intent(in) :: psibar
  type(vector), intent(in) :: v
  type(momentum), intent(in) :: k
  psibarv = f_ftvam (gr+gl, gr-gl, psibar, v, k)
end function f_ftrlm
```

⟨*Implementation of spinor currents*⟩+≡
```
pure function f_fv (gv, psibar, v) result (psibarv)
  type(conjspinor) :: psibarv
  complex(kind=default), intent(in) :: gv
  type(conjspinor), intent(in) :: psibar
  type(vector), intent(in) :: v
  complex(kind=default) :: vp, vm, v12, v12s
  vp = v%t + v%x(3)
  vm = v%t - v%x(3)
  v12  =  v%x(1) + (0,1)*v%x(2)
  v12s =  v%x(1) - (0,1)*v%x(2)
  psibarv%a(1) = gv * (   psibar%a(3) * vp   + psibar%a(4) * v12)
  psibarv%a(2) = gv * (   psibar%a(3) * v12s + psibar%a(4) * vm )
  psibarv%a(3) = gv * (   psibar%a(1) * vm   - psibar%a(2) * v12)
  psibarv%a(4) = gv * ( - psibar%a(1) * v12s + psibar%a(2) * vp )
end function f_fv
```

⟨*Implementation of spinor currents*⟩+≡
```
pure function f_fa (ga, psibar, v) result (psibarv)
  type(conjspinor) :: psibarv
  complex(kind=default), intent(in) :: ga
  type(vector), intent(in) :: v
  type(conjspinor), intent(in) :: psibar
  complex(kind=default) :: vp, vm, v12, v12s
  vp = v%t + v%x(3)
  vm = v%t - v%x(3)
  v12  =  v%x(1) + (0,1)*v%x(2)
  v12s =  v%x(1) - (0,1)*v%x(2)
  psibarv%a(1) = ga * (   psibar%a(3) * vp   + psibar%a(4) * v12)
  psibarv%a(2) = ga * (   psibar%a(3) * v12s + psibar%a(4) * vm )
  psibarv%a(3) = ga * ( - psibar%a(1) * vm   + psibar%a(2) * v12)
  psibarv%a(4) = ga * (   psibar%a(1) * v12s - psibar%a(2) * vp )
end function f_fa
```

⟨*Implementation of spinor currents*⟩+≡
```
pure function f_fvl (gl, psibar, v) result (psibarv)
```

```
    type(conjspinor) :: psibarv
    complex(kind=default), intent(in) :: gl
    type(conjspinor), intent(in) :: psibar
    type(vector), intent(in) :: v
    complex(kind=default) :: gl2
    complex(kind=default) :: vp, vm, v12, v12s
    gl2 = 2 * gl
    vp = v%t + v%x(3)
    vm = v%t - v%x(3)
    v12  =  v%x(1) + (0,1)*v%x(2)
    v12s =  v%x(1) - (0,1)*v%x(2)
    psibarv%a(1) = gl2 * (   psibar%a(3) * vp   + psibar%a(4) * v12)
    psibarv%a(2) = gl2 * (   psibar%a(3) * v12s + psibar%a(4) * vm )
    psibarv%a(3) = 0
    psibarv%a(4) = 0
  end function f_fvl
```

⟨*Implementation of spinor currents*⟩+≡

```
  pure function f_fvr (gr, psibar, v) result (psibarv)
    type(conjspinor) :: psibarv
    complex(kind=default), intent(in) :: gr
    type(conjspinor), intent(in) :: psibar
    type(vector), intent(in) :: v
    complex(kind=default) :: gr2
    complex(kind=default) :: vp, vm, v12, v12s
    gr2 = 2 * gr
    vp = v%t + v%x(3)
    vm = v%t - v%x(3)
    v12  =  v%x(1) + (0,1)*v%x(2)
    v12s =  v%x(1) - (0,1)*v%x(2)
    psibarv%a(1) = 0
    psibarv%a(2) = 0
    psibarv%a(3) = gr2 * (   psibar%a(1) * vm   - psibar%a(2) * v12)
    psibarv%a(4) = gr2 * ( - psibar%a(1) * v12s + psibar%a(2) * vp )
  end function f_fvr
```

⟨*Implementation of spinor currents*⟩+≡

```
  pure function f_fvlr (gl, gr, psibar, v) result (psibarv)
    type(conjspinor) :: psibarv
    complex(kind=default), intent(in) :: gl, gr
    type(conjspinor), intent(in) :: psibar
    type(vector), intent(in) :: v
    psibarv = f_fva (gl+gr, gl-gr, psibar, v)
  end function f_fvlr
```

X.13.2 Fermionic Scalar and Pseudo Scalar Couplings

⟨*Declaration of spinor currents*⟩+≡

```
  public :: sp_ff, s_ff, p_ff, sl_ff, sr_ff, slr_ff
```

⟨*Implementation of spinor currents*⟩+≡

```
  pure function sp_ff (gs, gp, psibar, psi) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: gs, gp
    type(conjspinor), intent(in) :: psibar
```

```
    type(spinor), intent(in) :: psi
    j =    (gs - gp) * (psibar%a(1)*psi%a(1) + psibar%a(2)*psi%a(2)) &
         + (gs + gp) * (psibar%a(3)*psi%a(3) + psibar%a(4)*psi%a(4))
  end function sp_ff
```

⟨*Implementation of spinor currents*⟩+≡

```
  pure function s_ff (gs, psibar, psi) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: gs
    type(conjspinor), intent(in) :: psibar
    type(spinor), intent(in) :: psi
    j = gs * (psibar * psi)
  end function s_ff
```

⟨*Implementation of spinor currents*⟩+≡

```
  pure function p_ff (gp, psibar, psi) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: gp
    type(conjspinor), intent(in) :: psibar
    type(spinor), intent(in) :: psi
    j = gp * (  psibar%a(3)*psi%a(3) + psibar%a(4)*psi%a(4) &
              - psibar%a(1)*psi%a(1) - psibar%a(2)*psi%a(2))
  end function p_ff
```

⟨*Implementation of spinor currents*⟩+≡

```
  pure function sl_ff (gl, psibar, psi) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: gl
    type(conjspinor), intent(in) :: psibar
    type(spinor), intent(in) :: psi
    j =  2 * gl * (psibar%a(1)*psi%a(1) + psibar%a(2)*psi%a(2))
  end function sl_ff
```

⟨*Implementation of spinor currents*⟩+≡

```
  pure function sr_ff (gr, psibar, psi) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: gr
    type(conjspinor), intent(in) :: psibar
    type(spinor), intent(in) :: psi
    j = 2 * gr * (psibar%a(3)*psi%a(3) + psibar%a(4)*psi%a(4))
  end function sr_ff
```

$$g_L(1-\gamma_5) + g_R(1+\gamma_5) = (g_R+g_L) + (g_R-g_L)\gamma_5 = g_S + g_P\gamma_5 \tag{X.55}$$

⟨*Implementation of spinor currents*⟩+≡

```
  pure function slr_ff (gl, gr, psibar, psi) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: gl, gr
    type(conjspinor), intent(in) :: psibar
    type(spinor), intent(in) :: psi
    j = sp_ff (gr+gl, gr-gl, psibar, psi)
  end function slr_ff
```

⟨*Declaration of spinor currents*⟩+≡

```
  public :: f_spf, f_sf, f_pf, f_slf, f_srf, f_slrf
```

⟨*Implementation of spinor currents*⟩+≡
```
pure function f_spf (gs, gp, phi, psi) result (phipsi)
  type(spinor) :: phipsi
  complex(kind=default), intent(in) :: gs, gp
  complex(kind=default), intent(in) :: phi
  type(spinor), intent(in) :: psi
  phipsi%a(1:2) = ((gs - gp) * phi) * psi%a(1:2)
  phipsi%a(3:4) = ((gs + gp) * phi) * psi%a(3:4)
end function f_spf
```

⟨*Implementation of spinor currents*⟩+≡
```
pure function f_sf (gs, phi, psi) result (phipsi)
  type(spinor) :: phipsi
  complex(kind=default), intent(in) :: gs
  complex(kind=default), intent(in) :: phi
  type(spinor), intent(in) :: psi
  phipsi%a = (gs * phi) * psi%a
end function f_sf
```

⟨*Implementation of spinor currents*⟩+≡
```
pure function f_pf (gp, phi, psi) result (phipsi)
  type(spinor) :: phipsi
  complex(kind=default), intent(in) :: gp
  complex(kind=default), intent(in) :: phi
  type(spinor), intent(in) :: psi
  phipsi%a(1:2) = (- gp * phi) * psi%a(1:2)
  phipsi%a(3:4) = (  gp * phi) * psi%a(3:4)
end function f_pf
```

⟨*Implementation of spinor currents*⟩+≡
```
pure function f_slf (gl, phi, psi) result (phipsi)
  type(spinor) :: phipsi
  complex(kind=default), intent(in) :: gl
  complex(kind=default), intent(in) :: phi
  type(spinor), intent(in) :: psi
  phipsi%a(1:2) = (2 * gl * phi) * psi%a(1:2)
  phipsi%a(3:4) = 0
end function f_slf
```

⟨*Implementation of spinor currents*⟩+≡
```
pure function f_srf (gr, phi, psi) result (phipsi)
  type(spinor) :: phipsi
  complex(kind=default), intent(in) :: gr
  complex(kind=default), intent(in) :: phi
  type(spinor), intent(in) :: psi
  phipsi%a(1:2) = 0
  phipsi%a(3:4) = (2 * gr * phi) * psi%a(3:4)
end function f_srf
```

⟨*Implementation of spinor currents*⟩+≡
```
pure function f_slrf (gl, gr, phi, psi) result (phipsi)
  type(spinor) :: phipsi
  complex(kind=default), intent(in) :: gl, gr
  complex(kind=default), intent(in) :: phi
  type(spinor), intent(in) :: psi
  phipsi =  f_spf (gr+gl, gr-gl, phi, psi)
end function f_slrf
```

⟨*Declaration of spinor currents*⟩+≡
```
public :: f_fsp, f_fs, f_fp, f_fsl, f_fsr, f_fslr
```

⟨*Implementation of spinor currents*⟩+≡
```
pure function f_fsp (gs, gp, psibar, phi) result (psibarphi)
  type(conjspinor) :: psibarphi
  complex(kind=default), intent(in) :: gs, gp
  type(conjspinor), intent(in) :: psibar
  complex(kind=default), intent(in) :: phi
  psibarphi%a(1:2) = ((gs - gp) * phi) * psibar%a(1:2)
  psibarphi%a(3:4) = ((gs + gp) * phi) * psibar%a(3:4)
end function f_fsp
```

⟨*Implementation of spinor currents*⟩+≡
```
pure function f_fs (gs, psibar, phi) result (psibarphi)
  type(conjspinor) :: psibarphi
  complex(kind=default), intent(in) :: gs
  type(conjspinor), intent(in) :: psibar
  complex(kind=default), intent(in) :: phi
  psibarphi%a = (gs * phi) * psibar%a
end function f_fs
```

⟨*Implementation of spinor currents*⟩+≡
```
pure function f_fp (gp, psibar, phi) result (psibarphi)
  type(conjspinor) :: psibarphi
  complex(kind=default), intent(in) :: gp
  type(conjspinor), intent(in) :: psibar
  complex(kind=default), intent(in) :: phi
  psibarphi%a(1:2) = (- gp * phi) * psibar%a(1:2)
  psibarphi%a(3:4) = (  gp * phi) * psibar%a(3:4)
end function f_fp
```

⟨*Implementation of spinor currents*⟩+≡
```
pure function f_fsl (gl, psibar, phi) result (psibarphi)
  type(conjspinor) :: psibarphi
  complex(kind=default), intent(in) :: gl
  type(conjspinor), intent(in) :: psibar
  complex(kind=default), intent(in) :: phi
  psibarphi%a(1:2) = (2 * gl * phi) * psibar%a(1:2)
  psibarphi%a(3:4) = 0
end function f_fsl
```

⟨*Implementation of spinor currents*⟩+≡
```
pure function f_fsr (gr, psibar, phi) result (psibarphi)
  type(conjspinor) :: psibarphi
  complex(kind=default), intent(in) :: gr
  type(conjspinor), intent(in) :: psibar
  complex(kind=default), intent(in) :: phi
  psibarphi%a(1:2) = 0
  psibarphi%a(3:4) = (2 * gr * phi) * psibar%a(3:4)
end function f_fsr
```

⟨*Implementation of spinor currents*⟩+≡
```
pure function f_fslr (gl, gr, psibar, phi) result (psibarphi)
  type(conjspinor) :: psibarphi
  complex(kind=default), intent(in) :: gl, gr
  type(conjspinor), intent(in) :: psibar
```

```
    complex(kind=default), intent(in) :: phi
    psibarphi = f_fsp (gr+gl, gr-gl, psibar, phi)
  end function f_fslr
```

⟨*Declaration of spinor currents*⟩+≡

```
  public :: f_gravf, f_fgrav
```

⟨*Implementation of spinor currents*⟩+≡

```
  pure function f_gravf (g, m, kb, k, t, psi) result (tpsi)
    type(spinor) :: tpsi
    complex(kind=default), intent(in) :: g
    real(kind=default), intent(in) :: m
    type(spinor), intent(in) :: psi
    type(tensor), intent(in) :: t
    type(momentum), intent(in) :: kb, k
    complex(kind=default) :: g2, g8, t_tr
    type(vector) :: kkb
    kkb = k + kb
    g2 = g / 2.0_default
    g8 = g / 8.0_default
    t_tr = t%t(0,0) - t%t(1,1) - t%t(2,2) - t%t(3,3)
    tpsi = (- f_sf (g2, cmplx (m,0.0, kind=default), psi) &
            - f_vf ((g8*m), kkb, psi)) * t_tr - &
    f_vf (g8,(t*kkb + kkb*t),psi)
  end function f_gravf
```

⟨*Implementation of spinor currents*⟩+≡

```
  pure function f_fgrav (g, m, kb, k, psibar, t) result (psibart)
    type(conjspinor) :: psibart
    complex(kind=default), intent(in) :: g
    real(kind=default), intent(in) :: m
    type(conjspinor), intent(in) :: psibar
    type(tensor), intent(in) :: t
    type(momentum), intent(in) :: kb, k
    type(vector) :: kkb
    complex(kind=default) :: g2, g8, t_tr
    kkb = k + kb
    g2 = g / 2.0_default
    g8 = g / 8.0_default
    t_tr = t%t(0,0) - t%t(1,1) - t%t(2,2) - t%t(3,3)
    psibart = (- f_fs (g2, psibar, cmplx (m, 0.0, kind=default)) &
        - f_fv ((g8 * m), psibar, kkb)) * t_tr - &
          f_fv (g8,psibar,(t*kkb + kkb*t))
  end function f_fgrav
```

X.13.3 On Shell Wave Functions

⟨*Declaration of spinor on shell wave functions*⟩≡

```
  public :: u, ubar, v, vbar
  private :: chi_plus, chi_minus
```

$$\chi_+(\vec{p}) = \frac{1}{\sqrt{2|\vec{p}|(|\vec{p}| + p_3)}} \begin{pmatrix} |\vec{p}| + p_3 \\ p_1 + \mathrm{i}p_2 \end{pmatrix} \tag{X.56a}$$

$$\chi_-(\vec{p}) = \frac{1}{\sqrt{2|\vec{p}|(|\vec{p}|+p_3)}} \begin{pmatrix} -p_1 + \mathrm{i}p_2 \\ |\vec{p}| + p_3 \end{pmatrix} \tag{X.56b}$$

⟨*Implementation of spinor on shell wave functions*⟩≡
```
pure function chi_plus (p) result (chi)
  complex(kind=default), dimension(2) :: chi
  type(momentum), intent(in) :: p
  real(kind=default) :: pabs
  pabs = sqrt (dot_product (p%x, p%x))
  if (pabs + p%x(3) <= 1000 * epsilon (pabs) * pabs) then
     chi = (/ cmplx ( 0.0, 0.0, kind=default), &
              cmplx ( 1.0, 0.0, kind=default) /)
  else
     chi = 1 / sqrt (2*pabs*(pabs + p%x(3))) &
          * (/ cmplx (pabs + p%x(3), kind=default), &
               cmplx (p%x(1), p%x(2), kind=default) /)
  end if
end function chi_plus
```

⟨*Implementation of spinor on shell wave functions*⟩+≡
```
pure function chi_minus (p) result (chi)
  complex(kind=default), dimension(2) :: chi
  type(momentum), intent(in) :: p
  real(kind=default) :: pabs
  pabs = sqrt (dot_product (p%x, p%x))
  if (pabs + p%x(3) <= 1000 * epsilon (pabs) * pabs) then
     chi = (/ cmplx (-1.0, 0.0, kind=default), &
              cmplx ( 0.0, 0.0, kind=default) /)
  else
     chi = 1 / sqrt (2*pabs*(pabs + p%x(3))) &
          * (/ cmplx (-p%x(1), p%x(2), kind=default), &
               cmplx (pabs + p%x(3), kind=default) /)
  end if
end function chi_minus
```

$$u_\pm(p,|m|) = \begin{pmatrix} \sqrt{p_0 \mp |\vec{p}|} \cdot \chi_\pm(\vec{p}) \\ \sqrt{p_0 \pm |\vec{p}|} \cdot \chi_\pm(\vec{p}) \end{pmatrix} \qquad u_\pm(p,-|m|) = \begin{pmatrix} -i\sqrt{p_0 \mp |\vec{p}|} \cdot \chi_\pm(\vec{p}) \\ +i\sqrt{p_0 \pm |\vec{p}|} \cdot \chi_\pm(\vec{p}) \end{pmatrix} \tag{X.57}$$

Determining the mass from the momenta is a numerically haphazardous for light particles. Therefore, we accept some redundancy and pass the mass explicitely. Even if the mass is not used in the chiral representation, we do so for symmetry with polarization vectors and to be prepared for other representations.

⟨*Implementation of spinor on shell wave functions*⟩+≡
```
pure function u (mass, p, s) result (psi)
  type(spinor) :: psi
  real(kind=default), intent(in) :: mass
  type(momentum), intent(in) :: p
  integer, intent(in) :: s
  complex(kind=default), dimension(2) :: chi
  real(kind=default) :: pabs, delta, m
  m = abs(mass)
  pabs = sqrt (dot_product (p%x, p%x))
  if (m < epsilon (m) * pabs) then
```

```
        delta = 0
    else
        delta = sqrt (max (p%t - pabs, 0._default))
    end if
    select case (s)
    case (1)
       chi = chi_plus (p)
       psi%a(1:2) = delta * chi
       psi%a(3:4) = sqrt (p%t + pabs) * chi
    case (-1)
       chi = chi_minus (p)
       psi%a(1:2) = sqrt (p%t + pabs) * chi
       psi%a(3:4) = delta * chi
    case default
       pabs = m ! make the compiler happy and use m
       psi%a = 0
    end select
    if (mass < 0) then
       psi%a(1:2) = - imago * psi%a(1:2)
       psi%a(3:4) = + imago * psi%a(3:4)
    end if
  end function u
```

⟨*Implementation of spinor on shell wave functions*⟩+≡

```
  pure function ubar (m, p, s) result (psibar)
    type(conjspinor) :: psibar
    real(kind=default), intent(in) :: m
    type(momentum), intent(in) :: p
    integer, intent(in) :: s
    type(spinor) :: psi
    psi = u (m, p, s)
    psibar%a(1:2) = conjg (psi%a(3:4))
    psibar%a(3:4) = conjg (psi%a(1:2))
  end function ubar
```

$$v_{\pm}(p) = \begin{pmatrix} \mp\sqrt{p_0 \pm |\vec{p}|} \cdot \chi_{\mp}(\vec{p}) \\ \pm\sqrt{p_0 \mp |\vec{p}|} \cdot \chi_{\mp}(\vec{p}) \end{pmatrix} \tag{X.58}$$

⟨*Implementation of spinor on shell wave functions*⟩+≡

```
  pure function v (mass, p, s) result (psi)
    type(spinor) :: psi
    real(kind=default), intent(in) :: mass
    type(momentum), intent(in) :: p
    integer, intent(in) :: s
    complex(kind=default), dimension(2) :: chi
    real(kind=default) :: pabs, delta, m
    m = abs(mass)
    pabs = sqrt (dot_product (p%x, p%x))
    if (m < epsilon (m) * pabs) then
        delta = 0
    else
        delta = sqrt (max (p%t - pabs, 0._default))
    end if
    select case (s)
    case (1)
```

```
      chi = chi_minus (p)
      psi%a(1:2) = - sqrt (p%t + pabs) * chi
      psi%a(3:4) =   delta * chi
   case (-1)
      chi = chi_plus (p)
      psi%a(1:2) =   delta * chi
      psi%a(3:4) = - sqrt (p%t + pabs) * chi
   case default
      pabs = m ! make the compiler happy and use m
      psi%a = 0
   end select
   if (mass < 0) then
      psi%a(1:2) = - imago * psi%a(1:2)
      psi%a(3:4) = + imago * psi%a(3:4)
    end if
 end function v
```

⟨*Implementation of spinor on shell wave functions*⟩+≡

```
 pure function vbar (m, p, s) result (psibar)
   type(conjspinor) :: psibar
   real(kind=default), intent(in) :: m
   type(momentum), intent(in) :: p
   integer, intent(in) :: s
   type(spinor) :: psi
   psi = v (m, p, s)
   psibar%a(1:2) = conjg (psi%a(3:4))
   psibar%a(3:4) = conjg (psi%a(1:2))
 end function vbar
```

X.13.4 Off Shell Wave Functions

I've just taken this over from Christian Schwinn's version.

⟨*Declaration of spinor off shell wave functions*⟩≡

```
 public :: brs_u, brs_ubar, brs_v, brs_vbar
```

The off-shell wave functions needed for gauge checking are obtained from the LSZ-formulas:

$$\left\langle \mathrm{Out}\middle|d^\dagger\middle|\mathrm{In}\right\rangle = i\int d^4x\bar{v}e^{-ikx}(i\partial\!\!\!/ - m)\left\langle \mathrm{Out}|\psi|\mathrm{In}\right\rangle \tag{X.59a}$$

$$\left\langle \mathrm{Out}|b|\mathrm{In}\right\rangle = -i\int d^4x\bar{u}e^{ikx}(i\partial\!\!\!/ - m)\left\langle \mathrm{Out}|\psi|\mathrm{In}\right\rangle \tag{X.59b}$$

$$\left\langle \mathrm{Out}|d|\mathrm{In}\right\rangle = i\int d^4x\left\langle \mathrm{Out}\middle|\bar{\psi}\middle|\mathrm{In}\right\rangle(-i\overleftarrow{\partial\!\!\!/} - m)ve^{ikx} \tag{X.59c}$$

$$\left\langle \mathrm{Out}\middle|b^\dagger\middle|\mathrm{In}\right\rangle = -i\int d^4x\left\langle \mathrm{Out}\middle|\bar{\psi}\middle|\mathrm{In}\right\rangle(-i\overleftarrow{\partial\!\!\!/} - m)ue^{-ikx} \tag{X.59d}$$

Since the relative sign between fermions and antifermions is ignored for on-shell amplitudes we must also ignore it here, so all wavefunctions must have a $(-i)$ factor. In momentum space we have:

$$brsu(p) = (-i)(p\!\!\!/ - m)u(p) \tag{X.60}$$

⟨*Implementation of spinor off shell wave functions*⟩≡

```
  pure function brs_u (m, p, s) result (dpsi)
      type(spinor) :: dpsi,psi
      real(kind=default), intent(in) :: m
      type(momentum), intent(in) :: p
      integer, intent(in) :: s
      type (vector)::vp
      complex(kind=default), parameter :: one = (1, 0)
      vp=p
      psi=u(m,p,s)
      dpsi=cmplx(0.0,-1.0)*(f_vf(one,vp,psi)-m*psi)
  end function brs_u
```

$$brsv(p) = i(\not{p} + m)v(p) \tag{X.61}$$

⟨*Implementation of spinor off shell wave functions*⟩+≡

```
  pure function brs_v (m, p, s) result (dpsi)
      type(spinor) :: dpsi, psi
      real(kind=default), intent(in) :: m
      type(momentum), intent(in) :: p
      integer, intent(in) ::   s
      type (vector)::vp
      complex(kind=default), parameter :: one = (1, 0)
      vp=p
      psi=v(m,p,s)
      dpsi=cmplx(0.0,1.0)*(f_vf(one,vp,psi)+m*psi)
  end function brs_v
```

$$brs\bar{u}(p) = (-i)\bar{u}(p)(\not{p} - m) \tag{X.62}$$

⟨*Implementation of spinor off shell wave functions*⟩+≡

```
   pure function brs_ubar (m, p, s)result (dpsibar)
      type(conjspinor) :: dpsibar, psibar
      real(kind=default), intent(in) :: m
      type(momentum), intent(in) :: p
      integer, intent(in) :: s
      type (vector)::vp
      complex(kind=default), parameter :: one = (1, 0)
       vp=p
       psibar=ubar(m,p,s)
      dpsibar=cmplx(0.0,-1.0)*(f_fv(one,psibar,vp)-m*psibar)
    end function brs_ubar
```

$$brs\bar{v}(p) = (i)\bar{v}(p)(\not{p} + m) \tag{X.63}$$

⟨*Implementation of spinor off shell wave functions*⟩+≡

```
   pure function brs_vbar (m, p, s) result (dpsibar)
      type(conjspinor) :: dpsibar,psibar
      real(kind=default), intent(in) :: m
      type(momentum), intent(in) :: p
      integer, intent(in) :: s
      type(vector)::vp
      complex(kind=default), parameter :: one = (1, 0)
      vp=p
      psibar=vbar(m,p,s)
```

```
    dpsibar=cmplx(0.0,1.0)*(f_fv(one,psibar,vp)+m*psibar)
  end function brs_vbar
```

NB: The remarks on momentum flow in the propagators don't apply here since the incoming momenta are flipped for the wave functions.

X.13.5 Propagators

NB: the common factor of i is extracted:

⟨*Declaration of spinor propagators*⟩≡

```
  public :: pr_psi, pr_psibar
  public :: pj_psi, pj_psibar
  public :: pg_psi, pg_psibar
```

$$\frac{i(-\not p+m)}{p^2-m^2+\mathrm{i}m\Gamma}\psi \tag{X.64}$$

NB: the sign of the momentum comes about because all momenta are treated as *outgoing* and the particle charge flow is therefore opposite to the momentum.

⟨*Implementation of spinor propagators*⟩≡

```
  pure function pr_psi (p, m, w, psi) result (ppsi)
    type(spinor) :: ppsi
    type(momentum), intent(in) :: p
    real(kind=default), intent(in) :: m, w
    type(spinor), intent(in) :: psi
    type(vector) :: vp
    complex(kind=default), parameter :: one = (1, 0)
    vp = p
    ppsi = (1 / cmplx (p*p - m**2, m*w, kind=default)) &
         * (- f_vf (one, vp, psi) + m * psi)
  end function pr_psi
```

$$\sqrt{\frac{\pi}{M\Gamma}}(-\not p+m)\psi \tag{X.65}$$

⟨*Implementation of spinor propagators*⟩+≡

```
  pure function pj_psi (p, m, w, psi) result (ppsi)
    type(spinor) :: ppsi
    type(momentum), intent(in) :: p
    real(kind=default), intent(in) :: m, w
    type(spinor), intent(in) :: psi
    type(vector) :: vp
    complex(kind=default), parameter :: one = (1, 0)
    vp = p
    ppsi = (0, -1) * sqrt (PI / m / w) * (- f_vf (one, vp, psi) + m * psi)
  end function pj_psi
```

⟨*Implementation of spinor propagators*⟩+≡

```
  pure function pg_psi (p, m, w, psi) result (ppsi)
    type(spinor) :: ppsi
    type(momentum), intent(in) :: p
    real(kind=default), intent(in) :: m, w
    type(spinor), intent(in) :: psi
    type(vector) :: vp
    complex(kind=default), parameter :: one = (1, 0)
```

```
    vp = p
    ppsi = gauss(p*p, m, w) *  (- f_vf (one, vp, psi) + m * psi)
  end function pg_psi
```

$$\bar{\psi}\frac{i(\not p + m)}{p^2 - m^2 + \mathrm{i}m\Gamma} \tag{X.66}$$

NB: the sign of the momentum comes about because all momenta are treated as *outgoing* and the antiparticle charge flow is therefore parallel to the momentum.

⟨*Implementation of spinor propagators*⟩+≡

```
  pure function pr_psibar (p, m, w, psibar) result (ppsibar)
    type(conjspinor) :: ppsibar
    type(momentum), intent(in) :: p
    real(kind=default), intent(in) :: m, w
    type(conjspinor), intent(in) :: psibar
    type(vector) :: vp
    complex(kind=default), parameter :: one = (1, 0)
    vp = p
    ppsibar = (1 / cmplx (p*p - m**2, m*w, kind=default)) &
         * (f_fv (one, psibar, vp) + m * psibar)
  end function pr_psibar
```

$$\sqrt{\frac{\pi}{M\Gamma}}\bar{\psi}(\not p + m) \tag{X.67}$$

NB: the sign of the momentum comes about because all momenta are treated as *outgoing* and the antiparticle charge flow is therefore parallel to the momentum.

⟨*Implementation of spinor propagators*⟩+≡

```
  pure function pj_psibar (p, m, w, psibar) result (ppsibar)
    type(conjspinor) :: ppsibar
    type(momentum), intent(in) :: p
    real(kind=default), intent(in) :: m, w
    type(conjspinor), intent(in) :: psibar
    type(vector) :: vp
    complex(kind=default), parameter :: one = (1, 0)
    vp = p
    ppsibar = (0, -1) * sqrt (PI / m / w) * (f_fv (one, psibar, vp) + m * psibar)
  end function pj_psibar
```

⟨*Implementation of spinor propagators*⟩+≡

```
  pure function pg_psibar (p, m, w, psibar) result (ppsibar)
    type(conjspinor) :: ppsibar
    type(momentum), intent(in) :: p
    real(kind=default), intent(in) :: m, w
    type(conjspinor), intent(in) :: psibar
    type(vector) :: vp
    complex(kind=default), parameter :: one = (1, 0)
    vp = p
    ppsibar = gauss (p*p, m, w) * (f_fv (one, psibar, vp) + m * psibar)
  end function pg_psibar
```

$$\frac{i(-\not p + m)}{p^2 - m^2 + \mathrm{i}m\Gamma}\sum_n \psi_n \otimes \bar{\psi}_n \tag{X.68}$$

NB: the temporary variables `psi(1:4)` are not nice, but the compilers should be able to optimize the unnecessary copies away. In any case, even if the copies are performed, they are (probably) negligible compared to the floating point multiplications anyway ...

⟨*(Not used yet) Declaration of operations for spinors*⟩≡
```
type, public :: spinordyad
   ! private (omegalib needs access, but DON'T TOUCH IT!)
   complex(kind=default), dimension(4,4) :: a
end type spinordyad
```

⟨*(Not used yet) Implementation of spinor propagators*⟩≡
```
pure function pr_dyadleft (p, m, w, psipsibar) result (psipsibarp)
  type(spinordyad) :: psipsibarp
  type(momentum), intent(in) :: p
  real(kind=default), intent(in) :: m, w
  type(spinordyad), intent(in) :: psipsibar
  integer :: i
  type(vector) :: vp
  type(spinor), dimension(4) :: psi
  complex(kind=default) :: pole
  complex(kind=default), parameter :: one = (1, 0)
  vp = p
  pole = 1 / cmplx (p*p - m**2, m*w, kind=default)
  do i = 1, 4
     psi(i)%a = psipsibar%a(:,i)
     psi(i) = pole * (- f_vf (one, vp, psi(i)) + m * psi(i))
     psipsibarp%a(:,i) = psi(i)%a
  end do
end function pr_dyadleft
```

$$\sum_n \psi_n \otimes \bar{\psi}_n \frac{i(\not p + m)}{p^2 - m^2 + im\Gamma} \tag{X.69}$$

⟨*(Not used yet) Implementation of spinor propagators*⟩+≡
```
pure function pr_dyadright (p, m, w, psipsibar) result (psipsibarp)
  type(spinordyad) :: psipsibarp
  type(momentum), intent(in) :: p
  real(kind=default), intent(in) :: m, w
  type(spinordyad), intent(in) :: psipsibar
  integer :: i
  type(vector) :: vp
  type(conjspinor), dimension(4) :: psibar
  complex(kind=default) :: pole
  complex(kind=default), parameter :: one = (1, 0)
  vp = p
  pole = 1 / cmplx (p*p - m**2, m*w, kind=default)
  do i = 1, 4
     psibar(i)%a = psipsibar%a(i,:)
     psibar(i) = pole * (f_fv (one, psibar(i), vp) + m * psibar(i))
     psipsibarp%a(i,:) = psibar(i)%a
  end do
end function pr_dyadright
```

X.14 Spinor Couplings Revisited

⟨omega_bispinor_couplings.f90⟩≡
```
  ⟨Copyleft⟩
  module omega_bispinor_couplings
    use kinds
    use constants
    use omega_bispinors
    use omega_vectorspinors
    use omega_vectors
    use omega_couplings
    implicit none
    private
    ⟨Declaration of bispinor on shell wave functions⟩
    ⟨Declaration of bispinor off shell wave functions⟩
    ⟨Declaration of bispinor currents⟩
    ⟨Declaration of bispinor propagators⟩
    integer, parameter, public :: omega_bispinor_cpls_2010_01_A = 0
  contains
    ⟨Implementation of bispinor on shell wave functions⟩
    ⟨Implementation of bispinor off shell wave functions⟩
    ⟨Implementation of bispinor currents⟩
    ⟨Implementation of bispinor propagators⟩
  end module omega_bispinor_couplings
```

See table X.1 for the names of Fortran functions. We could have used long names instead, but this would increase the chance of running past continuation line limits without adding much to the legibility.

X.14.1 Fermionic Vector and Axial Couplings

There's more than one chiral representation. This one is compatible with HELAS [5].

$$\gamma^0 = \begin{pmatrix} 0 & \mathbf{1} \\ \mathbf{1} & 0 \end{pmatrix},\ \gamma^i = \begin{pmatrix} 0 & \sigma^i \\ -\sigma^i & 0 \end{pmatrix},\ \gamma_5 = i\gamma^0\gamma^1\gamma^2\gamma^3 = \begin{pmatrix} -\mathbf{1} & 0 \\ 0 & \mathbf{1} \end{pmatrix}, \tag{X.70a}$$

$$C = \begin{pmatrix} \epsilon & 0 \\ 0 & -\epsilon \end{pmatrix}, \qquad \epsilon = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}. \tag{X.70b}$$

Therefore

$$g_S + g_P\gamma_5 = \begin{pmatrix} g_S - g_P & 0 & 0 & 0 \\ 0 & g_S - g_P & 0 & 0 \\ 0 & 0 & g_S + g_P & 0 \\ 0 & 0 & 0 & g_S + g_P \end{pmatrix} \tag{X.71a}$$

$$g_V\gamma^0 - g_A\gamma^0\gamma_5 = \begin{pmatrix} 0 & 0 & g_V - g_A & 0 \\ 0 & 0 & 0 & g_V - g_A \\ g_V + g_A & 0 & 0 & 0 \\ 0 & g_V + g_A & 0 & 0 \end{pmatrix} \tag{X.71b}$$

$$g_V\gamma^1 - g_A\gamma^1\gamma_5 = \begin{pmatrix} 0 & 0 & 0 & g_V - g_A \\ 0 & 0 & g_V - g_A & 0 \\ 0 & -g_V - g_A & 0 & 0 \\ -g_V - g_A & 0 & 0 & 0 \end{pmatrix} \tag{X.71c}$$

$$g_V\gamma^2 - g_A\gamma^2\gamma_5 = \begin{pmatrix} 0 & 0 & 0 & -\mathrm{i}(g_V-g_A) \\ 0 & 0 & \mathrm{i}(g_V-g_A) & 0 \\ 0 & \mathrm{i}(g_V+g_A) & 0 & 0 \\ -\mathrm{i}(g_V+g_A) & 0 & 0 & 0 \end{pmatrix} \tag{X.71d}$$

$$g_V\gamma^3 - g_A\gamma^3\gamma_5 = \begin{pmatrix} 0 & 0 & g_V-g_A & 0 \\ 0 & 0 & 0 & -g_V+g_A \\ -g_V-g_A & 0 & 0 & 0 \\ 0 & g_V+g_A & 0 & 0 \end{pmatrix} \tag{X.71e}$$

and

$$C(g_S + g_P\gamma_5) = \begin{pmatrix} 0 & g_S-g_P & 0 & 0 \\ -g_S+g_P & 0 & 0 & 0 \\ 0 & 0 & 0 & -g_S-g_P \\ 0 & 0 & g_S+g_P & 0 \end{pmatrix} \tag{X.72a}$$

$$C(g_V\gamma^0 - g_A\gamma^0\gamma_5) = \begin{pmatrix} 0 & 0 & 0 & g_V-g_A \\ 0 & 0 & -g_V+g_A & 0 \\ 0 & -g_V-g_A & 0 & 0 \\ g_V+g_A & 0 & 0 & 0 \end{pmatrix} \tag{X.72b}$$

$$C(g_V\gamma^1 - g_A\gamma^1\gamma_5) = \begin{pmatrix} 0 & 0 & g_V-g_A & 0 \\ 0 & 0 & 0 & -g_V+g_A \\ g_V+g_A & 0 & 0 & 0 \\ 0 & -g_V-g_A & 0 & 0 \end{pmatrix} \tag{X.72c}$$

$$C(g_V\gamma^2 - g_A\gamma^2\gamma_5) = \begin{pmatrix} 0 & 0 & \mathrm{i}(g_V-g_A) & 0 \\ 0 & 0 & 0 & \mathrm{i}(g_V-g_A) \\ \mathrm{i}(g_V+g_A) & 0 & 0 & 0 \\ 0 & \mathrm{i}(g_V+g_A) & 0 & 0 \end{pmatrix} \tag{X.72d}$$

$$C(g_V\gamma^3 - g_A\gamma^3\gamma_5) = \begin{pmatrix} 0 & 0 & 0 & -g_V+g_A \\ 0 & 0 & -g_V+g_A & 0 \\ 0 & -g_V-g_A & 0 & 0 \\ -g_V-g_A & 0 & 0 & 0 \end{pmatrix} \tag{X.72e}$$

⟨*Declaration of bispinor currents*⟩≡

```
public :: va_ff, v_ff, a_ff, vl_ff, vr_ff, vlr_ff, va2_ff
```

⟨*Implementation of bispinor currents*⟩≡

```
pure function va_ff (gv, ga, psil, psir) result (j)
  type(vector) :: j
  complex(kind=default), intent(in) :: gv, ga
  type(bispinor), intent(in) :: psil, psir
  complex(kind=default) :: gl, gr
  complex(kind=default) :: g13, g14, g23, g24, g31, g32, g41, g42
  gl = gv + ga
  gr = gv - ga
  g13 = psil%a(1)*psir%a(3)
  g14 = psil%a(1)*psir%a(4)
  g23 = psil%a(2)*psir%a(3)
  g24 = psil%a(2)*psir%a(4)
```

```
    g31 = psil%a(3)*psir%a(1)
    g32 = psil%a(3)*psir%a(2)
    g41 = psil%a(4)*psir%a(1)
    g42 = psil%a(4)*psir%a(2)
    j%t    =  gr * (   g14 - g23) + gl * ( - g32 + g41)
    j%x(1) =  gr * (   g13 - g24) + gl * (   g31 - g42)
    j%x(2) = (gr * (   g13 + g24) + gl * (   g31 + g42)) * (0, 1)
    j%x(3) =  gr * ( - g14 - g23) + gl * ( - g32 - g41)
  end function va_ff
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function va2_ff (gva, psil, psir) result (j)
    type(vector) :: j
    complex(kind=default), intent(in), dimension(2) :: gva
    type(bispinor), intent(in) :: psil, psir
    complex(kind=default) :: gl, gr
    complex(kind=default) :: g13, g14, g23, g24, g31, g32, g41, g42
    gl = gva(1) + gva(2)
    gr = gva(1) - gva(2)
    g13 = psil%a(1)*psir%a(3)
    g14 = psil%a(1)*psir%a(4)
    g23 = psil%a(2)*psir%a(3)
    g24 = psil%a(2)*psir%a(4)
    g31 = psil%a(3)*psir%a(1)
    g32 = psil%a(3)*psir%a(2)
    g41 = psil%a(4)*psir%a(1)
    g42 = psil%a(4)*psir%a(2)
    j%t    =  gr * (   g14 - g23) + gl * ( - g32 + g41)
    j%x(1) =  gr * (   g13 - g24) + gl * (   g31 - g42)
    j%x(2) = (gr * (   g13 + g24) + gl * (   g31 + g42)) * (0, 1)
    j%x(3) =  gr * ( - g14 - g23) + gl * ( - g32 - g41)
  end function va2_ff
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function v_ff (gv, psil, psir) result (j)
    type(vector) :: j
    complex(kind=default), intent(in) :: gv
    type(bispinor), intent(in) :: psil, psir
    complex(kind=default) :: g13, g14, g23, g24, g31, g32, g41, g42
    g13 = psil%a(1)*psir%a(3)
    g14 = psil%a(1)*psir%a(4)
    g23 = psil%a(2)*psir%a(3)
    g24 = psil%a(2)*psir%a(4)
    g31 = psil%a(3)*psir%a(1)
    g32 = psil%a(3)*psir%a(2)
    g41 = psil%a(4)*psir%a(1)
    g42 = psil%a(4)*psir%a(2)
    j%t    =   gv * (   g14 - g23 - g32 + g41)
    j%x(1) =   gv * (   g13 - g24 + g31 - g42)
    j%x(2) =   gv * (   g13 + g24 + g31 + g42) * (0, 1)
    j%x(3) =   gv * ( - g14 - g23 - g32 - g41)
  end function v_ff
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function a_ff (ga, psil, psir) result (j)
    type(vector) :: j
```

```
    complex(kind=default), intent(in) :: ga
    type(bispinor), intent(in) :: psil, psir
    complex(kind=default) :: g13, g14, g23, g24, g31, g32, g41, g42
    g13 = psil%a(1)*psir%a(3)
    g14 = psil%a(1)*psir%a(4)
    g23 = psil%a(2)*psir%a(3)
    g24 = psil%a(2)*psir%a(4)
    g31 = psil%a(3)*psir%a(1)
    g32 = psil%a(3)*psir%a(2)
    g41 = psil%a(4)*psir%a(1)
    g42 = psil%a(4)*psir%a(2)
    j%t    =  -ga * (   g14 - g23 + g32 - g41)
    j%x(1) =  -ga * (   g13 - g24 - g31 + g42)
    j%x(2) =  -ga * (   g13 + g24 - g31 - g42) * (0, 1)
    j%x(3) =  -ga * ( - g14 - g23 + g32 + g41)
  end function a_ff
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function vl_ff (gl, psil, psir) result (j)
    type(vector) :: j
    complex(kind=default), intent(in) :: gl
    type(bispinor), intent(in) :: psil, psir
    complex(kind=default) :: gl2
    complex(kind=default) :: g31, g32, g41, g42
    gl2 = 2 * gl
    g31 = psil%a(3)*psir%a(1)
    g32 = psil%a(3)*psir%a(2)
    g41 = psil%a(4)*psir%a(1)
    g42 = psil%a(4)*psir%a(2)
    j%t    =   gl2 * ( - g32 + g41)
    j%x(1) =   gl2 * (   g31 - g42)
    j%x(2) =   gl2 * (   g31 + g42) * (0, 1)
    j%x(3) =   gl2 * ( - g32 - g41)
  end function vl_ff
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function vr_ff (gr, psil, psir) result (j)
    type(vector) :: j
    complex(kind=default), intent(in) :: gr
    type(bispinor), intent(in) :: psil, psir
    complex(kind=default) :: gr2
    complex(kind=default) :: g13, g14, g23, g24
    gr2 = 2 * gr
    g13 = psil%a(1)*psir%a(3)
    g14 = psil%a(1)*psir%a(4)
    g23 = psil%a(2)*psir%a(3)
    g24 = psil%a(2)*psir%a(4)
    j%t    = gr2 * (   g14 - g23)
    j%x(1) = gr2 * (   g13 - g24)
    j%x(2) = gr2 * (   g13 + g24) * (0, 1)
    j%x(3) = gr2 * ( - g14 - g23)
  end function vr_ff
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function vlr_ff (gl, gr, psibar, psi) result (j)
    type(vector) :: j
```

```
    complex(kind=default), intent(in) :: gl, gr
    type(bispinor), intent(in) :: psibar
    type(bispinor), intent(in) :: psi
    j = va_ff (gl+gr, gl-gr, psibar, psi)
  end function vlr_ff
```

and

$$\not v - \not a\gamma_5 = \begin{pmatrix} 0 & 0 & v_- - a_- & -v^* + a^* \\ 0 & 0 & -v + a & v_+ - a_+ \\ v_+ + a_+ & v^* + a^* & 0 & 0 \\ v + a & v_- + a_- & 0 & 0 \end{pmatrix} \tag{X.73}$$

with $v_\pm = v_0 \pm v_3$, $a_\pm = a_0 \pm a_3$, $v = v_1 + \mathrm{i}v_2$, $v^* = v_1 - \mathrm{i}v_2$, $a = a_1 + \mathrm{i}a_2$, and $a^* = a_1 - \mathrm{i}a_2$. But note that $\cdot^*$ is *not* complex conjugation for complex v_μ or a_μ.

⟨*Declaration of bispinor currents*⟩+≡

```
  public :: f_vaf, f_vf, f_af, f_vlf, f_vrf, f_vlrf, f_va2f
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function f_vaf (gv, ga, v, psi) result (vpsi)
    type(bispinor) :: vpsi
    complex(kind=default), intent(in) :: gv, ga
    type(vector), intent(in) :: v
    type(bispinor), intent(in) :: psi
    complex(kind=default) :: gl, gr
    complex(kind=default) :: vp, vm, v12, v12s
    gl = gv + ga
    gr = gv - ga
    vp = v%t + v%x(3)
    vm = v%t - v%x(3)
    v12  =  v%x(1) + (0,1)*v%x(2)
    v12s =  v%x(1) - (0,1)*v%x(2)
    vpsi%a(1) = gr * (   vm  * psi%a(3) - v12s * psi%a(4))
    vpsi%a(2) = gr * ( - v12 * psi%a(3) + vp   * psi%a(4))
    vpsi%a(3) = gl * (   vp  * psi%a(1) + v12s * psi%a(2))
    vpsi%a(4) = gl * (   v12 * psi%a(1) + vm   * psi%a(2))
  end function f_vaf
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function f_va2f (gva, v, psi) result (vpsi)
    type(bispinor) :: vpsi
    complex(kind=default), intent(in), dimension(2) :: gva
    type(vector), intent(in) :: v
    type(bispinor), intent(in) :: psi
    complex(kind=default) :: gl, gr
    complex(kind=default) :: vp, vm, v12, v12s
    gl = gva(1) + gva(2)
    gr = gva(1) - gva(2)
    vp = v%t + v%x(3)
    vm = v%t - v%x(3)
    v12  =  v%x(1) + (0,1)*v%x(2)
    v12s =  v%x(1) - (0,1)*v%x(2)
    vpsi%a(1) = gr * (   vm  * psi%a(3) - v12s * psi%a(4))
    vpsi%a(2) = gr * ( - v12 * psi%a(3) + vp   * psi%a(4))
    vpsi%a(3) = gl * (   vp  * psi%a(1) + v12s * psi%a(2))
```

```
    vpsi%a(4) = gl * (   v12 * psi%a(1) + vm   * psi%a(2))
  end function f_va2f
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function f_vf (gv, v, psi) result (vpsi)
    type(bispinor) :: vpsi
    complex(kind=default), intent(in) :: gv
    type(vector), intent(in) :: v
    type(bispinor), intent(in) :: psi
    complex(kind=default) :: vp, vm, v12, v12s
    vp = v%t + v%x(3)
    vm = v%t - v%x(3)
    v12  =  v%x(1) + (0,1)*v%x(2)
    v12s =  v%x(1) - (0,1)*v%x(2)
    vpsi%a(1) = gv * (   vm  * psi%a(3) - v12s * psi%a(4))
    vpsi%a(2) = gv * ( - v12 * psi%a(3) + vp   * psi%a(4))
    vpsi%a(3) = gv * (   vp  * psi%a(1) + v12s * psi%a(2))
    vpsi%a(4) = gv * (   v12 * psi%a(1) + vm   * psi%a(2))
  end function f_vf
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function f_af (ga, v, psi) result (vpsi)
    type(bispinor) :: vpsi
    complex(kind=default), intent(in) :: ga
    type(vector), intent(in) :: v
    type(bispinor), intent(in) :: psi
    complex(kind=default) :: vp, vm, v12, v12s
    vp = v%t + v%x(3)
    vm = v%t - v%x(3)
    v12  =  v%x(1) + (0,1)*v%x(2)
    v12s =  v%x(1) - (0,1)*v%x(2)
    vpsi%a(1) = ga * ( - vm  * psi%a(3) + v12s * psi%a(4))
    vpsi%a(2) = ga * (   v12 * psi%a(3) - vp   * psi%a(4))
    vpsi%a(3) = ga * (   vp  * psi%a(1) + v12s * psi%a(2))
    vpsi%a(4) = ga * (   v12 * psi%a(1) + vm   * psi%a(2))
  end function f_af
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function f_vlf (gl, v, psi) result (vpsi)
    type(bispinor) :: vpsi
    complex(kind=default), intent(in) :: gl
    type(vector), intent(in) :: v
    type(bispinor), intent(in) :: psi
    complex(kind=default) :: gl2
    complex(kind=default) :: vp, vm, v12, v12s
    gl2 = 2 * gl
    vp = v%t + v%x(3)
    vm = v%t - v%x(3)
    v12  =  v%x(1) + (0,1)*v%x(2)
    v12s =  v%x(1) - (0,1)*v%x(2)
    vpsi%a(1) = 0
    vpsi%a(2) = 0
    vpsi%a(3) = gl2 * (   vp  * psi%a(1) + v12s * psi%a(2))
    vpsi%a(4) = gl2 * (   v12 * psi%a(1) + vm   * psi%a(2))
  end function f_vlf
```

⟨*Implementation of bispinor currents*⟩+≡
```
  pure function f_vrf (gr, v, psi) result (vpsi)
    type(bispinor) :: vpsi
    complex(kind=default), intent(in) :: gr
    type(vector), intent(in) :: v
    type(bispinor), intent(in) :: psi
    complex(kind=default) :: gr2
    complex(kind=default) :: vp, vm, v12, v12s
    gr2 = 2 * gr
    vp = v%t + v%x(3)
    vm = v%t - v%x(3)
    v12  =  v%x(1) + (0,1)*v%x(2)
    v12s =  v%x(1) - (0,1)*v%x(2)
    vpsi%a(1) = gr2 * (    vm  * psi%a(3) - v12s * psi%a(4))
    vpsi%a(2) = gr2 * ( - v12 * psi%a(3) + vp   * psi%a(4))
    vpsi%a(3) = 0
    vpsi%a(4) = 0
  end function f_vrf
```

⟨*Implementation of bispinor currents*⟩+≡
```
  pure function f_vlrf (gl, gr, v, psi) result (vpsi)
    type(bispinor) :: vpsi
    complex(kind=default), intent(in) :: gl, gr
    type(vector), intent(in) :: v
    type(bispinor), intent(in) :: psi
    vpsi = f_vaf (gl+gr, gl-gr, v, psi)
  end function f_vlrf
```

X.14.2 Fermionic Scalar and Pseudo Scalar Couplings

⟨*Declaration of bispinor currents*⟩+≡
```
  public :: sp_ff, s_ff, p_ff, sl_ff, sr_ff, slr_ff
```

⟨*Implementation of bispinor currents*⟩+≡
```
  pure function sp_ff (gs, gp, psil, psir) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: gs, gp
    type(bispinor), intent(in) :: psil, psir
    j =     (gs - gp) * (psil%a(1)*psir%a(2) - psil%a(2)*psir%a(1)) &
         + (gs + gp) * (- psil%a(3)*psir%a(4) + psil%a(4)*psir%a(3))
  end function sp_ff
```

⟨*Implementation of bispinor currents*⟩+≡
```
  pure function s_ff (gs, psil, psir) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: gs
    type(bispinor), intent(in) :: psil, psir
    j = gs * (psil * psir)
  end function s_ff
```

⟨*Implementation of bispinor currents*⟩+≡
```
  pure function p_ff (gp, psil, psir) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: gp
    type(bispinor), intent(in) :: psil, psir
    j = gp * (- psil%a(1)*psir%a(2) + psil%a(2)*psir%a(1) &
```

```
             - psil%a(3)*psir%a(4) + psil%a(4)*psir%a(3))
  end function p_ff
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function sl_ff (gl, psil, psir) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: gl
    type(bispinor), intent(in) :: psil, psir
    j =  2 * gl * (psil%a(1)*psir%a(2) - psil%a(2)*psir%a(1))
  end function sl_ff
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function sr_ff (gr, psil, psir) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: gr
    type(bispinor), intent(in) :: psil, psir
    j = 2 * gr * (- psil%a(3)*psir%a(4) + psil%a(4)*psir%a(3))
  end function sr_ff
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function slr_ff (gl, gr, psibar, psi) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: gl, gr
    type(bispinor), intent(in) :: psibar
    type(bispinor), intent(in) :: psi
    j = sp_ff (gr+gl, gr-gl, psibar, psi)
  end function slr_ff
```

⟨*Declaration of bispinor currents*⟩+≡

```
  public :: f_spf, f_sf, f_pf, f_slf, f_srf, f_slrf
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function f_spf (gs, gp, phi, psi) result (phipsi)
    type(bispinor) :: phipsi
    complex(kind=default), intent(in) :: gs, gp
    complex(kind=default), intent(in) :: phi
    type(bispinor), intent(in) :: psi
    phipsi%a(1:2) = ((gs - gp) * phi) * psi%a(1:2)
    phipsi%a(3:4) = ((gs + gp) * phi) * psi%a(3:4)
  end function f_spf
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function f_sf (gs, phi, psi) result (phipsi)
    type(bispinor) :: phipsi
    complex(kind=default), intent(in) :: gs
    complex(kind=default), intent(in) :: phi
    type(bispinor), intent(in) :: psi
    phipsi%a = (gs * phi) * psi%a
  end function f_sf
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function f_pf (gp, phi, psi) result (phipsi)
    type(bispinor) :: phipsi
    complex(kind=default), intent(in) :: gp
    complex(kind=default), intent(in) :: phi
    type(bispinor), intent(in) :: psi
    phipsi%a(1:2) = (- gp * phi) * psi%a(1:2)
    phipsi%a(3:4) = (  gp * phi) * psi%a(3:4)
```

```
  end function f_pf
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function f_slf (gl, phi, psi) result (phipsi)
    type(bispinor) :: phipsi
    complex(kind=default), intent(in) :: gl
    complex(kind=default), intent(in) :: phi
    type(bispinor), intent(in) :: psi
    phipsi%a(1:2) = (2 * gl * phi) * psi%a(1:2)
    phipsi%a(3:4) = 0
  end function f_slf
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function f_srf (gr, phi, psi) result (phipsi)
    type(bispinor) :: phipsi
    complex(kind=default), intent(in) :: gr
    complex(kind=default), intent(in) :: phi
    type(bispinor), intent(in) :: psi
    phipsi%a(1:2) = 0
    phipsi%a(3:4) = (2 * gr * phi) * psi%a(3:4)
  end function f_srf
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function f_slrf (gl, gr, phi, psi) result (phipsi)
    type(bispinor) :: phipsi
    complex(kind=default), intent(in) :: gl, gr
    complex(kind=default), intent(in) :: phi
    type(bispinor), intent(in) :: psi
    phipsi =  f_spf (gr+gl, gr-gl, phi, psi)
  end function f_slrf
```

X.14.3 Couplings for BRST Transformations

3-Couplings

The lists of needed gamma matrices can be found in the next subsection with the gravitino couplings.

⟨*Declaration of bispinor currents*⟩+≡

```
  private :: vv_ff, f_vvf
```

⟨*Declaration of bispinor currents*⟩+≡

```
  public :: vmom_ff, mom_ff, mom5_ff, moml_ff, momr_ff, lmom_ff, rmom_ff
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function vv_ff (psibar, psi, k) result (psibarpsi)
    type(vector) :: psibarpsi
    type(bispinor), intent(in) :: psibar, psi
    type(vector), intent(in) :: k
    complex(kind=default) :: kp, km, k12, k12s
    type(bispinor) :: kgpsi1, kgpsi2, kgpsi3, kgpsi4
    kp = k%t + k%x(3)
    km = k%t - k%x(3)
    k12  =  k%x(1) + (0,1)*k%x(2)
    k12s =  k%x(1) - (0,1)*k%x(2)
    kgpsi1%a(1) = -k%x(3) * psi%a(1) - k12s * psi%a(2)
    kgpsi1%a(2) = -k12 * psi%a(1) + k%x(3) * psi%a(2)
```

```
    kgpsi1%a(3) = k%x(3) * psi%a(3) + k12s * psi%a(4)
    kgpsi1%a(4) = k12 * psi%a(3) - k%x(3) * psi%a(4)
    kgpsi2%a(1) = ((0,-1) * k%x(2)) * psi%a(1) - km * psi%a(2)
    kgpsi2%a(2) = - kp * psi%a(1) + ((0,1) * k%x(2)) * psi%a(2)
    kgpsi2%a(3) = ((0,-1) * k%x(2)) * psi%a(3) + kp * psi%a(4)
    kgpsi2%a(4) = km * psi%a(3) + ((0,1) * k%x(2)) * psi%a(4)
    kgpsi3%a(1) = (0,1) * (k%x(1) * psi%a(1) + km * psi%a(2))
    kgpsi3%a(2) = (0,-1) * (kp * psi%a(1) + k%x(1) * psi%a(2))
    kgpsi3%a(3) = (0,1) * (k%x(1) * psi%a(3) - kp * psi%a(4))
    kgpsi3%a(4) = (0,1) * (km * psi%a(3) - k%x(1) * psi%a(4))
    kgpsi4%a(1) = -k%t * psi%a(1) - k12s * psi%a(2)
    kgpsi4%a(2) = k12 * psi%a(1) + k%t * psi%a(2)
    kgpsi4%a(3) = k%t * psi%a(3) - k12s * psi%a(4)
    kgpsi4%a(4) = k12 * psi%a(3) - k%t * psi%a(4)
    psibarpsi%t    = 2 * (psibar * kgpsi1)
    psibarpsi%x(1) = 2 * (psibar * kgpsi2)
    psibarpsi%x(2) = 2 * (psibar * kgpsi3)
    psibarpsi%x(3) = 2 * (psibar * kgpsi4)
  end function vv_ff
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function f_vvf (v, psi, k) result (kvpsi)
    type(bispinor) :: kvpsi
    type(bispinor), intent(in) :: psi
    type(vector), intent(in) :: k, v
      complex(kind=default) :: kv30, kv21, kv01, kv31, kv02, kv32
    complex(kind=default) :: ap, am, bp, bm, bps, bms
    kv30 = k%x(3) * v%t - k%t * v%x(3)
    kv21 = (0,1) * (k%x(2) * v%x(1) - k%x(1) * v%x(2))
    kv01 = k%t * v%x(1) - k%x(1) * v%t
    kv31 = k%x(3) * v%x(1) - k%x(1) * v%x(3)
    kv02 = (0,1) * (k%t * v%x(2) - k%x(2) * v%t)
    kv32 = (0,1) * (k%x(3) * v%x(2) - k%x(2) * v%x(3))
    ap  = 2 * (kv30 + kv21)
    am  = 2 * (-kv30 + kv21)
    bp  = 2 * (kv01 + kv31 + kv02 + kv32)
    bm  = 2 * (kv01 - kv31 + kv02 - kv32)
    bps = 2 * (kv01 + kv31 - kv02 - kv32)
    bms = 2 * (kv01 - kv31 - kv02 + kv32)
    kvpsi%a(1) = am * psi%a(1) + bms * psi%a(2)
    kvpsi%a(2) = bp * psi%a(1) - am * psi%a(2)
    kvpsi%a(3) = ap * psi%a(3) - bps * psi%a(4)
    kvpsi%a(4) = -bm * psi%a(3) - ap * psi%a(4)
  end function f_vvf
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function vmom_ff (g, psibar, psi, k) result (psibarpsi)
    type(vector) :: psibarpsi
    complex(kind=default), intent(in) :: g
    type(bispinor), intent(in) :: psibar, psi
    type(momentum), intent(in) :: k
    type(vector) :: vk
    vk = k
    psibarpsi = g * vv_ff (psibar, psi, vk)
  end function vmom_ff
```

⟨*Implementation of bispinor currents*⟩+≡
```
  pure function mom_ff (g, m, psibar, psi, k) result (psibarpsi)
    complex(kind=default) :: psibarpsi
    type(bispinor), intent(in) :: psibar, psi
    type(momentum), intent(in) :: k
    complex(kind=default), intent(in) :: g, m
    type(bispinor) :: kmpsi
    complex(kind=default) :: kp, km, k12, k12s
    kp = k%t + k%x(3)
    km = k%t - k%x(3)
    k12  =  k%x(1) + (0,1)*k%x(2)
    k12s =  k%x(1) - (0,1)*k%x(2)
    kmpsi%a(1) = km * psi%a(3) - k12s * psi%a(4)
    kmpsi%a(2) = kp * psi%a(4) - k12 * psi%a(3)
    kmpsi%a(3) = kp * psi%a(1) + k12s * psi%a(2)
    kmpsi%a(4) = k12 * psi%a(1) + km * psi%a(2)
    psibarpsi = g * (psibar * kmpsi) + s_ff (m, psibar, psi)
  end function mom_ff
```

⟨*Implementation of bispinor currents*⟩+≡
```
  pure function mom5_ff (g, m, psibar, psi, k) result (psibarpsi)
    complex(kind=default) :: psibarpsi
    type(bispinor), intent(in) :: psibar, psi
    type(momentum), intent(in) :: k
    complex(kind=default), intent(in) :: g, m
    type(bispinor) :: g5psi
    g5psi%a(1:2) = - psi%a(1:2)
    g5psi%a(3:4) = psi%a(3:4)
    psibarpsi = mom_ff (g, m, psibar, g5psi, k)
  end function mom5_ff
```

⟨*Implementation of bispinor currents*⟩+≡
```
  pure function moml_ff (g, m, psibar, psi, k) result (psibarpsi)
    complex(kind=default) :: psibarpsi
    type(bispinor), intent(in) :: psibar, psi
    type(momentum), intent(in) :: k
    complex(kind=default), intent(in) :: g, m
    type(bispinor) :: leftpsi
    leftpsi%a(1:2) = 2 * psi%a(1:2)
    leftpsi%a(3:4) = 0
    psibarpsi = mom_ff (g, m, psibar, leftpsi, k)
  end function moml_ff
```

⟨*Implementation of bispinor currents*⟩+≡
```
  pure function momr_ff (g, m, psibar, psi, k) result (psibarpsi)
    complex(kind=default) :: psibarpsi
    type(bispinor), intent(in) :: psibar, psi
    type(momentum), intent(in) :: k
    complex(kind=default), intent(in) :: g, m
    type(bispinor) :: rightpsi
    rightpsi%a(1:2) = 0
    rightpsi%a(3:4) = 2 * psi%a(3:4)
    psibarpsi = mom_ff (g, m, psibar, rightpsi, k)
  end function momr_ff
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function lmom_ff (g, m, psibar, psi, k) result (psibarpsi)
    complex(kind=default) :: psibarpsi
    type(bispinor), intent(in) :: psibar, psi
    type(momentum), intent(in) :: k
    complex(kind=default), intent(in) :: g, m
    psibarpsi = mom_ff  (g, m, psibar, psi, k) + &
                mom5_ff (g,-m, psibar, psi, k)
  end function lmom_ff
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function rmom_ff (g, m, psibar, psi, k) result (psibarpsi)
    complex(kind=default) :: psibarpsi
    type(bispinor), intent(in) :: psibar, psi
    type(momentum), intent(in) :: k
    complex(kind=default), intent(in) :: g, m
    psibarpsi = mom_ff  (g, m, psibar, psi, k) - &
                mom5_ff (g,-m, psibar, psi, k)
  end function rmom_ff
```

⟨*Declaration of bispinor currents*⟩+≡

```
  public :: f_vmomf, f_momf, f_mom5f, f_momlf, f_momrf, f_lmomf, f_rmomf
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function f_vmomf (g, v, psi, k) result (kvpsi)
    type(bispinor) :: kvpsi
    type(bispinor), intent(in) :: psi
    complex(kind=default), intent(in) :: g
    type(momentum), intent(in) :: k
    type(vector), intent(in) :: v
    type(vector) :: vk
    vk = k
    kvpsi = g * f_vvf (v, psi, vk)
  end function f_vmomf
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function f_momf (g, m, phi, psi, k) result (kmpsi)
    type(bispinor) :: kmpsi
    type(bispinor), intent(in) :: psi
    complex(kind=default), intent(in) :: phi, g, m
    type(momentum), intent(in) :: k
    complex(kind=default) :: kp, km, k12, k12s
    kp = k%t + k%x(3)
    km = k%t - k%x(3)
    k12  =  k%x(1) + (0,1)*k%x(2)
    k12s =  k%x(1) - (0,1)*k%x(2)
    kmpsi%a(1) = km * psi%a(3) - k12s * psi%a(4)
    kmpsi%a(2) = -k12 * psi%a(3) + kp * psi%a(4)
    kmpsi%a(3) = kp * psi%a(1) + k12s * psi%a(2)
    kmpsi%a(4) = k12 * psi%a(1) + km * psi%a(2)
    kmpsi = g * (phi * kmpsi) + f_sf (m, phi, psi)
  end function f_momf
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function f_mom5f (g, m, phi, psi, k) result (kmpsi)
    type(bispinor) :: kmpsi
    type(bispinor), intent(in) :: psi
    complex(kind=default), intent(in) :: phi, g, m
```

```
    type(momentum), intent(in) :: k
    type(bispinor) :: g5psi
    g5psi%a(1:2) = - psi%a(1:2)
    g5psi%a(3:4) =   psi%a(3:4)
    kmpsi = f_momf (g, m, phi, g5psi, k)
  end function f_mom5f
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function f_momlf (g, m, phi, psi, k) result (kmpsi)
    type(bispinor) :: kmpsi
    type(bispinor), intent(in) :: psi
    complex(kind=default), intent(in) :: phi, g, m
    type(momentum), intent(in) :: k
    type(bispinor) :: leftpsi
    leftpsi%a(1:2) = 2 * psi%a(1:2)
    leftpsi%a(3:4) = 0
    kmpsi = f_momf (g, m, phi, leftpsi, k)
  end function f_momlf
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function f_momrf (g, m, phi, psi, k) result (kmpsi)
    type(bispinor) :: kmpsi
    type(bispinor), intent(in) :: psi
    complex(kind=default), intent(in) :: phi, g, m
    type(momentum), intent(in) :: k
    type(bispinor) :: rightpsi
    rightpsi%a(1:2) = 0
    rightpsi%a(3:4) = 2 * psi%a(3:4)
    kmpsi = f_momf (g, m, phi, rightpsi, k)
  end function f_momrf
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function f_lmomf (g, m, phi, psi, k) result (kmpsi)
    type(bispinor) :: kmpsi
    type(bispinor), intent(in) :: psi
    complex(kind=default), intent(in) :: phi, g, m
    type(momentum), intent(in) :: k
    kmpsi = f_momf  (g, m, phi, psi, k) + &
            f_mom5f (g,-m, phi, psi, k)
  end function f_lmomf
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function f_rmomf (g, m, phi, psi, k) result (kmpsi)
    type(bispinor) :: kmpsi
    type(bispinor), intent(in) :: psi
    complex(kind=default), intent(in) :: phi, g, m
    type(momentum), intent(in) :: k
    kmpsi = f_momf  (g, m, phi, psi, k) - &
            f_mom5f (g,-m, phi, psi, k)
  end function f_rmomf
```

4-Couplings

⟨*Declaration of bispinor currents*⟩+≡

```
  public :: v2_ff, sv1_ff, sv2_ff, pv1_ff, pv2_ff, svl1_ff, svl2_ff, &
       svr1_ff, svr2_ff, svlr1_ff, svlr2_ff
```

⟨*Implementation of bispinor currents*⟩+≡
```
  pure function v2_ff (g, psibar, v, psi) result (v2)
    type(vector) :: v2
    complex (kind=default), intent(in) :: g
    type(bispinor), intent(in) :: psibar, psi
    type(vector), intent(in) :: v
    v2 = (-g) * vv_ff (psibar, psi, v)
  end function v2_ff
```

⟨*Implementation of bispinor currents*⟩+≡
```
  pure function sv1_ff (g, psibar, v, psi) result (phi)
    complex(kind=default) :: phi
    type(bispinor), intent(in) :: psibar, psi
    type(vector), intent(in) :: v
    complex(kind=default), intent(in) :: g
    phi = psibar * f_vf (g, v, psi)
  end function sv1_ff
```

⟨*Implementation of bispinor currents*⟩+≡
```
  pure function sv2_ff (g, psibar, phi, psi) result (v)
    type(vector) :: v
    complex(kind=default), intent(in) :: phi, g
    type(bispinor), intent(in) :: psibar, psi
    v = phi * v_ff (g, psibar, psi)
  end function sv2_ff
```

⟨*Implementation of bispinor currents*⟩+≡
```
  pure function pv1_ff (g, psibar, v, psi) result (phi)
    complex(kind=default) :: phi
    type(bispinor), intent(in) :: psibar, psi
    type(vector), intent(in) :: v
    complex(kind=default), intent(in) :: g
    phi = - (psibar * f_af (g, v, psi))
  end function pv1_ff
```

⟨*Implementation of bispinor currents*⟩+≡
```
  pure function pv2_ff (g, psibar, phi, psi) result (v)
    type(vector) :: v
    complex(kind=default), intent(in) :: phi, g
    type(bispinor), intent(in) :: psibar, psi
    v = -(phi * a_ff (g, psibar, psi))
  end function pv2_ff
```

⟨*Implementation of bispinor currents*⟩+≡
```
  pure function svl1_ff (g, psibar, v, psi) result (phi)
    complex(kind=default) :: phi
    type(bispinor), intent(in) :: psibar, psi
    type(vector), intent(in) :: v
    complex(kind=default), intent(in) :: g
    phi = psibar * f_vlf (g, v, psi)
  end function svl1_ff
```

⟨*Implementation of bispinor currents*⟩+≡
```
  pure function svl2_ff (g, psibar, phi, psi) result (v)
    type(vector) :: v
    complex(kind=default), intent(in) :: phi, g
    type(bispinor), intent(in) :: psibar, psi
```

```
    v = phi * vl_ff (g, psibar, psi)
  end function svl2_ff
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function svr1_ff (g, psibar, v, psi) result (phi)
    complex(kind=default) :: phi
    type(bispinor), intent(in) :: psibar, psi
    type(vector), intent(in) :: v
    complex(kind=default), intent(in) :: g
    phi = psibar * f_vrf (g, v, psi)
  end function svr1_ff
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function svr2_ff (g, psibar, phi, psi) result (v)
    type(vector) :: v
    complex(kind=default), intent(in) :: phi, g
    type(bispinor), intent(in) :: psibar, psi
    v = phi * vr_ff (g, psibar, psi)
  end function svr2_ff
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function svlr1_ff (gl, gr, psibar, v, psi) result (phi)
    complex(kind=default) :: phi
    type(bispinor), intent(in) :: psibar, psi
    type(vector), intent(in) :: v
    complex(kind=default), intent(in) :: gl, gr
    phi = psibar * f_vlrf (gl, gr, v, psi)
  end function svlr1_ff
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function svlr2_ff (gl, gr, psibar, phi, psi) result (v)
    type(vector) :: v
    complex(kind=default), intent(in) :: phi, gl, gr
    type(bispinor), intent(in) :: psibar, psi
    v = phi * vlr_ff (gl, gr, psibar, psi)
  end function svlr2_ff
```

⟨*Declaration of bispinor currents*⟩+≡

```
  public :: f_v2f, f_svf, f_pvf, f_svlf, f_svrf, f_svlrf
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function f_v2f (g, v1, v2, psi) result (vpsi)
    type(bispinor) :: vpsi
    complex(kind=default), intent(in) :: g
    type(bispinor), intent(in) :: psi
    type(vector), intent(in) :: v1, v2
    vpsi = g * f_vvf (v2, psi, v1)
  end function f_v2f
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function f_svf (g, phi, v, psi) result (pvpsi)
    type(bispinor) :: pvpsi
    complex(kind=default), intent(in) :: g, phi
    type(bispinor), intent(in) :: psi
    type(vector), intent(in) :: v
    pvpsi = phi * f_vf (g, v, psi)
  end function f_svf
```

⟨*Implementation of bispinor currents*⟩+≡
```
  pure function f_pvf (g, phi, v, psi) result (pvpsi)
    type(bispinor) :: pvpsi
    complex(kind=default), intent(in) :: g, phi
    type(bispinor), intent(in) :: psi
    type(vector), intent(in) :: v
    pvpsi = -(phi * f_af (g, v, psi))
  end function f_pvf
```

⟨*Implementation of bispinor currents*⟩+≡
```
  pure function f_svlf (g, phi, v, psi) result (pvpsi)
    type(bispinor) :: pvpsi
    complex(kind=default), intent(in) :: g, phi
    type(bispinor), intent(in) :: psi
    type(vector), intent(in) :: v
    pvpsi = phi * f_vlf (g, v, psi)
  end function f_svlf
```

⟨*Implementation of bispinor currents*⟩+≡
```
  pure function f_svrf (g, phi, v, psi) result (pvpsi)
    type(bispinor) :: pvpsi
    complex(kind=default), intent(in) :: g, phi
    type(bispinor), intent(in) :: psi
    type(vector), intent(in) :: v
    pvpsi = phi * f_vrf (g, v, psi)
  end function f_svrf
```

⟨*Implementation of bispinor currents*⟩+≡
```
  pure function f_svlrf (gl, gr, phi, v, psi) result (pvpsi)
    type(bispinor) :: pvpsi
    complex(kind=default), intent(in) :: gl, gr, phi
    type(bispinor), intent(in) :: psi
    type(vector), intent(in) :: v
    pvpsi = phi * f_vlrf (gl, gr, v, psi)
  end function f_svlrf
```

X.14.4 Gravitino Couplings

⟨*Declaration of bispinor currents*⟩+≡
```
  public :: pot_grf, pot_fgr, s_grf, s_fgr, p_grf, p_fgr, &
       sl_grf, sl_fgr, sr_grf, sr_fgr, slr_grf, slr_fgr
```

⟨*Declaration of bispinor currents*⟩+≡
```
  private :: fgvgr, fgvg5gr, fggvvgr, grkgf, grkggf, grkkggf, &
       fgkgr, fg5gkgr, grvgf, grg5vgf, grkgggf, fggkggr
```

⟨*Implementation of bispinor currents*⟩+≡
```
  pure function pot_grf (g, gravbar, psi) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: g
    type(vectorspinor), intent(in) :: gravbar
    type(bispinor), intent(in) :: psi
    type(vectorspinor) :: gamma_psi
    gamma_psi%psi(1)%a(1) = psi%a(3)
    gamma_psi%psi(1)%a(2) = psi%a(4)
    gamma_psi%psi(1)%a(3) = psi%a(1)
```

```
    gamma_psi%psi(1)%a(4) = psi%a(2)
    gamma_psi%psi(2)%a(1) = psi%a(4)
    gamma_psi%psi(2)%a(2) = psi%a(3)
    gamma_psi%psi(2)%a(3) = - psi%a(2)
    gamma_psi%psi(2)%a(4) = - psi%a(1)
    gamma_psi%psi(3)%a(1) = (0,-1) * psi%a(4)
    gamma_psi%psi(3)%a(2) = (0,1) * psi%a(3)
    gamma_psi%psi(3)%a(3) = (0,1) * psi%a(2)
    gamma_psi%psi(3)%a(4) = (0,-1) * psi%a(1)
    gamma_psi%psi(4)%a(1) = psi%a(3)
    gamma_psi%psi(4)%a(2) = - psi%a(4)
    gamma_psi%psi(4)%a(3) = - psi%a(1)
    gamma_psi%psi(4)%a(4) = psi%a(2)
    j = g * (gravbar * gamma_psi)
  end function pot_grf
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function pot_fgr (g, psibar, grav) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: g
    type(bispinor), intent(in) :: psibar
    type(vectorspinor), intent(in) :: grav
    type(bispinor) :: gamma_grav
    gamma_grav%a(1) = grav%psi(1)%a(3) - grav%psi(2)%a(4) + &
           ((0,1)*grav%psi(3)%a(4)) - grav%psi(4)%a(3)
    gamma_grav%a(2) = grav%psi(1)%a(4) - grav%psi(2)%a(3) - &
           ((0,1)*grav%psi(3)%a(3)) + grav%psi(4)%a(4)
    gamma_grav%a(3) = grav%psi(1)%a(1) + grav%psi(2)%a(2) - &
           ((0,1)*grav%psi(3)%a(2)) + grav%psi(4)%a(1)
    gamma_grav%a(4) = grav%psi(1)%a(2) + grav%psi(2)%a(1) + &
           ((0,1)*grav%psi(3)%a(1)) - grav%psi(4)%a(2)
    j = g * (psibar * gamma_grav)
  end function pot_fgr
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function grvgf (gravbar, psi, k) result (j)
    complex(kind=default) :: j
    complex(kind=default) :: kp, km, k12, k12s
    type(vectorspinor), intent(in) :: gravbar
    type(bispinor), intent(in) :: psi
    type(vector), intent(in) :: k
    type(vectorspinor) :: kg_psi
    kp = k%t + k%x(3)
    km = k%t - k%x(3)
    k12  =  k%x(1) + (0,1)*k%x(2)
    k12s =  k%x(1) - (0,1)*k%x(2)
    !!! Since we are taking the spinor product here, NO explicit
    !!! charge conjugation matrix is needed!
    kg_psi%psi(1)%a(1) = km * psi%a(1) - k12s * psi%a(2)
    kg_psi%psi(1)%a(2) = (-k12) * psi%a(1) + kp * psi%a(2)
    kg_psi%psi(1)%a(3) = kp * psi%a(3) + k12s * psi%a(4)
    kg_psi%psi(1)%a(4) = k12 * psi%a(3) + km * psi%a(4)
    kg_psi%psi(2)%a(1) = k12s * psi%a(1) - km * psi%a(2)
    kg_psi%psi(2)%a(2) = (-kp) * psi%a(1) + k12 * psi%a(2)
    kg_psi%psi(2)%a(3) = k12s * psi%a(3) + kp * psi%a(4)
```

```
    kg_psi%psi(2)%a(4) = km * psi%a(3) + k12 * psi%a(4)
    kg_psi%psi(3)%a(1) = (0,1) * (k12s * psi%a(1) + km * psi%a(2))
    kg_psi%psi(3)%a(2) = (0,1) * (- kp * psi%a(1) - k12 * psi%a(2))
    kg_psi%psi(3)%a(3) = (0,1) * (k12s * psi%a(3) - kp * psi%a(4))
    kg_psi%psi(3)%a(4) = (0,1) * (km * psi%a(3) - k12 * psi%a(4))
    kg_psi%psi(4)%a(1) = (-km) * psi%a(1) - k12s * psi%a(2)
    kg_psi%psi(4)%a(2) = k12 * psi%a(1) + kp * psi%a(2)
    kg_psi%psi(4)%a(3) = kp * psi%a(3) - k12s * psi%a(4)
    kg_psi%psi(4)%a(4) = k12 * psi%a(3) - km * psi%a(4)
    j = gravbar * kg_psi
  end function grvgf
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function grg5vgf (gravbar, psi, k) result (j)
    complex(kind=default) :: j
    type(vectorspinor), intent(in) :: gravbar
    type(bispinor), intent(in) :: psi
    type(vector), intent(in) :: k
    type(bispinor) :: g5_psi
    g5_psi%a(1:2) = - psi%a(1:2)
    g5_psi%a(3:4) =   psi%a(3:4)
    j = grvgf (gravbar, g5_psi, k)
  end function grg5vgf
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function s_grf (g, gravbar, psi, k) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: g
    type(vectorspinor), intent(in) :: gravbar
    type(bispinor), intent(in) :: psi
    type(momentum), intent(in) :: k
    type(vector) :: vk
    vk = k
    j = g * grvgf (gravbar, psi, vk)
  end function s_grf
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function sl_grf (gl, gravbar, psi, k) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: gl
    type(vectorspinor), intent(in) :: gravbar
    type(bispinor), intent(in) :: psi
    type(bispinor) :: psi_l
    type(momentum), intent(in) :: k
    psi_l%a(1:2) = psi%a(1:2)
    psi_l%a(3:4) = 0
    j = s_grf (gl, gravbar, psi_l, k)
  end function sl_grf
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function sr_grf (gr, gravbar, psi, k) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: gr
    type(vectorspinor), intent(in) :: gravbar
    type(bispinor), intent(in) :: psi
    type(bispinor) :: psi_r
```

```
    type(momentum), intent(in) :: k
    psi_r%a(1:2) = 0
    psi_r%a(3:4) = psi%a(3:4)
    j = s_grf (gr, gravbar, psi_r, k)
  end function sr_grf
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function slr_grf (gl, gr, gravbar, psi, k) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: gl, gr
    type(vectorspinor), intent(in) :: gravbar
    type(bispinor), intent(in) :: psi
    type(momentum), intent(in) :: k
    j = sl_grf (gl, gravbar, psi, k) + sr_grf (gr, gravbar, psi, k)
  end function slr_grf
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function fgkgr (psibar, grav, k) result (j)
    complex(kind=default) :: j
    complex(kind=default) :: kp, km, k12, k12s
    type(bispinor), intent(in) :: psibar
    type(vectorspinor), intent(in) :: grav
    type(vector), intent(in) :: k
    type(bispinor) :: gk_grav
    kp = k%t + k%x(3)
    km = k%t - k%x(3)
    k12  =  k%x(1) + (0,1)*k%x(2)
    k12s =  k%x(1) - (0,1)*k%x(2)
    !!! Since we are taking the spinor product here, NO explicit
    !!! charge conjugation matrix is needed!
    gk_grav%a(1) =  kp * grav%psi(1)%a(1) + k12s * grav%psi(1)%a(2) &
                 - k12 * grav%psi(2)%a(1) - km * grav%psi(2)%a(2) &
                 + (0,1) * k12 * grav%psi(3)%a(1)   &
                 + (0,1) * km * grav%psi(3)%a(2) &
                 - kp * grav%psi(4)%a(1) - k12s * grav%psi(4)%a(2)
    gk_grav%a(2) = k12 * grav%psi(1)%a(1) + km * grav%psi(1)%a(2) &
                 - kp * grav%psi(2)%a(1) - k12s * grav%psi(2)%a(2) &
                 - (0,1) * kp * grav%psi(3)%a(1) &
                 - (0,1) * k12s * grav%psi(3)%a(2)  &
                 + k12 * grav%psi(4)%a(1) + km * grav%psi(4)%a(2)
    gk_grav%a(3) = km * grav%psi(1)%a(3) - k12s * grav%psi(1)%a(4) &
                 - k12 * grav%psi(2)%a(3) + kp * grav%psi(2)%a(4) &
                 + (0,1) * k12 * grav%psi(3)%a(3)   &
                 - (0,1) * kp * grav%psi(3)%a(4) &
                 + km * grav%psi(4)%a(3) - k12s * grav%psi(4)%a(4)
    gk_grav%a(4) = - k12 * grav%psi(1)%a(3) + kp * grav%psi(1)%a(4) &
                 + km * grav%psi(2)%a(3) - k12s * grav%psi(2)%a(4) &
                 + (0,1) * km * grav%psi(3)%a(3) &
                 - (0,1) * k12s * grav%psi(3)%a(4)  &
                 + k12 * grav%psi(4)%a(3) - kp * grav%psi(4)%a(4)
    j = psibar * gk_grav
  end function fgkgr
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function fg5gkgr (psibar, grav, k) result (j)
    complex(kind=default) :: j
```

```
    type(bispinor), intent(in) :: psibar
    type(vectorspinor), intent(in) :: grav
    type(vector), intent(in) :: k
    type(bispinor) :: psibar_g5
    psibar_g5%a(1:2) = - psibar%a(1:2)
    psibar_g5%a(3:4) =   psibar%a(3:4)
    j = fgkgr (psibar_g5, grav, k)
  end function fg5gkgr
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function s_fgr (g, psibar, grav, k) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: g
    type(bispinor), intent(in) :: psibar
    type(vectorspinor), intent(in) :: grav
    type(momentum), intent(in) :: k
    type(vector) :: vk
    vk = k
    j = g * fgkgr (psibar, grav, vk)
  end function s_fgr
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function sl_fgr (gl, psibar, grav, k) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: gl
    type(bispinor), intent(in) :: psibar
    type(bispinor) :: psibar_l
    type(vectorspinor), intent(in) :: grav
    type(momentum), intent(in) :: k
    psibar_l%a(1:2) = psibar%a(1:2)
    psibar_l%a(3:4) = 0
    j = s_fgr (gl, psibar_l, grav, k)
  end function sl_fgr
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function sr_fgr (gr, psibar, grav, k) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: gr
    type(bispinor), intent(in) :: psibar
    type(bispinor) :: psibar_r
    type(vectorspinor), intent(in) :: grav
    type(momentum), intent(in) :: k
    psibar_r%a(1:2) = 0
    psibar_r%a(3:4) = psibar%a(3:4)
    j = s_fgr (gr, psibar_r, grav, k)
  end function sr_fgr
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function slr_fgr (gl, gr, psibar, grav, k) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: gl, gr
    type(bispinor), intent(in) :: psibar
    type(vectorspinor), intent(in) :: grav
    type(momentum), intent(in) :: k
    j = sl_fgr (gl, psibar, grav, k) + sr_fgr (gr, psibar, grav, k)
  end function slr_fgr
```

⟨*Implementation of bispinor currents*⟩+≡
```
  pure function p_grf (g, gravbar, psi, k) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: g
    type(vectorspinor), intent(in) :: gravbar
    type(bispinor), intent(in) :: psi
    type(momentum), intent(in) :: k
    type(vector) :: vk
    vk = k
    j = g * grg5vgf (gravbar, psi, vk)
  end function p_grf
```
⟨*Implementation of bispinor currents*⟩+≡
```
  pure function p_fgr (g, psibar, grav, k) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: g
    type(bispinor), intent(in) :: psibar
    type(vectorspinor), intent(in) :: grav
    type(momentum), intent(in) :: k
    type(vector) :: vk
    vk = k
    j = g * fg5gkgr (psibar, grav, vk)
  end function p_fgr
```
⟨*Declaration of bispinor currents*⟩+≡
```
  public :: f_potgr, f_sgr, f_pgr, f_vgr, f_vlrgr, f_slgr, f_srgr, f_slrgr
```
⟨*Implementation of bispinor currents*⟩+≡
```
  pure function f_potgr (g, phi, psi) result (phipsi)
    type(bispinor) :: phipsi
    complex(kind=default), intent(in) :: g
    complex(kind=default), intent(in) :: phi
    type(vectorspinor), intent(in) :: psi
    phipsi%a(1) = (g * phi) * (psi%psi(1)%a(3) - psi%psi(2)%a(4) + &
                  ((0,1)*psi%psi(3)%a(4)) - psi%psi(4)%a(3))
    phipsi%a(2) = (g * phi) * (psi%psi(1)%a(4) - psi%psi(2)%a(3) - &
                  ((0,1)*psi%psi(3)%a(3)) + psi%psi(4)%a(4))
    phipsi%a(3) = (g * phi) * (psi%psi(1)%a(1) + psi%psi(2)%a(2) - &
                  ((0,1)*psi%psi(3)%a(2)) + psi%psi(4)%a(1))
    phipsi%a(4) = (g * phi) * (psi%psi(1)%a(2) + psi%psi(2)%a(1) + &
                  ((0,1)*psi%psi(3)%a(1)) - psi%psi(4)%a(2))
  end function f_potgr
```
The slashed notation:

$$\not{k} = \begin{pmatrix} 0 & 0 & k_- & -k^* \\ 0 & 0 & -k & k_+ \\ k_+ & k^* & 0 & 0 \\ k & k_- & 0 & 0 \end{pmatrix}, \qquad \not{k}\gamma_5 = \begin{pmatrix} 0 & 0 & k_- & -k^* \\ 0 & 0 & -k & k_+ \\ -k_+ & -k^* & 0 & 0 \\ -k & -k_- & 0 & 0 \end{pmatrix} \tag{X.74}$$

with $k_\pm = k_0 \pm k_3$, $k = k_1 + \mathrm{i}k_2$, $k^* = k_1 - \mathrm{i}k_2$. But note that $\cdot^*$ is *not* complex conjugation for complex k_μ.

$$\gamma^0\not{k} = \begin{pmatrix} k_+ & k^* & 0 & 0 \\ k & k_- & 0 & 0 \\ 0 & 0 & k_- & -k^* \\ 0 & 0 & -k & k_+ \end{pmatrix}, \qquad \gamma^0\not{k}\gamma^5 = \begin{pmatrix} -k_+ & -k^* & 0 & 0 \\ -k & -k_- & 0 & 0 \\ 0 & 0 & k_- & -k^* \\ 0 & 0 & -k & k_+ \end{pmatrix} \tag{X.75a}$$

$$\gamma^1 \not{k} = \begin{pmatrix} k & k_- & 0 & 0 \\ k_+ & k^* & 0 & 0 \\ 0 & 0 & k & -k_+ \\ 0 & 0 & -k_- & k^* \end{pmatrix}, \qquad \gamma^1 \not{k} \gamma^5 = \begin{pmatrix} -k & -k_- & 0 & 0 \\ -k_+ & -k^* & 0 & 0 \\ 0 & 0 & k & -k_+ \\ 0 & 0 & -k_- & k^* \end{pmatrix} \tag{X.75b}$$

$$\gamma^2 \not{k} = \begin{pmatrix} -\mathrm{i}k & -\mathrm{i}k_- & 0 & 0 \\ \mathrm{i}k_+ & \mathrm{i}k^* & 0 & 0 \\ 0 & 0 & -\mathrm{i}k & \mathrm{i}k_+ \\ 0 & 0 & -\mathrm{i}k_- & \mathrm{i}k^* \end{pmatrix}, \qquad \gamma^2 \not{k} \gamma^5 = \begin{pmatrix} \mathrm{i}k & \mathrm{i}k_- & 0 & 0 \\ -\mathrm{i}k_+ & -\mathrm{i}k^* & 0 & 0 \\ 0 & 0 & -\mathrm{i}k & \mathrm{i}k_+ \\ 0 & 0 & -\mathrm{i}k_- & \mathrm{i}k^* \end{pmatrix} \tag{X.75c}$$

$$\gamma^3 \not{k} = \begin{pmatrix} k_+ & k^* & 0 & 0 \\ -k & -k_- & 0 & 0 \\ 0 & 0 & -k_- & k^* \\ 0 & 0 & -k & k_+ \end{pmatrix}, \qquad \gamma^3 \not{k} \gamma^5 = \begin{pmatrix} -k_+ & -k^* & 0 & 0 \\ k & k_- & 0 & 0 \\ 0 & 0 & -k_- & k^* \\ 0 & 0 & -k & k_+ \end{pmatrix} \tag{X.75d}$$

and

$$\not{k} \gamma^0 = \begin{pmatrix} k_- & -k^* & 0 & 0 \\ -k & k_+ & 0 & 0 \\ 0 & 0 & k_+ & k^* \\ 0 & 0 & k & k_- \end{pmatrix}, \qquad \not{k} \gamma^0 \gamma^5 = \begin{pmatrix} -k_- & k^* & 0 & 0 \\ k & -k_+ & 0 & 0 \\ 0 & 0 & k_+ & k^* \\ 0 & 0 & k & k_- \end{pmatrix} \tag{X.76a}$$

$$\not{k} \gamma^1 = \begin{pmatrix} k^* & -k_- & 0 & 0 \\ -k_+ & k & 0 & 0 \\ 0 & 0 & k^* & k_+ \\ 0 & 0 & k_- & k \end{pmatrix}, \qquad \not{k} \gamma^1 \gamma^5 = \begin{pmatrix} -k^* & k_- & 0 & 0 \\ k_+ & -k & 0 & 0 \\ 0 & 0 & k^* & k_+ \\ 0 & 0 & k_- & k \end{pmatrix} \tag{X.76b}$$

$$\not{k} \gamma^2 = \begin{pmatrix} \mathrm{i}k^* & \mathrm{i}k_- & 0 & 0 \\ -\mathrm{i}k_+ & -\mathrm{i}k & 0 & 0 \\ 0 & 0 & \mathrm{i}k^* & -\mathrm{i}k_+ \\ 0 & 0 & \mathrm{i}k_- & -\mathrm{i}k \end{pmatrix}, \qquad \not{k} \gamma^2 \gamma^5 = \begin{pmatrix} -\mathrm{i}k^* & -\mathrm{i}k_- & 0 & 0 \\ \mathrm{i}k_+ & \mathrm{i}k & 0 & 0 \\ 0 & 0 & \mathrm{i}k^* & -\mathrm{i}k_+ \\ 0 & 0 & \mathrm{i}k_- & -\mathrm{i}k \end{pmatrix} \tag{X.76c}$$

$$\not{k} \gamma^3 = \begin{pmatrix} -k_- & -k^* & 0 & 0 \\ k & k_+ & 0 & 0 \\ 0 & 0 & k_+ & -k^* \\ 0 & 0 & k & -k_- \end{pmatrix}, \qquad \not{k} \gamma^3 \gamma^5 = \begin{pmatrix} k_- & k^* & 0 & 0 \\ -k & -k_+ & 0 & 0 \\ 0 & 0 & k_+ & -k^* \\ 0 & 0 & k & -k_- \end{pmatrix} \tag{X.76d}$$

and

$$C\gamma^0 \not{k} = \begin{pmatrix} k & k_- & 0 & 0 \\ -k_+ & -k^* & 0 & 0 \\ 0 & 0 & k & -k_+ \\ 0 & 0 & k_- & -k^* \end{pmatrix}, \qquad C\gamma^0 \not{k} \gamma^5 = \begin{pmatrix} -k & -k_- & 0 & 0 \\ k_+ & k^* & 0 & 0 \\ 0 & 0 & k & -k_+ \\ 0 & 0 & k_- & -k^* \end{pmatrix} \tag{X.77a}$$

$$C\gamma^1 \not{k} = \begin{pmatrix} k_+ & k^* & 0 & 0 \\ -k & -k_- & 0 & 0 \\ 0 & 0 & k_- & -k^* \\ 0 & 0 & k & -k_+ \end{pmatrix}, \qquad C\gamma^1 \not{k} \gamma^5 = \begin{pmatrix} -k_+ & -k^* & 0 & 0 \\ k & k_- & 0 & 0 \\ 0 & 0 & k_- & -k^* \\ 0 & 0 & k & -k_+ \end{pmatrix} \tag{X.77b}$$

$$C\gamma^2 \not{k} = \begin{pmatrix} \mathrm{i}k_+ & \mathrm{i}k^* & 0 & 0 \\ \mathrm{i}k & \mathrm{i}k_- & 0 & 0 \\ 0 & 0 & \mathrm{i}k_- & -\mathrm{i}k^* \\ 0 & 0 & -\mathrm{i}k & \mathrm{i}k_+ \end{pmatrix}, \qquad C\gamma^2 \not{k} \gamma^5 = \begin{pmatrix} -\mathrm{i}k_+ & -\mathrm{i}k^* & 0 & 0 \\ -\mathrm{i}k & -\mathrm{i}k_- & 0 & 0 \\ 0 & 0 & \mathrm{i}k_- & -\mathrm{i}k^* \\ 0 & 0 & -\mathrm{i}k & \mathrm{i}k_+ \end{pmatrix} \tag{X.77c}$$

$$C\gamma^3 \not{k} = \begin{pmatrix} -k & -k_- & 0 & 0 \\ -k_+ & -k^* & 0 & 0 \\ 0 & 0 & k & -k_+ \\ 0 & 0 & -k_- & k^* \end{pmatrix}, \qquad C\gamma^3 \not{k} \gamma^5 = \begin{pmatrix} k & k_- & 0 & 0 \\ k_+ & k^* & 0 & 0 \\ 0 & 0 & k & -k_+ \\ 0 & 0 & -k_- & k^* \end{pmatrix} \tag{X.77d}$$

and

$$C\not{k}\gamma^0 = \begin{pmatrix} -k & k^+ & 0 & 0 \\ -k_- & k^* & 0 & 0 \\ 0 & 0 & -k & -k_- \\ 0 & 0 & k_+ & k^* \end{pmatrix}, \qquad C\not{k}\gamma^0\gamma^5 = \begin{pmatrix} k & -k_+ & 0 & 0 \\ k_- & -k^* & 0 & 0 \\ 0 & 0 & -k & -k_- \\ 0 & 0 & k_+ & k^* \end{pmatrix} \tag{X.78a}$$

$$C\not{k}\gamma^1 = \begin{pmatrix} -k_+ & k & 0 & 0 \\ -k^* & k_- & 0 & 0 \\ 0 & 0 & -k_- & -k \\ 0 & 0 & k^* & k_+ \end{pmatrix}, \qquad C\not{k}\gamma^1\gamma^5 = \begin{pmatrix} k_+ & -k & 0 & 0 \\ k^* & -k_- & 0 & 0 \\ 0 & 0 & -k_- & -k \\ 0 & 0 & k^* & k_+ \end{pmatrix} \tag{X.78b}$$

$$C\not{k}\gamma^2 = \begin{pmatrix} -\mathrm{i}k_+ & -\mathrm{i}k & 0 & 0 \\ -\mathrm{i}k^* & -\mathrm{i}k_- & 0 & 0 \\ 0 & 0 & -\mathrm{i}k_- & \mathrm{i}k \\ 0 & 0 & \mathrm{i}k^* & -\mathrm{i}k_+ \end{pmatrix}, \qquad C\not{k}\gamma^2\gamma^5 = \begin{pmatrix} \mathrm{i}k_+ & \mathrm{i}k & 0 & 0 \\ \mathrm{i}k^* & \mathrm{i}k_- & 0 & 0 \\ 0 & 0 & -\mathrm{i}k_- & \mathrm{i}k \\ 0 & 0 & \mathrm{i}k^* & -\mathrm{i}k_+ \end{pmatrix} \tag{X.78c}$$

$$C\not{k}\gamma^3 = \begin{pmatrix} k & k_+ & 0 & 0 \\ k_- & k^* & 0 & 0 \\ 0 & 0 & -k & k_- \\ 0 & 0 & k_+ & -k^* \end{pmatrix}, \qquad C\not{k}\gamma^3\gamma^5 = \begin{pmatrix} -k & -k_+ & 0 & 0 \\ -k_- & -k^* & 0 & 0 \\ 0 & 0 & -k & k_- \\ 0 & 0 & k_+ & -k^* \end{pmatrix} \tag{X.78d}$$

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function fgvgr (psi, k) result (kpsi)
    type(bispinor) :: kpsi
    complex(kind=default) :: kp, km, k12, k12s
    type(vector), intent(in) :: k
    type(vectorspinor), intent(in) :: psi
    kp = k%t + k%x(3)
    km = k%t - k%x(3)
    k12  =  k%x(1) + (0,1)*k%x(2)
    k12s =  k%x(1) - (0,1)*k%x(2)
    kpsi%a(1) = kp * psi%psi(1)%a(1) + k12s * psi%psi(1)%a(2) &
```

```
              - k12 * psi%psi(2)%a(1) - km * psi%psi(2)%a(2) &
              + (0,1) * k12 * psi%psi(3)%a(1) + (0,1) * km * psi%psi(3)%a(2) &
              - kp * psi%psi(4)%a(1) - k12s * psi%psi(4)%a(2)
    kpsi%a(2) = k12 * psi%psi(1)%a(1) + km * psi%psi(1)%a(2) &
              - kp * psi%psi(2)%a(1) - k12s * psi%psi(2)%a(2) &
              - (0,1) * kp * psi%psi(3)%a(1) - (0,1) * k12s * psi%psi(3)%a(2) &
              + k12 * psi%psi(4)%a(1) + km * psi%psi(4)%a(2)
    kpsi%a(3) = km * psi%psi(1)%a(3) - k12s * psi%psi(1)%a(4) &
              - k12 * psi%psi(2)%a(3) + kp * psi%psi(2)%a(4) &
              + (0,1) * k12 * psi%psi(3)%a(3) - (0,1) * kp * psi%psi(3)%a(4) &
              + km * psi%psi(4)%a(3) - k12s * psi%psi(4)%a(4)
    kpsi%a(4) = - k12 * psi%psi(1)%a(3) + kp * psi%psi(1)%a(4) &
              + km * psi%psi(2)%a(3) - k12s * psi%psi(2)%a(4) &
              + (0,1) * km * psi%psi(3)%a(3) - (0,1) * k12s * psi%psi(3)%a(4) &
              + k12 * psi%psi(4)%a(3) - kp * psi%psi(4)%a(4)
  end function fgvgr
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function f_sgr (g, phi, psi, k) result (phipsi)
    type(bispinor) :: phipsi
    complex(kind=default), intent(in) :: g
    complex(kind=default), intent(in) :: phi
    type(momentum), intent(in) :: k
    type(vectorspinor), intent(in) :: psi
    type(vector) :: vk
    vk = k
    phipsi = (g * phi) * fgvgr (psi, vk)
  end function f_sgr
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function f_slgr (gl, phi, psi, k) result (phipsi)
    type(bispinor) :: phipsi
    complex(kind=default), intent(in) :: gl
    complex(kind=default), intent(in) :: phi
    type(momentum), intent(in) :: k
    type(vectorspinor), intent(in) :: psi
    phipsi = f_sgr (gl, phi, psi, k)
    phipsi%a(3:4) = 0
  end function f_slgr
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function f_srgr (gr, phi, psi, k) result (phipsi)
    type(bispinor) :: phipsi
    complex(kind=default), intent(in) :: gr
    complex(kind=default), intent(in) :: phi
    type(momentum), intent(in) :: k
    type(vectorspinor), intent(in) :: psi
    phipsi = f_sgr (gr, phi, psi, k)
    phipsi%a(1:2) = 0
  end function f_srgr
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function f_slrgr (gl, gr, phi, psi, k) result (phipsi)
    type(bispinor) :: phipsi, phipsi_l, phipsi_r
    complex(kind=default), intent(in) :: gl, gr
    complex(kind=default), intent(in) :: phi
```

```
    type(momentum), intent(in) :: k
    type(vectorspinor), intent(in) :: psi
    phipsi_l = f_slgr (gl, phi, psi, k)
    phipsi_r = f_srgr (gr, phi, psi, k)
    phipsi%a(1:2) = phipsi_l%a(1:2)
    phipsi%a(3:4) = phipsi_r%a(3:4)
  end function f_slrgr
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function fgvg5gr (psi, k) result (kpsi)
    type(bispinor) :: kpsi
    type(vector), intent(in) :: k
    type(vectorspinor), intent(in) :: psi
    type(bispinor) :: kpsi_dum
    kpsi_dum = fgvgr (psi, k)
    kpsi%a(1:2) = - kpsi_dum%a(1:2)
    kpsi%a(3:4) =   kpsi_dum%a(3:4)
  end function fgvg5gr
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function f_pgr (g, phi, psi, k) result (phipsi)
    type(bispinor) :: phipsi
    complex(kind=default), intent(in) :: g
    complex(kind=default), intent(in) :: phi
    type(momentum), intent(in) :: k
    type(vectorspinor), intent(in) :: psi
    type(vector) :: vk
    vk = k
    phipsi = (g * phi) * fgvg5gr (psi, vk)
  end function f_pgr
```

The needed construction of gamma matrices involving the commutator of two gamma matrices. For the slashed terms we use as usual the abbreviations $k_\pm = k_0 \pm k_3$, $k = k_1 + \mathrm{i}k_2$, $k^* = k_1 - \mathrm{i}k_2$ and analogous expressions for the vector v^μ. We remind you that $\cdot^*$ is *not* complex conjugation for complex k_μ. Furthermore we introduce (in what follows the brackets around the vector indices have the usual meaning of antisymmetrizing with respect to the indices inside the brackets, here without a factor two in the denominator)

$$a_+ = k_+v_- + kv^* - k_-v_+ - k^*v = 2(k_{[3}v_{0]} + \mathrm{i}k_{[2}v_{1]}) \tag{X.79a}$$

$$a_- = k_-v_+ + kv^* - k_+v_- - k^*v = 2(-k_{[3}v_{0]} + \mathrm{i}k_{[2}v_{1]}) \tag{X.79b}$$

$$b_+ = 2(k_+v - kv_+) = 2(k_{[0}v_{1]} + k_{[3}v_{1]} + \mathrm{i}k_{[0}v_{2]} + \mathrm{i}k_{[3}v_{2]}) \tag{X.79c}$$

$$b_- = 2(k_-v - kv_-) = 2(k_{[0}v_{1]} - k_{[3}v_{1]} + \mathrm{i}k_{[0}v_{2]} - \mathrm{i}k_{[3}v_{2]}) \tag{X.79d}$$

$$b_{+*} = 2(k_+v^* - k^*v_+) = 2(k_{[0}v_{1]} + k_{[3}v_{1]} - \mathrm{i}k_{[0}v_{2]} - \mathrm{i}k_{[3}v_{2]}) \tag{X.79e}$$

$$b_{-*} = 2(k_-v^* - k^*v_-) = 2(k_{[0}v_{1]} - k_{[3}v_{1]} - \mathrm{i}k_{[0}v_{2]} + \mathrm{i}k_{[3}v_{2]}) \tag{X.79f}$$

Of course, one could introduce a more advanced notation, but we don't want to become confused.

$$[\not{k},\gamma^0] = \begin{pmatrix} -2k_3 & -2k^* & 0 & 0 \\ -2k & 2k_3 & 0 & 0 \\ 0 & 0 & 2k_3 & 2k^* \\ 0 & 0 & 2k & -2k_3 \end{pmatrix} \tag{X.80a}$$

$$[\not{k},\gamma^1] = \begin{pmatrix} -2\mathrm{i}k_2 & -2k_- & 0 & 0 \\ -2k_+ & 2\mathrm{i}k_2 & 0 & 0 \\ 0 & 0 & -2\mathrm{i}k_2 & 2k_+ \\ 0 & 0 & 2k_- & 2\mathrm{i}k_2 \end{pmatrix} \tag{X.80b}$$

$$[\not{k},\gamma^2] = \begin{pmatrix} 2\mathrm{i}k_1 & 2\mathrm{i}k_- & 0 & 0 \\ -2\mathrm{i}k_+ & -2\mathrm{i}k_1 & 0 & 0 \\ 0 & 0 & 2\mathrm{i}k_1 & -2\mathrm{i}k_+ \\ 0 & 0 & 2\mathrm{i}k_- & -2\mathrm{i}k_1 \end{pmatrix} \tag{X.80c}$$

$$[\not{k},\gamma^3] = \begin{pmatrix} -2k_0 & -2k^* & 0 & 0 \\ 2k & 2k_0 & 0 & 0 \\ 0 & 0 & 2k_0 & -2k^* \\ 0 & 0 & 2k & -2k_0 \end{pmatrix} \tag{X.80d}$$

$$[\not{k},\not{V}] = \begin{pmatrix} a_- & b_{-*} & 0 & 0 \\ b_+ & -a_- & 0 & 0 \\ 0 & 0 & a_+ & -b_{+*} \\ 0 & 0 & -b_- & -a_+ \end{pmatrix} \tag{X.80e}$$

$$\gamma^5\gamma^0[\not{k},\not{V}] = \begin{pmatrix} 0 & 0 & -a_+ & b_{+*} \\ 0 & 0 & b_- & a_+ \\ a_- & b_{-*} & 0 & 0 \\ b_+ & -a_- & 0 & 0 \end{pmatrix} \tag{X.80f}$$

$$\gamma^5\gamma^1[\not{k},\not{V}] = \begin{pmatrix} 0 & 0 & b_- & a_+ \\ 0 & 0 & -a_+ & b_{+*} \\ -b_+ & a_- & 0 & 0 \\ -a_- & -b_{-*} & 0 & 0 \end{pmatrix} \tag{X.80g}$$

$$\gamma^5\gamma^2[\not{k},\not{V}] = \begin{pmatrix} 0 & 0 & -\mathrm{i}b_- & -\mathrm{i}a_+ \\ 0 & 0 & -\mathrm{i}a_+ & \mathrm{i}b_{+*} \\ \mathrm{i}b_+ & -\mathrm{i}a_- & 0 & 0 \\ -\mathrm{i}a_- & -\mathrm{i}b_{-*} & 0 & 0 \end{pmatrix} \tag{X.80h}$$

$$\gamma^5\gamma^3[\not{k},\not{V}] = \begin{pmatrix} 0 & 0 & -a_+ & b_{+*} \\ 0 & 0 & -b_- & -a_+ \\ -a_- & -b_{-*} & 0 & 0 \\ b_+ & -a_- & 0 & 0 \end{pmatrix} \tag{X.80i}$$

and

$$[\not{k},\not{V}]\gamma^0\gamma^5 = \begin{pmatrix} 0 & 0 & a_- & b_{-*} \\ 0 & 0 & b_+ & -a_- \\ -a_+ & b_{+*} & 0 & 0 \\ b_- & a_+ & 0 & 0 \end{pmatrix} \tag{X.81a}$$

$$[\not{k},\not{V}]\gamma^1\gamma^5 = \begin{pmatrix} 0 & 0 & b_{-*} & a_- \\ 0 & 0 & -a_- & b_+ \\ -b_{+*} & a_+ & 0 & 0 \\ -a_+ & -b_- & 0 & 0 \end{pmatrix} \tag{X.81b}$$

$$[\not{k},\not{V}]\gamma^2\gamma^5 = \begin{pmatrix} 0 & 0 & \mathrm{i}b_{-*} & -\mathrm{i}a_- \\ 0 & 0 & -\mathrm{i}a_- & -\mathrm{i}b_+ \\ -\mathrm{i}b_{+*} & -\mathrm{i}a_+ & 0 & 0 \\ -\mathrm{i}a_+ & \mathrm{i}b_- & 0 & 0 \end{pmatrix} \tag{X.81c}$$

$$[\not{k},\not{V}]\gamma^3\gamma^5 = \begin{pmatrix} 0 & 0 & a_- & -b_{-*} \\ 0 & 0 & b_+ & a_- \\ a_+ & b_{+*} & 0 & 0 \\ -b_- & a_+ & 0 & 0 \end{pmatrix} \tag{X.81d}$$

In what follows l always means twice the value of k, e.g. $l_+ = 2k_+$. We use the abbreviation $C^{\mu\nu} \equiv C[\not{k},\gamma^\mu]\gamma^\nu\gamma^5$.

$$C^{00} = \begin{pmatrix} 0 & 0 & -l & -l_3 \\ 0 & 0 & l_3 & l^* \\ l & -l_3 & 0 & 0 \\ -l_3 & -l^* & 0 & 0 \end{pmatrix}, \qquad C^{20} = \begin{pmatrix} 0 & 0 & -\mathrm{i}l_+ & -\mathrm{i}l_1 \\ 0 & 0 & -\mathrm{i}l_1 & -\mathrm{i}l_- \\ \mathrm{i}l_- & -\mathrm{i}l_1 & 0 & 0 \\ -\mathrm{i}l_1 & \mathrm{i}l_+ & 0 & 0 \end{pmatrix} \tag{X.82a}$$

$$C^{01} = \begin{pmatrix} 0 & 0 & l_3 & -l \\ 0 & 0 & l^* & l_3 \\ l_3 & -l & 0 & 0 \\ l^* & l_3 & 0 & 0 \end{pmatrix}, \qquad C^{21} = \begin{pmatrix} 0 & 0 & -\mathrm{i}l_1 & -\mathrm{i}l_+ \\ 0 & 0 & -\mathrm{i}l_- & -\mathrm{i}l_1 \\ \mathrm{i}l_1 & -\mathrm{i}l_- & 0 & 0 \\ -\mathrm{i}l_+ & \mathrm{i}l_1 & 0 & 0 \end{pmatrix} \tag{X.82b}$$

$$C^{02} = \begin{pmatrix} 0 & 0 & \mathrm{i}l_3 & \mathrm{i}l \\ 0 & 0 & \mathrm{i}l^* & -\mathrm{i}l_3 \\ \mathrm{i}l_3 & \mathrm{i}l & 0 & 0 \\ \mathrm{i}l^* & -\mathrm{i}l_3 & 0 & 0 \end{pmatrix}, \qquad C^{22} = \begin{pmatrix} 0 & 0 & l_1 & -l_+ \\ 0 & 0 & l_- & -l_1 \\ -l_1 & -l_- & 0 & 0 \\ l_+ & l_1 & 0 & 0 \end{pmatrix} \tag{X.82c}$$

$$C^{03} = \begin{pmatrix} 0 & 0 & -l & -l_3 \\ 0 & 0 & l_3 & -l^* \\ -l & -l_3 & 0 & 0 \\ l_3 & -l^* & 0 & 0 \end{pmatrix}, \qquad C^{23} = \begin{pmatrix} 0 & 0 & -\mathrm{i}l_+ & \mathrm{i}l_1 \\ 0 & 0 & -\mathrm{i}l_1 & \mathrm{i}l_- \\ -\mathrm{i}l_- & -\mathrm{i}l_1 & 0 & 0 \\ \mathrm{i}l_1 & \mathrm{i}l_+ & 0 & 0 \end{pmatrix} \tag{X.82d}$$

$$C^{10} = \begin{pmatrix} 0 & 0 & -l_+ & \mathrm{i}l_2 \\ 0 & 0 & \mathrm{i}l_2 & l_- \\ l_- & \mathrm{i}l_2 & 0 & 0 \\ \mathrm{i}l_2 & -l_+ & 0 & 0 \end{pmatrix}, \qquad C^{30} = \begin{pmatrix} 0 & 0 & l & l_0 \\ 0 & 0 & l_0 & l^* \\ l & -l_0 & 0 & 0 \\ -l_0 & l^* & 0 & 0 \end{pmatrix} \tag{X.82e}$$

$$C^{11} = \begin{pmatrix} 0 & 0 & \mathrm{i}l_2 & -l_+ \\ 0 & 0 & l_- & \mathrm{i}l_2 \\ -\mathrm{i}l_2 & -l_- & 0 & 0 \\ l_+ & -\mathrm{i}l_2 & 0 & 0 \end{pmatrix}, \qquad C^{31} = \begin{pmatrix} 0 & 0 & l_0 & l \\ 0 & 0 & l^* & l_0 \\ l_0 & -l & 0 & 0 \\ -l^* & l_0 & 0 & 0 \end{pmatrix} \tag{X.82f}$$

$$C^{12} = \begin{pmatrix} 0 & 0 & -l_2 & \mathrm{i}l_+ \\ 0 & 0 & \mathrm{i}l_- & l_2 \\ l_2 & \mathrm{i}l_- & 0 & 0 \\ \mathrm{i}l_+ & -l_2 & 0 & 0 \end{pmatrix}, \qquad C^{32} = \begin{pmatrix} 0 & 0 & \mathrm{i}l_0 & -\mathrm{i}l \\ 0 & 0 & \mathrm{i}l^* & -\mathrm{i}l_0 \\ \mathrm{i}l_0 & \mathrm{i}l & 0 & 0 \\ -\mathrm{i}l^* & -\mathrm{i}l_0 & 0 & 0 \end{pmatrix} \tag{X.82g}$$

$$C^{13} = \begin{pmatrix} 0 & 0 & -l_+ & -\mathrm{i}l_2 \\ 0 & 0 & \mathrm{i}l_2 & -l_- \\ -l_- & \mathrm{i}l_2 & 0 & 0 \\ -\mathrm{i}l_2 & -l_+ & 0 & 0 \end{pmatrix}, \qquad C^{33} = \begin{pmatrix} 0 & 0 & l & -l_0 \\ 0 & 0 & l_0 & -l^* \\ -l & -l_0 & 0 & 0 \\ l_0 & l^* & 0 & 0 \end{pmatrix} \tag{X.82h}$$

and, with the abbreviation $\tilde{C}^{\mu\nu} \equiv C\gamma^5\gamma^\nu[\not{k}, \gamma^\mu]$ (note the reversed order of the indices!)

$$\tilde{C}^{00} = \begin{pmatrix} 0 & 0 & -l & l_3 \\ 0 & 0 & l_3 & l^* \\ l & -l_3 & 0 & 0 \\ -l_3 & -l^* & 0 & 0 \end{pmatrix}, \qquad \tilde{C}^{20} = \begin{pmatrix} 0 & 0 & -\mathrm{i}l_- & \mathrm{i}l_1 \\ 0 & 0 & \mathrm{i}l_1 & -\mathrm{i}l_+ \\ \mathrm{i}l_+ & \mathrm{i}l_1 & 0 & 0 \\ \mathrm{i}l_1 & \mathrm{i}l_- & 0 & 0 \end{pmatrix} \tag{X.83a}$$

$$\tilde{C}^{01} = \begin{pmatrix} 0 & 0 & -l_3 & -l^* \\ 0 & 0 & l & -l_3 \\ -l_3 & -l^* & 0 & 0 \\ l & -l_3 & 0 & 0 \end{pmatrix}, \qquad \tilde{C}^{21} = \begin{pmatrix} 0 & 0 & -\mathrm{i}l_1 & \mathrm{i}l_+ \\ 0 & 0 & \mathrm{i}l_- & -\mathrm{i}l_1 \\ \mathrm{i}l_1 & \mathrm{i}l_- & 0 & 0 \\ \mathrm{i}l_+ & \mathrm{i}l_1 & 0 & 0 \end{pmatrix} \tag{X.83b}$$

$$\tilde{C}^{02} = \begin{pmatrix} 0 & 0 & -\mathrm{i}l_3 & -\mathrm{i}l^* \\ 0 & 0 & -\mathrm{i}l & \mathrm{i}l_3 \\ -\mathrm{i}l_3 & -\mathrm{i}l^* & 0 & 0 \\ -\mathrm{i}l & \mathrm{i}l_3 & 0 & 0 \end{pmatrix}, \qquad \tilde{C}^{22} = \begin{pmatrix} 0 & 0 & l_1 & -l_+ \\ 0 & 0 & l_- & -l_1 \\ -l_1 & -l_- & 0 & 0 \\ l_+ & l_1 & 0 & 0 \end{pmatrix} \tag{X.83c}$$

$$\tilde{C}^{03} = \begin{pmatrix} 0 & 0 & l & -l_3 \\ 0 & 0 & l_3 & l^* \\ l & -l_3 & 0 & 0 \\ l_3 & l^* & 0 & 0 \end{pmatrix}, \qquad \tilde{C}^{23} = \begin{pmatrix} 0 & 0 & \mathrm{i}l_- & -\mathrm{i}l_1 \\ 0 & 0 & \mathrm{i}l_1 & -\mathrm{i}l_+ \\ \mathrm{i}l_+ & \mathrm{i}l_1 & 0 & 0 \\ -\mathrm{i}l_1 & -\mathrm{i}l_- & 0 & 0 \end{pmatrix} \tag{X.83d}$$

$$\tilde{C}^{10} = \begin{pmatrix} 0 & 0 & -l_- & -\mathrm{i}l_2 \\ 0 & 0 & -\mathrm{i}l_2 & l_+ \\ l_+ & -\mathrm{i}l_2 & 0 & 0 \\ -\mathrm{i}l_2 & -l_- & 0 & 0 \end{pmatrix}, \qquad \tilde{C}^{30} = \begin{pmatrix} 0 & 0 & -l & l_0 \\ 0 & 0 & l_0 & -l^* \\ -l & -l_0 & 0 & 0 \\ -l_0 & -l^* & 0 & 0 \end{pmatrix} \tag{X.83e}$$

$$\tilde{C}^{11} = \begin{pmatrix} 0 & 0 & \mathrm{i}l_2 & -l_+ \\ 0 & 0 & l_- & \mathrm{i}l_2 \\ -\mathrm{i}l_2 & -l_- & 0 & 0 \\ l_+ & -\mathrm{i}l_2 & 0 & 0 \end{pmatrix}, \qquad \tilde{C}^{31} = \begin{pmatrix} 0 & 0 & -l_0 & l^* \\ 0 & 0 & l & -l_0 \\ -l_0 & -l^* & 0 & 0 \\ -l & -l_0 & 0 & 0 \end{pmatrix} \tag{X.83f}$$

$$\tilde{C}^{12} = \begin{pmatrix} 0 & 0 & -l_2 & -\mathrm{i}l_+ \\ 0 & 0 & -\mathrm{i}l_- & l_2 \\ l_2 & -\mathrm{i}l_- & 0 & 0 \\ -\mathrm{i}l_+ & -l_2 & 0 & 0 \end{pmatrix}, \qquad \tilde{C}^{32} = \begin{pmatrix} 0 & 0 & -\mathrm{i}l_0 & \mathrm{i}l^* \\ 0 & 0 & -\mathrm{i}l & \mathrm{i}l_0 \\ -\mathrm{i}l_0 & -\mathrm{i}l^* & 0 & 0 \\ \mathrm{i}l & \mathrm{i}l_0 & 0 & 0 \end{pmatrix} \tag{X.83g}$$

$$\tilde{C}^{13} = \begin{pmatrix} 0 & 0 & l_- & \mathrm{i}l_2 \\ 0 & 0 & -\mathrm{i}l_2 & l_+ \\ l_+ & -\mathrm{i}l_2 & 0 & 0 \\ \mathrm{i}l_2 & l_- & 0 & 0 \end{pmatrix}, \qquad \tilde{C}^{33} = \begin{pmatrix} 0 & 0 & l & -l_0 \\ 0 & 0 & l_0 & -l^* \\ -l & -l_0 & 0 & 0 \\ l_0 & l^* & 0 & 0 \end{pmatrix} \tag{X.83h}$$

⟨*Implementation of bispinor currents*⟩+≡
```
  pure function fggvvgr (v, psi, k) result (psikv)
    type(bispinor) :: psikv
    type(vectorspinor), intent(in) :: psi
    type(vector), intent(in) :: v, k
    complex(kind=default) :: kv30, kv21, kv01, kv31, kv02, kv32
    complex(kind=default) :: ap, am, bp, bm, bps, bms
    kv30 = k%x(3) * v%t - k%t * v%x(3)
    kv21 = (0,1) * (k%x(2) * v%x(1) - k%x(1) * v%x(2))
    kv01 = k%t * v%x(1) - k%x(1) * v%t
    kv31 = k%x(3) * v%x(1) - k%x(1) * v%x(3)
    kv02 = (0,1) * (k%t * v%x(2) - k%x(2) * v%t)
    kv32 = (0,1) * (k%x(3) * v%x(2) - k%x(2) * v%x(3))
    ap  = 2 * (kv30 + kv21)
    am  = 2 * (-kv30 + kv21)
    bp  = 2 * (kv01 + kv31 + kv02 + kv32)
    bm  = 2 * (kv01 - kv31 + kv02 - kv32)
    bps = 2 * (kv01 + kv31 - kv02 - kv32)
    bms = 2 * (kv01 - kv31 - kv02 + kv32)
    psikv%a(1) = (-ap) * psi%psi(1)%a(3) + bps * psi%psi(1)%a(4) &
               + (-bm) * psi%psi(2)%a(3) + (-ap) * psi%psi(2)%a(4) &
               + (0,1) * (bm * psi%psi(3)%a(3) + ap * psi%psi(3)%a(4)) &
               + ap * psi%psi(4)%a(3) + (-bps) * psi%psi(4)%a(4)
    psikv%a(2) =  bm * psi%psi(1)%a(3) + ap * psi%psi(1)%a(4) &
               + ap * psi%psi(2)%a(3) + (-bps) * psi%psi(2)%a(4) &
               + (0,1) * (ap * psi%psi(3)%a(3) - bps * psi%psi(3)%a(4)) &
               + bm * psi%psi(4)%a(3) + ap * psi%psi(4)%a(4)
    psikv%a(3) =  am * psi%psi(1)%a(1) + bms * psi%psi(1)%a(2) &
               + bp * psi%psi(2)%a(1) + (-am) * psi%psi(2)%a(2) &
               + (0,-1) * (bp * psi%psi(3)%a(1) + (-am) * psi%psi(3)%a(2)) &
               + am * psi%psi(4)%a(1) + bms * psi%psi(4)%a(2)
    psikv%a(4) =  bp * psi%psi(1)%a(1) + (-am) * psi%psi(1)%a(2) &
               + am * psi%psi(2)%a(1) + bms * psi%psi(2)%a(2) &
               + (0,1) * (am * psi%psi(3)%a(1) + bms * psi%psi(3)%a(2)) &
               + (-bp) * psi%psi(4)%a(1) + am * psi%psi(4)%a(2)
  end function fggvvgr
```

⟨*Implementation of bispinor currents*⟩+≡
```
  pure function f_vgr (g, v, psi, k) result (psikkkv)
    type(bispinor) :: psikkkv
    type(vectorspinor), intent(in) :: psi
    type(vector), intent(in) :: v
    type(momentum), intent(in) :: k
    complex(kind=default), intent(in) :: g
    type(vector) :: vk
    vk = k
    psikkkv = g * (fggvvgr (v, psi, vk))
  end function f_vgr
```

⟨*Implementation of bispinor currents*⟩+≡
```
  pure function f_vlrgr (gl, gr, v, psi, k) result (psikv)
    type(bispinor) :: psikv
    type(vectorspinor), intent(in) :: psi
    type(vector), intent(in) :: v
    type(momentum), intent(in) :: k
```

```
    complex(kind=default), intent(in) :: gl, gr
    type(vector) :: vk
    vk = k
    psikv = fggvvgr (v, psi, vk)
    psikv%a(1:2) = gl * psikv%a(1:2)
    psikv%a(3:4) = gr * psikv%a(3:4)
  end function f_vlrgr
```

⟨*Declaration of bispinor currents*⟩+≡

```
  public :: gr_potf, gr_sf, gr_pf, gr_vf, gr_vlrf, gr_slf, gr_srf, gr_slrf
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function gr_potf (g, phi, psi) result (phipsi)
    type(vectorspinor) :: phipsi
    complex(kind=default), intent(in) :: g
    complex(kind=default), intent(in) :: phi
    type(bispinor), intent(in) :: psi
    phipsi%psi(1)%a(1) = (g * phi) * psi%a(3)
    phipsi%psi(1)%a(2) = (g * phi) * psi%a(4)
    phipsi%psi(1)%a(3) = (g * phi) * psi%a(1)
    phipsi%psi(1)%a(4) = (g * phi) * psi%a(2)
    phipsi%psi(2)%a(1) = (g * phi) * psi%a(4)
    phipsi%psi(2)%a(2) = (g * phi) * psi%a(3)
    phipsi%psi(2)%a(3) = ((-g) * phi) * psi%a(2)
    phipsi%psi(2)%a(4) = ((-g) * phi) * psi%a(1)
    phipsi%psi(3)%a(1) = ((0,-1) * g * phi) * psi%a(4)
    phipsi%psi(3)%a(2) = ((0,1) * g * phi) * psi%a(3)
    phipsi%psi(3)%a(3) = ((0,1) * g * phi) * psi%a(2)
    phipsi%psi(3)%a(4) = ((0,-1) * g * phi) * psi%a(1)
    phipsi%psi(4)%a(1) = (g * phi) * psi%a(3)
    phipsi%psi(4)%a(2) = ((-g) * phi) * psi%a(4)
    phipsi%psi(4)%a(3) = ((-g) * phi) * psi%a(1)
    phipsi%psi(4)%a(4) = (g * phi) * psi%a(2)
  end function gr_potf
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function grkgf (psi, k) result (kpsi)
    type(vectorspinor) :: kpsi
    complex(kind=default) :: kp, km, k12, k12s
    type(bispinor), intent(in) :: psi
    type(vector), intent(in) :: k
    kp = k%t + k%x(3)
    km = k%t - k%x(3)
    k12  =  k%x(1) + (0,1)*k%x(2)
    k12s =  k%x(1) - (0,1)*k%x(2)
    kpsi%psi(1)%a(1) = km * psi%a(1) - k12s * psi%a(2)
    kpsi%psi(1)%a(2) = (-k12) * psi%a(1) + kp * psi%a(2)
    kpsi%psi(1)%a(3) = kp * psi%a(3) + k12s * psi%a(4)
    kpsi%psi(1)%a(4) = k12 * psi%a(3) + km * psi%a(4)
    kpsi%psi(2)%a(1) = k12s * psi%a(1) - km * psi%a(2)
    kpsi%psi(2)%a(2) = (-kp) * psi%a(1) + k12 * psi%a(2)
    kpsi%psi(2)%a(3) = k12s * psi%a(3) + kp * psi%a(4)
    kpsi%psi(2)%a(4) = km * psi%a(3) + k12 * psi%a(4)
    kpsi%psi(3)%a(1) = (0,1) * (k12s * psi%a(1) + km * psi%a(2))
    kpsi%psi(3)%a(2) = (0,-1) * (kp * psi%a(1) + k12 * psi%a(2))
    kpsi%psi(3)%a(3) = (0,1) * (k12s * psi%a(3) - kp * psi%a(4))
```

```
    kpsi%psi(3)%a(4) = (0,1) * (km * psi%a(3) - k12 * psi%a(4))
    kpsi%psi(4)%a(1) = -(km * psi%a(1) + k12s * psi%a(2))
    kpsi%psi(4)%a(2) = k12 * psi%a(1) + kp * psi%a(2)
    kpsi%psi(4)%a(3) = kp * psi%a(3) - k12s * psi%a(4)
    kpsi%psi(4)%a(4) = k12 * psi%a(3) - km * psi%a(4)
  end function grkgf
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function gr_sf (g, phi, psi, k) result (phipsi)
    type(vectorspinor) :: phipsi
    complex(kind=default), intent(in) :: g
    complex(kind=default), intent(in) :: phi
    type(bispinor), intent(in) :: psi
    type(momentum), intent(in) :: k
    type(vector) :: vk
    vk = k
    phipsi = (g * phi) * grkgf (psi, vk)
  end function gr_sf
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function gr_slf (gl, phi, psi, k) result (phipsi)
    type(vectorspinor) :: phipsi
    complex(kind=default), intent(in) :: gl
    complex(kind=default), intent(in) :: phi
    type(bispinor), intent(in) :: psi
    type(bispinor) :: psi_l
    type(momentum), intent(in) :: k
    psi_l%a(1:2) = psi%a(1:2)
    psi_l%a(3:4) = 0
    phipsi = gr_sf (gl, phi, psi_l, k)
  end function gr_slf
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function gr_srf (gr, phi, psi, k) result (phipsi)
    type(vectorspinor) :: phipsi
    complex(kind=default), intent(in) :: gr
    complex(kind=default), intent(in) :: phi
    type(bispinor), intent(in) :: psi
    type(bispinor) :: psi_r
    type(momentum), intent(in) :: k
    psi_r%a(1:2) = 0
    psi_r%a(3:4) = psi%a(3:4)
    phipsi = gr_sf (gr, phi, psi_r, k)
  end function gr_srf
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function gr_slrf (gl, gr, phi, psi, k) result (phipsi)
    type(vectorspinor) :: phipsi
    complex(kind=default), intent(in) :: gl, gr
    complex(kind=default), intent(in) :: phi
    type(bispinor), intent(in) :: psi
    type(momentum), intent(in) :: k
    phipsi = gr_slf (gl, phi, psi, k) + gr_srf (gr, phi, psi, k)
  end function gr_slrf
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function grkggf (psi, k) result (kpsi)
```

```
    type(vectorspinor) :: kpsi
    complex(kind=default) :: kp, km, k12, k12s
    type(bispinor), intent(in) :: psi
    type(vector), intent(in) :: k
    kp = k%t + k%x(3)
    km = k%t - k%x(3)
    k12  =  k%x(1) + (0,1)*k%x(2)
    k12s =  k%x(1) - (0,1)*k%x(2)
    kpsi%psi(1)%a(1) = (-km) * psi%a(1) + k12s * psi%a(2)
    kpsi%psi(1)%a(2) = k12 * psi%a(1) - kp * psi%a(2)
    kpsi%psi(1)%a(3) = kp * psi%a(3) + k12s * psi%a(4)
    kpsi%psi(1)%a(4) = k12 * psi%a(3) + km * psi%a(4)
    kpsi%psi(2)%a(1) = (-k12s) * psi%a(1) + km * psi%a(2)
    kpsi%psi(2)%a(2) = kp * psi%a(1) - k12 * psi%a(2)
    kpsi%psi(2)%a(3) = k12s * psi%a(3) + kp * psi%a(4)
    kpsi%psi(2)%a(4) = km * psi%a(3) + k12 * psi%a(4)
    kpsi%psi(3)%a(1) = (0,-1) * (k12s * psi%a(1) + km * psi%a(2))
    kpsi%psi(3)%a(2) = (0,1) * (kp * psi%a(1) + k12 * psi%a(2))
    kpsi%psi(3)%a(3) = (0,1) * (k12s * psi%a(3) - kp * psi%a(4))
    kpsi%psi(3)%a(4) = (0,1) * (km * psi%a(3) - k12 * psi%a(4))
    kpsi%psi(4)%a(1) = km * psi%a(1) + k12s * psi%a(2)
    kpsi%psi(4)%a(2) = -(k12 * psi%a(1) + kp * psi%a(2))
    kpsi%psi(4)%a(3) = kp * psi%a(3) - k12s * psi%a(4)
    kpsi%psi(4)%a(4) = k12 * psi%a(3) - km * psi%a(4)
  end function grkggf
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function gr_pf (g, phi, psi, k) result (phipsi)
    type(vectorspinor) :: phipsi
    complex(kind=default), intent(in) :: g
    complex(kind=default), intent(in) :: phi
    type(bispinor), intent(in) :: psi
    type(momentum), intent(in) :: k
    type(vector) :: vk
    vk = k
    phipsi = (g * phi) * grkggf (psi, vk)
  end function gr_pf
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function grkkggf (v, psi, k) result (psikv)
    type(vectorspinor) :: psikv
    type(bispinor), intent(in) :: psi
    type(vector), intent(in) :: v, k
    complex(kind=default) :: kv30, kv21, kv01, kv31, kv02, kv32
    complex(kind=default) :: ap, am, bp, bm, bps, bms, imago
    imago = (0.0_default,1.0_default)
    kv30 = k%x(3) * v%t - k%t * v%x(3)
    kv21 = imago * (k%x(2) * v%x(1) - k%x(1) * v%x(2))
    kv01 = k%t * v%x(1) - k%x(1) * v%t
    kv31 = k%x(3) * v%x(1) - k%x(1) * v%x(3)
    kv02 = imago * (k%t * v%x(2) - k%x(2) * v%t)
    kv32 = imago * (k%x(3) * v%x(2) - k%x(2) * v%x(3))
    ap  = 2 * (kv30 + kv21)
    am  = 2 * ((-kv30) + kv21)
    bp  = 2 * (kv01 + kv31 + kv02 + kv32)
```

```
    bm  = 2 * (kv01 - kv31 + kv02 - kv32)
    bps = 2 * (kv01 + kv31 - kv02 - kv32)
    bms = 2 * (kv01 - kv31 - kv02 + kv32)
    psikv%psi(1)%a(1) = am * psi%a(3) + bms * psi%a(4)
    psikv%psi(1)%a(2) = bp * psi%a(3) + (-am) * psi%a(4)
    psikv%psi(1)%a(3) = (-ap) * psi%a(1) + bps * psi%a(2)
    psikv%psi(1)%a(4) = bm * psi%a(1) + ap * psi%a(2)
    psikv%psi(2)%a(1) = bms * psi%a(3) + am * psi%a(4)
    psikv%psi(2)%a(2) = (-am) * psi%a(3) + bp * psi%a(4)
    psikv%psi(2)%a(3) = (-bps) * psi%a(1) + ap * psi%a(2)
    psikv%psi(2)%a(4) = (-ap) * psi%a(1) + (-bm) * psi%a(2)
    psikv%psi(3)%a(1) = imago * (bms * psi%a(3) - am * psi%a(4))
    psikv%psi(3)%a(2) = (-imago) * (am * psi%a(3) + bp * psi%a(4))
    psikv%psi(3)%a(3) = (-imago) * (bps * psi%a(1) + ap * psi%a(2))
    psikv%psi(3)%a(4) = imago * ((-ap) * psi%a(1) + bm * psi%a(2))
    psikv%psi(4)%a(1) = am * psi%a(3) + (-bms) * psi%a(4)
    psikv%psi(4)%a(2) = bp * psi%a(3) + am * psi%a(4)
    psikv%psi(4)%a(3) = ap * psi%a(1) + bps * psi%a(2)
    psikv%psi(4)%a(4) = (-bm) * psi%a(1) + ap * psi%a(2)
  end function grkkggf
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function gr_vf (g, v, psi, k) result (psikv)
    type(vectorspinor) :: psikv
    type(bispinor), intent(in) :: psi
    type(vector), intent(in) :: v
    type(momentum), intent(in) :: k
    complex(kind=default), intent(in) :: g
    type(vector) :: vk
    vk = k
    psikv = g * (grkkggf (v, psi, vk))
  end function gr_vf
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function gr_vlrf (gl, gr, v, psi, k) result (psikv)
    type(vectorspinor) :: psikv
    type(bispinor), intent(in) :: psi
    type(bispinor) :: psi_l, psi_r
    type(vector), intent(in) :: v
    type(momentum), intent(in) :: k
    complex(kind=default), intent(in) :: gl, gr
    type(vector) :: vk
    vk = k
    psi_l%a(1:2) = psi%a(1:2)
    psi_l%a(3:4) = 0
    psi_r%a(1:2) = 0
    psi_r%a(3:4) = psi%a(3:4)
    psikv = gl * grkkggf (v, psi_l, vk) + gr * grkkggf (v, psi_r, vk)
  end function gr_vlrf
```

⟨*Declaration of bispinor currents*⟩+≡

```
  public :: v_grf, v_fgr
```

⟨*Declaration of bispinor currents*⟩+≡

```
  public :: vlr_grf, vlr_fgr
```

$V^\mu = \psi_\rho^T C^{\mu\rho} \psi$

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function grkgggf (psil, psir, k) result (j)
    type(vector) :: j
    type(vectorspinor), intent(in) :: psil
    type(bispinor), intent(in) :: psir
    type(vector), intent(in) :: k
    type(vectorspinor) :: c_psir0, c_psir1, c_psir2, c_psir3
    complex(kind=default) :: kp, km, k12, k12s, ik2
    kp = k%t + k%x(3)
    km = k%t - k%x(3)
    k12  =  (k%x(1) + (0,1)*k%x(2))
    k12s =  (k%x(1) - (0,1)*k%x(2))
    ik2 = (0,1) * k%x(2)
    !!! New version:
    c_psir0%psi(1)%a(1) = (-k%x(3)) * psir%a(3) + (-k12s) * psir%a(4)
    c_psir0%psi(1)%a(2) = (-k12) * psir%a(3) + k%x(3) * psir%a(4)
    c_psir0%psi(1)%a(3) = (-k%x(3)) * psir%a(1) + (-k12s) * psir%a(2)
    c_psir0%psi(1)%a(4) = (-k12) * psir%a(1) + k%x(3) * psir%a(2)
    c_psir0%psi(2)%a(1) = (-k12s) * psir%a(3) + (-k%x(3)) * psir%a(4)
    c_psir0%psi(2)%a(2) = k%x(3) * psir%a(3) + (-k12) * psir%a(4)
    c_psir0%psi(2)%a(3) = k12s * psir%a(1) + k%x(3) * psir%a(2)
    c_psir0%psi(2)%a(4) = (-k%x(3)) * psir%a(1) + k12 * psir%a(2)
    c_psir0%psi(3)%a(1) = (0,1) * ((-k12s) * psir%a(3) + k%x(3) * psir%a(4))
    c_psir0%psi(3)%a(2) = (0,1) * (k%x(3) * psir%a(3) + k12 * psir%a(4))
    c_psir0%psi(3)%a(3) = (0,1) * (k12s * psir%a(1) + (-k%x(3)) * psir%a(2))
    c_psir0%psi(3)%a(4) = (0,1) * ((-k%x(3)) * psir%a(1) + (-k12) * psir%a(2))
    c_psir0%psi(4)%a(1) = (-k%x(3)) * psir%a(3) + k12s * psir%a(4)
    c_psir0%psi(4)%a(2) = (-k12) * psir%a(3) + (-k%x(3)) * psir%a(4)
    c_psir0%psi(4)%a(3) = k%x(3) * psir%a(1) + (-k12s) * psir%a(2)
    c_psir0%psi(4)%a(4) = k12 * psir%a(1) + k%x(3) * psir%a(2)
    !!!
    c_psir1%psi(1)%a(1) = (-ik2) * psir%a(3) + (-km) * psir%a(4)
    c_psir1%psi(1)%a(2) = (-kp) * psir%a(3) + ik2 * psir%a(4)
    c_psir1%psi(1)%a(3) = ik2 * psir%a(1) + (-kp) * psir%a(2)
    c_psir1%psi(1)%a(4) = (-km) * psir%a(1) + (-ik2) * psir%a(2)
    c_psir1%psi(2)%a(1) = (-km) * psir%a(3) + (-ik2) * psir%a(4)
    c_psir1%psi(2)%a(2) = ik2 * psir%a(3) + (-kp) * psir%a(4)
    c_psir1%psi(2)%a(3) = kp * psir%a(1) + (-ik2) * psir%a(2)
    c_psir1%psi(2)%a(4) = ik2 * psir%a(1) + km * psir%a(2)
    c_psir1%psi(3)%a(1) = ((0,-1) * km) * psir%a(3) + (-k%x(2)) * psir%a(4)
    c_psir1%psi(3)%a(2) = (-k%x(2)) * psir%a(3) + ((0,1) * kp) * psir%a(4)
    c_psir1%psi(3)%a(3) = ((0,1) * kp) * psir%a(1) + (-k%x(2)) * psir%a(2)
    c_psir1%psi(3)%a(4) = (-k%x(2)) * psir%a(1) + ((0,-1) * km) * psir%a(2)
    c_psir1%psi(4)%a(1) = (-ik2) * psir%a(3) + km * psir%a(4)
    c_psir1%psi(4)%a(2) = (-kp) * psir%a(3) + (-ik2) * psir%a(4)
    c_psir1%psi(4)%a(3) = (-ik2) *  psir%a(1) + (-kp) * psir%a(2)
    c_psir1%psi(4)%a(4) = km * psir%a(1) + (-ik2) * psir%a(2)
    !!!
    c_psir2%psi(1)%a(1) = (0,1) * (k%x(1) * psir%a(3) + km * psir%a(4))
    c_psir2%psi(1)%a(2) = (0,-1) * (kp * psir%a(3) + k%x(1) * psir%a(4))
    c_psir2%psi(1)%a(3) = (0,1) * ((-k%x(1)) * psir%a(1) + kp * psir%a(2))
    c_psir2%psi(1)%a(4) = (0,1) * ((-km) * psir%a(1) + k%x(1) * psir%a(2))
    c_psir2%psi(2)%a(1) = (0,1) * (km * psir%a(3) + k%x(1) * psir%a(4))
```

```
    c_psir2%psi(2)%a(2) = (0,-1) * (k%x(1) * psir%a(3) + kp * psir%a(4))
    c_psir2%psi(2)%a(3) = (0,-1) * (kp * psir%a(1) + (-k%x(1)) * psir%a(2))
    c_psir2%psi(2)%a(4) = (0,-1) * (k%x(1) * psir%a(1) + (-km) * psir%a(2))
    c_psir2%psi(3)%a(1) = (-km) * psir%a(3) + k%x(1) * psir%a(4)
    c_psir2%psi(3)%a(2) = k%x(1) * psir%a(3) + (-kp) * psir%a(4)
    c_psir2%psi(3)%a(3) = kp * psir%a(1) + k%x(1) * psir%a(2)
    c_psir2%psi(3)%a(4) = k%x(1) * psir%a(1) + km * psir%a(2)
    c_psir2%psi(4)%a(1) = (0,1) * (k%x(1) * psir%a(3) + (-km) * psir%a(4))
    c_psir2%psi(4)%a(2) = (0,1) * ((-kp) * psir%a(3) + k%x(1) * psir%a(4))
    c_psir2%psi(4)%a(3) = (0,1) * (k%x(1) * psir%a(1) + kp * psir%a(2))
    c_psir2%psi(4)%a(4) = (0,1) * (km * psir%a(1) + k%x(1) * psir%a(2))
    !!!
    c_psir3%psi(1)%a(1) = (-k%t) * psir%a(3) - k12s * psir%a(4)
    c_psir3%psi(1)%a(2) = k12 * psir%a(3) + k%t * psir%a(4)
    c_psir3%psi(1)%a(3) = (-k%t) * psir%a(1) + k12s * psir%a(2)
    c_psir3%psi(1)%a(4) = (-k12) * psir%a(1) + k%t * psir%a(2)
    c_psir3%psi(2)%a(1) = (-k12s) * psir%a(3) + (-k%t) * psir%a(4)
    c_psir3%psi(2)%a(2) = k%t * psir%a(3) + k12 * psir%a(4)
    c_psir3%psi(2)%a(3) = (-k12s) * psir%a(1) + k%t * psir%a(2)
    c_psir3%psi(2)%a(4) = (-k%t) * psir%a(1) + k12 * psir%a(2)
    c_psir3%psi(3)%a(1) = (0,-1) * (k12s * psir%a(3) + (-k%t) * psir%a(4))
    c_psir3%psi(3)%a(2) = (0,1) * (k%t * psir%a(3) + (-k12) * psir%a(4))
    c_psir3%psi(3)%a(3) = (0,-1) * (k12s * psir%a(1) + k%t * psir%a(2))
    c_psir3%psi(3)%a(4) = (0,-1) * (k%t * psir%a(1) + k12 * psir%a(2))
    c_psir3%psi(4)%a(1) = (-k%t) * psir%a(3) + k12s * psir%a(4)
    c_psir3%psi(4)%a(2) = k12 * psir%a(3) + (-k%t) * psir%a(4)
    c_psir3%psi(4)%a(3) = k%t * psir%a(1) + k12s * psir%a(2)
    c_psir3%psi(4)%a(4) = k12 * psir%a(1) + k%t * psir%a(2)
    j%t    =   2 * (psil * c_psir0)
    j%x(1) =   2 * (psil * c_psir1)
    j%x(2) =   2 * (psil * c_psir2)
    j%x(3) =   2 * (psil * c_psir3)
  end function grkgggf
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function v_grf (g, psil, psir, k) result (j)
    type(vector) :: j
    complex(kind=default), intent(in) :: g
    type(vectorspinor), intent(in) :: psil
    type(bispinor), intent(in) :: psir
    type(momentum), intent(in) :: k
    type(vector) :: vk
    vk = k
    j = g * grkgggf (psil, psir, vk)
  end function v_grf
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function vlr_grf (gl, gr, psil, psir, k) result (j)
    type(vector) :: j
    complex(kind=default), intent(in) :: gl, gr
    type(vectorspinor), intent(in) :: psil
    type(bispinor), intent(in) :: psir
    type(bispinor) :: psir_l, psir_r
    type(momentum), intent(in) :: k
    type(vector) :: vk
```

```
    vk = k
    psir_l%a(1:2) = psir%a(1:2)
    psir_l%a(3:4) = 0
    psir_r%a(1:2) = 0
    psir_r%a(3:4) = psir%a(3:4)
    j = gl * grkgggf (psil, psir_l, vk) + gr * grkgggf (psil, psir_r, vk)
  end function vlr_grf
```

$V^\mu = \psi^T \tilde{C}^{\mu\rho}\psi_\rho$; remember the reversed index order in $\tilde{C}$.

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function fggkggr (psil, psir, k) result (j)
    type(vector) :: j
    type(vectorspinor), intent(in) :: psir
    type(bispinor), intent(in) :: psil
    type(vector), intent(in) :: k
    type(bispinor) :: c_psir0, c_psir1, c_psir2, c_psir3
    complex(kind=default) :: kp, km, k12, k12s, ik1, ik2
    kp = k%t + k%x(3)
    km = k%t - k%x(3)
    k12  =  k%x(1) + (0,1)*k%x(2)
    k12s =  k%x(1) - (0,1)*k%x(2)
    ik1 = (0,1) * k%x(1)
    ik2 = (0,1) * k%x(2)
    c_psir0%a(1) = k%x(3) * (psir%psi(1)%a(4) + psir%psi(4)%a(4) &
                 + psir%psi(2)%a(3) + (0,1) * psir%psi(3)%a(3)) &
                 - k12 * (psir%psi(1)%a(3) + psir%psi(4)%a(3)) &
                 + k12s * (psir%psi(2)%a(4) + (0,1) * psir%psi(3)%a(4))
    c_psir0%a(2) = k%x(3) * (psir%psi(1)%a(3) - psir%psi(4)%a(3) + &
                   psir%psi(2)%a(4) - (0,1) * psir%psi(3)%a(4)) + &
                   k12s * (psir%psi(1)%a(4) - psir%psi(4)%a(4)) - &
                   k12 * (psir%psi(2)%a(3) - (0,1) * psir%psi(3)%a(3))
    c_psir0%a(3) = k%x(3) * (-psir%psi(1)%a(2) + psir%psi(4)%a(2) + &
                   psir%psi(2)%a(1) + (0,1) * psir%psi(3)%a(1)) + &
                   k12 * (psir%psi(1)%a(1) - psir%psi(4)%a(1)) + &
                   k12s * (psir%psi(2)%a(2) + (0,1) * psir%psi(3)%a(2))
    c_psir0%a(4) = k%x(3) * (-psir%psi(1)%a(1) - psir%psi(4)%a(1) + &
                   psir%psi(2)%a(2) - (0,1) * psir%psi(3)%a(2)) -  &
                   k12s * (psir%psi(1)%a(2) + psir%psi(4)%a(2)) - &
                   k12 * (psir%psi(2)%a(1) - (0,1) * psir%psi(3)%a(1))
    !!!
    c_psir1%a(1) = ik2 * (-psir%psi(1)%a(4) - psir%psi(4)%a(4) - &
                   psir%psi(2)%a(3) - (0,1) * psir%psi(3)%a(3)) - &
                   km * (psir%psi(1)%a(3) + psir%psi(4)%a(3)) + &
                   kp * (psir%psi(2)%a(4) + (0,1) * psir%psi(3)%a(4))
    c_psir1%a(2) = ik2 * (-psir%psi(1)%a(3) - psir%psi(2)%a(4) + &
                   psir%psi(4)%a(3) + (0,1) * psir%psi(3)%a(4)) + &
                   kp * (psir%psi(1)%a(4) - psir%psi(4)%a(4)) - &
                   km * (psir%psi(2)%a(3) - (0,1) * psir%psi(3)%a(3))
    c_psir1%a(3) = ik2 * (-psir%psi(1)%a(2) + psir%psi(2)%a(1) + &
                   psir%psi(4)%a(2) + (0,1) * psir%psi(3)%a(1)) + &
                   kp * (psir%psi(1)%a(1) - psir%psi(4)%a(1)) + &
                   km * (psir%psi(2)%a(2) + (0,1) * psir%psi(3)%a(2))
    c_psir1%a(4) = ik2 * (-psir%psi(1)%a(1) + psir%psi(2)%a(2) - &
                   psir%psi(4)%a(1) - (0,1) * psir%psi(3)%a(2)) - &
                   km * (psir%psi(1)%a(2) + psir%psi(4)%a(2)) - &
```

```
                  kp * (psir%psi(2)%a(1) - (0,1) * psir%psi(3)%a(1))
    !!!
    c_psir2%a(1) = ik1 * (psir%psi(2)%a(3) + psir%psi(1)%a(4) &
                  + psir%psi(4)%a(4) + (0,1) * psir%psi(3)%a(3)) - &
                  ((0,1)*km) * (psir%psi(1)%a(3) + psir%psi(4)%a(3)) &
                  + kp * (psir%psi(3)%a(4) - (0,1) * psir%psi(2)%a(4))
    c_psir2%a(2) = ik1 * (psir%psi(1)%a(3) + psir%psi(2)%a(4) - &
                  psir%psi(4)%a(3) - (0,1) * psir%psi(3)%a(4)) - &
                  ((0,1)*kp) * (psir%psi(1)%a(4) - psir%psi(4)%a(4)) &
                  - km * (psir%psi(3)%a(3) + (0,1) * psir%psi(2)%a(3))
    c_psir2%a(3) = ik1 * (psir%psi(1)%a(2) - psir%psi(2)%a(1) - &
                  psir%psi(4)%a(2) - (0,1) * psir%psi(3)%a(1)) + &
                  ((0,1)*kp) * (psir%psi(1)%a(1) - psir%psi(4)%a(1)) &
                  + km * (psir%psi(3)%a(2) - (0,1) * psir%psi(2)%a(2))
    c_psir2%a(4) = ik1 * (psir%psi(1)%a(1) - psir%psi(2)%a(2) + &
                  psir%psi(4)%a(1) + (0,1) * psir%psi(3)%a(2)) + &
                  ((0,1)*km) * (psir%psi(1)%a(2) + psir%psi(4)%a(2)) - &
                  kp * (psir%psi(3)%a(1) + (0,1) * psir%psi(2)%a(1))
    !!!
    c_psir3%a(1) = k%t * (psir%psi(1)%a(4) + psir%psi(4)%a(4) + &
                  psir%psi(2)%a(3) + (0,1) * psir%psi(3)%a(3)) - &
                  k12 * (psir%psi(1)%a(3) + psir%psi(4)%a(3)) - &
                  k12s * (psir%psi(2)%a(4) + (0,1) * psir%psi(3)%a(4))
    c_psir3%a(2) = k%t * (psir%psi(1)%a(3) - psir%psi(4)%a(3) + &
                  psir%psi(2)%a(4) - (0,1) * psir%psi(3)%a(4)) - &
                  k12s * (psir%psi(1)%a(4) - psir%psi(4)%a(4)) - &
                  k12 * (psir%psi(2)%a(3) - (0,1) * psir%psi(3)%a(3))
    c_psir3%a(3) = k%t * (-psir%psi(1)%a(2) + psir%psi(2)%a(1) + &
                  psir%psi(4)%a(2) + (0,1) * psir%psi(3)%a(1)) - &
                  k12 * (psir%psi(1)%a(1) - psir%psi(4)%a(1)) + &
                  k12s * (psir%psi(2)%a(2) + (0,1) * psir%psi(3)%a(2))
    c_psir3%a(4) = k%t * (-psir%psi(1)%a(1) + psir%psi(2)%a(2) - &
                  psir%psi(4)%a(1) - (0,1) * psir%psi(3)%a(2)) - &
                  k12s * (psir%psi(1)%a(2) + psir%psi(4)%a(2)) + &
                  k12 * (psir%psi(2)%a(1) - (0,1) * psir%psi(3)%a(1))
    !!! Because we explicitly multiplied the charge conjugation matrix
    !!! we have to omit it from the spinor product and take the
    !!! ordinary product!
    j%t    =   2 * dot_product (conjg (psil%a), c_psir0%a)
    j%x(1) =   2 * dot_product (conjg (psil%a), c_psir1%a)
    j%x(2) =   2 * dot_product (conjg (psil%a), c_psir2%a)
    j%x(3) =   2 * dot_product (conjg (psil%a), c_psir3%a)
  end function fggkggr
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function v_fgr (g, psil, psir, k) result (j)
    type(vector) :: j
    complex(kind=default), intent(in) :: g
    type(vectorspinor), intent(in) :: psir
    type(bispinor), intent(in) :: psil
    type(momentum), intent(in) :: k
    type(vector) :: vk
    vk = k
    j = g * fggkggr (psil, psir, vk)
  end function v_fgr
```

⟨*Implementation of bispinor currents*⟩+≡
```
  pure function vlr_fgr (gl, gr, psil, psir, k) result (j)
    type(vector) :: j
    complex(kind=default), intent(in) :: gl, gr
    type(vectorspinor), intent(in) :: psir
    type(bispinor), intent(in) :: psil
    type(bispinor) :: psil_l
    type(bispinor) :: psil_r
    type(momentum), intent(in) :: k
    type(vector) :: vk
    vk = k
    psil_l%a(1:2) = psil%a(1:2)
    psil_l%a(3:4) = 0
    psil_r%a(1:2) = 0
    psil_r%a(3:4) = psil%a(3:4)
    j = gl * fggkggr (psil_l, psir, vk) + gr * fggkggr (psil_r, psir, vk)
  end function vlr_fgr
```

X.14.5 Gravitino 4-Couplings

⟨*Declaration of bispinor currents*⟩+≡
```
  public :: f_s2gr, f_svgr, f_slvgr, f_srvgr, f_slrvgr, f_pvgr, f_v2gr, f_v2lrgr
```

⟨*Implementation of bispinor currents*⟩+≡
```
  pure function f_s2gr (g, phi1, phi2, psi) result (phipsi)
    type(bispinor) :: phipsi
    type(vectorspinor), intent(in) :: psi
    complex(kind=default), intent(in) :: g
    complex(kind=default), intent(in) :: phi1, phi2
    phipsi = phi2 * f_potgr (g, phi1, psi)
  end function f_s2gr
```

⟨*Implementation of bispinor currents*⟩+≡
```
  pure function f_svgr (g, phi, v, grav) result (phigrav)
    type(bispinor) :: phigrav
    type(vectorspinor), intent(in) :: grav
    type(vector), intent(in) :: v
    complex(kind=default), intent(in) :: g, phi
    phigrav = (g * phi) * fgvg5gr (grav, v)
  end function f_svgr
```

⟨*Implementation of bispinor currents*⟩+≡
```
  pure function f_slvgr (gl, phi, v, grav) result (phigrav)
    type(bispinor) :: phigrav, phidum
    type(vectorspinor), intent(in) :: grav
    type(vector), intent(in) :: v
    complex(kind=default), intent(in) :: gl, phi
    phidum = (gl * phi) * fgvg5gr (grav, v)
    phigrav%a(1:2) = phidum%a(1:2)
    phigrav%a(3:4) = 0
  end function f_slvgr
```

⟨*Implementation of bispinor currents*⟩+≡
```
  pure function f_srvgr (gr, phi, v, grav) result (phigrav)
    type(bispinor) :: phigrav, phidum
    type(vectorspinor), intent(in) :: grav
```

```
    type(vector), intent(in) :: v
    complex(kind=default), intent(in) :: gr, phi
    phidum = (gr * phi) * fgvg5gr (grav, v)
    phigrav%a(1:2) = 0
    phigrav%a(3:4) = phidum%a(3:4)
  end function f_srvgr
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function f_slrvgr (gl, gr, phi, v, grav) result (phigrav)
    type(bispinor) :: phigrav
    type(vectorspinor), intent(in) :: grav
    type(vector), intent(in) :: v
    complex(kind=default), intent(in) :: gl, gr, phi
    phigrav = f_slvgr (gl, phi, v, grav) + f_srvgr (gr, phi, v, grav)
  end function f_slrvgr
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function f_pvgr (g, phi, v, grav) result (phigrav)
    type(bispinor) :: phigrav
    type(vectorspinor), intent(in) :: grav
    type(vector), intent(in) :: v
    complex(kind=default), intent(in) :: g, phi
    phigrav = (g * phi) * fgvgr (grav, v)
  end function f_pvgr
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function f_v2gr (g, v1, v2, grav) result (psi)
    type(bispinor) :: psi
    complex(kind=default), intent(in) :: g
    type(vectorspinor), intent(in) :: grav
    type(vector), intent(in) :: v1, v2
    psi = g * fggvvgr (v2, grav, v1)
  end function f_v2gr
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function f_v2lrgr (gl, gr, v1, v2, grav) result (psi)
    type(bispinor) :: psi
    complex(kind=default), intent(in) :: gl, gr
    type(vectorspinor), intent(in) :: grav
    type(vector), intent(in) :: v1, v2
    psi = fggvvgr (v2, grav, v1)
    psi%a(1:2) = gl * psi%a(1:2)
    psi%a(3:4) = gr * psi%a(3:4)
  end function f_v2lrgr
```

⟨*Declaration of bispinor currents*⟩+≡

```
  public :: gr_s2f, gr_svf, gr_pvf, gr_slvf, gr_srvf, gr_slrvf, gr_v2f, gr_v2lrf
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function gr_s2f (g, phi1, phi2, psi) result (phipsi)
    type(vectorspinor) :: phipsi
    type(bispinor), intent(in) :: psi
    complex(kind=default), intent(in) :: g
    complex(kind=default), intent(in) :: phi1, phi2
    phipsi = phi2 * gr_potf (g, phi1, psi)
  end function gr_s2f
```

⟨*Implementation of bispinor currents*⟩+≡
```
  pure function gr_svf (g, phi, v, psi) result (phipsi)
    type(vectorspinor) :: phipsi
    type(bispinor), intent(in) :: psi
    type(vector), intent(in) :: v
    complex(kind=default), intent(in) :: g, phi
    phipsi = (g * phi) * grkggf (psi, v)
  end function gr_svf
```

⟨*Implementation of bispinor currents*⟩+≡
```
  pure function gr_slvf (gl, phi, v, psi) result (phipsi)
    type(vectorspinor) :: phipsi
    type(bispinor), intent(in) :: psi
    type(bispinor) :: psi_l
    type(vector), intent(in) :: v
    complex(kind=default), intent(in) :: gl, phi
    psi_l%a(1:2) = psi%a(1:2)
    psi_l%a(3:4) = 0
    phipsi = (gl * phi) * grkggf (psi_l, v)
  end function gr_slvf
```

⟨*Implementation of bispinor currents*⟩+≡
```
  pure function gr_srvf (gr, phi, v, psi) result (phipsi)
    type(vectorspinor) :: phipsi
    type(bispinor), intent(in) :: psi
    type(bispinor) :: psi_r
    type(vector), intent(in) :: v
    complex(kind=default), intent(in) :: gr, phi
    psi_r%a(1:2) = 0
    psi_r%a(3:4) = psi%a(3:4)
    phipsi = (gr * phi) * grkggf (psi_r, v)
  end function gr_srvf
```

⟨*Implementation of bispinor currents*⟩+≡
```
  pure function gr_slrvf (gl, gr, phi, v, psi) result (phipsi)
    type(vectorspinor) :: phipsi
    type(bispinor), intent(in) :: psi
    type(vector), intent(in) :: v
    complex(kind=default), intent(in) :: gl, gr, phi
    phipsi = gr_slvf (gl, phi, v, psi) + gr_srvf (gr, phi, v, psi)
  end function gr_slrvf
```

⟨*Implementation of bispinor currents*⟩+≡
```
  pure function gr_pvf (g, phi, v, psi) result (phipsi)
    type(vectorspinor) :: phipsi
    type(bispinor), intent(in) :: psi
    type(vector), intent(in) :: v
    complex(kind=default), intent(in) :: g, phi
    phipsi = (g * phi) * grkgf (psi, v)
  end function gr_pvf
```

⟨*Implementation of bispinor currents*⟩+≡
```
  pure function gr_v2f (g, v1, v2, psi) result (vvpsi)
    type(vectorspinor) :: vvpsi
    complex(kind=default), intent(in) :: g
    type(bispinor), intent(in) :: psi
    type(vector), intent(in) :: v1, v2
```

```
    vvpsi = g * grkkggf (v2, psi, v1)
  end function gr_v2f
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function gr_v2lrf (gl, gr, v1, v2, psi) result (vvpsi)
    type(vectorspinor) :: vvpsi
    complex(kind=default), intent(in) :: gl, gr
    type(bispinor), intent(in) :: psi
    type(bispinor) :: psi_l, psi_r
    type(vector), intent(in) :: v1, v2
    psi_l%a(1:2) = psi%a(1:2)
    psi_l%a(3:4) = 0
    psi_r%a(1:2) = 0
    psi_r%a(3:4) = psi%a(3:4)
    vvpsi = gl * grkkggf (v2, psi_l, v1) + gr * grkkggf (v2, psi_r, v1)
  end function gr_v2lrf
```

⟨*Declaration of bispinor currents*⟩+≡

```
  public :: s2_grf, s2_fgr, sv1_grf, sv2_grf, sv1_fgr, sv2_fgr, &
            slv1_grf, slv2_grf, slv1_fgr, slv2_fgr, &
            srv1_grf, srv2_grf, srv1_fgr, srv2_fgr, &
            slrv1_grf, slrv2_grf, slrv1_fgr, slrv2_fgr, &
            pv1_grf, pv2_grf, pv1_fgr, pv2_fgr, v2_grf, v2_fgr, &
            v2lr_grf, v2lr_fgr
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function s2_grf (g, gravbar, phi, psi) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: g, phi
    type(vectorspinor), intent(in) :: gravbar
    type(bispinor), intent(in) :: psi
    j = phi * pot_grf (g, gravbar, psi)
  end function s2_grf
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function s2_fgr (g, psibar, phi, grav) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: g, phi
    type(bispinor), intent(in) :: psibar
    type(vectorspinor), intent(in) :: grav
    j = phi * pot_fgr (g, psibar, grav)
  end function s2_fgr
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function sv1_grf (g, gravbar, v, psi) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: g
    type(vectorspinor), intent(in) :: gravbar
    type(bispinor), intent(in) :: psi
    type(vector), intent(in) :: v
    j = g * grg5vgf (gravbar, psi, v)
  end function sv1_grf
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function slv1_grf (gl, gravbar, v, psi) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: gl
```

```
    type(vectorspinor), intent(in) :: gravbar
    type(bispinor), intent(in) :: psi
    type(bispinor) :: psi_l
    type(vector), intent(in) :: v
    psi_l%a(1:2) = psi%a(1:2)
    psi_l%a(3:4) = 0
    j = gl * grg5vgf (gravbar, psi_l, v)
  end function slv1_grf
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function srv1_grf (gr, gravbar, v, psi) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: gr
    type(vectorspinor), intent(in) :: gravbar
    type(bispinor), intent(in) :: psi
    type(bispinor) :: psi_r
    type(vector), intent(in) :: v
    psi_r%a(1:2) = 0
    psi_r%a(3:4) = psi%a(3:4)
    j = gr * grg5vgf (gravbar, psi_r, v)
  end function srv1_grf
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function slrv1_grf (gl, gr, gravbar, v, psi) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: gl, gr
    type(vectorspinor), intent(in) :: gravbar
    type(bispinor), intent(in) :: psi
    type(bispinor) :: psi_l, psi_r
    type(vector), intent(in) :: v
    psi_l%a(1:2) = psi%a(1:2)
    psi_l%a(3:4) = 0
    psi_r%a(1:2) = 0
    psi_r%a(3:4) = psi%a(3:4)
    j = gl * grg5vgf (gravbar, psi_l, v) + gr * grg5vgf (gravbar, psi_r, v)
  end function slrv1_grf
```

$$C\gamma^0\gamma^0 = -C\gamma^1\gamma^1 = -C\gamma^2\gamma^2 = C\gamma^3\gamma^3 = C = \begin{pmatrix} 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{X.84a}$$

$$C\gamma^0\gamma^1 = -C\gamma^1\gamma^0 = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{X.84b}$$

$$C\gamma^0\gamma^2 = -C\gamma^2\gamma^0 = \begin{pmatrix} -\mathrm{i} & 0 & 0 & 0 \\ 0 & -\mathrm{i} & 0 & 0 \\ 0 & 0 & -\mathrm{i} & 0 \\ 0 & 0 & 0 & -\mathrm{i} \end{pmatrix} \tag{X.84c}$$

$$C\gamma^0\gamma^3 = -C\gamma^3\gamma^0 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{X.84d}$$

$$C\gamma^1\gamma^2 = -C\gamma^2\gamma^1 = \begin{pmatrix} 0 & \mathrm{i} & 0 & 0 \\ \mathrm{i} & 0 & 0 & 0 \\ 0 & 0 & 0 & -\mathrm{i} \\ 0 & 0 & -\mathrm{i} & 0 \end{pmatrix} \tag{X.84e}$$

$$C\gamma^1\gamma^3 = -C\gamma^3\gamma^1 = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{X.84f}$$

$$C\gamma^2\gamma^3 = -C\gamma^3\gamma^2 = \begin{pmatrix} -\mathrm{i} & 0 & 0 & 0 \\ 0 & \mathrm{i} & 0 & 0 \\ 0 & 0 & \mathrm{i} & 0 \\ 0 & 0 & 0 & -\mathrm{i} \end{pmatrix} \tag{X.84g}$$

⟨*Implementation of bispinor currents*⟩+≡

```
pure function sv2_grf (g, gravbar, phi, psi) result (j)
  type(vector) :: j
  complex(kind=default), intent(in) :: g, phi
  type(vectorspinor), intent(in) :: gravbar
  type(bispinor), intent(in) :: psi
  type(vectorspinor) :: g0_psi, g1_psi, g2_psi, g3_psi
  g0_psi%psi(1)%a(1:2) = - psi%a(1:2)
  g0_psi%psi(1)%a(3:4) = psi%a(3:4)
  g0_psi%psi(2)%a(1) = psi%a(2)
  g0_psi%psi(2)%a(2) = psi%a(1)
  g0_psi%psi(2)%a(3) = psi%a(4)
  g0_psi%psi(2)%a(4) = psi%a(3)
  g0_psi%psi(3)%a(1) = (0,-1) * psi%a(2)
  g0_psi%psi(3)%a(2) = (0,1) * psi%a(1)
  g0_psi%psi(3)%a(3) = (0,-1) * psi%a(4)
  g0_psi%psi(3)%a(4) = (0,1) * psi%a(3)
  g0_psi%psi(4)%a(1) = psi%a(1)
  g0_psi%psi(4)%a(2) = - psi%a(2)
  g0_psi%psi(4)%a(3) = psi%a(3)
  g0_psi%psi(4)%a(4) = - psi%a(4)
  g1_psi%psi(1)%a(1:4) = - g0_psi%psi(2)%a(1:4)
  g1_psi%psi(2)%a(1:4) = - g0_psi%psi(1)%a(1:4)
  g1_psi%psi(3)%a(1) = (0,1) * psi%a(1)
  g1_psi%psi(3)%a(2) = (0,-1) * psi%a(2)
  g1_psi%psi(3)%a(3) = (0,-1) * psi%a(3)
  g1_psi%psi(3)%a(4) = (0,1) * psi%a(4)
  g1_psi%psi(4)%a(1) = - psi%a(2)
  g1_psi%psi(4)%a(2) = psi%a(1)
  g1_psi%psi(4)%a(3) = psi%a(4)
  g1_psi%psi(4)%a(4) = - psi%a(3)
  g2_psi%psi(1)%a(1:4) = - g0_psi%psi(3)%a(1:4)
  g2_psi%psi(2)%a(1:4) = - g1_psi%psi(3)%a(1:4)
  g2_psi%psi(3)%a(1:4) = - g0_psi%psi(1)%a(1:4)
  g2_psi%psi(4)%a(1) = (0,1) * psi%a(2)
```

```
    g2_psi%psi(4)%a(2) = (0,1) * psi%a(1)
    g2_psi%psi(4)%a(3) = (0,-1) * psi%a(4)
    g2_psi%psi(4)%a(4) = (0,-1) * psi%a(3)
    g3_psi%psi(1)%a(1:4) = - g0_psi%psi(4)%a(1:4)
    g3_psi%psi(2)%a(1:4) = - g1_psi%psi(4)%a(1:4)
    g3_psi%psi(3)%a(1:4) = - g2_psi%psi(4)%a(1:4)
    g3_psi%psi(4)%a(1:4) = - g0_psi%psi(1)%a(1:4)
    j%t    =   (g * phi) * (gravbar * g0_psi)
    j%x(1) =   (g * phi) * (gravbar * g1_psi)
    j%x(2) =   (g * phi) * (gravbar * g2_psi)
    j%x(3) =   (g * phi) * (gravbar * g3_psi)
  end function sv2_grf
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function slv2_grf (gl, gravbar, phi, psi) result (j)
    type(vector) :: j
    complex(kind=default), intent(in) :: gl, phi
    type(vectorspinor), intent(in) :: gravbar
    type(bispinor), intent(in) :: psi
    type(bispinor) :: psi_l
    psi_l%a(1:2) = psi%a(1:2)
    psi_l%a(3:4) = 0
    j = sv2_grf (gl, gravbar, phi, psi_l)
  end function slv2_grf
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function srv2_grf (gr, gravbar, phi, psi) result (j)
    type(vector) :: j
    complex(kind=default), intent(in) :: gr, phi
    type(vectorspinor), intent(in) :: gravbar
    type(bispinor), intent(in) :: psi
    type(bispinor) :: psi_r
    psi_r%a(1:2) = 0
    psi_r%a(3:4) = psi%a(3:4)
    j = sv2_grf (gr, gravbar, phi, psi_r)
  end function srv2_grf
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function slrv2_grf (gl, gr, gravbar, phi, psi) result (j)
    type(vector) :: j
    complex(kind=default), intent(in) :: gl, gr, phi
    type(vectorspinor), intent(in) :: gravbar
    type(bispinor), intent(in) :: psi
    type(bispinor) :: psi_l, psi_r
    psi_l%a(1:2) = psi%a(1:2)
    psi_l%a(3:4) = 0
    psi_r%a(1:2) = 0
    psi_r%a(3:4) = psi%a(3:4)
    j = sv2_grf (gl, gravbar, phi, psi_l) + sv2_grf (gr, gravbar, phi, psi_r)
  end function slrv2_grf
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function sv1_fgr (g, psibar, v, grav) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: g
    type(bispinor), intent(in) :: psibar
```

```
    type(vectorspinor), intent(in) :: grav
    type(vector), intent(in) :: v
    j = g * fg5gkgr (psibar, grav, v)
  end function sv1_fgr
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function slv1_fgr (gl, psibar, v, grav) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: gl
    type(bispinor), intent(in) :: psibar
    type(bispinor) :: psibar_l
    type(vectorspinor), intent(in) :: grav
    type(vector), intent(in) :: v
    psibar_l%a(1:2) = psibar%a(1:2)
    psibar_l%a(3:4) = 0
    j = gl * fg5gkgr (psibar_l, grav, v)
  end function slv1_fgr
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function srv1_fgr (gr, psibar, v, grav) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: gr
    type(bispinor), intent(in) :: psibar
    type(bispinor) :: psibar_r
    type(vectorspinor), intent(in) :: grav
    type(vector), intent(in) :: v
    psibar_r%a(1:2) = 0
    psibar_r%a(3:4) = psibar%a(3:4)
    j = gr * fg5gkgr (psibar_r, grav, v)
  end function srv1_fgr
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function slrv1_fgr (gl, gr, psibar, v, grav) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: gl, gr
    type(bispinor), intent(in) :: psibar
    type(bispinor) :: psibar_l, psibar_r
    type(vectorspinor), intent(in) :: grav
    type(vector), intent(in) :: v
    psibar_l%a(1:2) = psibar%a(1:2)
    psibar_l%a(3:4) = 0
    psibar_r%a(1:2) = 0
    psibar_r%a(3:4) = psibar%a(3:4)
    j = gl * fg5gkgr (psibar_l, grav, v)  + gr * fg5gkgr (psibar_r, grav, v)
  end function slrv1_fgr
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function sv2_fgr (g, psibar, phi, grav) result (j)
    type(vector) :: j
    complex(kind=default), intent(in) :: g, phi
    type(bispinor), intent(in) :: psibar
    type(vectorspinor), intent(in) :: grav
    type(bispinor) :: g0_grav, g1_grav, g2_grav, g3_grav
    g0_grav%a(1) = -grav%psi(1)%a(1) +  grav%psi(2)%a(2) - &
                  (0,1) * grav%psi(3)%a(2) + grav%psi(4)%a(1)
    g0_grav%a(2) = -grav%psi(1)%a(2) + grav%psi(2)%a(1) + &
```

```
                 (0,1) * grav%psi(3)%a(1) - grav%psi(4)%a(2)
    g0_grav%a(3) = grav%psi(1)%a(3) + grav%psi(2)%a(4) - &
                 (0,1) * grav%psi(3)%a(4) + grav%psi(4)%a(3)
    g0_grav%a(4) = grav%psi(1)%a(4) + grav%psi(2)%a(3) + &
                 (0,1) * grav%psi(3)%a(3) - grav%psi(4)%a(4)
    !!!
    g1_grav%a(1) = grav%psi(1)%a(2) - grav%psi(2)%a(1) + &
                 (0,1) * grav%psi(3)%a(1) - grav%psi(4)%a(2)
    g1_grav%a(2) = grav%psi(1)%a(1) - grav%psi(2)%a(2) - &
                 (0,1) * grav%psi(3)%a(2) + grav%psi(4)%a(1)
    g1_grav%a(3) = grav%psi(1)%a(4) + grav%psi(2)%a(3) - &
                 (0,1) * grav%psi(3)%a(3) + grav%psi(4)%a(4)
    g1_grav%a(4) = grav%psi(1)%a(3) + grav%psi(2)%a(4) + &
                 (0,1) * grav%psi(3)%a(4) - grav%psi(4)%a(3)
    !!!
    g2_grav%a(1) = (0,1) * (-grav%psi(1)%a(2) - grav%psi(2)%a(1) + &
                 grav%psi(4)%a(2)) - grav%psi(3)%a(1)
    g2_grav%a(2) = (0,1) * (grav%psi(1)%a(1) + grav%psi(2)%a(2) + &
                 grav%psi(4)%a(1)) - grav%psi(3)%a(2)
    g2_grav%a(3) = (0,1) * (-grav%psi(1)%a(4) + grav%psi(2)%a(3) - &
                 grav%psi(4)%a(4)) + grav%psi(3)%a(3)
    g2_grav%a(4) = (0,1) * (grav%psi(1)%a(3) - grav%psi(2)%a(4) - &
                 grav%psi(4)%a(3)) + grav%psi(3)%a(4)
    !!!
    g3_grav%a(1) = -grav%psi(1)%a(2) + grav%psi(2)%a(2) - &
                 (0,1) * grav%psi(3)%a(2) - grav%psi(4)%a(1)
    g3_grav%a(2) = grav%psi(1)%a(1) - grav%psi(2)%a(1) - &
                 (0,1) * grav%psi(3)%a(1) - grav%psi(4)%a(2)
    g3_grav%a(3) = -grav%psi(1)%a(2) - grav%psi(2)%a(4) + &
                 (0,1) * grav%psi(3)%a(4) + grav%psi(4)%a(3)
    g3_grav%a(4) = -grav%psi(1)%a(4) + grav%psi(2)%a(3) + &
                 (0,1) * grav%psi(3)%a(3) + grav%psi(4)%a(4)
    j%t    =   (g * phi) * (psibar * g0_grav)
    j%x(1) =   (g * phi) * (psibar * g1_grav)
    j%x(2) =   (g * phi) * (psibar * g2_grav)
    j%x(3) =   (g * phi) * (psibar * g3_grav)
  end function sv2_fgr
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function slv2_fgr (gl, psibar, phi, grav) result (j)
    type(vector) :: j
    complex(kind=default), intent(in) :: gl, phi
    type(bispinor), intent(in) :: psibar
    type(bispinor) :: psibar_l
    type(vectorspinor), intent(in) :: grav
    psibar_l%a(1:2) = psibar%a(1:2)
    psibar_l%a(3:4) = 0
    j = sv2_fgr (gl, psibar_l, phi, grav)
  end function slv2_fgr
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function srv2_fgr (gr, psibar, phi, grav) result (j)
    type(vector) :: j
    complex(kind=default), intent(in) :: gr, phi
    type(bispinor), intent(in) :: psibar
```

```
    type(bispinor) :: psibar_r
    type(vectorspinor), intent(in) :: grav
    psibar_r%a(1:2) = 0
    psibar_r%a(3:4) = psibar%a(3:4)
    j = sv2_fgr (gr, psibar_r, phi, grav)
  end function srv2_fgr
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function slrv2_fgr (gl, gr, psibar, phi, grav) result (j)
    type(vector) :: j
    complex(kind=default), intent(in) :: gl, gr, phi
    type(bispinor), intent(in) :: psibar
    type(bispinor) :: psibar_l, psibar_r
    type(vectorspinor), intent(in) :: grav
    psibar_l%a(1:2) = psibar%a(1:2)
    psibar_l%a(3:4) = 0
    psibar_r%a(1:2) = 0
    psibar_r%a(3:4) = psibar%a(3:4)
    j = sv2_fgr (gl, psibar_l, phi, grav) + sv2_fgr (gr, psibar_r, phi, grav)
  end function slrv2_fgr
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function pv1_grf (g, gravbar, v, psi) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: g
    type(vectorspinor), intent(in) :: gravbar
    type(bispinor), intent(in) :: psi
    type(vector), intent(in) :: v
    j = g * grvgf (gravbar, psi, v)
  end function pv1_grf
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function pv2_grf (g, gravbar, phi, psi) result (j)
    type(vector) :: j
    complex(kind=default), intent(in) :: g, phi
    type(vectorspinor), intent(in) :: gravbar
    type(bispinor), intent(in) :: psi
    type(bispinor) :: g5_psi
    g5_psi%a(1:2) = - psi%a(1:2)
    g5_psi%a(3:4) = psi%a(3:4)
    j = sv2_grf (g, gravbar, phi, g5_psi)
  end function pv2_grf
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function pv1_fgr (g, psibar, v, grav) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: g
    type(bispinor), intent(in) :: psibar
    type(vectorspinor), intent(in) :: grav
    type(vector), intent(in) :: v
    j = g * fgkgr (psibar, grav, v)
  end function pv1_fgr
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function pv2_fgr (g, psibar, phi, grav) result (j)
    type(vector) :: j
    complex(kind=default), intent(in) :: g, phi
```

```
    type(vectorspinor), intent(in) :: grav
    type(bispinor), intent(in) :: psibar
    type(bispinor) :: psibar_g5
    psibar_g5%a(1:2) = - psibar%a(1:2)
    psibar_g5%a(3:4) = psibar%a(3:4)
    j = sv2_fgr (g, psibar_g5, phi, grav)
  end function pv2_fgr
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function v2_grf (g, gravbar, v, psi) result (j)
    type(vector) :: j
    complex(kind=default), intent(in) :: g
    type(vectorspinor), intent(in) :: gravbar
    type(bispinor), intent(in) :: psi
    type(vector), intent(in) :: v
    j = -g * grkgggf (gravbar, psi, v)
  end function v2_grf
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function v2lr_grf (gl, gr, gravbar, v, psi) result (j)
    type(vector) :: j
    complex(kind=default), intent(in) :: gl, gr
    type(vectorspinor), intent(in) :: gravbar
    type(bispinor), intent(in) :: psi
    type(bispinor) :: psi_l, psi_r
    type(vector), intent(in) :: v
    psi_l%a(1:2) = psi%a(1:2)
    psi_l%a(3:4) = 0
    psi_r%a(1:2) = 0
    psi_r%a(3:4) = psi%a(3:4)
    j = -(gl * grkgggf (gravbar, psi_l, v) + gr * grkgggf (gravbar, psi_r, v))
  end function v2lr_grf
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function v2_fgr (g, psibar, v, grav) result (j)
    type(vector) :: j
    complex(kind=default), intent(in) :: g
    type(vectorspinor), intent(in) :: grav
    type(bispinor), intent(in) :: psibar
    type(vector), intent(in) :: v
    j = -g * fggkggr (psibar, grav, v)
  end function v2_fgr
```

⟨*Implementation of bispinor currents*⟩+≡

```
  pure function v2lr_fgr (gl, gr, psibar, v, grav) result (j)
    type(vector) :: j
    complex(kind=default), intent(in) :: gl, gr
    type(vectorspinor), intent(in) :: grav
    type(bispinor), intent(in) :: psibar
    type(bispinor) :: psibar_l, psibar_r
    type(vector), intent(in) :: v
    psibar_l%a(1:2) = psibar%a(1:2)
    psibar_l%a(3:4) = 0
    psibar_r%a(1:2) = 0
    psibar_r%a(3:4) = psibar%a(3:4)
    j = -(gl * fggkggr (psibar_l, grav, v) + gr * fggkggr (psibar_r, grav, v))
  end function v2lr_fgr
```

X.14.6 On Shell Wave Functions

⟨*Declaration of bispinor on shell wave functions*⟩≡
```
public :: u, v, ghost
```

$$\chi_+(\vec p) = \frac{1}{\sqrt{2|\vec p|(|\vec p| + p_3)}} \begin{pmatrix} |\vec p| + p_3 \\ p_1 + \mathrm{i} p_2 \end{pmatrix} \tag{X.85a}$$

$$\chi_-(\vec p) = \frac{1}{\sqrt{2|\vec p|(|\vec p| + p_3)}} \begin{pmatrix} -p_1 + \mathrm{i} p_2 \\ |\vec p| + p_3 \end{pmatrix} \tag{X.85b}$$

$$u_\pm(p) = \begin{pmatrix} \sqrt{p_0 \mp |\vec p|} \cdot \chi_\pm(\vec p) \\ \sqrt{p_0 \pm |\vec p|} \cdot \chi_\pm(\vec p) \end{pmatrix} \tag{X.86}$$

⟨*Implementation of bispinor on shell wave functions*⟩≡
```
pure function u (mass, p, s) result (psi)
  type(bispinor) :: psi
  real(kind=default), intent(in) :: mass
  type(momentum), intent(in) :: p
  integer, intent(in) :: s
  complex(kind=default), dimension(2) :: chip, chim
  real(kind=default) :: pabs, norm, delta, m
  m = abs(mass)
  pabs = sqrt (dot_product (p%x, p%x))
  if (m < epsilon (m) * pabs) then
      delta = 0
  else
      delta = sqrt (max (p%t - pabs, 0._default))
  end if
  if (pabs + p%x(3) <= 1000 * epsilon (pabs) * pabs) then
     chip = (/ cmplx ( 0.0, 0.0, kind=default), &
               cmplx ( 1.0, 0.0, kind=default) /)
     chim = (/ cmplx (-1.0, 0.0, kind=default), &
               cmplx ( 0.0, 0.0, kind=default) /)
  else
     norm = 1 / sqrt (2*pabs*(pabs + p%x(3)))
     chip = norm * (/ cmplx (pabs + p%x(3), kind=default), &
                      cmplx (p%x(1), p%x(2), kind=default) /)
     chim = norm * (/ cmplx (-p%x(1), p%x(2), kind=default), &
                      cmplx (pabs + p%x(3), kind=default) /)
  end if
  if (s > 0) then
     psi%a(1:2) = delta * chip
     psi%a(3:4) = sqrt (p%t + pabs) * chip
  else
     psi%a(1:2) = sqrt (p%t + pabs) * chim
     psi%a(3:4) = delta * chim
  end if
  pabs = m ! make the compiler happy and use m
  if (mass < 0) then
     psi%a(1:2) = - imago * psi%a(1:2)
     psi%a(3:4) = + imago * psi%a(3:4)
  end if
end function u
```

$$v_{\pm}(p) = \begin{pmatrix} \mp\sqrt{p_0 \pm |\vec{p}|} \cdot \chi_{\mp}(\vec{p}) \\ \pm\sqrt{p_0 \mp |\vec{p}|} \cdot \chi_{\mp}(\vec{p}) \end{pmatrix} \tag{X.87}$$

⟨*Implementation of bispinor on shell wave functions*⟩+≡

```
  pure function v (mass, p, s) result (psi)
    type(bispinor) :: psi
    real(kind=default), intent(in) :: mass
    type(momentum), intent(in) :: p
    integer, intent(in) :: s
    complex(kind=default), dimension(2) :: chip, chim
    real(kind=default) :: pabs, norm, delta, m
    pabs = sqrt (dot_product (p%x, p%x))
    m = abs(mass)
    if (m < epsilon (m) * pabs) then
        delta = 0
    else
        delta = sqrt (max (p%t - pabs, 0._default))
    end if
    if (pabs + p%x(3) <= 1000 * epsilon (pabs) * pabs) then
       chip = (/ cmplx ( 0.0, 0.0, kind=default), &
                 cmplx ( 1.0, 0.0, kind=default) /)
       chim = (/ cmplx (-1.0, 0.0, kind=default), &
                 cmplx ( 0.0, 0.0, kind=default) /)
    else
       norm = 1 / sqrt (2*pabs*(pabs + p%x(3)))
       chip = norm * (/ cmplx (pabs + p%x(3), kind=default), &
                        cmplx (p%x(1), p%x(2), kind=default) /)
       chim = norm * (/ cmplx (-p%x(1), p%x(2), kind=default), &
                        cmplx (pabs + p%x(3), kind=default) /)
    end if
    if (s > 0) then
       psi%a(1:2) = - sqrt (p%t + pabs) * chim
       psi%a(3:4) = delta * chim
    else
       psi%a(1:2) = delta * chip
       psi%a(3:4) = - sqrt (p%t + pabs) * chip
    end if
    pabs = m ! make the compiler happy and use m
    if (mass < 0) then
       psi%a(1:2) = - imago * psi%a(1:2)
       psi%a(3:4) = + imago * psi%a(3:4)
    end if
  end function v
```

⟨*Implementation of bispinor on shell wave functions*⟩+≡

```
  pure function ghost (m, p, s) result (psi)
      type(bispinor) :: psi
      real(kind=default), intent(in) :: m
      type(momentum), intent(in) :: p
      integer, intent(in) :: s
      psi%a(:) = 0
      select case (s)
      case (1)
         psi%a(1)   = 1
```

```
      psi%a(2:4) = 0
   case (2)
      psi%a(1)   = 0
      psi%a(2)   = 1
      psi%a(3:4) = 0
   case (3)
      psi%a(1:2) = 0
      psi%a(3)   = 1
      psi%a(4)   = 0
   case (4)
      psi%a(1:3) = 0
      psi%a(4)   = 1
   case (5)
      psi%a(1) =    1.4
      psi%a(2) = -  2.3
      psi%a(3) = - 71.5
      psi%a(4) =    0.1
   end select
end function ghost
```

X.14.7 Off Shell Wave Functions

This is the same as for the Dirac fermions except that the expressions for [ubar] and [vbar] are missing.

⟨*Declaration of bispinor off shell wave functions*⟩≡

```
public :: brs_u, brs_v
```

In momentum space we have:

$$brsu(p) = (-i)(\not{p} - m)u(p) \tag{X.88}$$

⟨*Implementation of bispinor off shell wave functions*⟩≡

```
pure function brs_u (m, p, s) result (dpsi)
    type(bispinor) :: dpsi, psi
    real(kind=default), intent(in) :: m
    type(momentum), intent(in) :: p
    integer, intent(in) :: s
    type (vector)::vp
    complex(kind=default), parameter :: one = (1, 0)
    vp=p
    psi=u(m,p,s)
    dpsi=cmplx(0.0,-1.0)*(f_vf(one,vp,psi)-m*psi)
end function brs_u
```

$$brsv(p) = i(\not{p} + m)v(p) \tag{X.89}$$

⟨*Implementation of bispinor off shell wave functions*⟩+≡

```
pure function brs_v (m, p, s) result (dpsi)
    type(bispinor) :: dpsi, psi
    real(kind=default), intent(in) :: m
    type(momentum), intent(in) :: p
    integer, intent(in) ::   s
    type (vector)::vp
    complex(kind=default), parameter :: one = (1, 0)
```

```
    vp=p
    psi=v(m,p,s)
    dpsi=cmplx(0.0,1.0)*(f_vf(one,vp,psi)+m*psi)
  end function brs_v
```

X.14.8 Propagators

⟨*Declaration of bispinor propagators*⟩≡
```
  public :: pr_psi, pr_grav
  public :: pj_psi, pg_psi
```

$$\frac{\mathrm{i}(-\not{p}+m)}{p^2-m^2+\mathrm{i}m\Gamma}\psi \tag{X.90}$$

NB: the sign of the momentum comes about because all momenta are treated as *outgoing* and the particle charge flow is therefore opposite to the momentum.

⟨*Implementation of bispinor propagators*⟩≡
```
  pure function pr_psi (p, m, w, psi) result (ppsi)
    type(bispinor) :: ppsi
    type(momentum), intent(in) :: p
    real(kind=default), intent(in) :: m, w
    type(bispinor), intent(in) :: psi
    type(vector) :: vp
    complex(kind=default), parameter :: one = (1, 0)
    vp = p
    ppsi = (1 / cmplx (p*p - m**2, m*w, kind=default)) &
         * (- f_vf (one, vp, psi) + m * psi)
  end function pr_psi
```

$$\sqrt{\frac{\pi}{M\Gamma}}(-\not{p}+m)\psi \tag{X.91}$$

⟨*Implementation of bispinor propagators*⟩+≡
```
  pure function pj_psi (p, m, w, psi) result (ppsi)
    type(bispinor) :: ppsi
    type(momentum), intent(in) :: p
    real(kind=default), intent(in) :: m, w
    type(bispinor), intent(in) :: psi
    type(vector) :: vp
    complex(kind=default), parameter :: one = (1, 0)
    vp = p
    ppsi = (0, -1) * sqrt (PI / m / w) * (- f_vf (one, vp, psi) + m * psi)
  end function pj_psi
```

⟨*Implementation of bispinor propagators*⟩+≡
```
  pure function pg_psi (p, m, w, psi) result (ppsi)
    type(bispinor) :: ppsi
    type(momentum), intent(in) :: p
    real(kind=default), intent(in) :: m, w
    type(bispinor), intent(in) :: psi
    type(vector) :: vp
    complex(kind=default), parameter :: one = (1, 0)
    vp = p
    ppsi = gauss (p*p, m, w) * (- f_vf (one, vp, psi) + m * psi)
  end function pg_psi
```

$$\frac{\mathrm{i}\Big\{(-\not p+m)\left(-\eta_{\mu\nu}+\frac{p_\mu p_\nu}{m^2}\right)+\frac{1}{3}\left(\gamma_\mu-\frac{p_\mu}{m}\right)(\not p+m)\left(\gamma_\nu-\frac{p_\nu}{m}\right)\Big\}}{p^2-m^2+\mathrm{i}m\Gamma}\,\psi^\nu \quad (X.92)$$

⟨*Implementation of bispinor propagators*⟩+≡

```
pure function pr_grav (p, m, w, grav) result (propgrav)
  type(vectorspinor) :: propgrav
  type(momentum), intent(in) :: p
  real(kind=default), intent(in) :: m, w
  type(vectorspinor), intent(in) :: grav
  type(vector) :: vp
  type(bispinor) :: pgrav, ggrav, ggrav1, ggrav2, ppgrav
  type(vectorspinor) :: etagrav_dum, etagrav, pppgrav, &
                        gg_grav_dum, gg_grav
  complex(kind=default), parameter :: one = (1, 0)
  real(kind=default) :: minv
  integer :: i
  vp = p
  minv = 1/m
  pgrav = p%t    * grav%psi(1) - p%x(1) * grav%psi(2) - &
          p%x(2) * grav%psi(3) - p%x(3) * grav%psi(4)
  ggrav%a(1) = grav%psi(1)%a(3) - grav%psi(2)%a(4) + (0,1) * &
               grav%psi(3)%a(4) - grav%psi(4)%a(3)
  ggrav%a(2) = grav%psi(1)%a(4) - grav%psi(2)%a(3) - (0,1) * &
               grav%psi(3)%a(3) + grav%psi(4)%a(4)
  ggrav%a(3) = grav%psi(1)%a(1) + grav%psi(2)%a(2) - (0,1) * &
               grav%psi(3)%a(2) + grav%psi(4)%a(1)
  ggrav%a(4) = grav%psi(1)%a(2) + grav%psi(2)%a(1) + (0,1) * &
               grav%psi(3)%a(1) - grav%psi(4)%a(2)
  ggrav1 = ggrav - minv * pgrav
  ggrav2 = f_vf (one, vp, ggrav1) + m * ggrav - pgrav
  ppgrav = (-minv**2) * f_vf (one, vp, pgrav) + minv * pgrav
  do i = 1, 4
  etagrav_dum%psi(i) = f_vf (one, vp, grav%psi(i))
  end do
  etagrav = etagrav_dum - m * grav
  pppgrav%psi(1) = p%t    * ppgrav
  pppgrav%psi(2) = p%x(1) * ppgrav
  pppgrav%psi(3) = p%x(2) * ppgrav
  pppgrav%psi(4) = p%x(3) * ppgrav
  gg_grav_dum%psi(1) = p%t    * ggrav2
  gg_grav_dum%psi(2) = p%x(1) * ggrav2
  gg_grav_dum%psi(3) = p%x(2) * ggrav2
  gg_grav_dum%psi(4) = p%x(3) * ggrav2
  gg_grav = gr_potf (one, one, ggrav2) - minv * gg_grav_dum
  propgrav = (1 / cmplx (p*p - m**2, m*w, kind=default)) * &
       (etagrav + pppgrav + (1/3.0_default) * gg_grav)
end function pr_grav
```

X.15 Polarization vectorspinors

Here we construct the wavefunctions for (massive) gravitinos out of the wavefunctions of (massive) vectorbosons and (massive) Majorana fermions.

$$\psi^{\mu}_{(u;3/2)}(k) = \epsilon^{\mu}_{+}(k) \cdot u(k,+) \tag{X.93a}$$

$$\psi^{\mu}_{(u;1/2)}(k) = \sqrt{\frac{1}{3}}\,\epsilon^{\mu}_{+}(k) \cdot u(k,-) + \sqrt{\frac{2}{3}}\,\epsilon^{\mu}_{0}(k) \cdot u(k,+) \tag{X.93b}$$

$$\psi^{\mu}_{(u;-1/2)}(k) = \sqrt{\frac{2}{3}}\,\epsilon^{\mu}_{0}(k) \cdot u(k,-) + \sqrt{\frac{1}{3}}\,\epsilon^{\mu}_{-}(k) \cdot u(k,+) \tag{X.93c}$$

$$\psi^{\mu}_{(u;-3/2)}(k) = \epsilon^{\mu}_{-}(k) \cdot u(k,-) \tag{X.93d}$$

and in the same manner for $\psi^{\mu}_{(v;s)}$ with u replaced by v and with the conjugated polarization vectors. These gravitino wavefunctions obey the Dirac equation, they are transverse and they fulfill the irreducibility condition

$$\gamma_{\mu}\psi^{\mu}_{(u/v;s)} = 0. \tag{X.94}$$

```
⟨omega_vspinor_polarizations.f90⟩≡
  ⟨Copyleft⟩
  module omega_vspinor_polarizations
    use kinds
    use constants
    use omega_vectors
    use omega_bispinors
    use omega_bispinor_couplings
    use omega_vectorspinors
    implicit none
    ⟨Declaration of polarization vectorspinors⟩
    integer, parameter, public :: omega_vspinor_pols_2010_01_A = 0
  contains
    ⟨Implementation of polarization vectorspinors⟩
  end module omega_vspinor_polarizations
```

```
⟨Declaration of polarization vectorspinors⟩≡
  public :: ueps, veps
  private :: eps
  private :: outer_product
```

Here we implement the polarization vectors for vectorbosons with trigonometric functions, without the rotating of components done in HELAS [5]. These are only used for generating the polarization vectorspinors.

$$\epsilon^{\mu}_{+}(k) = \frac{-e^{+\mathrm{i}\phi}}{\sqrt{2}}\,(0; \cos\theta\cos\phi - \mathrm{i}\sin\phi, \cos\theta\sin\phi + \mathrm{i}\cos\phi, -\sin\theta) \tag{X.95a}$$

$$\epsilon^{\mu}_{-}(k) = \frac{e^{-\mathrm{i}\phi}}{\sqrt{2}}\,(0; \cos\theta\cos\phi + \mathrm{i}\sin\phi, \cos\theta\sin\phi - \mathrm{i}\cos\phi, -\sin\theta) \tag{X.95b}$$

$$\epsilon^{\mu}_{0}(k) = \frac{1}{m}\left(|\vec{k}|; k^0\sin\theta\cos\phi, k^0\sin\theta\sin\phi, k^0\cos\theta\right) \tag{X.95c}$$

Determining the mass from the momenta is a numerically haphazardous for light particles. Therefore, we accept some redundancy and pass the mass explicitely.

For the case that the momentum lies totally in the z-direction we take the convention $\cos\phi = 1$ and $\sin\phi = 0$.

⟨*Implementation of polarization vectorspinors*⟩≡

```
  pure function eps (mass, k, s) result (e)
    type(vector) :: e
    real(kind=default), intent(in) :: mass
    type(momentum), intent(in) :: k
    integer, intent(in) :: s
    real(kind=default) :: kabs, kabs2, sqrt2, m
    real(kind=default) :: cos_phi, sin_phi, cos_th, sin_th
    complex(kind=default) :: epiphi, emiphi
    sqrt2 = sqrt (2.0_default)
    kabs2 = dot_product (k%x, k%x)
    m = abs(mass)
    if (kabs2 > 0) then
       kabs = sqrt (kabs2)
       if ((k%x(1) == 0) .and. (k%x(2) == 0)) then
          cos_phi = 1
          sin_phi = 0
       else
          cos_phi = k%x(1) / sqrt(k%x(1)**2 + k%x(2)**2)
          sin_phi = k%x(2) / sqrt(k%x(1)**2 + k%x(2)**2)
       end if
       cos_th = k%x(3) / kabs
       sin_th = sqrt(1 - cos_th**2)
       epiphi = cos_phi + (0,1) * sin_phi
       emiphi = cos_phi - (0,1) * sin_phi
       e%t = 0
       e%x = 0
       select case (s)
       case (1)
          e%x(1) = epiphi * (-cos_th * cos_phi + (0,1) * sin_phi) / sqrt2
          e%x(2) = epiphi * (-cos_th * sin_phi - (0,1) * cos_phi) / sqrt2
          e%x(3) = epiphi * ( sin_th / sqrt2)
       case (-1)
          e%x(1) = emiphi * ( cos_th * cos_phi + (0,1) * sin_phi) / sqrt2
          e%x(2) = emiphi * ( cos_th * sin_phi - (0,1) * cos_phi) / sqrt2
          e%x(3) = emiphi * (-sin_th / sqrt2)
       case (0)
          if (m > 0) then
             e%t = kabs / m
             e%x = k%t / (m*kabs) * k%x
          end if
       case (4)
          if (m > 0) then
             e = (1 / m) * k
          else
             e = (1 / k%t) * k
          end if
       end select
    else   !!! for particles in their rest frame defined to be
           !!! polarized along the 3-direction
       e%t = 0
       e%x = 0
```

```
      select case (s)
      case (1)
         e%x(1) = cmplx ( - 1,   0, kind=default) / sqrt2
         e%x(2) = cmplx (   0,   1, kind=default) / sqrt2
      case (-1)
         e%x(1) = cmplx (   1,   0, kind=default) / sqrt2
         e%x(2) = cmplx (   0,   1, kind=default) / sqrt2
      case (0)
         if (m > 0) then
            e%x(3) = 1
         end if
      case (4)
         if (m > 0) then
            e = (1 / m) * k
         else
            e = (1 / k%t) * k
         end if
      end select
   end if
 end function eps
```

⟨*Implementation of polarization vectorspinors*⟩+≡

```
 pure function ueps (m, k, s) result (t)
   type(vectorspinor) :: t
   real(kind=default), intent(in) :: m
   type(momentum), intent(in) :: k
   integer, intent(in) :: s
   integer :: i
   type(vector) :: ep, e0, em
   type(bispinor) :: up, um
   do i = 1, 4
     t%psi(i)%a = 0
   end do
   select case (s)
   case (2)
      ep = eps (m, k, 1)
      up = u (m, k, 1)
      t = outer_product (ep, up)
   case (1)
      ep = eps (m, k, 1)
      e0 = eps (m, k, 0)
      up = u (m, k, 1)
      um = u (m, k, -1)
      t = (1 / sqrt (3.0_default)) * (outer_product (ep, um) &
           + sqrt (2.0_default) * outer_product (e0, up))
   case (-1)
      e0 = eps (m, k, 0)
      em = eps (m, k, -1)
      up = u (m, k, 1)
      um = u (m, k, -1)
      t = (1 / sqrt (3.0_default)) * (sqrt (2.0_default) * &
           outer_product (e0, um) + outer_product (em, up))
   case (-2)
      em = eps (m, k, -1)
      um = u (m, k, -1)
```

```
      t = outer_product (em, um)
    end select
  end function ueps
```

⟨*Implementation of polarization vectorspinors*⟩+≡

```
  pure function veps (m, k, s) result (t)
    type(vectorspinor) :: t
    real(kind=default), intent(in) :: m
    type(momentum), intent(in) :: k
    integer, intent(in) :: s
    integer :: i
    type(vector) :: ep, e0, em
    type(bispinor) :: vp, vm
    do i = 1, 4
      t%psi(i)%a = 0
    end do
    select case (s)
    case (2)
       ep = conjg(eps (m, k, 1))
       vp = v (m, k, 1)
       t = outer_product (ep, vp)
    case (1)
       ep = conjg(eps (m, k, 1))
       e0 = conjg(eps (m, k, 0))
       vp = v (m, k, 1)
       vm = v (m, k, -1)
       t = (1 / sqrt (3.0_default)) * (outer_product (ep, vm) &
            + sqrt (2.0_default) * outer_product (e0, vp))
    case (-1)
       e0 = conjg(eps (m, k,  0))
       em = conjg(eps (m, k, -1))
       vp = v (m, k, 1)
       vm = v (m, k, -1)
       t = (1 / sqrt (3.0_default)) * (sqrt (2.0_default) &
            * outer_product (e0, vm) + outer_product (em, vp))
    case (-2)
       em = conjg(eps (m, k, -1))
       vm = v (m, k, -1)
       t = outer_product (em, vm)
    end select
  end function veps
```

⟨*Implementation of polarization vectorspinors*⟩+≡

```
  pure function outer_product (ve, sp) result (vs)
    type(vectorspinor) :: vs
    type(vector), intent(in) :: ve
    type(bispinor), intent(in) :: sp
    integer :: i
    vs%psi(1)%a(1:4) = ve%t * sp%a(1:4)
    do i = 1, 3
       vs%psi((i+1))%a(1:4) = ve%x(i) * sp%a(1:4)
    end do
  end function outer_product
```

X.16 Color

⟨omega_color.f90⟩≡
```
  ⟨Copyleft⟩
  module omega_color
    use kinds
    implicit none
    private
    ⟨Declaration of color types⟩
    ⟨Declaration of color functions⟩
    integer, parameter, public :: omega_color_2010_01_A = 0
  contains
    ⟨Implementation of color functions⟩
  end module omega_color
```

X.16.1 Color Sum

⟨Declaration of color types⟩≡
```
  public :: omega_color_factor
  type omega_color_factor
     integer :: i1, i2
     real(kind=default) :: factor
  end type omega_color_factor
```

⟨Declaration of color functions⟩≡
```
  public :: omega_color_sum
```

The `!$omp` instruction will result in parallel code if compiled with support for OpenMP otherwise it is ignored.

⟨Implementation of color functions⟩≡
```
  ⟨pure unless OpenMP⟩
  function omega_color_sum (flv, hel, amp, cf) result (amp2)
    complex(kind=default) :: amp2
    integer, intent(in) :: flv, hel
    complex(kind=default), dimension(:,:,:), intent(in) :: amp
    type(omega_color_factor), dimension(:), intent(in) :: cf
    integer :: n
    amp2 = 0
    !$omp parallel do reduction(+:amp2)
    do n = 1, size (cf)
       amp2 = amp2 + cf(n)%factor * &
                     amp(flv,cf(n)%i1,hel) * conjg (amp(flv,cf(n)%i2,hel))
    end do
    !$omp end parallel do
  end function omega_color_sum
```

In the bytecode for the OVM, we only save the symmetric part of the color factor table. This almost halves the size of n gluon amplitudes for $n > 6$. For $2 \to (5,6)\,g$ the reduced color factor table still amounts for $\sim (75,93)\%$ of the bytecode, making it desirable to omit it completely by computing it dynamically to reduce memory requirements. Note that $2\mathrm{Re}(A_{i_1} A^*_{i_2}) = A_{i_1} A^*_{i_2} + A_{i_2} A^*_{i_1}$.

⟨Declaration of color functions⟩+≡
```
  public :: ovm_color_sum
```

⟨*Implementation of color functions*⟩+≡

```
⟨pure unless OpenMP⟩
function ovm_color_sum (flv, hel, amp, cf) result (amp2)
  real(kind=default) :: amp2
  integer, intent(in) :: flv, hel
  complex(kind=default), dimension(:,:,:), intent(in) :: amp
  type(omega_color_factor), dimension(:), intent(in) :: cf
  integer :: n
  amp2 = 0
  !$omp parallel do reduction(+:amp2)
  do n = 1, size (cf)
     if (cf(n)%i1 == cf(n)%i2) then
        amp2 = amp2 + cf(n)%factor * &
               real(amp(flv,cf(n)%i1,hel) * conjg(amp(flv,cf(n)%i2,hel)))
     else
        amp2 = amp2 + cf(n)%factor * 2 * &
               real(amp(flv,cf(n)%i1,hel) * conjg(amp(flv,cf(n)%i2,hel)))
     end if
  end do
  !$omp end parallel do
end function ovm_color_sum
```

X.17 Utilities

⟨omega_utils.f90⟩≡

```
⟨Copyleft⟩
module omega_utils
  use kinds
  use omega_vectors
  use omega_polarizations
  implicit none
  private
  ⟨Declaration of utility functions⟩
  ⟨Numerical tolerances⟩
  integer, parameter, public :: omega_utils_2010_01_A = 0
contains
  ⟨Implementation of utility functions⟩
end module omega_utils
```

X.17.1 Helicity Selection Rule Heuristics

⟨*Declaration of utility functions*⟩≡

```
public :: omega_update_helicity_selection
```

⟨*Implementation of utility functions*⟩≡

```
pure subroutine omega_update_helicity_selection &
           (count, amp, max_abs, sum_abs, mask, threshold, cutoff, mask_dirty)
  integer, intent(inout) :: count
  complex(kind=default), dimension(:,:,:), intent(in) :: amp
  real(kind=default), dimension(:), intent(inout) :: max_abs
  real(kind=default), intent(inout) :: sum_abs
  logical, dimension(:), intent(inout) :: mask
  real(kind=default), intent(in) :: threshold
```

```
    integer, intent(in) :: cutoff
    logical, intent(out) :: mask_dirty
    integer :: h
    real(kind=default) :: avg
    mask_dirty = .false.
    if (threshold > 0) then
       count = count + 1
       if (count <= cutoff) then
          forall (h = lbound (amp, 3) : ubound (amp, 3))
             max_abs(h) = max (max_abs(h), maxval (abs (amp(:,:,h))))
          end forall
          sum_abs = sum_abs + sum (abs (amp))
          if (count == cutoff) then
             avg = sum_abs / size (amp) / cutoff
             mask = max_abs >= threshold * epsilon (avg) * avg
             mask_dirty = .true.
          end if
       end if
    end if
  end subroutine omega_update_helicity_selection
```

X.17.2 Diagnostics

⟨*Declaration of utility functions*⟩+≡

```
  public :: omega_report_helicity_selection
```

We shoul try to use `msg_message` from WHIZARD's `diagnostics` module, but this would spoil independent builds.

⟨*Implementation of utility functions*⟩+≡

```
  subroutine omega_report_helicity_selection (mask, spin_states, threshold, unit)
    logical, dimension(:), intent(in) :: mask
    integer, dimension(:,:), intent(in) :: spin_states
    real(kind=default), intent(in) :: threshold
    integer, intent(in), optional :: unit
    integer :: u
    integer :: h, i
    if (present(unit)) then
       u = unit
    else
       u = 6
    end if
    if (u >= 0) then
       write (unit = u, &
            fmt = "('| ','Contributing Helicity Combinations: ', I5, ' of ', I5)") &
            count (mask), size (mask)
       write (unit = u, &
            fmt = "('| ','Threshold: amp / avg > ', E9.2, ' = ', E9.2, ' * epsilon()')") &
            threshold * epsilon (threshold), threshold
       i = 0
       do h = 1, size (mask)
          if (mask(h)) then
             i = i + 1
             write (unit = u, fmt = "('| ',I4,': ',20I4)") i, spin_states (:, h)
          end if
```

```
    end do
  end if
end subroutine omega_report_helicity_selection
```

⟨*Declaration of utility functions*⟩+≡

```
public :: omega_ward_warn, omega_ward_panic
```

The O'Mega amplitudes have only one particle off shell and are the sum of *all* possible diagrams with the other particles on-shell.

The problem with these gauge checks is that are numerically very small amplitudes that vanish analytically and that violate transversality. The hard part is to determine the thresholds that make threse tests usable.

⟨*Implementation of utility functions*⟩+≡

```
subroutine omega_ward_warn (name, m, k, e)
  character(len=*), intent(in) :: name
  real(kind=default), intent(in) :: m
  type(momentum), intent(in) :: k
  type(vector), intent(in) :: e
  type(vector) :: ek
  real(kind=default) :: abs_eke, abs_ek_abs_e
  ek = eps (m, k, 4)
  abs_eke = abs (ek * e)
  abs_ek_abs_e = abs (ek) * abs (e)
  print *, name, ":", abs_eke / abs_ek_abs_e, abs (ek), abs (e)
  if (abs_eke > 1000 * epsilon (abs_ek_abs_e)) then
     print *, "O'Mega: warning: non-transverse vector field: ", &
          name, ":", abs_eke / abs_ek_abs_e, abs (e)
  end if
end subroutine omega_ward_warn
```

⟨*Implementation of utility functions*⟩+≡

```
subroutine omega_ward_panic (name, m, k, e)
  character(len=*), intent(in) :: name
  real(kind=default), intent(in) :: m
  type(momentum), intent(in) :: k
  type(vector), intent(in) :: e
  type(vector) :: ek
  real(kind=default) :: abs_eke, abs_ek_abs_e
  ek = eps (m, k, 4)
  abs_eke = abs (ek * e)
  abs_ek_abs_e = abs (ek) * abs (e)
  if (abs_eke > 1000 * epsilon (abs_ek_abs_e)) then
     print *, "O'Mega: panic: non-transverse vector field: ", &
          name, ":", abs_eke / abs_ek_abs_e, abs (e)
     stop
  end if
end subroutine omega_ward_panic
```

⟨*Declaration of utility functions*⟩+≡

```
public :: omega_slavnov_warn, omega_slavnov_panic
```

⟨*Implementation of utility functions*⟩+≡

```
subroutine omega_slavnov_warn (name, m, k, e, phi)
  character(len=*), intent(in) :: name
  real(kind=default), intent(in) :: m
```

```
    type(momentum), intent(in) :: k
    type(vector), intent(in) :: e
    complex(kind=default), intent(in) :: phi
    type(vector) :: ek
    real(kind=default) :: abs_eke, abs_ek_abs_e
    ek = eps (m, k, 4)
    abs_eke = abs (ek * e - phi)
    abs_ek_abs_e = abs (ek) * abs (e)
    print *, name, ":", abs_eke / abs_ek_abs_e, abs (ek), abs (e)
    if (abs_eke > 1000 * epsilon (abs_ek_abs_e)) then
       print *, "O'Mega: warning: non-transverse vector field: ", &
            name, ":", abs_eke / abs_ek_abs_e, abs (e)
    end if
  end subroutine omega_slavnov_warn
```

⟨*Implementation of utility functions*⟩+≡

```
  subroutine omega_slavnov_panic (name, m, k, e, phi)
    character(len=*), intent(in) :: name
    real(kind=default), intent(in) :: m
    type(momentum), intent(in) :: k
    type(vector), intent(in) :: e
    complex(kind=default), intent(in) :: phi
    type(vector) :: ek
    real(kind=default) :: abs_eke, abs_ek_abs_e
    ek = eps (m, k, 4)
    abs_eke = abs (ek * e - phi)
    abs_ek_abs_e = abs (ek) * abs (e)
    if (abs_eke > 1000 * epsilon (abs_ek_abs_e)) then
       print *, "O'Mega: panic: non-transverse vector field: ", &
            name, ":", abs_eke / abs_ek_abs_e, abs (e)
       stop
    end if
  end subroutine omega_slavnov_panic
```

⟨*Declaration of utility functions*⟩+≡

```
  public :: omega_check_arguments_warn, omega_check_arguments_panic
```

⟨*Implementation of utility functions*⟩+≡

```
  subroutine omega_check_arguments_warn (n, k)
    integer, intent(in) :: n
    real(kind=default), dimension(0:,:), intent(in) :: k
    integer :: i
    i = size(k,dim=1)
    if (i /= 4) then
       print *, "O'Mega: warning: wrong # of dimensions:", i
    end if
    i = size(k,dim=2)
    if (i /= n) then
       print *, "O'Mega: warning: wrong # of momenta:", i, &
            ", expected", n
    end if
  end subroutine omega_check_arguments_warn
```

⟨*Implementation of utility functions*⟩+≡

```
  subroutine omega_check_arguments_panic (n, k)
    integer, intent(in) :: n
```

```
    real(kind=default), dimension(0:,:), intent(in) :: k
    logical :: error
    integer :: i
    error = .false.
    i = size(k,dim=1)
    if (i /= n) then
       print *, "O'Mega: warning: wrong # of dimensions:", i
       error = .true.
    end if
    i = size(k,dim=2)
    if (i /= n) then
       print *, "O'Mega: warning: wrong # of momenta:", i, &
            ", expected", n
       error = .true.
    end if
    if (error) then
       stop
    end if
  end subroutine omega_check_arguments_panic
```

⟨*Declaration of utility functions*⟩+≡

```
  public :: omega_check_helicities_warn, omega_check_helicities_panic
  private :: omega_check_helicity
```

⟨*Implementation of utility functions*⟩+≡

```
  function omega_check_helicity (m, smax, s) result (error)
    real(kind=default), intent(in) :: m
    integer, intent(in) :: smax, s
    logical :: error
    select case (smax)
    case (0)
       error = (s /= 0)
    case (1)
       error = (abs (s) /= 1)
    case (2)
       if (m == 0.0_default) then
          error = .not. (abs (s) == 1 .or. abs (s) == 4)
       else
          error = .not. (abs (s) <= 1 .or. abs (s) == 4)
       end if
    case (4)
       error = .true.
    case default
       error = .true.
    end select
  end function omega_check_helicity
```

⟨*Implementation of utility functions*⟩+≡

```
  subroutine omega_check_helicities_warn (m, smax, s)
    real(kind=default), dimension(:), intent(in) :: m
    integer, dimension(:), intent(in) :: smax, s
    integer :: i
    do i = 1, size (m)
       if (omega_check_helicity (m(i), smax(i), s(i))) then
          print *, "O'Mega: warning: invalid helicity", s(i)
       end if
```

```
    end do
  end subroutine omega_check_helicities_warn
```

⟨*Implementation of utility functions*⟩+≡

```
  subroutine omega_check_helicities_panic (m, smax, s)
    real(kind=default), dimension(:), intent(in) :: m
    integer, dimension(:), intent(in) :: smax, s
    logical :: error
    logical :: error1
    integer :: i
    error = .false.
    do i = 1, size (m)
       error1 = omega_check_helicity (m(i), smax(i), s(i))
       if (error1) then
          print *, "O'Mega: panic: invalid helicity", s(i)
          error = .true.
       end if
    end do
    if (error) then
       stop
    end if
  end subroutine omega_check_helicities_panic
```

⟨*Declaration of utility functions*⟩+≡

```
  public :: omega_check_momenta_warn, omega_check_momenta_panic
  private :: check_momentum_conservation, check_mass_shell
```

⟨*Numerical tolerances*⟩≡

```
  integer, parameter, private :: MOMENTUM_TOLERANCE = 10000
```

⟨*Implementation of utility functions*⟩+≡

```
  function check_momentum_conservation (k) result (error)
    real(kind=default), dimension(0:,:), intent(in) :: k
    logical :: error
    error = any (abs (sum (k(:,3:), dim = 2) - k(:,1) - k(:,2)) > &
         MOMENTUM_TOLERANCE * epsilon (maxval (abs (k), dim = 2)))
    if (error) then
       print *, sum (k(:,3:), dim = 2) - k(:,1) - k(:,2)
       print *, MOMENTUM_TOLERANCE * epsilon (maxval (abs (k), dim = 2)), &
            maxval (abs (k), dim = 2)
    end if
  end function check_momentum_conservation
```

⟨*Numerical tolerances*⟩+≡

```
  integer, parameter, private :: ON_SHELL_TOLERANCE = 1000000
```

⟨*Implementation of utility functions*⟩+≡

```
  function check_mass_shell (m, k) result (error)
    real(kind=default), intent(in) :: m
    real(kind=default), dimension(0:), intent(in) :: k
    real(kind=default) :: e2
    logical :: error
    e2 = k(1)**2 + k(2)**2 + k(3)**2 + m**2
    error = abs (k(0)**2 - e2) > ON_SHELL_TOLERANCE * epsilon (max (k(0)**2, e2))
    if (error) then
       print *, k(0)**2 - e2
       print *, ON_SHELL_TOLERANCE * epsilon (max (k(0)**2, e2)), max (k(0)**2, e2)
```

```
    end if
  end function check_mass_shell
```

⟨*Implementation of utility functions*⟩+≡

```
  subroutine omega_check_momenta_warn (m, k)
    real(kind=default), dimension(:), intent(in) :: m
    real(kind=default), dimension(0:,:), intent(in) :: k
    integer :: i
    if (check_momentum_conservation (k)) then
       print *, "O'Mega: warning: momentum not conserved"
    end if
    do i = 1, size(m)
       if (check_mass_shell (m(i), k(:,i))) then
          print *, "O'Mega: warning: particle #", i, "not on-shell"
       end if
    end do
  end subroutine omega_check_momenta_warn
```

⟨*Implementation of utility functions*⟩+≡

```
  subroutine omega_check_momenta_panic (m, k)
    real(kind=default), dimension(:), intent(in) :: m
    real(kind=default), dimension(0:,:), intent(in) :: k
    logical :: error
    logical :: error1
    integer :: i
    error = check_momentum_conservation (k)
    if (error) then
       print *, "O'Mega: panic: momentum not conserved"
    end if
    do i = 1, size(m)
       error1 = check_mass_shell (m(i), k(0:,i))
       if (error1) then
          print *, "O'Mega: panic: particle #", i, "not on-shell"
          error = .true.
       end if
    end do
    if (error) then
       stop
    end if
  end subroutine omega_check_momenta_panic
```

X.17.3 Obsolete Summation

Spin/Helicity Summation

⟨*Declaration of obsolete utility functions*⟩≡

```
  public :: omega_sum, omega_sum_nonzero, omega_nonzero
  private :: state_index
```

⟨*Implementation of obsolete utility functions*⟩≡

```
  pure function omega_sum (omega, p, states, fixed) result (sigma)
    real(kind=default) :: sigma
    real(kind=default), dimension(0:,:), intent(in) :: p
    integer, dimension(:), intent(in), optional :: states, fixed
    ⟨interface for O'Mega Amplitude⟩
```

```
    integer, dimension(size(p,dim=2)) :: s, nstates
    integer :: j
    complex(kind=default) :: a
    if (present (states)) then
       nstates = states
    else
       nstates = 2
    end if
    sigma = 0
    s = -1
    sum_spins: do
       if (present (fixed)) then
          !!! print *, 's = ', s, ', fixed = ', fixed, ', nstates = ', nstates, &
          !!!    ', fixed|s = ', merge (fixed, s, mask = nstates == 0)
          a = omega (p, merge (fixed, s, mask = nstates == 0))
       else
          a = omega (p, s)
       end if
       sigma = sigma + a * conjg(a)
       ⟨Step s like a n-ary number and terminate when all (s == -1)⟩
    end do sum_spins
    sigma = sigma / num_states (2, nstates(1:2))
  end function omega_sum
```

We're looping over all spins like a n-ary numbers $(-1,\ldots,-1,-1)$, $(-1,\ldots,-1,0)$, $(-1,\ldots,-1,1)$, $(-1,\ldots,0,-1)$, $\ldots$, $(1,\ldots,1,0)$, $(1,\ldots,1,1)$:

⟨*Step* `s` *like a n-ary number and terminate when* `all (s == -1)`⟩≡

```
  do j = size (p, dim = 2), 1, -1
     select case (nstates (j))
     case (3) ! massive vectors
        s(j) = modulo (s(j) + 2, 3) - 1
     case (2) ! spinors, massless vectors
        s(j) = - s(j)
     case (1) ! scalars
        s(j) = -1
     case (0) ! fized spin
        s(j) = -1
     case default ! ???
        s(j) = -1
     end select
     if (s(j) /= -1) then
        cycle sum_spins
     end if
  end do
  exit sum_spins
```

The dual operation evaluates an n-number:

⟨*Implementation of obsolete utility functions*⟩+≡

```
  pure function state_index (s, states) result (n)
    integer, dimension(:), intent(in) :: s
    integer, dimension(:), intent(in), optional :: states
    integer :: n
    integer :: j, p
    n = 1
    p = 1
```

```
    if (present (states)) then
       do j = size (s), 1, -1
          select case (states(j))
          case (3)
             n = n + p * (s(j) + 1)
          case (2)
             n = n + p * (s(j) + 1) / 2
          end select
          p = p * states(j)
       end do
    else
       do j = size (s), 1, -1
          n = n + p * (s(j) + 1) / 2
          p = p * 2
       end do
    end if
  end function state_index
```

⟨interface *for O'Mega Amplitude*⟩≡

```
  interface
     pure function omega (p, s) result (me)
       use kinds
       implicit none
       complex(kind=default) :: me
       real(kind=default), dimension(0:,:), intent(in) :: p
       integer, dimension(:), intent(in) :: s
     end function omega
  end interface
```

⟨*Declaration of obsolete utility functions*⟩+≡

```
  public :: num_states
```

⟨*Implementation of obsolete utility functions*⟩+≡

```
  pure function num_states (n, states) result (ns)
    integer, intent(in) :: n
    integer, dimension(:), intent(in), optional :: states
    integer :: ns
    if (present (states)) then
       ns = product (states, mask = states == 2 .or. states == 3)
    else
       ns = 2**n
    end if
  end function num_states
```

X.18 omega95

⟨omega95.f90⟩≡

```
  ⟨Copyleft⟩
  module omega95
    use constants
    use omega_spinors
    use omega_vectors
    use omega_polarizations
    use omega_tensors
    use omega_tensor_polarizations
```

```
    use omega_couplings
    use omega_spinor_couplings
    use omega_color
    use omega_utils
    public
  end module omega95
```

X.19 omega95 Revisited

```
⟨omega95_bispinors.f90⟩≡
  ⟨Copyleft⟩
  module omega95_bispinors
    use constants
    use omega_bispinors
    use omega_vectors
    use omega_vectorspinors
    use omega_polarizations
    use omega_vspinor_polarizations
    use omega_couplings
    use omega_bispinor_couplings
    use omega_color
    use omega_utils
    public
  end module omega95_bispinors
```

X.20 Testing

```
⟨omega_testtools.f90⟩≡
  ⟨Copyleft⟩
  module omega_testtools
    use kinds
    implicit none
    private
    real(kind=default), parameter, private :: ABS_THRESHOLD_DEFAULT = 1E-17
    real(kind=default), parameter, private :: THRESHOLD_DEFAULT = 0.6
    real(kind=default), parameter, private :: THRESHOLD_WARN = 0.8
    ⟨Declaration of test support functions⟩
  contains
    ⟨Implementation of test support functions⟩
  end module omega_testtools
```

Quantify the agreement of two real or complex numbers

$$\text{agreement}(x,y) = \frac{\ln \Delta(x,y)}{\ln \epsilon} \in [0,1] \tag{X.96}$$

with

$$\Delta(x,y) = \frac{|x-y|}{\max(|x|,|y|)} \tag{X.97}$$

and values outside $[0,1]$ replaced the closed value in the interval. In other words

- 1 for $x - y = \max(|x|,|y|) \cdot \mathcal{O}(\epsilon)$ and

- 0 for $x - y = \max(|x|, |y|) \cdot \mathcal{O}(1)$

with logarithmic interpolation. The cases $x = 0$ and $y = 0$ must be treated separately.

⟨*Declaration of test support functions*⟩≡
```
public :: agreement
interface agreement
   module procedure agreement_real, agreement_complex, &
          agreement_real_complex, agreement_complex_real, &
          agreement_integer_complex, agreement_complex_integer, &
          agreement_integer_real, agreement_real_integer
end interface
private :: agreement_real, agreement_complex, &
     agreement_real_complex, agreement_complex_real, &
     agreement_integer_complex, agreement_complex_integer, &
     agreement_integer_real, agreement_real_integer
```

⟨*Implementation of test support functions*⟩≡
```
elemental function agreement_real (x, y, base) result (a)
  real(kind=default) :: a
  real(kind=default), intent(in) :: x, y
  real(kind=default), intent(in), optional :: base
  real(kind=default) :: scale, dxy
  if (present (base)) then
     scale = max (abs (x), abs (y), abs (base))
  else
     scale = max (abs (x), abs (y))
  end if
  if (ieee_is_nan (x) .or. ieee_is_nan (y)) then
     a = 0
  else if (scale <= 0) then
     a = -1
  else
     dxy = abs (x - y) / scale
     if (dxy <= 0.0_default) then
        a = 1
     else
        a = log (dxy) / log (epsilon (scale))
        a = max (0.0_default, min (1.0_default, a))
        if (ieee_is_nan (a)) then
           a = 0
        end if
     end if
  end if
  if (ieee_is_nan (a)) then
     a = 0
  end if
end function agreement_real
```

Poor man's replacement

⟨*Implementation of test support functions*⟩+≡
```
elemental function ieee_is_nan (x) result (yorn)
  logical :: yorn
  real (kind=default), intent(in) :: x
  yorn = (x /= x)
```

```
  end function ieee_is_nan
```

⟨*Implementation of test support functions*⟩+≡

```
  elemental function agreement_complex (x, y, base) result (a)
    real(kind=default) :: a
    complex(kind=default), intent(in) :: x, y
    real(kind=default), intent(in), optional :: base
    real(kind=default) :: scale, dxy
    if (present (base)) then
       scale = max (abs (x), abs (y), abs (base))
    else
       scale = max (abs (x), abs (y))
    end if
    if (      ieee_is_nan (real (x, kind=default)) .or. ieee_is_nan (aimag (x)) &
         .or. ieee_is_nan (real (y, kind=default)) .or. ieee_is_nan (aimag (y))) then
       a = 0
    else if (scale <= 0) then
       a = -1
    else
       dxy = abs (x - y) / scale
       if (dxy <= 0.0_default) then
          a = 1
       else
          a = log (dxy) / log (epsilon (scale))
          a = max (0.0_default, min (1.0_default, a))
          if (ieee_is_nan (a)) then
             a = 0
          end if
       end if
    end if
    if (ieee_is_nan (a)) then
       a = 0
    end if
  end function agreement_complex
```

⟨*Implementation of test support functions*⟩+≡

```
  elemental function agreement_real_complex (x, y, base) result (a)
    real(kind=default) :: a
    real(kind=default), intent(in) :: x
    complex(kind=default), intent(in) :: y
    real(kind=default), intent(in), optional :: base
    a = agreement_complex (cmplx (x, kind=default), y, base)
  end function agreement_real_complex
```

⟨*Implementation of test support functions*⟩+≡

```
  elemental function agreement_complex_real (x, y, base) result (a)
    real(kind=default) :: a
    complex(kind=default), intent(in) :: x
    real(kind=default), intent(in) :: y
    real(kind=default), intent(in), optional :: base
    a = agreement_complex (x, cmplx (y, kind=default), base)
  end function agreement_complex_real
```

⟨*Implementation of test support functions*⟩+≡

```
  elemental function agreement_integer_complex (x, y, base) result (a)
    real(kind=default) :: a
```

```
    integer, intent(in) :: x
    complex(kind=default), intent(in) :: y
    real(kind=default), intent(in), optional :: base
    a = agreement_complex (cmplx (x, kind=default), y, base)
  end function agreement_integer_complex
```

⟨*Implementation of test support functions*⟩+≡

```
  elemental function agreement_complex_integer (x, y, base) result (a)
    real(kind=default) :: a
    complex(kind=default), intent(in) :: x
    integer, intent(in) :: y
    real(kind=default), intent(in), optional :: base
    a = agreement_complex (x, cmplx (y, kind=default), base)
  end function agreement_complex_integer
```

⟨*Implementation of test support functions*⟩+≡

```
  elemental function agreement_integer_real (x, y, base) result (a)
    real(kind=default) :: a
    integer, intent(in) :: x
    real(kind=default), intent(in) :: y
    real(kind=default), intent(in), optional :: base
    a = agreement_real (real(x, kind=default), y, base)
  end function agreement_integer_real
```

⟨*Implementation of test support functions*⟩+≡

```
  elemental function agreement_real_integer (x, y, base) result (a)
    real(kind=default) :: a
    real(kind=default), intent(in) :: x
    integer, intent(in) :: y
    real(kind=default), intent(in), optional :: base
    a = agreement_real (x, real (y, kind=default), base)
  end function agreement_real_integer
```

⟨*Declaration of test support functions*⟩+≡

```
  public:: vanishes
  interface vanishes
     module procedure vanishes_real, vanishes_complex
  end interface
  private :: vanishes_real, vanishes_complex
```

⟨*Implementation of test support functions*⟩+≡

```
  elemental function vanishes_real (x, scale) result (a)
    real(kind=default) :: a
    real(kind=default), intent(in) :: x
    real(kind=default), intent(in), optional :: scale
    real(kind=default) :: scaled_x
    if (x == 0.0_default) then
       a = 1
       return
    else if (ieee_is_nan (x)) then
       a = 0
       return
    end if
    scaled_x = x
    if (present (scale)) then
       if (scale /= 0) then
          scaled_x = x / abs (scale)
```

```
      else
         a = 0
         return
      end if
   else
   end if
   a = log (abs (scaled_x)) / log (epsilon (scaled_x))
   a = max (0.0_default, min (1.0_default, a))
   if (ieee_is_nan (a)) then
      a = 0
   end if
 end function vanishes_real
```

⟨*Implementation of test support functions*⟩+≡

```
 elemental function vanishes_complex (x, scale) result (a)
   real(kind=default) :: a
   complex(kind=default), intent(in) :: x
   real(kind=default), intent(in), optional :: scale
   a = vanishes_real (abs (x), scale)
 end function vanishes_complex
```

⟨*Declaration of test support functions*⟩+≡

```
 public :: expect
 interface expect
    module procedure expect_integer, expect_real, expect_complex, &
         expect_real_integer, expect_integer_real, &
         expect_complex_integer, expect_integer_complex, &
         expect_complex_real, expect_real_complex
 end interface
 private :: expect_integer, expect_real, expect_complex, &
      expect_real_integer, expect_integer_real, &
      expect_complex_integer, expect_integer_complex, &
      expect_complex_real, expect_real_complex
```

⟨*Implementation of test support functions*⟩+≡

```
 subroutine expect_integer (x, x0, msg, passed, quiet, buffer, unit)
   integer, intent(in) :: x, x0
   character(len=*), intent(in) :: msg
   logical, intent(inout), optional :: passed
   logical, intent(in), optional :: quiet
   character(len=*), intent(inout), optional :: buffer
   integer, intent(in), optional :: unit
   logical :: failed, verbose
   character(len=*), parameter :: fmt = "(1X,A,': ',A)"
   character(len=*), parameter :: &
        fmt_verbose = "(1X,A,': ',A,' [expected ',I6,', got ',I6,']')"
   failed = .false.
   verbose = .true.
   if (present (quiet)) then
      verbose = .not.quiet
   end if
   if (x == x0) then
      if (verbose) then
         if (.not. (present (buffer) .or. present (unit))) then
            write (unit = *, fmt = fmt) msg, "passed"
         end if
```

```
        if (present (unit)) then
           write (unit = unit, fmt = fmt) msg, "passed"
        end if
        if (present (buffer)) then
           write (unit = buffer, fmt = fmt) msg, "passed"
        end if
     end if
  else
     if (.not. (present (buffer) .or. present (unit))) then
        write (unit = *, fmt = fmt_verbose) msg, "failed", x0, x
     end if
     if (present (unit)) then
        write (unit = unit, fmt = fmt_verbose) msg, "failed", x0, x
     end if
     if (present (buffer)) then
        write (unit = buffer, fmt = fmt_verbose) msg, "failed", x0, x
     end if
     failed = .true.
  end if
  if (present (passed)) then
     passed = passed .and. .not.failed
  end if
end subroutine expect_integer
```

⟨*Implementation of test support functions*⟩+≡

```
subroutine expect_real (x, x0, msg, passed, threshold, quiet, abs_threshold)
  real(kind=default), intent(in) :: x, x0
  character(len=*), intent(in) :: msg
  logical, intent(inout), optional :: passed
  real(kind=default), intent(in), optional :: threshold
  real(kind=default), intent(in), optional :: abs_threshold
  logical, intent(in), optional :: quiet
  logical :: failed, verbose
  real(kind=default) :: agreement_threshold, abs_agreement_threshold
  character(len=*), parameter :: fmt = "(1X,A,': ',A,' at ',I4,'%')"
  character(len=*), parameter :: fmt_verbose = "(1X,A,': ',A,' at ',I4,'%'," // &
       "' [expected ',E10.3,', got ',E10.3,']')"
  real(kind=default) :: a
  failed = .false.
  verbose = .true.
  if (present (quiet)) then
     verbose = .not.quiet
  end if
  if (x == x0) then
     if (verbose) then
        write (unit = *, fmt = fmt) msg, "passed", 100
     end if
  else
     if (x0 == 0) then
        a = vanishes (x)
     else
        a = agreement (x, x0)
     end if
     if (present (threshold)) then
        agreement_threshold = threshold
```

```
      else
         agreement_threshold = THRESHOLD_DEFAULT
      end if
      if (present (abs_threshold)) then
         abs_agreement_threshold = abs_threshold
      else
         abs_agreement_threshold = ABS_THRESHOLD_DEFAULT
      end if
      if (a >= agreement_threshold .or. &
          max(abs(x), abs(x0)) <= abs_agreement_threshold) then
         if (verbose) then
            if (a >= THRESHOLD_WARN) then
               write (unit = *, fmt = fmt) msg, "passed", int (a * 100)
            else
               write (unit = *, fmt = fmt_verbose) msg, "passed", int (a * 100), x0, x
            end if
         end if
      else
         failed = .true.
         write (unit = *, fmt = fmt_verbose) msg, "failed", int (a * 100), x0, x
      end if
    end if
    if (present (passed)) then
      passed = passed .and. .not.failed
    end if
  end subroutine expect_real
```

⟨*Implementation of test support functions*⟩+≡

```
  subroutine expect_complex (x, x0, msg, passed, threshold, quiet, abs_threshold)
    complex(kind=default), intent(in) :: x, x0
    character(len=*), intent(in) :: msg
    logical, intent(inout), optional :: passed
    real(kind=default), intent(in), optional :: threshold
    real(kind=default), intent(in), optional :: abs_threshold
    logical, intent(in), optional :: quiet
    logical :: failed, verbose
    real(kind=default) :: agreement_threshold, abs_agreement_threshold
    character(len=*), parameter :: fmt = "(1X,A,': ',A,' at ',I4,'%')"
    character(len=*), parameter :: fmt_verbose = "(1X,A,': ',A,' at ',I4,'%'," // &
         "' [expected (',E10.3,',',E10.3,'), got (',E10.3,',',E10.3,')]')"
    character(len=*), parameter :: fmt_phase = "(1X,A,': ',A,' at ',I4,'%'," // &
         "' [modulus passed at ',I4,'%',', phases ',F5.3,' vs. ',F5.3,']')"
    real(kind=default) :: a, a_modulus
    failed = .false.
    verbose = .true.
    if (present (quiet)) then
      verbose = .not.quiet
    end if
    if (x == x0) then
      if (verbose) then
         write (unit = *, fmt = fmt) msg, "passed", 100
      end if
    else
      if (x0 == 0) then
         a = vanishes (x)
```

```
      else
         a = agreement (x, x0)
      end if
      if (present (threshold)) then
         agreement_threshold = threshold
      else
         agreement_threshold = THRESHOLD_DEFAULT
      end if
      if (present (abs_threshold)) then
         abs_agreement_threshold = abs_threshold
      else
         abs_agreement_threshold = ABS_THRESHOLD_DEFAULT
      end if
      if (a >= agreement_threshold .or. &
          max(abs(x), abs(x0)) <= abs_agreement_threshold) then
         if (verbose) then
            if (a >= THRESHOLD_WARN) then
               write (unit = *, fmt = fmt) msg, "passed", int (a * 100)
            else
               write (unit = *, fmt = fmt_verbose) msg, "passed", int (a * 100), x0, x
            end if
         end if
      else
         a_modulus = agreement (abs (x), abs (x0))
         if (a_modulus >= agreement_threshold) then
            write (unit = *, fmt = fmt_phase) msg, "failed", int (a * 100), &
                 int (a_modulus * 100), &
                 atan2 (real (x, kind=default), aimag (x)), &
                 atan2 (real (x0, kind=default), aimag (x0))
         else
            write (unit = *, fmt = fmt_verbose) msg, "failed", int (a * 100), x0, x
         end if
         failed = .true.
      end if
   end if
   if (present (passed)) then
      passed = passed .and. .not.failed
   end if
 end subroutine expect_complex
```

⟨*Implementation of test support functions*⟩+≡

```
 subroutine expect_real_integer (x, x0, msg, passed, threshold, quiet)
   real(kind=default), intent(in) :: x
   integer, intent(in) :: x0
   character(len=*), intent(in) :: msg
   real(kind=default), intent(in), optional :: threshold
   logical, intent(inout), optional :: passed
   logical, intent(in), optional :: quiet
   call expect_real (x, real (x0, kind=default), msg, passed, threshold, quiet)
 end subroutine expect_real_integer
```

⟨*Implementation of test support functions*⟩+≡

```
 subroutine expect_integer_real (x, x0, msg, passed, threshold, quiet)
   integer, intent(in) :: x
   real(kind=default), intent(in) :: x0
```

```
    character(len=*), intent(in) :: msg
    real(kind=default), intent(in), optional :: threshold
    logical, intent(inout), optional :: passed
    logical, intent(in), optional :: quiet
    call expect_real (real (x, kind=default), x0, msg, passed, threshold, quiet)
  end subroutine expect_integer_real
```

⟨*Implementation of test support functions*⟩+≡

```
  subroutine expect_complex_integer (x, x0, msg, passed, threshold, quiet)
    complex(kind=default), intent(in) :: x
    integer, intent(in) :: x0
    character(len=*), intent(in) :: msg
    logical, intent(inout), optional :: passed
    real(kind=default), intent(in), optional :: threshold
    logical, intent(in), optional :: quiet
    call expect_complex (x, cmplx (x0, kind=default), msg, passed, threshold, quiet)
  end subroutine expect_complex_integer
```

⟨*Implementation of test support functions*⟩+≡

```
  subroutine expect_integer_complex (x, x0, msg, passed, threshold, quiet)
    integer, intent(in) :: x
    complex(kind=default), intent(in) :: x0
    character(len=*), intent(in) :: msg
    logical, intent(inout), optional :: passed
    real(kind=default), intent(in), optional :: threshold
    logical, intent(in), optional :: quiet
    call expect_complex (cmplx (x, kind=default), x0, msg, passed, threshold, quiet)
  end subroutine expect_integer_complex
```

⟨*Implementation of test support functions*⟩+≡

```
  subroutine expect_complex_real (x, x0, msg, passed, threshold, quiet)
    complex(kind=default), intent(in) :: x
    real(kind=default), intent(in) :: x0
    character(len=*), intent(in) :: msg
    logical, intent(inout), optional :: passed
    real(kind=default), intent(in), optional :: threshold
    logical, intent(in), optional :: quiet
    call expect_complex (x, cmplx (x0, kind=default), msg, passed, threshold, quiet)
  end subroutine expect_complex_real
```

⟨*Implementation of test support functions*⟩+≡

```
  subroutine expect_real_complex (x, x0, msg, passed, threshold, quiet)
    real(kind=default), intent(in) :: x
    complex(kind=default), intent(in) :: x0
    character(len=*), intent(in) :: msg
    logical, intent(inout), optional :: passed
    real(kind=default), intent(in), optional :: threshold
    logical, intent(in), optional :: quiet
    call expect_complex (cmplx (x, kind=default), x0, msg, passed, threshold, quiet)
  end subroutine expect_real_complex
```

⟨*Declaration of test support functions*⟩+≡

```
  public :: expect_zero
  interface expect_zero
     module procedure expect_zero_integer, expect_zero_real, expect_zero_complex
  end interface
  private :: expect_zero_integer, expect_zero_real, expect_zero_complex
```

⟨*Implementation of test support functions*⟩+≡
```
  subroutine expect_zero_integer (x, msg, passed)
    integer, intent(in) :: x
    character(len=*), intent(in) :: msg
    logical, intent(inout), optional :: passed
    call expect_integer (x, 0, msg, passed)
  end subroutine expect_zero_integer
```

⟨*Implementation of test support functions*⟩+≡
```
  subroutine expect_zero_real (x, scale, msg, passed, threshold, quiet)
    real(kind=default), intent(in) :: x, scale
    character(len=*), intent(in) :: msg
    logical, intent(inout), optional :: passed
    real(kind=default), intent(in), optional :: threshold
    logical, intent(in), optional :: quiet
    logical :: failed, verbose
    real(kind=default) :: agreement_threshold
    character(len=*), parameter :: fmt = "(1X,A,': ',A,' at ',I4,'%')"
    character(len=*), parameter :: fmt_verbose = "(1X,A,': ',A,' at ',I4,'%'," // &
         "' [expected 0 (relative to ',E10.3,') got ',E10.3,']')"
    real(kind=default) :: a
    failed = .false.
    verbose = .true.
    if (present (quiet)) then
       verbose = .not.quiet
    end if
    if (x == 0) then
       if (verbose) then
          write (unit = *, fmt = fmt) msg, "passed", 100
       end if
    else
       a = vanishes (x, scale = scale)
       if (present (threshold)) then
          agreement_threshold = threshold
       else
          agreement_threshold = THRESHOLD_DEFAULT
       end if
       if (a >= agreement_threshold) then
          if (verbose) then
             if (a >= THRESHOLD_WARN) then
                write (unit = *, fmt = fmt) msg, "passed", int (a * 100)
             else
                write (unit = *, fmt = fmt_verbose) msg, "passed", int (a * 100), scale, x
             end if
          end if
       else
          failed = .true.
          write (unit = *, fmt = fmt_verbose) msg, "failed", int (a * 100), scale, x
       end if
    end if
    if (present (passed)) then
       passed = passed .and. .not.failed
    end if
  end subroutine expect_zero_real
```

⟨*Implementation of test support functions*⟩+≡
```
subroutine expect_zero_complex (x, scale, msg, passed, threshold, quiet)
  complex(kind=default), intent(in) :: x
  real(kind=default), intent(in) :: scale
  character(len=*), intent(in) :: msg
  logical, intent(inout), optional :: passed
  real(kind=default), intent(in), optional :: threshold
  logical, intent(in), optional :: quiet
  call expect_zero_real (abs (x), scale, msg, passed, threshold, quiet)
end subroutine expect_zero_complex
```

⟨*Implementation of test support functions*⟩+≡
```
subroutine print_matrix (a)
  complex(kind=default), dimension(:,:), intent(in) :: a
  integer :: row
  do row = 1, size (a, dim=1)
     write (unit = *, fmt = "(10(tr2, f5.2, '+', f5.2, 'I'))") a(row,:)
  end do
end subroutine print_matrix
```

⟨*Declaration of test support functions*⟩+≡
```
public :: print_matrix
```

⟨test_omega95.f90⟩≡
```
⟨Copyleft⟩
program test_omega95
  use kinds
  use omega95
  use omega_testtools
  implicit none
  real(kind=default) :: m, pabs, qabs, w
  real(kind=default), dimension(0:3) :: r
  complex(kind=default) :: c_one, c_nil
  type(momentum) :: p, q, p0
  type(vector) :: vp, vq, vtest, v0
  type(tensor) :: ttest
  integer, dimension(8) :: date_time
  integer :: rsize
  logical :: passed
  call date_and_time (values = date_time)
  call random_seed (size = rsize)
  call random_seed (put = spread (product (date_time), dim = 1, ncopies = rsize))
  w = 1.4142
  c_one = 1.0_default
  c_nil = 0.0_default
  m = 13
  pabs = 42
  qabs = 137
  call random_number (r)
  vtest%t = cmplx (10.0_default * r(0), kind=default)
  vtest%x(1:3) = cmplx (10.0_default * r(1:3), kind=default)
  ttest = vtest.tprod.vtest
  call random_momentum (p, pabs, m)
  call random_momentum (q, qabs, m)
  call random_momentum (p0, 0.0_default, m)
  vp = p
```

```
  vq = q
  v0 = p0
  passed = .true.
  ⟨Test omega95⟩
  if (.not. passed) then
    stop 1
  end if
end program test_omega95
```

⟨*Test* omega95⟩≡

```
print *, "*** Checking the equations of motion ***:"
call expect (abs(f_vf(c_one,vp,u(m,p,+1))-m*u(m,p,+1)), 0, "|[p-m]u(+)|=0", passed)
call expect (abs(f_vf(c_one,vp,u(m,p,-1))-m*u(m,p,-1)), 0, "|[p-m]u(-)|=0", passed)
call expect (abs(f_vf(c_one,vp,v(m,p,+1))+m*v(m,p,+1)), 0, "|[p+m]v(+)|=0", passed)
call expect (abs(f_vf(c_one,vp,v(m,p,-1))+m*v(m,p,-1)), 0, "|[p+m]v(-)|=0", passed)
call expect (abs(f_fv(c_one,ubar(m,p,+1),vp)-m*ubar(m,p,+1)), 0, "|ubar(+)[p-m]|=0", passed)
call expect (abs(f_fv(c_one,ubar(m,p,-1),vp)-m*ubar(m,p,-1)), 0, "|ubar(-)[p-m]|=0", passed)
call expect (abs(f_fv(c_one,vbar(m,p,+1),vp)+m*vbar(m,p,+1)), 0, "|vbar(+)[p+m]|=0", passed)
call expect (abs(f_fv(c_one,vbar(m,p,-1),vp)+m*vbar(m,p,-1)), 0, "|vbar(-)[p+m]|=0", passed)
print *, "*** Checking the equations of motion for negative mass***:"
call expect (abs(f_vf(c_one,vp,u(-m,p,+1))+m*u(-m,p,+1)), 0, "|[p+m]u(+)|=0", passed)
call expect (abs(f_vf(c_one,vp,u(-m,p,-1))+m*u(-m,p,-1)), 0, "|[p+m]u(-)|=0", passed)
call expect (abs(f_vf(c_one,vp,v(-m,p,+1))-m*v(-m,p,+1)), 0, "|[p-m]v(+)|=0", passed)
call expect (abs(f_vf(c_one,vp,v(-m,p,-1))-m*v(-m,p,-1)), 0, "|[p-m]v(-)|=0", passed)
call expect (abs(f_fv(c_one,ubar(-m,p,+1),vp)+m*ubar(-m,p,+1)), 0, "|ubar(+)[p+m]|=0", passed)
call expect (abs(f_fv(c_one,ubar(-m,p,-1),vp)+m*ubar(-m,p,-1)), 0, "|ubar(-)[p+m]|=0", passed)
call expect (abs(f_fv(c_one,vbar(-m,p,+1),vp)-m*vbar(-m,p,+1)), 0, "|vbar(+)[p-m]|=0", passed)
call expect (abs(f_fv(c_one,vbar(-m,p,-1),vp)-m*vbar(-m,p,-1)), 0, "|vbar(-)[p-m]|=0", passed)
```

⟨*Test* omega95⟩+≡

```
print *, "*** Checking the normalization ***:"
call expect (ubar(m,p,+1)*u(m,p,+1), +2*m, "ubar(+)*u(+)=+2m", passed)
call expect (ubar(m,p,-1)*u(m,p,-1), +2*m, "ubar(-)*u(-)=+2m", passed)
call expect (vbar(m,p,+1)*v(m,p,+1), -2*m, "vbar(+)*v(+)=-2m", passed)
call expect (vbar(m,p,-1)*v(m,p,-1), -2*m, "vbar(-)*v(-)=-2m", passed)
call expect (ubar(m,p,+1)*v(m,p,+1),    0, "ubar(+)*v(+)=0  ", passed)
call expect (ubar(m,p,-1)*v(m,p,-1),    0, "ubar(-)*v(-)=0  ", passed)
call expect (vbar(m,p,+1)*u(m,p,+1),    0, "vbar(+)*u(+)=0  ", passed)
call expect (vbar(m,p,-1)*u(m,p,-1),    0, "vbar(-)*u(-)=0  ", passed)
print *, "*** Checking the normalization for negative masses***:"
call expect (ubar(-m,p,+1)*u(-m,p,+1), -2*m, "ubar(+)*u(+)=-2m", passed)
call expect (ubar(-m,p,-1)*u(-m,p,-1), -2*m, "ubar(-)*u(-)=-2m", passed)
call expect (vbar(-m,p,+1)*v(-m,p,+1), +2*m, "vbar(+)*v(+)=+2m", passed)
call expect (vbar(-m,p,-1)*v(-m,p,-1), +2*m, "vbar(-)*v(-)=+2m", passed)
call expect (ubar(-m,p,+1)*v(-m,p,+1),    0, "ubar(+)*v(+)=0  ", passed)
call expect (ubar(-m,p,-1)*v(-m,p,-1),    0, "ubar(-)*v(-)=0  ", passed)
call expect (vbar(-m,p,+1)*u(-m,p,+1),    0, "vbar(+)*u(+)=0  ", passed)
call expect (vbar(-m,p,-1)*u(-m,p,-1),    0, "vbar(-)*u(-)=0  ", passed)
```

⟨*Test* omega95⟩+≡

```
print *, "*** Checking the currents ***:"
call expect (abs(v_ff(c_one,ubar(m,p,+1),u(m,p,+1))-2*vp), 0, "ubar(+).V.u(+)=2p", passed)
call expect (abs(v_ff(c_one,ubar(m,p,-1),u(m,p,-1))-2*vp), 0, "ubar(-).V.u(-)=2p", passed)
call expect (abs(v_ff(c_one,vbar(m,p,+1),v(m,p,+1))-2*vp), 0, "vbar(+).V.v(+)=2p", passed)
call expect (abs(v_ff(c_one,vbar(m,p,-1),v(m,p,-1))-2*vp), 0, "vbar(-).V.v(-)=2p", passed)
print *, "*** Checking the currents for negative masses***:"
```

```
  call expect (abs(v_ff(c_one,ubar(-m,p,+1),u(-m,p,+1))-2*vp), 0, "ubar(+).V.u(+)=2p", passed)
  call expect (abs(v_ff(c_one,ubar(-m,p,-1),u(-m,p,-1))-2*vp), 0, "ubar(-).V.u(-)=2p", passed)
  call expect (abs(v_ff(c_one,vbar(-m,p,+1),v(-m,p,+1))-2*vp), 0, "vbar(+).V.v(+)=2p", passed)
  call expect (abs(v_ff(c_one,vbar(-m,p,-1),v(-m,p,-1))-2*vp), 0, "vbar(-).V.v(-)=2p", passed)
```

⟨*Test* omega95⟩+≡

```
  print *, "*** Checking current conservation ***:"
  call expect ((vp-vq)*v_ff(c_one,ubar(m,p,+1),u(m,q,+1)), 0, "d(ubar(+).V.u(+))=0", passed)
  call expect ((vp-vq)*v_ff(c_one,ubar(m,p,-1),u(m,q,-1)), 0, "d(ubar(-).V.u(-))=0", passed)
  call expect ((vp-vq)*v_ff(c_one,vbar(m,p,+1),v(m,q,+1)), 0, "d(vbar(+).V.v(+))=0", passed)
  call expect ((vp-vq)*v_ff(c_one,vbar(m,p,-1),v(m,q,-1)), 0, "d(vbar(-).V.v(-))=0", passed)
  print *, "*** Checking current conservation for negative masses***:"
  call expect ((vp-vq)*v_ff(c_one,ubar(-m,p,+1),u(-m,q,+1)), 0, "d(ubar(+).V.u(+))=0", passed)
  call expect ((vp-vq)*v_ff(c_one,ubar(-m,p,-1),u(-m,q,-1)), 0, "d(ubar(-).V.u(-))=0", passed)
  call expect ((vp-vq)*v_ff(c_one,vbar(-m,p,+1),v(-m,q,+1)), 0, "d(vbar(+).V.v(+))=0", passed)
  call expect ((vp-vq)*v_ff(c_one,vbar(-m,p,-1),v(-m,q,-1)), 0, "d(vbar(-).V.v(-))=0", passed)
```

⟨*Test* omega95⟩+≡

```
  if (m == 0) then
     print *, "*** Checking axial current conservation ***:"
     call expect ((vp-vq)*a_ff(c_one,ubar(m,p,+1),u(m,q,+1)), 0, "d(ubar(+).A.u(+))=0", passed)
     call expect ((vp-vq)*a_ff(c_one,ubar(m,p,-1),u(m,q,-1)), 0, "d(ubar(-).A.u(-))=0", passed)
     call expect ((vp-vq)*a_ff(c_one,vbar(m,p,+1),v(m,q,+1)), 0, "d(vbar(+).A.v(+))=0", passed)
     call expect ((vp-vq)*a_ff(c_one,vbar(m,p,-1),v(m,q,-1)), 0, "d(vbar(-).A.v(-))=0", passed)
  end if
```

⟨*Test* omega95⟩+≡

```
  print *, "*** Checking implementation of the sigma vertex funktions ***:"
  call expect ((vp*tvam_ff(c_one,c_nil,ubar(m,p,+1),u(m,q,+1),q) - (p*q-m**2)*(ubar(m,p,+1)*u(m,q,+1
               "p*[ubar(p,+).(Isigma*q).u(q,+)] - (p*q-m^2)*ubar(p,+).u(q,+) = 0", passed)
  call expect ((vp*tvam_ff(c_one,c_nil,ubar(m,p,-1),u(m,q,-1),q) - (p*q-m**2)*(ubar(m,p,-1)*u(m,q,-1
               "p*[ubar(p,-).(Isigma*q).u(q,-)] - (p*q-m^2)*ubar(p,-).u(q,-) = 0", passed)
  call expect ((vp*tvam_ff(c_one,c_nil,vbar(m,p,+1),v(m,q,+1),q) - (p*q-m**2)*(vbar(m,p,+1)*v(m,q,+1
               "p*[vbar(p,+).(Isigma*q).v(q,+)] - (p*q-m^2)*vbar(p,+).v(q,+) = 0", passed)
  call expect ((vp*tvam_ff(c_one,c_nil,vbar(m,p,-1),v(m,q,-1),q) - (p*q-m**2)*(vbar(m,p,-1)*v(m,q,-1
               "p*[vbar(p,-).(Isigma*q).v(q,-)] - (p*q-m^2)*vbar(p,-).v(q,-) = 0", passed)
  call expect ((ubar(m,p,+1)*f_tvamf(c_one,c_nil,vp,u(m,q,+1),q) - (p*q-m**2)*(ubar(m,p,+1)*u(m,q,+1
               "ubar(p,+).[p*(Isigma*q).u(q,+)] - (p*q-m^2)*ubar(p,+).u(q,+) = 0", passed)
  call expect ((ubar(m,p,-1)*f_tvamf(c_one,c_nil,vp,u(m,q,-1),q) - (p*q-m**2)*(ubar(m,p,-1)*u(m,q,-1
               "ubar(p,-).[p*(Isigma*q).u(q,-)] - (p*q-m^2)*ubar(p,-).u(q,-) = 0", passed)
  call expect ((vbar(m,p,+1)*f_tvamf(c_one,c_nil,vp,v(m,q,+1),q) - (p*q-m**2)*(vbar(m,p,+1)*v(m,q,+1
               "vbar(p,+).[p*(Isigma*q).v(q,+)] - (p*q-m^2)*vbar(p,+).v(q,+) = 0", passed)
  call expect ((vbar(m,p,-1)*f_tvamf(c_one,c_nil,vp,v(m,q,-1),q) - (p*q-m**2)*(vbar(m,p,-1)*v(m,q,-1
               "vbar(p,-).[p*(Isigma*q).v(q,-)] - (p*q-m^2)*vbar(p,-).v(q,-) = 0", passed)
  call expect ((f_ftvam(c_one,c_nil,ubar(m,p,+1),vp,q)*u(m,q,+1) - (p*q-m**2)*(ubar(m,p,+1)*u(m,q,+1
               "[ubar(p,+).p*(Isigma*q)].u(q,+) - (p*q-m^2)*ubar(p,+).u(q,+) = 0", passed)
  call expect ((f_ftvam(c_one,c_nil,ubar(m,p,-1),vp,q)*u(m,q,-1) - (p*q-m**2)*(ubar(m,p,-1)*u(m,q,-1
               "[ubar(p,-).p*(Isigma*q)].u(q,-) - (p*q-m^2)*ubar(p,-).u(q,-) = 0", passed)
  call expect ((f_ftvam(c_one,c_nil,vbar(m,p,+1),vp,q)*v(m,q,+1) - (p*q-m**2)*(vbar(m,p,+1)*v(m,q,+1
               "[vbar(p,+).p*(Isigma*q)].v(q,+) - (p*q-m^2)*vbar(p,+).v(q,+) = 0", passed)
  call expect ((f_ftvam(c_one,c_nil,vbar(m,p,-1),vp,q)*v(m,q,-1) - (p*q-m**2)*(vbar(m,p,-1)*v(m,q,-1
               "[vbar(p,-).p*(Isigma*q)].v(q,-) - (p*q-m^2)*vbar(p,-).v(q,-) = 0", passed)

  call expect ((vp*tvam_ff(c_nil,c_one,ubar(m,p,+1),u(m,q,+1),q) - (p*q+m**2)*p_ff(c_one,ubar(m,p,+1
               "p*[ubar(p,+).(Isigma*q).g5.u(q,+)] - (p*q+m^2)*ubar(p,+).g5.u(q,+) = 0", passed)
  call expect ((vp*tvam_ff(c_nil,c_one,ubar(m,p,-1),u(m,q,-1),q) - (p*q+m**2)*p_ff(c_one,ubar(m,p,-1
```

```
                "p*[ubar(p,-).(Isigma*q).g5.u(q,-)] - (p*q+m^2)*ubar(p,-).g5.u(q,-) = 0", passed)
call expect ((vp*tvam_ff(c_nil,c_one,vbar(m,p,+1),v(m,q,+1),q) - (p*q+m**2)*p_ff(c_one,vbar(m,p,+1
                "p*[vbar(p,+).(Isigma*q).g5.v(q,+)] - (p*q+m^2)*vbar(p,+).g5.v(q,+) = 0", passed)
call expect ((vp*tvam_ff(c_nil,c_one,vbar(m,p,-1),v(m,q,-1),q) - (p*q+m**2)*p_ff(c_one,vbar(m,p,-1
                "p*[vbar(p,-).(Isigma*q).g5.v(q,-)] - (p*q+m^2)*vbar(p,-).g5.v(q,-) = 0", passed)
call expect ((ubar(m,p,+1)*f_tvamf(c_nil,c_one,vp,u(m,q,+1),q) - (p*q+m**2)*p_ff(c_one,ubar(m,p,+1
                "p*[ubar(p,+).(Isigma*q).g5.u(q,+)] - (p*q+m^2)*ubar(p,+).g5.u(q,+) = 0", passed)
call expect ((ubar(m,p,-1)*f_tvamf(c_nil,c_one,vp,u(m,q,-1),q) - (p*q+m**2)*p_ff(c_one,ubar(m,p,-1
                "p*[ubar(p,-).(Isigma*q).g5.u(q,-)] - (p*q+m^2)*ubar(p,-).g5.u(q,-) = 0", passed)
call expect ((vbar(m,p,+1)*f_tvamf(c_nil,c_one,vp,v(m,q,+1),q) - (p*q+m**2)*p_ff(c_one,vbar(m,p,+1
                "p*[vbar(p,+).(Isigma*q).g5.v(q,+)] - (p*q+m^2)*vbar(p,+).g5.v(q,+) = 0", passed)
call expect ((vbar(m,p,-1)*f_tvamf(c_nil,c_one,vp,v(m,q,-1),q) - (p*q+m**2)*p_ff(c_one,vbar(m,p,-1
                "p*[vbar(p,-).(Isigma*q).g5.v(q,-)] - (p*q+m^2)*vbar(p,-).g5.v(q,-) = 0", passed)
call expect ((f_ftvam(c_nil,c_one,ubar(m,p,+1),vp,q)*u(m,q,+1) - (p*q+m**2)*p_ff(c_one,ubar(m,p,+1
                "p*[ubar(p,+).(Isigma*q).g5.u(q,+)] - (p*q+m^2)*ubar(p,+).g5.u(q,+) = 0", passed)
call expect ((f_ftvam(c_nil,c_one,ubar(m,p,-1),vp,q)*u(m,q,-1) - (p*q+m**2)*p_ff(c_one,ubar(m,p,-1
                "p*[ubar(p,-).(Isigma*q).g5.u(q,-)] - (p*q+m^2)*ubar(p,-).g5.u(q,-) = 0", passed)
call expect ((f_ftvam(c_nil,c_one,vbar(m,p,+1),vp,q)*v(m,q,+1) - (p*q+m**2)*p_ff(c_one,vbar(m,p,+1
                "p*[vbar(p,+).(Isigma*q).g5.v(q,+)] - (p*q+m^2)*vbar(p,+).g5.v(q,+) = 0", passed)
call expect ((f_ftvam(c_nil,c_one,vbar(m,p,-1),vp,q)*v(m,q,-1) - (p*q+m**2)*p_ff(c_one,vbar(m,p,-1
                "p*[vbar(p,-).(Isigma*q).g5.v(q,-)] - (p*q+m^2)*vbar(p,-).g5.v(q,-) = 0", passed)
```

⟨*Test* omega95⟩+≡

```
print *, "*** Checking polarisation vectors: ***"
call expect (conjg(eps(m,p, 1))*eps(m,p, 1), -1, "e( 1).e( 1)=-1", passed)
call expect (conjg(eps(m,p, 1))*eps(m,p,-1),  0, "e( 1).e(-1)= 0", passed)
call expect (conjg(eps(m,p,-1))*eps(m,p, 1),  0, "e(-1).e( 1)= 0", passed)
call expect (conjg(eps(m,p,-1))*eps(m,p,-1), -1, "e(-1).e(-1)=-1", passed)
call expect (                 p*eps(m,p, 1),  0, "    p.e( 1)= 0", passed)
call expect (                 p*eps(m,p,-1),  0, "    p.e(-1)= 0", passed)
if (m > 0) then
   call expect (conjg(eps(m,p, 1))*eps(m,p, 0),  0, "e( 1).e( 0)= 0", passed)
   call expect (conjg(eps(m,p, 0))*eps(m,p, 1),  0, "e( 0).e( 1)= 0", passed)
   call expect (conjg(eps(m,p, 0))*eps(m,p, 0), -1, "e( 0).e( 0)=-1", passed)
   call expect (conjg(eps(m,p, 0))*eps(m,p,-1),  0, "e( 0).e(-1)= 0", passed)
   call expect (conjg(eps(m,p,-1))*eps(m,p, 0),  0, "e(-1).e( 0)= 0", passed)
   call expect (                 p*eps(m,p, 0),  0, "    p.e( 0)= 0", passed)
end if
```

⟨*Test* omega95⟩+≡

```
print *, "*** Checking epsilon tensor: ***"
call expect (  pseudo_scalar(eps(m,p,1),eps(m,q,1),eps(m,p,0),eps(m,q,0)), &
             - pseudo_scalar(eps(m,q,1),eps(m,p,1),eps(m,p,0),eps(m,q,0)), "eps(1<->2)", passed)
call expect (  pseudo_scalar(eps(m,p,1),eps(m,q,1),eps(m,p,0),eps(m,q,0)), &
             - pseudo_scalar(eps(m,p,0),eps(m,q,1),eps(m,p,1),eps(m,q,0)), "eps(1<->3)", passed)
call expect (  pseudo_scalar(eps(m,p,1),eps(m,q,1),eps(m,p,0),eps(m,q,0)), &
             - pseudo_scalar(eps(m,q,0),eps(m,q,1),eps(m,p,0),eps(m,p,1)), "eps(1<->4)", passed)
call expect (  pseudo_scalar(eps(m,p,1),eps(m,q,1),eps(m,p,0),eps(m,q,0)), &
             - pseudo_scalar(eps(m,p,1),eps(m,p,0),eps(m,q,1),eps(m,q,0)), "eps(2<->3)", passed)
call expect (  pseudo_scalar(eps(m,p,1),eps(m,q,1),eps(m,p,0),eps(m,q,0)), &
             - pseudo_scalar(eps(m,p,1),eps(m,q,0),eps(m,p,0),eps(m,q,1)), "eps(2<->4)", passed)
call expect (  pseudo_scalar(eps(m,p,1),eps(m,q,1),eps(m,p,0),eps(m,q,0)), &
             - pseudo_scalar(eps(m,p,1),eps(m,q,1),eps(m,q,0),eps(m,p,0)), "eps(3<->4)", passed)
call expect (  pseudo_scalar(eps(m,p,1),eps(m,q,1),eps(m,p,0),eps(m,q,0)), &
               eps(m,p,1)*pseudo_vector(eps(m,q,1),eps(m,p,0),eps(m,q,0)), "eps'", passed)
```

$$\frac{1}{2}[x\wedge y]^*_{\mu\nu}[x\wedge y]^{\mu\nu} = \frac{1}{2}(x^*_\mu y^*_\nu - x^*_\nu y^*_\mu)(x^\mu y^\nu - x^\nu y^\mu) = (x^*x)(y^*y) - (x^*y)(y^*x) \tag{X.98}$$

⟨*Test* omega95⟩+≡

```
  print *, "*** Checking tensors: ***"
  call expect (conjg(p.wedge.q)*(p.wedge.q), (p*p)*(q*q)-(p*q)**2, &
       "[p,q].[q,p]=p.p*q.q-p.q^2", passed)
  call expect (conjg(p.wedge.q)*(q.wedge.p), (p*q)**2-(p*p)*(q*q), &
       "[p,q].[q,p]=p.q^2-p.p*q.q", passed)
```

i. e.

$$\frac{1}{2}[p\wedge\epsilon(p,i)]^*_{\mu\nu}[p\wedge\epsilon(p,j)]^{\mu\nu} = -p^2\delta_{ij} \tag{X.99}$$

⟨*Test* omega95⟩+≡

```
  call expect (conjg(p.wedge.eps(m,p, 1))*(p.wedge.eps(m,p, 1)), -p*p, &
       "[p,e( 1)].[p,e( 1)]=-p.p", passed)
  call expect (conjg(p.wedge.eps(m,p, 1))*(p.wedge.eps(m,p,-1)),    0, &
       "[p,e( 1)].[p,e(-1)]=0", passed)
  call expect (conjg(p.wedge.eps(m,p,-1))*(p.wedge.eps(m,p, 1)),    0, &
       "[p,e(-1)].[p,e( 1)]=0", passed)
  call expect (conjg(p.wedge.eps(m,p,-1))*(p.wedge.eps(m,p,-1)), -p*p, &
       "[p,e(-1)].[p,e(-1)]=-p.p", passed)
  if (m > 0) then
     call expect (conjg(p.wedge.eps(m,p, 1))*(p.wedge.eps(m,p, 0)),    0, &
          "[p,e( 1)].[p,e( 0)]=0", passed)
     call expect (conjg(p.wedge.eps(m,p, 0))*(p.wedge.eps(m,p, 1)),    0, &
          "[p,e( 0)].[p,e( 1)]=0", passed)
     call expect (conjg(p.wedge.eps(m,p, 0))*(p.wedge.eps(m,p, 0)), -p*p, &
          "[p,e( 0)].[p,e( 0)]=-p.p", passed)
     call expect (conjg(p.wedge.eps(m,p, 0))*(p.wedge.eps(m,p,-1)),    0, &
          "[p,e( 1)].[p,e(-1)]=0", passed)
     call expect (conjg(p.wedge.eps(m,p,-1))*(p.wedge.eps(m,p, 0)),    0, &
          "[p,e(-1)].[p,e( 0)]=0", passed)
  end if
```

also

$$[x\wedge y]_{\mu\nu}z^\nu = x_\mu(yz) - y_\mu(xz) \tag{X.100}$$
$$z_\mu[x\wedge y]^{\mu\nu} = (zx)y^\nu - (zy)x^\nu \tag{X.101}$$

⟨*Test* omega95⟩+≡

```
  call expect (abs ((p.wedge.eps(m,p, 1))*p + (p*p)*eps(m,p, 1)), 0, &
       "[p,e( 1)].p=-p.p*e( 1)]", passed)
  call expect (abs ((p.wedge.eps(m,p, 0))*p + (p*p)*eps(m,p, 0)), 0, &
       "[p,e( 0)].p=-p.p*e( 0)]", passed)
  call expect (abs ((p.wedge.eps(m,p,-1))*p + (p*p)*eps(m,p,-1)), 0, &
       "[p,e(-1)].p=-p.p*e(-1)]", passed)
  call expect (abs (p*(p.wedge.eps(m,p, 1)) - (p*p)*eps(m,p, 1)), 0, &
       "p.[p,e( 1)]=p.p*e( 1)]", passed)
  call expect (abs (p*(p.wedge.eps(m,p, 0)) - (p*p)*eps(m,p, 0)), 0, &
       "p.[p,e( 0)]=p.p*e( 0)]", passed)
  call expect (abs (p*(p.wedge.eps(m,p,-1)) - (p*p)*eps(m,p,-1)), 0, &
       "p.[p,e(-1)]=p.p*e(-1)]", passed)
```

⟨*Test* omega95⟩+≡
```
print *, "*** Checking polarisation tensors: ***"
call expect (conjg(eps2(m,p, 2))*eps2(m,p, 2), 1, "e2( 2).e2( 2)=1", passed)
call expect (conjg(eps2(m,p, 2))*eps2(m,p,-2), 0, "e2( 2).e2(-2)=0", passed)
call expect (conjg(eps2(m,p,-2))*eps2(m,p, 2), 0, "e2(-2).e2( 2)=0", passed)
call expect (conjg(eps2(m,p,-2))*eps2(m,p,-2), 1, "e2(-2).e2(-2)=1", passed)
if (m > 0) then
   call expect (conjg(eps2(m,p, 2))*eps2(m,p, 1), 0, "e2( 2).e2( 1)=0", passed)
   call expect (conjg(eps2(m,p, 2))*eps2(m,p, 0), 0, "e2( 2).e2( 0)=0", passed)
   call expect (conjg(eps2(m,p, 2))*eps2(m,p,-1), 0, "e2( 2).e2(-1)=0", passed)
   call expect (conjg(eps2(m,p, 1))*eps2(m,p, 2), 0, "e2( 1).e2( 2)=0", passed)
   call expect (conjg(eps2(m,p, 1))*eps2(m,p, 1), 1, "e2( 1).e2( 1)=1", passed)
   call expect (conjg(eps2(m,p, 1))*eps2(m,p, 0), 0, "e2( 1).e2( 0)=0", passed)
   call expect (conjg(eps2(m,p, 1))*eps2(m,p,-1), 0, "e2( 1).e2(-1)=0", passed)
   call expect (conjg(eps2(m,p, 1))*eps2(m,p,-2), 0, "e2( 1).e2(-2)=0", passed)
   call expect (conjg(eps2(m,p, 0))*eps2(m,p, 2), 0, "e2( 0).e2( 2)=0", passed)
   call expect (conjg(eps2(m,p, 0))*eps2(m,p, 1), 0, "e2( 0).e2( 1)=0", passed)
   call expect (conjg(eps2(m,p, 0))*eps2(m,p, 0), 1, "e2( 0).e2( 0)=1", passed)
   call expect (conjg(eps2(m,p, 0))*eps2(m,p,-1), 0, "e2( 0).e2(-1)=0", passed)
   call expect (conjg(eps2(m,p, 0))*eps2(m,p,-2), 0, "e2( 0).e2(-2)=0", passed)
   call expect (conjg(eps2(m,p,-1))*eps2(m,p, 2), 0, "e2(-1).e2( 2)=0", passed)
   call expect (conjg(eps2(m,p,-1))*eps2(m,p, 1), 0, "e2(-1).e2( 1)=0", passed)
   call expect (conjg(eps2(m,p,-1))*eps2(m,p, 0), 0, "e2(-1).e2( 0)=0", passed)
   call expect (conjg(eps2(m,p,-1))*eps2(m,p,-1), 1, "e2(-1).e2(-1)=1", passed)
   call expect (conjg(eps2(m,p,-1))*eps2(m,p,-2), 0, "e2(-1).e2(-2)=0", passed)
   call expect (conjg(eps2(m,p,-2))*eps2(m,p, 1), 0, "e2(-2).e2( 1)=0", passed)
   call expect (conjg(eps2(m,p,-2))*eps2(m,p, 0), 0, "e2(-2).e2( 0)=0", passed)
   call expect (conjg(eps2(m,p,-2))*eps2(m,p,-1), 0, "e2(-2).e2(-1)=0", passed)
end if
```

⟨*Test* omega95⟩+≡
```
call expect (          abs(p*eps2(m,p, 2)  ), 0, " |p.e2( 2)|  =0", passed)
call expect (            abs(eps2(m,p, 2)*p), 0, "   |e2( 2).p|=0", passed)
call expect (          abs(p*eps2(m,p,-2)  ), 0, " |p.e2(-2)|  =0", passed)
call expect (            abs(eps2(m,p,-2)*p), 0, "   |e2(-2).p|=0", passed)
if (m > 0) then
   call expect (          abs(p*eps2(m,p, 1)  ), 0, " |p.e2( 1)|  =0", passed)
   call expect (            abs(eps2(m,p, 1)*p), 0, "   |e2( 1).p|=0", passed)
   call expect (          abs(p*eps2(m,p, 0)  ), 0, " |p.e2( 0)|  =0", passed)
   call expect (            abs(eps2(m,p, 0)*p), 0, "   |e2( 0).p|=0", passed)
   call expect (          abs(p*eps2(m,p,-1)  ), 0, " |p.e2(-1)|  =0", passed)
   call expect (            abs(eps2(m,p,-1)*p), 0, "   |e2(-1).p|=0", passed)
end if
```

⟨*XXX Test* omega95⟩≡
```
print *, " *** Checking the polarization tensors for massive gravitons:"
call expect (abs(p * eps2(m,p,2)),  0, "p.e(+2)=0", passed)
call expect (abs(p * eps2(m,p,1)),  0, "p.e(+1)=0", passed)
call expect (abs(p * eps2(m,p,0)),  0, "p.e( 0)=0", passed)
call expect (abs(p * eps2(m,p,-1)), 0, "p.e(-1)=0", passed)
call expect (abs(p * eps2(m,p,-2)), 0, "p.e(-2)=0", passed)
call expect (abs(trace(eps2 (m,p,2))),  0, "Tr[e(+2)]=0", passed)
call expect (abs(trace(eps2 (m,p,1))),  0, "Tr[e(+1)]=0", passed)
call expect (abs(trace(eps2 (m,p,0))),  0, "Tr[e( 0)]=0", passed)
call expect (abs(trace(eps2 (m,p,-1))), 0, "Tr[e(-1)]=0", passed)
```

```
  call expect (abs(trace(eps2 (m,p,-2))), 0, "Tr[e(-2)]=0", passed)
  call expect (abs(eps2(m,p,2) * eps2(m,p,2)),   1, &
       "e(2).e(2)   = 1", passed)
  call expect (abs(eps2(m,p,2) * eps2(m,p,1)),   0, &
       "e(2).e(1)   = 0", passed)
  call expect (abs(eps2(m,p,2) * eps2(m,p,0)),   0, &
       "e(2).e(0)   = 0", passed)
  call expect (abs(eps2(m,p,2) * eps2(m,p,-1)),  0, &
       "e(2).e(-1)  = 0", passed)
  call expect (abs(eps2(m,p,2) * eps2(m,p,-2)),  0, &
       "e(2).e(-2)  = 0", passed)
  call expect (abs(eps2(m,p,1) * eps2(m,p,1)),   1, &
       "e(1).e(1)   = 1", passed)
  call expect (abs(eps2(m,p,1) * eps2(m,p,0)),   0, &
       "e(1).e(0)   = 0", passed)
  call expect (abs(eps2(m,p,1) * eps2(m,p,-1)),  0, &
       "e(1).e(-1)  = 0", passed)
  call expect (abs(eps2(m,p,1) * eps2(m,p,-2)),  0, &
       "e(1).e(-2)  = 0", passed)
  call expect (abs(eps2(m,p,0) * eps2(m,p,0)),   1, &
       "e(0).e(0)   = 1", passed)
  call expect (abs(eps2(m,p,0) * eps2(m,p,-1)),  0, &
       "e(0).e(-1)  = 0", passed)
  call expect (abs(eps2(m,p,0) * eps2(m,p,-2)),  0, &
       "e(0).e(-2)  = 0", passed)
  call expect (abs(eps2(m,p,-1) * eps2(m,p,-1)), 1, &
       "e(-1).e(-1) = 1", passed)
  call expect (abs(eps2(m,p,-1) * eps2(m,p,-2)), 0, &
       "e(-1).e(-2) = 0", passed)
  call expect (abs(eps2(m,p,-2) * eps2(m,p,-2)), 1, &
       "e(-2).e(-2) = 1", passed)
```

⟨*Test* omega95⟩+≡

```
  print *, " *** Checking the graviton propagator:"
  call expect (abs(p * (cmplx (p*p - m**2, m*w, kind=default) * &
               pr_tensor(p,m,w,eps2(m,p,-2)))), 0, "p.pr.e(-2)", passed)
  call expect (abs(p * (cmplx (p*p - m**2, m*w, kind=default) * &
               pr_tensor(p,m,w,eps2(m,p,-1)))), 0, "p.pr.e(-1)", passed)
  call expect (abs(p * (cmplx (p*p - m**2, m*w, kind=default) * &
               pr_tensor(p,m,w,eps2(m,p,0)))), 0, "p.pr.e(0)", passed)
  call expect (abs(p * (cmplx (p*p - m**2, m*w, kind=default) * &
               pr_tensor(p,m,w,eps2(m,p,1)))), 0, "p.pr.e(1)", passed)
  call expect (abs(p * (cmplx (p*p - m**2, m*w, kind=default) * &
               pr_tensor(p,m,w,eps2(m,p,2)))), 0, "p.pr.e(2)", passed)
  call expect (abs(p * (cmplx (p*p - m**2, m*w, kind=default) * &
               pr_tensor(p,m,w,ttest))), 0, "p.pr.ttest", passed)
```

⟨test_omega95_bispinors.f90⟩≡

```
  ⟨Copyleft⟩
  program test_omega95_bispinors
    use kinds
    use omega95_bispinors
    use omega_vspinor_polarizations
    use omega_testtools
    implicit none
```

```
  integer :: i, j
  real(kind=default) :: m, pabs, qabs, tabs, zabs, w
  real(kind=default), dimension(4) :: r
  complex(kind=default) :: c_one, c_two
  type(momentum) :: p, q, t, z, p_0
  type(vector) :: vp, vq, vt, vz
  type(vectorspinor) :: testv
  logical :: passed
  call random_seed ()
  c_one = 1
  c_two = 2
  w = 1.4142
  m = 13
  pabs = 42
  qabs = 137
  tabs = 84
  zabs = 3.1415
  p_0%t = m
  p_0%x = 0
  call random_momentum (p, pabs, m)
  call random_momentum (q, qabs, m)
  call random_momentum (t, tabs, m)
  call random_momentum (z, zabs, m)
  call random_number (r)
  do i = 1, 4
     testv%psi(1)%a(i) = (0.0_default, 0.0_default)
  end do
  do i = 2, 3
     do j = 1, 4
        testv%psi(i)%a(j) = cmplx (10.0_default * r(j))
    end do
  end do
  testv%psi(4)%a(1) = (1.0_default, 0.0_default)
  testv%psi(4)%a(2) = (0.0_default, 2.0_default)
  testv%psi(4)%a(3) = (1.0_default, 0.0_default)
  testv%psi(4)%a(4) = (3.0_default, 0.0_default)
  vp = p
  vq = q
  vt = t
  vz = z
  passed = .true.
  ⟨Test omega95_bispinors⟩
  if (.not. passed) then
    stop 1
  end if
end program test_omega95_bispinors
```

⟨*Test* omega95_bispinors⟩≡

```
print *, "*** Checking the equations of motion ***:"
call expect (abs(f_vf(c_one,vp,u(m,p,+1))-m*u(m,p,+1)), 0, "|[p-m]u(+)|=0", passed)
call expect (abs(f_vf(c_one,vp,u(m,p,-1))-m*u(m,p,-1)), 0, "|[p-m]u(-)|=0", passed)
call expect (abs(f_vf(c_one,vp,v(m,p,+1))+m*v(m,p,+1)), 0, "|[p+m]v(+)|=0", passed)
call expect (abs(f_vf(c_one,vp,v(m,p,-1))+m*v(m,p,-1)), 0, "|[p+m]v(-)|=0", passed)
print *, "*** Checking the equations of motion for negative masses***:"
call expect (abs(f_vf(c_one,vp,u(-m,p,+1))+m*u(-m,p,+1)), 0, "|[p+m]u(+)|=0", passed)
```

```
call expect (abs(f_vf(c_one,vp,u(-m,p,-1))+m*u(-m,p,-1)), 0, "|[p+m]u(-)|=0", passed)
call expect (abs(f_vf(c_one,vp,v(-m,p,+1))-m*v(-m,p,+1)), 0, "|[p-m]v(+)|=0", passed)
call expect (abs(f_vf(c_one,vp,v(-m,p,-1))-m*v(-m,p,-1)), 0, "|[p-m]v(-)|=0", passed)
```

⟨*Test* omega95_bispinors⟩+≡

```
print *, "*** Checking the normalization ***:"
call expect (s_ff(c_one,v(m,p,+1),u(m,p,+1)), +2*m, "ubar(+)*u(+)=+2m", passed)
call expect (s_ff(c_one,v(m,p,-1),u(m,p,-1)), +2*m, "ubar(-)*u(-)=+2m", passed)
call expect (s_ff(c_one,u(m,p,+1),v(m,p,+1)), -2*m, "vbar(+)*v(+)=-2m", passed)
call expect (s_ff(c_one,u(m,p,-1),v(m,p,-1)), -2*m, "vbar(-)*v(-)=-2m", passed)
call expect (s_ff(c_one,v(m,p,+1),v(m,p,+1)),    0, "ubar(+)*v(+)=0  ", passed)
call expect (s_ff(c_one,v(m,p,-1),v(m,p,-1)),    0, "ubar(-)*v(-)=0  ", passed)
call expect (s_ff(c_one,u(m,p,+1),u(m,p,+1)),    0, "vbar(+)*u(+)=0  ", passed)
call expect (s_ff(c_one,u(m,p,-1),u(m,p,-1)),    0, "vbar(-)*u(-)=0  ", passed)
print *, "*** Checking the normalization for negative masses***:"
call expect (s_ff(c_one,v(-m,p,+1),u(-m,p,+1)), -2*m, "ubar(+)*u(+)=-2m", passed)
call expect (s_ff(c_one,v(-m,p,-1),u(-m,p,-1)), -2*m, "ubar(-)*u(-)=-2m", passed)
call expect (s_ff(c_one,u(-m,p,+1),v(-m,p,+1)), +2*m, "vbar(+)*v(+)=+2m", passed)
call expect (s_ff(c_one,u(-m,p,-1),v(-m,p,-1)), +2*m, "vbar(-)*v(-)=+2m", passed)
call expect (s_ff(c_one,v(-m,p,+1),v(-m,p,+1)),    0, "ubar(+)*v(+)=0  ", passed)
call expect (s_ff(c_one,v(-m,p,-1),v(-m,p,-1)),    0, "ubar(-)*v(-)=0  ", passed)
call expect (s_ff(c_one,u(-m,p,+1),u(-m,p,+1)),    0, "vbar(+)*u(+)=0  ", passed)
call expect (s_ff(c_one,u(-m,p,-1),u(-m,p,-1)),    0, "vbar(-)*u(-)=0  ", passed)
```

⟨*Test* omega95_bispinors⟩+≡

```
print *, "*** Checking the currents ***:"
call expect (abs(v_ff(c_one,v(m,p,+1),u(m,p,+1))-2*vp), 0, "ubar(+).V.u(+)=2p", passed)
call expect (abs(v_ff(c_one,v(m,p,-1),u(m,p,-1))-2*vp), 0, "ubar(-).V.u(-)=2p", passed)
call expect (abs(v_ff(c_one,u(m,p,+1),v(m,p,+1))-2*vp), 0, "vbar(+).V.v(+)=2p", passed)
call expect (abs(v_ff(c_one,u(m,p,-1),v(m,p,-1))-2*vp), 0, "vbar(-).V.v(-)=2p", passed)
print *, "*** Checking the currents for negative masses***:"
call expect (abs(v_ff(c_one,v(-m,p,+1),u(-m,p,+1))-2*vp), 0, "ubar(+).V.u(+)=2p", passed)
call expect (abs(v_ff(c_one,v(-m,p,-1),u(-m,p,-1))-2*vp), 0, "ubar(-).V.u(-)=2p", passed)
call expect (abs(v_ff(c_one,u(-m,p,+1),v(-m,p,+1))-2*vp), 0, "vbar(+).V.v(+)=2p", passed)
call expect (abs(v_ff(c_one,u(-m,p,-1),v(-m,p,-1))-2*vp), 0, "vbar(-).V.v(-)=2p", passed)
```

⟨*Test* omega95_bispinors⟩+≡

```
print *, "*** Checking current conservation ***:"
call expect ((vp-vq)*v_ff(c_one,v(m,p,+1),u(m,q,+1)), 0, "d(ubar(+).V.u(+))=0", passed)
call expect ((vp-vq)*v_ff(c_one,v(m,p,-1),u(m,q,-1)), 0, "d(ubar(-).V.u(-))=0", passed)
call expect ((vp-vq)*v_ff(c_one,u(m,p,+1),v(m,q,+1)), 0, "d(vbar(+).V.v(+))=0", passed)
call expect ((vp-vq)*v_ff(c_one,u(m,p,-1),v(m,q,-1)), 0, "d(vbar(-).V.v(-))=0", passed)
```

⟨*Test* omega95_bispinors⟩+≡

```
print *, "*** Checking current conservation for negative masses***:"
call expect ((vp-vq)*v_ff(c_one,v(-m,p,+1),u(-m,q,+1)), 0, "d(ubar(+).V.u(+))=0", passed)
call expect ((vp-vq)*v_ff(c_one,v(-m,p,-1),u(-m,q,-1)), 0, "d(ubar(-).V.u(-))=0", passed)
call expect ((vp-vq)*v_ff(c_one,u(-m,p,+1),v(-m,q,+1)), 0, "d(vbar(+).V.v(+))=0", passed)
call expect ((vp-vq)*v_ff(c_one,u(-m,p,-1),v(-m,q,-1)), 0, "d(vbar(-).V.v(-))=0", passed)
```

⟨*Test* omega95_bispinors⟩+≡

```
if (m == 0) then
   print *, "*** Checking axial current conservation ***:"
   call expect ((vp-vq)*a_ff(c_one,v(m,p,+1),u(m,q,+1)), 0, "d(ubar(+).A.u(+))=0", passed)
   call expect ((vp-vq)*a_ff(c_one,v(m,p,-1),u(m,q,-1)), 0, "d(ubar(-).A.u(-))=0", passed)
   call expect ((vp-vq)*a_ff(c_one,u(m,p,+1),v(m,q,+1)), 0, "d(vbar(+).A.v(+))=0", passed)
   call expect ((vp-vq)*a_ff(c_one,u(m,p,-1),v(m,q,-1)), 0, "d(vbar(-).A.v(-))=0", passed)
```

```
  end if
```

⟨*Test* omega95_bispinors⟩+≡

```
  print *, "*** Checking polarization vectors: ***"
  call expect (conjg(eps(m,p, 1))*eps(m,p, 1), -1, "e( 1).e( 1)=-1", passed)
  call expect (conjg(eps(m,p, 1))*eps(m,p,-1),  0, "e( 1).e(-1)= 0", passed)
  call expect (conjg(eps(m,p,-1))*eps(m,p, 1),  0, "e(-1).e( 1)= 0", passed)
  call expect (conjg(eps(m,p,-1))*eps(m,p,-1), -1, "e(-1).e(-1)=-1", passed)
  call expect (                  p*eps(m,p, 1),  0, "    p.e( 1)= 0", passed)
  call expect (                  p*eps(m,p,-1),  0, "    p.e(-1)= 0", passed)
  if (m > 0) then
     call expect (conjg(eps(m,p, 1))*eps(m,p, 0),  0, "e( 1).e( 0)= 0", passed)
     call expect (conjg(eps(m,p, 0))*eps(m,p, 1),  0, "e( 0).e( 1)= 0", passed)
     call expect (conjg(eps(m,p, 0))*eps(m,p, 0), -1, "e( 0).e( 0)=-1", passed)
     call expect (conjg(eps(m,p, 0))*eps(m,p,-1),  0, "e( 0).e(-1)= 0", passed)
     call expect (conjg(eps(m,p,-1))*eps(m,p, 0),  0, "e(-1).e( 0)= 0", passed)
     call expect (                  p*eps(m,p, 0),  0, "    p.e( 0)= 0", passed)
  end if
```

⟨*Test* omega95_bispinors⟩+≡

```
  print *, "*** Checking polarization vectorspinors: ***"
  call expect (abs(p * ueps(m, p,  2)),  0, "p.ueps ( 2)= 0", passed)
  call expect (abs(p * ueps(m, p,  1)),  0, "p.ueps ( 1)= 0", passed)
  call expect (abs(p * ueps(m, p, -1)),  0, "p.ueps (-1)= 0", passed)
  call expect (abs(p * ueps(m, p, -2)),  0, "p.ueps (-2)= 0", passed)
  call expect (abs(p * veps(m, p,  2)),  0, "p.veps ( 2)= 0", passed)
  call expect (abs(p * veps(m, p,  1)),  0, "p.veps ( 1)= 0", passed)
  call expect (abs(p * veps(m, p, -1)),  0, "p.veps (-1)= 0", passed)
  call expect (abs(p * veps(m, p, -2)),  0, "p.veps (-2)= 0", passed)
  print *, "*** Checking polarization vectorspinors (neg. masses): ***"
  call expect (abs(p * ueps(-m, p,  2)),  0, "p.ueps ( 2)= 0", passed)
  call expect (abs(p * ueps(-m, p,  1)),  0, "p.ueps ( 1)= 0", passed)
  call expect (abs(p * ueps(-m, p, -1)),  0, "p.ueps (-1)= 0", passed)
  call expect (abs(p * ueps(-m, p, -2)),  0, "p.ueps (-2)= 0", passed)
  call expect (abs(p * veps(-m, p,  2)),  0, "p.veps ( 2)= 0", passed)
  call expect (abs(p * veps(-m, p,  1)),  0, "p.veps ( 1)= 0", passed)
  call expect (abs(p * veps(-m, p, -1)),  0, "p.veps (-1)= 0", passed)
  call expect (abs(p * veps(-m, p, -2)),  0, "p.veps (-2)= 0", passed)
  print *, "*** in the rest frame ***"
  call expect (abs(p_0 * ueps(m, p_0,  2)),  0, "p0.ueps ( 2)= 0", passed)
  call expect (abs(p_0 * ueps(m, p_0,  1)),  0, "p0.ueps ( 1)= 0", passed)
  call expect (abs(p_0 * ueps(m, p_0, -1)),  0, "p0.ueps (-1)= 0", passed)
  call expect (abs(p_0 * ueps(m, p_0, -2)),  0, "p0.ueps (-2)= 0", passed)
  call expect (abs(p_0 * veps(m, p_0,  2)),  0, "p0.veps ( 2)= 0", passed)
  call expect (abs(p_0 * veps(m, p_0,  1)),  0, "p0.veps ( 1)= 0", passed)
  call expect (abs(p_0 * veps(m, p_0, -1)),  0, "p0.veps (-1)= 0", passed)
  call expect (abs(p_0 * veps(m, p_0, -2)),  0, "p0.veps (-2)= 0", passed)
  print *, "*** in the rest frame (neg. masses) ***"
  call expect (abs(p_0 * ueps(-m, p_0,  2)),  0, "p0.ueps ( 2)= 0", passed)
  call expect (abs(p_0 * ueps(-m, p_0,  1)),  0, "p0.ueps ( 1)= 0", passed)
  call expect (abs(p_0 * ueps(-m, p_0, -1)),  0, "p0.ueps (-1)= 0", passed)
  call expect (abs(p_0 * ueps(-m, p_0, -2)),  0, "p0.ueps (-2)= 0", passed)
  call expect (abs(p_0 * veps(-m, p_0,  2)),  0, "p0.veps ( 2)= 0", passed)
  call expect (abs(p_0 * veps(-m, p_0,  1)),  0, "p0.veps ( 1)= 0", passed)
  call expect (abs(p_0 * veps(-m, p_0, -1)),  0, "p0.veps (-1)= 0", passed)
```

```
  call expect (abs(p_0 * veps(-m, p_0, -2)),  0, "p0.veps (-2)= 0", passed)
```

⟨*Test* omega95_bispinors⟩+≡

```
  print *, "*** Checking the irreducibility condition: ***"
  call expect (abs(f_potgr (c_one, c_one, ueps(m, p,  2))),  0, "g.ueps ( 2)", passed)
  call expect (abs(f_potgr (c_one, c_one, ueps(m, p,  1))),  0, "g.ueps ( 1)", passed)
  call expect (abs(f_potgr (c_one, c_one, ueps(m, p, -1))),  0, "g.ueps (-1)", passed)
  call expect (abs(f_potgr (c_one, c_one, ueps(m, p, -2))),  0, "g.ueps (-2)", passed)
  call expect (abs(f_potgr (c_one, c_one, veps(m, p,  2))),  0, "g.veps ( 2)", passed)
  call expect (abs(f_potgr (c_one, c_one, veps(m, p,  1))),  0, "g.veps ( 1)", passed)
  call expect (abs(f_potgr (c_one, c_one, veps(m, p, -1))),  0, "g.veps (-1)", passed)
  call expect (abs(f_potgr (c_one, c_one, veps(m, p, -2))),  0, "g.veps (-2)", passed)
  print *, "*** Checking the irreducibility condition (neg. masses): ***"
  call expect (abs(f_potgr (c_one, c_one, ueps(-m, p,  2))),  0, "g.ueps ( 2)", passed)
  call expect (abs(f_potgr (c_one, c_one, ueps(-m, p,  1))),  0, "g.ueps ( 1)", passed)
  call expect (abs(f_potgr (c_one, c_one, ueps(-m, p, -1))),  0, "g.ueps (-1)", passed)
  call expect (abs(f_potgr (c_one, c_one, ueps(-m, p, -2))),  0, "g.ueps (-2)", passed)
  call expect (abs(f_potgr (c_one, c_one, veps(-m, p,  2))),  0, "g.veps ( 2)", passed)
  call expect (abs(f_potgr (c_one, c_one, veps(-m, p,  1))),  0, "g.veps ( 1)", passed)
  call expect (abs(f_potgr (c_one, c_one, veps(-m, p, -1))),  0, "g.veps (-1)", passed)
  call expect (abs(f_potgr (c_one, c_one, veps(-m, p, -2))),  0, "g.veps (-2)", passed)
  print *, "*** in the rest frame ***"
  call expect (abs(f_potgr (c_one, c_one, ueps(m, p_0,  2))),  0, "g.ueps ( 2)", passed)
  call expect (abs(f_potgr (c_one, c_one, ueps(m, p_0,  1))),  0, "g.ueps ( 1)", passed)
  call expect (abs(f_potgr (c_one, c_one, ueps(m, p_0, -1))),  0, "g.ueps (-1)", passed)
  call expect (abs(f_potgr (c_one, c_one, ueps(m, p_0, -2))),  0, "g.ueps (-2)", passed)
  call expect (abs(f_potgr (c_one, c_one, veps(m, p_0,  2))),  0, "g.veps ( 2)", passed)
  call expect (abs(f_potgr (c_one, c_one, veps(m, p_0,  1))),  0, "g.veps ( 1)", passed)
  call expect (abs(f_potgr (c_one, c_one, veps(m, p_0, -1))),  0, "g.veps (-1)", passed)
  call expect (abs(f_potgr (c_one, c_one, veps(m, p_0, -2))),  0, "g.veps (-2)", passed)
  print *, "*** in the rest frame (neg. masses) ***"
  call expect (abs(f_potgr (c_one, c_one, ueps(m, p_0,  2))),  0, "g.ueps ( 2)", passed)
  call expect (abs(f_potgr (c_one, c_one, ueps(m, p_0,  1))),  0, "g.ueps ( 1)", passed)
  call expect (abs(f_potgr (c_one, c_one, ueps(m, p_0, -1))),  0, "g.ueps (-1)", passed)
  call expect (abs(f_potgr (c_one, c_one, ueps(m, p_0, -2))),  0, "g.ueps (-2)", passed)
  call expect (abs(f_potgr (c_one, c_one, veps(m, p_0,  2))),  0, "g.veps ( 2)", passed)
  call expect (abs(f_potgr (c_one, c_one, veps(m, p_0,  1))),  0, "g.veps ( 1)", passed)
  call expect (abs(f_potgr (c_one, c_one, veps(m, p_0, -1))),  0, "g.veps (-1)", passed)
  call expect (abs(f_potgr (c_one, c_one, veps(m, p_0, -2))),  0, "g.veps (-2)", passed)
```

⟨*Test* omega95_bispinors⟩+≡

```
  print *, "*** Testing vectorspinor normalization ***"
  call expect (veps(m,p, 2)*ueps(m,p, 2), -2*m, "ueps( 2).ueps( 2)= -2m", passed)
  call expect (veps(m,p, 1)*ueps(m,p, 1), -2*m, "ueps( 1).ueps( 1)= -2m", passed)
  call expect (veps(m,p,-1)*ueps(m,p,-1), -2*m, "ueps(-1).ueps(-1)= -2m", passed)
  call expect (veps(m,p,-2)*ueps(m,p,-2), -2*m, "ueps(-2).ueps(-2)= -2m", passed)
  call expect (ueps(m,p, 2)*veps(m,p, 2),  2*m, "veps( 2).veps( 2)= +2m", passed)
  call expect (ueps(m,p, 1)*veps(m,p, 1),  2*m, "veps( 1).veps( 1)= +2m", passed)
  call expect (ueps(m,p,-1)*veps(m,p,-1),  2*m, "veps(-1).veps(-1)= +2m", passed)
  call expect (ueps(m,p,-2)*veps(m,p,-2),  2*m, "veps(-2).veps(-2)= +2m", passed)
  call expect (ueps(m,p, 2)*ueps(m,p, 2),    0, "ueps( 2).veps( 2)=   0", passed)
  call expect (ueps(m,p, 1)*ueps(m,p, 1),    0, "ueps( 1).veps( 1)=   0", passed)
  call expect (ueps(m,p,-1)*ueps(m,p,-1),    0, "ueps(-1).veps(-1)=   0", passed)
  call expect (ueps(m,p,-2)*ueps(m,p,-2),    0, "ueps(-2).veps(-2)=   0", passed)
  call expect (veps(m,p, 2)*veps(m,p, 2),    0, "veps( 2).ueps( 2)=   0", passed)
```

```
call expect (veps(m,p, 1)*veps(m,p, 1),    0, "veps( 1).ueps( 1)=   0", passed)
call expect (veps(m,p,-1)*veps(m,p,-1),    0, "veps(-1).ueps(-1)=   0", passed)
call expect (veps(m,p,-2)*veps(m,p,-2),    0, "veps(-2).ueps(-2)=   0", passed)
print *, "*** Testing vectorspinor normalization (neg. masses) ***"
call expect (veps(-m,p, 2)*ueps(-m,p, 2), +2*m, "ueps( 2).ueps( 2)= +2m", passed)
call expect (veps(-m,p, 1)*ueps(-m,p, 1), +2*m, "ueps( 1).ueps( 1)= +2m", passed)
call expect (veps(-m,p,-1)*ueps(-m,p,-1), +2*m, "ueps(-1).ueps(-1)= +2m", passed)
call expect (veps(-m,p,-2)*ueps(-m,p,-2), +2*m, "ueps(-2).ueps(-2)= +2m", passed)
call expect (ueps(-m,p, 2)*veps(-m,p, 2), -2*m, "veps( 2).veps( 2)= -2m", passed)
call expect (ueps(-m,p, 1)*veps(-m,p, 1), -2*m, "veps( 1).veps( 1)= -2m", passed)
call expect (ueps(-m,p,-1)*veps(-m,p,-1), -2*m, "veps(-1).veps(-1)= -2m", passed)
call expect (ueps(-m,p,-2)*veps(-m,p,-2), -2*m, "veps(-2).veps(-2)= -2m", passed)
call expect (ueps(-m,p, 2)*ueps(-m,p, 2),    0, "ueps( 2).veps( 2)=   0", passed)
call expect (ueps(-m,p, 1)*ueps(-m,p, 1),    0, "ueps( 1).veps( 1)=   0", passed)
call expect (ueps(-m,p,-1)*ueps(-m,p,-1),    0, "ueps(-1).veps(-1)=   0", passed)
call expect (ueps(-m,p,-2)*ueps(-m,p,-2),    0, "ueps(-2).veps(-2)=   0", passed)
call expect (veps(-m,p, 2)*veps(-m,p, 2),    0, "veps( 2).ueps( 2)=   0", passed)
call expect (veps(-m,p, 1)*veps(-m,p, 1),    0, "veps( 1).ueps( 1)=   0", passed)
call expect (veps(-m,p,-1)*veps(-m,p,-1),    0, "veps(-1).ueps(-1)=   0", passed)
call expect (veps(-m,p,-2)*veps(-m,p,-2),    0, "veps(-2).ueps(-2)=   0", passed)
print *, "*** in the rest frame ***"
call expect (veps(m,p_0, 2)*ueps(m,p_0, 2), -2*m, "ueps( 2).ueps( 2)= -2m", passed)
call expect (veps(m,p_0, 1)*ueps(m,p_0, 1), -2*m, "ueps( 1).ueps( 1)= -2m", passed)
call expect (veps(m,p_0,-1)*ueps(m,p_0,-1), -2*m, "ueps(-1).ueps(-1)= -2m", passed)
call expect (veps(m,p_0,-2)*ueps(m,p_0,-2), -2*m, "ueps(-2).ueps(-2)= -2m", passed)
call expect (ueps(m,p_0, 2)*veps(m,p_0, 2),  2*m, "veps( 2).veps( 2)= +2m", passed)
call expect (ueps(m,p_0, 1)*veps(m,p_0, 1),  2*m, "veps( 1).veps( 1)= +2m", passed)
call expect (ueps(m,p_0,-1)*veps(m,p_0,-1),  2*m, "veps(-1).veps(-1)= +2m", passed)
call expect (ueps(m,p_0,-2)*veps(m,p_0,-2),  2*m, "veps(-2).veps(-2)= +2m", passed)
call expect (ueps(m,p_0, 2)*ueps(m,p_0, 2),    0, "ueps( 2).veps( 2)=   0", passed)
call expect (ueps(m,p_0, 1)*ueps(m,p_0, 1),    0, "ueps( 1).veps( 1)=   0", passed)
call expect (ueps(m,p_0,-1)*ueps(m,p_0,-1),    0, "ueps(-1).veps(-1)=   0", passed)
call expect (ueps(m,p_0,-2)*ueps(m,p_0,-2),    0, "ueps(-2).veps(-2)=   0", passed)
call expect (veps(m,p_0, 2)*veps(m,p_0, 2),    0, "veps( 2).ueps( 2)=   0", passed)
call expect (veps(m,p_0, 1)*veps(m,p_0, 1),    0, "veps( 1).ueps( 1)=   0", passed)
call expect (veps(m,p_0,-1)*veps(m,p_0,-1),    0, "veps(-1).ueps(-1)=   0", passed)
call expect (veps(m,p_0,-2)*veps(m,p_0,-2),    0, "veps(-2).ueps(-2)=   0", passed)
print *, "*** in the rest frame (neg. masses) ***"
call expect (veps(-m,p_0, 2)*ueps(-m,p_0, 2), +2*m, "ueps( 2).ueps( 2)= +2m", passed)
call expect (veps(-m,p_0, 1)*ueps(-m,p_0, 1), +2*m, "ueps( 1).ueps( 1)= +2m", passed)
call expect (veps(-m,p_0,-1)*ueps(-m,p_0,-1), +2*m, "ueps(-1).ueps(-1)= +2m", passed)
call expect (veps(-m,p_0,-2)*ueps(-m,p_0,-2), +2*m, "ueps(-2).ueps(-2)= +2m", passed)
call expect (ueps(-m,p_0, 2)*veps(-m,p_0, 2), -2*m, "veps( 2).veps( 2)= -2m", passed)
call expect (ueps(-m,p_0, 1)*veps(-m,p_0, 1), -2*m, "veps( 1).veps( 1)= -2m", passed)
call expect (ueps(-m,p_0,-1)*veps(-m,p_0,-1), -2*m, "veps(-1).veps(-1)= -2m", passed)
call expect (ueps(-m,p_0,-2)*veps(-m,p_0,-2), -2*m, "veps(-2).veps(-2)= -2m", passed)
call expect (ueps(-m,p_0, 2)*ueps(-m,p_0, 2),    0, "ueps( 2).veps( 2)=   0", passed)
call expect (ueps(-m,p_0, 1)*ueps(-m,p_0, 1),    0, "ueps( 1).veps( 1)=   0", passed)
call expect (ueps(-m,p_0,-1)*ueps(-m,p_0,-1),    0, "ueps(-1).veps(-1)=   0", passed)
call expect (ueps(-m,p_0,-2)*ueps(-m,p_0,-2),    0, "ueps(-2).veps(-2)=   0", passed)
call expect (veps(-m,p_0, 2)*veps(-m,p_0, 2),    0, "veps( 2).ueps( 2)=   0", passed)
call expect (veps(-m,p_0, 1)*veps(-m,p_0, 1),    0, "veps( 1).ueps( 1)=   0", passed)
call expect (veps(-m,p_0,-1)*veps(-m,p_0,-1),    0, "veps(-1).ueps(-1)=   0", passed)
call expect (veps(-m,p_0,-2)*veps(-m,p_0,-2),    0, "veps(-2).ueps(-2)=   0", passed)
```

⟨*Test* omega95_bispinors⟩+≡
```
  print *, "*** Majorana properties of gravitino vertices: ***"
  call expect (abs(u (m,q,1) * f_sgr (c_one, c_one, ueps(m,p,2), t) + &
     ueps(m,p,2) * gr_sf(c_one,c_one,u(m,q,1),t)),  0, "f_sgr     + gr_sf     = 0", passed)
  !!! call expect (abs(u (m,q,-1) * f_sgr (c_one, c_one, ueps(m,p,2), t) + &
  !!!     ueps(m,p,2) * gr_sf(c_one,c_one,u(m,q,-1),t)),  0, "f_sgr     + gr_sf     = 0", passed)
  !!! call expect (abs(u (m,q,1) * f_sgr (c_one, c_one, ueps(m,p,1), t) + &
  !!!     ueps(m,p,1) * gr_sf(c_one,c_one,u(m,q,1),t)),  0, "f_sgr     + gr_sf     = 0", passed)
  !!! call expect (abs(u (m,q,-1) * f_sgr (c_one, c_one, ueps(m,p,1), t) + &
  !!!     ueps(m,p,1) * gr_sf(c_one,c_one,u(m,q,-1),t)),  0, "f_sgr     + gr_sf     = 0", passed)
  !!! call expect (abs(u (m,q,1) * f_sgr (c_one, c_one, ueps(m,p,-1), t) + &
  !!!     ueps(m,p,-1) * gr_sf(c_one,c_one,u(m,q,1),t)),  0, "f_sgr   + gr_sf       = 0", passed)
  !!! call expect (abs(u (m,q,-1) * f_sgr (c_one, c_one, ueps(m,p,-1), t) + &
  !!!     ueps(m,p,-1) * gr_sf(c_one,c_one,u(m,q,-1),t)),  0, "f_sgr     + gr_sf     = 0", passed)
  !!! call expect (abs(u (m,q,1) * f_sgr (c_one, c_one, ueps(m,p,-2), t) + &
  !!!     ueps(m,p,-2) * gr_sf(c_one,c_one,u(m,q,1),t)),  0, "f_sgr     + gr_sf     = 0", passed)
  !!! call expect (abs(u (m,q,-1) * f_sgr (c_one, c_one, ueps(m,p,-2), t) + &
  !!!     ueps(m,p,-2) * gr_sf(c_one,c_one,u(m,q,-1),t)),  0, "f_sgr     + gr_sf     = 0", passed)
  call expect (abs(u (m,q,1) * f_slgr (c_one, c_one, ueps(m,p,2), t) + &
     ueps(m,p,2) * gr_slf(c_one,c_one,u(m,q,1),t)),  0, "f_slgr    + gr_slf    = 0", passed)
  call expect (abs(u (m,q,1) * f_srgr (c_one, c_one, ueps(m,p,2), t) + &
     ueps(m,p,2) * gr_srf(c_one,c_one,u(m,q,1),t)),  0, "f_srgr    + gr_srf    = 0", passed)
  call expect (abs(u (m,q,1) * f_slrgr (c_one, c_two, c_one, ueps(m,p,2), t) + &
     ueps(m,p,2) * gr_slrf(c_one,c_two,c_one,u(m,q,1),t)),  0, "f_slrgr   + gr_slrf   = 0", passed)
  call expect (abs(u (m,q,1) * f_pgr (c_one, c_one, ueps(m,p,2), t) + &
     ueps(m,p,2) * gr_pf(c_one,c_one,u(m,q,1),t)),  0, "f_pgr     + gr_pf     = 0", passed)
  call expect (abs(u (m,q,1) * f_vgr (c_one, vt, ueps(m,p,2), p+q) + &
     ueps(m,p,2) * gr_vf(c_one,vt,u(m,q,1),p+q)),  0, "f_vgr     + gr_vf     = 0", passed)
  call expect (abs(u (m,q,1) * f_vlrgr (c_one, c_two, vt, ueps(m,p,2), p+q) + &
     ueps(m,p,2) * gr_vlrf(c_one,c_two,vt,u(m,q,1),p+q)),  0, "f_vlrgr   + gr_vlrf   = 0", &
     passed, threshold = 0.5_default)
  !!! call expect (abs(u (m,q,-1) * f_vgr (c_one, vt, ueps(m,p,2), p+q) + &
  !!!     ueps(m,p,2) * gr_vf(c_one,vt,u(m,q,-1),p+q)),  0, "f_vgr     + gr_vf     = 0", passed)
  !!! call expect (abs(u (m,q,1) * f_vgr (c_one, vt, ueps(m,p,1), p+q) + &
  !!!     ueps(m,p,1) * gr_vf(c_one,vt,u(m,q,1),p+q)),  0, "f_vgr     + gr_vf     = 0", passed)
  !!! call expect (abs(u (m,q,-1) * f_vgr (c_one, vt, ueps(m,p,1), p+q) + &
  !!!     ueps(m,p,1) * gr_vf(c_one,vt,u(m,q,-1),p+q)),  0, "f_vgr     + gr_vf     = 0", passed)
  !!! call expect (abs(u (m,q,1) * f_vgr (c_one, vt, ueps(m,p,-1), p+q) + &
  !!!     ueps(m,p,-1) * gr_vf(c_one,vt,u(m,q,1),p+q)),  0, "f_vgr     + gr_vf     = 0", passed)
  !!! call expect (abs(u (m,q,-1) * f_vgr (c_one, vt, veps(m,p,-1), p+q) + &
  !!!     veps(m,p,-1) * gr_vf(c_one,vt,u(m,q,-1),p+q)),  0, "f_vgr     + gr_vf     = 0", passed)
  !!! call expect (abs(v (m,q,1) * f_vgr (c_one, vt, ueps(m,p,-2), p+q) + &
  !!!     ueps(m,p,-2) * gr_vf(c_one,vt,v(m,q,1),p+q)),  0, "f_vgr     + gr_vf     = 0", passed)
  !!! call expect (abs(u (m,q,-1) * f_vgr (c_one, vt, ueps(m,p,-2), p+q) + &
  !!!     ueps(m,p,-2) * gr_vf(c_one,vt,u(m,q,-1),p+q)),  0, "f_vgr     + gr_vf     = 0", passed)
  call expect (abs(s_grf (c_one, ueps(m,p,2), u(m,q,1),t) + &
     s_fgr(c_one,u(m,q,1),ueps(m,p,2),t)),  0, "s_grf     + s_fgr     = 0", passed)
  call expect (abs(sl_grf (c_one, ueps(m,p,2), u(m,q,1),t) + &
     sl_fgr(c_one,u(m,q,1),ueps(m,p,2),t)),  0, "sl_grf    + sl_fgr    = 0", passed)
  call expect (abs(sr_grf (c_one, ueps(m,p,2), u(m,q,1),t) + &
     sr_fgr(c_one,u(m,q,1),ueps(m,p,2),t)),  0, "sr_grf    + sr_fgr    = 0", passed)
  call expect (abs(slr_grf (c_one, c_two, ueps(m,p,2), u(m,q,1),t) + &
     slr_fgr(c_one,c_two,u(m,q,1),ueps(m,p,2),t)),  0, "slr_grf   + slr_fgr   = 0", passed)
  call expect (abs(p_grf (c_one, ueps(m,p,2), u(m,q,1),t) + &
```

```
    p_fgr(c_one,u(m,q,1),ueps(m,p,2),t)),  0, "p_grf     + p_fgr     = 0", passed)
  call expect (abs(v_grf (c_one, ueps(m,p,2), u(m,q,1),t) + &
    v_fgr(c_one,u(m,q,1),ueps(m,p,2),t)),  0, "v_grf     + v_fgr     = 0", passed)
  call expect (abs(vlr_grf (c_one, c_two, ueps(m,p,2), u(m,q,1),t) + &
    vlr_fgr(c_one,c_two,u(m,q,1),ueps(m,p,2),t)),  0, "vlr_grf   + vlr_fgr   = 0", passed)
  call expect (abs(u(m,p,1) * f_potgr (c_one,c_one,testv) - testv * gr_potf &
    (c_one,c_one,u (m,p,1))), 0, "f_potgr   - gr_potf   = 0", passed)
  call expect (abs (pot_fgr (c_one,u(m,p,1),testv) - pot_grf(c_one, &
    testv,u(m,p,1))), 0, "pot_fgr   - pot_grf   = 0", passed)
  call expect (abs(u(m,p,1) * f_s2gr (c_one,c_one,c_one,testv) - testv * gr_s2f &
    (c_one,c_one,c_one,u (m,p,1))), 0, "f_s2gr    - gr_s2f    = 0", passed)
  call expect (abs (s2_fgr (c_one,u(m,p,1),c_one,testv) - s2_grf(c_one, &
    testv,c_one,u(m,p,1))), 0, "s2_fgr    - s2_grf    = 0", passed)
  call expect (abs(u (m,q,1) * f_svgr (c_one, c_one, vt, ueps(m,p,2)) + &
    ueps(m,p,2) * gr_svf(c_one,c_one,vt,u(m,q,1))),  0, "f_svgr    + gr_svf    = 0", passed)
  call expect (abs(u (m,q,1) * f_slvgr (c_one, c_one, vt, ueps(m,p,2)) + &
    ueps(m,p,2) * gr_slvf(c_one,c_one,vt,u(m,q,1))),  0, "f_slvgr   + gr_slvf   = 0", passed)
  call expect (abs(u (m,q,1) * f_srvgr (c_one, c_one, vt, ueps(m,p,2)) + &
    ueps(m,p,2) * gr_srvf(c_one,c_one,vt,u(m,q,1))),  0, "f_srvgr   + gr_srvf   = 0", passed)
  call expect (abs(u (m,q,1) * f_slrvgr (c_one, c_two, c_one, vt, ueps(m,p,2)) + &
    ueps(m,p,2) * gr_slrvf(c_one,c_two,c_one,vt,u(m,q,1))),  0, "f_slrvgr  + gr_slrvf  = 0", passed
  call expect (abs (sv1_fgr (c_one,u(m,p,1),vt,ueps(m,q,2)) + sv1_grf(c_one, &
    ueps(m,q,2),vt,u(m,p,1))), 0, "sv1_fgr   + sv1_grf   = 0", passed)
  call expect (abs (sv2_fgr (c_one,u(m,p,1),c_one,ueps(m,q,2)) + sv2_grf(c_one, &
    ueps(m,q,2),c_one,u(m,p,1))), 0, "sv2_fgr   + sv2_grf   = 0", passed)
  call expect (abs (slv1_fgr (c_one,u(m,p,1),vt,ueps(m,q,2)) + slv1_grf(c_one, &
    ueps(m,q,2),vt,u(m,p,1))), 0, "slv1_fgr  + slv1_grf  = 0", passed)
  call expect (abs (srv2_fgr (c_one,u(m,p,1),c_one,ueps(m,q,2)) + srv2_grf(c_one, &
    ueps(m,q,2),c_one,u(m,p,1))), 0, "srv2_fgr  + srv2_grf  = 0", passed)
  call expect (abs (slrv1_fgr (c_one,c_two,u(m,p,1),vt,ueps(m,q,2)) + slrv1_grf(c_one,c_two, &
    ueps(m,q,2),vt,u(m,p,1))), 0, "slrv1_fgr + slrv1_grf = 0", passed)
  call expect (abs (slrv2_fgr (c_one,c_two,u(m,p,1),c_one,ueps(m,q,2)) + slrv2_grf(c_one, &
    c_two,ueps(m,q,2),c_one,u(m,p,1))), 0, "slrv2_fgr + slrv2_grf = 0", passed)
  call expect (abs(u (m,q,1) * f_pvgr (c_one, c_one, vt, ueps(m,p,2)) + &
    ueps(m,p,2) * gr_pvf(c_one,c_one,vt,u(m,q,1))),  0, "f_pvgr    + gr_pvf    = 0", passed)
  call expect (abs (pv1_fgr (c_one,u(m,p,1),vt,ueps(m,q,2)) + pv1_grf(c_one, &
    ueps(m,q,2),vt,u(m,p,1))), 0, "pv1_fgr   + pv1_grf   = 0", passed)
  call expect (abs (pv2_fgr (c_one,u(m,p,1),c_one,ueps(m,q,2)) + pv2_grf(c_one, &
    ueps(m,q,2),c_one,u(m,p,1))), 0, "pv2_fgr   + pv2_grf   = 0", passed)
  call expect (abs(u (m,q,1) * f_v2gr (c_one, vt, vz, ueps(m,p,2)) + &
    ueps(m,p,2) * gr_v2f(c_one,vt,vz,u(m,q,1))),  0, "f_v2gr    + gr_v2f    = 0", passed)
  call expect (abs(u (m,q,1) * f_v2lrgr (c_one, c_two, vt, vz, ueps(m,p,2)) + &
    ueps(m,p,2) * gr_v2lrf(c_one,c_two,vt,vz,u(m,q,1))),  0, "f_v2lrgr  + gr_v2lrf  = 0", passed)
  call expect (abs (v2_fgr (c_one,u(m,p,1),vt,ueps(m,q,2)) + v2_grf(c_one, &
    ueps(m,q,2),vt,u(m,p,1))), 0, "v2_fgr    + v2_grf    = 0", passed)
  call expect (abs (v2lr_fgr (c_one,c_two,u(m,p,1),vt,ueps(m,q,2)) + v2lr_grf(c_one, c_two, &
    ueps(m,q,2),vt,u(m,p,1))), 0, "v2lr_fgr  + v2lr_grf  = 0", passed)
```

⟨*Test* omega95_bispinors⟩+≡

```
  print *, "*** Testing the gravitino propagator: ***"
  print *, "Transversality:"
  call expect (abs(p * (cmplx (p*p - m**2, m*w, kind=default) * &
              pr_grav(p,m,w,testv))), 0, "p.pr.test", passed)
  call expect (abs(p * (cmplx (p*p - m**2, m*w, kind=default) * &
              pr_grav(p,m,w,ueps(m,p,2)))),  0, "p.pr.ueps ( 2)", passed)
```

```
  call expect (abs(p * (cmplx (p*p - m**2, m*w, kind=default) * &
               pr_grav(p,m,w,ueps(m,p,1)))),  0, "p.pr.ueps ( 1)", passed)
  call expect (abs(p * (cmplx (p*p - m**2, m*w, kind=default) * &
               pr_grav(p,m,w,ueps(m,p,-1)))), 0, "p.pr.ueps (-1)", passed)
  call expect (abs(p * (cmplx (p*p - m**2, m*w, kind=default) * &
               pr_grav(p,m,w,ueps(m,p,-2)))), 0, "p.pr.ueps (-2)", passed)
  call expect (abs(p * (cmplx (p*p - m**2, m*w, kind=default) * &
               pr_grav(p,m,w,veps(m,p,2)))),  0, "p.pr.veps ( 2)", passed)
  call expect (abs(p * (cmplx (p*p - m**2, m*w, kind=default) * &
               pr_grav(p,m,w,veps(m,p,1)))),  0, "p.pr.veps ( 1)", passed)
  call expect (abs(p * (cmplx (p*p - m**2, m*w, kind=default) * &
               pr_grav(p,m,w,veps(m,p,-1)))), 0, "p.pr.veps (-1)", passed)
  call expect (abs(p * (cmplx (p*p - m**2, m*w, kind=default) * &
               pr_grav(p,m,w,veps(m,p,-2)))), 0, "p.pr.veps (-2)", passed)
  print *, "Irreducibility:"
  call expect (abs(f_potgr (c_one, c_one, (cmplx (p*p - m**2, m*w, &
               kind=default) * pr_grav(p,m,w,testv)))), 0, "g.pr.test", passed)
  call expect (abs(f_potgr (c_one, c_one, (cmplx (p*p - m**2, m*w, &
               kind=default) * pr_grav(p,m,w,ueps(m,p,2))))), 0, &
               "g.pr.ueps ( 2)", passed)
  call expect (abs(f_potgr (c_one, c_one, (cmplx (p*p - m**2, m*w, &
               kind=default) * pr_grav(p,m,w,ueps(m,p,1))))), 0, &
               "g.pr.ueps ( 1)", passed)
  call expect (abs(f_potgr (c_one, c_one, (cmplx (p*p - m**2, m*w, &
               kind=default) * pr_grav(p,m,w,ueps(m,p,-1))))), 0, &
               "g.pr.ueps (-1)", passed)
  call expect (abs(f_potgr (c_one, c_one, (cmplx (p*p - m**2, m*w, &
               kind=default) * pr_grav(p,m,w,ueps(m,p,-2))))), 0, &
               "g.pr.ueps (-2)", passed)
  call expect (abs(f_potgr (c_one, c_one, (cmplx (p*p - m**2, m*w, &
               kind=default) * pr_grav(p,m,w,veps(m,p,2))))), 0, &
               "g.pr.veps ( 2)", passed)
  call expect (abs(f_potgr (c_one, c_one, (cmplx (p*p - m**2, m*w, &
               kind=default) * pr_grav(p,m,w,veps(m,p,1))))), 0, &
               "g.pr.veps ( 1)", passed)
  call expect (abs(f_potgr (c_one, c_one, (cmplx (p*p - m**2, m*w, &
               kind=default) * pr_grav(p,m,w,veps(m,p,-1))))), 0, &
               "g.pr.veps (-1)", passed)
  call expect (abs(f_potgr (c_one, c_one, (cmplx (p*p - m**2, m*w, &
               kind=default) * pr_grav(p,m,w,veps(m,p,-2))))), 0, &
               "g.pr.veps (-2)", passed)
```

```
⟨omega_bundle.f90⟩≡
  ⟨omega_vectors.f90⟩
  ⟨omega_spinors.f90⟩
  ⟨omega_bispinors.f90⟩
  ⟨omega_vectorspinors.f90⟩
  ⟨omega_polarizations.f90⟩
  ⟨omega_tensors.f90⟩
  ⟨omega_tensor_polarizations.f90⟩
  ⟨omega_couplings.f90⟩
  ⟨omega_spinor_couplings.f90⟩
  ⟨omega_bispinor_couplings.f90⟩
  ⟨omega_vspinor_polarizations.f90⟩
  ⟨omega_utils.f90⟩
```

```
  ⟨omega95.f90⟩
  ⟨omega95_bispinors.f90⟩
  ⟨omega_parameters.f90⟩
  ⟨omega_parameters_madgraph.f90⟩
```

```
⟨omega_bundle_whizard.f90⟩≡
  ⟨omega_bundle.f90⟩
  ⟨omega_parameters_whizard.f90⟩
```

X.21 O'Mega Virtual Machine

This module defines the O'Mega Virtual Machine (OVM) completely, whereby all environmental dependencies like masses, widths and couplings have to be given to the constructor `vm%init` at runtime.

Support for Majorana particles and vectorspinors is only partially, especially all fusions are missing. Maybe it would be easier to make an additional `omegavm95_bispinors` to avoid namespace issues. Non-type specific chunks could be reused

```
⟨omegavm95.f90⟩≡
  ⟨Copyleft⟩
  module omegavm95
    use kinds, only: default
    use constants
    use iso_varying_string, string_t => varying_string
    use, intrinsic :: iso_fortran_env, only : input_unit, output_unit, error_unit
    use omega95
    use omega95_bispinors, only: bispinor, vectorspinor, veps, pr_grav
    use omega95_bispinors, only: bi_u => u
    use omega95_bispinors, only: bi_v => v
    use omega95_bispinors, only: bi_pr_psi => pr_psi
    use omega95_bispinors, only: operator (*), operator (+)
    use omega_color, only: ovm_color_sum, OCF => omega_color_factor
    implicit none
    private
    ⟨Utilities Declarations⟩
    ⟨OVM Data Declarations⟩
    ⟨OVM Instructions⟩
  contains
    ⟨OVM Procedure Implementations⟩
    ⟨Utilities Procedure Implementations⟩
  end module omegavm95
```

This might not be the proper place but I don't know where to put it

```
⟨Utilities Declarations⟩≡
  integer, parameter, public :: stdin = input_unit
  integer, parameter, public :: stdout = output_unit
  integer, parameter, public :: stderr = error_unit
  integer, parameter :: MIN_UNIT = 11, MAX_UNIT = 99
```

```
⟨OVM Procedure Implementations⟩≡
  subroutine find_free_unit (u, iostat)
    integer, intent(out) :: u
    integer, intent(out), optional :: iostat
    logical :: exists, is_open
```

```
    integer :: i, status
    do i = MIN_UNIT, MAX_UNIT
       inquire (unit = i, exist = exists, opened = is_open, &
            iostat = status)
       if (status == 0) then
          if (exists .and. .not. is_open) then
             u = i
             if (present (iostat)) then
                iostat = 0
             end if
             return
          end if
       end if
    end do
    if (present (iostat)) then
       iostat = -1
    end if
    u = -1
  end subroutine find_free_unit
```

These abstract data types would ideally be the interface to communicate quantum numbers between O'Mega and Whizard. This gives full flexibility to change the representation at any time

⟨*Utilities Declarations*⟩+≡

```
  public :: color_t
  type color_t
  contains
    procedure :: write => color_write
  end type color_t

  public :: col_discrete
  type, extends(color_t) :: col_discrete
    integer :: i
  end type col_discrete

  public :: flavor_t
  type flavor_t
  contains
    procedure :: write => flavor_write
  end type flavor_t

  public :: flv_discrete
  type, extends(flavor_t) :: flv_discrete
    integer :: i
  end type flv_discrete

  public :: helicity_t
  type :: helicity_t
  contains
    procedure :: write => helicity_write
  end type helicity_t

  public :: hel_discrete
  type, extends(helicity_t) :: hel_discrete
```

```
    integer :: i
  end type hel_discrete

  public :: hel_trigonometric
  type, extends(helicity_t) :: hel_trigonometric
    real :: theta
  end type hel_trigonometric

  public :: hel_exponential
  type, extends(helicity_t) :: hel_exponential
    real :: phi
  end type hel_exponential

  public :: hel_spherical
  type, extends(helicity_t) :: hel_spherical
    real :: theta, phi
  end type hel_spherical
```

⟨*Utilities Procedure Implementations*⟩≡

```
  subroutine color_write (color, fh)
    class(color_t), intent(in) :: color
    integer, intent(in) :: fh
    select type(color)
    type is (col_discrete)
       write(fh, *) 'color_discrete%i           = ', color%i
    end select
  end subroutine color_write

  subroutine helicity_write (helicity, fh)
    class(helicity_t), intent(in) :: helicity
    integer, intent(in) :: fh
    select type(helicity)
    type is (hel_discrete)
       write(fh, *) 'helicity_discrete%i           = ', helicity%i
    type is (hel_trigonometric)
       write(fh, *) 'helicity_trigonometric%theta  = ', helicity%theta
    type is (hel_exponential)
       write(fh, *) 'helicity_exponential%phi      = ', helicity%phi
    type is (hel_spherical)
       write(fh, *) 'helicity_spherical%phi        = ', helicity%phi
       write(fh, *) 'helicity_spherical%theta      = ', helicity%theta
    end select
  end subroutine helicity_write

  subroutine flavor_write (flavor, fh)
    class(flavor_t), intent(in) :: flavor
    integer, intent(in) :: fh
    select type(flavor)
    type is (flv_discrete)
       write(fh, *) 'flavor_discrete%i           = ', flavor%i
    end select
  end subroutine flavor_write
```

X.21.1 Memory Layout

Some internal parameters

⟨*OVM Data Declarations*⟩≡
```
integer, parameter :: len_instructions = 8
integer, parameter :: N_version_lines = 2
! Comment lines including the first header description line
integer, parameter :: N_comments = 6
! Actual data lines plus intermediate description lines
! 'description \n 1 2 3 \n description \n 3 2 1' would count as 3
integer, parameter :: N_header_lines = 5
real(default), parameter, public :: N_ = three
```

This is the basic type of a VM

⟨*OVM Data Declarations*⟩+≡
```
type :: basic_vm_t
   private
   logical :: verbose
   type(string_t) :: bytecode_file
   integer :: bytecode_fh, out_fh
   integer :: N_instructions, N_levels
   integer :: N_table_lines
   integer, dimension(:, :), allocatable :: instructions
   integer, dimension(:), allocatable :: levels
end type
```

To allow for a lazy evaluation of amplitudes, we have to keep track whether a wave function has already been computed, to avoid multiple-computing that would arise when the bytecode has redundant fusions, which is necessary for flavor and color MC (and helicity MC when we use Weyl-van-der-Waerden-spinors)

⟨*OVM Data Declarations*⟩+≡
```
type :: vm_scalar
   logical :: c
   complex(kind=default) :: v
end type

type :: vm_spinor
   logical :: c
   type(spinor) :: v
end type

type :: vm_conjspinor
   logical :: c
   type(conjspinor) :: v
end type

type :: vm_bispinor
   logical :: c
   type(bispinor) :: v
end type

type :: vm_vector
```

```
    logical :: c
    type(vector) :: v
  end type

  type :: vm_tensor_2
    logical :: c
    type(tensor) :: v
  end type

  type :: vm_tensor_1
    logical :: c
    type(tensor2odd) :: v
  end type

  type :: vm_vectorspinor
    logical :: c
    type(vectorspinor) :: v
  end type
```

We need a memory pool for all the intermediate results

⟨*OVM Data Declarations*⟩+≡

```
  type, public, extends (basic_vm_t) :: vm_t
    private
    type(string_t) :: version
    type(string_t) :: model
    integer :: N_momenta, N_particles, N_prt_in, N_prt_out, N_amplitudes
    ! helicities = helicity combinations
    integer :: N_helicities, N_col_flows, N_col_indices, N_flavors, N_col_factors

    integer :: N_scalars, N_spinors, N_conjspinors, N_bispinors
    integer :: N_vectors, N_tensors_2, N_tensors_1, N_vectorspinors

    integer :: N_coupl_real, N_coupl_real2, N_coupl_cmplx, N_coupl_cmplx2

    integer, dimension(:, :), allocatable :: table_flavor
    integer, dimension(:, :, :), allocatable :: table_color_flows
    integer, dimension(:, :), allocatable :: table_spin
    logical, dimension(:, :), allocatable :: table_ghost_flags
    type(OCF), dimension(:), allocatable :: table_color_factors
    logical, dimension(:, :), allocatable :: table_flv_col_is_allowed

    real(default), dimension(:), allocatable :: coupl_real
    real(default), dimension(:, :), allocatable :: coupl_real2
    complex(default), dimension(:), allocatable :: coupl_cmplx
    complex(default), dimension(:, :), allocatable :: coupl_cmplx2
    real(default), dimension(:), allocatable :: mass
    real(default), dimension(:), allocatable :: width

    type(momentum), dimension(:), allocatable :: momenta
    complex(default), dimension(:), allocatable :: amplitudes
    complex(default), dimension(:, :, :), allocatable :: table_amplitudes
    class(flavor_t), dimension(:), allocatable :: flavor
    class(color_t), dimension(:), allocatable :: color
    ! gfortran 4.7
```

```
    !class(helicity_t), dimension(:), pointer :: helicity => null()
    integer, dimension(:), allocatable :: helicity

    type(vm_scalar), dimension(:), allocatable :: scalars
    type(vm_spinor), dimension(:), allocatable :: spinors
    type(vm_conjspinor), dimension(:), allocatable :: conjspinors
    type(vm_bispinor), dimension(:), allocatable :: bispinors
    type(vm_vector), dimension(:), allocatable :: vectors
    type(vm_tensor_2), dimension(:), allocatable :: tensors_2
    type(vm_tensor_1), dimension(:), allocatable :: tensors_1
    type(vm_vectorspinor), dimension(:), allocatable :: vectorspinors

    logical, dimension(:), allocatable :: hel_is_allowed
    real(default), dimension(:), allocatable :: hel_max_abs
    real(default) :: hel_sum_abs = 0, hel_threshold = 1E10
    integer :: hel_count = 0, hel_cutoff = 100
    integer, dimension(:), allocatable :: hel_map
    integer :: hel_finite

    logical :: openmp

  contains
   ⟨VM: TBP⟩
  end type
```

⟨*OVM Procedure Implementations*⟩+≡

```
  subroutine alloc_arrays (vm)
    type(vm_t), intent(inout) :: vm
    integer :: i
    allocate (vm%table_flavor(vm%N_particles, vm%N_flavors))
    allocate (vm%table_color_flows(vm%N_col_indices, vm%N_particles, &
                                   vm%N_col_flows))
    allocate (vm%table_spin(vm%N_particles, vm%N_helicities))
    allocate (vm%table_ghost_flags(vm%N_particles, vm%N_col_flows))
    allocate (vm%table_color_factors(vm%N_col_factors))
    allocate (vm%table_flv_col_is_allowed(vm%N_flavors, vm%N_col_flows))
    allocate (vm%momenta(vm%N_momenta))
    allocate (vm%amplitudes(vm%N_amplitudes))
    allocate (vm%table_amplitudes(vm%N_flavors, vm%N_col_flows, &
                                  vm%N_helicities))
    vm%table_amplitudes = zero
    allocate (vm%scalars(vm%N_scalars))
    allocate (vm%spinors(vm%N_spinors))
    allocate (vm%conjspinors(vm%N_conjspinors))
    allocate (vm%bispinors(vm%N_bispinors))
    allocate (vm%vectors(vm%N_vectors))
    allocate (vm%tensors_2(vm%N_tensors_2))
    allocate (vm%tensors_1(vm%N_tensors_1))
    allocate (vm%vectorspinors(vm%N_vectorspinors))
    allocate (vm%hel_is_allowed(vm%N_helicities))
    vm%hel_is_allowed = .True.
    allocate (vm%hel_max_abs(vm%N_helicities))
    vm%hel_max_abs = 0
    allocate (vm%hel_map(vm%N_helicities))
```

```
    vm%hel_map = (/(i, i = 1, vm%N_helicities)/)
    vm%hel_finite = vm%N_helicities
  end subroutine alloc_arrays
```

X.21.2 Controlling the VM

These type-bound procedures steer the VM

⟨*VM: TBP*⟩≡

```
    procedure :: init => vm_init
    procedure :: write => vm_write
    procedure :: reset => vm_reset
    procedure :: run => vm_run
    procedure :: final => vm_final
```

The `init` completely sets the environment for the OVM. Parameters can be changed with `reset` without reloading the bytecode.

⟨*OVM Procedure Implementations*⟩+≡

```
  subroutine vm_init (vm, bytecode_file, version, model, &
      coupl_real, coupl_real2, coupl_cmplx, coupl_cmplx2, &
      mass, width, verbose, out_fh, openmp)
    class(vm_t), intent(out) :: vm
    type(string_t), intent(in) :: bytecode_file
    type(string_t), intent(in) :: version
    type(string_t), intent(in) :: model
    real(default), dimension(:), optional, intent(in) :: coupl_real
    real(default), dimension(:, :), optional, intent(in) :: coupl_real2
    complex(default), dimension(:), optional, intent(in) :: coupl_cmplx
    complex(default), dimension(:, :), optional, intent(in) :: coupl_cmplx2
    real(default), dimension(:), optional, intent(in) :: mass
    real(default), dimension(:), optional, intent(in) :: width
    logical, optional, intent(in) :: verbose
    integer, optional, intent(in) :: out_fh
    logical, optional, intent(in) :: openmp
    vm%bytecode_file = bytecode_file
    vm%version = version
    vm%model = model
    if (present (coupl_real)) then
       allocate (vm%coupl_real (size (coupl_real)), source=coupl_real)
    end if
    if (present (coupl_real2)) then
       allocate (vm%coupl_real2 (2, size (coupl_real2, 2)), source=coupl_real2)
    end if
    if (present (coupl_cmplx)) then
       allocate (vm%coupl_cmplx (size (coupl_cmplx)), source=coupl_cmplx)
    end if
    if (present (coupl_cmplx2)) then
       allocate (vm%coupl_cmplx2 (2, size (coupl_cmplx2, 2)), &
                 source=coupl_cmplx2)
    end if
    if (present (mass)) then
       allocate (vm%mass(size(mass)), source=mass)
    end if
    if (present (width)) then
```

```
      allocate (vm%width(size (width)), source=width)
   end if
   if (present (openmp)) then
      vm%openmp = openmp
   else
      vm%openmp = .false.
   end if

   call basic_init (vm, verbose, out_fh)
 end subroutine vm_init
```

⟨*OVM Procedure Implementations*⟩+≡

```
 subroutine vm_reset (vm, &
     coupl_real, coupl_real2, coupl_cmplx, coupl_cmplx2, &
     mass, width, verbose, out_fh)
   class(vm_t), intent(inout) :: vm
   real(default), dimension(:), optional, intent(in) :: coupl_real
   real(default), dimension(:, :), optional, intent(in) :: coupl_real2
   complex(default), dimension(:), optional, intent(in) :: coupl_cmplx
   complex(default), dimension(:, :), optional, intent(in) :: coupl_cmplx2
   real(default), dimension(:), optional, intent(in) :: mass
   real(default), dimension(:), optional, intent(in) :: width
   logical, optional, intent(in) :: verbose
   integer, optional, intent(in) :: out_fh
   if (present (coupl_real)) then
      vm%coupl_real = coupl_real
   end if
   if (present (coupl_real2)) then
      vm%coupl_real2 = coupl_real2
   end if
   if (present (coupl_cmplx)) then
      vm%coupl_cmplx = coupl_cmplx
   end if
   if (present (coupl_cmplx2)) then
      vm%coupl_cmplx2 = coupl_cmplx2
   end if
   if (present (mass)) then
      vm%mass = mass
   end if
   if (present (width)) then
      vm%width = width
   end if
   if (present (verbose)) then
      vm%verbose = verbose
   end if
   if (present (out_fh)) then
      vm%out_fh = out_fh
   end if
 end subroutine vm_reset
```

Mainly for debugging

⟨*OVM Procedure Implementations*⟩+≡

```
 subroutine vm_write (vm)
```

```
class(vm_t), intent(in) :: vm
integer :: i, j, k
call basic_write (vm)
write(vm%out_fh, *) 'table_flavor              = ', vm%table_flavor
write(vm%out_fh, *) 'table_color_flows         = ', vm%table_color_flows
write(vm%out_fh, *) 'table_spin                = ', vm%table_spin
write(vm%out_fh, *) 'table_ghost_flags         = ', vm%table_ghost_flags
write(vm%out_fh, *) 'table_color_factors       = ', vm%table_color_factors
write(vm%out_fh, *) 'table_flv_col_is_allowed  = ', &
                  vm%table_flv_col_is_allowed
do i = 1, vm%N_flavors
   do j = 1, vm%N_col_flows
      do k = 1, vm%N_helicities
         write(vm%out_fh, *) 'table_amplitudes(f,c,h), f, c, h = ', vm%table_amplitudes(i,j,k),
      end do
   end do
end do
if (allocated(vm%coupl_real)) then
   write(vm%out_fh, *) 'coupl_real          = ', vm%coupl_real
end if
if (allocated(vm%coupl_real2)) then
   write(vm%out_fh, *) 'coupl_real2         = ', vm%coupl_real2
end if
if (allocated(vm%coupl_cmplx)) then
   write(vm%out_fh, *) 'coupl_cmplx         = ', vm%coupl_cmplx
end if
if (allocated(vm%coupl_cmplx2)) then
   write(vm%out_fh, *) 'coupl_cmplx2        = ', vm%coupl_cmplx2
end if
write(vm%out_fh, *) 'mass                = ', vm%mass
write(vm%out_fh, *) 'width               = ', vm%width
write(vm%out_fh, *) 'momenta             = ', vm%momenta
! gfortran 4.7
!do i = 1, size(vm%flavor)
   !call vm%flavor(i)%write (vm%out_fh)
!end do
!do i = 1, size(vm%color)
   !call vm%color(i)%write (vm%out_fh)
!end do
!do i = 1, size(vm%helicity)
   !call vm%helicity(i)%write (vm%out_fh)
!end do
write(vm%out_fh, *) 'helicity            = ', vm%helicity
write(vm%out_fh, *) 'amplitudes       = ', vm%amplitudes
write(vm%out_fh, *) 'scalars       = ', vm%scalars
write(vm%out_fh, *) 'spinors       = ', vm%spinors
write(vm%out_fh, *) 'conjspinors   = ', vm%conjspinors
write(vm%out_fh, *) 'bispinors     = ', vm%bispinors
write(vm%out_fh, *) 'vectors       = ', vm%vectors
write(vm%out_fh, *) 'tensors_2     = ', vm%tensors_2
write(vm%out_fh, *) 'tensors_1     = ', vm%tensors_1
write(vm%out_fh, *) 'vectorspinors = ', vm%vectorspinors
write(vm%out_fh, *) 'N_momenta       = ', vm%N_momenta
write(vm%out_fh, *) 'N_particles     = ', vm%N_particles
```

```
    write(vm%out_fh, *) 'N_prt_in        = ', vm%N_prt_in
    write(vm%out_fh, *) 'N_prt_out       = ', vm%N_prt_out
    write(vm%out_fh, *) 'N_amplitudes    = ', vm%N_amplitudes
    write(vm%out_fh, *) 'N_helicities    = ', vm%N_helicities
    write(vm%out_fh, *) 'N_col_flows     = ', vm%N_col_flows
    write(vm%out_fh, *) 'N_col_indices   = ', vm%N_col_indices
    write(vm%out_fh, *) 'N_flavors       = ', vm%N_flavors
    write(vm%out_fh, *) 'N_col_factors   = ', vm%N_col_factors
    write(vm%out_fh, *) 'N_scalars       = ', vm%N_scalars
    write(vm%out_fh, *) 'N_spinors       = ', vm%N_spinors
    write(vm%out_fh, *) 'N_conjspinors   = ', vm%N_conjspinors
    write(vm%out_fh, *) 'N_bispinors     = ', vm%N_bispinors
    write(vm%out_fh, *) 'N_vectors       = ', vm%N_vectors
    write(vm%out_fh, *) 'N_tensors_2     = ', vm%N_tensors_2
    write(vm%out_fh, *) 'N_tensors_1     = ', vm%N_tensors_1
    write(vm%out_fh, *) 'N_vectorspinors = ', vm%N_vectorspinors
    write(vm%out_fh, *) 'Overall size of VM: '
    ! GNU extension
    ! write(vm%out_fh, *) 'sizeof(wavefunctions) = ', &
    !   sizeof(vm%scalars) + sizeof(vm%spinors) + sizeof(vm%conjspinors) + &
    !   sizeof(vm%bispinors) + sizeof(vm%vectors) + sizeof(vm%tensors_2) + &
    !   sizeof(vm%tensors_1) +  sizeof(vm%vectorspinors)
    ! write(vm%out_fh, *) 'sizeof(mometa) = ', sizeof(vm%momenta)
    ! write(vm%out_fh, *) 'sizeof(amplitudes) = ', sizeof(vm%amplitudes)
    ! write(vm%out_fh, *) 'sizeof(tables) = ', &
    !   sizeof(vm%table_amplitudes) + sizeof(vm%table_spin) + &
    !   sizeof(vm%table_flavor) + sizeof(vm%table_flv_col_is_allowed) + &
    !   sizeof(vm%table_color_flows) + sizeof(vm%table_color_factors) + &
    !   sizeof(vm%table_ghost_flags)
  end subroutine vm_write
```

Most of this is redundant (Fortran will deallocate when we leave the scope) but when we change from `allocatable`s to `pointer`s, it is necessary to avoid leaks

⟨*OVM Procedure Implementations*⟩+≡

```
  subroutine vm_final (vm)
    class(vm_t), intent(inout) :: vm
    deallocate (vm%table_flavor)
    deallocate (vm%table_color_flows)
    deallocate (vm%table_spin)
    deallocate (vm%table_ghost_flags)
    deallocate (vm%table_color_factors)
    deallocate (vm%table_flv_col_is_allowed)
    if (allocated (vm%coupl_real)) then
       deallocate (vm%coupl_real)
    end if
    if (allocated (vm%coupl_real2)) then
       deallocate (vm%coupl_real2)
    end if
    if (allocated (vm%coupl_cmplx)) then
       deallocate (vm%coupl_cmplx)
    end if
    if (allocated (vm%coupl_cmplx2)) then
       deallocate (vm%coupl_cmplx2)
```

```
    end if
    if (allocated (vm%mass)) then
       deallocate (vm%mass)
    end if
    if (allocated (vm%width)) then
       deallocate (vm%width)
    end if
    deallocate (vm%momenta)
    deallocate (vm%flavor)
    deallocate (vm%color)
    deallocate (vm%helicity)
    deallocate (vm%amplitudes)
    deallocate (vm%table_amplitudes)
    deallocate (vm%scalars)
    deallocate (vm%spinors)
    deallocate (vm%conjspinors)
    deallocate (vm%bispinors)
    deallocate (vm%vectors)
    deallocate (vm%tensors_2)
    deallocate (vm%tensors_1)
    deallocate (vm%vectorspinors)
  end subroutine vm_final
```

Handing over the polymorph object helicity didn't work out as planned. A work-around is the use of `pointers`. `flavor` and `color` are not yet used but would have to be changed to `pointers` as well. At least this potentially avoids copying. Actually, neither the allocatable nor the pointer version works in `gfortran 4.7` due to the broken `select type`. Back to Stone Age, i.e. integers.

⟨*OVM Procedure Implementations*⟩+≡

```
  subroutine vm_run (vm, mom, flavor, color, helicity)
    class(vm_t), intent(inout) :: vm
    real(default), dimension(0:3, *), intent(in) :: mom
    class(flavor_t), dimension(:), optional, intent(in) :: flavor
    class(color_t), dimension(:), optional, intent(in) :: color
    ! gfortran 4.7
    !class(helicity_t), dimension(:), optional, target, intent(in) :: helicity
    integer, dimension(:), optional, intent(in) :: helicity
    integer :: i, h, hi
    do i = 1, vm%N_momenta
      if (i <= vm%N_prt_in) then
        vm%momenta(i) = - mom(:, i)          ! incoming, crossing symmetry
      else
        vm%momenta(i) = mom(:, i)            ! outgoing
      end if
    end do
    if (present (flavor)) then
       allocate(vm%flavor(size(flavor)), source=flavor)
    else
       if (.not. (allocated (vm%flavor))) then
          allocate(flv_discrete::vm%flavor(vm%N_particles))
       end if
    end if
    if (present (color)) then
```

```
      allocate(vm%color(size(color)), source=color)
   else
      if (.not. (allocated (vm%color))) then
         allocate(col_discrete::vm%color(vm%N_col_flows))
      end if
   end if
   ! gfortran 4.7
   if (present (helicity)) then
      !vm%helicity => helicity
      vm%helicity = helicity
      call vm_run_one_helicity (vm, 1)
   else
     !if (.not. (associated (vm%helicity))) then
        !allocate(hel_discrete::vm%helicity(vm%N_particles))
     !end if
     if (.not. (allocated (vm%helicity))) then
        allocate(vm%helicity(vm%N_particles))
     end if
     if (vm%hel_finite == 0) return
     do hi = 1, vm%hel_finite
        h = vm%hel_map(hi)
        !<Work around [[gfortran 4.7 Bug 56731]] Implementation>>
        vm%helicity = vm%table_spin(:,h)
        call vm_run_one_helicity (vm, h)
     end do
   end if
 end subroutine vm_run
```

This only removes the `ICE` but still leads to a segmentation fault in `gfortran 4.7`. I am running out of ideas how to make this compiler work with arrays of polymorph datatypes.

⟨*Work around* `gfortran 4.7 Bug 56731` *Declarations*⟩≡
```
 integer :: hj
```

⟨*Work around* `gfortran 4.7 Bug 56731` *Implementation*⟩≡
```
 do hj = 1, size(vm%helicity)
    select type (hel => vm%helicity(hj))
    type is (hel_discrete)
       hel%i = vm%table_spin(hj,h)
    end select
 end do
```

⟨*Original version*⟩≡
```
 select type (hel => vm%helicity)
 type is (hel_discrete)
    hel(:)%i = vm%table_spin(:,h)
 end select
```

⟨*OVM Procedure Implementations*⟩+≡
```
 subroutine vm_run_one_helicity (vm, h)
   class(vm_t), intent(inout) :: vm
   integer, intent(in) :: h
   integer :: f, c, i
   vm%amplitudes = zero
   if (vm%N_levels > 0) then
```

```
      call null_all_wfs (vm)
      call iterate_instructions (vm)
   end if
   i = 1
   do c = 1, vm%N_col_flows
      do f = 1, vm%N_flavors
         if (vm%table_flv_col_is_allowed(f,c)) then
            vm%table_amplitudes(f,c,h) = vm%amplitudes(i)
            i = i + 1
         end if
      end do
   end do
 end subroutine
```

⟨*OVM Procedure Implementations*⟩+≡

```
 subroutine null_all_wfs (vm)
   type(vm_t), intent(inout) :: vm
   integer :: i, j
   vm%scalars%c = .False.
   vm%scalars%v = zero
   vm%spinors%c = .False.
   vm%conjspinors%c = .False.
   vm%bispinors%c = .False.
   vm%vectorspinors%c = .False.
   do i = 1, 4
      vm%spinors%v%a(i) = zero
      vm%conjspinors%v%a(i) = zero
      vm%bispinors%v%a(i) = zero
      do j = 1, 4
         vm%vectorspinors%v%psi(i)%a(j) = zero
      end do
   end do
   vm%vectors%c = .False.
   vm%vectors%v%t = zero
   vm%tensors_1%c = .False.
   vm%tensors_2%c = .False.
   do i = 1, 3
      vm%vectors%v%x(i) = zero
      vm%tensors_1%v%e(i) = zero
      vm%tensors_1%v%b(i) = zero
      do j = 1, 3
         vm%tensors_2%v%t(i,j) = zero
      end do
   end do
 end subroutine
```

X.21.3 Reading the bytecode

⟨*OVM Procedure Implementations*⟩+≡

```
 subroutine load_header (vm, IO)
   type(vm_t), intent(inout) :: vm
   integer, intent(inout) :: IO
```

```
    integer, dimension(len_instructions) :: line
    read(vm%bytecode_fh, fmt = *, iostat = IO) line
    vm%N_momenta = line(1)
    vm%N_particles = line(2)
    vm%N_prt_in = line(3)
    vm%N_prt_out = line(4)
    vm%N_amplitudes = line(5)
    vm%N_helicities = line(6)
    vm%N_col_flows = line(7)
    if (vm%N_momenta == 0) then
       vm%N_col_indices = 2
    else
       vm%N_col_indices = line(8)
    end if
    read(vm%bytecode_fh, fmt = *, iostat = IO)
    read(vm%bytecode_fh, fmt = *, iostat = IO) line
    vm%N_flavors = line(1)
    vm%N_col_factors = line(2)
    vm%N_scalars = line(3)
    vm%N_spinors = line(4)
    vm%N_conjspinors = line(5)
    vm%N_bispinors = line(6)
    vm%N_vectors = line(7)
    vm%N_tensors_2 = line(8)
    read(vm%bytecode_fh, fmt = *, iostat = IO)
    read(vm%bytecode_fh, fmt = *, iostat = IO) line
    vm%N_tensors_1 = line(1)
    vm%N_vectorspinors = line(2)
    ! Add 1 for seperating label lines like 'Another table'
    vm%N_table_lines = vm%N_helicities + 1 + vm%N_flavors + 1 + vm%N_col_flows &
      + 1 + vm%N_col_flows + 1 + vm%N_col_factors + 1 + vm%N_col_flows
  end subroutine load_header
```

⟨*OVM Procedure Implementations*⟩+≡

```
  subroutine read_tables (vm, IO)
    type(vm_t), intent(inout) :: vm
    integer, intent(inout) :: IO
    integer :: i
    integer, dimension(2) :: tmpcf
    integer, dimension(3) :: tmpfactor
    integer, dimension(vm%N_flavors) :: tmpF
    integer, dimension(vm%N_particles) :: tmpP
    real(default) :: factor
    do i = 1, vm%N_helicities
      read(vm%bytecode_fh, fmt = *, iostat = IO) vm%table_spin(:, i)
    end do

    read(vm%bytecode_fh, fmt = *, iostat = IO)
    do i = 1, vm%N_flavors
      read(vm%bytecode_fh, fmt = *, iostat = IO) vm%table_flavor(:, i)
    end do

    read(vm%bytecode_fh, fmt = *, iostat = IO)
    do i = 1, vm%N_col_flows
```

```
      read(vm%bytecode_fh, fmt = *, iostat = IO) vm%table_color_flows(:, :, i)
    end do

    read(vm%bytecode_fh, fmt = *, iostat = IO)
    do i = 1, vm%N_col_flows
      read(vm%bytecode_fh, fmt = *, iostat = IO) tmpP
      vm%table_ghost_flags(:, i) = int_to_log(tmpP)
    end do

    read(vm%bytecode_fh, fmt = *, iostat = IO)
    do i = 1, vm%N_col_factors
      read(vm%bytecode_fh, fmt = '(2I9)', iostat = IO, advance='no') tmpcf
      factor = zero
      do
        read(vm%bytecode_fh, fmt = '(3I9)', iostat = IO, advance='no', EOR=10) tmpfactor
        factor = factor + color_factor(tmpfactor(1), tmpfactor(2), tmpfactor(3))
      end do
      10 vm%table_color_factors(i) = OCF(tmpcf(1), tmpcf(2), factor)
    end do

    read(vm%bytecode_fh, fmt = *, iostat = IO)
    do i = 1, vm%N_col_flows
      read(vm%bytecode_fh, fmt = *, iostat = IO) tmpF
      vm%table_flv_col_is_allowed(:, i) = int_to_log(tmpF)
    end do
  end subroutine read_tables
```

This checking has proven useful more than once

⟨*OVM Procedure Implementations*⟩+≡

```
  subroutine extended_version_check (vm, IO)
    type(vm_t), intent(in) :: vm
    integer, intent(inout) :: IO
    character(256) :: buffer
    read(vm%bytecode_fh, fmt = *, iostat = IO) buffer
    if (vm%model /= buffer) then
      print *, "Bytecode has been generated for another/older model."
      stop 1
    else
      if (vm%verbose) then
         write (vm%out_fh, fmt = *) "Using the model: "
         write (vm%out_fh, fmt = *) char(vm%model)
      end if
    end if
  end subroutine extended_version_check
```

This chunk is copied verbatim from the `basic_vm`

⟨*OVM Procedure Implementations*⟩+≡

```
  subroutine basic_init (vm, verbose, out_fh)
    type(vm_t), intent(inout) :: vm
    logical, optional, intent(in) :: verbose
    integer, optional, intent(in) :: out_fh
    if (present (verbose)) then
       vm%verbose = verbose
```

```
  else
     vm%verbose = .true.
  end if
  if (present (out_fh)) then
     vm%out_fh = out_fh
   else
     vm%out_fh = stdout
  end if
  call set_stream (vm)
  call alloc_and_count (vm)
  if (vm%N_levels > 0) then
     call read_bytecode (vm)
     call sanity_check (vm)
  end if
  close (vm%bytecode_fh)
end subroutine basic_init

subroutine basic_write (vm)
  type(vm_t), intent(in) :: vm
  integer :: i
  write (vm%out_fh, *) '=====> VM ', char(vm%version), ' <====='
  write (vm%out_fh, *) 'verbose          =    ', vm%verbose
  write (vm%out_fh, *) 'bytecode_file    =    ', char (vm%bytecode_file)
  write (vm%out_fh, *) 'N_instructions   =    ', vm%N_instructions
  write (vm%out_fh, *) 'N_levels         =    ', vm%N_levels
  write (vm%out_fh, *) 'instructions     =    '
  do i = 1, vm%N_instructions
     write (vm%out_fh, *) vm%instructions(:, i)
  end do
  write (vm%out_fh, *) 'levels           =    ', vm%levels
end subroutine basic_write

subroutine alloc_and_count (vm)
  type(vm_t), intent(inout) :: vm
  integer, dimension(len_instructions) :: line
  character(256) :: buffer
  integer :: i, IO
  read(vm%bytecode_fh, fmt = *, iostat = IO) buffer
  if (vm%version /= buffer) then
    print *, "Bytecode has been generated with another Omega.Targets."
    stop 1
  else
    if (vm%verbose) then
       write (vm%out_fh, fmt = *) "Bytecode version fits."
    end if
  end if
  call extended_version_check (vm, IO)
  if (vm%verbose) then
     write (vm%out_fh, fmt = *) "Trying to allocate."
  end if
  do i = 1, N_comments
    read(vm%bytecode_fh, fmt = *, iostat = IO)
  end do
  call load_header (vm, IO)
```

```
  call alloc_arrays (vm)
  if (vm%N_momenta /= 0) then
     do i = 1, vm%N_table_lines + 1
       read(vm%bytecode_fh, fmt = *, iostat = IO)
     end do
     vm%N_instructions = 0
     vm%N_levels = 0
     do
       read(vm%bytecode_fh, fmt = *, end = 42) line
       if (line(1) /= 0) then
         vm%N_instructions = vm%N_instructions + 1
       else
         vm%N_levels = vm%N_levels + 1
       end if
     end do
     42 rewind(vm%bytecode_fh, iostat = IO)
     allocate (vm%instructions(len_instructions, vm%N_instructions))
     allocate (vm%levels(vm%N_levels))
     if (IO /= 0) then
       print *, "Error: vm.alloc : Couldn't load bytecode!"
       stop 1
     end if
  end if
end subroutine alloc_and_count

subroutine read_bytecode (vm)
  type(vm_t), intent(inout) :: vm
  integer, dimension(len_instructions) :: line
  integer :: i, j, IO
  ! Jump over version number, comments, header and first table description
  do i = 1, N_version_lines + N_comments + N_header_lines + 1
    read (vm%bytecode_fh, fmt = *, iostat = IO)
  end do
  call read_tables (vm, IO)
  read (vm%bytecode_fh, fmt = *, iostat = IO)
  i = 0; j = 0
  do
    read (vm%bytecode_fh, fmt = *, iostat = IO) line
    if (IO /= 0) exit
    if (line(1) == 0) then
      if (j <= vm%N_levels) then
        j = j + 1
        vm%levels(j) = i                 ! last index of a level is saved
      else
        print *, 'Error: vm.read_bytecode: File has more levels than anticipated!'
        stop 1
      end if
    else
      if (i <= vm%N_instructions) then
        i = i + 1                        ! A valid instruction line
        vm%instructions(:, i) = line
      else
        print *, 'Error: vm.read_bytecode: File is larger than anticipated!'
        stop 1
```

```
      end if
    end if
  end do
end subroutine read_bytecode

subroutine iterate_instructions (vm)
  type(vm_t), intent(inout) :: vm
  integer :: i, j
  if (vm%openmp) then
     !$omp parallel
     do j = 1, vm%N_levels - 1
        !$omp do schedule (static)
        do i = vm%levels (j) + 1, vm%levels (j + 1)
           call decode (vm, i)
        end do
        !$omp end do
     end do
     !$omp end parallel
  else
     do j = 1, vm%N_levels - 1
        do i = vm%levels (j) + 1, vm%levels (j + 1)
           call decode (vm, i)
        end do
     end do
  end if
end subroutine iterate_instructions

subroutine set_stream (vm)
  type(vm_t), intent(inout) :: vm
  integer :: IO
  call find_free_unit (vm%bytecode_fh, IO)
  open (vm%bytecode_fh, file = char (vm%bytecode_file), form = 'formatted', &
    access = 'sequential', status = 'old', position = 'rewind', iostat = IO, &
    action = 'read')
  if (IO /= 0) then
    print *, "Error: vm.set_stream: Bytecode file '", char(vm%bytecode_file), &
             "' not found!"
    stop 1
  end if
end subroutine set_stream

subroutine sanity_check (vm)
  type(vm_t), intent(in) :: vm
  if (vm%levels(1) /= 0) then
     print *, "Error: vm.vm_init: levels(1) != 0"
     stop 1
  end if
  if (vm%levels(vm%N_levels) /= vm%N_instructions) then
     print *, "Error: vm.vm_init: levels(N_levels) != N_instructions"
     stop 1
  end if
  if (vm%verbose) then
    write(vm%out_fh, *) "vm passed sanity check. Starting calculation."
  end if
```

```
end subroutine sanity_check
```

X.21.4 Main Decode Function

This is the heart of the OVM

⟨*OVM Procedure Implementations*⟩+≡

```
! pure & ! if no warnings
subroutine decode (vm, instruction_index)
  type(vm_t), intent(inout) :: vm
  integer, intent(in) :: instruction_index
  integer, dimension(len_instructions) :: i, curr
  complex(default) :: braket
  integer :: tmp
  real(default) :: w
  i = vm%instructions (:, instruction_index)
  select case (i(1))
  case ( : -1)       ! Jump over subinstructions

  ⟨cases of decode⟩
  case (0)
    print *, 'Error: Levelbreak put in decode! Line:', &
             instruction_index
    stop 1
  case default
    print *, "Error: Decode has case not catched! Line: ", &
             instruction_index
    stop 1
  end select
end subroutine decode
```

Momenta

The most trivial instruction

⟨*OVM Instructions*⟩≡

```
integer, parameter :: ovm_ADD_MOMENTA = 1
```

⟨cases *of* decode⟩≡

```
case (ovm_ADD_MOMENTA)
   vm%momenta(i(4)) = vm%momenta(i(5)) + vm%momenta(i(6))
   if (i(7) > 0) then
      vm%momenta(i(4)) = vm%momenta(i(4)) + vm%momenta(i(7))
   end if
```

Loading External states

⟨*OVM Instructions*⟩+≡

```
integer, parameter :: ovm_LOAD_SCALAR = 10
integer, parameter :: ovm_LOAD_SPINOR_INC = 11
integer, parameter :: ovm_LOAD_SPINOR_OUT = 12
```

```
  integer, parameter :: ovm_LOAD_CONJSPINOR_INC = 13
  integer, parameter :: ovm_LOAD_CONJSPINOR_OUT = 14
  integer, parameter :: ovm_LOAD_MAJORANA_INC = 15
  integer, parameter :: ovm_LOAD_MAJORANA_OUT = 16
  integer, parameter :: ovm_LOAD_VECTOR_INC = 17
  integer, parameter :: ovm_LOAD_VECTOR_OUT = 18
  integer, parameter :: ovm_LOAD_VECTORSPINOR_INC = 19
  integer, parameter :: ovm_LOAD_VECTORSPINOR_OUT = 20
  integer, parameter :: ovm_LOAD_TENSOR2_INC = 21
  integer, parameter :: ovm_LOAD_TENSOR2_OUT = 22
  integer, parameter :: ovm_LOAD_BRS_SCALAR = 30
  integer, parameter :: ovm_LOAD_BRS_SPINOR_INC = 31
  integer, parameter :: ovm_LOAD_BRS_SPINOR_OUT = 32
  integer, parameter :: ovm_LOAD_BRS_CONJSPINOR_INC = 33
  integer, parameter :: ovm_LOAD_BRS_CONJSPINOR_OUT = 34
  integer, parameter :: ovm_LOAD_BRS_VECTOR_INC = 37
  integer, parameter :: ovm_LOAD_BRS_VECTOR_OUT = 38
  integer, parameter :: ovm_LOAD_MAJORANA_GHOST_INC = 23
  integer, parameter :: ovm_LOAD_MAJORANA_GHOST_OUT = 24
  integer, parameter :: ovm_LOAD_BRS_MAJORANA_INC = 35
  integer, parameter :: ovm_LOAD_BRS_MAJORANA_OUT = 36
```

⟨cases *of* decode⟩+≡

```
  case (ovm_LOAD_SCALAR)
    vm%scalars(i(4))%v = one
    vm%scalars(i(4))%c = .True.

  case (ovm_LOAD_SPINOR_INC)
     call load_spinor(vm%spinors(i(4)), - ⟨p⟩, ⟨m⟩, &
                      vm%helicity(i(5)), ovm_LOAD_SPINOR_INC)

  case (ovm_LOAD_SPINOR_OUT)
     call load_spinor(vm%spinors(i(4)), ⟨p⟩, ⟨m⟩, &
                      vm%helicity(i(5)), ovm_LOAD_SPINOR_OUT)

  case (ovm_LOAD_CONJSPINOR_INC)
     call load_conjspinor(vm%conjspinors(i(4)), - ⟨p⟩, &
       ⟨m⟩, vm%helicity(i(5)), ovm_LOAD_CONJSPINOR_INC)

  case (ovm_LOAD_CONJSPINOR_OUT)
     call load_conjspinor(vm%conjspinors(i(4)), ⟨p⟩, &
       ⟨m⟩, vm%helicity(i(5)), ovm_LOAD_CONJSPINOR_OUT)

  case (ovm_LOAD_MAJORANA_INC)
     call load_bispinor(vm%bispinors(i(4)), - ⟨p⟩, &
       ⟨m⟩, vm%helicity(i(5)), ovm_LOAD_MAJORANA_INC)

  case (ovm_LOAD_MAJORANA_OUT)
     call load_bispinor(vm%bispinors(i(4)), ⟨p⟩, ⟨m⟩, &
                      vm%helicity(i(5)), ovm_LOAD_MAJORANA_OUT)

  case (ovm_LOAD_VECTOR_INC)
     call load_vector(vm%vectors(i(4)), - ⟨p⟩, ⟨m⟩, &
                      vm%helicity(i(5)), ovm_LOAD_VECTOR_INC)
```

```
case (ovm_LOAD_VECTOR_OUT)
   call load_vector(vm%vectors(i(4)), ⟨p⟩, ⟨m⟩, &
                    vm%helicity(i(5)), ovm_LOAD_VECTOR_OUT)

case (ovm_LOAD_VECTORSPINOR_INC)
  !select type (h => vm%helicity(i(5)))
  !type is (hel_discrete)
     !vm%vectorspinors(i(4))%v = veps(⟨m⟩, - ⟨p⟩, &
                                  !h%i)
  !end select
  vm%vectorspinors(i(4))%v = veps(⟨m⟩, - ⟨p⟩, &
                              vm%helicity(i(5)))
  vm%vectorspinors(i(4))%c = .True.

case (ovm_LOAD_VECTORSPINOR_OUT)
  !select type (h => vm%helicity(i(5)))
  !type is (hel_discrete)
     !vm%vectorspinors(i(4))%v = veps(⟨m⟩, ⟨p⟩, &
                                  !h%i)
  !end select
  vm%vectorspinors(i(4))%v = veps(⟨m⟩, ⟨p⟩, &
                              vm%helicity(i(5)))
  vm%vectorspinors(i(4))%c = .True.

case (ovm_LOAD_TENSOR2_INC)
  !select type (h => vm%helicity(i(5)))
  !type is (hel_discrete)
     !vm%tensors_2(i(4))%v = eps2(⟨m⟩, - ⟨p⟩, &
                              !h%i)
  !end select
  vm%tensors_2(i(4))%c = .True.

case (ovm_LOAD_TENSOR2_OUT)
  !select type (h => vm%helicity(i(5)))
  !type is (hel_discrete)
     !vm%tensors_2(i(4))%v = eps2(⟨m⟩, ⟨p⟩, h%i)
  !end select
  vm%tensors_2(i(4))%c = .True.

case (ovm_LOAD_BRS_SCALAR)
  vm%scalars(i(4))%v = (0, -1) * (⟨p⟩ * ⟨p⟩ - &
                              ⟨m⟩**2)
  vm%scalars(i(4))%c = .True.

case (ovm_LOAD_BRS_SPINOR_INC)
  print *, 'not implemented'
  stop 1
case (ovm_LOAD_BRS_SPINOR_OUT)
  print *, 'not implemented'
  stop 1
case (ovm_LOAD_BRS_CONJSPINOR_INC)
  print *, 'not implemented'
  stop 1
```

```
case (ovm_LOAD_BRS_CONJSPINOR_OUT)
  print *, 'not implemented'
  stop 1
case (ovm_LOAD_BRS_VECTOR_INC)
  print *, 'not implemented'
  stop 1
case (ovm_LOAD_BRS_VECTOR_OUT)
  print *, 'not implemented'
  stop 1
case (ovm_LOAD_MAJORANA_GHOST_INC)
  print *, 'not implemented'
  stop 1
case (ovm_LOAD_MAJORANA_GHOST_OUT)
  print *, 'not implemented'
  stop 1
case (ovm_LOAD_BRS_MAJORANA_INC)
  print *, 'not implemented'
  stop 1
case (ovm_LOAD_BRS_MAJORANA_OUT)
  print *, 'not implemented'
  stop 1
```

Brakets and Fusions

NB: during, execution, the type of the coupling constant is implicit in the instruction

$\langle$*OVM Instructions*$\rangle$+≡

```
integer, parameter :: ovm_CALC_BRAKET = 2

integer, parameter :: ovm_FUSE_V_FF = -1
integer, parameter :: ovm_FUSE_F_VF = -2
integer, parameter :: ovm_FUSE_F_FV = -3
integer, parameter :: ovm_FUSE_VA_FF = -4
integer, parameter :: ovm_FUSE_F_VAF = -5
integer, parameter :: ovm_FUSE_F_FVA = -6
integer, parameter :: ovm_FUSE_VA2_FF = -7
integer, parameter :: ovm_FUSE_F_VA2F = -8
integer, parameter :: ovm_FUSE_F_FVA2 = -9
integer, parameter :: ovm_FUSE_A_FF = -10
integer, parameter :: ovm_FUSE_F_AF = -11
integer, parameter :: ovm_FUSE_F_FA = -12
integer, parameter :: ovm_FUSE_VL_FF = -13
integer, parameter :: ovm_FUSE_F_VLF = -14
integer, parameter :: ovm_FUSE_F_FVL = -15
integer, parameter :: ovm_FUSE_VR_FF = -16
integer, parameter :: ovm_FUSE_F_VRF = -17
integer, parameter :: ovm_FUSE_F_FVR = -18
integer, parameter :: ovm_FUSE_VLR_FF = -19
integer, parameter :: ovm_FUSE_F_VLRF = -20
integer, parameter :: ovm_FUSE_F_FVLR = -21
integer, parameter :: ovm_FUSE_SP_FF = -22
integer, parameter :: ovm_FUSE_F_SPF = -23
integer, parameter :: ovm_FUSE_F_FSP = -24
```

```
  integer, parameter :: ovm_FUSE_S_FF = -25
  integer, parameter :: ovm_FUSE_F_SF = -26
  integer, parameter :: ovm_FUSE_F_FS = -27
  integer, parameter :: ovm_FUSE_P_FF = -28
  integer, parameter :: ovm_FUSE_F_PF = -29
  integer, parameter :: ovm_FUSE_F_FP = -30
  integer, parameter :: ovm_FUSE_SL_FF = -31
  integer, parameter :: ovm_FUSE_F_SLF = -32
  integer, parameter :: ovm_FUSE_F_FSL = -33
  integer, parameter :: ovm_FUSE_SR_FF = -34
  integer, parameter :: ovm_FUSE_F_SRF = -35
  integer, parameter :: ovm_FUSE_F_FSR = -36
  integer, parameter :: ovm_FUSE_SLR_FF = -37
  integer, parameter :: ovm_FUSE_F_SLRF = -38
  integer, parameter :: ovm_FUSE_F_FSLR = -39

  integer, parameter :: ovm_FUSE_G_GG = -40
  integer, parameter :: ovm_FUSE_V_SS = -41
  integer, parameter :: ovm_FUSE_S_VV = -42
  integer, parameter :: ovm_FUSE_S_VS = -43
  integer, parameter :: ovm_FUSE_V_SV = -44
  integer, parameter :: ovm_FUSE_S_SS = -45
  integer, parameter :: ovm_FUSE_S_SVV = -46
  integer, parameter :: ovm_FUSE_V_SSV = -47
  integer, parameter :: ovm_FUSE_S_SSS = -48
  integer, parameter :: ovm_FUSE_V_VVV = -49

  integer, parameter :: ovm_FUSE_S_G2 = -50
  integer, parameter :: ovm_FUSE_G_SG = -51
  integer, parameter :: ovm_FUSE_G_GS = -52
  integer, parameter :: ovm_FUSE_S_G2_SKEW = -53
  integer, parameter :: ovm_FUSE_G_SG_SKEW = -54
  integer, parameter :: ovm_FUSE_G_GS_SKEW = -55
```

Shorthands

$\langle p \rangle \equiv$
```
  vm%momenta(i(5))
```

$\langle m \rangle \equiv$
```
  vm%mass(i(2))
```

$\langle p1 \rangle \equiv$
```
  vm%momenta(curr(6))
```

$\langle p2 \rangle \equiv$
```
  vm%momenta(curr(8))
```

$\langle v1 \rangle \equiv$
```
  vm%vectors(curr(5))%v
```

$\langle v2 \rangle \equiv$
```
  vm%vectors(curr(7))%v
```

$\langle s1 \rangle \equiv$
```
  vm%scalars(curr(5))%v
```

$\langle s2 \rangle \equiv$
```
  vm%scalars(curr(7))%v
```

⟨*c*⟩≡
```
  sgn_coupl_cmplx(vm, curr(2))
```
⟨*c1*⟩≡
```
  sgn_coupl_cmplx2(vm, curr(2), 1)
```
⟨*c2*⟩≡
```
  sgn_coupl_cmplx2(vm, curr(2), 2)
```
⟨*check for matching color and flavor amplitude of braket (old)*⟩≡
```
  if ((i(4) == o%cols(1)) .or. (i(4) == o%cols(2)) .or. &
    ((mode%col_MC .eq. FULL_SUM) .or. (mode%col_MC .eq. DIAG_COL))) then
```
Just a stub for now. Will be reimplemented with the polymorph type `color` similar to the `select type(helicity)` when we need it.

⟨*check for matching color and flavor amplitude*⟩≡

⟨`case`*s of* `decode`⟩+≡
```
  case (ovm_CALC_BRAKET)
     ⟨check for matching color and flavor amplitude⟩
     tmp = instruction_index + 1
     do
       if (tmp > vm%N_instructions) exit
       curr = vm%instructions(:, tmp)
       if (curr(1) >= 0) exit                        ! End of fusions
       select case (curr(1))
       case (ovm_FUSE_V_FF, ovm_FUSE_VL_FF, ovm_FUSE_VR_FF)
         braket = vm%vectors(curr(4))%v * vec_ff(vm, curr)

       case (ovm_FUSE_F_VF, ovm_FUSE_F_VLF, ovm_FUSE_F_VRF)
         braket = vm%conjspinors(curr(4))%v * ferm_vf(vm, curr)

       case (ovm_FUSE_F_FV, ovm_FUSE_F_FVL, ovm_FUSE_F_FVR)
         braket = ferm_fv(vm, curr) * vm%spinors(curr(4))%v

       case (ovm_FUSE_VA_FF)
         braket = vm%vectors(curr(4))%v * vec_ff2(vm, curr)

       case (ovm_FUSE_F_VAF)
         braket = vm%conjspinors(curr(4))%v * ferm_vf2(vm, curr)

       case (ovm_FUSE_F_FVA)
         braket = ferm_fv2(vm, curr) * vm%spinors(curr(4))%v

       case (ovm_FUSE_S_FF, ovm_FUSE_SP_FF)
         braket = vm%scalars(curr(4))%v * scal_ff(vm, curr)

       case (ovm_FUSE_F_SF, ovm_FUSE_F_SPF)
         braket = vm%conjspinors(curr(4))%v * ferm_sf(vm, curr)

       case (ovm_FUSE_F_FS, ovm_FUSE_F_FSP)
         braket = ferm_fs(vm, curr) * vm%spinors(curr(4))%v

       case (ovm_FUSE_G_GG)
         braket = vm%vectors(curr(4))%v * &
           g_gg(⟨c⟩, &
                ⟨v1⟩, ⟨p1⟩, &
```

```
         ⟨v2⟩, ⟨p2⟩)

case (ovm_FUSE_S_VV)
  braket = vm%scalars(curr(4))%v * ⟨c⟩ * &
           (⟨v1⟩ * vm%vectors(curr(6))%v)

case (ovm_FUSE_V_SS)
  braket = vm%vectors(curr(4))%v * &
           v_ss(⟨c⟩, ⟨s1⟩, ⟨p1⟩, &
                       ⟨s2⟩, ⟨p2⟩)

case (ovm_FUSE_S_G2, ovm_FUSE_S_G2_SKEW)
  braket = vm%scalars(curr(4))%v * scal_g2(vm, curr)

case (ovm_FUSE_G_SG, ovm_FUSE_G_GS, ovm_FUSE_G_SG_SKEW, ovm_FUSE_G_GS_SKEW)
  braket = vm%vectors(curr(4))%v * gauge_sg(vm, curr)

case (ovm_FUSE_S_VS)
  braket = vm%scalars(curr(4))%v * &
    s_vs(⟨c⟩, &
         ⟨v1⟩, ⟨p1⟩, &
         ⟨s2⟩, ⟨p2⟩)

case (ovm_FUSE_V_SV)
  braket = (vm%vectors(curr(4))%v * vm%vectors(curr(6))%v) * &
           (⟨c⟩ * ⟨s1⟩)

case (ovm_FUSE_S_SS)
  braket = vm%scalars(curr(4))%v * &
    ⟨c⟩ * &
    (⟨s1⟩ * vm%scalars(curr(6))%v)

case (ovm_FUSE_S_SSS)
  braket = vm%scalars(curr(4))%v * &
    ⟨c⟩ * &
    (⟨s1⟩ * vm%scalars(curr(6))%v * &
     ⟨s2⟩)

case (ovm_FUSE_S_SVV)
  braket = vm%scalars(curr(4))%v * &
    ⟨c⟩ * &
    ⟨s1⟩ * (vm%vectors(curr(6))%v * &
                              ⟨v2⟩)

case (ovm_FUSE_V_SSV)
  braket = vm%vectors(curr(4))%v * &
    (⟨c⟩ * ⟨s1⟩ * &
     vm%scalars(curr(6))%v) * ⟨v2⟩

case (ovm_FUSE_V_VVV)
  braket = ⟨c⟩ * &
    (⟨v1⟩ * vm%vectors(curr(6))%v) * &
    (vm%vectors(curr(4))%v * ⟨v2⟩)
```

```
    case default
      print *, 'Braket', curr(1), 'not implemented'
      stop 1

    end select
    vm%amplitudes(i(4)) = vm%amplitudes(i(4)) + curr(3) * braket
    tmp = tmp + 1
  end do

  vm%amplitudes(i(4)) = vm%amplitudes(i(4)) * i(2)
  if (i(5) > 1) then
    vm%amplitudes(i(4)) = vm%amplitudes(i(4)) * &         ! Symmetry factor
                          (one / sqrt(real(i(5), kind=default)))
  end if
```

Propagators

⟨*OVM Instructions*⟩+≡

```
integer, parameter :: ovm_PROPAGATE_SCALAR = 51
integer, parameter :: ovm_PROPAGATE_COL_SCALAR = 52
integer, parameter :: ovm_PROPAGATE_GHOST = 53
integer, parameter :: ovm_PROPAGATE_SPINOR = 54
integer, parameter :: ovm_PROPAGATE_CONJSPINOR = 55
integer, parameter :: ovm_PROPAGATE_MAJORANA = 56
integer, parameter :: ovm_PROPAGATE_COL_MAJORANA = 57
integer, parameter :: ovm_PROPAGATE_UNITARITY = 58
integer, parameter :: ovm_PROPAGATE_COL_UNITARITY = 59
integer, parameter :: ovm_PROPAGATE_FEYNMAN = 60
integer, parameter :: ovm_PROPAGATE_COL_FEYNMAN = 61
integer, parameter :: ovm_PROPAGATE_VECTORSPINOR = 62
integer, parameter :: ovm_PROPAGATE_TENSOR2 = 63
integer, parameter :: ovm_PROPAGATE_NONE = 64
```

⟨*check for matching color and flavor amplitude of propagator (old)*⟩≡

```
if ((mode%col_MC .eq. FULL_SUM) .or. (mode%col_MC .eq. DIAG_COL)) then
  select case(i(1))
  case (ovm_PROPAGATE_PSI)
    go = .not. vm%spinors%c(i(4))
  case (ovm_PROPAGATE_PSIBAR)
    go = .not. vm%conjspinors%c(i(4))
  case (ovm_PROPAGATE_UNITARITY, ovm_PROPAGATE_FEYNMAN, &
    ovm_PROPAGATE_COL_FEYNMAN)
    go = .not. vm%vectors%c(i(4))
  end select
else
  go = (i(8) == o%cols(1)) .or. (i(8) == o%cols(2))
end if
if (go) then
```

⟨case*s of* decode⟩+≡

```
⟨check for matching color and flavor amplitude⟩
case (ovm_PROPAGATE_SCALAR : ovm_PROPAGATE_NONE)
   tmp = instruction_index + 1
   do
```

```
curr = vm%instructions(:,tmp)
if (curr(1) >= 0) exit                        ! End of fusions
select case (curr(1))
case (ovm_FUSE_V_FF, ovm_FUSE_VL_FF, ovm_FUSE_VR_FF)
  vm%vectors(curr(4))%v = vm%vectors(curr(4))%v + curr(3) * &
                          vec_ff(vm, curr)

case (ovm_FUSE_F_VF, ovm_FUSE_F_VLF, ovm_FUSE_F_VRF)
  vm%spinors(curr(4))%v = vm%spinors(curr(4))%v + curr(3) * &
                          ferm_vf(vm, curr)

case (ovm_FUSE_F_FV, ovm_FUSE_F_FVL, ovm_FUSE_F_FVR)
  vm%conjspinors(curr(4))%v = vm%conjspinors(curr(4))%v + curr(3) * &
                              ferm_fv(vm, curr)

case (ovm_FUSE_VA_FF)
  vm%vectors(curr(4))%v = vm%vectors(curr(4))%v + curr(3) * &
                          vec_ff2(vm, curr)

case (ovm_FUSE_F_VAF)
  vm%spinors(curr(4))%v = vm%spinors(curr(4))%v + curr(3) * &
                          ferm_vf2(vm, curr)

case (ovm_FUSE_F_FVA)
  vm%conjspinors(curr(4))%v = vm%conjspinors(curr(4))%v + curr(3) * &
                              ferm_fv2(vm, curr)

case (ovm_FUSE_S_FF, ovm_FUSE_SP_FF)
  vm%scalars(curr(4))%v = vm%scalars(curr(4))%v + curr(3) * &
                          scal_ff(vm, curr)

case (ovm_FUSE_F_SF, ovm_FUSE_F_SPF)
  vm%spinors(curr(4))%v = vm%spinors(curr(4))%v + curr(3) * &
                          ferm_sf(vm, curr)

case (ovm_FUSE_F_FS, ovm_FUSE_F_FSP)
  vm%conjspinors(curr(4))%v = vm%conjspinors(curr(4))%v + curr(3) * &
                              ferm_fs(vm, curr)

case (ovm_FUSE_G_GG)
  vm%vectors(curr(4))%v = vm%vectors(curr(4))%v + curr(3) * &
    g_gg(⟨c⟩, ⟨v1⟩, &
         ⟨p1⟩, ⟨v2⟩, &
         ⟨p2⟩)

case (ovm_FUSE_S_VV)
  vm%scalars(curr(4))%v = vm%scalars(curr(4))%v + curr(3) * &
    ⟨c⟩ * &
    (⟨v1⟩ * vm%vectors(curr(6))%v)

case (ovm_FUSE_V_SS)
  vm%vectors(curr(4))%v = vm%vectors(curr(4))%v + curr(3) * &
           v_ss(⟨c⟩, ⟨s1⟩, ⟨p1⟩, &
                       ⟨s2⟩, ⟨p2⟩)
```

```
case (ovm_FUSE_S_G2, ovm_FUSE_S_G2_SKEW)
  vm%scalars(curr(4))%v = vm%scalars(curr(4))%v + &
                          scal_g2(vm, curr) * curr(3)

case (ovm_FUSE_G_SG, ovm_FUSE_G_GS, ovm_FUSE_G_SG_SKEW, ovm_FUSE_G_GS_SKEW)
  vm%vectors(curr(4))%v = vm%vectors(curr(4))%v + &
                          gauge_sg(vm, curr) * curr(3)

case (ovm_FUSE_S_VS)
  vm%scalars(curr(4))%v = vm%scalars(curr(4))%v + &
    s_vs(⟨c⟩, &
         ⟨v1⟩, ⟨p1⟩, &
         ⟨s2⟩, ⟨p2⟩) * curr(3)

case (ovm_FUSE_V_SV)
  vm%vectors(curr(4))%v = vm%vectors(curr(4))%v + &
    vm%vectors(curr(6))%v * &
    (⟨c⟩ * ⟨s1⟩ * curr(3))

case (ovm_FUSE_S_SS)
  vm%scalars(curr(4))%v = vm%scalars(curr(4))%v + &
    ⟨c⟩ * &
    (⟨s1⟩ * vm%scalars(curr(6))%v) * curr(3)

case (ovm_FUSE_S_SSS)
  vm%scalars(curr(4))%v = vm%scalars(curr(4))%v + &
    ⟨c⟩ * &
    (⟨s1⟩ * vm%scalars(curr(6))%v * &
     ⟨s2⟩) * curr(3)

case (ovm_FUSE_S_SVV)
  vm%scalars(curr(4))%v = vm%scalars(curr(4))%v + &
    ⟨c⟩ * &
    ⟨s1⟩ * (vm%vectors(curr(6))%v * &
                             ⟨v2⟩) * curr(3)

case (ovm_FUSE_V_SSV)
  vm%vectors(curr(4))%v = vm%vectors(curr(4))%v + &
    (⟨c⟩ * ⟨s1⟩ * &
     vm%scalars(curr(6))%v) * ⟨v2⟩ * curr(3)

case (ovm_FUSE_V_VVV)
  vm%vectors(curr(4))%v = vm%vectors(curr(4))%v + &
    (⟨c⟩ * (⟨v1⟩ * &
     vm%vectors(curr(6))%v)) * curr(3) * ⟨v2⟩

case default
  print *, 'Fusion', curr(1), 'not implemented'
  stop 1

end select
tmp = tmp + 1
```

```
    end do

    select case (i(3))
    case (0)
      w = zero

    case (1)
      w = vm%width(i(2))

    case (2)
      w = wd_tl(⟨p⟩, vm%width(i(2)))

    case default
       print *, 'not implemented'
       stop 1

    end select

    select case (i(1))
    ⟨propagator cases in decode⟩
    end select
```

⟨*propagator* cases *in* decode⟩≡

```
  case (ovm_PROPAGATE_SCALAR)
    vm%scalars(i(4))%v = pr_phi(⟨p⟩, ⟨m⟩, &
                                 w, vm%scalars(i(4))%v)
    vm%scalars(i(4))%c = .True.

  case (ovm_PROPAGATE_COL_SCALAR)
    vm%scalars(i(4))%v = - one / N_ * pr_phi(⟨p⟩, &
                          ⟨m⟩, w, vm%scalars(i(4))%v)
    vm%scalars(i(4))%c = .True.

  case (ovm_PROPAGATE_GHOST)
    vm%scalars(i(4))%v = imago * pr_phi(⟨p⟩, ⟨m⟩, &
                                         w, vm%scalars(i(4))%v)
    vm%scalars(i(4))%c = .True.

  case (ovm_PROPAGATE_SPINOR)
    vm%spinors(i(4))%v = pr_psi(⟨p⟩, ⟨m⟩, &
                                 w, vm%spinors(i(4))%v)
    vm%spinors(i(4))%c = .True.

  case (ovm_PROPAGATE_CONJSPINOR)
    vm%conjspinors(i(4))%v = pr_psibar(⟨p⟩, ⟨m⟩, &
                                        w, vm%conjspinors(i(4))%v)
    vm%conjspinors(i(4))%c = .True.

  case (ovm_PROPAGATE_MAJORANA)
    vm%bispinors(i(4))%v = bi_pr_psi(⟨p⟩, ⟨m⟩, &
                                      w, vm%bispinors(i(4))%v)
    vm%bispinors(i(4))%c = .True.

  case (ovm_PROPAGATE_COL_MAJORANA)
```

```
    vm%bispinors(i(4))%v = (- one / N_) * &
                           bi_pr_psi(⟨p⟩, ⟨m⟩, &
                                  w, vm%bispinors(i(4))%v)
    vm%bispinors(i(4))%c = .True.

  case (ovm_PROPAGATE_UNITARITY)
    vm%vectors(i(4))%v = pr_unitarity(⟨p⟩, ⟨m⟩, &
                                      w, vm%vectors(i(4))%v)
    vm%vectors(i(4))%c = .True.

  case (ovm_PROPAGATE_COL_UNITARITY)
    vm%vectors(i(4))%v = - one / N_ * pr_unitarity(⟨p⟩, &
                         ⟨m⟩, w, vm%vectors(i(4))%v)
    vm%vectors(i(4))%c = .True.

  case (ovm_PROPAGATE_FEYNMAN)
    vm%vectors(i(4))%v = pr_feynman(⟨p⟩, vm%vectors(i(4))%v)
    vm%vectors(i(4))%c = .True.

  case (ovm_PROPAGATE_COL_FEYNMAN)
    vm%vectors(i(4))%v = - one / N_ * &
                         pr_feynman(⟨p⟩, vm%vectors(i(4))%v)
    vm%vectors(i(4))%c = .True.

  case (ovm_PROPAGATE_VECTORSPINOR)
    vm%vectorspinors(i(4))%v = pr_grav(⟨p⟩, ⟨m⟩, &
                                  w, vm%vectorspinors(i(4))%v)
    vm%vectorspinors(i(4))%c = .True.

  case (ovm_PROPAGATE_TENSOR2)
    vm%tensors_2(i(4))%v = pr_tensor(⟨p⟩, ⟨m⟩, &
                                     w, vm%tensors_2(i(4))%v)
    vm%tensors_2(i(4))%c = .True.

  case (ovm_PROPAGATE_NONE)
  ! This will not work with color MC. Appropriate type%c has to be set to
  ! .True.
```

X.21.5 Helper functions

Factoring out these parts helps a lot to keep sane but might hurt the performance of the VM noticably. In that case, we have to copy & paste to avoid the additional function calls. Note that with preprocessor macros, we could maintain this factorized form (and factor out even more since types don't have to match), in case we would decide to allow this

⟨*load outer wave function*⟩≡

```
  !select type (h)
  !type is (hel_trigonometric)
     !wf%v = (cos (h%theta) * load_wf (m, p, + 1) + &
             !sin (h%theta) * load_wf (m, p, - 1)) * sqrt2
  !type is (hel_exponential)
     !wf%v = exp (+ imago * h%phi) * load_wf (m, p, + 1) + &
```

```
          !exp (- imago * h%phi) * load_wf (m, p, - 1)
  !type is (hel_spherical)
     !wf%v = (exp (+ imago * h%phi) * cos (h%theta) * load_wf (m, p, + 1) + &
            !exp (- imago * h%phi) * sin (h%theta) * load_wf (m, p, - 1)) * &
          !sqrt2
  !type is(hel_discrete)
      !wf%v = load_wf (m, p, h%i)
  !end select
  wf%v = load_wf (m, p, h)
  wf%c = .True.
```

Caveat: Helicity MC not tested with Majorana particles but should be fine

⟨*check for matching color and flavor amplitude of wf (old)*⟩≡

```
  if ((mode%col_MC .eq. FULL_SUM) .or. (mode%col_MC .eq. DIAG_COL)) then
    go = .not. vm%spinors%c(i(4))
  else
    go = (i(8) == o%cols(1)) .or. (i(8) == o%cols(2))
  end if
  if (go) ..
```

⟨*OVM Procedure Implementations*⟩+≡

```
  subroutine load_bispinor(wf, p, m, h, opcode)
    type(vm_bispinor), intent(out) :: wf
    type(momentum), intent(in) :: p
    real(default), intent(in) :: m
    !class(helicity_t), intent(in) :: h
    integer, intent(in) :: h
    integer, intent(in) :: opcode
    procedure(bi_u), pointer :: load_wf
    ⟨check for matching color and flavor amplitude⟩
    select case (opcode)
    case (ovm_LOAD_MAJORANA_INC)
       load_wf => bi_u
    case (ovm_LOAD_MAJORANA_OUT)
       load_wf => bi_v
    case default
       load_wf => null()
    end select
    ⟨load outer wave function⟩
  end subroutine load_bispinor

  subroutine load_spinor(wf, p, m, h, opcode)
    type(vm_spinor), intent(out) :: wf
    type(momentum), intent(in) :: p
    real(default), intent(in) :: m
    !class(helicity_t), intent(in) :: h
    integer, intent(in) :: h
    integer, intent(in) :: opcode
    procedure(u), pointer :: load_wf
    ⟨check for matching color and flavor amplitude⟩
    select case (opcode)
    case (ovm_LOAD_SPINOR_INC)
       load_wf => u
    case (ovm_LOAD_SPINOR_OUT)
       load_wf => v
```

```
    case default
       load_wf => null()
    end select
    ⟨load outer wave function⟩
  end subroutine load_spinor

  subroutine load_conjspinor(wf, p, m, h, opcode)
    type(vm_conjspinor), intent(out) :: wf
    type(momentum), intent(in) :: p
    real(default), intent(in) :: m
    !class(helicity_t), intent(in) :: h
    integer, intent(in) :: h
    integer, intent(in) :: opcode
    procedure(ubar), pointer :: load_wf
    ⟨check for matching color and flavor amplitude⟩
    select case (opcode)
    case (ovm_LOAD_CONJSPINOR_INC)
       load_wf => vbar
    case (ovm_LOAD_CONJSPINOR_OUT)
       load_wf => ubar
    case default
       load_wf => null()
    end select
    ⟨load outer wave function⟩
  end subroutine load_conjspinor

  subroutine load_vector(wf, p, m, h, opcode)
    type(vm_vector), intent(out) :: wf
    type(momentum), intent(in) :: p
    real(default), intent(in) :: m
    !class(helicity_t), intent(in) :: h
    integer, intent(in) :: h
    integer, intent(in) :: opcode
    procedure(eps), pointer :: load_wf
    ⟨check for matching color and flavor amplitude⟩
    load_wf => eps
    ⟨load outer wave function⟩
    if (opcode == ovm_LOAD_VECTOR_OUT) then
       wf%v = conjg(wf%v)
    end if
  end subroutine load_vector
```

⟨*OVM Procedure Implementations*⟩+≡

```
  function ferm_vf(vm, curr) result (x)
    type(spinor) :: x
    class(vm_t), intent(in) :: vm
    integer, dimension(:), intent(in) :: curr
    procedure(f_vf), pointer :: load_wf
    select case (curr(1))
    case (ovm_FUSE_F_VF)
       load_wf => f_vf
    case (ovm_FUSE_F_VLF)
       load_wf => f_vlf
    case (ovm_FUSE_F_VRF)
```

```
    load_wf => f_vrf
  case default
    load_wf => null()
  end select
  x = load_wf(⟨c⟩, ⟨v1⟩, vm%spinors(curr(6))%v)
end function ferm_vf

function ferm_vf2(vm, curr) result (x)
  type(spinor) :: x
  class(vm_t), intent(in) :: vm
  integer, dimension(:), intent(in) :: curr
  procedure(f_vaf), pointer :: load_wf
  select case (curr(1))
  case (ovm_FUSE_F_VAF)
    load_wf => f_vaf
  case default
    load_wf => null()
  end select
  x = f_vaf(⟨c1⟩, ⟨c2⟩, ⟨v1⟩, vm%spinors(curr(6))%v)
end function ferm_vf2

function ferm_sf(vm, curr) result (x)
  type(spinor) :: x
  class(vm_t), intent(in) :: vm
  integer, dimension(:), intent(in) :: curr
  select case (curr(1))
  case (ovm_FUSE_F_SF)
    x = f_sf(⟨c⟩, ⟨s1⟩, vm%spinors(curr(6))%v)
  case (ovm_FUSE_F_SPF)
    x = f_spf(⟨c1⟩, ⟨c2⟩, ⟨s1⟩, vm%spinors(curr(6))%v)
  case default
  end select
end function ferm_sf

function ferm_fv(vm, curr) result (x)
  type(conjspinor) :: x
  class(vm_t), intent(in) :: vm
  integer, dimension(:), intent(in) :: curr
  procedure(f_fv), pointer :: load_wf
  select case (curr(1))
  case (ovm_FUSE_F_FV)
    load_wf => f_fv
  case (ovm_FUSE_F_FVL)
    load_wf => f_fvl
  case (ovm_FUSE_F_FVR)
    load_wf => f_fvr
  case default
    load_wf => null()
  end select
  x = load_wf(⟨c⟩, vm%conjspinors(curr(5))%v, vm%vectors(curr(6))%v)
end function ferm_fv

function ferm_fv2(vm, curr) result (x)
  type(conjspinor) :: x
```

```
  class(vm_t), intent(in) :: vm
  integer, dimension(:), intent(in) :: curr
  procedure(f_fva), pointer :: load_wf
  select case (curr(1))
  case (ovm_FUSE_F_FVA)
     load_wf => f_fva
  case default
     load_wf => null()
  end select
  x = f_fva(⟨c1⟩, ⟨c2⟩, &
            vm%conjspinors(curr(5))%v, vm%vectors(curr(6))%v)
end function ferm_fv2

function ferm_fs(vm, curr) result (x)
  type(conjspinor) :: x
  class(vm_t), intent(in) :: vm
  integer, dimension(:), intent(in) :: curr
  procedure(f_fs), pointer :: load_wf
  select case (curr(1))
  case (ovm_FUSE_F_FS)
     x = f_fs(⟨c⟩, vm%conjspinors(curr(5))%v, vm%scalars(curr(6))%v)
  case (ovm_FUSE_F_FSP)
     x = f_fsp(⟨c1⟩, ⟨c2⟩, &
          vm%conjspinors(curr(5))%v, vm%scalars(curr(6))%v)
  case default
     x%a = zero
  end select
end function ferm_fs

function vec_ff(vm, curr) result (x)
  type(vector) :: x
  class(vm_t), intent(in) :: vm
  integer, dimension(:), intent(in) :: curr
  procedure(v_ff), pointer :: load_wf
  select case (curr(1))
  case (ovm_FUSE_V_FF)
     load_wf => v_ff
  case (ovm_FUSE_VL_FF)
     load_wf => vl_ff
  case (ovm_FUSE_VR_FF)
     load_wf => vr_ff
  case default
     load_wf => null()
  end select
  x = load_wf(⟨c⟩, vm%conjspinors(curr(5))%v, vm%spinors(curr(6))%v)
end function vec_ff

function vec_ff2(vm, curr) result (x)
  type(vector) :: x
  class(vm_t), intent(in) :: vm
  integer, dimension(:), intent(in) :: curr
  procedure(va_ff), pointer :: load_wf
  select case (curr(1))
  case (ovm_FUSE_VA_FF)
```

```
    load_wf => va_ff
  case default
    load_wf => null()
  end select
  x = load_wf(⟨c1⟩, ⟨c2⟩, &
            vm%conjspinors(curr(5))%v, vm%spinors(curr(6))%v)
end function vec_ff2

function scal_ff(vm, curr) result (x)
  complex(default) :: x
  class(vm_t), intent(in) :: vm
  integer, dimension(:), intent(in) :: curr
  select case (curr(1))
  case (ovm_FUSE_S_FF)
     x = s_ff(⟨c⟩, &
          vm%conjspinors(curr(5))%v, vm%spinors(curr(6))%v)
  case (ovm_FUSE_SP_FF)
     x = sp_ff(⟨c1⟩, ⟨c2⟩, &
          vm%conjspinors(curr(5))%v, vm%spinors(curr(6))%v)
  case default
     x = zero
  end select
end function scal_ff

function scal_g2(vm, curr) result (x)
  complex(default) :: x
  class(vm_t), intent(in) :: vm
  integer, dimension(:), intent(in) :: curr
  select case (curr(1))
  case (ovm_FUSE_S_G2)
     x = ⟨c⟩ * ((⟨p1⟩ * ⟨v2⟩) * &
                (⟨p2⟩ * ⟨v1⟩) - &
                (⟨p1⟩ * ⟨p2⟩) * &
                (⟨v2⟩ * ⟨v1⟩))
  case (ovm_FUSE_S_G2_SKEW)
     x = - phi_vv(⟨c⟩, ⟨p1⟩, ⟨p2⟩, &
                ⟨v1⟩, ⟨v2⟩)
  case default
     x = zero
  end select
end function scal_g2

pure function gauge_sg(vm, curr) result (x)
  type(vector) :: x
  class(vm_t), intent(in) :: vm
  integer, dimension(:), intent(in) :: curr
  select case (curr(1))
  case (ovm_FUSE_G_SG)
     x = ⟨c⟩ * ⟨s1⟩ * ( &
          -((⟨p1⟩ + ⟨p2⟩) * &
             ⟨v2⟩) * ⟨p2⟩ - &
          (-(⟨p1⟩ + ⟨p2⟩) * &
             ⟨p2⟩) * ⟨v2⟩)
  case (ovm_FUSE_G_GS)
```

```
      x = ⟨c⟩ * ⟨s1⟩ * ( &
           -((⟨p1⟩ + ⟨p2⟩) * &
              ⟨v2⟩) * ⟨p2⟩ - &
           (-(⟨p1⟩ + ⟨p2⟩) * &
              ⟨p2⟩) * ⟨v2⟩)
   case (ovm_FUSE_G_SG_SKEW)
      x = - v_phiv(⟨c⟩, ⟨s1⟩, ⟨p1⟩, &
                   ⟨p2⟩, ⟨v2⟩)
   case (ovm_FUSE_G_GS_SKEW)
      x = - v_phiv(⟨c⟩, ⟨s2⟩, ⟨p1⟩, &
                   ⟨p2⟩, ⟨v1⟩)
   case default
      x = [zero, zero, zero, zero]
   end select
 end function gauge_sg
```

Some really tiny ones that hopefully get inlined by the compiler

⟨*OVM Procedure Implementations*⟩+≡

```
 elemental function sgn_coupl_cmplx(vm, j) result (s)
   class(vm_t), intent(in) :: vm
   integer, intent(in) :: j
   complex(default) :: s
   s = isign(1, j) * vm%coupl_cmplx(abs(j))
 end function sgn_coupl_cmplx

 elemental function sgn_coupl_cmplx2(vm, j, i) result (s)
   class(vm_t), intent(in) :: vm
   integer, intent(in) :: j, i
   complex(default) :: s
   if (i == 1) then
      s = isign(1, j) * vm%coupl_cmplx2(i, abs(j))
    else
      s = isign(1, j) * vm%coupl_cmplx2(i, abs(j))
   end if
 end function sgn_coupl_cmplx2

 elemental function int_to_log(i) result(yorn)
   integer, intent(in) :: i
   logical :: yorn
   if (i /= 0) then
     yorn = .true.
   else
     yorn = .false.
   end if
 end function

 elemental function color_factor(num, den, pwr) result (cf)
   integer, intent(in) :: num, den, pwr
   real(kind=default) :: cf
   if (pwr == 0) then
     cf = (one * num) / den
   else
     cf = (one * num) / den * (N_**pwr)
   end if
```

```
  end function color_factor
```

X.21.6 O'Mega Interface

We want to keep the interface close to the native Fortran code but of course one has to hand over the vm additionally

⟨*VM: TBP*⟩+≡
```
    procedure :: number_particles_in => vm_number_particles_in
    procedure :: number_particles_out => vm_number_particles_out
    procedure :: number_color_indices => vm_number_color_indices
    procedure :: reset_helicity_selection => vm_reset_helicity_selection
    procedure :: new_event => vm_new_event
    procedure :: color_sum => vm_color_sum
    procedure :: spin_states => vm_spin_states
    procedure :: number_spin_states => vm_number_spin_states
    procedure :: number_color_flows => vm_number_color_flows
    procedure :: flavor_states => vm_flavor_states
    procedure :: number_flavor_states => vm_number_flavor_states
    procedure :: color_flows => vm_color_flows
    procedure :: color_factors => vm_color_factors
    procedure :: number_color_factors => vm_number_color_factors
    procedure :: is_allowed => vm_is_allowed
    procedure :: get_amplitude => vm_get_amplitude
```

⟨*OVM Procedure Implementations*⟩+≡
```
  elemental function vm_number_particles_in (vm) result (n)
    class(vm_t), intent(in) :: vm
    integer :: n
    n = vm%N_prt_in
  end function vm_number_particles_in

  elemental function vm_number_particles_out (vm) result (n)
    class(vm_t), intent(in) :: vm
    integer :: n
    n = vm%N_prt_out
  end function vm_number_particles_out

  elemental function vm_number_spin_states (vm) result (n)
    class(vm_t), intent(in) :: vm
    integer :: n
    n = vm%N_helicities
  end function vm_number_spin_states

  pure subroutine vm_spin_states (vm, a)
    class(vm_t), intent(in) :: vm
    integer, dimension(:,:), intent(out) :: a
    a = vm%table_spin
  end subroutine vm_spin_states

  elemental function vm_number_flavor_states (vm) result (n)
    class(vm_t), intent(in) :: vm
    integer :: n
    n = vm%N_flavors
```

```
end function vm_number_flavor_states

pure subroutine vm_flavor_states (vm, a)
  class(vm_t), intent(in) :: vm
  integer, dimension(:,:), intent(out) :: a
  a = vm%table_flavor
end subroutine vm_flavor_states

elemental function vm_number_color_indices (vm) result (n)
  class(vm_t), intent(in) :: vm
  integer :: n
  n = vm%N_col_indices
end function vm_number_color_indices

elemental function vm_number_color_flows (vm) result (n)
  class(vm_t), intent(in) :: vm
  integer :: n
  n = vm%N_col_flows
end function vm_number_color_flows

pure subroutine vm_color_flows (vm, a, g)
  class(vm_t), intent(in) :: vm
  integer, dimension(:,:,:), intent(out) :: a
  logical, dimension(:,:), intent(out) :: g
  a = vm%table_color_flows
  g = vm%table_ghost_flags
end subroutine vm_color_flows

elemental function vm_number_color_factors (vm) result (n)
  class(vm_t), intent(in) :: vm
  integer :: n
  n = vm%N_col_factors
end function vm_number_color_factors

pure subroutine vm_color_factors (vm, cf)
  class(vm_t), intent(in) :: vm
  type(OCF), dimension(:), intent(out) :: cf
  cf = vm%table_color_factors
end subroutine vm_color_factors

! pure & ! pure unless OpenMp
function vm_color_sum (vm, flv, hel) result (amp2)
  class(vm_t), intent(in) :: vm
  integer, intent(in) :: flv, hel
  real(default) :: amp2
  amp2 = ovm_color_sum (flv, hel, vm%table_amplitudes, vm%table_color_factors)
end function vm_color_sum

subroutine vm_new_event (vm, p)
  class(vm_t), intent(inout) :: vm
  real(default), dimension(0:3,*), intent(in) :: p
  logical :: mask_dirty
  integer :: hel
  call vm%run (p)
```

```
    if ((vm%hel_threshold .gt. 0) .and. (vm%hel_count .le. vm%hel_cutoff)) then
       call omega_update_helicity_selection (vm%hel_count, vm%table_amplitudes, &
         vm%hel_max_abs, vm%hel_sum_abs, vm%hel_is_allowed, vm%hel_threshold, &
         vm%hel_cutoff, mask_dirty)
       if (mask_dirty) then
          vm%hel_finite = 0
          do hel = 1, vm%N_helicities
             if (vm%hel_is_allowed(hel)) then
                vm%hel_finite = vm%hel_finite + 1
                vm%hel_map(vm%hel_finite) = hel
             end if
          end do
      end if
    end if
  end subroutine vm_new_event

  pure subroutine vm_reset_helicity_selection (vm, threshold, cutoff)
    class(vm_t), intent(inout) :: vm
    real(kind=default), intent(in) :: threshold
    integer, intent(in) :: cutoff
    integer :: i
    vm%hel_is_allowed = .True.
    vm%hel_max_abs = 0
    vm%hel_sum_abs = 0
    vm%hel_count = 0
    vm%hel_threshold = threshold
    vm%hel_cutoff = cutoff
    vm%hel_map = (/(i, i = 1, vm%N_helicities)/)
    vm%hel_finite = vm%N_helicities
  end subroutine vm_reset_helicity_selection

  pure function vm_is_allowed (vm, flv, hel, col) result (yorn)
    class(vm_t), intent(in) :: vm
    logical :: yorn
    integer, intent(in) :: flv, hel, col
    yorn = vm%table_flv_col_is_allowed(flv,col) .and. vm%hel_is_allowed(hel)
  end function vm_is_allowed

  pure function vm_get_amplitude (vm, flv, hel, col) result (amp_result)
    class(vm_t), intent(in) :: vm
    complex(kind=default) :: amp_result
    integer, intent(in) :: flv, hel, col
    amp_result = vm%table_amplitudes(flv, col, hel)
  end function vm_get_amplitude
```

⟨*Copyleft*⟩≡

```
  !  $Id: omegalib.nw 6465 2015-01-10 15:22:31Z jr_reuter $
  !
  !  Copyright (C) 1999-2009 by
  !      Wolfgang Kilian <kilian@physik.uni-siegen.de>
  !      Thorsten Ohl <ohl@physik.uni-wuerzburg.de>
  !      Juergen Reuter <juergen.reuter@desy.de>
  !
  !  WHIZARD is free software; you can redistribute it and/or modify it
```

```
!  under the terms of the GNU General Public License as published by
!  the Free Software Foundation; either version 2, or (at your option)
!  any later version.
!
!  WHIZARD is distributed in the hope that it will be useful, but
!  WITHOUT ANY WARRANTY; without even the implied warranty of
!  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
!  GNU General Public License for more details.
!
!  You should have received a copy of the GNU General Public License
!  along with this program; if not, write to the Free Software
!  Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

—Y— Index

This index has been generated automatically and might not be 100%ly accurate. In particular, hyperlinks have been observed to be off by one page.

*, **27**, **27**, **19**, used: 826, 826, 828, 789, 790, 790, 792, 795, 795, 812, 27, 29, 30, 30, 31, 32, 32, 32, 32, 47, 56, 62, 86, 86, 565, 188, 189, 96, 108, 112, 122, 263, 267, 484, 487, **??**, **??**, 519, 522, 555, 559, 294, 315, 330, 331, 342, 361, 373, 383, 394, 399, 401, 424, 441, **??**, **??**

+, **27**, **27**, **19**, used: **??**, **??**, **??**, **??**, 756, 756, 743, 743, 746, 753, 826, 828, 790, 790, 790, 793, 793, 794, 27, 29, 29, 30, 30, 30, 31, 32, 32, 32, 33, 47, 56, 57, 60, 62, 63, 65, 181, 181, 84, 86, 700, 565, 568, 572, 574, 211, 97, 98, 116, 120, 122, 122, 124, 709, 715, 263, 484, 487, **??**, **??**, 519, 522, 555, 559, 294, 315, 330, 331, 342, 361, 373, 383, 394, 399, 405, 424, 441, **??**, **??**

−, **27**, **27**, **19**, used: **??**, 756, 749, 751, 752, 752, 797, 797, 798, 727, 727, 834, 835, 826, 790, 791, 802, 803, 818, 21, 27, 29, 29, 30, 30, 31, 31, 31, 33, 33, 56, 57, 57, 60, 61, 62, 62, 63, 65, 66, 66, 86, 568, 569, 569, 570, 575, **??**, 186, 211, 211, 214, 111, 124, 131, 708, 709, 237, 263, 268, **??**, **??**, **??**, 496, 497, 527, 527, 428, 70

/, **27**, **27**, **19**, used: 826, 826, 790, 790, 790, 791, 21, 24, 24, 26, 27, 31, 32, 74, 237, **??**, **??**, 428

//, **240**, **453**, **??**, **??**, **500**, **531**, **277**, **300**, **321**, **335**, **348**, **366**, **377**, **387**, **400**, **413**, **428**, **??**, **??**, used: 240, 240, 240, 240, 241, 454, 454, 454, 454, 454, **??**, **??**, 500, 500, 501, 501, 501, 531, 532, 532, 277, 278, 278, 300, 300, 300, 321, 321, 321, 335, 336, 336, 336, 336, 348, 348, 348, 349, 349, 366, 366, 366, 367, 367, 377, 377, 377, 378, 378, 387, 387, 387, 388, 388, 401, 401, 401, 401, 413, 413, 414, 414, 414, 428, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**

<, **27**, **27**, **19**, used: 756, 757, 757, 757, 758, 758, 758, 759, 761, 762, 762, 762, 763, 763, 764, 764, 742, 745, 746, 746, 752, 826, 789, 789, 790, 791, 791, 792, 794, 794, 795, 813, 818, 818, 24, 27, 29, 30, 31, 31, 33, 45, 55, 56, 56, 57, 57, 60, 86, 568, 569, 70, 184, 192, 193, 193, 197, 198, 198, 199, 199, 200, 204, 205, 214, 215, 95, 98, 125, 708, 260, 261, 262, **??**, **??**, **??**, **??**, **??**, **??**, 497, 527, 438, 439, 440, 719, 719, 719, 720, 721, 722, 722, 723

<<, **707**, used: 710, 710

<=, **27**, **27**, **19**, used: **??**, 742, 742, 742, 742, 743, 743, 791, 792, 802, 803, 812, 816, 27, 28, 47, 48, 55, 56, 61, 63, 693, 575, 74, 107, 124, 131, 709, 448, 448, 498, 499, 530, 530, 402, 409

<>, **27**, **27**, **19**, used: **??**, 761, 744, 745, 746, 751, 752, 752, 797, 797, 798, 798, 772, 774, 835, 836, 826, 829, 793, 793, 794, 812, 816, 817, 11, 12, 13, 16, 27, 42, 56, 62, 64, 64, 65, 218, 219, 699, 701, 568, 568, 569, 193, 200, 200, 204, 213, 213, 213, 216, 102, 102, 130, 130

=, **27**, **27**, **19**, used: **??**, **??**, **??**,

??, **??**, **??**, **??**, 757, 757, 757, 758, 758, 758, 759, 759, 760, 761, 762, 762, 762, 763, 763, 763, 764, 764, 744, 744, 745, 745, 745, 746, 753, 738, 770, 770, 773, 826, 826, 828, 829, 829, 790, 791, 791, 792, 794, 794, 795, 795, 809, 810, 810, 811, 811, 817, 818, 11, 12, 27, 29, 30, 30, 31, 31, 31, 32, 55, 56, 56, 56, 57, 57, 57, 57, 57, 58, 58, 58, 58, 58, 58, 59, 59, 59, 59, 60, 62, 62, 62, 62, 63, 63, 63, 64, 64, 64, 64, 64, 64, 64, 64, 64, 65, 66, 694, 181, 181, 84, 85, 86, 86, 218, 219, 219, 565, 565, 565, 565, 565, 569, 569, 570, 570, 573, 575, 76, 77, 78, 192, 192, 192, 193, 194, 195, 195, 196, 196, 197, 197, 197, 198, 198, 198, 200, 201, 202, 205, 214, 95, 104, 104, 119, 119, 125, 131, 271, 271, 324, 325, 408, 721

>, **27**, **27**, **19**, used: **??**, 756, 756, 743, 743, 749, 751, 752, 753, 732, 733, 835, 790, 790, 791, 792, 792, 794, 802, 803, 803, 17, 21, 27, 30, 55, 56, 56, 59, 59, 60, 62, 64, 64, 64, 64, 65, 66, 568, 570, 186, 192, 192, 193, 197, 197, 198, 199, 199, 199, 204, 205, 211, 211, 212, 237, 238, 239, 240, 240, 241, 260, 261, 262, 263, 446, 448, 450, 454, 454, 454, 481, 484, 487, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, 496, 497, 499, 500, 501, 501, 501, 518, 519, 522, 527, 527, 530, 531, 531, 532, 532, 554, 555, 559, 275, 277, 277, 278, 278, 293, 294, 298, 299, 300, 300, 300, 315, 315, 320, 320, 321, 321, 321, 328, 329, 330, 330, 331, 334, 335, 336, 336, 336, 342, 342, 345, 347, 348, 349, 349, 360, 361, 364, 365, 366, 367, 367, 372, 373, 375, 377, 377, 378, 378, 383, 383, 385, 387, 387, 388, 388, 393, 394, 412, 413, 414, 414, 414, 424, 424, 428, 438, 439, 440, 441, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, 719, 719, 719, 720, 722, 722, 723

>:, **??**, **??**, used:

>::, **??**, **??**, used: 765, 799, 799, 799, 799, 799, 800, 800, 800

>:::, **??**, **??**, used: 766, 766, 799, 800, 800, 800, 800

>=, **27**, **27**, **19**, used: 756, 756, 759, 760, 746, 751, 752, 836, 790, 803, 27, 29, 57, 57, 62, 63, 568, 569, 569, 575, 98, 110, 124, 709, 237, 238, 448, 448, **??**, **??**, 498, 499, 530, 530, 275, 276, 298, 298, 319, 320, 333, 334, 345, 346, 364, 364, 375, 376, 385, 386, 402, 411, 412, 428, **??**, **??**, **??**, **??**

>>, **707**, used:

?*arg_specs* (label), **??**, **??**, used:

?*cmp* (label), **??**, **744**, **747**, **??**, **740**, **742**, used:

?*env* (label), **??**, **??**, used:

?*epsilon* (label), **??**, **??**, used:

?*exit_code* (label), **??**, **??**, used:

?*f* (label), **28**, **31**, **20**, used:

?*foutput* (label), **??**, **??**, used:

?*mode* (label), **??**, **??**, used: **??**

?*msg* (label), **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, used: **??**, **??**, **??**, **??**

?*pp_diff* (label), **??**, **??**, used:

?*prefix* (label), **??**, **??**, used:

?*printer* (label), **??**, **??**, used:
?*set_verbose* (label), **??**, **??**, used:
?*sinput* (label), **??**, **??**, used:
?*skip* (label), **??**, **??**, used:
?*stride* (label), **743**, **743**, **740**, **740**, used:
?*suffix* (label), **??**, **??**, used:
?*use_stderr* (label), **??**, **??**, used:

?*verbose* (label), **??**, **??**, **??**, **??**, used:
@?, **??**, **??**, used:
A, **447**, **397**, **137**, used: 447, 449, 457, 457, 458, 459, 462, 463, 464, 467, 467, 467, 468, 469, 471, 471, 472, 472, 473, 473, 476, 479, 398, 403, 403, 404, 405, 405, 407
A (module), **730**, **104**, used: 730, 730, 730, 730, 104, 104, 104, 105, 108, 108, 109, 110, 110, 111, 111, 111, 111, 111, 111, 112, 112, 112, 112, 116, 116, 117, 117, 117, 117, 117, 117, 118, 118, 118, 119, 120, 120, 121, 121, 121
abs, **54**, **56**, **62**, **51**, used: 826, 697, 186, 229, 229, 233, 233, 262, 263, 264, 446, 481, 487, 487, **??**, **??**, **??**, **??**, **??**, **??**, 497, 498, 518, 522, 522, 527, 529, 554, 560, 560, 293, 294, 294, 315, 316, 316, 328, 329, 330, 331, 331, 342, 343, 343, 360, 362, 362, 372, 373, 373, 383, 383, 384, 393, 394, 394, 424, 425, 425, 440, 441, 441, **??**, **??**, **??**, **??**, **??**, **??**
actions (field), **730**, used: 730, 730, 730, **??**, **??**
AD, **450**, used:
add, **755**, **759**, **762**, **781**, **781**, **782**, **782**, **769**, **769**, **770**, **772**, **773**, **826**, **826**, **827**, **828**, **830**, **831**, **30**, **43**, **44**, **48**, **54**, **56**, **62**, **181**, **181**, **181**, **181**, **181**, **182**, **565**, **754**, **780**, **780**, **767**, **768**, **768**, **823**, **824**, **825**, **51**, **180**, **564**, used: **??**, **??**, 765, 761, 746, 749, 781, 781, 782, 785, 785, 785, 786, 770, 770, 773, 773, 828, 830, 830, 830, 830, 830, 831, 831, 831, 831, 831, 730, 816, 816, 29, 30, 30, 30, 44, 44, 44, 45, 45, 46, 46, 48, 181, 181, 181, 182, 220, 220, 220, 220, 221, 699, 700, 700, 700, 702, 565, **??**, 575, 74, 186, 214, 215, 216, 103, 104, 107, 108, 111, 116, 117, 119, 120, 120, 120, 123, 124, 124, 131, 132, 133, 567
add′, **56**, used: 56, 56
add1, **565**, used: 565, 565, 565
addn, **565**, used: 565, 565
addto_ketset, **120**, used: 120
addto_ketset_map, **120**, used: 120
add_complex_array_tag, **416**, used: 416
add_degree, **29**, used: 29
add_derivatives, **830**, used: 830, 830
add_flavors, **700**, used: 700
add_fusion2, **220**, used: 220
add_fusion3, **220**, used: 221
add_fusionn, **220**, used:
add_node, **41**, **45**, **37**, used: 45, 47, 111, 118
add_non_empty, **770**, **773**, used: 770, 773
add_offspring, **41**, **45**, **110**, **37**, used: 110, 110, 118
add_offspring_unsafe, **41**, **45**, **37**, used: 45, 47
add_permute3, **220**, used: 220
add_permute4, **221**, used: 221
add_to_list, **124**, used: 124
add_tree, **699**, used: 699
add_unique, **816**, used: 817
add_vertex3, **220**, used: 221

add_vertex4, **221**, used: 221
add_vertices, **104**, used: 104
AdjSUN, **81**, **80**, used: 81, 574, 230, 238, 448, ??, ??, 499, 530, 275, 298, 320, 334, 345, 364, 375, 385, 399, 412, 428, ??, ??, 719
AH, **274**, **297**, used: 275, 277, 281, 281, 282, 283, 284, 285, 285, 286, 287, 291, 298, 299, 303, 303, 304, 305, 305, 306, 307, 307, 308, 312
Al, **450**, used:
AL, **450**, used:
Algebra (module), **826**, **823**, used: 181, 240, 453, ??, ??, 500, 531, 277, 300, 321, 335, 348, 366, 377, 387, 400, 413, 428, ??, ??, 181, 181
All, **178**, used: 713
allowed, **211**, **94**, **121**, **90**, used: 212, 213, 121
allows_fusion, **696**, used: 696
allows_momentum_fusion, **696**, used: 696
all_bosons, **398**, used: 406
all_compatible, **74**, used: 76, 76
all_feynman, **396**, used: 396, 400
all_leptons, **398**, used: 406
all_quarks, **398**, used: 406
alpha (camllex regexpr), **566**, used: 566, 566
alphanum (camllex regexpr), **566**, used: 566
Alpha_QED, **241**, **450**, ??, ??, **278**, **300**, **321**, **337**, **349**, **367**, **378**, **388**, **414**, **428**, ??, ??, used: 243, 244, ??, ??, 428
Alpha_WWWW0, **241**, ??, ??, **428**, used: 249, ??, ??, 428
Alpha_WWWW2, **241**, ??, ??, **428**, used: 249, ??, ??, 428
Alpha_ZZWW0, **241**, ??, ??, **428**, used: 249, ??, ??, 428
Alpha_ZZWW1, **241**, ??, ??, **428**, used: 249, ??, ??, 428
Alpha_ZZZZ, **241**, ??, ??, **428**, used: 249, ??, ??, 428
AltH (module), ??, ??, used: ??
alt_fuse2, **271**, used:
alt_fuse3, **271**, used:
amplitude, **207**, **208**, **116**, **178**, **178**, used: 208, 121, 121
amplitude (type), **698**, **699**, **94**, **100**, **103**, **122**, **127**, **128**, **698**, **89**, **91**, used: 698, 699, 94, 101, 122, 127, 128, 128, 129, 129, 698, 698, 89, 89, 90, 90, 90, 90, 90, 91, 91, 92, 92, 92, 92, 93
Amplitude (module), **101**, used: 104, 117
Amplitude (sig), **100**, used: 101
amplitudes, **94**, **121**, **127**, **132**, **89**, **92**, used: 133, 715, 717
amplitudes (type), **127**, **129**, **178**, **91**, used: 127, 178, 179, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92
amplitudes′ (type), **129**, used:
amplitudes_by_flavor, **708**, used: 712, 712, 712
amplitudes_to_channel, **178**, used: 715, 716
amplitudes_to_feynmf, **712**, used: 717
amplitudes_to_feynmf_color_only, **712**, used: 717
amplitudes_to_feynmf_sans_color, **712**, used: 717
amplitude_to_dot, **94**, **125**, **91**, used: 716
analyze_tree, **702**, used:
And, **69**, **74**, **68**, used: 69, 75
AND, ??, ??, used: ??, 70
AND (camlyacc token), **71**, used: 71, 71
angle (type), **450**, used: 450
anomalous_bbA, **254**, ??, ??, **433**, used: 258, ??, ??, 436

anomalous_bbWW, **256**, ??, ??, **435**, used: 258, ??, ??, 436
anomalous_bbZ, **254**, ??, ??, **433**, used: 258, ??, ??, 436
anomalous_gauge_higgs, **252**, **339**, **352**, **369**, **430**, used: 253, 430
anomalous_gauge_higgs4, **253**, **339**, **352**, **369**, **430**, used: 253, 430
anomalous_higgs, **253**, **339**, **352**, **369**, **430**, used: 253, 431
anomalous_higgs4, **253**, **339**, **352**, **369**, **430**, used: 253, 431
anomalous_quartic_gauge, **249**, ??, ??, **428**, used: 251, ??, ??, 428
anomalous_tbW, **255**, ??, ??, **433**, used: 258, ??, ??, 436
anomalous_tbWA, **255**, ??, ??, **434**, used: 258, ??, ??, 436
anomalous_tbWZ, **256**, ??, ??, **434**, used: 258, ??, ??, 436
anomalous_top_DuDd, **257**, ??, ??, **436**, used: 258, ??, ??, 436
anomalous_top_qBuB, **257**, ??, ??, **435**, used: 258, ??, ??, 436
anomalous_top_qBuB_ff, **257**, ??, ??, **435**, used: 257, ??, ??, 435
anomalous_top_qBuB_tt, **257**, ??, ??, **435**, used: 257, ??, ??, 435
anomalous_top_qGuG, **256**, ??, ??, **435**, used: 258, ??, ??, 436
anomalous_top_qGuG_ff, **256**, ??, ??, **435**, used: 256, ??, ??, 435
anomalous_top_qGuG_tt, **256**, ??, ??, **435**, used: 256, ??, ??, 435
anomalous_top_quqd1, **258**, ??, used: 258, ??
anomalous_top_quqd1_ff, **258**, ??, used: 258, ??
anomalous_top_quqd1_tq, **257**, ??, used: 258, ??
anomalous_top_quqd8, **258**, ??, used: 258, ??
anomalous_top_quqd8_ff, **258**, ??, used: 258, ??
anomalous_top_quqd8_tq, **258**, ??, used: 258, ??
anomalous_top_qW, **257**, ??, ??, **436**, used: 258, ??, ??, 436
anomalous_top_qW_ff, **257**, ??, ??, **436**, used: 257, ??, ??, 436
anomalous_top_qW_tq, **257**, ??, ??, **435**, used: 257, ??, ??, 436
anomalous_triple_gauge, **247**, ??, ??, **428**, used: 248, ??, ??, 428
anomalous_ttA, **253**, ??, ??, **432**, used: 258, ??, ??, 436
anomalous_ttG, **254**, ??, ??, **433**, used: 258, ??, ??, 436
anomalous_ttGG, **255**, ??, ??, **434**, used: 258, ??, ??, 436
anomalous_ttH, **255**, **434**, used: 258, 436
anomalous_ttWW, **256**, ??, ??, **435**, used: 258, ??, ??, 436
anomalous_ttZ, **254**, ??, ??, **433**, used: 258, ??, ??, 436
anomaly_higgs, **290**, **311**, **326**, **339**, **352**, **369**, **380**, **391**, used: 291, 312, 327, 340, 353, 370, 380, 391
Anomtop (module), ??, used: ??
anom_ferm_ass, **273**, **273**, **273**, **274**, **272**, used: 321, 323, 324, 325, 325, 327, 328, 329, 330

AntiFermion, **95**, **125**, used: 95, 96, 125, 126
AntiMuon, **227**, used: 228, 228, 229, 229
AntiTau, **227**, used: 228, 228, 229, 229
any (type), **210**, **211**, **209**, used: 210, 211, 209
Any, **210**, **211**, **209**, used: 211, 212
Any_flavor, **69**, **74**, **68**, used: 69, 75
Application, **564**, **564**, used: 564
apply, **765**, **765**, **564**, **764**, **564**, used: 765, ??, 567
apply_invalid_lengths, **799**, used: 800
apply_ok, **765**, **799**, used: 766, 800
arguments, **703**, used: 703, 703
Arguments, **178**, used: 713
ariadne_sort, **747**, **742**, used: 797, 216
ariadne_unsort, **747**, **742**, used:
arity, **9**, **10**, **12**, **13**, **16**, **6**, used:
array, **796**, **797**, **798**, **796**, used:

array_list, **416**, used: 416
assert_bool, **??**, **??**, used: ??
assert_command, **??**, **??**, used:
assert_equal, **??**, **??**, **??**, **??**, **??**, **??**, **??**, used: ??, ??, ??, ??, ??, 765, 799, 799, 800, 800, 800
assert_failure, **??**, **??**, used: ??, ??, ??, ??, ??
assert_raises, **??**, **??**, used: 799, 799, 799
assert_string, **??**, **??**, used:
Atan2, **166**, used:
atom, **827**, **827**, **828**, **828**, **830**, **831**, **824**, **824**, **825**, used: 830
Atom, **166**, used: 244, 245, 245, ??, ??, 280, 280, 302, 302, 322, 322, 337, 337, 350, 350, 367, 367, 378, 378, 389, 389, 416, 416, 428, ??, ??
atoms, **827**, **827**, **828**, **829**, **830**, **832**, **824**, **825**, **825**, used: 830, 832
Atpsi, **278**, **300**, **321**, used:
attach_to_fst_sorted, **792**, used: 793
attach_to_fst_unsorted, **792**, used: 793, 793, 793
AU, **450**, used:
augment_basis, **786**, used: 786
augment_basis_overlapping, **786**, used: 786
author, **727**, **726**, used: 728
author (field), **727**, **726**, used: 728, 802, 9, 20, 40, 54, 218, 698, 568, 183, 93, 224, 444, ??, 495, 526, 273, ??, 678, ??, 718
Aux_Col_Scalar, **137**, used: 184
Aux_Col_Tensor_1, **137**, used: 184
Aux_Col_Vector, **137**, used: 184
Aux_ConjSpinor, **137**, used:
Aux_DScalar_DScalar, **138**, used: 188
Aux_Gauge_Gauge, **138**, used: 188, 255, 255, 256, 256, 256, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, 434, 434, 434, 435, 435
Aux_Majorana, **137**, used:
Aux_Scalar, **137**, used: 238, ??, ??, 428
Aux_Scalar_Scalar, **138**, used: 188
Aux_Scalar_Vector, **138**, used: 188
Aux_Spinor, **137**, used:
Aux_Tensor_1, **137**, used: 238, ??, ??, 428
Aux_top, **236**, **??**, **??**, **428**, used: 237, 239, 255, 255, 256, 256, 256, 256, 256, 257, 257, 257, 257, 257, 257, 258, 258, 258, 259, ??, 428, 434, 434, 434,

435, 435, 435, 435, 435, 435, 435, 436, 436, 437
aux_top_flavors, **237**, ??, ??, **428**, used: 237, 237, ??, ??, 428
Aux_Vector, **137**, used: 238, ??, ??, 428
Aux_Vector_DScalar, **138**, used: 188
Aux_Vector_Vector, **138**, used: 188
A_0, **450**, used:
A_P, **450**, used:
B, **230**, used: 230, 230, 231, 232
B1, **344**, used: 344, 347, 353
B2, **344**, used: 344, 347, 353
bag (type), **211**, used: 211
bal, **756**, used: 756, 757, 758, 758, 759
baryon, **241**, **454**, ??, ??, **501**, **532**, **278**, **300**, **321**, **336**, **349**, **367**, **378**, **388**, **414**, **428**, ??, ??, used: 241, 454, ??, ??, 501, 532, 278, 300, 321, 336, 349, 367, 378, 388, 414, 428, ??, ??
baryonic_vertex, **184**, used: 193, 193, 196
base, **781**, **781**, **782**, **782**, **780**, **780**, used: 782, 215
base (type), **781**, **781**, **781**, **782**, **782**, **213**, **216**, **118**, **779**, **780**, **780**, used: 781, 781, 781, 781, 782, 782, 782, 779, 780, 780, 780, 780, 780, 780, 780, 780
basis, **784**, **786**, **784**, used: 786, 78, 215
Bbar, **230**, used: 230, 230, 231, 232
BBB, **138**, used: 188, 461, 461, 507, 507, 538, 539, 540, 540
BBWW, **236**, ??, ??, **428**, used: 237, 256, 259, ??, ??, ??, ??, ??, ??, 428, 435, 437
bcdi_of_flavor, **398**, used: 409, 409
Be, **450**, used:
begin_step, **734**, **732**, used: 133
Binary (module), **10**, **21**, **125**, **8**, **19**, **91**, used: 15, 22, 23, 33, 34, 125, 127, 718, ??, 19, 20
Binary (sig), **10**, **8**, used: 8
Binary_Majorana (module), **127**, **91**, used:
binomial, **790**, **787**, used: 86
binomial′, **790**, used: 790, 790
bits, **61**, used: 62, 62, 62, 62, 63, 63, 63, 64, 64, 65
Bits (module), **61**, **53**, used: 66, 66, 67, 53
bits0, **61**, used: 61
BitsW (module), **66**, **53**, used: 66, 67
boson (type), **397**, **137**, used: 397, 138
Boson, **95**, **125**, **397**, used: 95, 96, 125, 126, 397, 400, 403, 403, 403, 403, 404, 404, 404, 404, 404, 405, 405, 405, 405, 405, 405, 406, 407, 408
boson2 (type), **137**, used: 139
Bound (sig), **17**, **8**, used: 24, 33, 127, 127, 8, 19, 20, 91, 91
bounded_power, **16**, used: 17
bounded_power_fold, **16**, used: 16, 17
bra, **94**, **100**, **103**, **122**, **89**, used: 122
bracket, ??, ??, used: ??
bracket_tmpfile, ??, ??, used: ??
braket (type), **94**, **100**, **103**, **122**, **89**, used: 94, 100, 100, 100, 101, 103, 122, 89, 90
brakets, **94**, **101**, **103**, **122**, **90**, used: 122
brakets (field), **100**, **103**, used: 103, 116, 120, 121, 122, 122, 123, 123, 125, 125
Branch, **693**, used: 693, 693, 693, 694
Broken_Gauge (sig), **177**, used:
broken__is_unit, **829**, used:
BRS, **135**, used: 570
BSM_anom (module), **274**, **272**, used: ??

BSM_bsm (module), **273**, **272**, used: ??, ??, ??, ??, ??, ??, ??, ??, ??
BSM_flags (sig), **273**, **272**, used: 273, 273, 274, 274, 297, 319, 332, 343, 363, 374, 384, 272
BSM_ungauged (module), **273**, **272**, used: ??
buff_printf, ??, used: ??, ??
build_forest, **47**, used: 47, 48
bundle (field), **118**, used: 118, 118, 118, 119, 119, 119
Bundle (module), **781**, **779**, used: 213, 216, 118
bundle_to_strings, **214**, used:
by_color, **213**, used: 213, 215, 216, 216, 216
c (type), **218**, **218**, **218**, **222**, **571**, **575**, **225**, **226**, **228**, **231**, **245**, **266**, **269**, **455**, ??, ??, **503**, **534**, **280**, **302**, **323**, **337**, **350**, **368**, **379**, **389**, **406**, **416**, **428**, ??, ??, **720**, **217**, **217**, **217**, used: 218, 218, 218, 219, 222, 217, 217, 217
C, **230**, used: 230, 230, 231, 232
C (module), **827**, **828**, **829**, **831**, **84**, **183**, **112**, **128**, **708**, **824**, **825**, **578**, used: 827, 828, 86, 112, 116, 117, 133, 134, 715, 717, 824, 825, 825, 825, 825
C1, **446**, **497**, **528**, used: 446, 447, 458, 458, 462, 463, 476, 477, 478, 479, 497, 498, 506, 513, 514, 528, 529, 537, 548, 549
C1c, **446**, **497**, **528**, used: 446, 447, 462, 479, 497, 498, 514, 528, 529, 549
C2, **446**, **497**, **528**, used: 446, 447, 458, 458, 462, 463, 476, 477, 478, 479, 497, 498, 506, 513, 514, 528, 529, 537, 548, 549
C2 (module), **85**, used: 86
C2c, **446**, **497**, **528**, used: 446, 447, 462, 479, 497, 498, 514, 528, 529, 549
C3, **528**, used: 528, 529, 549
C3c, **528**, used: 528, 529, 549
C4, **528**, used: 528, 529, 549
C4c, **528**, used: 528, 529, 549
CA (module), **117**, used: 117, 117, 118, 118, 118, 118, 118, 119, 119, 119, 119, 120, 120, 120, 120, 121, 121, 121, 121, 121, 121, 121, 122, 122, 122, 122, 122, 123, 123, 123, 123, 123, 123, 124, 124, 124, 124, 124, 125, 125, 125
Cache (module), **737**, **736**, used: 105
Cache_Default, **713**, used: 713
Cache_Ignore, **98**, used: 98
Cache_Initialize, **713**, used: 713
cache_mode (type), **98**, **713**, used:
cache_name, **105**, used: 105, 105, 105
cache_option, **98**, **713**, used: 98, 105, 713, 715
Cache_Overwrite, **98**, used: 98
cache_prefix, **729**, **729**, used: 105
cache_suffix, **729**, **729**, used: 105
Cache_Use, **98**, used: 98
call_function, **703**, used: 703
call_subroutine, **703**, used: 703
canonicalize, **755**, **761**, **764**, **809**, **754**, **806**, used: 809, 709
cascade (camlyacc non-terminal), **71**, used: 71
Cascade (module), **73**, **72**, used: 95, 112, 128, 708, 91
cascades (camlyacc non-terminal), **71**, used: 71, 71
Cascade_lexer (module), ??, ??, **70**, used: 75
Cascade_parser (module), ??, ??, **71**, used: ??, 75, ??, 70
Cascade_syntax (module), **68**, **68**, used: ??, ??, 74, ??, 71, 71
Cbar, **230**, used: 230, 230, 231, 232

CF (module), **708**, used: 708, 713, 715, 715, 716, 716, 716, 716, 716
CF_aux, **184**, used: 185, 186, 186, 193, 193, 200, 200, 200, 205
cf_aux_propagator, **184**, used: 185
cf_in (type), **184**, used: 184
CF_in, **184**, used: 185, 186, 186, 192, 192, 197, 197, 198, 199, 199, 205
CF_io, **184**, used: 185, 186, 186, 193, 193, 194, 200, 200, 200, 201, 202, 202, 205
cf_out (type), **184**, used: 184
CF_out, **184**, used: 185, 186, 186, 192, 193, 197, 198, 198, 199, 199, 205
Ch (module), **222**, **571**, **184**, **207**, **225**, **226**, **228**, **231**, **240**, **266**, **269**, **453**, ??, ??, **500**, **531**, **277**, **300**, **321**, **335**, **348**, **366**, **377**, **387**, **400**, **413**, **428**, ??, ??, **720**, **172**, used: 571, 184, 207, 211, 122, 267, 269, 172, 224
channel, **733**, **732**, used: 133
channel (field), **732**, used: 733, 733
channel (type), **732**, used: 732
Channel, **732**, **128**, used: 733, 128
Chaotic (module), **43**, **39**, used:
char (type), **446**, **497**, **528**, used:
char (camllex regexpr), **70**, used:
charge, **240**, **454**, ??, ??, **501**, **531**, **277**, **300**, **321**, **336**, **348**, **366**, **377**, **387**, **414**, **428**, ??, ??, used: 241, 454, ??, ??, 501, 532, 278, 300, 321, 336, 349, 367, 378, 388, 414, 428, ??, ??
charged_chargino_currents, **456**, **504**, **535**, used: 477, 513, 548
charged_currents, **456**, **504**, **535**, **282**, **303**, **325**, **338**, **350**, **368**, **379**, **389**, **417**, ??, ??, used: 477, 513, 548, 291, 312, 327, 340, 353, 370, 380, 391, 422, ??, ??
charged_currents′, **246**, ??, ??, **428**, ??, used: 246, 246, ??, ??, 428, ??
charged_currents″, **246**, ??, ??, **428**, used: 246, ??, ??, 428
charged_currents_ckm, **246**, ??, ??, **428**, used: 258, ??, ??, 436
charged_currents_triv, **246**, ??, ??, **428**, used: 258, ??, ??, 436
charged_heavy_currents, **282**, used: 291
charged_quark_currents, **456**, **504**, **535**, used: 477, 513, 548
charged_slepton_currents, **456**, **504**, **535**, used: 477, 513, 548
charged_squark_currents, **456**, **504**, **536**, used: 477, 513, 548
charged_squark_currents′, **456**, **504**, **536**, used: 456, 504, 536
charged_up_currents, ??, used: ??
charged_Z, **457**, **505**, **536**, used: 477, 513, 548
charges, **222**, **571**, **184**, **207**, **225**, **226**, **228**, **231**, **241**, **266**, **269**, **454**, ??, ??, **501**, **532**, **278**, **300**, **321**, **336**, **349**, **367**, **378**, **388**, **401**, **414**, **428**, ??, ??, **720**, **172**, used: 571, 184, 207, 211, 122, 266, 269
Charges (module), **181**, **180**, used: 222, 225, 226, 228, 231, 240, 266, 453, ??, ??, 500, 531, 277, 300, 321, 335, 348, 366, 377, 387, 400, 413, 428, ??, ??, 720,

172, 224, 224, 224, 224, 444, ??, 495, 525, 272, 272, 273, ??
Chargino, **447**, **498**, **529**, used: 447, 449, 455, 456, 457, 458, 458, 458, 458, 461, 461, 462, 462, 476, 479, 498, 500, 503, 504, 505, 506, 506, 506, 507, 507, 514, 529, 531, 535, 535, 536, 537, 537, 537, 538, 539, 549
charlist, **529**, used: 529
check, **55**, used: 55, 55
check_charges, **94**, **122**, **90**, used: 715
chg, **508**, **539**, used: 539, 539, 540, 540
Chi, **137**, used: 456, 457, 457, 458, 458, 459, 459, 459, 462, 462, 476, 476, 504, 505, 505, 506, 506, 506, 507, 535, 536, 536, 537, 537, 538, 538, 541, 369
Chibar, **137**, used: 456, 457, 457, 458, 458, 459, 459, 459, 462, 504, 505, 505, 506, 506, 506, 507, 535, 536, 536, 537, 537, 538, 538, 541, 541
chiggs (type), **528**, used: 529
CHiggs, **529**, used: 529, 531, 537, 537, 537, 543, 543, 543, 543, 544, 544, 544, 544, 544, 546, 547, 547, 549
children, **94**, **100**, **103**, **121**, **89**, used: 121
children (type), **20**, **22**, **24**, **26**, **26**, **33**, **33**, **41**, **41**, **42**, **42**, **43**, **45**, **94**, **97**, **97**, **97**, **98**, **100**, **103**, **109**, **121**, **18**, **36**, **37**, **40**, **89**, **93**, used: 20, 41, 41, 42, 42, 42, 43, 45, 97, 97, 98, 100, 100, 101, 103, 109, 121, 18, 19, 20, 20, 36, 36, 37, 37, 37, 38, 38, 39, 40, 40, 40, 93, 93
choose, **755**, **758**, **762**, **794**, **754**, **789**, used: 10, 10, 17, 17, 74, 496, 527
choose′, **794**, used: 794, 794
choose2, **10**, **496**, **527**, used: 11, 12, 496, 527
choose3, **10**, used: 12
choose_opt, **755**, **758**, **762**, **754**, used:
chopn, **743**, **740**, used:
chopn′, **742**, used: 742, 743
chopn″, **742**, used: 742, 742
CKetSet (module), **119**, used: 119, 120, 120
CKM_12, **450**, used:
CKM_13, **450**, used:
CKM_23, **450**, used:
ckm_present, **234**, **234**, **234**, **234**, **235**, **235**, **235**, **235**, **236**, **444**, **444**, **444**, **445**, **445**, **445**, **445**, ??, ??, **495**, **495**, **496**, **496**, **526**, **526**, **526**, **410**, **410**, **410**, **427**, **428**, **224**, **443**, ??, **495**, **525**, **272**, **273**, used: 240, 258, 454, 461, 466, 466, 470, 477, 478, ??, ??, ??, ??, 500, 507, 511, 511, 513, 513, 539, 545, 545, 547, 548, 413, 418, 428, 436
classify, **745**, **31**, **741**, used: 790, 793, 793, 17, 17, 32
classify_nodes, **124**, used: 124
class_size, **30**, used: 30, 30, 31
class_size_3, **30**, used: 30
class_size_n, **30**, used: 30
clist, **529**, **402**, used: 529, 407
clone, **745**, **269**, **741**, used: 745, 777, 16, 269, 269
clone3, **269**, used: 270
clone4, **269**, used: 270
clonen, **270**, used: 270
cloop, **496**, **527**, used: 496, 496, 527, 527
cloop_kk, **398**, used: 398, 402, 403, 403, 403, 403, 404, 404, 404, 404
cloop_kk2, **398**, used: 402
close_channel, **733**, used: 734
CM (module), **207**, **117**, **128**, **708**, used: 207, 208, 208,

117, 117, 118, 119, 119, 119, 120, 121, 121, 123, 124, 125, 129, 129, 709, 710, 710, 712
CMap (module), **129**, used: 133, 133
cmdline, **731**, **730**, used: 713
cmp_float, **??**, **??**, used:
coarsest_partition, **78**, **215**, used: 79, 215
coarsest_partition′, **78**, used: 78, 78
collect23, **26**, used: 26
collect3, **26**, used: 26
collect34, **26**, used: 26
collect4, **26**, used: 26
collect_derivatives, **830**, used: 830
collect_vertices, **104**, used: 104
COLON, **??**, **??**, used: **??**, 70
COLON (camlyacc token), **71**, used: 71, 72
color, **222**, **571**, **184**, **207**, **225**, **226**, **228**, **230**, **238**, **266**, **267**, **448**, **??**, **??**, **499**, **530**, **275**, **298**, **320**, **334**, **345**, **364**, **375**, **385**, **399**, **412**, **428**, **??**, **??**, **719**, **172**, used: 571, 184, 186, 186, 191, 195, 203, 204, 205, 205, 205, 207, 213, 129, 710, 710, 266, 267
color (label), **223**, **175**, used: 575
color (type), **83**, **83**, **86**, **80**, used: 83, 83, 87, 80, 80
Color (module), **81**, **80**, used: 571, 572, 572, 573, 574, 183, 207, 213, 127, 129, 129, 129, 134, 134, 134, 225, 226, 228, 230, 238, 448, **??**, **??**, 499, 530, 275, 298, 320, 334, 345, 364, 375, 385, 399, 412, 428, **??**, **??**, 719, 172, 175, 178, 178, 92, 92
colored_only, **712**, used: 712
colored_vertex, **184**, used: 192, 192, 192, 192, 193, 193, 193, 193, 195, 197, 197, 197, 197, 198, 198, 198, 199, 199, 199, 199, 199, 200, 200, 203, 203
Colorize (module), **183**, **183**, used: 95, 97, 117, 128, 708, 91, 93
Colorized (sig), **178**, used: 183
Colorized_Gauge (sig), **178**, used: 183
colorize_amplitude, **120**, used: 121
colorize_amplitudes, **121**, used: 121
colorize_braket, **120**, used: 120
colorize_braket1, **119**, used: 120
colorize_coupling, **119**, used: 119, 119
colorize_crossed_amplitude, **206**, used: 207
colorize_crossed_amplitude1, **205**, **206**, used: 205, 206, 206
colorize_dag, **118**, used: 120
colorize_external, **119**, used: 120
colorize_fusion, **119**, used: 120
colorize_fusion2, **191**, used: 204
colorize_fusion3, **195**, used: 204
colorize_fusionn, **203**, used: 204
colorize_nodes, **118**, used: 118
colorize_sterile_nodes, **118**, used: 118
colorize_wf, **117**, used: 119
colors, **129**, used:
colors (type), **129**, used:
color_current, **231**, used: 232
color_currents, **245**, **??**, **??**, **281**, **303**, **323**, **338**, **351**, **379**, **389**, **417**, **428**, **??**, **??**, used: 258, **??**, **??**, 291, 312, 327, 340, 353, 380, 391, 422, 436, **??**, **??**
color_factors, **127**, **129**, **92**, used:
color_factors (field), **129**, used: 129, 134, 134
color_flow, **129**, used: 133, 133
color_flows, **127**, **129**, **92**, used:
color_flows (field), **129**, used: 129, 134
color_flow_ambiguous, **184**, used: 199, 200

color_flow_of_string, **184**, used: 186
columns, **569**, used: 570
column_tabs, **569**, used: 570
col_currents, **462**, **508**, **539**, used: 477, 513, 548
col_lqino_currents, **542**, used: 548
col_lq_currents, **542**, used: 548
col_sfermion_currents, **462**, **508**, **542**, used: 477, 513, 548
Combinatorics (module), **789**, **787**, used: 10, 10, 17, 17, 24, 24, 27, 33, 34, 86, 74, 204, 96, 117, 127, 496, 527
combine_decays, **696**, used: 696, 697
compare, **??**, **??**, **??**, **??**, **765**, **755**, **761**, **763**, **744**, **746**, **781**, **784**, **769**, **769**, **772**, **730**, **9**, **11**, **12**, **13**, **16**, **27**, **27**, **28**, **40**, **41**, **42**, **42**, **43**, **43**, **43**, **54**, **56**, **62**, **81**, **218**, **218**, **219**, **219**, **699**, **700**, **702**, **575**, **78**, **186**, **214**, **215**, **215**, **215**, **103**, **103**, **104**, **109**, **111**, **116**, **119**, **119**, **123**, **124**, **129**, **129**, **131**, **225**, **226**, **228**, **231**, **245**, **266**, **269**, **455**, **??**, **??**, **503**, **534**, **280**, **302**, **323**, **337**, **350**, **368**, **379**, **389**, **406**, **416**, **428**, **??**, **??**, **720**, **??**, **??**, **754**, **740**, **783**, **767**, **768**, **6**, **19**, **35**, **36**, **40**, **51**, **80**, **217**, used: ??, ??, ??, ??, ??, 765, 763, 744, 746, 747, 747, 749, 753, 797, 797, 797, 820, 820, 828, 829, 830, 830, 831, 832, 730, 793, 793, 809, 816, 816, 817, 16, 27, 28, 42, 43, 45, 47, 48, 55, 56, 62, 694, 696, 81, 218, 219, 219, 699, 700, 700, 701, 702, 575, 78, 186, 213, 213, 214, 215, 215, 215, 216, 216, 96, 102, 102, 103, 104, 109, 110, 116, 118, 119, 124, 127, 129, 129, 130, 130, 131, 133, 709, 714, 225, 226, 228, 231, 245, 266, 269, 455, ??, ??, 503, 534, 280, 302, 323, 337, 350, 368, 379, 389, 406, 416, 428, ??, ??, 720
compare', **761**, used: 761, 761
compare_base, **781**, **213**, **216**, **118**, **779**, used: 781
compare_elt, **781**, **213**, **216**, **118**, **779**, used: 781, 213
Comphep (module), **568**, **568**, used:
Comphep_lexer (module), **??**, **??**, **566**, used: 573
Comphep_parser (module), **??**, **??**, **567**, used: ??, 573, ??, 566
Comphep_syntax (module), **564**, **564**, used: ??, ??, 571, 572, 575, ??, 567, 567
complement_index_sets, **746**, used: 746
complete_fusion_tower, **111**, used: 116
Complex, **171**, used: 244, ??, ??, 428
Complex_Array, **171**, used: 416
compose, **755**, **759**, **764**, **796**, **797**, **798**, **754**, **796**, used: 759, 764, 800, 800
compose_inverse_ok, **800**, used: 800
compose_ok, **800**, used: 800
compress, **749**, **748**, used: 749
compress2, **749**, **748**, used:
compressed (type), **748**, **748**, used: 748, 748
compressed2 (type), **748**, **748**, used: 748, 748
concat, **757**, used: ??, ??, ??, 757, 759, 784, 784, 735, 737, 738, 738, 738, 820, 829, 831, 832, 810, 812, 17, 17, 55, 62, 697, 697, 83, 701, 701, 701, 703, 576, 69, 70, 75, 75, 75, 78, 78, 186, 203, 213, 214,

215, 124, 129, 709, 710, 710, 712, 713, 714, 715, 723
Config (module), **729**, **729**, used: 737, 105
conj, **81**, **84**, **85**, used: 82, 82, 82
Conj, **166**, used:
ConjSpinor, **135**, used: 570, 228, 230, 237, 448, ??, ??, 498, 530, 275, 275, 298, 298, 319, 333, 345, 364, 375, 385, 399, 411, 428, ??, ??, 719
conjugate, **81**, **84**, **218**, **222**, **571**, **575**, **186**, **207**, **94**, **100**, **101**, **121**, **225**, **225**, **226**, **226**, **228**, **228**, **230**, **231**, **239**, **245**, **266**, **266**, **268**, **269**, **449**, **455**, ??, ??, **500**, **503**, **531**, **534**, **277**, **280**, **299**, **302**, **320**, **323**, **335**, **337**, **347**, **350**, **365**, **368**, **376**, **379**, **386**, **389**, **400**, **406**, **412**, **416**, **428**, ??, ??, ??, ??, **720**, **720**, **174**, **80**, **217**, **88**, used: 84, 84, 220, 221, 571, 573, 575, 76, 77, 78, 186, 186, 206, 206, 207, 211, 216, 216, 101, 104, 112, 116, 119, 120, 121, 123, 710, 710, 710, 710, 225, 226, 228, 231, 245, 266, 266, 268, 269, 455, ??, ??, 503, 505, 534, 536, 280, 302, 323, 337, 350, 368, 379, 389, 406, 416, 428, ??, ??, 720
conjugate (label), **223**, **175**, used: 575
Conjugates, **571**, used: 573
conjugate_lorentz, **570**, used: 570, 573
conjugate_sans_color, **186**, **207**, **178**, **178**, used: 207
conj_char, **446**, **497**, **528**, used: 449, 455, 456, 457, 458, 458, 458, 458, 461, 461, 462, 476, 500, 503, 504, 505, 506, 506, 506, 507, 507, 531, 535, 535, 536, 537, 537, 537, 538, 539
conj_chiggs, **528**, used: 531
conj_iso, **402**, used: 403, 403, 404
conj_symbol, **487**, **522**, **560**, used: 487, 522, 560, 561
cons, **755**, **758**, **761**, **820**, **808**, **754**, **820**, **805**, used: 758, 759, 761, 46, 47
Const, **166**, used: 575, 239, 244, 245, 245, 245, 245, 449, ??, ??, 276, 280, 280, 280, 280, 299, 302, 302, 302, 302, 320, 322, 322, 322, 322, 335, 337, 337, 337, 337, 346, 350, 350, 350, 350, 365, 367, 367, 368, 368, 376, 378, 378, 379, 379, 386, 389, 389, 389, 389, 412, 416, 416, 416, 416, 416, 428, ??, ??, ??, ??, ??, ??
constant (type), **222**, **571**, **184**, **207**, **94**, **99**, **225**, **226**, **228**, **230**, **241**, **266**, **268**, **450**, ??, ??, **501**, **532**, **278**, **300**, **321**, **337**, **349**, **367**, **378**, **388**, **401**, **414**, **428**, ??, ??, **719**, **174**, **88**, used: 571, 575, 184, 207, 94, 95, 97, 99, 99, 99, 100, 104, 105, 225, 226, 228, 231, 245, 266, 266, 268, 269, 455, ??, ??, 503, 534, 280, 302, 323, 337, 350, 368, 379, 389, 406, 416, 428, ??, ??, 720, 174, 175, 175, 175, 175, 175, 217, 183, 183, 89, 91, 93
Constant, **137**, used: 575, 236, 445, ??, ??, 496, 526, 274, 297, 319, 333, 344, 363, 374, 384, 396, 410, 428, ??, ??
constant_symbol, **222**, **571**, **188**, **207**, **226**, **227**, **230**, **233**, **264**, **267**, **268**, **487**, ??, ??, **522**, **560**, **294**, **316**, **331**, **343**, **362**, **374**, **384**,

394, 409, 425, 441, ??, ??, 723, 175, used: 571, 188, 207, 267, 268
constant_symbol (label), 223, 175, used: 575
constraints, 94, 101, 103, 122, 127, 129, 90, 92, used: 122
constraints (field), 100, 103, 129, used: 103, 116, 117, 121, 129, 134, 134
consume, 82, used: 82
contract, 81, 82, used: 83, 86
contract4 (type), 139, used: 139, 140
copy_matrix, 833, 833, used: 836, 837, 837
Cos, 450, 166, used:
Cos2al, 450, used:
Cos2am2b, 450, used:
Cos2be, 450, used:
Cos4be, 450, used:
Cosamb, 450, used:
Cosapb, 450, used:
Cospsi, 278, 300, 321, used:
Costhw, 241, ??, ??, 278, 300, 321, 337, 349, 367, 378, 388, 414, 428, ??, ??, used: 244, 245, ??, ??, 280, 302, 322, 337, 350, 367, 378, 389, 416, 428, ??
Cot, 166, used:
count, 82, 83, used: 86
Count (module), 28, 20, used:
Count (sig), 28, 19, used: 20
count_color_strings, 204, used: 204
count_diagrams, 94, 122, 90, used: 716
count_flavors, 572, used: 574
count_fusions, 94, 122, 90, used: 716
count_ghosts, 84, used: 86
count_ghosts1, 84, used: 84
count_maps, 699, used: 699
count_non_zero, 66, used: 66
count_non_zero', 66, used: 66, 66
count_processes, 129, used: 133
count_propagators, 94, 122, 90, used: 716
count_trees, 41, 47, 39, used: 122
coupling, 94, 99, 99, 100, 103, 121, 89, used: 103, 121
coupling (field), 99, 99, used: 99, 108, 112
coupling (type), 94, 97, 97, 97, 98, 100, 103, 121, 89, 93, used: 94, 97, 99, 99, 100, 100, 100, 101, 103, 103, 103, 109, 121, 131, 89, 91, 93
Coupling (module), 135, used: 218, 219, 220, 570, 571, 572, 572, 572, 574, 575, 575, 183, 94, 99, 99, 99, 100, 104, 105, 225, 226, 227, 230, 236, 239, 266, 445, 449, 457, 457, 457, 459, 461, 461, 462, ??, ??, ??, ??, 496, 505, 526, 537, 274, 276, 297, 299, 319, 320, 333, 335, 344, 346, 363, 365, 374, 376, 384, 386, 390, 396, 410, 412, 428, 428, ??, ??, ??, ??, 718, 172, 174, 174, 174, 175, 175, 175, 217, 89
coupling_tag, 94, 99, 99, 100, 103, 121, 89, used: 103, 121
coupling_tag (field), 99, 99, used: 99, 108, 112
coupling_tag_raw, 99, used: 99
coupling_to_string, 97, 97, 97, 98, 93, used: 99
Cpp (module), 578, used:
create, 756, 733, 730, 730, used: ??, ??, 756, 733, 733, 733, 221, 575, 576, 98, 236, 445, ??, ??, 496, 526, 274, 297, 319, 333, 344, 363, 374, 384, 396, 410, 428, ??, ??
created (field), 732, used: 733, 734
create_class, 29, used: 31

CRing (sig), **826**, **823**, used: 826, 823
cross, **123**, used: 123
crossing, **210**, **216**, **210**, used:
Crossing_Bundle (module), **216**, used:
Crossing_Projection (module), **216**, used: 216
cross_in, **84**, used:
cross_out, **84**, used: 85, 85
cross_uncolored, **206**, used: 207
CS (module), **74**, used: 75
csign (type), **397**, used: 397, 397
Custom, **137**, used: 236, 445, ??, ??, 496, 526, 274, 297, 319, 333, 344, 363, 374, 384, 396, 410, 428, ??, ??
CWFBundle (module), **118**, used: 118, 119, 119, 119, 120
CWFMap (module), **119**, used: 119, 120, 120, 120, 121
Cycle (exn), **41**, **45**, **37**, used:
Cycles (module), **82**, used: 84, 85
Cycles (sig), **81**, used: 82
C_12_34, **139**, used: 190, 232, 248, 249, 250, 463, ??, ??, 508, 542, 284, 305, 326, 338, 351, 369, 380, 390, 405, 420, 428, ??, ??, 721
C_13_42, **139**, used: 190, 232, 248, 249, 250, 463, ??, ??, 508, 542, 284, 305, 326, 338, 351, 369, 380, 390, 405, 420, 428, ??, ??, 721
C_14_23, **139**, used: 190, 232, 248, 249, 250, 463, ??, ??, 508, 542, 284, 305, 326, 338, 351, 369, 380, 390, 405, 420, 428, ??, ??, 721
C_qugd1L_bt, **241**, ??, used: 257, ??
C_qugd1L_tb, **241**, ??, used: 257, ??
C_qugd1R_bt, **241**, ??, used: 257, ??
C_qugd1R_tb, **241**, ??, used: 257, ??
C_qugd8L_bt, **241**, ??, used: 258, ??
C_qugd8L_tb, **241**, ??, used: 258, ??
C_qugd8R_bt, **241**, ??, used: 258, ??
C_qugd8R_tb, **241**, ??, used: 258, ??
d (field), **55**, used: 55, 55, 55, 56, 56, 56, 56, 57, 57, 57, 58, 58, 59, 59, 60, 60, 66
D, **230**, **236**, **447**, ??, ??, **498**, **529**, **274**, **297**, **319**, **333**, **344**, **363**, **374**, **384**, **411**, **428**, ??, ??, used: 230, 230, 231, 232, 237, 239, 243, 245, 245, 246, 246, 246, 246, 254, 254, 255, 255, 256, 256, 256, 257, 257, 257, 257, 257, 258, 258, 258, 259, 447, 449, 453, 455, 455, 456, 457, 457, 457, 458, 459, 461, 462, 462, 476, 479, ??, 498, 500, 503, 503, 504, 505, 505, 506, 507, 507, 508, 514, 529, 531, 534, 535, 535, 537, 537, 538, 538, 539, 539, 540, 540, 540, 540, 549, 275, 277, 281, 281, 281, 281, 281, 282, 282, 282, 282, 283, 291, 297, 299, 302, 303, 303, 303, 303, 303, 304, 304, 304, 304, 312, 319, 320, 323, 323, 323, 323, 324, 325, 325, 325, 327, 333, 335, 338, 338, 338, 338, 338, 338, 340, 344, 347, 350, 350, 350, 351, 351, 351, 353, 363, 365, 368, 368, 368, 368, 370, 375, 376, 379, 379, 379, 379, 379, 381, 385, 386, 389, 389, 389, 389, 390, 391, 411, 412, 417, 417, 417, 417, 418, 418, 418, 418, 422, 428, 433, 433,

433, 434, 434, 435, 435, 435, 435, 436, 436, 437, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??
D (module), **782**, **100**, **103**, used: 782, 782, 100, 101, 103, 110, 110, 111, 111, 111, 111, 111, 112, 112, 116, 117, 117, 117, 118, 118, 118, 118, 120, 120, 120, 122, 123, 123, 124, 124, 125
D′ (module), **110**, used: 110
dag (field), **118**, used: 118, 120
DAG (module), **40**, **35**, used: 100, 103, 109, 110
dag_to_dot, **124**, used: 125, 125
data (field), **809**, **810**, used: 809, 809, 811, 811
date, **727**, **726**, used: 728
date (field), **727**, **726**, used: 728, 802, 9, 20, 40, 54, 218, 698, 568, 183, 93, 224, 444, ??, 495, 526, 273, ??, 678, ??, 718
Dbar, **230**, used: 230, 230, 231, 232
debug, **712**, used: 713
decay (type), **692**, **693**, **210**, **211**, **691**, **209**, used: 692, 693, 695, 695, 210, 211, 691, 691, 691, 691, 209, 209
Decay, **210**, **211**, **209**, used: 211, 212, 212
Decay (module), **54**, **59**, **64**, **53**, used: 116
decays (field), **695**, used: 696, 696, 696, 697
decay_of_momenta, **692**, **694**, **691**, used:
declare, **222**, used: 222
declare_argument, **703**, used: 703, 703
decode, **717**, used: 717
Default (module), **798**, **67**, **796**, **53**, used: 707, 707, 717, 53, 707
DefaultW (module), **67**, **53**, used: 707
default_function, **703**, used: 703
default_subroutine, **703**, used: 703
default_width, **236**, **445**, ??, ??, **496**, **526**, **274**, **297**, **319**, **333**, **344**, **363**, **374**, **384**, **396**, **410**, **428**, ??, ??, used: 236, 239, 445, 449, ??, ??, ??, ??, 496, 499, 526, 531, 274, 276, 297, 299, 319, 320, 333, 334, 344, 346, 363, 365, 374, 376, 384, 386, 396, 400, 410, 412, 428, 428, ??, ??, ??, ??
Delta, **450**, used:
den (field), **83**, **84**, **80**, used: 86
dependencies, **41**, **46**, **94**, **101**, **103**, **122**, **38**, **89**, used: 116, 117, 120, 122, 131, 132
dependencies (field), **100**, **103**, used: 103, 116, 117, 121
derive, **827**, **829**, **824**, used: 830, 830, 830
derived (field), **171**, used: 575, 225, 227, 229, 232, 245, 455, ??, ??, 503, 534, 280, 302, 323, 337, 350, 368, 379, 389, 407, 416, 428, ??, ??, 721
derived_arrays (field), **171**, used: 575, 225, 227, 229, 232, 245, 455, ??, ??, 503, 534, 280, 302, 323, 337, 350, 368, 379, 389, 407, 416, 428, ??, ??, 721
derived_parameters, **244**, ??, ??, **280**, **302**, **322**, **337**, **350**, **367**, **378**, **389**, **428**, ??, ??, used: 245, ??, ??, 280, 302, 323, 337, 350, 368, 379, 389, 428, ??, ??
derived_parameter_arrays, **245**, ??, ??, **280**, **302**, **322**, **337**, **350**, **368**, **379**, **389**, **416**, **428**, ??, ??, used: 245, ??, ??, 280, 302, 323,

337, 350, 368, 379, 389, 416, 428, ??, ??
derive_inner, **827**, **830**, **824**, used:
derive_inner′, **827**, **830**, **824**, used:
derive_outer, **827**, **830**, **825**, used: 831
descend, **771**, **771**, **771**, **772**, **774**, **774**, **774**, **774**, used: 771, 771, 771, 772, 774, 774, 774, 774
description, **727**, **28**, **73**, **76**, **726**, **73**, used: 728, 55, 61, 117, 134, 267
description (field), **727**, **76**, used: 727, 76, 76, 79
DH, **447**, used:
diag, **496**, **527**, used: 496, 496, 527, 527
diagnostic (type), **178**, used: 178
diagram, **710**, used: 711
diagrams, **28**, **31**, **707**, **717**, **717**, **20**, **707**, used: 32
diagrams (field), **813**, **807**, used: 814, 815, 711, 711, 711
diagrams_per_keystone, **28**, **32**, **20**, used: 32
diagrams_via_keystones, **28**, **32**, **20**, used:
diagram_class (type), **29**, used:
diagram_sans_color, **710**, used: 711
dictionary, **127**, **129**, **92**, used:
dictionary (field), **129**, used: 129, 134
diet, **395**, **395**, **395**, **395**, **395**, **396**, **396**, **272**, used: 396, 406
diff, **210**, **216**, **210**, used: ??, 746, 785, 216, 133
Diff, **166**, used: 244, 245, ??, ??, 280, 302, 322, 337, 350, 367, 378, 389, 416, 428, ??
DIFF_ELEMENT (sig), ??, ??, used: ??, ??, ??, ??
digit (camllex regexpr), **70**, **566**, used: 70, 566, 566
digits, **732**, used: 733, 734
digits (field), **732**, used: 733, 734, 734
dim, **54**, **55**, **61**, **51**, used: 62, 62, 62, 64, 65, 66, 697, 697, 74
dim0, **61**, used: 61, 62, 62, 62, 63, 63, 63
Dim4_Vector_Vector_Vector_L, **138**, used: 188, 247, ??, ??, 428
Dim4_Vector_Vector_Vector_L5, **138**, used: 188, 247, ??, ??, 428
Dim4_Vector_Vector_Vector_T, **138**, used: 188, 247, ??, ??, 428
Dim4_Vector_Vector_Vector_T5, **138**, used: 188, 247, ??, ??, 428
Dim5_Scalar_Gauge2, **138**, used: 188, 252, 253, 463, 508, 339, 352, 369, 391, 430, 430, 720
Dim5_Scalar_Gauge2_Skew, **138**, used: 188, 463, 508, 290, 311, 326
Dim5_Scalar_Vector_Vector_T, **138**, used: 188
Dim5_Scalar_Vector_Vector_TU, **138**, used: 188, 252
Dim5_Scalar_Vector_Vector_U, **138**, used: 188, 252, 430
Dim5_Tensor_2_Vector_Vector_1, **138**, used: 188
Dim5_Tensor_2_Vector_Vector_2, **138**, used: 188
Dim6_Gauge_Gauge_Gauge, **138**, used: 188, 247, ??, ??, 428
Dim6_Gauge_Gauge_Gauge_5, **138**, used: 188, 247, ??, ??, 428
Dim6_Vector_Vector_Vector_T, **138**, used: 188
Dim7_Tensor_2_Vector_Vector_T, **138**, used: 188
disambiguate_fusions, **131**, used: 134
Discrete (module), **43**, **39**, used:
disjoint, **785**, used: 786
distribute_degrees, **31**, used: 31

distribute_degrees′, **31**, used: 31, 31
distribute_degrees″, **31**, used: 31, 31
DIV, ??, ??, used: ??, 566
DIV (camlyacc token), **567**, used: 567, 567
divide, **565**, **564**, used: ??, 567
division, **818**, used: 818
DL, **236**, ??, ??, **428**, used: 237, 257, 259, ??, ??, ??, ??, ??, ??, 428, 436, 437
Dodd, **297**, used: 297
domain (type), **765**, **765**, **764**, used: 765, 765, 764, 765
dot, **564**, **564**, used: ??, 567
DOT, ??, ??, used: ??, 566
DOT (camlyacc token), **567**, used: 567, 567
Dotproduct, **564**, **564**, used: 564
DR, **236**, ??, ??, **428**, used: 237, 257, 259, ??, ??, ??, ??, ??, ??, 428, 436, 437
drb, **61**, used: 61, 62, 62, 62, 65, 66
drb0, **61**, used: 62, 62, 63, 63
DScalar2_Vector2, **139**, used: 189
DScalar4, **139**, used: 189
dummy, **733**, **732**, used: 133
Dummy (module), **578**, used:
dummy_flavor, **572**, used: 574
dump, **819**, **808**, used:
Duplicate (exn), **54**, **55**, **61**, **692**, **693**, **50**, **691**, used:
Dyn (module), **781**, **780**, used: 782, 213
Dyn (sig), **781**, **780**, used: 780
D_Alpha_WWWW0_S, **241**, ??, ??, **428**, used: 250, ??, ??, 428
D_Alpha_WWWW0_T, **241**, ??, ??, **428**, used: 250, ??, ??, 428
D_Alpha_WWWW0_U, **241**, ??, ??, **428**, used: 250, ??, ??, 428
D_Alpha_WWWW2_S, **241**, ??, ??, **428**, used: 250, ??, ??, 428
D_Alpha_WWWW2_T, **241**, ??, ??, **428**, used: 250, ??, ??, 428
D_Alpha_ZZWW0_S, **241**, ??, ??, **428**, used: 250, ??, ??, 428
D_Alpha_ZZWW0_T, **241**, ??, ??, **428**, used: 250, ??, ??, 428
D_Alpha_ZZWW1_S, **241**, ??, ??, **428**, used: 250, ??, ??, 428
D_Alpha_ZZWW1_T, **241**, ??, ??, **428**, used: 250, ??, ??, 428
D_Alpha_ZZWW1_U, **241**, ??, ??, **428**, used: 250, ??, ??, 428
D_Alpha_ZZZZ_S, **241**, ??, ??, **428**, used: 250, ??, ??, 428
D_Alpha_ZZZZ_T, **241**, ??, ??, **428**, used: 250, ??, ??, 428
D_K1_L, **344**, used: 344, 347, 353
D_K1_R, **344**, used: 344, 347, 353
D_K2_L, **344**, used: 344, 347, 353
D_K2_R, **344**, used: 344, 347, 353
e (type), ??, ??, ??, ??, used: ??, ??, ??, ??, ??, ??, ??
e (camlyacc non-terminal), **567**, used: 567, 567
E, **241**, **450**, ??, ??, **501**, **532**, **278**, **300**, **321**, **337**, **349**, **367**, **378**, **388**, **414**, **428**, ??, ??, used: 244, ??, ??, 428
edge (type), **41**, **41**, **42**, **42**, **43**, **45**, **109**, **36**, **37**, **40**, used: 41, 41, 42, 42, 42, 43, 45, 100, 109, 36, 36, 37, 37, 37, 38, 38, 38, 39, 39, 39, 40, 40, 40
Edges (module), **109**, used: 109
edges_feynmf, **812**, used: 812

edges_feynmf′, **812**, used: 812, 812
EEnd, **??**, **??**, used: ??
Eidelta, **450**, used:
electromagnetic_currents, **245**, **??**, **??**, **281**, **302**, **323**, **338**, **350**, **368**, **379**, **389**, **417**, **428**, **??**, **??**, used: 258, ??, ??, 291, 312, 327, 340, 353, 370, 380, 391, 422, 436, ??, ??
electromagnetic_currents′, **??**, used: ??
electromagnetic_currents_2, **455**, **503**, **535**, used: 477, 513, 548
electromagnetic_currents_3, **455**, **503**, **534**, used: 477, 513, 548
electromagnetic_sfermion_currents, **455**, **503**, **534**, used: 477, 513, 548
Electron, **227**, used: 228, 228, 229, 229
elements, **755**, **761**, **763**, **754**, used: 746, 781, 781, 784, 784, 785, 829, 829, 830, 832, 44, 47, 700, 186, 215, 215, 216, 103, 120, 123, 124
elements′, **761**, used: 761, 761
eliminate_common_fusions, **132**, used: 134
eliminate_common_fusions1, **132**, used: 132
elt (type), **781**, **781**, **781**, **782**, **782**, **784**, **784**, **213**, **216**, **118**, **779**, **780**, **780**, **783**, used: 781, 781, 781, 781, 781, 782, 782, 782, 784, 213, 779, 780, 780, 780, 780, 780, 780, 780, 780, 783, 784
Elt_Base (sig), **781**, **779**, used: 781, 782, 780, 780
embedding, **748**, **748**, used:
embedding (field), **748**, used: 748, 749, 749, 750
embedding1, **749**, **748**, used:
embedding1 (field), **748**, used: 749, 749, 750
embedding2, **749**, **748**, used:
embedding2 (field), **748**, used: 749, 749, 750
embed_in_decay, **694**, used: 694, 694, 695
embed_in_decays, **695**, used: 695, 696
empty, **755**, **756**, **761**, **784**, **784**, **769**, **769**, **770**, **772**, **773**, **730**, **41**, **43**, **44**, **45**, **48**, **696**, **127**, **134**, **754**, **783**, **767**, **768**, **768**, **730**, **37**, **92**, used: ??, ??, 765, 761, 746, 746, 749, 781, 781, 784, 785, 785, 785, 786, 786, 770, 770, 770, 770, 772, 773, 773, 773, 774, 828, 829, 829, 831, 831, 730, 730, 816, 817, 29, 44, 44, 44, 45, 45, 45, 46, 46, 47, 48, 48, 48, 696, 220, 221, 222, 700, 700, 700, 702, 78, 186, 215, 216, 103, 104, 111, 111, 116, 117, 118, 119, 120, 120, 123, 124, 124, 131, 132, 133, 715, 715, 225, 226, 227, 230, 718
Empty, **755**, used: 756, 756, 757, 758, 758, 759, 760, 760, 761
em_lqino_currents, **542**, used: 548
em_lq_currents, **542**, used: 548
em_up_type_currents, **??**, used: ??
END, **??**, **??**, **??**, **??**, used: ??, ??, 70, 566
END (camlyacc token), **71**, **567**, used: 71, 567
end_step, **734**, **732**, used: 133
enumerate, **743**, **740**, used: 747
EPowSet (module), **784**, used: 784, 784, 784, 785, 785, 785, 786, 786, 786
equal, **769**, **769**, **772**, **81**, **84**, **85**, **218**, **219**, **219**, **767**, **768**, used: 82, 82

EResult, **??**, **??**, used: **??**
ESet (module), **784**, used: 784, 784, 784, 785, 785, 785, 785
EStart, **??**, **??**, used: **??**
Eta, **274**, **297**, **319**, used: 275, 277, 283, 290, 291, 298, 299, 304, 311, 312, 319, 320, 325, 326, 327, 327
eta_higgs_gauge, **327**, used: 327
eval, **41**, **47**, **692**, **695**, **39**, **691**, used: 47, 47
eval_decay, **692**, **693**, **691**, used: 693, 695
eval_memoized, **41**, **48**, **39**, used: 48, 123
eval_memoized', **48**, used: 48
eval_offspring, **47**, used: 48
ew_order, **98**, used: 98, 98
ew_order (type), **98**, used:
EW_order, **98**, used: 98, 98
exists, **737**, **738**, **736**, used: 746, 32, 205
expand_decays, **210**, **213**, **209**, used: 714
expand_home, **735**, **735**, used: 737
expand_scatterings, **210**, **212**, **209**, used: 714
export, **769**, **772**, **772**, **775**, **768**, **769**, used:
export', **772**, **774**, used: 772, 772, 774, 775
expr, **??**, **??**, used: 573
expr (type), **166**, used: 166, 171, 175, 175
expr (camlyacc non-terminal), **567**, used: 567
ext (field), **29**, used: 29, 30, 30, 30, 30
extend, **730**, **730**, used: 730, 730, 128
externals, **94**, **101**, **103**, **122**, **90**, used: 700, 122, 709, 709, 711, 716, 717
externals (field), **100**, **103**, used: 103, 116
external_color_flows, **204**, used: 206
external_edges, **816**, used: 817
external_edges_from, **816**, used: 816, 816
external_flavors, **222**, **571**, **186**, **207**, **225**, **226**, **228**, **230**, **237**, **266**, **269**, **447**, **??**, **??**, **498**, **529**, **275**, **298**, **319**, **333**, **344**, **363**, **375**, **385**, **407**, **411**, **428**, **??**, **??**, **719**, **175**, used: 571, 207, 713, 225, 226, 228, 230, 237, 266, 269, 448, **??**, **??**, 498, 529, 275, 298, 319, 333, 345, 364, 375, 385, 411, 428, **??**, **??**, 719
external_flavors', **447**, used: 447
external_flavors'', **447**, used: 447
external_ghosts, **205**, used: 206
external_wfs, **100**, **101**, **112**, used: 116, 120
ExtMSSM (module), **526**, **525**, used: **??**
extMSSM_flags (sig), **526**, **525**, used: 526, 526, 526, 525
ext_edges (field), **816**, used: 817, 817, 819
ext_incidence, **817**, used: 819
ext_layout (type), **818**, **807**, used: 818, 807, 807
ext_nodes (field), **816**, used: 817, 818, 819, 819, 819, 819, 819
f (type), **218**, **218**, **218**, **222**, **571**, **575**, **225**, **226**, **228**, **231**, **245**, **266**, **269**, **455**, **??**, **??**, **503**, **534**, **280**, **302**, **323**, **337**, **350**, **368**, **379**, **389**, **406**, **416**, **428**, **??**, **??**, **720**, **217**, **217**, **217**, used: 218, 218, 218, 218, 218, 219, 219, 219, 222, 217, 217, 217, 217
F (module), **699**, **575**, **109**, **128**, **708**, **225**, **226**, **228**, **231**, **245**, **266**, **269**, **455**, **??**, **??**, **503**, **534**, **280**, **302**, **323**, **337**, **350**, **368**, **379**, **389**, **406**, **416**, **428**, **??**, **??**, **720**, used: 699, 699, 699, 699, 700, 700, 575, 109, 128, 128, 129, 129,

131, 131, 132, 133, 134, 709, 709, 709, 709, 709, 710, 710, 711, 712, 715, 715, 716, 716, 716, 716, 717, 717, 225, 229, 232, 259, 267, 270, 479, ??, ??, 514, 549, 291, 312, 327, 340, 353, 370, 381, 391, 406, 422, 437, ??, ??, 721
F12, **140**, used: 220
F123, **140**, used: 221
F124, **140**, used: 221
F13, **140**, used: 220
F132, **140**, used: 221
F134, **140**, used: 221
F142, **140**, used: 221
F143, **140**, used: 221
F2 (module), **218**, used: 219, 220
F2′ (module), **220**, used: 220, 220
F21, **140**, used: 220
F213, **140**, used: 221
F214, **140**, used: 221
F23, **140**, used: 220, 227
F231, **140**, used: 221
F234, **140**, used: 221, 227
F241, **140**, used: 221
F243, **140**, used: 221
F3 (module), **219**, used: 219, 221
F3′ (module), **221**, used: 221, 221
F31, **140**, used: 220
F312, **140**, used: 221
F314, **140**, used: 221
F32, **140**, used: 220
F321, **140**, used: 221
F324, **140**, used: 221
F341, **140**, used: 221
F342, **140**, used: 221
F412, **140**, used: 221
F413, **140**, used: 221
F421, **140**, used: 221
F423, **140**, used: 221
F431, **140**, used: 221
F432, **140**, used: 221
factor, **83**, **86**, **81**, used: 134
factor (type), **83**, **84**, **80**, used: 83, 127, 129, 81, 92
factorial, **789**, **27**, **27**, **787**, **19**, used: 790, 790, 27, 32, 32
factorial′, **789**, used: 789, 789
factorize, **745**, **741**, used: 793, 794
factorized_keystones, **794**, **788**, used: 24, 24
factorized_partitions, **793**, **788**, used:
factorize_brakets, **120**, used: 120
Fake_Graded_Map (module), **48**, used: 49
Fake_Grading (module), **43**, used: 49
False, **69**, **74**, **68**, used: 69, 69, 75
family, **237**, **447**, ??, ??, **498**, **529**, **275**, **297**, **319**, **333**, **344**, **363**, **375**, **385**, **411**, **428**, ??, ??, used: 237, 447, ??, ??, 498, 529, 275, 298, 319, 333, 344, 363, 375, 385, 411, 428, ??, ??
fan, **810**, **806**, used: 810
FBF, **138**, used: 188, 229, 231, 245, 245, 246, 246, 246, 246, 246, 253, 254, 254, 254, 254, 254, 254, 255, 255, 255, 255, 256, 256, 256, 256, 256, 257, 257, 257, 257, 257, 257, 258, 258, 258, 455, 455, 455, 456, 456, 456, 457, 457, 457, 457, 457, 457, 458, 458, 458, 458, 458, 459, 459, 459, 461, 461, 462, ??, 503, 503, 503, 504, 504, 504, 505, 505, 505, 505, 505, 506, 506, 506, 506, 506, 507, 507, 507, 508, 534, 535, 535, 535, 535, 535, 536, 536, 536, 536, 537, 537, 537, 537, 537, 537, 538, 538, 538, 539, 539, 539, 539, 540, 540, 540, 540, 540, 540, 541,

541, 541, 541, 541, 542, 542, 542, 281, 281, 281, 281, 281, 282, 282, 282, 282, 283, 302, 303, 303, 303, 303, 303, 304, 304, 304, 304, 323, 323, 323, 323, 324, 325, 325, 325, 325, 338, 338, 338, 338, 338, 350, 350, 350, 351, 351, 368, 368, 368, 368, 379, 379, 379, 379, 379, 389, 389, 389, 389, 390, 403, 403, 403, 403, 404, 404, 404, 404, 417, 417, 417, 417, 417, 418, 418, 418, 418, 418, 418, 419, 428, 432, 433, 433, 433, 433, 433, 434, 434, 434, 434, 435, 435, 435, 435, 435, 435, 435, 436, 436, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, 720

fermion, **222**, **571**, **186**, **207**, **225**, **226**, **228**, **231**, **239**, **266**, **267**, **450**, ??, ??, **500**, **531**, **277**, **299**, **320**, **335**, **347**, **365**, **377**, **387**, **400**, **413**, **428**, ??, ??, **720**, **174**, used: 571, 186, 207, 95, 125, 266, 267

fermion (label), **223**, **175**, used: 575

fermion (type), **397**, **137**, used: 397, 138, 139

Fermion, **95**, **125**, **397**, used: 95, 96, 125, 126, 397, 400, 403, 403, 403, 403, 404, 404, 404, 404

fermionbar (type), **137**, used: 138, 139

fermion_of_lorentz, **570**, used: 570, 573, 573

ferm_of_sff, **453**, used:

feynmf (type), **811**, **806**, used: 813, 806, 807

feynmf_footer, **814**, used: 815, 815

feynmf_header, **814**, used: 815, 815

feynmf_levels, **815**, used: 815, 815, 815

feynmf_levels (type), **813**, **807**, used: 813, 807, 807, 807

feynmf_levels_plain, **815**, **807**, used:

feynmf_levels_wrapped, **815**, **807**, used:

feynmf_set, **814**, **711**, used: 814, 711, 712

feynmf_set (type), **813**, **807**, used: 813, 813, 807, 807

feynmf_sets, **814**, used: 814, 815

feynmf_sets (type), **813**, **807**, used: 807, 807

feynmf_sets_plain, **814**, **807**, used:

feynmf_sets_wrapped, **815**, **807**, used: 712, 712, 712

feynmf_set_sans_color, **711**, used: 711, 711

feynmf_set_sans_color_empty, **711**, used: 712

feynmf_style, **709**, used: 710, 710

fiber, **781**, **782**, **782**, **782**, **780**, **780**, used:

fiber (type), **781**, **781**, **782**, **782**, **780**, **780**, used: 781, 782, 782, 780, 780, 780, 780

Fiber (module), **781**, used: 781, 781, 781, 781

fibered_dag (type), **118**, used:

fibers, **781**, **781**, **782**, **782**, **780**, **780**, used: 782

fiber_to_string, **214**, used: 214

field, **208**, **237**, **267**, ??, ??, **274**, **333**, **344**, **363**, **375**, **385**, **411**, **428**, ??, ??, **177**, used: 267, 269, 269, 269, 270

field (type), **208**, **237**, **267**, ??, ??, **274**, **333**, **344**, **363**, **375**, **385**, **411**, **428**, ??, ??, **177**, used: 177

file, **733**, **732**, used: 133

File, **732**, **128**, used: 733, 128

filter_keystone, **112**, used: 117

filter_offspring, **110**, used: 110

filter_sign, **795**, used: 795, 795, 795, 795

finalize, **697**, used: 697
finalize1, **694**, used: 694, 697
finalize_decay, **694**, used: 694
find, **755**, **757**, **762**, **737**, **738**, **769**, **769**, **770**, **772**, **773**, **43**, **44**, **48**, **754**, **736**, **767**, **768**, **768**, used: 765, 757, 762, 781, 781, 738, 770, 770, 770, 773, 773, 773, 828, 830, 831, 730, 817, 29, 44, 44, 45, 46, 46, 46, 47, 47, 47, 48, 219, 219, 219, 699, 700, 575, 104, 111, 116, 117, 119, 121, 124, 131, 133, 408
find1, **770**, **773**, used: 770, 771, 771, 771, 773, 773, 774, 774
find_first, **737**, used: 738
find_fst, **82**, used: 82, 82
find_opt, **755**, **757**, **762**, **754**, used: 757, 762, 749
find_snd, **82**, used: 82, 82, 82
first_match, **808**, **692**, used: 809, 692
first_match', **692**, used: 692, 692
first_non_white, **568**, used: 569
first_pair, **692**, used: 696
first_pair', **692**, used: 692, 692
flatmap, **743**, **741**, used: 743, 830, 832, 795, 808, 808, 816, 816, 816, 816, 17, 24, 24, 33, 34, 223, 701, 74, 204, 204, 204, 212, 213, 108, 119, 120, 120, 123, 123, 131, 133, 709, 712, 713, 225, 226, 228, 230, 237, 237, 246, 246, 256, 257, 257, 258, 258, 258, 269, 269, 270, 447, 448, 458, 458, 459, 459, 459, 461, 461, 463, 465, 466, 466, 466, 469, 470, 470, 470, 471, 472, 474, 474, 475, 475, 476, 477, 478, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, 498, 498, 507, 509, 510, 511, 511, 511, 512, 512, 513, 513, 514, 529, 529, 538, 539, 540, 540, 540, 540, 540, 541, 541, 541, 542, 542, 542, 543, 544, 545, 545, 545, 546, 546, 547, 548, 549, 275, 275, 291, 298, 298, 312, 319, 319, 323, 325, 327, 333, 333, 340, 344, 345, 353, 363, 364, 370, 375, 375, 380, 385, 385, 391, 398, 411, 411, 422, 428, 435, 435, 436, 436, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, 719
flatten_keystones, **108**, used: 124
flavor, **94**, **100**, **101**, **121**, **88**, used: 699, 700, 108, 112, 117, 119, 121, 709, 710, 710, 717
flavor (field), **100**, **101**, used: 101, 101, 102, 104, 108, 108, 111, 112, 117, 118, 119, 119, 123, 124
flavor (type), **222**, **571**, **73**, **74**, **184**, **207**, **210**, **210**, **94**, **95**, **95**, **100**, **101**, **121**, **125**, **127**, **128**, **707**, **708**, **225**, **226**, **227**, **230**, **236**, **266**, **267**, **447**, **??**, **??**, **498**, **529**, **274**, **297**, **319**, **333**, **344**, **363**, **374**, **384**, **397**, **411**, **428**, **??**, **??**, **718**, **172**, **72**, **209**, **88**, **91**, **707**, used: 699, 699, 700, 571, 575, 73, 74, 74, 74, 76, 184, 184, 207, 210, 210, 210, 211, 214, 216, 94, 95, 95, 95, 95, 95, 97, 100, 100, 100, 101, 101, 101, 101, 103, 104, 104, 105, 121, 125, 125, 127, 128, 128, 129, 707, 708, 225, 226, 228, 231, 241, 245, 266, 266, 269, 450, 455, **??**, **??**, 501, 503, 532, 534, 278, 280, 300, 302, 321, 323, 337, 337, 349, 350, 367, 368, 378, 379, 388, 389, 406, 414, 416, 428, **??**, **??**, **??**, **??**, 720, 172, 172, 172, 172, 174,

174, 174, 174, 174, 175, 175, 175, 175, 175, 175, 175, 177, 177, 178, 178, 178, 178, 217, 73, 73, 73, 73, 183, 183, 209, 209, 210, 210, 88, 90, 91, 91, 93, 93, 707, 707

Flavor (sig), **218**, **217**, used: 218, 217

FLAVOR, **??**, **??**, used: ??, ??, 70

FLAVOR (camlyacc token), **71**, used: 72

flavors, **222**, **571**, **186**, **207**, **214**, **127**, **129**, **225**, **226**, **228**, **230**, **237**, **266**, **269**, **448**, **??**, **??**, **498**, **529**, **275**, **298**, **319**, **333**, **345**, **364**, **375**, **385**, **406**, **411**, **428**, **??**, **??**, **719**, **175**, **92**, used: 571, 576, 186, 207, 215, 105, 117, 266, 269, 408

flavors (field), **129**, used: 129, 134

flavors (label), **223**, **175**, used:

flavors (type), **129**, used:

flavors_of_particles, **574**, used: 576

flavors_to_string, **701**, **75**, used: 701, 701, 701, 75

flavor_keystone, **108**, used: 108

flavor_keystones, **108**, used: 117

flavor_list (camlyacc non-terminal), **72**, used: 71, 72

flavor_of_antiparticle, **573**, used: 574

flavor_of_b, **397**, used: 406, 407

flavor_of_f, **397**, used: 406, 407

flavor_of_particle, **573**, used: 574

flavor_of_string, **222**, **571**, **186**, **207**, **226**, **227**, **229**, **232**, **259**, **267**, **268**, **479**, **??**, **??**, **514**, **549**, **291**, **312**, **327**, **340**, **353**, **370**, **381**, **391**, **408**, **422**, **437**, **??**, **??**, **721**, **175**, used: 571, 75, 186, 207, 211, 267, 268

flavor_of_string (label), **223**, **175**, used:

flavor_sans_color, **184**, **207**, **94**, **121**, **178**, **178**, **88**, used: 184, 204, 204, 204, 207, 117, 118, 119, 121, 129, 709, 709

flavor_sans_color (type), **184**, **207**, **94**, **121**, **178**, **178**, **88**, used: 207, 94, 95, 97, 178, 178, 178, 178, 183, 183, 88, 89, 90, 91, 93

Flavor_Set (module), **700**, used: 700, 700, 700

flavor_sums, **214**, used: 215

flavor_symbol, **222**, **571**, **187**, **207**, **226**, **227**, **229**, **233**, **262**, **267**, **267**, **481**, **??**, **??**, **518**, **554**, **293**, **315**, **330**, **342**, **360**, **372**, **383**, **393**, **408**, **424**, **440**, **??**, **??**, **723**, **175**, used: 571, 187, 207, 124, 264, 267, 267, 487, ??, ??, 294, 316, 331, 343, 362, 374, 384, 394, 425, 441, ??, ??

flavor_symbol (label), **223**, **175**, used: 575

flavor_to_string, **222**, **571**, **187**, **207**, **226**, **227**, **229**, **232**, **260**, **267**, **268**, **480**, **??**, **??**, **515**, **551**, **292**, **313**, **328**, **340**, **354**, **371**, **381**, **392**, **407**, **422**, **438**, **??**, **??**, **722**, **175**, used: 701, 571, 576, 75, 187, 203, 207, 213, 129, 709, 713, 715, 267, 268, 408

flavor_to_string (label), **223**, **175**, used: 575

flavor_to_TeX, **222**, **571**, **187**, **207**, **226**, **227**, **229**, **232**, **261**, **267**, **268**, **482**, **??**, **??**, **516**, **552**, **292**, **314**, **329**, **341**, **357**, **372**, **382**, **392**, **408**, **423**, **439**, **??**, **??**, **722**, **175**, used: 571, 187, 207, 709, 710, 710, 712, 267, 268

flavor_to_TeX (label), **223**, **175**,

used: 575
flip_incoming, **696**, used: 696
flip_s_channel_in, **54**, **59**, **64**, **53**, used: 696, 699
flow, **207**, **208**, **178**, **178**, used: 208, 129
Flow (module), **83**, **81**, used: 207, 127, 129, 129, 129, 134, 134, 134, 178, 178, 92, 92
Flow (sig), **83**, **80**, used: 81
FMap (module), **129**, used: 133, 133
Fn (module), **219**, used: 219
fold, **755**, **760**, **764**, **769**, **769**, **771**, **772**, **774**, **777**, **810**, **41**, **41**, **42**, **42**, **43**, **43**, **44**, **46**, **48**, **109**, **754**, **767**, **768**, **768**, **776**, **806**, **36**, **38**, **40**, used: 760, 761, 764, 781, 781, 786, 786, 771, 774, 828, 829, 829, 829, 830, 830, 830, 830, 830, 830, 831, 831, 831, 832, 777, 810, 810, 810, 16, 17, 17, 30, 43, 44, 44, 44, 45, 46, 46, 46, 46, 46, 47, 47, 47, 47, 47, 48, 48, 700, 212, 213, 104, 109, 116, 117, 118, 120, 120, 124, 131
fold (type), **778**, **776**, used: 776
fold′, **771**, **774**, used: 771, 774
fold2, **830**, **777**, **776**, used: 830, 777, 777, 11, 11, 12, 15, 47, 108, 108, 119, 123
fold2 (type), **778**, **776**, used: 776
fold2_rev, **777**, used: 777
fold3, **777**, **776**, used: 777, 12, 12, 15
fold3_rev, **777**, used: 777
fold_left, **9**, **11**, **12**, **14**, **16**, **6**, used: **??**, **??**, **??**, 765, 743, 743, 744, 744, 746, 746, 785, 735, 829, 830, 830, 830, 830, 831, 831, 730, 777, 777, 777, 777, 790, 790, 792, 792, 792, 792, 793, 793, 793, 793, 794, 795, 795, 796, 816, 817, 818, 16, 26, 26, 29, 29, 29, 31, 31, 32, 32, 32, 32, 32, 47, 48, 61, 181, 181, 181, 182, 84, 220, 221, 699, 700, 700, 572, 574, 575, 74, 186, 214, 97, 98, 103, 108, 110, 111, 111, 112, 119, 120, 121, 122, 122, 131, 132, 132, 133
fold_left2, **743**, **741**, used:
fold_lefti, **??**, used: **??**, **??**
fold_left_internal, **9**, **11**, **12**, **14**, **16**, **6**, used: 108
fold_lines, **569**, used: 570
fold_nodes, **41**, **46**, **37**, used: 111, 118, 124
fold_rev, **777**, used: 777, 777
fold_right, **9**, **11**, **12**, **14**, **16**, **6**, used: 744, 746, 746, 781, 784, 785, 785, 803, 812, 11, 11, 12, 13, 15, 16, 16, 17, 42, 694, 696, 700, 700, 702, 565, 570, 75, 214, 214, 214, 215, 215, 216, 103, 104, 118, 119, 124
fold_right2, **744**, **741**, used: 123
fold_right_internal, **9**, **11**, **12**, **14**, **16**, **6**, used: 99
forest, **41**, **47**, **94**, **123**, **39**, **91**, used: 709, 711, 716, 717
Forest (module), **42**, **36**, used: 103, 109
Forest (sig), **41**, **36**, used: 42, 42, 43, 49, 36, 39, 40
forest′, **123**, used: 123
Forest_Grader (sig), **42**, **40**, used: 40
forest_memoized, **41**, **48**, **39**, used: 123
forest_sans_color, **709**, used: 711, 711, 712
format_flavor, **701**, used: 702
format_generation, **268**, used: 268, 268
format_maps, **701**, used:
format_p, **709**, used: 709, 709, 712, 712, 712
Fortran (module), **678**, **678**, **578**, used: 718, **??**, 718,

??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??
Fortran77 (module), **578**, used:
Fortran_Majorana (module), **578**, used: ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, 723
for_all, **9**, **11**, **12**, **14**, **16**, **41**, **42**, **42**, **43**, **109**, **6**, **36**, **40**, used: 744, 16, 31, 42, 43, 45, 181, 182, 74, 76, 76, 77, 78, 203, 205, 211, 108, 109, 112
four_gluon, **232**, **721**, used: 232, 721
FSet (module), **214**, used: 214, 214, 214, 214, 215
fspec_of_gen, **397**, used: 409
fspec_of_iso, **397**, used: 409
fspec_of_kk2, **397**, used: 409
fspec_of_kkmode, **397**, used: 409
Fudged, **137**, used: 239, 449, ??, ??, 499, 531, 276, 299, 320, 334, 346, 365, 376, 386, 412, 428, ??
functions_file, **576**, used: 576, 576
fuse, **809**, **218**, **220**, **222**, **571**, **204**, **207**, **97**, **97**, **97**, **98**, **108**, **225**, **227**, **229**, **232**, **259**, **267**, **270**, **479**, ??, ??, **514**, **549**, **291**, **312**, **327**, **340**, **353**, **370**, **381**, **391**, **406**, **422**, **437**, ??, ??, **721**, **174**, **806**, **217**, **93**, used: 571, 575, 204, 207, 104, 110, 119, 123, 225, 229, 232, 259, 267, 270, 479, ??, ??, 514, 549, 291, 312, 327, 340, 353, 370, 381, 391, 406, 422, 437, ??, ??, 721
fuse (label), **223**, **175**, used: 575
fuse2, **11**, **218**, **219**, **222**, **571**, **204**, **207**, **225**, **227**, **229**, **232**, **259**, **267**, **270**, **479**, ??, ??, **514**, **549**, **291**, **312**, **327**, **340**, **353**, **370**, **381**, **391**, **406**, **422**, **437**, ??, ??, **721**, **174**, **217**, used: 11, 15, 220, 220, 571, 575, 204, 204, 207, 225, 229, 232, 259, 267, 270, 271, 479, ??, ??, 514, 549, 291, 312, 327, 340, 353, 370, 381, 391, 406, 422, 437, ??, ??, 721
fuse2 (type), **140**, used: 140
fuse3, **12**, **218**, **219**, **222**, **571**, **204**, **207**, **225**, **227**, **229**, **232**, **259**, **267**, **270**, **479**, ??, ??, **514**, **549**, **291**, **312**, **327**, **340**, **353**, **370**, **381**, **391**, **406**, **422**, **437**, ??, ??, **721**, **174**, **217**, used: 13, 15, 220, 220, 571, 575, 204, 204, 207, 225, 229, 232, 259, 267, 270, 271, 479, ??, ??, 514, 549, 291, 312, 327, 340, 353, 370, 381, 391, 406, 422, 437, ??, ??, 721
fuse3 (type), **140**, used: 140
fusen, **219**, used: 220, 220
fusen (type), **140**, used: 140
fuse_c_wf, **119**, used: 119, 119
fuse_list, **204**, used: 204
fuse_n, **17**, **17**, used: 17
fuse_rhs, **104**, used: 104, 108
fuse_trees, **123**, used: 123
fusion (type), **94**, **100**, **103**, **121**, **127**, **128**, **89**, **91**, used: 94, 100, 100, 101, 103, 121, 127, 128, 128, 129, 89, 90, 92, 93
Fusion (module), **93**, **88**, used: 699, 708, 708, 718, ??, ??, ??, 718, ??, 723, 179, 698, 707
fusions, **94**, **101**, **103**, **122**, **127**, **129**, **90**, **92**, used: 122, 131, 132
fusions (field), **100**, **103**, **129**, used: 103, 116, 122, 122,

124, 129, 134
Fusions (module), **218**, **217**, used: 575, 225, 226, 228, 231, 245, 266, 269, 455, ??, ??, 503, 534, 280, 302, 323, 337, 350, 368, 379, 389, 406, 416, 428, ??, ??, 720
Fusions (sig), **218**, **217**, used: 218, 217
fusion_dag, **101**, **103**, used:
fusion_dag (field), **100**, **103**, used: 103, 116, 117, 120, 121, 122, 123, 123, 125
fusion_tower, **111**, used: 111, 111
fusion_tower (field), **100**, **103**, used: 116, 117, 121, 125
fusion_tower', **110**, used: 110, 111
f_aux (field), **572**, used: 572, 573, 573
f_aux_top (type), **236**, ??, ??, **428**, used: 236, ??, ??, 428
f_color (field), **572**, used: 572, 573, 573, 575
f_conjugate (field), **572**, used: 572, 573, 573, 575
f_fermion (field), **572**, used: 572, 573, 573, 575
f_mass (field), **572**, used: 572, 573, 573, 575
f_name (field), **572**, used: 572, 573, 573, 575
f_pdg (field), **572**, used: 572, 573, 573, 575
f_propagator (field), **572**, used: 572, 573, 573, 575
f_spin (field), **572**, used: 572, 573, 573, 575
f_symbol (field), **572**, used: 572, 573, 573, 575
f_width (field), **572**, used: 572, 573, 573, 575
g (type), **222**, **571**, **217**, used: 222, 217
G, **225**, **236**, **267**, **446**, **450**, ??, ??, **497**, **501**, **527**, **532**, **274**, **333**, **344**, **363**, **374**, **384**, **397**, **411**, **428**, ??, ??, **718**, used: 225, 225, 237, 239, 243, 244, 245, 246, 248, 251, 251, 252, 253, 253, 253, 254, 254, 254, 254, 254, 254, 255, 255, 255, 256, 256, 256, 259, 267, 269, ??, 274, 277, 281, 291, 333, 335, 337, 338, 338, 339, 339, 339, 339, 340, 340, 344, 347, 350, 351, 351, 352, 352, 352, 352, 353, 353, 363, 365, 368, 368, 369, 369, 369, 369, 369, 370, 370, 375, 376, 379, 380, 380, 380, 380, 380, 381, 385, 386, 389, 390, 390, 390, 390, 391, 391, 391, 398, 404, 405, 406, 407, 408, 411, 412, 417, 419, 420, 420, 420, 422, 422, 428, 428, 429, 430, 430, 431, 431, 431, 432, 432, 432, 432, 432, 432, 433, 433, 433, 433, 433, 434, 434, 434, 435, 435, 437, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, 719, 720, 720, 720, 720, 721, 721, 721
G (module), **42**, **43**, **43**, **43**, **45**, **109**, **39**, used: 42, 44, 44, 44, 44, 45, 45, 48, 49, 39, 40
G2, **230**, **241**, ??, ??, **278**, **300**, **321**, **337**, **349**, **378**, **388**, **414**, **428**, ??, ??, used: 232, 248, ??, ??, 284, 305, 326, 338, 351, 380, 390, 420, 428, ??, ??
G22, **349**, used: 351
G3, **226**, **719**, used: 227, 227, 720, 720
G4, **226**, **719**, used: 227, 227,

721, 721
G5_AWW, **241**, ??, ??, **428**, used: 247, ??, ??, 428
G5_ZWW, **241**, ??, ??, **428**, used: 247, ??, ??, 428
Ga, **236**, **447**, ??, ??, **498**, **529**, **274**, **297**, **319**, **333**, **344**, **363**, **374**, **384**, **411**, **428**, ??, ??, used: 237, 239, 245, 246, 247, 248, 252, 253, 253, 253, 254, 254, 255, 259, 447, 449, 455, 455, 455, 462, 463, 463, 463, 464, 464, 465, 465, 466, 466, 476, 476, 479, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, 498, 500, 503, 503, 503, 508, 508, 508, 509, 509, 510, 510, 510, 510, 511, 511, 514, 529, 531, 534, 534, 535, 542, 542, 542, 542, 542, 542, 543, 543, 543, 544, 544, 544, 545, 545, 546, 549, 275, 277, 281, 281, 283, 283, 284, 284, 286, 287, 290, 290, 291, 298, 299, 302, 303, 305, 305, 305, 305, 307, 308, 311, 312, 312, 319, 320, 323, 325, 325, 325, 326, 326, 327, 327, 333, 335, 338, 338, 338, 339, 339, 340, 340, 344, 347, 350, 351, 351, 352, 352, 353, 353, 363, 365, 368, 368, 369, 369, 369, 370, 370, 375, 376, 379, 380, 380, 380, 381, 385, 386, 389, 390, 390, 391, 391, 391, 411, 412, 417, 419, 420, 420, 420, 422, 422, 428, 430, 430, 431, 432, 433, 434, 437, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??
gauge (type), **222**, **571**, **184**, **207**, **225**, **226**, **228**, **230**, **237**, **266**, **267**, **447**, ??, ??, **498**, **529**, **274**, **297**, **319**, **333**, **344**, **363**, **375**, **385**, **400**, **411**, **428**, ??, ??, **719**, **174**, used: 571, 572, 184, 207, 267, 174, 175, 175, 217
Gauge, **208**, **237**, **267**, ??, ??, **274**, **333**, **344**, **363**, **375**, **385**, **411**, **428**, ??, ??, **177**, **178**, used: 713, 237, 267, ??, ??, 274, 333, 344, 363, 375, 385, 411, 428, ??, ??
Gauge (module), **207**, **183**, used:
Gauge (sig), **176**, used: 207, 267, 177, 177, 178, 183, 224, 224, ??, 272, 273, ??
gauge4, **232**, **248**, **463**, ??, ??, **508**, **542**, **284**, **305**, **326**, **338**, **351**, **369**, **380**, **390**, **405**, **420**, **428**, ??, ??, **721**, used: 232, 248, 463, ??, ??, 508, 542, 284, 284, 305, 305, 326, 338, 351, 369, 380, 390, 405, 406, 420, 428, ??, ??, 721
gauge_boson, **208**, **237**, **267**, ??, ??, **274**, **333**, **344**, **363**, **375**, **385**, **411**, **428**, ??, ??, **177**, used: 237, 267, ??, ??, 275, 333, 344, 363, 375, 385, 411, 428, ??, ??
gauge_boson (type), **208**, **236**, **267**, ??, ??, **274**, **333**, **344**, **363**, **374**, **384**, **411**, **428**, ??, ??, **176**, used: 208, 208, 236, 237, 267, ??, ??, 274, 274, 333, 333, 344, 344, 363, 363, 374, 375, 384, 385, 411, 411, 428, ??, ??, ??, ??, 177, 177, 177, 177, 177, 178
Gauge_Gauge_Gauge, **138**, used: 188, 231, 246, 463, ??, ??, 508, 542, 283, 283, 305, 305, 325, 325, 338, 351, 368, 380, 390, 404, 405, 405, 419, 428, ??, ??, 720
gauge_higgs, **253**, **463**, **509**, **543**, **290**, **311**, **327**, **339**, **352**,

369, **380**, **390**, **420**, **430**, **??**, **??**, used: 258, 477, 513, 548, 291, 312, 327, 340, 353, 370, 380, 391, 422, 436, **??**, **??**
gauge_higgs4, **253**, **464**, **510**, **544**, **290**, **311**, **327**, **339**, **352**, **370**, **380**, **390**, **420**, **430**, **??**, **??**, used: 259, 478, 514, 549, 291, 312, 327, 340, 353, 370, 381, 391, 422, 437, **??**, **??**
gauge_higgs4_GaGaCC, **509**, **543**, used: 510, 544
gauge_higgs4_GaWPC, **510**, **544**, used: 510, 544
gauge_higgs4_GaWSC, **510**, **544**, used: 510, 544
gauge_higgs4_WWCC, **509**, **544**, used: 510, 544
gauge_higgs4_WWPP, **509**, **544**, used: 510, 544
gauge_higgs4_WWSS, **509**, **544**, used: 510, 544
gauge_higgs4_ZGaCC, **509**, **543**, used: 510, 544
gauge_higgs4_ZWPC, **510**, **544**, used: 510, 544
gauge_higgs4_ZWSC, **509**, **544**, used: 510, 544
gauge_higgs4_ZZCC, **509**, **543**, used: 510, 544
gauge_higgs4_ZZPP, **509**, **543**, used: 510, 544
gauge_higgs4_ZZSS, **509**, **543**, used: 510, 544
gauge_higgs_GaCC, **508**, **543**, used: 509, 543
gauge_higgs_GlGlP, **508**, used: 509
gauge_higgs_GlGlS, **508**, used: 509
gauge_higgs_gold, **464**, used: 478
gauge_higgs_gold4, **465**, used: 478
gauge_higgs_WPC, **508**, **543**, used: 509, 543
gauge_higgs_WSC, **508**, **543**, used: 509, 543
gauge_higgs_WWS, **508**, **543**, used: 509, 543
gauge_higgs_ZCC, **508**, **543**, used: 509, 543
gauge_higgs_ZSP, **508**, **543**, used: 509, 543
gauge_higgs_ZZS, **508**, **543**, used: 509, 543
gauge_sfermion4, **465**, **511**, **545**, used: 478, 514, 549
gauge_sfermion4', **465**, **510**, **544**, used: 465, 511, 545
gauge_sfermion4'', **465**, **510**, **545**, used: 465, 511, 545
gauge_squark4, **466**, **511**, **545**, used: 478, 514, 549
gauge_squark4', **466**, **511**, **545**, used: 466, 511, 545
gauge_squark4'', **466**, **511**, **545**, used: 466, 511, 545
gauge_symbol, **222**, **571**, **188**, **207**, **226**, **227**, **229**, **233**, **237**, **266**, **267**, **447**, **??**, **??**, **498**, **529**, **274**, **274**, **297**, **319**, **333**, **344**, **363**, **375**, **385**, **400**, **411**, **428**, **??**, **??**, **723**, **175**, used: 571, 188, 207, 267
gauge_symbol (label), **223**, **175**, used: 575
Gauss, **69**, **74**, **68**, used: 69, 75
GAUSS, **??**, **??**, used: **??**, 70
GAUSS (camlyacc token), **71**, used: 71
Gauss_not, **69**, **74**, **68**, used: 69
GBBG, **139**, used: 189, 476, 476, 476, 476, 476, 290, 312, 327
GBG, **138**, used: 188, 462, 462, 462, 462, 462, 369, 369
gcd, **826**, used: 826, 826, 826
gen (type), **446**, **497**, **527**, used: 446, 497, 527
Gen0, **397**, used: 398
Gen1, **397**, used: 398
Gen2, **397**, used: 398
General_Flow (module), **86**, used:
generation, **240**, **454**, **??**, **??**, **500**, **336**, **348**, **366**, **377**,

387, **413**, **428**, **??**, **??**, used: 241, 454, **??**, **??**, 501, 336, 349, 367, 378, 388, 414, 428, **??**, **??**
generation (type), **397**, used: 397, 401
generation', **240**, **454**, **??**, **??**, **500**, **335**, **348**, **366**, **377**, **387**, **413**, **428**, **??**, **??**, used: 240, 454, **??**, **??**, 500, 336, 348, 366, 377, 387, 413, 428, **??**, **??**
generations, **269**, **718**, used: 269, 269, 269, 269, 718, 718, 718, 719
generations_pairs, **718**, used: 720
generations_quadruples, **718**, used: 721, 721
generations_triples, **718**, used: 720, 720, 720
get_kk2, **402**, used: 404, 404, 405, 405, 405
GF (module), **109**, used: 110
GG, **446**, **497**, **527**, used:
ghh, **281**, used: 286
ghost, **83**, **83**, **80**, used: 207
Ghost, **83**, **86**, used: 83, 84
ghost_flag, **84**, used: 84, 84, 84
ghost_flags, **83**, **84**, **80**, used:
Gl, **230**, **236**, **447**, **??**, **??**, **498**, **529**, **274**, **297**, **319**, **333**, **344**, **363**, **374**, **384**, **411**, **428**, **??**, **??**, used: 230, 230, 231, 231, 232, 232, 237, 239, 245, 246, 248, 253, 254, 255, 259, 447, 449, 457, 462, 462, 462, 463, 463, 463, 466, 466, 467, 476, 476, 479, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, 498, 500, 505, 508, 508, 508, 508, 508, 511, 511, 512, 514, 529, 531, 536, 539, 542, 542, 542, 542, 542, 542, 542, 545, 546, 546, 549, 275, 277, 281, 283, 284, 290, 291, 298, 299, 303, 305, 305, 311, 312, 319, 320, 323, 325, 326, 326, 327, 333, 335, 338, 338, 338, 339, 339, 340, 344, 347, 351, 351, 351, 352, 352, 353, 363, 365, 369, 370, 375, 376, 379, 380, 380, 381, 385, 386, 389, 390, 390, 391, 391, 411, 412, 417, 419, 420, 422, 428, 430, 433, 434, 437, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**
global_verbose, **??**, used: **??**, **??**, **??**, **??**
Gluino, **447**, **498**, **529**, used: 447, 449, 457, 459, 462, 476, 479, 498, 500, 505, 507, 514, 529, 531, 536, 538, 541, 549
gluon2_lq2, **542**, used: 549
gluon2_lq2', **542**, used: 542
gluon2_phi, **720**, used: 721
gluon2_squark2, **467**, **512**, **546**, used: 478, 514, 549
gluon2_squark2', **512**, **546**, used: 512, 546
gluon3_phi, **721**, used: 721
gluon4, **463**, used:
gluon_gauge_squark, **466**, **512**, **546**, used: 478, 514, 549
gluon_gauge_squark', **466**, **511**, **546**, used: 466, 512, 546
gluon_gauge_squark'', **466**, **511**, **546**, used: 466, 512, 546
gluon_w_squark, **466**, **511**, **545**, used: 478, 514, 549
gluon_w_squark', **466**, **511**, **545**, used: 466, 511, 545
gluon_w_squark'', **466**, **511**, **545**, used: 466, 511, 545
Gl_K1, **344**, used: 344, 347, 351, 351, 353
Gl_K2, **344**, used: 344, 347, 351, 351, 353
gobble_arrows, **569**, used: 570
gobble_white, **569**, used: 569, 569, 570
goldstone, **222**, **571**, **185**, **207**, **225**, **226**, **228**, **230**, **239**, **266**, **268**, **449**, **??**, **??**, **500**, **531**, **276**, **299**, **320**,

335, 346, 365, 376, 386, 407, 412, 428, ??, ??, 720, 175, 177, used: 571, 185, 207, 104, 111, 266, 268
goldstone (label), 223, 175, used: 575
goldstone (type), 177, used: 177, 177
Goldstone, 177, used:
goldstone4, 468, used: 478
goldstone_charg_neutr, 458, used: 478
goldstone_neutr, 459, used: 478
goldstone_neutr′, 459, used: 459
goldstone_neutr′′, 459, used: 459
goldstone_sfermion, 470, used: 478
goldstone_sfermion′, 470, used: 470
goldstone_sfermion′′, 470, used: 470
goldstone_sfermion′′′, 470, used: 470
goldstone_squark, 470, used: 478
goldstone_squark′, 470, used: 470
goldstone_squark_a, 470, used: 470
goldstone_squark_b, 470, used: 470
goldstone_vertices, 253, ??, ??, 290, 312, 327, 340, 353, 370, 380, 391, 422, 431, ??, ??, used: 258, ??, ??, 291, 312, 327, 340, 353, 370, 380, 391, 422, 436, ??, ??
graded (type), 9, 11, 12, 15, 17, 7, used: 9, 7
Graded (module), 48, 40, used: 110
Graded (sig), 48, 40, used: 40
Graded_Forest (sig), 42, 40, used: 42, 45, 48, 40, 40
Graded_Map (module), 44, used: 48
Graded_Map (sig), 43, used: 44, 44, 48
Graded_Map_Maker (sig), 44, used: 45
Graded_Ord (sig), 42, 39, used: 42, 42, 44, 44, 48, 39, 40
graded_sym_power, 9, 11, 13, 15, 17, 7, used:
graded_sym_power_fold, 9, 11, 13, 15, 17, 7, used: 11, 13, 15, 17, 110
Grader (sig), 42, 39, used: 42, 43, 39, 40
Grade_Forest (module), 43, 40, used: 49
graph (type), 816, 807, used: 818, 807, 807
graph_of_tree, 817, 807, used:
Grav, 333, 344, 137, used: 462, 462, 462, 462, 476, 476, 476, 333, 335, 338, 339, 339, 340, 344, 347, 351, 352, 352, 353, 369
Gravbar, 137, used: 462, 462, 462, 462, 462, 476, 476, 476, 476, 476, 369, 369
gravitino, 444, 444, 444, 445, 445, 445, 445, 443, used: 447, 478, 478
gravitino_coup, 369, used: 370
gravitino_gauge, 369, used: 370
gravitino_gaugino_3, 462, used: 463
Graviton_Scalar_Scalar, 138, used: 188, 339, 352
Graviton_Spinor_Spinor, 138, used: 188, 338, 351
Graviton_Vector_Vector, 138, used: 188, 339, 352
gravity_currents, 338, 351, used: 340, 353
gravity_gauge, 339, 352, used: 340, 353
gravity_higgs, 339, 352, used: 339, 352
GravTest (module), 363, 272, used: ??
Grino, 447, 363, used: 447, 449, 462, 462, 462, 462, 462, 476, 476, 476, 476, 476, 479, 363, 365, 369, 369, 370

Groves (module), **267**, **224**, used: 272
grow, **110**, used: 110
Gs, **230**, **241**, **450**, ??, ??, **501**, **532**, **278**, **300**, **321**, **337**, **349**, **378**, **388**, **414**, **428**, ??, ??, used: 231, 232, 245, 457, 462, 462, ??, ??, 505, 508, 508, 536, 539, 542, 542, 542, 281, 303, 323, 338, 351, 379, 389, 417, 428, ??, ??
G_AASFSF, **450**, used: 471
G_aaww, **401**, used: 402, 405
G_AAWW, **241**, ??, ??, **278**, **300**, **321**, **337**, **349**, **367**, **378**, **388**, **414**, **428**, ??, ??, used: 248, ??, ??, 284, 284, 287, 305, 305, 308, 326, 338, 351, 369, 380, 390, 420, 428, ??, ??
G_AG0SFSF, **450**, used: 472, 473
G_AGSNSL, **450**, used: 472
G_AGSUSD, **450**, used: 473
G_AHpHm, **414**, used: 420
G_AHPsip, **278**, **300**, used: 286, 307
G_AHTHT, **278**, **300**, used: 281, 303
G_AHTHTH, **278**, **300**, used: 281, 303
G_AHTT, **278**, **300**, used: 281, 303
G_ASFSF, **450**, used: 469
G_aww, **401**, used: 402, 404
G_AZWW, **241**, ??, ??, **278**, **300**, **321**, **337**, **349**, **367**, **378**, **388**, **414**, **428**, ??, ??, used: 248, ??, ??, 284, 284, 305, 305, 326, 338, 351, 369, 380, 390, 420, 428, ??, ??
G_a_lep, **401**, used: 402, 403
g_a_quark, **402**, used: 402
G_a_quark, **401**, used: 402, 403
G_CAC, **450**, used: 458
G_CC, **241**, **450**, ??, ??, **501**, **532**, **278**, **300**, **321**, **337**, **349**, **367**, **378**, **388**, **414**, **428**, ??, ??, used: 244, 246, 246, 456, ??, ??, 504, 535, 282, 303, 304, 323, 325, 327, 338, 350, 368, 379, 389, 417, 428, ??, ??
G_CCQ, **241**, **450**, ??, ??, **501**, **532**, **337**, **349**, **367**, **428**, used: 246, 456, ??, ??, 504, 535, 428
G_CCtop, **278**, **300**, used: 281, 303
G_CC_heavy, **278**, **300**, used: 282, 303
G_CC_W, **278**, **300**, used: 281, 303
G_CC_WH, **278**, **300**, used: 281, 303
G_CGC, **450**, used: 458
G_CGN, **450**, used: 458
G_CH1C, **450**, used: 458
G_CH2C, **450**, used: 458
G_CHN, **450**, used: 458
G_CICIA, **450**, used: 459
G_CICIG, **450**, used: 459
G_CICIH1, **450**, used: 459
G_CICIH2, **450**, used: 459
G_CICIP, **501**, **532**, used: 506, 537
G_CICIS, **501**, **532**, used: 506, 537
G_CPC, **501**, **532**, used: 506, 537
G_CSC, **501**, **532**, used: 506, 537
G_CWN, **450**, **501**, **532**, used: 456, 504, 535
G_CZC, **450**, **501**, **532**, used: 457, 505, 536
G_Ebb, **278**, **300**, **321**, used: 283, 304, 325
G_EGaGa, **278**, **300**, **321**, used: 290, 311, 326
G_EGaZ, **278**, **300**, **321**, used: 290, 311, 326
G_EGlGl, **278**, **300**, **321**, used: 290, 311, 326
G_Etht, **278**, **300**, **321**, used: 283, 304, 325
G_Ethth, **278**, **300**, **321**, used:

G_Ett, **278**, **300**, **321**, used: 283, 304, 325
G_FWW, **??**, **428**, used: **??**, 432
G_FWW_T, **??**, **428**, used: **??**, 432
G_FZZ, **??**, **428**, used: **??**, 432
G_FZZ_T, **??**, **428**, used: **??**, 432
G_G0G0SFSF, **450**, used: 472, 473
G_GG4, **450**, used: 468
G_GGSFSF, **450**, used: 472, 473
G_GGSNSL, **450**, used: 472
G_GGSUSD, **450**, used: 473
G_GH, **450**, used: 463, 464
G_GH4, **450**, used: 464
G_GH4_GaGaCC, **501**, **532**, used: 509, 543
G_GH4_GaWPC, **501**, **532**, used: 510, 544
G_GH4_GaWSC, **501**, **532**, used: 510, 544
G_GH4_WWCC, **501**, **532**, used: 509, 544
G_GH4_WWPP, **501**, **532**, used: 509, 544
G_GH4_WWSS, **501**, **532**, used: 509, 544
G_GH4_ZGaCC, **501**, **532**, used: 509, 543
G_GH4_ZWPC, **501**, **532**, used: 510, 544
G_GH4_ZWSC, **501**, **532**, used: 509, 544
G_GH4_ZZCC, **501**, **532**, used: 509, 543
G_GH4_ZZPP, **501**, **532**, used: 509, 543
G_GH4_ZZSS, **501**, **532**, used: 509, 543
G_GHGo, **450**, used: 464
G_GHGo4, **450**, used: 465
G_GH_GaCC, **501**, **532**, used: 508, 543
G_GH_WPC, **501**, **532**, used: 508, 543
G_GH_WSC, **501**, **532**, used: 508, 543
G_GH_WWS, **501**, **532**, used: 508, 543
G_GH_ZCC, **501**, **532**, used: 508, 543
G_GH_ZSP, **501**, **532**, used: 508, 543
G_GH_ZZS, **501**, **532**, used: 508, 543
G_GLGLA, **450**, used: 463
G_GLGLH, **450**, used: 463
G_GLGLHH, **450**, used: 463
G_GlGlLQLQ, **532**, used: 542
G_GlGlSQSQ, **450**, **501**, **532**, used: 467, 512, 546
G_GlPSQSQ, **450**, **501**, **532**, used: 466, 511, 546
G_GLUGLUA0, **501**, used: 508
G_GLUGLUH0, **501**, used: 508
G_GlWSUSD, **450**, **501**, **532**, used: 466, 511, 545
G_GlZSFSF, **450**, **501**, **532**, used: 466, 511, 546
G_GoSFSF, **450**, used: 470
G_GoSNSL, **450**, used: 470
G_Gr4A_Sd, **450**, used: 476
G_Gr4A_Sdc, **450**, used: 476
G_Gr4A_Sl, **450**, used: 476
G_Gr4A_Slc, **450**, used: 476
G_Gr4A_Su, **450**, used: 476
G_Gr4A_Suc, **450**, used: 476
G_Gr4Gl_Sd, **450**, used: 476
G_Gr4Gl_Sdc, **450**, used: 476
G_Gr4Gl_Su, **450**, used: 476
G_Gr4Gl_Suc, **450**, used: 476
G_Gr4W_Sd, **450**, used: 476
G_Gr4W_Sdc, **450**, used: 476
G_Gr4W_Sl, **450**, used: 476
G_Gr4W_Slc, **450**, used:
G_Gr4W_Sn, **450**, used: 476
G_Gr4W_Snc, **450**, used: 476
G_Gr4W_Su, **450**, used: 476
G_Gr4W_Suc, **450**, used: 476
G_Gr4Z_Sd, **450**, used: 476
G_Gr4Z_Sdc, **450**, used: 476
G_Gr4Z_Sl, **450**, used: 476
G_Gr4Z_Slc, **450**, used: 476
G_Gr4Z_Sn, **450**, used: 476
G_Gr4Z_Snc, **450**, used: 476
G_Gr4Z_Su, **450**, used: 476
G_Gr4Z_Suc, **450**, used: 476
G_Gr4_A_Ch, **450**, used: 476
G_Gr4_H_A, **450**, used: 476

$G\_Gr4\_H\_Z$, **450**, used: 476
G_Gr4_Neu, **450**, used: 476
$G\_Gr4\_W\_H$, **450**, used: 476
$G\_Gr4\_W\_Hc$, **450**, used: 476
$G\_Gr4\_Z\_Ch$, **450**, used: 476
$G\_Gr4\_Z\_H1$, **450**, used: 476
$G\_Gr4\_Z\_H2$, **450**, used: 476
$G\_Gr4\_Z\_H3$, **450**, used: 476
G_Grav, **450**, **337**, **349**, **367**, used: 462, 338, 339, 339, 351, 352, 352, 369, 369
G_GravGl, **450**, used: 476
G_Grav_D, **450**, used: 462
G_Grav_Dc, **450**, used: 462
G_Grav_L, **450**, used: 462
G_Grav_Lc, **450**, used: 462
G_Grav_N, **450**, used: 462
G_Grav_U, **450**, used: 462
G_Grav_Uc, **450**, used: 462
$G\_Gr\_A\_Neu$, **450**, used: 462
G_Gr_Ch, **450**, used: 462
$G\_Gr\_H1\_Neu$, **450**, used: 462
$G\_Gr\_H2\_Neu$, **450**, used: 462
$G\_Gr\_H3\_Neu$, **450**, used: 462
$G\_Gr\_H\_Ch$, **450**, used: 462
$G\_Gr\_Z\_Neu$, **450**, used: 462
G_GSUSD, **450**, used: 470
G_h111, **414**, used: 421
G_h1111, **414**, used: 421
G_h1112, **414**, used: 421
G_h1113, **414**, used: 421
G_h112, **414**, used: 421
G_h1122, **414**, used: 421
G_h1123, **414**, used: 421
G_h113, **414**, used: 421
G_h1133, **414**, used: 421
G_h1222, **414**, used: 421
G_h1223, **414**, used: 421
G_h123, **414**, used: 421
G_h1233, **414**, used: 421
G_h1333, **414**, used: 421
G_h1bb, **414**, used: 416, 418
G_h1bd, **414**, used: 416
G_h1bs, **414**, used: 416
G_h1cc, **414**, used: 416, 417
G_h1ct, **414**, used: 416, 418
G_h1cu, **414**, used: 416, 418
G_h1db, **414**, used: 416, 418
G_h1dd, **414**, used: 416, 418
G_h1ds, **414**, used: 416, 418
G_h1e1e1, **414**, used: 416, 418
G_h1e1e2, **414**, used: 416, 419
G_h1e1e3, **414**, used: 416, 419
G_h1e2e1, **414**, used: 416, 419
G_h1e2e2, **414**, used: 416, 418
G_h1e2e3, **414**, used: 416, 419
G_h1e3e1, **414**, used: 416, 419
G_h1e3e2, **414**, used: 416, 419
G_h1e3e3, **414**, used: 416, 418
$G\_H1GSNSL$, **450**, used: 472
$G\_H1GSUSD$, **450**, used: 473
$G\_H1H1SFSF$, **450**, used: 471, 471
$G\_H1H2SFSF$, **450**, used: 471, 471
$G\_h1HpAWm$, **414**, used: 420
$G\_h1HpAWmC$, **414**, used: 420
G_h1HpHm, **414**, used: 421
$G\_h1HpZWm$, **414**, used: 420
$G\_h1HpZWmC$, **414**, used: 420
G_h1sb, **414**, used: 416, 418
G_h1sd, **414**, used: 416, 418
G_H1SFSF, **450**, used: 469, 469
G_h1ss, **414**, used: 416, 418
G_h1tc, **414**, used: 416, 418
G_h1tt, **414**, used: 416, 417
G_h1tu, **414**, used: 416, 418
G_h1uc, **414**, used: 416, 418
G_h1ut, **414**, used: 416, 418
G_h1uu, **414**, used: 416, 417
G_h1WpWm, **414**, used: 420
G_h1ZZ, **414**, used: 420
G_h221, **414**, used: 421
G_h222, **414**, used: 421
G_h2222, **414**, used: 421
G_h2223, **414**, used: 421
G_h223, **414**, used: 421
G_h2233, **414**, used: 421
G_h2333, **414**, used: 421
G_h2bb, **414**, used: 416, 418
G_h2bd, **414**, used: 416
G_h2bs, **414**, used: 416
G_h2cc, **414**, used: 416, 417
G_h2ct, **414**, used: 416, 418
G_h2cu, **414**, used: 416, 418
G_h2db, **414**, used: 416, 418
G_h2dd, **414**, used: 416, 418
G_h2ds, **414**, used: 416, 418
G_h2e1e1, **414**, used: 416, 418
G_h2e1e2, **414**, used: 416, 419

G_h2e1e3, **414**, used: 416, 419
G_h2e2e1, **414**, used: 416, 419
G_h2e2e2, **414**, used: 416, 418
G_h2e2e3, **414**, used: 416, 419
G_h2e3e1, **414**, used: 416, 419
G_h2e3e2, **414**, used: 416, 419
G_h2e3e3, **414**, used: 416, 418
G_H2GSNSL, **450**, used: 472
G_H2GSUSD, **450**, used: 473
G_H2H2SFSF, **450**, used: 471, 471
G_h2HpAWm, **414**, used: 420
G_h2HpAWmC, **414**, used: 420
G_h2HpHm, **414**, used: 421
G_h2HpZWm, **414**, used: 420
G_h2HpZWmC, **414**, used: 420
G_h2sb, **414**, used: 416, 418
G_h2sd, **414**, used: 416, 418
G_H2SFSF, **450**, used: 469, 469
G_h2ss, **414**, used: 416, 418
G_h2tc, **414**, used: 416, 418
G_h2tt, **414**, used: 416, 417
G_h2tu, **414**, used: 416, 418
G_h2uc, **414**, used: 416, 418
G_h2ut, **414**, used: 416, 418
G_h2uu, **414**, used: 416, 417
G_h2WpWm, **414**, used: 420
G_h2ZZ, **414**, used: 420
G_H3, **241**, **450**, **278**, **300**, **321**, **337**, **349**, **367**, **378**, **388**, **428**, **??**, **??**, used: 251, 467, 467, 290, 311, 326, 339, 352, 369, 380, 390, 429, **??**, **??**
G_h331, **414**, used: 421
G_h332, **414**, used: 421
G_h333, **414**, used: 421
G_h3333, **414**, used: 421
G_h3bb, **414**, used: 416, 418
G_h3bd, **414**, used: 416
G_h3bs, **414**, used: 416
G_h3cc, **414**, used: 416, 417
G_h3ct, **414**, used: 416, 418
G_h3cu, **414**, used: 416, 418
G_h3db, **414**, used: 416, 418
G_h3dd, **414**, used: 416, 418
G_h3ds, **414**, used: 416, 418
G_h3e1e1, **414**, used: 416, 418
G_h3e1e2, **414**, used: 416, 419
G_h3e1e3, **414**, used: 416, 419
G_h3e2e1, **414**, used: 416, 419
G_h3e2e2, **414**, used: 416, 418
G_h3e2e3, **414**, used: 416, 419
G_h3e3e1, **414**, used: 416, 419
G_h3e3e2, **414**, used: 416, 419
G_h3e3e3, **414**, used: 416, 418
G_h3HpAWm, **414**, used: 420
G_h3HpAWmC, **414**, used: 420
G_h3HpHm, **414**, used: 421
G_h3HpZWm, **414**, used: 420
G_h3HpZWmC, **414**, used: 420
G_h3sb, **414**, used: 416
G_h3sd, **414**, used: 416, 418
G_h3ss, **414**, used: 416, 418
G_h3tc, **414**, used: 416, 418
G_h3tt, **414**, used: 416, 417
G_h3tu, **414**, used: 416, 418
G_h3uc, **414**, used: 416, 418
G_h3ut, **414**, used: 416, 418
G_h3uu, **414**, used: 416, 417
G_h3WpWm, **414**, used: 420
G_h3ZZ, **414**, used: 420
G_H3_SCC, **501**, **532**, used: 512, 546
G_H3_SPP, **501**, **532**, used: 512, 546
G_H3_SSS, **501**, **532**, used: 512, 546
G_H4, **241**, **450**, **278**, **300**, **321**, **337**, **349**, **367**, **378**, **428**, **??**, **??**, used: 251, 467, 468, 290, 311, 326, 339, 352, 369, 380, 429, **??**, **??**
G_H4_1, **388**, used: 390
G_H4_2, **388**, used: 390
G_H4_3, **388**, used: 390
G_H4_4, **388**, used: 390
G_H4_5, **388**, used: 390
G_HAHAH, **278**, **300**, used: 285, 306
G_HAHZ, **278**, **300**, used: 285, 306
G_HASLSN, **450**, used: 471
G_HASUSD, **450**, used: 472
G_hbb, **414**, used:
G_Hbb, **241**, **278**, **300**, **321**, **337**, **349**, **367**, **378**, **388**, **414**, **428**, **??**, **??**, used: 246, 282, 304, 325, 338,

351, 368, 379, 390, 428, ??, ??
G_hbc, **414**, used: 416
G_hbt, **414**, used: 416, 418
G_hbu, **414**, used: 416
G_hcb, **414**, used: 416
G_hcc, **414**, used:
G_Hcc, **241**, **278**, **300**, **321**, **337**, **349**, **367**, **378**, **388**, **414**, **428**, ??, ??, used: 246, 282, 304, 325, 338, 351, 368, 379, 390, 428, ??, ??
G_hcd, **414**, used: 416
G_hcs, **414**, used: 416, 418
G_Hcs, **414**, used:
G_hdc, **414**, used: 416
G_hdt, **414**, used: 416
G_hdu, **414**, used: 416, 418
G_he1n1, **414**, used: 416
G_he1n2, **414**, used: 416
G_he1n3, **414**, used: 416
G_he2n1, **414**, used: 416
G_he2n2, **414**, used: 416
G_he2n3, **414**, used: 416
G_he3n1, **414**, used: 416
G_he3n2, **414**, used: 416
G_he3n3, **414**, used: 416
G_heavy_HVV, **321**, used: 326
G_heavy_HWW, **321**, used: 326
G_heavy_HZZ, **321**, used:
G_HGaGa, **241**, **278**, **300**, **321**, **337**, **349**, **367**, **378**, **388**, **428**, used: 253, 339, 352, 369, 391, 430
G_HGaGa_anom, **241**, **428**, used: 252, 430
G_HGaZ, **241**, **278**, **300**, **321**, **337**, **349**, **367**, **378**, **388**, **428**, used: 253, 339, 352, 369, 391, 430
G_HGaZ_anom, **241**, **428**, used: 252, 430
G_HGaZ_u, **241**, **428**, used: 252, 430
G_Hgg, **241**, **321**, **337**, **349**, **367**, **378**, **388**, **428**, used: 253, 339, 352, 369, 391, 430
G_HGo3, **450**, used: 467
G_HGo4, **450**, used: 468
G_HGSFSF, **450**, used: 472, 473
G_HGSNSL, **450**, used: 472
G_HGSUSD, **450**, used: 473
G_HH1SLSN, **450**, used: 471
G_HH1SUSD, **450**, used: 472
G_HH2SLSN, **450**, used: 471
G_HH2SUSD, **450**, used: 472
G_HHAA, **278**, **300**, used: 287, 308
G_HHAHZ, **278**, **300**, used: 287, 308
G_HHS, **388**, used: 390
G_HHSFSF, **450**, used: 471, 471
G_HHtht, **278**, **300**, used: 290, 312
G_HHthth, **278**, **300**, **321**, used: 290, 312, 327
G_HHtt, **278**, **300**, used: 290, 312
G_HHWHW, **278**, **300**, used: 287, 308
G_hhWpWm, **414**, used: 420
G_HHWW, **241**, **278**, **300**, **321**, **337**, **349**, **367**, **378**, **388**, **428**, ??, ??, used: 251, 287, 287, 308, 308, 326, 326, 339, 352, 369, 380, 390, 429, ??, ??
G_HHZHAH, **278**, **300**, used: 287, 308
G_HHZHZ, **278**, **300**, used: 287, 308
G_hhZZ, **414**, used: 420
G_HHZZ, **241**, **278**, **300**, **321**, **337**, **349**, **367**, **378**, **388**, **428**, ??, ??, used: 251, 287, 287, 308, 308, 326, 326, 339, 352, 369, 380, 390, 429, ??, ??
G_HHZZH, **321**, used: 326
G_Hmm, **241**, **378**, **388**, **428**, used: 246, 379, 390, 428
G_hmumu, **414**, used:
G_Hmumu, **414**, used:
G_Hmunu, **414**, used:
G_hn1e1, **414**, used: 416
G_hn1e2, **414**, used: 416
G_hn1e3, **414**, used: 416
G_hn2e1, **414**, used: 416

G_hn2e2, **414**, used: 416
G_hn2e3, **414**, used: 416
G_hn3e1, **414**, used: 416
G_hn3e2, **414**, used: 416
G_hn3e3, **414**, used: 416
G_HpHm11, **414**, used: 421
G_HpHm12, **414**, used: 421
G_HpHm13, **414**, used: 421
G_HpHm22, **414**, used: 421
G_HpHm23, **414**, used: 421
G_HpHm33, **414**, used: 421
G_HpHmAA, **414**, used: 420
G_HpHmAZ, **414**, used: 420
G_HpHmHpHm, **414**, used: 421
G_HpHmWpWm, **414**, used: 420
G_HpHmZZ, **414**, used: 420
G_HPsi0AHAH, **278**, **300**, used: 287, 308
G_HPsi0WHW, **278**, **300**, used: 287, 308
G_HPsi0WW, **278**, **300**, used: 287, 308
G_HPsi0ZAH, **278**, **300**, used: 287, 308
G_HPsi0ZHAH, **278**, **300**, used: 287, 308
G_HPsi0ZHZ, **278**, **300**, used: 287, 308
G_HPsi0ZHZH, **278**, **300**, used: 287, 308
G_HPsi0ZZ, **278**, **300**, used: 287, 308
G_HPsippWHW, **278**, **300**, used: 287, 308
G_HPsippWHWH, **278**, **300**, used: 287, 308
G_HPsippWW, **278**, **300**, used: 287, 308
G_HPsipWA, **278**, **300**, used: 287, 308
G_HPsipWAH, **278**, **300**, used: 287, 308
G_HPsipWHA, **278**, **300**, used: 287, 308
G_HPsipWHAH, **278**, **300**, used: 287, 308
G_HPsipWHZ, **278**, **300**, used: 287, 308
G_HPsipWHZH, **278**, **300**, used: 287, 308
G_HPsipWZ, **278**, **300**, used: 287, 308
G_HPsipWZH, **278**, **300**, used: 287, 308
G_Hqhq, **321**, used: 325
G_hsc, **414**, used: 416, 418
G_HSF31, **450**, used:
G_HSF32, **450**, used:
G_HSF41, **450**, used:
G_HSF42, **450**, used:
G_HSNSL, **450**, **501**, **532**, used: 469, 512, 547
G_HSS, **388**, used: 390
G_hst, **414**, used: 416
G_hsu, **414**, used: 416
G_HSUSD, **450**, **501**, **532**, used: 470, 513, 547
G_HSWW, **388**, used: 390
G_HSZZ, **388**, used: 390
G_Htaunu, **414**, used:
G_htautau, **414**, used:
G_Htautau, **241**, **278**, **300**, **321**, **337**, **349**, **367**, **378**, **388**, **414**, **428**, **??**, **??**, used: 246, 282, 304, 325, 338, 351, 368, 379, 390, 428, **??**, **??**
G_htb, **414**, used: 416, 418
G_Htb, **414**, used:
G_htd, **414**, used: 416
G_Htht, **278**, **300**, **321**, used: 282, 304, 325
G_Hthth, **278**, **300**, **321**, used: 282, 304
G_hts, **414**, used: 416
G_htt, **414**, used:
G_Htt, **241**, **278**, **300**, **321**, **337**, **349**, **367**, **378**, **388**, **414**, **428**, **??**, **??**, used: 246, 282, 304, 325, 338, 351, 368, 379, 390, 428, **??**, **??**
G_hub, **414**, used: 416
G_hud, **414**, used: 416, 418
G_hus, **414**, used: 416
G_HWHW, **278**, **300**, used: 285, 306
G_HWHWH, **278**, **300**, used: 285, 306
G_HWW, **241**, **278**, **300**, **321**, **337**, **349**, **367**, **378**, **388**,

428, ??, ??, used: 251, 285, 306, 326, 339, 352, 369, 380, 390, 428, ??, ??
G_HWW_anom, **241**, **428**, used: 252, 430
G_HWW_u, **241**, **428**, used: 252, 430
G_HZHAH, **278**, **300**, used: 285, 306
G_HZHZ, **278**, **300**, used: 285, 306
G_HZZ, **241**, **278**, **300**, **321**, **337**, **349**, **367**, **378**, **388**, **428**, ??, ??, used: 251, 285, 306, 326, 339, 352, 369, 380, 390, 428, ??, ??
G_HZZ_anom, **241**, **428**, used: 252, 430
G_HZZ_u, **241**, **428**, used: 252, 430
G_LQ_EC_UC, **532**, used: 539, 539, 540, 540, 540, 540, 540, 540, 540, 540, 540, 540
G_LQ_GG, **532**, used: 541
G_LQ_NEU, **532**, used: 541, 541
G_LQ_P, **532**, used: 541
G_LQ_S, **532**, used: 541
G_LQ_SSD, **532**, used: 541
G_LQ_SSU, **532**, used: 540
G_NCHt, **321**, used: 323
G_NCH_D, **321**, used: 324
G_NCH_N, **321**, used: 324
G_NCH_U, **321**, used: 324
G_NC_down, **241**, **450**, ??, ??, **501**, **532**, **278**, **300**, **321**, **337**, **349**, **367**, **378**, **388**, **414**, **428**, ??, ??, used: 245, 246, 455, ??, ??, 503, 535, 280, 281, 302, 303, 322, 323, 337, 338, 350, 350, 368, 368, 379, 379, 389, 389, 416, 417, 428, ??, ??, ??
G_NC_H, **321**, used: 323
G_NC_heavy, **278**, **300**, used: 281, 303
G_NC_h_bot, **321**, used: 324
G_NC_h_down, **278**, **300**, **321**, ??, used: 280, 281, 302, 303, 322, 324, ??, ??
G_NC_h_lepton, **278**, **300**, **321**, ??, used: 280, 281, 302, 303, 322, 324, ??, ??
G_NC_h_neutrino, **278**, **300**, **321**, ??, used: 280, 281, 302, 303, 322, 324, ??, ??
G_NC_h_top, **321**, used: 324
G_NC_h_up, **278**, **300**, **321**, ??, used: 280, 282, 302, 304, 322, 324, ??, ??
G_NC_lepton, **241**, **450**, ??, ??, **501**, **532**, **278**, **300**, **321**, **337**, **349**, **367**, **378**, **388**, **414**, **428**, ??, ??, used: 245, 246, 455, ??, ??, 503, 535, 280, 281, 302, 303, 322, 323, 337, 338, 350, 350, 368, 368, 379, 379, 389, 389, 416, 417, 428, ??, ??, ??
G_NC_neutrino, **241**, **450**, ??, ??, **501**, **532**, **278**, **300**, **321**, **337**, **349**, **367**, **378**, **388**, **414**, **428**, ??, ??, used: 245, 246, 455, ??, ??, 503, 535, 280, 281, 302, 303, 322, 323, 337, 338, 350, 350, 368, 368, 379, 379, 389, 389, 416, 417, 428, ??, ??, ??
G_NC_top, ??, used: ??
G_NC_up, **241**, **450**, ??, ??, **501**, **532**, **278**, **300**, **321**, **337**, **349**, **367**, **378**, **388**, **414**, **428**, ??, ??, used: 245, 246, 455, ??, ??, 503, 535, 280, 281, 302, 303, 322, 323, 337, 338, 350, 350, 368, 368, 379, 379, 389, 389, 416, 417, 428, ??, ??, ??
G_NC_X, **321**, used: 323
G_NC_X_t, **321**, used: 323
G_NC_Y, **321**, used: 323
G_NC_Y_t, **321**, used: 323
G_NGC, **450**, used: 458

G_NHC, **450**, **501**, **532**, used: 458, 506, 537
G_NLQC, **532**, used: 542
G_NWC, **450**, **501**, **532**, used: 456, 504, 535
G_NZN, **450**, **501**, **532**, used: 457, 505, 536
$g\_over\_2\_costh$, **245**, **??**, **??**, **280**, **302**, **322**, **337**, **350**, **367**, **378**, **389**, **416**, **428**, **??**, used: 245, ??, ??, 280, 302, 322, 337, 350, 367, 378, 389, 416, 428, ??
$G\_PGlLQLQ$, **532**, used: 542
G_PNWW, **??**, **428**, used: ??, 432
G_PNZZ, **??**, **428**, used: ??, 432
G_PPA, **450**, used: 463
G_PPH, **450**, used: 463
G_PPHH, **450**, used: 463
G_PPLQLQ, **532**, used: 542
G_PPSFSF, **450**, **501**, **532**, used: 465, 510, 545
G_PPWW, **450**, **501**, **532**, used: 463, 508, 542
$G\_Psi00AH$, **278**, **300**, used: 287, 308
$G\_Psi00ZH$, **278**, **300**, used: 287, 308
$G\_Psi00ZHAH$, **278**, **300**, used: 287, 308
$G\_Psi01AH$, **278**, **300**, used: 286, 307
G_Psi01Z, **278**, **300**, used: 286, 307
$G\_Psi01ZH$, **278**, **300**, used: 286, 307
G_Psi0bb, **278**, **300**, used: 282, 304
G_Psi0cc, **278**, **300**, used: 282, 304
$G\_Psi0ppWHW$, **278**, **300**, used: 287, 308
$G\_Psi0ppWHWH$, **278**, **300**, used: 287, 308
$G\_Psi0ppWW$, **278**, **300**, used: 287, 308
$G\_Psi0pWA$, **278**, **300**, used: 287, 308
$G\_Psi0pWAH$, **278**, **300**, used: 287, 308
$G\_Psi0pWHA$, **278**, **300**, used: 287, 308
$G\_Psi0pWHAH$, **278**, **300**, used: 287, 308
$G\_Psi0pWHZ$, **278**, **300**, used: 287, 308
$G\_Psi0pWHZH$, **278**, **300**, used: 287, 308
$G\_Psi0pWZ$, **278**, **300**, used: 287, 308
$G\_Psi0pWZH$, **278**, **300**, used: 287, 308
$G\_Psi0tautau$, **278**, **300**, used: 282, 304
G_Psi0tt, **278**, **300**, used: 282, 304
$G\_Psi0tth$, **278**, **300**, used: 282, 304
G_Psi0W, **278**, **300**, used: 286, 307
G_Psi0WH, **278**, **300**, used: 286, 307
G_Psi1bb, **278**, **300**, used: 282, 304
G_Psi1cc, **278**, **300**, used: 282, 304
$G\_Psi1HAH$, **278**, **300**, used: 286, 307
G_Psi1HZ, **278**, **300**, used: 286, 307
$G\_Psi1HZH$, **278**, **300**, used: 286, 307
$G\_Psi1tautau$, **278**, **300**, used: 282, 304
G_Psi1tt, **278**, **300**, used: 282, 304
$G\_Psi1tth$, **278**, **300**, used: 282, 304
G_Psi1W, **278**, **300**, used: 286, 307
G_Psi1WH, **278**, **300**, used: 286, 307
$G\_PsiAHAH$, **278**, **300**, used: 285, 307
G_PsiAHW, **278**, **300**, used: 285, 307
$G\_PsiAHWH$, **278**, **300**, used: 285, 307

G_PsiccAAH, **278**, **300**, used: 287, 308
G_PsiccAZ, **278**, **300**, used: 287, 308
G_PsiccAZH, **278**, **300**, used: 287, 308
G_PsiccZAH, **278**, **300**, used: 287, 308
G_PsiccZZ, **278**, **300**, used: 287, 308
G_PsiccZZH, **278**, **300**, used: 287, 308
G_PsiHW, **278**, **300**, used: 286, 307
G_PsiHWH, **278**, **300**, used: 286, 307
G_Psipbth, **278**, **300**, used: 282, 304
G_Psipl3, **278**, **300**, used: 282, 304
G_PsippAAH, **278**, **300**, used: 287, 308
G_PsippAZ, **278**, **300**, used: 287, 308
G_PsiPPW, **278**, **300**, used: 286, 307
G_PsippWA, **278**, **300**, used: 287, 308
G_PsippWAH, **278**, **300**, used: 287, 308
G_PsiPPWH, **278**, **300**, used: 286, 307
G_PsippWHA, **278**, **300**, used: 287, 308
G_PsippWHAH, **278**, **300**, used: 287, 308
G_PsippWHW, **278**, **300**, used: 285, 307
G_PsippWHWH, **278**, **300**, used: 285, 307
G_PsippWHZ, **278**, **300**, used: 287, 308
G_PsippWHZH, **278**, **300**, used: 287, 308
G_PsippWW, **278**, **300**, used: 285, 307
G_PsippWZ, **278**, **300**, used: 287, 308
G_PsippWZH, **278**, **300**, used: 287, 308
G_PsippZAH, **278**, **300**, used: 287, 308
G_PsippZHZH, **278**, **300**, used: 287, 308
G_PsippZZ, **278**, **300**, used: 287, 308
G_Psipq2, **278**, **300**, used: 282, 304
G_Psipq3, **278**, **300**, used: 282, 304
G_PsiWHW, **278**, **300**, used: 285, 307
G_PsiWW, **278**, **300**, used: 285, 307
G_PsiZAH, **278**, **300**, used: 285, 307
G_PsiZHAH, **278**, **300**, used: 285, 307
G_PsiZHW, **278**, **300**, used: 285, 307
G_PsiZHWH, **278**, **300**, used: 285, 307
G_PsiZHZ, **278**, **300**, used: 285, 307
G_PsiZHZH, **278**, **300**, used: 285, 307
G_PsiZW, **278**, **300**, used: 285, 307
G_PsiZWH, **278**, **300**, used: 285, 307
G_PsiZZ, **278**, **300**, used: 285, 307
G_PWW, **??**, **428**, used: **??**, 432
G_PWZ, **??**, **428**, used: **??**, 432
G_PZWW, **450**, **501**, **532**, used: 463, 508, 542
G_s, **401**, used: 404
G_s2, **401**, used: 406
G_saa, **719**, used: 720
G_saaa, **719**, used:
G_Sbb, **388**, used: 390
G_Scc, **388**, used: 390
G_SD4, **450**, used: 475
G_SD4_2, **450**, used: 475
G_SF4, **450**, used:
G_SF4_3, **450**, used:
G_SF4_4, **450**, used:
G_SFSFP, **501**, **532**, used: 513, 547

G_SFSFS, **501**, **532**, used: 512, 513, 547, 547
G_SGaGa, **388**, used: 391
G_SGaZ, **388**, used: 391
G_Sgg, **388**, used: 391
G_SL2SQ2, **450**, used: 474
G_SL4, **450**, used: 474
G_SL4_2, **450**, used: 474
G_SLR_DbtR, **241**, ??, ??, **428**, used: 257, ??, ??, 436
G_SLSNW, **450**, **501**, **532**, used: 456, 504, 535
G_SL_DbtL, **241**, ??, ??, **428**, used: 257, ??, ??, 436
G_SL_DttL, **241**, ??, ??, **428**, used: 257, ??, ??, 436
G_SL_DttR, **241**, ??, ??, **428**, used: 257, ??, ??, 436
G_Smm, **388**, used: 390
G_SN2SL2_1, **450**, used: 474
G_SN2SL2_2, **450**, used: 474
G_SN2SQ2, **450**, used: 474
G_SN4, **450**, used: 474
G_SP_ttH, **241**, **428**, used: 255, 434
G_SR_DttR, **241**, ??, ??, **428**, used: 257, ??, ??, 436
G_SS, **450**, **501**, **532**, used: 463, 508, 542
G_SSWW, ??, **388**, **428**, used: ??, 390, 432
G_SSZZ, ??, **388**, **428**, used: ??, 390, 432
G_Stautau, **388**, used: 390
G_strong, **450**, **532**, **367**, used:
G_Stt, **388**, used: 390
G_SU2SD2, **450**, used: 475
G_SU4, **450**, used: 475
G_SU4_2, **450**, used: 475
G_SUSDSNSL, **450**, used: 475
G_SWS, **450**, used: 456
G_SWW, ??, **388**, **428**, used: ??, 390, 431
G_SWW_T, ??, **428**, used: ??, 431
G_SZZ, ??, **388**, **428**, used: ??, 390, 431
G_SZZ_T, ??, **428**, used: ??, 431
G_S_Sqrt, **450**, used: 459
G_TLR_btW, **241**, ??, ??, **428**, used: 255, ??, ??, 433
G_TLR_btWA, **241**, ??, ??, **428**, used: 255, ??, ??, 434
G_TLR_btWZ, **241**, ??, ??, **428**, used: 256, ??, ??, 434
G_TNWW, ??, **428**, used: ??, 432
G_TNZZ, ??, **428**, used: ??, 432
G_TRL_tbW, **241**, ??, ??, **428**, used: 255, ??, ??, 433
G_TRL_tbWA, **241**, ??, ??, **428**, used: 255, ??, ??, 434
G_TRL_tbWZ, **241**, ??, ??, **428**, used: 256, ??, ??, 434
G_TVA_bbA, **241**, ??, ??, **428**, used: 254, ??, ??, 433
G_TVA_bbWW, **241**, ??, ??, **428**, used: 256, ??, ??, 435
G_TVA_bbZ, **241**, ??, ??, **428**, used: 254, ??, ??, 433
G_TVA_ttA, **241**, ??, ??, **428**, used: 253, ??, ??, 432
G_TVA_ttG, **241**, ??, ??, **428**, used: 254, ??, ??, 433
G_TVA_ttGG, **241**, ??, ??, **428**, used: 255, ??, ??, 434
G_TVA_ttWW, **241**, ??, ??, **428**, used: 256, ??, ??, 435
G_TVA_ttZ, **241**, ??, ??, **428**, used: 254, ??, ??, 433
G_TWW, ??, **428**, used: ??, 432
G_TWZ, ??, **428**, used: ??, 432
G_VLR_btW, **241**, ??, ??, **428**, used: 255, ??, ??, 433
G_VLR_qBuB, **241**, ??, ??, **428**, used: 257, ??, ??, 435
G_VLR_qBuB_d, **241**, ??, ??, **428**, used: 257, ??, ??, 435
G_VLR_qBuB_e, **241**, ??, ??, **428**, used: 257, ??, ??, 435
G_VLR_qBuB_u, **241**, ??, ??, **428**, used: 257, ??, ??, 435
G_VLR_qGuG, **241**, ??, ??, **428**, used: 256, ??, ??,

435
G_VLR_tbW, **241**, ??, ??, **428**, used: 255, ??, ??, 433
G_VLR_ttZ, **241**, ??, ??, **428**, used: 254, ??, ??, 433
$G\_VL\_qBuB\_n$, **241**, ??, ??, **428**, used: 257, ??, ??, 435
G_VL_qW, **241**, ??, ??, **428**, used: 257, ??, ??, 435
$G\_VL\_qW\_d$, **241**, ??, ??, **428**, used: 257, ??, ??, 436
$G\_VL\_qW\_u$, **241**, ??, ??, **428**, used: 257, ??, ??, 436
G_weak, **241**, ??, ??, **278**, **300**, **321**, **337**, **349**, **367**, **378**, **388**, **414**, **428**, ??, ??, used: 244, 245, ??, ??, 280, 302, 322, 337, 350, 367, 378, 389, 416, 428, ??
G_WH3W, **278**, **300**, used: 284, 305
G_WH4, **278**, **300**, used: 284, 305
G_WHWAAH, **278**, **300**, used: 284, 305
G_WHWAZ, **278**, **300**, used: 284, 305
G_WHWAZH, **278**, **300**, used: 284, 305
$G\_WHWHAAH$, **278**, **300**, used: 284, 305
$G\_WHWHAZH$, **278**, **300**, used: 284, 305
G_WHWHWW, **278**, **300**, used: 284, 305
$G\_WHWHZAH$, **278**, **300**, used: 284, 305
$G\_WHWHZHAH$, **278**, **300**, used: 284, 305
$G\_WHWHZZH$, **278**, **300**, used: 284, 305
G_WHWWW, **278**, **300**, used: 284, 305
G_WHWZAH, **278**, **300**, used: 284, 305
$G\_WHWZHAH$, **278**, **300**, used: 284, 305
$G\_WHWZHZH$, **278**, **300**, used: 284, 305
G_WHWZZ, **278**, **300**, used: 284, 305
G_WHWZZH, **278**, **300**, used: 284, 305
G_WmHph1, **414**, used: 420
G_WmHph2, **414**, used: 420
G_WmHph3, **414**, used: 420
G_WpHmh1, **414**, used: 420
G_WpHmh2, **414**, used: 420
G_WpHmh3, **414**, used: 420
G_WPSLSN, **450**, **501**, **532**, used: 465, 510, 545
G_WPSUSD, **450**, **501**, **532**, used: 466, 511, 545
G_WSQ, **501**, **532**, used: 504, 536
G_WWAAH, **278**, **300**, used: 284, 305
G_WWAZH, **278**, **300**, used: 284, 305
G_WWSFSF, **450**, **501**, **532**, used: 465, 510, 511, 544, 545
g_wwww, **402**, used: 402
G_wwww, **401**, used: 402, 405
G_WWWW, **241**, **450**, ??, ??, **501**, **532**, **278**, **300**, **321**, **337**, **349**, **367**, **378**, **388**, **414**, **428**, ??, ??, used: 248, 463, ??, ??, 508, 542, 284, 284, 305, 305, 326, 338, 351, 369, 380, 390, 420, 428, ??, ??
g_wwz, **402**, used: 402
G_wwz, **401**, used: 402, 405
g_wwza, **402**, used: 402
G_wwza, **401**, used: 402, 405
G_WWZAH, **278**, **300**, used: 284, 305
G_WWZHAH, **278**, **300**, used: 284, 305
g_wwzz, **402**, used: 402
G_wwzz, **401**, used: 402, 405
G_WWZZH, **278**, **300**, used: 284, 305
G_WZSLSN, **450**, **501**, **532**, used: 465, 510, 545
G_WZSUSD, **450**, **501**, **532**, used: 466, 511, 545
g_w_lep, **402**, used: 402

G_w_lep, **401**, used: 402, 403
g_w_quark, **402**, used: 402
G_w_quark, **401**, used: 402, 403, 404
G_YUK, **450**, used: 457, 457
G_YUK_1, **450**, used: 457
G_YUK_2, **450**, used: 457
G_YUK_3, **450**, used: 458
G_YUK_4, **450**, used: 458
G_YUK_C, **450**, **501**, **532**, used: 461, 507, 538
G_YUK_DCU, **501**, **532**, used: 505, 537
G_YUK_FFP, **501**, **532**, used: 505, 536
G_YUK_FFS, **501**, **532**, used: 505, 536
G_YUK_G, **450**, **501**, **532**, used: 459, 507, 538
G_YUK_LCN, **501**, **532**, used: 505, 537
G_YUK_LQ_P, **532**, used: 541
G_YUK_LQ_S, **532**, used: 541
G_YUK_N, **450**, **501**, **532**, used: 459, 506, 538
G_YUK_Q, **450**, **501**, **532**, used: 461, 507, 539
G_YUK_UCD, **501**, **532**, used: 505, 537
G_Z, **450**, **501**, **532**, used:
G_ZEH, **321**, used: 327
G_ZGlLQLQ, **532**, used: 542
G_Zh1h2, **414**, used: 420
G_Zh1h3, **414**, used: 420
G_Zh2h3, **414**, used: 420
G_ZHEH, **321**, used: 327
G_ZHpHm, **414**, used: 420
G_ZHPsipp, **278**, **300**, used: 286, 307
G_ZHTHT, **278**, **300**, used: 281, 303
G_zhthth, **321**, used:
G_ZLQ, **532**, used: 542
G_ZPLQLQ, **532**, used: 542
G_ZPSFSF, **450**, **501**, **532**, used: 465, 510, 544
G_ZPsip, **278**, **300**, used: 286, 307
G_ZPsipp, **278**, **300**, used: 286, 307
G_ZSF, **450**, **501**, **532**, used: 456, 505, 536
G_ZTHT, **278**, **300**, used: 281, 303
G_ZZLQLQ, **532**, used: 542
G_ZZSFSF, **450**, **501**, **532**, used: 465, 510, 511, 544, 545
G_ZZWW, **241**, **450**, **??**, **??**, **501**, **532**, **278**, **300**, **321**, **337**, **349**, **367**, **378**, **388**, **414**, **428**, **??**, **??**, used: 248, 463, ??, ??, 508, 542, 284, 284, 305, 305, 326, 338, 351, 369, 380, 390, 420, 428, ??, ??
g_z_lep, **402**, used: 402
G_z_lep, **401**, used: 402, 404
g_z_quark, **402**, used: 402
G_z_quark, **401**, used: 402, 404
H, **236**, **274**, **297**, **319**, **333**, **344**, **363**, **374**, **384**, **428**, **??**, **??**, used: 237, 239, 246, 251, 251, 251, 251, 252, 253, 255, 259, 275, 277, 282, 282, 285, 285, 286, 287, 287, 290, 290, 290, 291, 298, 299, 304, 304, 306, 306, 307, 308, 308, 311, 311, 312, 312, 319, 320, 325, 325, 326, 326, 326, 326, 326, 326, 327, 327, 327, 333, 335, 338, 339, 339, 339, 339, 339, 339, 340, 344, 347, 351, 352, 352, 352, 352, 352, 352, 353, 363, 365, 368, 369, 369, 369, 369, 369, 370, 375, 376, 379, 380, 380, 380, 380, 381, 385, 386, 390, 390, 390, 390, 390, 391, 391, 428, 428, 429, 429, 429, 430, 430, 434, 437, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??
H1, **447**, used:
H1dm, **344**, used: 344, 347, 353
H1dp, **344**, used: 344, 347, 353
H1um, **344**, used: 344, 347, 353

H1up, **344**, used: 344, 347, 353
H2, **447**, used:
H2 (module), **219**, used: 219, 219, 220, 221
H2dm, **344**, used: 344, 347, 353
H2dp, **344**, used: 344, 347, 353
H2um, **344**, used: 344, 347, 353
H2up, **344**, used: 344, 347, 353
H3, **447**, used:
H3 (module), **219**, used: 219, 219, 220, 221
H4, **447**, used:
H5, **447**, used:
H6, **447**, used:
HA, **411**, used: 411, 412, 417, 418, 418, 418, 418, 419, 420, 420, 421, 421, 422
half, **245**, **??**, **??**, **280**, **302**, **322**, **337**, **350**, **368**, **379**, **389**, **416**, **428**, **??**, used: 245, ??, ??, 280, 302, 322, 337, 350, 368, 379, 389, 416, 428, ??
Half, **241**, **??**, **??**, **428**, used: 257, 257, 258, 258, ??, ??, ??, ??, ??, ??, 436, 436
harvest, **41**, **47**, **38**, used: 47, 47, 48, 111
harvest_goldstones, **111**, used:
harvest_list, **41**, **47**, **38**, used: 117, 120
hash, **737**, **737**, **218**, **219**, **219**, **105**, **736**, used: 218, 219, 219, 105, 105, 105
hash (type), **737**, **737**, **736**, used: 737, 737, 736
HC1, **528**, used: 528, 529, 537, 537, 537, 543, 543, 543, 543, 544, 544, 544, 544, 544, 546, 547, 547, 549
HC1c, **528**, used: 528, 529, 537, 537, 537, 543, 543, 543, 543, 544, 544, 544, 544, 544, 546, 547, 547, 549
HC2, **528**, used: 528, 529
HC2c, **528**, used: 528, 529
HC3, **528**, used: 528, 529
HC3c, **528**, used: 528, 529
HC4, **528**, used: 528, 529
HC4c, **528**, used: 528, 529
HC5, **528**, used: 528, 529
HC5c, **528**, used: 528, 529
hdn, **742**, **740**, used: 742
hdtl, **793**, used: 793, 794
header, **710**, used: 711
header (field), **813**, **807**, used: 814, 815, 711, 711, 711
header_sans_color, **710**, used: 711, 711
head_req, **205**, used: 205
Heavy, **397**, used: 398, 398, 405, 405, 407
Heavy2, **397**, used: 398, 402
heavy_currents, **325**, used: 327
heavy_gauge_higgs, **285**, **306**, **326**, used: 287, 308, 326
heavy_gauge_higgs4, **326**, used: 327
heavy_quartic_gauge, **284**, **305**, **326**, used: 285, 306, 326
heavy_top_currents, **281**, **303**, used: 291, 312
heavy_triple_gauge, **283**, **305**, **325**, used: 284, 305, 326
height, **756**, used: 756, 756, 756
Helac (module), **33**, **127**, **20**, **91**, used: 127
Helac_Binary (module), **33**, **20**, used:
Helac_Majorana (module), **127**, **91**, used:
helicities, **127**, **129**, **92**, used:
helicities (field), **129**, used: 129, 134
helicities (type), **129**, used:
helicity_table, **131**, used: 133
hgg, **281**, used: 285, 285, 285, 290, 290
Hh, **411**, used: 411, 412, 417, 418, 418, 418, 418, 419, 420, 420, 421, 421, 422
HH, **411**, used: 411, 412, 417, 418, 418, 418, 418, 419, 420, 420, 421, 421, 422
hhgg, **281**, used: 287, 287
higgs, **253**, **467**, **512**, **546**, **290**, **312**, **327**, **339**, **352**, **370**, **380**, **390**, **421**, **431**, **??**, **??**, used: 258, 477, 513, 548, 291, 312, 327, 340,

353, 370, 380, 391, 422, 436, ??, ??
higgs (type), **447**, used: 447, 450
higgs4, **253**, **467**, **512**, **546**, **290**, **312**, **327**, **340**, **352**, **370**, **380**, **390**, **421**, **431**, ??, ??, used: 259, 478, 514, 549, 291, 312, 327, 340, 353, 370, 381, 391, 422, 437, ??, ??
higgs_anom, **234**, **234**, **234**, **234**, **235**, **235**, **235**, **235**, **236**, **427**, **428**, **224**, **273**, used: 253, 253, 253, 253, 430, 430, 431, 431
higgs_charg_neutr, **458**, **506**, **537**, used: 477, 513, 548
higgs_ch_gravitino, **462**, used: 463
higgs_gold, **467**, used: 478
higgs_gold4, **468**, used: 478
higgs_gold_sfermion, **473**, used: 478
higgs_gold_sfermion', **473**, used: 473
higgs_gold_sneutrino, **472**, used: 478
higgs_gold_sneutrino', **472**, used: 472
higgs_gold_squark, **473**, used: 478
higgs_gold_squark', **473**, used: 473
higgs_hmm, **234**, **234**, **234**, **234**, **235**, **235**, **235**, **235**, **236**, **427**, **428**, **224**, **273**, used: 246, 428
higgs_neutr, **459**, **506**, **538**, used: 477, 513, 548
higgs_neutr', **459**, used: 459
higgs_neutr'', **459**, used: 459
higgs_neu_gravitino, **462**, used: 463
higgs_SCC, **512**, **546**, used: 512, 546
higgs_sfermion, **469**, **513**, **547**, used: 477, 513, 548
higgs_sfermion', **469**, **513**, **547**, used: 469, 513, 547
higgs_sfermion'', **469**, **513**, **547**, used: 469, 513, 547
higgs_sfermion4, **471**, used: 478
higgs_sfermion4', **471**, used: 471
higgs_sfermion_P, **513**, **547**, used: 513, 547
higgs_sfermion_S, **513**, **547**, used: 513, 547
higgs_sneutrino, **469**, **512**, **547**, used: 477, 513, 548
higgs_sneutrino', **469**, **512**, **547**, used: 469, 512, 547
higgs_sneutrino'', **469**, **512**, **547**, used: 469, 512, 547
higgs_sneutrino4, **471**, used: 478
higgs_sneutrino4', **471**, used: 471
higgs_SPP, **512**, **546**, used: 512, 546
higgs_squark, **470**, **513**, **547**, used: 477, 513, 548
higgs_squark', **470**, **513**, **547**, used: 470, 513, 547
higgs_squark4, **472**, used: 478
higgs_squark4', **472**, used: 472
higgs_squark_a, **470**, **513**, **547**, used: 470, 513, 547
higgs_squark_b, **470**, **513**, **547**, used: 470, 513, 547
higgs_SSS, **512**, **546**, used: 512, 546
higgs_triangle, **234**, **234**, **234**, **234**, **235**, **235**, **235**, **235**, **236**, **444**, **444**, **444**, **445**, **445**, **445**, **445**, **495**, **495**, **496**, **496**, **427**, **428**, **224**, **443**, **495**, **273**, used: 253, 463, 509, 430
higgs_triangle_vertices, **253**, **430**, used: 258, 436
Hit, **737**, **738**, **736**, used: 739
Hm, **447**, **498**, **411**, used: 447, 449, 457, 457, 458, 462, 463, 464, 467, 467, 467, 468, 469, 470, 471, 471, 472, 472, 473, 473, 476, 476, 479, 498, 500, 505, 505, 506, 508, 509, 509, 509, 509, 509, 510, 510, 510, 512, 512, 513, 514,

411, 412, 418, 420, 420, 421, 421
Hn (module), **219**, used: 219, 219, 220, 221
homogeneous, **745**, **740**, used: 745, 131, 133
homogeneous_final_state, **213**, used: 213, 215
Hp, **447**, **498**, **411**, used: 447, 449, 457, 457, 458, 462, 463, 464, 467, 467, 467, 468, 469, 470, 471, 471, 472, 472, 473, 473, 476, 476, 479, 498, 500, 505, 505, 506, 508, 509, 509, 509, 509, 509, 510, 510, 510, 512, 512, 513, 514, 411, 412, 418, 420, 420, 421, 421
HSExt (module), **384**, **272**, used: ??, ??
hs_of_flavor, **130**, used: 130, 131
hs_of_flavors, **130**, used: 131, 133
hs_of_lorentz, **130**, used: 130, 130
H_Heavy, **447**, used: 447, 449, 457, 458, 459, 462, 463, 464, 464, 465, 467, 467, 467, 468, 469, 469, 471, 471, 472, 472, 473, 476, 479
H_Light, **447**, used: 447, 449, 457, 458, 459, 462, 463, 464, 464, 465, 467, 467, 467, 468, 469, 469, 471, 471, 472, 472, 473, 476, 479
I, **564**, ??, **166**, **564**, ??, used: 564, ??, 244, ??, ??, 428, 566
I (camlyacc token), **567**, used: 567
id, **130**, used: 131
IG_s, **401**, used: 405
imag, **564**, **564**, used: ??, 567
IMap (module), **28**, used: 29, 29, 29, 29, 30, 30, 30
import, **75**, used: 75
import_prefixed, **703**, used: 703, 703
impossible, **803**, **10**, **184**, used: 803, 10, 10, 194
Impossible (exn), **803**, **10**, **23**, **43**, **49**, **95**, **96**, **123**, **126**, used:
in1 (field), **699**, used:
in2 (field), **699**, used:
include_anomalous, **??**, **??**, **??**, **??**, **??**, **??**, used:
include_ckm, **395**, **395**, **395**, **395**, **395**, **396**, **396**, **272**, used: 396, 401, 406
include_four, **444**, **444**, **444**, **445**, **445**, **445**, **445**, **443**, used: 478
include_goldstone, **444**, **444**, **444**, **445**, **445**, **445**, **445**, **443**, used: 447, 448, 478, 478
include_goldstones, **712**, used: 712, 715
include_hf, **395**, **395**, **395**, **395**, **395**, **396**, **396**, **272**, used: 396, 398, 407
incoming, **54**, **58**, **59**, **64**, **64**, **210**, **210**, **94**, **101**, **103**, **122**, **52**, **53**, **209**, **90**, used: 64, 122, 129, 129
incoming (field), **813**, **100**, **103**, **807**, used: 814, 815, 103, 116, 117, 121, 121
incomplete, **183**, **208**, used: 190, 191, 194, 203, 208, 208, 208, 208
Incomplete (exn), **692**, **694**, **691**, used:
Index, **566**, used: 566
Index′, **566**, used: 566
index_string, **753**, **751**, used: 727, 727
indices, **207**, used: 207
Infinity, **810**, used: 811
init, **575**, used: 833, 835, 818, 576, 110
initialize_cache, **94**, **105**, **127**, **134**, **90**, **92**, used: 134, 715

inject, **267**, used: 268, 268, 268, 269, 269, 271
injectl, **271**, used: 271, 271
inner (field), **813**, **807**, used: 814, 711, 711, 712
input (field), **171**, used: 575, 225, 227, 229, 232, 245, 455, ??, ??, 503, 534, 280, 302, 323, 337, 350, 368, 379, 389, 407, 416, 428, ??, ??, 721
input_functions, **574**, used: 574, 576
input_lagrangian, **574**, used: 574, 576
input_model, **574**, used:
input_parameters, **243**, ??, ??, **280**, **302**, **322**, **337**, **350**, **367**, **378**, **388**, **428**, ??, ??, used: 245, ??, ??, 280, 302, 323, 337, 350, 368, 379, 389, 428, ??, ??
input_particles, **574**, used: 574, 576
input_table, **570**, used: 574
input_variables, **574**, used: 574, 576
insert, **795**, used: 795, 795
insert1, **828**, **830**, **831**, used: 828, 830, 830, 830, 831
insert_inorder_signed, **795**, used: 796
insert_in_unfinished_decays, **696**, used: 696
insert_signed, **795**, used: 795, 795
inspect_partition, **20**, **21**, **24**, **26**, **26**, **33**, **33**, **19**, used:
Int (module), **27**, **124**, **19**, used: 124, 124
INT, ??, ??, ??, ??, used: ??, ??, 70, 566
INT (camlyacc token), **71**, **567**, used: 72, 567
integer, **564**, **564**, used: ??, 572, 575, 567
integer (type), **28**, **28**, **19**, used: 28, 28, 29, 20, 20, 20, 20
integer (camllex regexpr), **566**, used: 566
Integer, **564**, **564**, used: 564
Integer (sig), **27**, **19**, used: 27, 28, 19, 20
integer_range, **215**, used: 215
interleave, **745**, **741**, used: 746
interleave_nearest, **746**, **741**, used:
internal_edges, **816**, used: 817
internal_edges_from, **816**, used: 816, 816
int_edges (field), **816**, used: 817, 817, 819
int_incidence, **817**, used: 819
int_list_to_string, **70**, used:
int_node, **812**, used: 812
int_nodes (field), **816**, used: 817, 817, 819, 819, 819, 819
int_of_char, **446**, **497**, **528**, used: 448, 448, 450, 481, 499, 499, 500, 518, 528, 530, 530, 531, 554
int_of_csign, **397**, used: 399, 400, 401, 401, 401, 401
int_of_gen, **397**, used: 408
int_orders, **98**, used: 107
int_power, **86**, used: 86
invalid, **183**, used: 186, 186, 191
Invalid (exn), **730**, **730**, used:
invalid_flavor, **721**, used: 721
inverse, **796**, **797**, **798**, **796**, used: 797, 798, 800, 800
inverse_ok, **800**, used: 800
invert_array, **817**, used: 817
invert_array_unsafe, **816**, used:
InvPi (module), **781**, used: 781, 781, 781, 781, 781, 781
inv_pi, **781**, **781**, **782**, **782**, **780**, **780**, used: 782, 782, 782, 119, 119, 119
in_ghost_flags, **83**, **84**, **80**, used:
in_to_lists, **83**, **84**, **80**, used:
IPowSet (module), **78**, **215**, used: 78, 78, 215
ISet (module), **746**, **186**, **215**, used: 746, 746, 746, 186, 215, 215
iset_list_union, **746**, used: 746
iset_of_list, **746**, used: 746
isospin (type), **397**, used: 397, 401

Iso_down, **397**, used: 398, 402, 403
Iso_up, **397**, used: 398, 402
is_empty, **755**, **756**, **761**, **784**, **784**, **769**, **769**, **770**, **754**, **783**, **767**, **768**, used: 784, 785, 785, 786, 828, 829, 831, 46, 46, 78, 214
is_error, **??**, used: ??
is_failure, **??**, used: ??
is_gauss, **73**, **76**, **94**, **101**, **103**, **122**, **73**, **90**, used: 117, 122
is_gauss (field), **76**, **100**, **103**, used: 76, 76, 79, 103, 116, 117, 121
is_goldstone_of, **104**, used:
is_node, **41**, **45**, **37**, used: 111, 112, 112
is_null, **826**, **826**, **827**, **829**, **831**, **181**, **181**, **181**, **181**, **181**, **182**, **823**, **824**, **180**, used: 829, 830, 831, 831, 831, 832, 181, 182, 211, 122
is_offspring, **41**, **46**, **37**, used: 47
is_skip, **??**, used: ??
is_source, **104**, used: 108, 112, 112
is_sterile, **41**, **46**, **37**, used: 47, 47, 118
is_success, **??**, used:
is_todo, **??**, used: ??
is_unit, **826**, **826**, **827**, **827**, **828**, **829**, **823**, **824**, **824**, used: 829, 831, 832
is_white, **203**, used: 203
It (module), **183**, **183**, used: 207, 95, 97, 117, 128, 708, 91, 93
iter, **755**, **760**, **763**, **769**, **769**, **771**, **772**, **774**, **9**, **11**, **12**, **14**, **16**, **41**, **43**, **44**, **46**, **48**, **754**, **767**, **768**, **768**, **6**, **38**, used: ??, ??, ??, ??, ??, ??, ??, ??, 760, 763, 771, 772, 774, 774, 812, 814, 814, 814, 815, 815, 815, 815, 817, 817, 819, 819, 819, 16, 44, 46, 46, 48, 221, 702, 703, 703, 704, 705, 124, 133, 712, 716, 716, 716, 716, 716
iter', **771**, **774**, used: 771, 774
iteri, **744**, **741**, used: 744
iteri2, **744**, **741**, used:
iter_edges, **819**, **808**, used:
iter_incoming, **819**, **808**, used:
iter_internal, **819**, **808**, used:
iter_nodes, **41**, **46**, **37**, used: 124
iter_outgoing, **819**, **808**, used:
I_G1_AWW, **241**, ??, ??, **428**, used: 247, ??, ??, 428
I_G1_minus_kappa_minus_G4_AWW, **241**, ??, ??, **428**, used: 247, ??, ??, 428
I_G1_minus_kappa_minus_G4_ZWW, **241**, ??, ??, **428**, used: 247, ??, ??, 428
I_G1_minus_kappa_plus_G4_AWW, **241**, ??, ??, **428**, used: 247, ??, ??, 428
I_G1_minus_kappa_plus_G4_ZWW, **241**, ??, ??, **428**, used: 247, ??, ??, 428
I_G1_plus_kappa_minus_G4_AWW, **241**, ??, ??, **428**, used: 247, ??, ??, 428
I_G1_plus_kappa_minus_G4_ZWW, **241**, ??, ??, **428**, used: 247, ??, ??, 428
I_G1_plus_kappa_plus_G4_AWW, **241**, ??, ??, **428**, used: 247, ??, ??, 428
I_G1_plus_kappa_plus_G4_ZWW, **241**, ??, ??, **428**, used: 247, ??, ??, 428
I_G1_ZWW, **241**, ??, ??, **428**, used: 247, ??, ??, 428
I_G3, **719**, used: 720
I_Gs, **230**, **241**, ??, ??, **278**, **300**, **321**, **337**, **349**, **378**, **388**, **414**, **428**, ??, ??, used: 231, 246, 255, ??, ??, ??, ??, 283, 305, 325, 338, 351, 380, 390, 419, 428, 434, ??, ??
I_GsRt2, **349**, used: 351
I_G_Abb, **414**, used:
I_G_Acc, **414**, used:

I_G_AHWHW, **278**, **300**, used: 283, 305
I_G_AHWHWH, **278**, **300**, used: 283, 305
I_G_AHWW, **278**, **300**, used: 283, 305
I_G_Amumu, **414**, used:
I_G_Atautau, **414**, used:
I_G_Att, **414**, used:
I_G_CC, **321**, used: 323
I_G_Psi0ppWHW, **278**, **300**, used: 287, 308
I_G_Psi0ppWHWH, **278**, **300**, used: 287, 308
I_G_Psi0ppWW, **278**, **300**, used: 287, 308
I_G_Psi0pWA, **278**, **300**, used: 287, 308
I_G_Psi0pWAH, **278**, **300**, used: 287, 308
I_G_Psi0pWHA, **278**, **300**, used: 287, 308
I_G_Psi0pWHAH, **278**, **300**, used: 287, 308
I_G_Psi0pWHZ, **278**, **300**, used: 287, 308
I_G_Psi0pWHZH, **278**, **300**, used: 287, 308
I_G_Psi0pWZ, **278**, **300**, used: 287, 308
I_G_Psi0pWZH, **278**, **300**, used: 287, 308
I_G_S, **450**, **501**, **532**, used: 463, 508, 542
I_G_weak, **241**, **??**, **??**, **428**, used: 244, 255, 256, 256, 256, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, 428, 434, 434, 435, 435
I_G_WWW, **278**, **300**, **321**, **414**, **??**, used: 283, 305
I_G_Z1, **321**, used: 325
I_G_Z2, **321**, used: 325
I_G_Z3, **321**, used: 325
I_G_Z4, **321**, used: 325
I_G_Z5, **321**, used: 325
I_G_Z6, **321**, used: 325
I_G_ZHWHWH, **278**, **300**, used: 283, 305
I_G_ZHWW, **278**, **300**, used: 283, 305
I_G_ZWHW, **278**, **300**, used: 283, 305
I_G_ZWW, **241**, **450**, **??**, **??**, **501**, **532**, **278**, **300**, **321**, **337**, **349**, **367**, **378**, **388**, **414**, **428**, **??**, **??**, used: 244, 246, 253, 463, ??, ??, ??, ??, 508, 542, 283, 283, 290, 305, 305, 312, 325, 327, 338, 340, 351, 353, 368, 370, 380, 380, 390, 391, 419, 422, 428, 431, ??, ??, ??, ??
I_G_ZWW_K1, **349**, used: 351
I_G_ZWW_K2, **349**, used: 351
I_G_ZWW_K3, **349**, used: 351
I_kappa5_AWW, **241**, **??**, **??**, **428**, used: 247, ??, ??, 428
I_kappa5_ZWW, **241**, **??**, **??**, **428**, used: 247, ??, ??, 428
I_lambda5_AWW, **241**, **??**, **??**, **428**, used: 247, ??, ??, 428
I_lambda5_ZWW, **241**, **??**, **??**, **428**, used: 247, ??, ??, 428
I_lambda_AWW, **241**, **??**, **??**, **428**, used: 247, ??, ??, 428
I_lambda_ZWW, **241**, **??**, **??**, **428**, used: 247, ??, ??, 428
I_Q_H, **278**, **300**, **321**, used:
I_Q_W, **241**, **450**, **??**, **??**, **501**, **532**, **278**, **300**, **321**, **337**, **349**, **367**, **378**, **388**, **414**, **428**, **??**, **??**, used: 244, 246, 253, 463, ??, ??, ??, ??, 508, 542, 283, 283, 290, 305, 305, 312, 325, 325, 327, 338, 340, 351, 353, 368, 370, 380, 380, 390, 391, 419, 422, 428, 431, ??, ??, ??, ??
I_Q_W_K, **349**, used: 351
I_Q_ZH, **321**, used: 325

Java (module), **578**, used:
join, **756**, **735**, **735**, used: 756, 757, 757, 759, 760, 735
join_signs, **795**, used: 795
ket, **94**, **100**, **103**, **122**, **89**, used: 122
key (type), **737**, **737**, **769**, **769**, **770**, **43**, **44**, **48**, **736**, **767**, **768**, used: 737, 769, 769, 770, 770, 772, 43, 44, 44, 48, **??**, 736, 736, 767, 767, 767, 767, 768, 768, 768
Key (sig), **737**, **736**, used: 737, 736
keys, **44**, used: 44
keystone, **24**, used: 24
keystones, **794**, **20**, **24**, **26**, **26**, **26**, **28**, **32**, **33**, **34**, **788**, **18**, **20**, used: 794, 32, 117
keystones′, **24**, **33**, **34**, used: 26, 26, 26, 33, 34
kind, **177**, **177**, used:
kind (type), **177**, **177**, used: 177, 177
kk2 (type), **397**, used: 401
kkmode (type), **397**, used: 397, 397, 401
kmatrix_cuts, **107**, used: 108, 108, 119
k_matrix, **234**, **234**, **234**, **234**, **235**, **235**, **235**, **235**, **236**, **??**, **??**, **427**, **428**, **??**, **??**, **??**, **??**, **??**, **224**, **??**, **273**, **??**, used: 250, **??**, **??**, 428
K_Matrix_Coeff, **241**, **??**, **??**, **428**, used:
K_Matrix_Pole, **241**, **??**, **??**, **428**, used:
k_matrix_quartic_gauge, **250**, **??**, **??**, **428**, used: 251, **??**, **??**, 428
L, **236**, **447**, **??**, **??**, **498**, **529**, **274**, **297**, **319**, **333**, **344**, **363**, **374**, **384**, **411**, **428**, **??**, **??**, used: 237, 239, 243, 245, 246, 246, 246, 257, 257, 259, 447, 449, 453, 455, 455, 456, 457, 457, 459, 461, 462, 476, 476, 479, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, 498, 500, 503, 503, 504, 505, 505, 506, 507, 514, 529, 531, 534, 535, 535, 537, 537, 538, 538, 540, 540, 540, 540, 549, 275, 277, 281, 281, 281, 282, 282, 282, 282, 291, 297, 299, 302, 303, 303, 303, 304, 304, 312, 319, 320, 323, 323, 323, 324, 325, 325, 327, 333, 335, 338, 338, 338, 338, 338, 340, 344, 347, 350, 350, 350, 351, 351, 353, 363, 365, 368, 368, 368, 368, 369, 370, 375, 376, 379, 379, 379, 379, 381, 385, 386, 389, 389, 389, 390, 391, 411, 412, 417, 417, 417, 418, 419, 422, 428, 435, 436, 437, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**
label, **693**, used: 693, 693, 694, 695, 696, 696
label (field), **811**, **806**, used: 811, 812, 813, 813, 709
Label, **??**, **??**, used: **??**, **??**, **??**
lagrangian_file, **576**, used: 576, 576
Lambda, **501**, used:
last_begin (field), **732**, used: 733, 734, 734, 734
last_non_white, **568**, used: 569
last_reset (field), **732**, used: 733, 734, 734
LaTeX (module), **578**, used:
latex_section, **813**, used: 814, 815
layout, **819**, **807**, used:
layout (type), **818**, **807**, used: 807, 808
lcm, **826**, used:
leaf, **820**, **808**, **820**, **805**, used: 46, 47
Leaf, **820**, **808**, **693**, used: 820, 808, 810, 693, 693, 694, 694, 695

leafs, **808**, **806**, used: 808, 812, 816, 123
leafs_and_nodes, **816**, used: 817
leaf_label, **811**, **811**, used: 811, 812
leaf_node, **811**, used: 812, 812
leaf_or_int_node, **812**, used: 812
leaf_to_string, **810**, used: 810
left_to_right, **818**, **807**, used:
lepton, **240**, **454**, **??**, **??**, **501**, **532**, **278**, **300**, **321**, **336**, **349**, **367**, **378**, **388**, **397**, **414**, **428**, **??**, **??**, used: 241, 454, ??, ??, 501, 532, 278, 300, 321, 336, 349, 367, 378, 388, 398, 407, 414, 428, ??, ??
Lepton, **397**, used: 397, 400, 403, 403, 404
lep_family, **447**, used:
less, **54**, **58**, **64**, **51**, used: 694, 695
less′, **58**, used: 58, 58
lesseq, **54**, **58**, **63**, **51**, used: 64, 694, 695, 74
lesseq′, **58**, used: 58, 58
lhs, **94**, **100**, **103**, **121**, **89**, used: 103, 121, 122, 124, 131, 132
Light, **397**, used: 398, 398, 398, 405, 405, 407
Light2, **397**, used: 398, 398, 402
Light_Heavy, **397**, used: 398, 402
Linalg (module), **833**, **833**, used: 818, 819
line (type), **81**, **82**, used: 81, 82, 82
Line (sig), **81**, used: 82
linear, **828**, **831**, **825**, used: 831
Linear (sig), **828**, **825**, used: 831, 825
Lines, **83**, **86**, used: 83, 84
list, **796**, **797**, **798**, **731**, **777**, **796**, **730**, **776**, used:
List (module), **761**, **755**, used: ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, 765, 765, 765, 742, 742, 742, 743, 743, 743, 744, 744, 744, 744, 745, 745, 746, 746, 746, 746, 746, 746, 746, 747, 747, 749, 797, 797, 797, 800, 781, 784, 784, 784, 785, 785, 785, 785, 728, 735, 738, 836, 820, 820, 820, 826, 829, 829, 830, 830, 830, 830, 830, 831, 831, 832, 730, 731, 731, 777, 777, 777, 777, 777, 777, 777, 790, 790, 791, 791, 791, 792, 792, 792, 792, 792, 793, 793, 793, 793, 793, 793, 794, 794, 794, 794, 794, 794, 795, 795, 795, 795, 795, 795, 795, 795, 796, 802, 803, 803, 808, 809, 809, 810, 810, 810, 811, 812, 812, 814, 814, 814, 815, 815, 815, 815, 816, 817, 817, 817, 818, 818, 819, 819, 10, 10, 11, 11, 12, 13, 15, 16, 16, 16, 16, 16, 16, 17, 17, 17, 21, 24, 24, 24, 26, 26, 26, 29, 29, 29, 31, 31, 32, 32, 32, 32, 32, 33, 33, 33, 33, 34, 34, 47, 47, 48, 55, 55, 56, 57, 60, 61, 62, 62, 692, 692, 694, 694, 695, 696, 696, 697, 697, 181, 181, 181, 182, 82, 82, 82, 83, 84, 84, 84, 84, 84, 84, 84, 84, 84, 85, 85, 86, 87, 220, 221, 221, 699, 699, 700, 700, 700, 700, 700, 700, 700, 701, 701, 701, 701, 702, 702, 703, 703, 703, 704, 705, 565, 565, 565, 569, 569, 570, 570, 572, 574, 574, 575, 576, 69, 70, 74, 75, 75, 75, 75, 75, 76, 76, 77, 78, 78, 78, 78, 186, 189, 191, 203, 204, 204, 205, 205, 205, 206, 206, 207, 207, 211, 211, 211, 212, 213, 214, 214, 214, 214, 214, 214, 215, 215, 215, 216, 216, 216, 96, 101, 103, 103, 104, 108,

110, 110, 110, 111, 111, 111, 112, 112, 116, 116, 117, 117, 118, 119, 119, 119, 120, 120, 120, 121, 122, 122, 122, 122, 122, 123, 123, 123, 123, 124, 124, 124, 124, 129, 129, 129, 129, 130, 130, 131, 131, 131, 132, 132, 133, 133, 133, 133, 708, 709, 709, 709, 710, 710, 710, 710, 711, 711, 711, 712, 712, 712, 712, 712, 712, 712, 713, 714, 714, 715, 715, 716, 716, 716, 716, 716, 717, 237, 237, 237, 237, 245, 245, 246, 246, 246, 246, 246, 247, 248, 249, 250, 256, 257, 257, 258, 258, 269, 269, 269, 269, 270, 271, 456, 456, 459, 461, 461, 463, 465, 466, 466, 466, 469, 470, 470, 470, 471, 472, 473, 473, 474, 474, 475, 475, 476, 477, 478, 478, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, 496, 496, 497, 497, 504, 505, 505, 506, 507, 507, 507, 509, 510, 511, 511, 511, 512, 512, 512, 513, 513, 513, 527, 527, 527, 527, 529, 536, 536, 537, 538, 538, 538, 539, 541, 541, 541, 541, 543, 544, 545, 545, 545, 546, 546, 547, 547, 547, 548, 275, 275, 281, 281, 281, 281, 281, 282, 282, 282, 282, 283, 283, 283, 284, 284, 285, 285, 285, 286, 287, 287, 290, 290, 333, 333, 338, 338, 338, 338, 338, 338, 338, 338, 344, 344, 350, 350, 350, 351, 351, 351, 351, 351, 363, 363, 368, 368, 368, 368, 368, 369, 375, 375, 379, 379, 379, 379, 380, 380, 385, 385, 389, 389, 389, 389, 390, 390, 398, 402, 403, 404, 406, 407, 407, 408, 411, 411, 416, 417, 417, 417, 417, 419, 420, 428, 435, 435, 436, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, 718, 718, 718, 719, 720, 720, 720, 720, 721, 721, 723

list2, **777**, **776**, used: 11, 15, 123, 131, 246, 456, 456, 459, 461, 461, 463, 465, 466, 466, 466, 469, 470, 470, 470, 471, 472, 473, 473, 474, 474, 474, 474, 475, 475, 475, 475, 476, 477, 478, 478, **??**, **??**, 504, 505, 505, 507, 507, 507, 509, 511, 511, 511, 512, 512, 513, 513, 536, 536, 537, 538, 538, 539, 541, 541, 541, 543, 545, 545, 545, 546, 547, 547, 548, 428

list3, **777**, **776**, used: 12, 15, 506, 507, 513, 537, 538, 541, 547, 548

ListItem, **??**, **??**, used: **??**, **??**, **??**

lists, **41**, **47**, **38**, used: 117, 120

Lists (module), **55**, **53**, used: 66, 53

ListSimpleMake (module), **??**, **??**, used:

ListsW (module), **66**, **53**, used:

Littlest (module), **274**, **272**, used: **??**, **??**

littlest_gauge_higgs4, **287**, **308**, used:

Littlest_Tpar (module), **297**, **272**, used: **??**

LMOM, **137**, used:

load, **576**, used: 576

Lodd, **297**, used: 297

longest, **769**, **771**, **772**, **774**, **767**, **769**, used:

longest′, **771**, **774**, used: 771, 774

longest_first, **793**, used: 794

lookup_ketset, **119**, used: 120

loop_cs, **398**, used: 398, 407

loop_gen, **398**, used: 398, 402, 403, 403, 403, 403, 404, 404, 404, 404, 407
loop_iso, **398**, used: 398, 402, 403, 403, 403, 404, 404, 404, 404, 407
loop_kk, **398**, used: 398, 398, 402, 403, 403, 404, 404, 404, 404, 405, 405, 405, 405
loop_kk2, **398**, used: 398, 402, 405, 405
Loop_Tags (module), **97**, used:
lorentz, **222**, **571**, **184**, **207**, **225**, **226**, **228**, **230**, **238**, **266**, **267**, **448**, **??**, **??**, **499**, **530**, **275**, **298**, **319**, **334**, **345**, **364**, **375**, **385**, **399**, **411**, **428**, **??**, **??**, **719**, **172**, used: 571, 184, 207, 102, 126, 130, 131, 266, 267
lorentz (label), **223**, **175**, used: 575
lorentz (type), **135**, used: 571, 572, 135, 172, 175
lorentz_aux, **237**, **??**, **??**, **428**, used: 238, **??**, **??**, 428
lorentz_ordering, **102**, used: 102
lower (field), **813**, **807**, used: 815
lower (camllex regexpr), **70**, **566**, used: 70, 566
LPAREN, **??**, **??**, **??**, **??**, used: **??**, **??**, 70, 566
LPAREN (camlyacc token), **71**, **567**, used: 71, 567
LQ, **529**, used: 529, 531, 540, 540, 540, 540, 540, 541, 541, 541, 541, 541, 541, 542, 542, 542, 542, 542, 542, 549
LQino, **529**, used: 529, 531, 539, 539, 540, 540, 540, 540, 540, 540, 541, 541, 541, 541, 541, 542, 542, 542, 549
lqino_lq_gg, **541**, used: 548
lqino_lq_gg′, **541**, used: 541
lqino_lq_neu, **541**, used: 548
lqino_lq_neu′, **541**, used: 541
lqino_lq_neu2, **541**, used: 548
lqino_lq_neu2′, **541**, used: 541
lq_gauge4, **542**, used: 549
lq_gauge4′, **542**, used: 542
lq_gg_gauge2, **542**, used: 549
lq_gg_gauge2′, **542**, used: 542
lq_neutr_Z, **542**, used: 548
lq_phiggs, **541**, used: 548
lq_phiggs′, **541**, used: 541
lq_se_su, **541**, used: 548
lq_se_su′, **540**, used: 541
lq_shiggs, **541**, used: 548
lq_shiggs′, **541**, used: 541
lq_snu_sd, **541**, used: 548
lq_snu_sd′, **541**, used: 541
lu_backsubstitute, **836**, used: 836, 836
lu_decompose, **836**, **833**, used:
lu_decompose_in_place, **835**, used: 836, 836, 836
lu_decompose_split, **835**, used: 836
L_CN, **450**, used:
L_CNG, **450**, used:
L_K1_L, **344**, used: 344, 347, 353
L_K1_R, **344**, used: 344, 347, 353
L_K2_L, **344**, used: 344, 347, 353
L_K2_R, **344**, used: 344, 347, 353
L_NC, **450**, used:
L_NCH, **450**, used:
M, **236**, **267**, **450**, **??**, **??**, **274**, **333**, **344**, **363**, **374**, **384**, **411**, **428**, **??**, **??**, used: 237, 239, 243, 245, 246, 253, 254, 254, 254, 254, 254, 254, 255, 255, 255, 255, 256, 256, 256, 256, 257, 257, 257, 257, 258, 259, **??**, 274, 277, 281, 290, 291, 333, 335, 337, 340, 344, 347, 350, 353,

363, 365, 368, 369, 370, 375, 376, 379, 379, 381, 385, 386, 389, 390, 391, 411, 412, 417, 417, 418, 418, 418, 418, 418, 418, 419, 422, 428, 432, 433, 433, 433, 433, 433, 434, 434, 434, 434, 435, 435, 435, 435, 435, 436, 437, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??
M (module), **765**, **828**, **829**, **831**, **816**, **48**, **571**, used: 765, 765, 765, 828, 828, 828, 828, 828, 828, 829, 829, 829, 829, 829, 829, 829, 830, 830, 830, 830, 830, 830, 830, 830, 831, 831, 831, 831, 831, 831, 831, 832, 816, 816, 817, 817, 48, 571, 575, 576
M1, **446**, **497**, **528**, used: 447, 456, 456, 459, 461, 461, 463, 465, 466, 466, 466, 469, 469, 470, 470, 470, 471, 471, 472, 472, 473, 473, 474, 474, 474, 474, 475, 475, 475, 475, 476, 477, 478, 479, 498, 504, 505, 506, 507, 507, 507, 508, 511, 511, 511, 512, 512, 512, 513, 513, 513, 514, 529, 536, 536, 538, 538, 538, 539, 539, 539, 540, 540, 540, 540, 540, 540, 540, 541, 541, 541, 541, 541, 541, 541, 542, 542, 542, 545, 545, 545, 546, 546, 547, 547, 547, 548, 549
M1 (module), **44**, used: 44, 44, 44, 44
M2, **446**, **497**, **528**, used: 447, 456, 456, 459, 461, 461, 463, 465, 466, 466, 466, 469, 469, 470, 470, 470, 471, 471, 472, 472, 473, 473, 474, 474, 474, 474, 475, 475, 475, 475, 476, 477, 478, 479, 498, 504, 505, 507, 507, 507, 508, 511, 511, 511, 512, 512, 512, 513, 513, 513, 514, 529, 536, 536, 538, 538, 539, 539, 539, 540, 540, 540, 540, 540, 540, 540, 540, 540, 540, 541, 541, 541, 541, 541, 541, 541, 542, 542, 542, 545, 545, 545, 546, 546, 547, 547, 547, 548, 549
M2 (module), **44**, used: 44, 44, 44
main, ??, **707**, **713**, ??, **707**, used: 75, 718, ??, ??, ??, 718, ??, 723
main (camlyacc non-terminal), **71**, used: 71
Majorana, **125**, **135**, used: 125, 126, 448, 499, 530, 364, 719
Maj_Ghost, **135**, used:
make, **826**, **826**, **823**, used: ??, ??, 749, 798, 798, 834, 835, 826, 830, 830, 574, 124, 709, 240, 453, ??, ??, 500, 531, 277, 300, 321, 335, 348, 366, 377, 387, 400, 413, 428, ??, ??
Make (module), **765**, **782**, **784**, **737**, **770**, **49**, **693**, **699**, **74**, **210**, **125**, **708**, **765**, **780**, **784**, **736**, **768**, **39**, **691**, **698**, **73**, **210**, **707**, used: ??, ??, 765, 765, 746, 781, 784, 784, 772, 730, 28, 44, 44, 44, 45, 48, 219, 220, 221, 699, 700, 702, 78, 186, 214, 215, 215, 215, 216, 95, 103, 103, 104, 105, 111, 112, 116, 118, 119, 119, 123, 124, 125, 127, 127, 127, 127, 128, 129, 129, 131,

131, 708, 708, 718, ??, ??, ??, 718, ??, 723, 91

MakeMap (module), **772**, **768**, used:

MakePoly (module), **773**, **769**, used:

Maker (sig), **95**, **179**, **91**, used: 699, 128, 128, 708, 179, 698, 91, 91, 91, 91, 93, 707, 578, 578, 578

make_external_dag, **111**, used: 111

Make_Linear (module), **831**, **825**, used:

Make_Ring (module), **829**, **825**, used:

map, **755**, **760**, **763**, **769**, **769**, **771**, **772**, **774**, **827**, **828**, **829**, **831**, **810**, **9**, **11**, **12**, **14**, **16**, **41**, **46**, **692**, **695**, **754**, **767**, **768**, **768**, **824**, **825**, **806**, **6**, **38**, **691**, used: ??, 765, 760, 763, 744, 745, 746, 746, 747, 747, 749, 797, 800, 784, 784, 785, 728, 771, 774, 836, 820, 820, 828, 829, 830, 831, 731, 731, 777, 777, 792, 793, 793, 793, 793, 793, 794, 794, 794, 794, 795, 795, 795, 795, 803, 809, 809, 810, 810, 810, 811, 812, 816, 817, 818, 819, 10, 10, 16, 17, 17, 21, 24, 26, 26, 26, 32, 33, 33, 33, 34, 47, 55, 60, 62, 695, 696, 697, 697, 697, 83, 84, 84, 84, 84, 84, 84, 84, 84, 86, 699, 700, 700, 701, 701, 701, 701, 703, 565, 570, 575, 576, 69, 70, 74, 75, 75, 75, 75, 75, 78, 78, 78, 186, 189, 191, 203, 204, 206, 206, 207, 211, 213, 214, 214, 214, 215, 215, 216, 216, 101, 103, 108, 108, 110, 112, 112, 112, 116, 116, 117, 119, 119, 119, 120, 122, 122, 123, 123, 123, 123, 124, 124, 129, 129, 129, 130, 131, 131, 133, 133, 708, 709, 709, 710, 710, 710, 710, 711, 711, 711, 712, 712, 712, 712, 712, 713, 714, 714, 715, 716, 717, 237, 237, 237, 245, 245, 246, 246, 246, 246, 246, 247, 248, 249, 250, 256, 257, 257, 258, 258, 269, 269, 269, 269, 270, 271, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, 496, 497, 506, 509, 510, 512, 527, 527, 529, 538, 543, 544, 546, 275, 275, 281, 281, 281, 281, 281, 282, 282, 282, 282, 283, 283, 283, 284, 284, 285, 285, 285, 286, 287, 287, 290, 290, 333, 333, 338, 338, 338, 338, 338, 338, 338, 338, 344, 344, 350, 350, 350, 351, 351, 351, 351, 351, 363, 363, 368, 368, 368, 368, 368, 369, 375, 375, 379, 379, 379, 379, 380, 380, 385, 385, 389, 389, 389, 389, 390, 390, 398, 402, 406, 407, 407, 408, 411, 411, 416, 417, 417, 417, 417, 419, 420, 428, 435, 435, 436, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, 718, 718, 718, 719, 720, 720, 720, 720, 721, 721, 723

map2, **9**, **11**, **12**, **14**, **16**, **6**, used: 747, 777, 818, 16, 181, 182, 700, 700, 214, 108

mapi, ??, **755**, **760**, **764**, **744**, **769**, **769**, **771**, **772**, **774**, **754**, **741**, **767**, **768**, **768**,

used: ??, 760, 764, 771, 774, 819, 131
mapi′, **771**, **774**, used: 771, 774
mappable, **702**, used: 702
Mappable (module), **702**, used: 702, 702
map_amplitude_wfs, **116**, used:
map_decay, **692**, **693**, **691**, used: 693, 694, 695
map_nodes, **41**, **46**, **37**, used:
Map_S (sig), **769**, **768**, used: 770, 772, 768
mask, **61**, used: 61, 62, 65, 66, 66
mask2, **61**, used:
mask21, **61**, used: 61
mask5, **61**, used: 61
maskb, **61**, used: 61
maskd, **61**, used: 61
maskr, **61**, used: 61
mask_in, **64**, used: 64, 64, 64, 64, 64
mask_in1, **64**, used: 64, 64, 64, 64
mask_in2, **64**, used: 64
Mass, **241**, ??, ??, **278**, **300**, **321**, **337**, **349**, **367**, **378**, **388**, **414**, **428**, ??, ??, used: 243, 244, ??, ??, 428
massive, **177**, **178**, used:
massive (type), **177**, **177**, used: 177, 177, 177, 178
Massive, **177**, **177**, used:
Massive_Vector, **135**, used: 238, 448, ??, ??, 499, 530, 275, 298, 319, 334, 345, 364, 375, 385, 399, 411, 428, ??, ??
massless, **177**, **178**, used:
massless (type), **177**, **177**, used: 177, 177
Massless, **177**, **177**, used:
mass_symbol, **222**, **571**, **188**, **207**, **226**, **227**, **229**, **233**, **263**, **267**, **267**, **487**, ??, ??, **522**, **560**, **294**, **316**, **331**, **343**, **362**, **373**, **383**, **394**, **409**, **425**, **441**, ??, ??, **723**, **175**, used: 571, 188, 207, 267, 267
mass_symbol (label), **223**, **175**, used: 575
matching, **55**, used:
matmul, **833**, **833**, used:
matmulv, **834**, **833**, used: 818
Matter, **208**, **237**, **267**, ??, ??, **274**, **333**, **344**, **363**, **375**, **385**, **411**, **428**, ??, ??, **177**, used: 237, 267, ??, ??, 274, 333, 344, 363, 375, 385, 411, 428, ??, ??
matter_field, **208**, **237**, **267**, ??, ??, **274**, **333**, **344**, **363**, **375**, **385**, **411**, **428**, ??, ??, **177**, used: 237, 267, 271, 271, ??, ??, 275, 275, 333, 344, 363, 375, 385, 411, 428, ??, ??
matter_field (type), **208**, **236**, **267**, ??, ??, **274**, **333**, **344**, **363**, **374**, **384**, **411**, **428**, ??, ??, **176**, used: 208, 208, 236, 237, 267, ??, ??, 274, 274, 333, 333, 344, 344, 363, 363, 374, 375, 384, 385, 411, 411, 428, ??, ??, ??, ??, 177, 177
maxabsval, **834**, used: 835
max_arity, **9**, **10**, **12**, **13**, **15**, **16**, **17**, **26**, **32**, **6**, **8**, used: 16, 16, 17, 17, 24, 33
max_degree, **222**, **571**, **186**, **204**, **207**, **225**, **227**, **229**, **232**, **259**, **267**, **268**, **479**, ??, ??, **514**, **549**, **291**, **312**, **327**, **340**, **353**, **370**, **381**, **391**, **402**, **422**, **437**, ??, ??, **721**, **174**, used: 571, 186, 204, 207, 105, 117, 268
max_degree (label), **223**, **175**, used:
max_generations, **267**, used: 267, 269
max_subtree, **20**, **24**, **26**, **26**, **26**, **33**, **34**, **18**, used: 26, 26, 111
maybe_backtrace, ??, used: ??, ??

Maybe_Graded (module), **45**, used: 48, 49
maybe_read, **737**, **739**, **736**, used: 105
mem, **755**, **758**, **763**, **769**, **769**, **771**, **772**, **773**, **43**, **44**, **48**, **754**, **767**, **768**, **768**, used: ??, 758, 763, 786, 771, 773, 816, 44, 45, 45, 46, 48, 220, 221, 702, 76, 77, 78, 123, 132, 710, 710
merge, **757**, **730**, **698**, **701**, **730**, **698**, used: 757, 758, 758, 758, 758, 716
merge_overlapping, **214**, used: 214, 215
merge_partitions, **215**, used: 215
merge_sets, **700**, used: 700
mgm, **245**, ??, ??, **281**, **337**, **350**, **368**, **379**, **389**, **417**, **428**, ??, ??, used: 245, 245, 246, 246, 246, 246, ??, ??, 281, 281, 281, 281, 281, 282, 282, 338, 338, 338, 338, 350, 350, 350, 351, 368, 368, 368, 379, 379, 379, 379, 389, 389, 389, 389, 417, 417, 417, 417, 428, ??, ??, ??, ??, ??, ??, ??, ??, ??
mhm, **281**, used: 282, 282, 283
min, **28**, used: 743, 568
minimal_fusion_tower, **111**, used: 116
MINUS, ??, ??, used: ??, 566
MINUS (camlyacc token), **567**, used: 567, 567
minus_gauge4, **248**, **463**, ??, ??, **508**, **542**, **284**, **305**, **326**, **338**, **351**, **369**, **380**, **390**, **405**, **420**, **428**, ??, ??, used: 248, 463, ??, ??, 508, 542, 284, 284, 305, 305, 326, 338, 351, 369, 380, 390, 405, 405, 405, 420, 428, ??, ??
min_max_rank, **43**, **44**, **45**, **48**, **49**, **40**, used: 45, 110
mismatch, **55**, **61**, used: 55, 56, 56, 57, 57, 57, 58, 58, 60, 62, 62, 62, 63, 63, 63, 64
Mismatch, **84**, used: 738, 55, 61, 85, 85
Mismatch (exn), **737**, **738**, **54**, **55**, **61**, **127**, **128**, **736**, **51**, **91**, used:
Mismatched_arity (exn), **10**, **11**, **12**, **14**, **16**, **8**, used:
Miss, **737**, **738**, **736**, used: 739
Mixed23 (module), **13**, **26**, **127**, **8**, **19**, **91**, used: 26, 127, ??, ??, 718, ??, 723, 19
Mixed23 (sig), **13**, **8**, used: 8
Mixed23_Majorana (module), **127**, **91**, used: ??, ??, ??, ??, ??, ??, ??, ??, ??
mixed_fold_left, **111**, used: 111
mk_and, **69**, **68**, used: ??, 75, 71
mk_any_flavor, **69**, **68**, used: ??, 71
mk_false, **69**, **68**, used:
mk_gauss, **69**, **68**, used: ??, 71
mk_gauss_not, **69**, **68**, used: ??, 71
mk_off_shell, **69**, **68**, used: ??, 71
mk_off_shell_not, **69**, **68**, used: ??, 71
mk_on_shell, **69**, **68**, used: ??, 71
mk_on_shell_not, **69**, **68**, used: ??, 71
mk_or, **69**, **68**, used: ??, 71
mk_true, **69**, **68**, used: ??, 71
Model (module), **571**, **172**, **568**, used: 699, 74, 183, 207, 210, 95, 95, 95, 97, 98, 101, 125, 128, 128, 708, 267, 179, 217, 698, 568, 73, 183, 183, 210, 91, 93, 93, 707, 224, 224, 224, 224, 444, ??, 495, 525, 272, 272, 273, ??
Modellib_BSM (module), **273**, **272**, used: ??, ??, ??, ??,

??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??
Modellib_MSSM (module), **444**, **443**, used: ??, ??, ??, ??
Modellib_NMSSM (module), **495**, **495**, used: ??, ??, ??
Modellib_NoH (module), ??, ??, used: ??, ??
Modellib_PSSSM (module), **526**, **525**, used: ??
Modellib_SM (module), **224**, **224**, used: 718, ??, ??, ??, 718, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??
Modellib_Zprime (module), ??, ??, used: ??, ??
Modeltools (module), **218**, **217**, used: 571, 575, 225, 226, 228, 231, 245, 266, 269, 455, ??, ??, 503, 534, 280, 302, 323, 337, 350, 368, 379, 389, 406, 416, 428, ??, ??, 720
modname, **396**, used: 398, 400, 402, 408
mod_float2, **732**, used: 733
mom, **245**, ??, ??, **337**, **350**, **368**, **428**, ??, used: 256, 257, 257, 258, 258, ??, ??, ??, ??, ??, ??, ??, ??, 338, 338, 351, 351, 368, 435, 435, 436
MOM, **137**, used:
MOM5, **137**, used:
Momenta, **178**, used: 713
momentum, **94**, **100**, **101**, **121**, **88**, used: 699, 108, 121, 717
momentum (field), **100**, **101**, used: 101, 101, 102, 104, 108, 108, 109, 112, 117, 117, 117, 118, 119, 123, 123, 124, 124, 124
momentum (type), **692**, **693**, **691**, used: 692, 693, 694, 695, 691, 691, 691
momentum (camlyacc non-terminal), **72**, used: 72
Momentum, **566**, used: 566
Momentum (module), **54**, **50**, used: 693, 699, 74, 95, 97, 98, 101, 128, 128, 707, 707, 717, 179, 691, 698, 73, 91, 93, 93, 707
momentum_list, **94**, **100**, **101**, **121**, **88**, used: 121, 709, 709
momentum_list (camlyacc non-terminal), **72**, used: 71, 72
momentum_to_string, **75**, used: 75
momentum_to_TeX, **712**, used: 712, 712, 712, 712
MOML, **137**, used:
MOMR, **137**, used:
Mono (sig), **9**, **6**, used: 10, 12, 8, 8
MSSM (module), **445**, **444**, used: ??, ??, ??, ??
MSSM_flags (sig), **444**, **443**, used: 444, 444, 445, 445, 445, 445, 445, 444
MSSM_goldstone (module), **444**, **444**, used:
MSSM_Grav (module), **445**, **444**, used: ??
MSSM_Hgg (module), **445**, **444**, used: ??
MSSM_no_4 (module), **445**, **444**, used: ??
MSSM_no_4_ckm (module), **445**, **444**, used: ??
MSSM_no_goldstone (module), **444**, **444**, used:
Mu, **450**, **501**, used:
mul, **826**, **826**, **827**, **827**, **828**, **830**, **823**, **824**, **824**, used: 829, 830, 830, 830, 830, 830, 831
mul1, **828**, used: 828, 829, 829
MULT, ??, ??, used: ??, 566
MULT (camlyacc token), **567**, used: 567, 567
Multi (module), **128**, **93**, used: 708, 179
Multi (sig), **127**, **91**, used: 128, 93

multinomial, **790**, **787**, used:
multiplicity, **29**, **127**, **129**, **92**, used: 30, 30, 30
multiplicity (field), **129**, used: 129, 134
multiply, **746**, **565**, **741**, **564**, used: 565, ??, 567
multi_choose, **794**, **789**, used:
Multi_Maker (sig), **128**, **93**, used: 93
multi_split, **793**, **788**, used: 793, 794
multi_split′, **792**, used: 792, 793, 793
mult_vertex, **190**, used: 192, 193, 194, 196, 196, 197, 198, 199, 199, 200, 200, 202
mult_vertex3, **188**, used: 190
mult_vertex4, **189**, used: 190
mult_vertexn, **190**, used: 190
Muon, **227**, used: 228, 228, 229, 229
Mutable (module), **222**, **217**, used: 571
Mutable (sig), **175**, used: 217
M_N, **450**, used:
M_SF, **450**, used:
M_U, **450**, used:
M_V, **450**, used:
n (field), **695**, used:
N, **236**, **447**, **??**, **??**, **498**, **529**, **274**, **297**, **319**, **333**, **344**, **363**, **374**, **384**, **411**, **428**, **??**, **??**, used: 237, 239, 243, 246, 246, 257, 257, 259, 447, 449, 453, 455, 456, 457, 457, 459, 461, 462, 476, 476, 479, ??, ??, ??, ??, ??, ??, ??, ??, 498, 500, 503, 504, 505, 506, 507, 514, 529, 531, 535, 535, 537, 538, 538, 540, 540, 540, 540, 549, 275, 277, 281, 281, 282, 282, 282, 291, 297, 299, 303, 303, 303, 304, 312, 319, 320, 323, 323, 324, 325, 327, 333, 335, 338, 338, 338, 340, 344, 347, 350, 350, 351, 353, 363, 365, 368, 368, 370, 375, 376, 379, 379, 381, 385, 386, 389, 389, 391, 411, 412, 417, 417, 422, 428, 435, 436, 437, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??
N1, **446**, **497**, **528**, used: 447, 459, 459, 459, 462, 463, 476, 477, 478, 479, 498, 506, 507, 513, 514, 529, 538, 538, 548, 549
N10, **528**, used: 529, 549
N11, **528**, used: 529, 549
N2, **446**, **497**, **528**, used: 447, 459, 459, 459, 462, 463, 476, 477, 478, 479, 498, 506, 507, 513, 514, 529, 538, 538, 548, 549
N3, **446**, **497**, **528**, used: 447, 459, 459, 459, 462, 463, 476, 477, 478, 479, 498, 506, 507, 513, 514, 529, 538, 538, 548, 549
N4, **446**, **497**, **528**, used: 447, 459, 459, 459, 462, 463, 476, 477, 478, 479, 498, 506, 507, 513, 514, 529, 538, 538, 548, 549
N5, **497**, **528**, used: 498, 506, 507, 513, 514, 529, 538, 538, 548, 549
N6, **528**, used: 529, 549
N7, **528**, used: 529, 549
N8, **528**, used: 529, 549
N9, **528**, used: 529, 549
name, **727**, **726**, used: 728, 105, 105, 267
name (field), **727**, used: 727
Nary (module), **15**, **24**, **127**, **8**, **19**, **91**, used: 17, 24, 26, 32, 33, 127, 127, 19, 20
Nary (sig), **15**, **8**, used: 8, 8
Nary4 (module), **26**, used: 26, 26
Nary_Majorana (module), **127**, **91**, used:
nc, **186**, **207**, **718**, **178**, **178**, used: 192, 193, 196, 207, 719

nc_coupling, **245**, ??, ??, **280**, **302**, **322**, **337**, **350**, **367**, **378**, **389**, **416**, **428**, ??, used: 245, ??, ??, 280, 302, 322, 337, 350, 368, 379, 389, 416, 428, ??
nc_value, **186**, used: 186
neg, **826**, **826**, **827**, **830**, **830**, **54**, **56**, **62**, **565**, **823**, **824**, **51**, **564**, used: 830, 831, 832, 56, 57, 57, 59, 62, 62, 63, 64, 696, ??, 74, 76, 77, 78, 123, 567
Neg, **397**, **166**, used: 244, 245, 245, ??, ??, 280, 280, 302, 302, 322, 322, 337, 337, 350, 350, 367, 368, 378, 379, 389, 389, 398, 400, 403, 403, 403, 403, 404, 404, 404, 404, 404, 405, 405, 405, 405, 405, 407, 408, 416, 416, 428, ??, ??
neg′, **56**, used: 56, 56
negate, **565**, used: 565, 565, 565
Negative (exn), **54**, **57**, **62**, **51**, used:
Negative_EW_order (exn), **98**, used:
Negative_QCD_EW_orders (exn), **98**, used:
Negative_QCD_order (exn), **98**, used:
neu (type), **446**, **497**, **528**, used: 447, 450, 498, 501, 529, 532
Neutralino, **447**, **498**, **529**, used: 447, 449, 456, 457, 458, 458, 459, 459, 459, 462, 462, 476, 479, 498, 500, 504, 505, 506, 506, 506, 514, 529, 531, 535, 536, 537, 537, 538, 541, 541, 549
neutral_currents, **246**, **455**, ??, ??, **503**, **535**, **281**, **303**, **323**, **338**, **350**, **368**, **379**, **389**, **417**, **428**, ??, ??, used: 258, 477, ??, ??, 513, 548, 291, 312, 327, 340, 353, 370, 380, 391, 422, 436, ??, ??
neutral_currents′, ??, used: ??
neutral_heavy_currents, **281**, **303**, **324**, ??, used: 291, 312, 327, ??
neutral_sfermion_currents, **456**, **505**, **536**, used: 477, 513, 548
neutral_sfermion_currents′, **456**, **505**, **536**, used: 456, 505, 536
neutral_up_type_currents, ??, used: ??
neutral_Z, **505**, **536**, used: 513, 548
neutral_Z_1, **457**, used: 477
neutral_Z_2, **457**, used: 477
neutr_lqino_current, **542**, used: 548
NH, **319**, used: 319, 320, 323, 324, 327
nl, **678**, used: 678
nlist, **529**, used: 529, 548
NMSSM (module), **495**, **495**, used: ??
NMSSM_CKM (module), **496**, **495**, used: ??
NMSSM_flags (sig), **495**, **495**, used: 495, 496, 496, 496, 495
NMSSM_func (module), **496**, **495**, used: ??, ??, ??
NMSSM_Hgg (module), **496**, **495**, used: ??
Nodd, **297**, used: 297
node, **808**, **806**, used:
node (field), **816**, used: 816
node (type), ??, **41**, **41**, **42**, **42**, **43**, **45**, **109**, ??, **36**, **37**, **40**, used: ??, 41, 41, 42, 42, 42, 45, 48, 100, ??, ??, 36, 36, 36, 37, 37, 37, 37, 37, 37, 37, 38, 38, 38, 38, 38, 39, 39, 39, 39, 40, 40, 40, 40
Node, **755**, **820**, **808**, used: 756, 756, 756, 758, 759, 760, 760, 761, 820, 808, 809, 809, 809, 809, 810, 811

nodes, **808**, **806**, used: 808, 816
Nodes (module), **41**, **42**, **42**, **43**, **109**, **36**, **40**, used: 41, 42, 42, 43, 45, 45, 45, 45, 47, 48, 109, 36, 40, 40, 40
node_to_string, **810**, used: 810
node_with_tension (type), **816**, used:
NoH (module), **??**, **??**, used: **??**
NoH_flags (sig), **??**, **??**, used: **??**, **??**, **??**, **??**, **??**
NoH_k_matrix (module), **??**, **??**, used: **??**, **??**
NOT, **??**, **??**, used: **??**, 70
NOT (camlyacc token), **71**, used: 71, 71
Not_invertible (exn), **816**, used:
Not_Square (exn), **833**, **833**, used:
no_cascades, **73**, **76**, **72**, used: 79, 717
No_Tags (module), **97**, used:
No_termination (exn), **10**, **11**, **15**, **17**, **8**, used:
null, **826**, **826**, **827**, **828**, **829**, **831**, **823**, **824**, **825**, used: 830, 830, 830, 830, 830, 830, 831, 831, 831, 181
Null (module), **181**, **180**, used: 222, 225, 226, 720, 224
null_coupling, **97**, **97**, **97**, **98**, **93**, used: 108, 112
null_wf, **97**, **97**, **97**, **98**, **93**, used: 101, 108, 108, 112
num (field), **83**, **84**, **80**, used: 86
num_ext, **29**, used: 29
num_prop, **29**, used: 29
N_K1, **344**, used: 344, 347, 353
N_K2, **344**, used: 344, 347, 353
O, **236**, **267**, **??**, **??**, **274**, **333**, **344**, **363**, **374**, **384**, **411**, **428**, **??**, **??**, used: 237, 239, 239, 245, 246, 251, 251, 251, 251, 252, 253, 253, 255, 255, 255, 256, 256, 256, 256, 257, 257, 257, 257, 258, 259, 267, 269, **??**, 274, 276, 277, 281, 290, 290, 290, 291, 333, 335, 335, 337, 339, 339, 339, 339, 339, 339, 339, 340, 340, 344, 346, 347, 350, 352, 352, 352, 352, 352, 352, 352, 353, 353, 363, 365, 365, 368, 369, 369, 369, 369, 369, 369, 369, 370, 370, 375, 376, 376, 379, 380, 380, 380, 380, 380, 381, 385, 386, 386, 390, 390, 390, 390, 390, 391, 391, 391, 411, 412, 412, 417, 418, 418, 418, 418, 418, 418, 419, 420, 420, 421, 421, 422, 422, 428, 428, 429, 429, 429, 430, 430, 431, 431, 431, 432, 432, 432, 432, 432, 434, 434, 434, 434, 435, 435, 435, 435, 435, 436, 437, **??**
O (module), **718**, **??**, **??**, **??**, **718**, **??**, **723**, used: 718, **??**, **??**, **??**, 718, **??**, 723
Ocaml (module), **578**, used:
of2, **10**, **11**, **13**, **13**, **15**, **16**, **8**, **8**, **8**, used: 11, 11, 15, 15, 17, 23, 26, 34
of2_kludge, **9**, **11**, **13**, **15**, **17**, **7**, used: 112
of3, **12**, **12**, **13**, **13**, **15**, **16**, **8**, **8**,

8, used: 13, 15, 26, 26
OFFSHELL, ??, ??, used: ??, 70
OFFSHELL (camlyacc token), **71**, used: 71
Offspring (module), **45**, used: 45, 45, 45, 46, 46, 46, 46, 46, 46, 47, 47, 47, 47
Off_shell, **69**, **74**, **68**, used: 69, 75
Off_shell_not, **69**, **74**, **68**, used: 69, 75
of_array, **796**, **797**, **797**, **796**, used:
of_int, **27**, **27**, **19**, used: 28, 31, 32
of_ints, **54**, **55**, **61**, **50**, used: 59, 66, 696, 697, 697, 75, 108
of_list, ??, ??, ??, **765**, **765**, **796**, **797**, **797**, **781**, **781**, **782**, **782**, **15**, **16**, **83**, **83**, ??, **764**, **796**, **780**, **780**, **8**, **80**, used: ??, ??, 765, 765, 746, 749, 797, 797, 797, 797, 798, 798, 799, 799, 799, 799, 799, 800, 800, 800, 800, 782, 817, 818, 818, 17, 26, 33, 207, 215, 120, 134
of_lists, **765**, **765**, **784**, **785**, **764**, **783**, used: 78, 215
of_list_gap, **799**, used: 799
of_list_ok, **799**, used: 799
of_list_overlap, **799**, used: 799
of_momenta, **692**, **697**, **691**, used:
of_momentum, **66**, **66**, **66**, **53**, used: 66, 700, 701, 702
of_string, **75**, used: 75
of_string_list, **73**, **75**, **72**, used: 715
of_subarray, **743**, **740**, used:
of_vertices, **218**, **221**, **217**, used: 575, 225, 229, 232, 259, 267, 270, 479, ??, ??, 514, 549, 291, 312, 327, 340, 353, 370, 381, 391, 406, 422, 437, ??, ??, 721
Omega (module), **707**, **707**, used: 718, ??, ??, ??, 718, ??, 723
Omega_2HDM (module), ??, used:
Omega_2HDM_CKM (module), ??, used:
Omega_2HDM_CKM_VM (module), ??, used:
Omega_2HDM_VM (module), ??, used:
Omega_AltH (module), ??, used:
Omega_GravTest (module), ??, used:
Omega_HSExt (module), ??, used:
Omega_HSExt_VM (module), ??, used:
Omega_Littlest (module), ??, used:
Omega_Littlest_Eta (module), ??, used:
Omega_Littlest_Tpar (module), ??, used:
Omega_MSSM (module), ??, used:
Omega_MSSM_CKM (module), ??, used:
Omega_MSSM_Grav (module), ??, used:
Omega_MSSM_Hgg (module), ??, used:
Omega_NMSSM (module), ??, used:
Omega_NMSSM_CKM (module), ??, used:
Omega_NMSSM_Hgg (module), ??, used:
Omega_NoH_rx (module), ??, used:
Omega_PSSSM (module), ??, used:
Omega_QCD (module), ??, used:
Omega_QCD_VM (module), ??, used:

Omega_QED (module), **718**, used:
Omega_QED_VM (module), **??**, used:
Omega_Simplest (module), **??**, used:
Omega_Simplest_univ (module), **??**, used:
Omega_SM (module), **718**, used:
Omega_SM_ac (module), **??**, used:
Omega_SM_ac_CKM (module), **??**, used:
Omega_SM_CKM (module), **??**, used:
Omega_SM_CKM_VM (module), **??**, used:
Omega_SM_Higgs (module), **??**, used:
Omega_SM_Higgs_VM (module), **??**, used:
Omega_SM_rx (module), **??**, used:
Omega_SM_top (module), **??**, used:
Omega_SM_top_anom (module), **??**, used:
Omega_SM_tt_threshold (module), **??**, used:
Omega_SM_VM (module), **??**, used:
Omega_SSC (module), **??**, used:
Omega_SYM (module), **718**, used:
Omega_Template (module), **??**, used:
Omega_Threeshl (module), **??**, used:
Omega_Threeshl_nohf (module), **??**, used:
Omega_UED (module), **??**, used:
Omega_Xdim (module), **??**, used:
Omega_Zprime (module), **??**, used:
Omega_Zprime_VM (module), **??**, used:
one, **829**, **27**, **27**, **19**, used: 829, 830, 29, 30, 32, 32
one_compatible, **74**, used: 74, 76
Only_Insertion, **137**, used: 238, 448, ??, ??, 276, 298, 320, 334, 346, 364, 376, 386, 412, 428, ??, ??
ONSHELL, **??**, **??**, used: ??, 70
ONSHELL (camlyacc token), **71**, used: 71
on_shell, **73**, **76**, **94**, **101**, **103**, **122**, **73**, **90**, used: 117, 122
on_shell (field), **76**, **100**, **103**, used: 76, 76, 79, 103, 116, 117, 121
On_shell, **69**, **74**, **68**, used: 69, 75
On_shell_not, **69**, **74**, **68**, used: 69, 75
openmp, **729**, **729**, used:
open_file, **733**, **732**, used:
Open_File, **732**, used: 733
open_in_bin_first, **738**, used: 738
Open_line (exn), **82**, **82**, used:
open_out_bin_last, **738**, used: 738
options, **222**, **576**, **184**, **207**, **94**, **98**, **127**, **128**, **225**, **226**, **227**, **230**, **236**, **266**, **267**, **445**, ??, ??, **496**, **526**, **274**, **297**, **319**, **333**, **344**, **363**, **374**, **384**, **396**, **410**, **428**, ??, ??, **718**, **175**, **178**, **88**, **91**, used: 184, 207, 128, 713, 266, 267
Options (module), **730**, **730**, used: 222, 576, 94, 98, 127, 128, 713, 225, 226, 227, 230, 236, 445, ??, ??, 496, 526, 274, 297, 319, 333, 344, 363, 374, 384, 396, 410, 428, ??, ??, 718, 175, 178, 88, 91
opt_array_to_list, **708**, used: 708
Or, **69**, **68**, used: 69
OR, **??**, **??**, used: ??, 70
OR (camlyacc token), **71**, used: 71, 71

Ord (sig), **40**, **35**, used: 41, 42, 42, 42, 43, 43, 43, 36, 36, 39, 39
ordered_multi_choose, **794**, **789**, used:
ordered_multi_split, **793**, **788**, used: 794
ordered_partitions, **793**, **788**, used:
ordered_partitions', **793**, used: 793, 793
ordered_split, **791**, **788**, used: 792, 793
ordered_split_unsafe, **791**, used: 791, 792
Ordered_Type (sig), **784**, **783**, used: 784, 784
orders, **225**, **226**, **228**, **230**, **242**, **453**, **??**, **??**, **503**, **534**, **280**, **302**, **322**, **337**, **350**, **367**, **378**, **388**, **402**, **416**, **428**, **??**, **??**, **719**, used:
orders (type), **225**, **226**, **228**, **230**, **242**, **453**, **??**, **??**, **502**, **534**, **280**, **302**, **322**, **337**, **349**, **367**, **378**, **388**, **402**, **416**, **428**, **??**, **??**, **719**, used:
order_flavor, **102**, used: 102
order_spin_table, **130**, used: 130
order_spin_table1, **130**, used: 130, 130
Order_Tags (module), **98**, used: 125
order_wf, **100**, **102**, used: 103, 103, 109, 111, 119, 123
other, **208**, **237**, **267**, **??**, **??**, **274**, **333**, **344**, **363**, **375**, **385**, **411**, **428**, **??**, **??**, **177**, used: 237, 237, 267, **??**, **??**, 275, 333, 344, 363, 375, 385, 411, 428, **??**, **??**
other (type), **208**, **236**, **267**, **??**, **??**, **274**, **333**, **344**, **363**, **374**, **384**, **411**, **428**, **??**, **??**, **177**, used: 208, 208, 236, 237, 267, **??**, **??**, 274, 274, 333, 333, 344, 344, 363, 363, 374, 375, 384, 385, 411, 411, 428, **??**, **??**, **??**, **??**, 177, 177
Other, **208**, **237**, **267**, **??**, **??**, **274**, **333**, **344**, **363**, **375**, **385**, **411**, **428**, **??**, **??**, **177**, used: 237, 267, **??**, **??**, 274, 333, 344, 363, 375, 385, 411, 428, **??**, **??**
OUnit (module), **??**, **??**, used: **??**, 765, 765, 799, 799, 765, 796
OUnitDiff (module), **??**, **??**, used:
out (field), **699**, used:
outer, **778**, **776**, used:
outer (field), **813**, **807**, used: 814, 711, 711, 712
outer_self, **778**, **776**, used:
outgoing, **54**, **58**, **59**, **64**, **64**, **210**, **210**, **94**, **101**, **103**, **122**, **52**, **53**, **209**, **90**, used: 122, 129, 129
outgoing (field), **100**, **103**, used: 103, 116, 117, 121, 121
out_ghost_flags, **83**, **84**, **80**, used:
Out_of_bounds (exn), **746**, **741**, used:
out_to_lists, **83**, **84**, **80**, used:
overlapping, **214**, used: 214
overlapping_flavor_sums, **215**, used: 215
Overlapping_indices (exn), **746**, **741**, used:
p (field), **55**, used: 55, 56, 56, 57, 57, 57, 58, 58
p (type), **73**, **74**, **94**, **100**, **101**, **121**, **128**, **72**, **88**, used: 73, 74, 76, 94, 95, 97, 100, 100, 101, 101, 121, 128, 73, 73, 73, 73, 73, 88, 91, 93
P, **137**, used: 457, 457, 458, 458, 459, 459, 505, 536, 282, 283, 304, 304, 325
P (module), **765**, **707**, used: 765, 708
P1, **498**, **528**, used: 498, 505, 506, 506, 509, 510, 512, 513, 514, 529, 537, 537,

538, 543, 544, 546, 547, 548, 549
P123, **190**, used: 201
P132, **190**, used: 201
P2, **498**, **528**, **137**, used: 498, 505, 506, 506, 509, 510, 512, 513, 514, 529, 537, 537, 538, 543, 544, 546, 547, 548, 549
P213, **190**, used: 201
P231, **190**, used: 201
p2s, **124**, **709**, used: 124, 709, 712
P3, **26**, **528**, used: 26, 529, 548, 549
P312, **190**, used: 201
P321, **190**, used: 201
P4, **26**, **528**, used: 26, 529, 548, 549
P5, **528**, used: 529, 548, 549
P6, **528**, used: 529, 548, 549
P7, **528**, used: 529, 548, 549
pack_map, **700**, used: 701
pack_tree, **700**, used: 700
pair, **130**, used: 131
pairs, **802**, **496**, **527**, **802**, used: 11, 15, 21, 33, 497, 497, 510, 513, 527, 527, 544, 548
pairs′, **802**, used: 802, 802
pair_or_triple (type), **13**, **8**, used: 13
parameters, **222**, **571**, **186**, **207**, **225**, **227**, **229**, **232**, **245**, **267**, **268**, **455**, **??**, **??**, **503**, **534**, **280**, **302**, **323**, **337**, **350**, **368**, **379**, **389**, **407**, **416**, **428**, **??**, **??**, **721**, **175**, used: 571, 186, 207, 267, 268
parameters (label), **223**, **175**, used: 575
parameters (type), **171**, used: 175, 175
parameters_to_channel, **178**, used: 715
Parents (module), **45**, used: 45, 45, 45, 45, 45, 45, 46, 46, 46, 46, 46, 46, 46, 46, 46, 47, 47, 47, 47, 47, 48
parse, **728**, **730**, **726**, **730**, used: **??**, 802, 9, 20, 40, 54, 218, 698, 568, 183, 93, 713, 224, 444, **??**, 495, 526, 273, **??**, 678, **??**, 718
parse1, **727**, used: 728
parse_color, **574**, used: 574
parse_decay, **210**, **212**, **209**, used: 714
parse_error, **??**, **71**, used: **??**, **??**, **??**, **??**
parse_expr, **573**, used: 573, 573, 574
parse_function_row, **573**, used: 574
parse_lagragian_row, **573**, used: 574
parse_particle_row, **574**, used: 574
parse_process, **210**, **211**, **209**, used: 212, 212
parse_scattering, **210**, **212**, **209**, used: 714
parse_source, **727**, used: 728
parse_spin, **574**, used: 574
parse_symbol, **573**, used: 574
parse_table, **574**, used: 574
parse_variable_row, **574**, used: 574
partial, **828**, **831**, **825**, used:
Partial (module), **765**, **764**, used:
particle (type), **571**, used:
particles_file, **576**, used: 576, 576
particle_flavor (type), **572**, used:
partition, **24**, **33**, **73**, **76**, **73**, used: 786, 24, 33, 133
partition (field), **76**, used: 76, 76
partition (type), **20**, **21**, **24**, **26**, **26**, **33**, **33**, **18**, used: 20, 18, 19
Partition (module), **802**, **802**, used: 11, 13, 15, 17, 21, 24, 33, 33
partitioned_sort, **746**, **741**, used: 215
partitions, **793**, **20**, **22**, **24**, **26**, **26**, **33**, **33**, **788**, **18**,

used: 793, 24, 24, 26, 26, 32, 33, 34
partitions', **793**, **21**, **24**, **33**, used: 793, 793, 794, 21, 22, 24, 24, 33, 33
partition_to_string, **78**, used: 79
part_to_string, **78**, used: 78
path (type), **??**, **??**, used: ??, ??, ??, ??, ??, ??
PBP, **138**, used: 188, 461, 461, 507, 507, 538, 539, 540, 540
pdg, **222**, **571**, **184**, **207**, **226**, **227**, **229**, **233**, **263**, **267**, **267**, **484**, **??**, **??**, **519**, **555**, **294**, **315**, **330**, **342**, **361**, **373**, **383**, **394**, **399**, **424**, **441**, **??**, **??**, **723**, **172**, used: 571, 184, 207, 229, 229, 233, 233, 263, 264, 267, 267, ??, ??, ??, ??, 294, 294, 316, 316, 343, 343, 362, 362, 373, 373, 383, 384, 394, 394, 425, 425, 441, 441, ??, ??, ??, ??
pdg (label), **223**, **175**, used: 575
pdg_heuristic, **570**, used: 573, 573
pdg_mw, **487**, **522**, **559**, **331**, used: 487, 487, 522, 522, 560, 560, 331, 331
perform_test, **??**, **??**, used: ??
permutation, **96**, **127**, used: 96, 127
Permutation (module), **796**, **796**, used:
permutation4 (type), **190**, used:
permutation_symmetry, **32**, used: 32
permute, **795**, **789**, used: 795, 204
permute3, **220**, used: 220
permute4, **221**, used: 221
permute_contract4, **190**, used: 191
permute_contract4_list, **191**, used: 191
permute_even, **795**, **789**, used:
permute_odd, **795**, **789**, used:
permute_signed, **795**, **789**, used: 795, 795, 795
permute_signed', **795**, used: 795, 795
permute_tensor, **795**, **789**, used:
permute_tensor_even, **795**, **789**, used:
permute_tensor_odd, **795**, **789**, used:
permute_tensor_signed, **795**, **789**, used: 795, 795
permute_vertex4, **191**, used: 201
permute_vertex4', **191**, used: 191
Phasespace (module), **692**, **691**, used:
Phi, **225**, **226**, **718**, used: 225, 225, 226, 226, 227, 227, 227, 719, 720, 721
Phi0, **236**, **447**, **??**, **??**, **274**, **297**, **319**, **333**, **344**, **363**, **374**, **384**, **411**, **428**, **??**, **??**, used: 237, 239, 239, 253, 447, 449, 449, 457, 458, 459, 464, 465, 467, 468, 468, 470, 472, 473, 473, 479, ??, ??, ??, ??, 275, 276, 277, 290, 298, 299, 299, 312, 319, 320, 320, 327, 333, 335, 335, 340, 344, 346, 347, 353, 363, 365, 365, 370, 375, 376, 376, 380, 385, 386, 386, 391, 411, 412, 412, 422, 428, 431, ??, ??, ??, ??, ??, ??, ??, ??
Phi3 (module), **225**, **224**, used:
Phi4 (module), **226**, **224**, used:
phiggs (type), **498**, **528**, used: 498, 501, 529, 532
PHiggs, **498**, **529**, used: 498, 500, 505, 506, 506, 508, 508, 509, 509, 510, 510, 512, 513, 514, 529, 531, 536, 537, 537, 541, 541, 543, 543, 544, 544, 544, 546, 547, 549
phiggs_neutr, **506**, **537**, used: 506, 538
Phim, **236**, **447**, **??**, **??**, **274**, **297**, **319**, **333**, **344**, **363**,

374, **384**, **411**, **428**, **??**, **??**, used: 237, 239, 239, 253, 447, 449, 449, 457, 458, 458, 464, 465, 467, 468, 468, 470, 470, 472, 473, 473, 479, ??, ??, ??, ??, 275, 276, 277, 290, 298, 299, 299, 312, 319, 320, 320, 327, 333, 335, 335, 340, 344, 346, 347, 353, 363, 365, 365, 370, 375, 376, 376, 380, 385, 386, 386, 391, 411, 412, 412, 422, 428, 431, ??, ??, ??, ??, ??, ??, ??, ??

Phino, **363**, used: 363, 365, 369, 370

Phip, **236**, **447**, **??**, **??**, **274**, **297**, **319**, **333**, **344**, **363**, **374**, **384**, **411**, **428**, **??**, **??**, used: 237, 239, 239, 253, 447, 449, 449, 457, 458, 458, 464, 465, 467, 468, 468, 470, 470, 472, 473, 473, 479, ??, ??, ??, ??, 275, 276, 277, 290, 298, 299, 299, 312, 319, 320, 320, 327, 333, 335, 335, 340, 344, 346, 347, 353, 363, 365, 365, 370, 375, 376, 376, 380, 385, 386, 386, 391, 411, 412, 412, 422, 428, 431, ??, ??, ??, ??, ??, ??, ??, ??

Photon, **227**, used: 228, 228, 229, 229

pi, **782**, **782**, **782**, **216**, **118**, **780**, **780**, used: 782, 782, 782

Pi, **241**, **450**, **??**, **??**, **278**, **300**, **321**, **337**, **349**, **367**, **378**, **388**, **414**, **428**, **??**, **??**, used: 244, ??, ??, 428

pivot_column, **835**, used: 835

plist, **529**, used: 529

PLUS, **??**, **??**, **??**, **??**, used: ??, ??, 70, 566

PLUS (camlyacc token), **71**, **567**, used: 71, 72, 567, 567

PM (module), **826**, used: 828, 829, 829, 830, 831, 832

Pmap (module), **755**, **754**, used: 749, 773, 826, 816, 769

PMap (module), **749**, used: 749

poles, **94**, **123**, **91**, used: 700, 123

Poles (module), **699**, used: 699, 699, 700, 701

poles′, **124**, used: 124

poles_beneath, **123**, used: 123

poles_to_whizard, **699**, used: 700

Pole_Map (module), **700**, used: 700, 700, 700

Poly (sig), **772**, **10**, **768**, **8**, used: 773, 10, 13, 15, 42, 97, 97, 97, 98, 98, 101, 769, 8, 8, 8, 36, 93

Pos, **397**, used: 398, 400, 403, 403, 403, 403, 404, 404, 404, 404, 404, 405, 405, 405, 405, 405, 407, 408

Positron, **227**, used: 228, 228, 229, 229

POT, **137**, used: 369

Pow, **166**, used:

power, **827**, **828**, **829**, **777**, **9**, **11**, **12**, **15**, **17**, **565**, **824**, **776**, **7**, **564**, used: 829, 830, 830, 830, ??, 718, 718, 718, 567

power (field), **83**, **84**, **80**, used:

power (type), **83**, **84**, **80**, used: 83, 84, 80

POWER, **??**, **??**, used: ??, 566

POWER (camlyacc token), **567**, used: 567, 567

power_fold, **9**, **11**, **12**, **15**, **17**, **7**, used: 15, 104

PowSet (module), **784**, **783**, used: 78, 215

pp_comma_separator, **??**, **??**, used:

pp_diff, **??**, **??**, **??**, **??**, used: ??, ??

pp_printer, **??**, **??**, **??**, **??**, **??**, **??**, used: ??, ??, ??, ??, ??, ??

pp_print_gen, **??**, used: ??, ??, ??

pp_print_sep, **??**, **??**, used: ??, ??, ??

pred, **27**, **27**, **19**, used: 742, 742, 742, 742, 743, 745, 746, 746, 749, 798, 798, 833, 834, 835, 835, 836, 829, 789, 790, 790, 791, 792, 792, 794, 802, 803, 818, 819, 11, 12, 16, 17, 17, 24, 27, 30, 30, 31, 32, 33, 33, 34, 61, 62, 65, 66, 86, 568, 569, 569, 215, 215, 134, 708, 497, 527
prepend_to_fst_unsorted, **792**, used: 793
Pre_Bundle (module), **213**, used: 213
print, **678**, **678**, used:
print_time, **799**, used: 800
Proc (module), **131**, **708**, used: 133, 133, 714
process (type), **210**, **211**, **127**, **128**, **209**, **91**, used: 210, 127, 129, 129, 209, 92, 92, 92
Process (module), **210**, **209**, used: 131, 708
processes, **127**, **129**, **92**, used:
processes (field), **129**, used: 129, 134
Process_Bundle (module), **213**, used: 215
process_sans_color, **129**, used: 133, 133
process_table, **127**, **129**, **92**, used: 708
process_table (field), **129**, used: 129, 134, 134
process_to_string, **129**, used: 133
Prod, **166**, used: 244, 245, 245, **??**, **??**, 280, 280, 302, 302, 322, 322, 337, 337, 350, 350, 367, 367, 378, 378, 389, 389, 416, 416, 428, **??**, **??**
product, **785**, **827**, **827**, **829**, **830**, **9**, **11**, **12**, **15**, **16**, **824**, **825**, **6**, used: 786, 11, 12, 119, 119
Product, **564**, **564**, used: 565, 565, 565
Product (module), **777**, **776**, used: 795, 795, 810, 11, 11, 12, 12, 15, 16, 17, 17, 47, 701, 212, 213, 108, 108, 119, 123, 123, 123, 131, 246, 456, 456, 459, 461, 461, 463, 465, 466, 466, 466, 469, 470, 470, 470, 471, 472, 473, 473, 474, 474, 474, 474, 475, 475, 475, 475, 476, 477, 478, 478, **??**, **??**, 504, 505, 505, 506, 507, 507, 507, 509, 511, 511, 511, 512, 512, 513, 513, 513, 536, 536, 537, 537, 538, 538, 539, 541, 541, 541, 541, 543, 545, 545, 545, 546, 547, 547, 547, 548, 428, 718, 718, 718
product_fold, **9**, **11**, **12**, **15**, **16**, **6**, used: 11, 12, 15, 16
Progress (module), **732**, **732**, used: 133, 133, 133
progress_mode (type), **128**, used:
progress_option, **128**, used: 128, 133
project, **267**, used: 267, 268, 268, 268, 268, 269, 271, 271
Projection (sig), **782**, **780**, used: 782, 780
prop (field), **29**, used: 29, 30, 30, 30, 30
propagator, **222**, **571**, **185**, **207**, **225**, **226**, **228**, **230**, **238**, **266**, **267**, **448**, **??**, **??**, **499**, **530**, **276**, **298**, **320**, **334**, **346**, **364**, **376**, **386**, **400**, **412**, **428**, **??**, **??**, **719**, **174**, used: 571, 185, 207, 104, 710, 710, 266, 267
propagator (label), **223**, **175**, used: 575
propagator (type), **137**, used: 572, 174, 175
propagator_of_lorentz, **572**, used: 573, 573

prop_aux, **238**, ??, ??, **428**, used: 238, ??, ??, 428
Prop_Col_Feynman, **137**, used: 184
Prop_Col_Majorana, **137**, used: 184
Prop_Col_Scalar, **137**, used: 184
Prop_Col_Unitarity, **137**, used: 184
Prop_ConjSpinor, **137**, used: 572, 228, 230, 238, 448, ??, ??, 499, 530, 276, 276, 298, 298, 320, 334, 346, 364, 376, 386, 400, 412, 428, ??, ??, 719
Prop_Feynman, **137**, used: 572, 228, 230, 238, 448, ??, ??, 499, 530, 276, 298, 320, 334, 346, 364, 376, 386, 400, 412, 428, ??, ??, 719
Prop_Gauge, **137**, used: 266
Prop_Ghost, **137**, used:
Prop_Majorana, **137**, used: 572, 448, 499, 530, 364, 719
Prop_Rxi, **137**, used: 266
Prop_Scalar, **137**, used: 572, 572, 225, 226, 238, 266, 448, ??, 499, 530, 276, 298, 320, 334, 346, 364, 376, 386, 412, 428, ??, ??, 719
prop_spinor, **238**, **448**, ??, ??, **499**, **530**, **276**, **298**, **320**, **334**, **346**, **364**, **376**, **386**, **412**, **428**, ??, ??, used: 238, 448, ??, ??, 499, 530, 276, 298, 320, 334, 346, 364, 376, 386, 412, 428, ??, ??
Prop_Spinor, **137**, used: 572, 228, 230, 238, 448, ??, ??, 499, 530, 276, 276, 298, 298, 320, 334, 346, 364, 376, 386, 400, 412, 428, ??, ??, 719
Prop_Tensor_2, **137**, used: ??, 334, 346, 428
Prop_Unitarity, **137**, used: 572, 238, 448, ??, ??, 499, 530, 276, 298, 320, 334, 346, 364, 376, 386, 400, 412, 428, ??, ??
Prop_Vectorspinor, **137**, used: 364
PSet (module), **215**, used: 216, 216
Psi, **137**, used: 229, 231, 245, 245, 246, 246, 246, 246, 246, 253, 254, 254, 254, 254, 254, 254, 255, 255, 255, 255, 256, 256, 256, 256, 256, 257, 257, 257, 257, 257, 257, 258, 258, 258, 455, 455, 455, 456, 456, 456, 457, 457, 457, 457, 458, 458, 458, 458, 458, 459, 461, 461, 462, 462, 462, 462, 462, 476, 476, 476, ??, 503, 503, 503, 504, 504, 504, 505, 505, 505, 506, 506, 506, 506, 507, 507, 507, 508, 534, 535, 535, 535, 535, 535, 536, 536, 537, 537, 537, 537, 537, 538, 538, 538, 539, 539, 539, 539, 540, 540, 540, 540, 540, 540, 540, 540, 541, 541, 541, 541, 542, 542, 542, 281, 281, 281, 281, 281, 282, 282, 282, 282, 283, 290, 302, 303, 303, 303, 303, 303, 304, 304, 304, 304, 312, 323, 323, 323, 323, 324, 325, 325, 325, 325, 327, 338, 338, 338, 338, 338, 350, 350, 350, 351, 351, 368, 368, 368, 368, 369, 379, 379, 379, 379, 379, 389, 389, 389, 389, 390, 403, 403, 403, 403, 404, 404, 404, 404, 417, 417, 417, 417, 417, 418, 418, 418, 418, 418, 418,

419, 428, 432, 433, 433, 433, 433, 433, 434, 434, 434, 434, 435, 435, 435, 435, 435, 435, 435, 436, 436, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, 720

Psi0, **274**, **297**, used: 275, 277, 282, 285, 286, 287, 291, 298, 299, 304, 307, 307, 308, 312

Psi1, **274**, **297**, used: 275, 277, 282, 286, 287, 291, 298, 299, 304, 307, 308, 312

Psibar, **137**, used: 229, 231, 245, 245, 246, 246, 246, 246, 246, 253, 254, 254, 254, 254, 254, 254, 255, 255, 255, 255, 256, 256, 256, 256, 256, 257, 257, 257, 257, 257, 257, 258, 258, 258, 455, 455, 455, 456, 456, 456, 457, 457, 457, 457, 458, 458, 458, 458, 458, 459, 461, 461, 462, 462, 462, 462, 462, 476, 476, 476, ??, 503, 503, 503, 504, 504, 504, 505, 505, 505, 506, 506, 506, 506, 507, 507, 507, 508, 534, 535, 535, 535, 535, 535, 536, 536, 537, 537, 537, 537, 537, 538, 538, 538, 539, 539, 539, 539, 540, 540, 540, 540, 540, 540, 540, 540, 541, 541, 541, 542, 542, 542, 281, 281, 281, 281, 281, 282, 282, 282, 282, 283, 290, 302, 303, 303, 303, 303, 303, 304, 304, 304, 304, 312, 323, 323, 323, 323, 324, 325, 325, 325, 325, 327, 338, 338, 338, 338, 338, 350, 350, 350, 351, 351, 368, 368, 368, 368, 369, 379, 379, 379, 379, 379, 389, 389, 389, 389, 390, 403, 403, 403, 403, 404, 404, 404, 404, 417, 417, 417, 417, 417, 418, 418, 418, 418, 418, 418, 419, 428, 432, 433, 433, 433, 433, 433, 434, 434, 434, 434, 435, 435, 435, 435, 435, 435, 435, 436, 436, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, 720

Psim, **274**, **297**, used: 275, 277, 282, 285, 286, 287, 291, 298, 299, 304, 307, 307, 308, 312

Psimm, **274**, **297**, used: 275, 277, 285, 286, 287, 291, 298, 299, 307, 307, 308, 312

Psip, **274**, **297**, used: 275, 277, 282, 285, 286, 287, 291, 298, 299, 304, 307, 307, 308, 312

Psipp, **274**, **297**, used: 275, 277, 285, 286, 287, 291, 298, 299, 307, 307, 308, 312

PSSSM (module), **526**, **525**, used: ??

PSSSM_QCD (module), **526**, **525**, used:

pullback, **184**, used: 184, 184, 185, 186, 188

pure_adjoints, **205**, used: 205

PV, **137**, used:

p_aux (field), **571**, used: 573, 573, 574

p_color (field), **571**, used: 573, 573, 574

p_mass (field), **571**, used: 573, 573, 574

p_name (field), **571**, used: 573, 573, 574

P_NNA, **450**, used:

P_NNG, **450**, used:

P_NNH1, **450**, used:

P_NNH2, **450**, used:

p_spin (field), **571**, used: 573, 573, 574

p_symbol (field), **571**, used: 572, 574, 574
p_to_string, **697**, used: 697, 697
P_Whizard (module), **707**, used: 708
p_width (field), **571**, used: 573, 573, 574
Q, **228**, **718**, used: 229, 229, 719, 720, 720, 721
Q (module), **181**, **181**, used:
QBUB, **236**, ??, ??, **428**, used: 237, 257, 257, 259, ??, ??, ??, ??, ??, ??, ??, ??, 428, 435, 435, 437
QCD (module), **230**, **224**, used: ??, ??
qcd_ew_check, **107**, used:
qcd_order, **98**, used: 98, 98
qcd_order (type), **98**, used:
QCD_order, **98**, used: 98, 98
QED (module), **227**, **224**, used: 718, ??
qgc, **248**, ??, ??, **281**, **338**, **351**, **369**, **380**, **390**, **420**, **428**, ??, ??, used: 248, 249, 250, ??, ??, 284, 284, 338, 351, 369, 380, 390, 420, 428, ??, ??
QGUG, **236**, ??, ??, **428**, used: 237, 256, 256, 259, ??, ??, ??, ??, ??, ??, ??, ??, 428, 435, 435, 437
QH, **319**, used: 319, 320, 323, 323, 324, 325, 325, 327, 327
qn_baryon, **401**, used: 401
qn_charge, **401**, used: 401
qn_generation, **401**, used: 401
qn_lepton, **401**, used: 401
QQ (module), **182**, **181**, used: 240, 266, 453, ??, ??, 500, 531, 277, 300, 321, 335, 348, 366, 377, 387, 400, 413, 428, ??, ??, 224, 224, 224, 444, ??, 495, 525, 272, 272, 273, ??
quark, **397**, used: 398, 407
Quark, **397**, used: 397, 400, 403, 403, 404, 404, 404
quark_current, **720**, used: 721
quark_currents, **304**, used: 312
quartic_anom, **234**, **234**, **234**, **234**, **235**, **235**, **235**, **235**, **236**, ??, ??, **427**, **428**, **224**, ??, **273**, used: 249, ??, ??, 428
quartic_gauge, **251**, **463**, ??, ??, **508**, **542**, **285**, **306**, **326**, **339**, **352**, **369**, **380**, **390**, **420**, **428**, ??, ??, used: 259, 478, ??, ??, 514, 549, 291, 312, 327, 340, 353, 370, 381, 391, 422, 437, ??, ??
quartic_gravitino, **476**, used: 478
quartic_grav_char, **476**, used: 476
quartic_grav_gauge, **476**, used: 476
quartic_grav_neu, **476**, used: 476
quartic_grav_sneutrino, **476**, used: 476
Quiet, **128**, used: 128
Quot, **166**, used: 244, 245, 245, 245, ??, ??, 280, 280, 280, 302, 302, 302, 322, 322, 322, 337, 337, 337, 350, 350, 350, 367, 368, 368, 378, 379, 379, 389, 389, 389, 416, 416, 416, 428, ??, ??, ??
quote, **753**, **751**, used: 714
QUQD1L, **236**, ??, used: 237, 257, 258, 259, ??, ??, ??, ??
QUQD1R, **236**, ??, used: 237, 257, 258, 259, ??, ??, ??, ??
QUQD8L, **236**, ??, used: 237, 258, 258, 259, ??, ??, ??, ??
QUQD8R, **236**, ??, used: 237, 258, 258, 259, ??, ??, ??, ??
QW, **236**, ??, ??, **428**, used: 237, 257, 257, 259, ??, ??, ??, ??, ??, ??, ??, ??, 428, 435, 436, 437
Q_charg, **450**, **501**, **532**, used: 455, 503, 535

Q_down, **241**, **450**, ??, ??, **501**, **532**, **278**, **300**, **321**, **337**, **349**, **367**, **378**, **388**, **414**, **428**, ??, ??, used: 244, 245, 455, 455, ??, ??, 503, 503, 534, 534, 542, 542, 281, 302, 323, 325, 338, 350, 368, 379, 389, 417, 428, ??, ??
Q_lepton, **241**, **450**, ??, ??, **501**, **532**, **278**, **300**, **321**, **337**, **349**, **367**, **378**, **388**, **414**, **428**, ??, ??, used: 244, 245, 455, 455, ??, ??, 503, 503, 534, 534, 281, 286, 302, 307, 323, 338, 350, 368, 379, 389, 417, 428, ??, ??
Q_top, ??, used: ??
Q_up, **241**, **450**, ??, ??, **501**, **532**, **278**, **300**, **321**, **337**, **349**, **367**, **378**, **388**, **414**, **428**, ??, ??, used: 244, 245, 455, 455, ??, ??, 503, 503, 534, 534, 281, 281, 302, 303, 323, 325, 338, 350, 368, 379, 389, 417, 428, ??, ??
Q_Z_up, **278**, **300**, **321**, used: 281, 303
r (field), **55**, used: 55, 55, 55, 56, 56, 56, 56, 57, 57, 57, 58, 58, 59, 59, 60, 60
raises, ??, used: ??
range, **743**, **740**, used: 746, 797, 797, 799, 800, 800, 800, 800, 16, 17, 60, 86, 575, 74, 204, 116, 117, 120, 269, 718
Range (exn), **54**, **55**, **61**, **51**, used:
rank, **42**, **43**, **43**, **43**, **45**, **48**, **54**, **55**, **61**, **83**, **83**, **87**, **109**, **39**, **40**, **51**, **80**, used: 44, 45, 62, 62, 62, 64, 64, 64, 64, 64, 64, 64, 64, 64, 65, 104, 109, 124
rank (type), **43**, **44**, **45**, **48**, **49**, **40**, used: 43, 44, 44, 48, 48, 40, 40
rank0, **61**, used: 61, 62, 62, 62, 63, 63, 63
ranked, **43**, **44**, **45**, **48**, **49**, **40**, used: 45, 110
ranks, **43**, **44**, **45**, **48**, **49**, **40**, used: 45
rank_default, **87**, used: 87
rank_set, **44**, used: 44
Rat (module), **181**, used: 181, 182
Rational (sig), **826**, **823**, used: 826, 827, 829, 823, 824, 825
raw (field), **730**, used: 730, 730, 730, 731, 731
raw (type), **727**, **564**, **726**, **564**, used: 564, ??, 571, 726, 564, 564, 564, ??, 567
rcs, **802**, **9**, **10**, **12**, **13**, **15**, **20**, **21**, **24**, **26**, **26**, **28**, **33**, **33**, **41**, **45**, **54**, **55**, **61**, **224**, **698**, **571**, **183**, **207**, **95**, **95**, **98**, **125**, **225**, **226**, **227**, **230**, **236**, **266**, **267**, **445**, ??, ??, **496**, **526**, **274**, **297**, **319**, **332**, **343**, **363**, **374**, **384**, **396**, **410**, **428**, ??, ??, **718**, **175**, **802**, **7**, **19**, **39**, **53**, used: 105, 105, 125, 712, 267
RCS (module), **727**, **726**, used: 802, 9, 9, 10, 12, 13, 15, 20, 20, 21, 24, 26, 26, 33, 33, 40, 41, 45, 54, 54, 55, 61, 218, 224, 698, 568, 183, 183, 207, 93, 94, 95, 95, 98, 105, 105, 105, 125, 712, 224, 225, 226, 227, 230, 236, 266, 267, 444, 445, ??, ??, ??, 495, 496, 526, 526, 273, 332, 343, 363, 374, 384, 396, 428, ??, 678, ??, 718, 175, 178, 802, 7, 19, 39, 53, 91
rcs_author (field), **727**, used: 727, 728
rcs_date (field), **727**, used: 727, 728

rcs_file, **9**, **20**, **40**, **54**, **218**, **568**, **183**, **93**, **224**, **444**, **??**, **495**, **526**, **273**, **??**, **678**, **??**, **718**, used: 10, 12, 13, 15, 21, 24, 26, 26, 28, 33, 33, 45, 55, 61, 224, 571, 183, 207, 95, 98, 125, 225, 226, 227, 230, 236, 266, 445, **??**, **??**, 496, 526, 274, 297, 319, 332, 343, 363, 374, 384, 396, 410, 428, **??**, **??**, 718
rcs_list, **94**, **125**, **178**, **91**, used: 712
rcs_revision (field), **727**, used: 727, 728
rcs_source (field), **727**, used: 727, 728
read, **737**, **738**, **736**, used: 739
read_generation, **268**, used: 268
read_lines, **712**, used:
read_lines_rev, **712**, used: 712, 713
Real, **171**, used: 575, 244, **??**, **??**, 428
Real_Array, **171**, used: 245, **??**, **??**, 280, 302, 322, 337, 350, 367, 378, 389, 416, 428, **??**
real_variable, **572**, used: 572, 573, 573, 575
Rec, **166**, used:
remove, **755**, **758**, **763**, **769**, **769**, **770**, **772**, **773**, **30**, **754**, **767**, **768**, **768**, used: **??**, 758, 763, 770, 770, 773, 773, 828, 829, 830, 831, 30, 30, 30, 215
remove_duplicate_final_states, **210**, **215**, **210**, used: 133
rename, **728**, **726**, used: 10, 12, 13, 15, 21, 24, 26, 26, 33, 33, 45, 55, 61, 224, 183, 207, 95, 98, 125, 225, 226, 227, 230, 236, 266, 267, 445, **??**, **??**, 496, 526, 332, 343, 363, 374, 384, 396, 428
repeat, **800**, used: 801
RError, **??**, **??**, used: **??**
reset, **734**, **732**, used:
result (type), **737**, **738**, **736**, used: 737, 736
result_flavour, **??**, used: **??**
result_msg, **??**, used: **??**
result_path, **??**, used: **??**
rev (field), **811**, **806**, used: 811, 812, 812, 813, 813, 709, 710, 710
revision, **727**, **726**, used: 728
revision (field), **727**, **726**, used: 728, 802, 9, 20, 40, 54, 218, 698, 568, 183, 93, 224, 444, **??**, 495, 526, 273, **??**, 678, **??**, 718
revmap, **398**, used: 398, 398
revmap2, **398**, used: 398
rev_flatmap, **743**, **741**, used: 795, 206
rev_multiply, **746**, used: 746, 746
Rf, **??**, **428**, used: **??**, **??**, **??**, **??**, 428, 432, 432, 437
rf3, **??**, **432**, used: **??**, 436
rf3t, **??**, **432**, used: **??**, 436
RFailure, **??**, **??**, used: **??**
rhs, **94**, **100**, **103**, **121**, **89**, used: 121
rhs (type), **94**, **100**, **103**, **121**, **89**, used: 94, 100, 100, 100, 103, 103, 108, 119, 121, 89, 89, 89, 89, 89
Ring (sig), **827**, **824**, used: 828, 829, 831, 825, 825
RMOM, **137**, used:
RPAREN, **??**, **??**, **??**, **??**, used: **??**, **??**, 70, 566
RPAREN (camlyacc token), **71**, **567**, used: 71, 567
rphi3, **??**, **432**, used: **??**, 436
Rphim, **??**, **428**, used: **??**, **??**, **??**, 428, 432, 437
Rphimm, **??**, **428**, used: **??**, **??**, **??**, 428, 432, 437
Rphin, **??**, **428**, used: **??**, **??**, **??**, 428, 432, 437
Rphip, **??**, **428**, used: **??**, **??**, **??**, 428, 432, 437
Rphipp, **??**, **428**, used: **??**, **??**, 428, 432

Rsigma, ??, **428**, used: ??, ??, ??, ??, ??, 428, 431, 431, 432, 437
rsigma3, ??, **431**, used: ??, 436
rsigma3t, ??, **431**, used: ??, 436
rsigma4, ??, **432**, used: ??, 437
RSkip, ??, ??, used: ??
RSuccess, ??, ??, used: ??
rt3, ??, **432**, used: ??, 436
Rtm, ??, **428**, used: ??, ??, ??, 428, 432, 437
Rtmm, ??, **428**, used: ??, ??, ??, 428, 432, 437
Rtn, ??, **428**, used: ??, ??, ??, 428, 432, 437
RTodo, ??, ??, used: ??
Rtp, ??, **428**, used: ??, ??, ??, 428, 432, 437
Rtpp, ??, **428**, used: ??, ??, 428, 432
Running, **137**, used:
run_test_tt, ??, ??, used: ??
run_test_tt_main, ??, ??, used:
R_CN, **450**, used:
R_CNG, **450**, used:
R_NC, **450**, used:
R_NCH, **450**, used:
S, **230**, **384**, **137**, used: 230, 230, 231, 232, 246, 457, 458, 459, 505, 536, 282, 282, 304, 304, 325, 338, 351, 368, 379, 385, 386, 390, 390, 390, 390, 390, 391, 391, 428, ??, ??
S (module), ??, **99**, **567**, used: ??, 99, 99, 567
S (sig), ??, ??, used: ??, ??, ??, ??
S1, **497**, **528**, used: 498, 505, 506, 506, 509, 510, 512, 512, 513, 514, 529, 537, 537, 538, 543, 544, 546, 547, 547, 548, 549
S1 (module), **44**, used: 44
S2, **497**, **528**, **137**, used: 498, 505, 506, 506, 509, 510, 512, 512, 513, 514, 529, 537, 537, 538, 543, 544, 546, 547, 547, 548, 549, 290, 312, 327
S2 (module), **44**, used: 44
S2L, **137**, used:
S2LR, **137**, used: 290, 312
S2P, **137**, used:
S2R, **137**, used:
S3, **497**, **528**, used: 498, 505, 506, 506, 509, 510, 512, 512, 513, 514, 529, 537, 537, 538, 543, 544, 546, 547, 547, 548, 549
S4, **528**, used: 529, 548, 549
S5, **528**, used: 529, 548, 549
S6, **528**, used: 529, 548, 549
S7, **528**, used: 529, 548, 549
S8, **528**, used: 529, 548, 549
S9, **528**, used: 529, 548, 549
sanitize_symbol, **570**, used: 573
sans_colors, **129**, used: 129
Sbar, **230**, used: 230, 230, 231, 232
Scalar, **135**, used: 572, 574, 225, 226, 237, 238, 448, ??, ??, 499, 530, 275, 298, 319, 334, 345, 364, 375, 385, 411, 428, ??, ??, 719
Scalar2_Vector2, **139**, used: 189, 251, 464, 465, 465, 466, 466, 466, 467, ??, 509, 509, 509, 509, 509, 509, 509, 509, 509, 510, 510, 510, 510, 510, 511, 511, 511, 511, 512, 542, 542, 542, 543, 543, 543, 543, 543, 544, 544, 544, 544, 544, 544, 544, 544, 545, 545, 545, 545, 546, 546, 287, 287, 308, 308, 326, 326, 339, 352, 369, 380, 390, 420, 429, 432, ??, ??, 721
Scalar4, **139**, used: 575, 189, 191, 227, 227, 251, 467, 468, 468, 471, 471, 472, 472, 473, 473, 474, 474, 474, 474, 474, 475, 475, 475, 475, 290, 311, 326, 339, 352, 369, 380, 390, 421, 429, ??, ??
Scalar_Scalar_Scalar, **138**, used: 575, 188, 225, 227, 227,

251, 467, 467, 469, 469, 470, 470, 470, 512, 512, 513, 513, 513, 540, 541, 541, 541, 546, 547, 547, 547, 547, 290, 311, 326, 339, 352, 369, 380, 390, 421, 429, ??, ??

Scalar_Vector_Vector, **138**, used: 188, 251, 253, 463, 464, ??, ??, ??, ??, 508, 543, 285, 285, 285, 290, 306, 306, 307, 312, 326, 326, 327, 339, 340, 352, 353, 369, 370, 380, 380, 390, 391, 420, 422, 428, 431, 431, 432, ??, ??, ??, ??

Scalar_Vector_Vector_t, **138**, used: 188, ??, 431

scale, **827**, **828**, **830**, **831**, **565**, **824**, **825**, used: 830, 831, 565, 565

scattering (type), **210**, **211**, **209**, used: 210, 211, 209, 209

Scattering, **210**, **211**, **209**, used: 211, 212, 212

Scattering (module), **54**, **58**, **64**, **52**, used: 696, 696, 699, 699, 699, 702, 107, 116, 123, 124

Sccs, **278**, **300**, **321**, used:

SD, **446**, **497**, **527**, used: 456, 459, 461, 465, 466, 469, 470, 471, 473, 474, 474, 505, 506, 507, 507, 510, 510, 511, 513, 513, 536, 538, 538, 539, 544, 545, 546, 547, 547

Sdown, **447**, **498**, **529**, used: 447, 449, 455, 456, 456, 459, 461, 462, 462, 465, 466, 466, 466, 467, 469, 470, 470, 470, 471, 472, 473, 473, 474, 474, 475, 475, 475, 476, 479, 498, 500, 503, 504, 505, 506, 507, 507, 508, 510, 510, 511, 511, 511, 512, 513, 513, 513, 514, 529, 531, 534, 536, 536, 538, 538, 539, 540, 540, 541, 542, 544, 545, 545, 545, 546, 546, 547, 547, 547, 549

sdown4_1gen, **475**, used: 478

sdown4_1gen', **475**, used: 475

sdown4_2gen, **475**, used: 478

sdown4_2gen', **475**, used: 475

search_path, **737**, used: 738, 738, 738

selectors (type), **73**, **76**, **94**, **112**, **127**, **128**, **72**, **89**, **91**, used: 73, 73, 73, 94, 95, 112, 127, 128, 128, 72, 72, 73, 73, 73, 73, 73, 73, 89, 91, 92, 93

select_p, **73**, **76**, **73**, used: 116

select_p (field), **76**, used: 76, 76, 79

select_wf, **73**, **76**, **73**, used: 116

select_wf (field), **76**, used: 76, 76, 79

Selfconjugate, **571**, used: 573

seq (type), **789**, **787**, used: 788, 788, 788, 788, 788, 788, 788, 789, 789, 789, 789, 789

set, **216**, used: ??, 746, 749, 749, 798, 798, 798, 833, 834, 834, 835, 835, 836, 817, 817, 818, 570, 574, 216, 133, 134

set (type), **784**, **778**, **776**, used:

Set (module), **??**, used: ??, ??, ??, ??, ??, ??, ??, 746, 781, 784, 784, 44, 44, 45, 220, 221, 700, 702, 186, 214, 215, 215, 103, 119, 123, 131

SetMake (module), **??**, **??**, used:

SetTestPath (module), **??**, used: ??

setup, **223**, **175**, used: 575

set_cache_name, **94**, **105**, **127**, **134**, **90**, **92**, used: 134

set_color, **222**, used: 223

set_conjugate, **222**, used: 223

set_constant_symbol, **222**, used: 223

set_external_flavors, **222**, used: 223

set_fermion, **222**, used: 223

set_flavors, **222**, used: 223
set_flavor_of_string, **222**, used: 223
set_flavor_symbol, **222**, used: 223
set_flavor_to_string, **222**, used: 223
set_flavor_to_TeX, **222**, used: 223
set_fuse, **222**, used: 223
set_fuse2, **222**, used: 223
set_fuse3, **222**, used: 223
set_gauge_symbol, **222**, used: 223
set_goldstone, **222**, used: 223
set_lorentz, **222**, used: 223
set_mass_symbol, **222**, used: 223
set_max_degree, **222**, used: 223
set_of_flavors, **700**, used: 700
set_of_list, **785**, used: 785
set_parameters, **222**, used: 223
set_pdg, **222**, used: 223
set_propagator, **222**, used: 223
set_to_string, **784**, used: 784
set_vertices, **222**, used: 223
set_width, **222**, used: 223
set_width_symbol, **222**, used: 223
sff (type), **446**, **497**, **527**, used: 450, 450, 501, 532
sfm (type), **446**, **497**, **528**, used: 447, 450, 498, 501, 529, 532
SG, **718**, used: 719, 721
shift_left_pred, **214**, used: 215
shift_left_pred′, **214**, used: 214
shiggs (type), **497**, **528**, used: 498, 501, 529, 532
SHiggs, **498**, **529**, used: 498, 500, 505, 506, 506, 508, 508, 509, 509, 509, 510, 512, 512, 513, 514, 529, 531, 536, 537, 537, 541, 541, 543, 543, 544, 544, 544, 546, 547, 547, 549
shiggs_neutr, **506**, **537**, used: 506, 538
shortest, **769**, **771**, **772**, **774**, **767**, **769**, used:
shortest′, **771**, **774**, used: 771, 774
shuffle, **798**, used: 800, 800, 800, 800
sign, **94**, **99**, **99**, **100**, **103**, **121**, **89**, used: 103, 121
sign (field), **99**, **99**, used: 99, 108, 112
sign (type), **99**, **99**, used: 99, 99
Simplest (module), **319**, **272**, used: ??, ??
Sin, **450**, **166**, used:
Sin2al, **450**, used:
Sin2am2b, **450**, used:
Sin2be, **450**, used:
Sin2thw, **241**, **450**, ??, ??, **278**, **300**, **321**, **337**, **349**, **367**, **378**, **388**, **414**, **428**, ??, ??, used: 243, 244, 245, ??, ??, 280, 302, 322, 337, 350, 367, 378, 389, 416, 428, ??
Sin4al, **450**, used:
Sin4be, **450**, used:
Sinamb, **450**, used:
Sinapb, **450**, used:
Singlet, **81**, **80**, used: 81, 572, 574, 225, 226, 228, 238, 448, ??, ??, 499, 530, 275, 298, 320, 334, 345, 364, 375, 385, 399, 412, 428, ??, ??, 719
singleton, **755**, **756**, **761**, **828**, **831**, **54**, **55**, **62**, **754**, **825**, **51**, used: 828, 829, 831, 214, 101, 112
Singular (exn), **833**, **833**, used:
Sinpsi, **278**, **300**, **321**, used:
Sinthw, **241**, ??, ??, **278**, **300**, **321**, **337**, **349**, **367**, **378**, **388**, **414**, **428**, ??, ??, used: 244, ??, ??, 428
size, **41**, **47**, **38**, used:
Skip (exn), ??, used:
skip_if, ??, ??, used: ??
SL, **446**, **497**, **527**, **363**, **137**, used: 257, 257, 258, 258, 258, 456, 457, 457, 459, 461, 461, 462, 465, 469, 470, 471, 473, ??, ??, ??, ??, ??, ??, 505, 505, 506, 507, 510, 510, 513, 513,

536, 537, 538, 538, 544, 545, 547, 547, 282, 304, 325, 363, 365, 369, 370, 436
Slepton, **447**, **498**, **529**, used: 447, 449, 455, 456, 456, 459, 461, 462, 465, 469, 469, 470, 471, 471, 472, 473, 474, 474, 474, 475, 476, 479, 498, 500, 503, 504, 505, 506, 507, 510, 510, 512, 513, 513, 514, 529, 531, 534, 535, 536, 538, 538, 539, 539, 540, 544, 545, 547, 547, 547, 549
slepton2_squark2, **474**, used: 478
slepton2_squark2′, **474**, used: 474
slepton2_squark2″, **474**, used: 474
slepton4_1gen, **474**, used: 478
slepton4_1gen′, **474**, used: 474
slepton4_2gen, **474**, used: 478
slepton4_2gen′, **474**, used: 474
slep_sneu_squark2, **475**, used: 478
slep_sneu_squark2′, **475**, used: 475
slep_sneu_squark2″, **475**, used: 475
slist, **529**, used: 529
slow_binomial, **790**, used: 790
SLR, **137**, used: 257, 457, 458, 458, 458, 458, 458, 459, 461, 461, 462, 462, 462, ??, ??, 505, 506, 506, 506, 506, 506, 507, 507, 507, 537, 537, 537, 537, 538, 538, 538, 539, 539, 539, 540, 540, 540, 540, 540, 540, 540, 540, 540, 540, 541, 541, 541, 541, 541, 282, 283, 304, 304, 436
SLRV, **137**, used: 476, 476, 476
SLV, **137**, used: 476, 476
SM (module), **236**, **266**, **224**, used: 266, 266, 266, 266, 266, 267, 272, 718, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??
Small_Rational (module), **826**, **823**, used: 181, 240, 453, ??, ??, 500, 531, 277, 300, 321, 335, 348, 366, 377, 387, 400, 413, 428, ??, ??, 181, 181
SM_anomalous (module), **234**, **224**, used: ??
SM_anomalous_ckm (module), **235**, **224**, used: ??
SM_anomalous_top (module), **235**, **224**, used: ??
SM_clones (module), **272**, **224**, used:
SM_flags (sig), **234**, ??, ??, **224**, ??, used: 234, 234, 234, 235, 235, 235, 235, 236, 236, ??, ??, ??, ??, ??, 224, 224, ??, ??
SM_gluons (module), **??**, used:
SM_Higgs (module), **235**, **224**, used: ??, ??
SM_k_matrix (module), **235**, **224**, used: ??
SM_no_anomalous (module), **234**, ??, ??, **224**, ??, used: 266, 272, 718, ??, ??, ??, ??
SM_no_anomalous_ckm (module), **234**, **224**, used: ??, ??
SM_Rxi (module), **266**, **224**, used:
SM_tt_threshold (module), **236**, **224**, used: ??
SN, **446**, **497**, **527**, used: 456, 459, 461, 465, 469, 471, 472, 505, 506, 507, 511, 512, 536, 538, 538, 545, 547
sneu2_slep2_1, **474**, used: 478
sneu2_slep2_1′, **474**, used: 474
sneu2_slep2_2, **474**, used: 478
sneu2_slep2_2′, **474**, used: 474
sneu2_squark2, **474**, used: 478
sneu2_squark2′, **474**, used: 474
Sneutrino, **447**, **498**, **529**, used: 447, 449, 456, 456, 459, 461, 462, 465, 469, 470,

471, 472, 474, 474, 474, 475, 476, 479, 498, 500, 504, 505, 506, 507, 510, 511, 512, 514, 529, 531, 535, 536, 538, 538, 540, 540, 541, 545, 545, 547, 549
sneutrino4, **474**, used: 478
sneutrino4', **474**, used: 474
solve, **837**, **833**, used:
solve_destructive, **836**, used: 837
solve_many, **837**, **833**, used: 819
solve_many_destructive, **836**, used: 837
sort, **809**, **692**, **695**, **806**, **691**, used: 746, 747, 747, 797, 797, 797, 820, 820, 830, 832, 793, 793, 809, 55, 694, 696, 700, 701, 110, 130, 133, 709, 714
sort', **809**, used: 809, 809
sort3, **696**, used: 696
sorted_keys, **44**, used:
sort_2i, **811**, used: 812, 816
sort_2i', **811**, used: 811, 811
sort_children, **820**, used: 820
sort_decay, **692**, **693**, **691**, used: 693, 695
sort_momenta, **694**, used: 694, 696
sort_section, **746**, used: 746
sort_signed, **796**, **789**, used: 96, 127
sort_spin_table, **130**, used: 131
source, **727**, **726**, used: 728
source (field), **727**, **726**, used: 728, 802, 9, 20, 40, 54, 218, 698, 568, 183, 93, 224, 444, ??, 495, 526, 273, ??, 678, ??, 718
SP, **137**, used: 459, 459, 537, 417, 418, 418, 418, 418, 418, 418, 419
spacelike, **54**, **59**, **60**, **64**, **64**, **52**, **53**, used: 696, 699
species, **811**, used: 811, 812, 812
spinor, **237**, **448**, ??, ??, **498**, **530**, **275**, **298**, **319**, **333**, **345**, **364**, **375**, **385**, **411**, **428**, ??, ??, used: 238, 448, ??, ??, 499, 530, 275, 298, 319, 334, 345, 364, 375, 385, 411, 428, ??, ??
Spinor, **135**, used: 570, 574, 228, 230, 237, 448, ??, ??, 498, 530, 275, 275, 298, 298, 319, 333, 345, 364, 375, 385, 399, 411, 428, ??, ??, 719
split, **757**, **735**, **792**, **9**, **11**, **12**, **15**, **16**, **54**, **60**, **65**, **735**, **788**, **6**, **52**, used: 757, 759, 760, 735, 832, 793, 795, 818, 16, 34, 700, 96, 108
split', **735**, **791**, used: 735, 735, 791, 791, 792
splitn, **742**, **740**, used: 206
splitn', **742**, used: 742, 742
split_color_string, **186**, used: 186
SPM, **137**, used: 255, 434
SQ, **718**, used: 719, 720, 720, 721, 721
Sqrt, **166**, used: 244, ??, ??, 428
square, **85**, used: 86
square (type), **84**, used:
Square, **84**, used: 85, 85
square2, **85**, used: 86
squark_current, **720**, used: 721
squark_seagull, **721**, used: 721
SR, **137**, used: 257, 257, 258, 258, 258, 457, 457, 459, 461, 461, ??, ??, ??, ??, ??, ??, 505, 537, 282, 304, 325, 436
SRV, **137**, used:
SSC (module), **428**, **273**, used: ??
SSC_flags (sig), **427**, **273**, used: 428, 428, 273, 273
SSC_kmatrix (module), **428**, **273**, used: ??
Stale, **737**, **738**, **736**, used: 739
standard_gauge_higgs, **251**, **287**, **308**, **326**, **339**, **352**, **369**, **428**, used: 253, 290, 311, 327, 339, 352, 369, 430
standard_gauge_higgs', **285**, **306**, **326**, used: 287, 308, 326

standard_gauge_higgs4, **251**, **287**, **308**, **326**, **339**, **352**, **369**, **429**, used: 253, 290, 311, 327, 339, 352, 370, 430
standard_higgs, **251**, **290**, **311**, **326**, **339**, **352**, **369**, **429**, used: 253, 290, 312, 327, 339, 352, 370, 431
standard_higgs4, **251**, **290**, **311**, **326**, **339**, **352**, **369**, **429**, used: 253, 290, 312, 327, 340, 352, 370, 431
standard_quartic_gauge, **248**, ??, ??, **284**, **305**, **326**, **338**, **351**, **369**, **428**, used: 251, ??, ??, 285, 306, 326, 339, 352, 369, 428
standard_triple_gauge, **246**, ??, ??, **283**, **305**, **325**, **338**, **351**, **368**, **428**, used: 248, ??, ??, 284, 305, 326, 338, 351, 369, 428
stat, **95**, **95**, **99**, **125**, used: 99, 112
stat (type), **95**, **95**, **99**, **125**, used: 95, 99, 108, 110, 111, 111, 111
Stat (sig), **95**, used: 95, 95, 125
state (type), **732**, used: 732
Stat_Dirac (module), **95**, used: 125, 127, 127, 127
stat_fuse, **95**, **96**, **99**, **126**, used: 99, 108, 112
stat_keystone, **112**, used: 112
Stat_Majorana (module), **125**, used: 127, 127, 127, 127
Stat_Maker (sig), **95**, used: 98
Stat_Map (module), **111**, used: 111
stat_sign, **95**, **96**, **99**, **127**, used: 99, 108, 112
step (field), **732**, used: 733, 734, 734, 734
steps (field), **732**, used: 734, 734
string_of_angle, **450**, used: 487
string_of_char, **446**, **497**, **528**, used: 481, 487, 518, 522, 554, 560, 561
string_of_fermion_gen, **498**, **529**, used: 522, 560
string_of_fermion_type, **498**, **529**, used: 522, 560
string_of_gen, **446**, **497**, **527**, used: 446, 497, 527
string_of_higgs, **447**, used: 447, 487
string_of_int_list, **723**, used: 723
string_of_neu, **446**, **497**, **528**, used: 481, 487, 518, 522, 551, 554, 560, 561
string_of_node, ??, ??, used: ??
string_of_path, ??, ??, used: ??, ??, ??
string_of_phiggs, **498**, **528**, used: 518, 522, 554, 560, 561
string_of_sff, **446**, **497**, **527**, used: 450, 487, 522, 560, 561
string_of_sfm, **446**, **497**, **528**, used: 487, 522, 560, 561
string_of_shiggs, **498**, **528**, used: 518, 522, 554, 560, 561
strip_before_a_keyword, **727**, used: 728
strip_before_keyword, **727**, used:
strip_dollars, **727**, used: 727, 727
strip_from_first, **752**, **751**, used: 727
strip_from_last, **752**, **751**, used: 727
strip_keyword, **727**, used: 727, 727
strip_prefix, **751**, **751**, used: 727, 727
strip_prefix_star, **752**, **751**, used: 727
strip_required_prefix, **752**, **751**, used: 727
strip_svn_repos, **728**, used: 728
sty, **813**, **806**, used:
style, **811**, used: 811
style (field), **811**, **806**, used: 811, 813, 813, 709
SU, **446**, **497**, **527**, used: 456, 459, 461, 465, 466, 469, 470, 471, 473, 474, 474, 505, 506, 507, 507, 510,

510, 511, 513, 513, 536, 538, 538, 539, 544, 545, 546, 547, 547
su0, **184**, used: 204, 205
sub, **826**, **826**, **827**, **828**, **830**, **831**, **54**, **57**, **62**, **823**, **824**, **825**, **51**, used: ??, ??, 751, 752, 752, 752, 752, 753, 727, 727, 830, 830, 831, 818, 57, 62, 694, 696, 568, ??, 186, 211, 268, 721, 70
sub′, **57**, used: 57, 57
substitute_char, **570**, used: 570
substring, **568**, used: 569, 569, 569
subtract, **565**, **564**, used: ??, 567
succ, **27**, **27**, **19**, used: 743, 744, 744, 745, 749, 751, 752, 752, 753, 798, 732, 734, 772, 774, 835, 835, 835, 836, 802, 803, 812, 814, 815, 816, 817, 16, 21, 24, 27, 30, 31, 32, 33, 47, 56, 59, 62, 66, 82, 84, 699, 568, 569, 569, 74, 186, 204, 211, 211, 212, 214, 110, 131, 133, 268, 496, 527
suite, **765**, **766**, **799**, **800**, **765**, **796**, used:
suite_apply, **766**, **800**, used: 766, 800
suite_compose, **800**, used: 800
suite_inverse, **800**, used: 800
suite_of_list, **799**, used: 800
sum, **827**, **828**, **830**, **831**, **818**, **181**, **181**, **181**, **181**, **181**, **182**, **825**, **825**, **180**, used: 818, 211, 122
Sum, **564**, **166**, **564**, used: 565, 565, 565
summary, **728**, **734**, **726**, **732**, used: 133, 712
SUM_1, **450**, used:
SUN, **81**, **80**, used: 81, 574, 230, 238, 448, ??, ??, 499, 530, 275, 298, 320, 334, 345, 364, 375, 385, 399, 412, 428, ??, ??, 719
sup (field), **809**, **810**, used: 809, 810, 811
Sup, **810**, **447**, **498**, **529**, used: 811, 447, 449, 455, 456, 456, 459, 461, 462, 462, 465, 466, 466, 466, 467, 469, 470, 470, 470, 471, 472, 473, 473, 474, 474, 475, 475, 475, 476, 479, 498, 500, 503, 504, 505, 506, 507, 507, 508, 510, 510, 511, 511, 511, 512, 513, 513, 513, 514, 529, 531, 534, 536, 536, 538, 538, 539, 540, 540, 540, 542, 544, 545, 545, 545, 546, 546, 547, 547, 547, 549
sup2_sdown2, **475**, used: 478
sup2_sdown2_1, **475**, used: 475
sup2_sdown2_2, **475**, used: 475
sup2_sdown2_3, **475**, used: 475
sup4_1gen, **475**, used: 478
sup4_1gen′, **475**, used: 475
sup4_2gen, **475**, used: 478
sup4_2gen′, **475**, used: 475
Supp, **278**, **300**, **321**, used:
Supp2, **278**, **300**, **321**, used:
supremum_or_infinity (type), **810**, used: 810
SV, **137**, used:
swap, **834**, used: 835, 835
SYM (module), **718**, used: 723
symbol, **564**, **564**, used: ??, 567
symbol (type), **571**, used: 571
symbol (camllex regexpr), **566**, used: 566
Symbol, **564**, **564**, used: 564
SYMBOL, ??, ??, used: ??, 566
SYMBOL (camlyacc token), **567**, used: 567
symmetry, **790**, **32**, **94**, **101**, **103**, **122**, **787**, **90**, used: 32, 117, 122
symmetry (field), **100**, **103**, used: 103, 116, 121
Syntax_Error (exn), **70**, **68**, used:
system_cache_dir, **729**, **729**, used: 737

s_channel, **54**, **59**, **64**, **94**, **123**, **124**, **53**, **91**, used: 699, 702
s_channel_in, **54**, **59**, **64**, **52**, used: 59, 59, 64, 64
s_channel_out, **54**, **59**, **64**, **52**, used: 59, 64
s_channel_out′, **59**, used: 59, 59
S_NNA, **450**, used:
S_NNG, **450**, used:
S_NNH1, **450**, used:
S_NNH2, **450**, used:
t (type), **??**, **??**, **??**, **??**, **765**, **765**, **765**, **755**, **755**, **761**, **746**, **796**, **797**, **797**, **781**, **781**, **781**, **782**, **782**, **784**, **784**, **784**, **727**, **737**, **737**, **732**, **769**, **769**, **770**, **772**, **773**, **820**, **826**, **826**, **827**, **827**, **828**, **828**, **829**, **831**, **730**, **730**, **808**, **9**, **10**, **12**, **13**, **16**, **27**, **27**, **28**, **40**, **41**, **41**, **42**, **42**, **43**, **43**, **43**, **43**, **44**, **45**, **48**, **54**, **55**, **61**, **66**, **66**, **66**, **692**, **695**, **181**, **181**, **181**, **181**, **181**, **182**, **81**, **81**, **81**, **82**, **83**, **83**, **84**, **85**, **87**, **218**, **218**, **219**, **219**, **219**, **698**, **699**, **699**, **700**, **702**, **69**, **73**, **74**, **78**, **186**, **210**, **210**, **214**, **215**, **215**, **215**, **99**, **99**, **103**, **103**, **104**, **105**, **109**, **111**, **116**, **119**, **119**, **123**, **124**, **129**, **129**, **131**, **140**, **??**, **??**, **764**, **754**, **796**, **780**, **780**, **783**, **783**, **726**, **736**, **736**, **732**, **767**, **768**, **768**, **820**, **823**, **824**, **824**, **825**, **730**, **805**, **6**, **19**, **35**, **36**, **37**, **40**, **50**, **53**, **691**, **180**, **80**, **80**, **217**, **698**, **68**, **72**, **209**, **71**, **71**, **567**, **567**, used: **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, 765, 765, 765, 755, 755, 796, 781, 781, 782, 782, 784, 784, 784, 784, 784, 737, 769, 769, 770, 772, 773, 820, 826, 826, 827, 827, 828, 828, 829, 831, 730, 808, 813, 9, 10, 12, 13, 15, 20, 22, 24, 26, 26, 27, 28, 29, 33, 33, 40, 41, 41, 42, 42, 42, 42, 42, 43, 43, 44, 44, 44, 44, 45, 45, 48, 48, 49, 54, 55, 61, 66, 66, 66, 692, 693, 181, 181, 181, 181, 181, 182, 81, 81, 82, 82, 82, 82, 83, 218, 219, 698, 699, 699, 699, 700, 571, 572, 69, **??**, 73, 73, 74, 74, 74, 210, 213, 215, 216, 94, 95, 95, 97, 97, 97, 97, 98, 98, 99, 99, 100, 100, 101, 101, 101, 103, 103, 104, 105, 109, 110, 111, 111, 111, 118, 127, 129, 129, 129, 131, 707, 717, 172, 172, 174, 175, 175, 175, 178, 178, 178, 178, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, 764, 765, 754, 796, 780, 780, 780, 780, 780, 780, 783, 783, 783, 783, 783, 784, 784, 784, 726, 726, 726, 726, 726, 736, 732, 767, 767, 767, 767, 767, 767, 768, 768, 768, 768, 768, 769, 769, 769, 820, 820, 823, 823, 824, 824, 824, 824, 824, 825, 825, 825, 825, 825, 825, 825, 825, 825, 825, 730, 802, 805, 805, 806, 806, 806, 806, 806, 806, 806, 807, 807, 6, 6, 6, 6, 6, 6, 6, 6, 7, 7, 7, 7, 7, 8, 8, 8, 8, 19, 19, 19, 20, 20, 20, 35, 36, 36, 36, 36, 36, 37, 37, 37, 37, 37, 37, 37, 37, 38, 38, 38, 38, 38, 38, 39, 39, 39, 39, 39, 39, 40, 40, 40, 40, 50, 51, 51, 51, 51, 51, 51, 51, 51, 51, 52, 52, 52, 52, 52, 52, 52, 52, 52, 53, 53, 53, 53, 53, 53, 53, 53, 53, 691, 691, 691, 691, 691, 180, 180, 180, 180,

180, 181, 181, 80, 80, 80, 81, 217, 698, 698, 68, 68, 68, ??, 72, 72, 73, 209, 209, 209, 210, 210, 210, 88, 89, 89, 91, 91, 91, 91, 92, 93, 93, 707, 71

T, **230**, used: 230, 230, 231, 232

T (module), **708**, used: 712, 713, 715, 716

T (sig), **765**, **755**, **796**, **782**, **784**, **737**, **769**, **20**, **41**, **54**, **692**, **181**, **698**, **73**, **210**, **94**, **707**, **172**, **178**, **764**, **754**, **796**, **780**, **783**, **736**, **767**, **18**, **37**, **50**, **691**, **180**, **698**, **72**, **209**, **88**, **707**, used: 765, 797, 797, 799, 770, 773, 48, 693, 181, 181, 181, 181, 182, 699, 74, 183, 210, 95, 95, 95, 97, 98, 125, 128, 128, 708, 172, 175, 176, 178, 179, 765, 755, 755, 796, 796, 796, 780, 784, 736, 768, 769, 19, 20, 20, 39, 40, 53, 691, 180, 180, 180, 181, 181, 698, 568, 73, 183, 210, 91, 93, 93, 707, 224, 224, 444, 495, 525, 272, 272

t′ (type), **215**, used: 215

T2, **13**, **8**, used: 13, 14, 14, 15, 15, 15

T3, **13**, **8**, used: 13, 14, 14, 15, 15, 15

table, **225**, **229**, **232**, **259**, **267**, **270**, **479**, ??, ??, **514**, **549**, **291**, **312**, **327**, **340**, **353**, **370**, **381**, **391**, **406**, **422**, **437**, ??, ??, **721**, used: 225, 229, 232, 259, 267, 270, 479, ??, ??, 514, 549, 291, 312, 327, 340, 353, 370, 381, 391, 406, 422, 437, ??, ??, 721

tag (field), **737**, used: 738, 738

tagged (type), **737**, used:

Tagged (module), **98**, used: 125, 125

Tagged_Binary (module), **125**, **93**, used:

Tagged_Coupling (module), **99**, used: 100, 103, 108, 112, 119

Tagged_Coupling (sig), **99**, used: 99

Tagged_Maker (sig), **97**, **93**, used: 93

Tagger (sig), **97**, **93**, used: 97, 98, 125, 93

Tags (module), **99**, **100**, **101**, used: 99, 99, 100, 100, 101, 101, 101, 101, 101, 108, 108, 112, 112

Tags (sig), **97**, **93**, used: 97, 100, 93

tail_opt, **205**, used: 205

Tan, **166**, used:

Tana, **450**, used:

Tanb, **450**, used:

Target (module), **178**, used: 708, 707, 578, 578, 578

Targets (module), **578**, **578**, used: 718, ??, ??, ??, 718, ??, 723

Targets_Kmatrix (module), **678**, **678**, used:

Tau, **227**, used: 228, 228, 229, 229

Tbar, **230**, used: 230, 230, 231, 232

TBWA, **236**, ??, ??, **428**, used: 237, 255, 259, ??, ??, ??, ??, ??, ??, 428, 434, 437

TBWZ, **236**, ??, ??, **428**, used: 237, 256, 259, ??, ??, ??, ??, ??, ??, 428, 434, 437

Template (module), **374**, **272**, used: ??

tension (field), **811**, **816**, **806**, used: 813, 816, 709

tension_to_equation, **818**, used: 819

Tensor_1, **135**, used: 237, **??**, **??**, 428
Tensor_2, **135**, used: **??**, 334, 345, 428
Tensor_2_Vector_Vector, **138**, used: 188
Tensor_2_Vector_Vector_1, **138**, used: 188, **??**, **??**, 432, 432
Tensor_2_Vector_Vector_t, **138**, used: 188, **??**, 432
Term (module), **828**, **825**, used:
Term (sig), **827**, **824**, used: 828, 829, 825, 825
Ternary (module), **12**, **26**, **8**, **19**, used: 15, 26, 19
Ternary (sig), **12**, **8**, used: 8
test (type), **??**, **??**, used: **??**, 765, 799, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, 765, 796
Test (module), **765**, **799**, **765**, **796**, used:
TestCase, **??**, **??**, used: **??**, **??**, **??**
TestLabel, **??**, **??**, used: **??**, **??**, **??**
TestList, **??**, **??**, used: **??**, **??**, **??**
test_case_count, **??**, **??**, used: **??**, **??**, **??**
test_case_paths, **??**, **??**, used: **??**
test_decorate, **??**, **??**, used: **??**
test_event (type), **??**, **??**, used: **??**
test_filter, **??**, **??**, used: **??**
test_fun (type), **??**, **??**, used: **??**, **??**, **??**, **??**
test_result (type), **??**, **??**, used: **??**, **??**, **??**, **??**, **??**
test_rhs, **112**, used: 112
test_sum, **60**, **64**, used: 60, 65
tex_lbl, **811**, used: 811, 812, 812
tgc, **246**, **??**, **??**, **281**, **338**, **351**, **368**, **380**, **390**, **419**, **428**, **??**, **??**, used: 246, 247, **??**, **??**, 283, 283, 338, 351, 368, 380, 390, 419, 428, **??**, **??**
THDM (module), **410**, **272**, used: **??**, **??**
THDM_CKM (module), **410**, **272**, used: **??**, **??**
THDM_flags (sig), **410**, **272**, used: 410, 410, 410, 272
this (field), **813**, **807**, used: 815
ThoArray (module), **748**, **748**, used:
ThoFilename (module), **735**, **735**, used: 737
ThoList (module), **742**, **740**, used: 797, 797, 797, 799, 800, 800, 800, 800, 830, 832, 777, 790, 793, 793, 793, 794, 795, 795, 808, 808, 816, 816, 816, 816, 16, 16, 17, 17, 17, 24, 24, 33, 34, 60, 86, 219, 223, 701, 575, 74, 204, 204, 204, 204, 206, 206, 212, 213, 213, 215, 216, 216, 216, 108, 116, 117, 119, 120, 120, 120, 123, 123, 123, 131, 131, 131, 133, 133, 133, 709, 709, 712, 713, 714, 225, 226, 228, 230, 237, 237, 246, 246, 256, 257, 257, 258, 258, 258, 269, 269, 269, 270, 447, 448, 458, 458, 459, 459, 459, 461, 461, 463, 465, 466, 466, 466, 469, 470, 470, 470, 471, 472, 474, 474, 475, 475, 476, 477, 478, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, 498, 498, 507, 509, 510, 511, 511, 511, 512, 512, 513, 513, 514, 529, 529, 538, 539, 540, 540, 540, 540, 540, 541, 541, 541, 542, 542, 542, 543, 544, 545, 545, 545, 546, 546, 547, 548, 549, 275, 275, 291, 298, 298, 312, 319, 319, 323, 325, 327, 333, 333, 340, 344, 345, 353, 363, 364, 370, 375, 375, 380, 385, 385, 391, 398, 411, 411, 422, 428, 435, 435, 436, 436, **??**, **??**,

??, ??, ??, ??, ??, ??, ??, 718, 719
ThoString (module), **751**, **751**, used: 727, 714
thread, **777**, **776**, used:
thread_unfinished_decays, **696**, used: 697
thread_unfinished_decays′, **696**, used: 696, 696, 697
three, **28**, used: 29, 29, 30, 30, 31
Threeshl (module), **396**, **272**, used: ??, ??
Threeshl_ckm (module), **395**, **272**, used:
Threeshl_ckm_no_hf (module), **395**, **272**, used:
Threeshl_diet (module), **396**, **272**, used:
Threeshl_diet_no_hf (module), **396**, **272**, used:
Threeshl_no_ckm (module), **395**, **272**, used: ??
Threeshl_no_ckm_no_hf (module), **395**, **272**, used: ??
Threeshl_options (sig), **395**, **272**, used: 395, 395, 395, 395, 396, 396, 396, 272
three_gluon, **231**, **720**, used: 232, 721
Thw, **450**, used:
Th_SF, **450**, used:
time, **766**, **799**, **799**, **801**, **796**, used: 799, 799, 733, 734, 734, 734, 734
timelike, **54**, **59**, **59**, **64**, **64**, **52**, **53**, used: 59, 60, 64, 64, 699, 702, 107, 116, 123, 124
Timelike, **137**, used: 225, 226, 228, 230, 236, 445, ??, ??, 496, 526, 274, 297, 319, 333, 344, 363, 374, 384, 396, 410, 428, ??, ??, 720
timelike_map, **699**, used: 699, 700
time_fun, ??, used: ??
time_to_string, **733**, used: 734, 734
tln, **742**, **740**, used: 742
TLR, **137**, used: 255, 256, ??, ??, ??, ??, 434, 434
TLRM, **137**, used: 255, ??, ??, 433
todo, ??, ??, used:
Todo (exn), ??, used:
token, ??, ??, ??, ??, used: ??, 573, ??, ??, 75, 70, 566
token (type), ??, ??, ??, ??, used: ??, ??, ??, ??
token (camllex parsing rule), **70**, **566**, used:
TopH, **274**, used: 275, 277, 281, 282, 283, 290, 291
TopHb, **274**, used: 275, 277, 281, 282, 283, 290, 291
Topology (module), **20**, **18**, used: 98, 125, 127, 127, 127, 127
Topp, **297**, used: 298, 299, 303, 304, 304, 312, 312
Toppb, **297**, used: 298, 299, 303, 304, 304, 312, 312
top_anom, **234**, **234**, **234**, **234**, **235**, **235**, **235**, **235**, **236**, ??, ??, **427**, **428**, **224**, ??, **273**, used: 253, 254, 254, 254, 254, 255, 255, 255, 255, 256, 256, 256, ??, 432, 433, 433, 433, 433, 433, 434, 434, 434, 434, 435, 435
top_anom_4f, **234**, **234**, **234**, **234**, **235**, **235**, **235**, **235**, **236**, ??, ??, **427**, **428**, **224**, ??, **273**, used: 256, 257, 257, 257, 258, 258, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, 435, 435, 436, 436
top_quartic, **290**, **312**, **327**, used: 291, 312
tower_of_dag, **110**, used:
tower_to_dot, **94**, **125**, **91**, used: 716
to_feynmf_channel, **812**, used: 814, 815
to_float, **826**, **826**, **823**, used:

to_gauss, **78**, used: 79
to_int, **27**, **27**, **19**, used: 32
to_ints, **54**, **55**, **62**, **51**, used: 62, 66, 697, 701, 75, 78, 101, 124
to_ints', **62**, used: 62, 62
to_list, **9**, **11**, **13**, **15**, **17**, **84**, **7**, used: 746, 797, 797, 797, 798, 798, 818, 819, 84, 84, 84, 103, 104, 107, 108, 108, 108, 119, 123, 123, 708, 714
to_lists, **784**, **785**, **83**, **84**, **783**, **80**, used: 78, 215
to_momentum, **66**, **66**, **66**, **53**, used: 66
to_on_shell, **77**, used: 79
to_ratio, **826**, **826**, **823**, used:
to_selectors, **73**, **79**, **72**, used: 715
to_select_p, **76**, used: 79
to_select_wf, **76**, used: 79
to_string, **784**, **784**, **784**, **820**, **826**, **826**, **827**, **827**, **828**, **829**, **831**, **832**, **810**, **27**, **27**, **54**, **55**, **62**, **697**, **81**, **82**, **83**, **84**, **85**, **69**, **73**, **75**, **78**, **213**, **215**, **783**, **784**, **820**, **823**, **824**, **825**, **825**, **806**, **19**, **52**, **68**, **72**, used: ??, ??, 784, 737, 820, 831, 832, 83, 75, 79, 214, 715, 716
to_string1, **697**, used: 697, 697
translate_constant, **575**, used: 575
translate_tensor3, **575**, used: 575
translate_tensor4, **575**, used: 575
transpose, **744**, **749**, **741**, used: 749, 750
tree (type), **699**, used: 699
Tree (module), **755**, **808**, **755**, **805**, used: 749, 816, 41, 47, 94, 123, 707, 709, 709, 710, 710, 711, 711, 711, 711, 711, 712, 712, 712, 712, 716, 717, 39, 91, 707
Tree2 (module), **820**, **820**, used: 41, 46, 94, 100, 101, 103, 131, 38, 89
trees, **698**, **700**, **698**, used: 716
trees (field), **699**, used: 700, 701
Trie, **770**, **773**, used: 770, 770, 770, 771, 771, 773, 773, 773, 774, 774
Trie (module), **769**, **767**, used:
triples, **803**, **496**, **527**, **802**, used: 13, 15, 512, 546
triplet_gauge2_higgs, **286**, **307**, used: 287, 308
triplet_gauge_higgs, **285**, **307**, used: 287, 308
triple_anom, **234**, **234**, **234**, **234**, **235**, **235**, **235**, **235**, **236**, ??, ??, **427**, **428**, **224**, ??, **273**, used: 248, ??, ??, 428
triple_gauge, **248**, **463**, ??, ??, **508**, **542**, **284**, **305**, **326**, **338**, **351**, **369**, **380**, **390**, **419**, **428**, ??, ??, used: 258, 477, ??, ??, 513, 548, 291, 312, 327, 340, 353, 370, 380, 391, 422, 436, ??, ??
triple_gravitino, **463**, used: 478
triple_gravitino', **462**, used: 463
triple_gravitino'', **462**, used: 463
TRL, **137**, used: 255, 256, ??, ??, ??, ??, 434, 434
TRLM, **137**, used: 255, ??, ??, 433
True, **69**, **74**, **68**, used: 69, 69, 75, 75
try_add, **54**, **57**, **63**, **51**, used: 60, 64
try_add', **56**, used: 56, 57
try_first, **738**, used: 738
try_fusion, **54**, **60**, **65**, **52**, used: 696
try_of_momenta, **697**, used:
try_sub, **54**, **57**, **63**, **51**, used: 60, 63, 64, 696
try_sub', **57**, used: 57, 57
try_thread_unfinished_decays, **697**, used: 697
TS (module), **727**, used: 727, 727, 727, 727, 727
TTGG, **236**, ??, ??, **428**, used: 237, 255, 259, ??, ??, ??,

??, ??, ??, 428, 434, 437
TTWW, **236**, ??, ??, **428**, used: 237, 256, 259, ??, ??, ??, ??, ??, ??, 428, 435, 437
tt_threshold, **234**, **234**, **234**, **234**, **235**, **235**, **235**, **235**, **236**, **224**, used: 254, 254, 263
tt_threshold_ttA, **254**, used: 258
tt_threshold_ttZ, **254**, used: 258
Tuple (module), **9**, **24**, **33**, **6**, used: 22, 23, 24, 24, 26, 26, 26, 33, 33, 33, 33, 34, 42, 97, 97, 97, 98, 98, 101, 125, 127, 127, 127, 127, 19, 20, 20, 36, 91, 91, 93
tuples, **803**, **802**, used: 803, 803, 17, 24, 33
tuples', **803**, used: 803, 803
tuple_of_list2, **23**, used: 24
TVA, **137**, used: 255, 256, 256, ??, ??, ??, ??, ??, ??, 434, 435, 435
TVAM, **137**, used: 253, 254, 254, 254, 254, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, 432, 433, 433, 433, 433
two, **28**, used: 29, 30, 30, 30, 31, 31, 31, 32
TwoHiggsDoublet (module), **410**, **272**, used: ??, ??, ??, ??
two_adjoints_couple_to_singlets, **205**, used: 205
two_and_one, **497**, **527**, used: 506, 512, 538, 546
two_and_one', **497**, **527**, used: 497, 497, 527, 527
t_channel (field), **695**, used: 696, 696, 696, 697
U, **230**, **236**, **447**, ??, ??, **498**, **529**, **274**, **297**, **319**, **333**, **344**, **363**, **374**, **384**, **411**, **428**, ??, ??, used: 230, 230, 231, 232, 237, 239, 243, 245, 245, 246, 246, 246, 246, 253, 254, 254, 254, 254, 255, 255, 255, 255, 256, 256, 256, 256, 257, 257, 257, 257, 257, 257, 258, 258, 258, 259, 447, 449, 453, 455, 455, 456, 457, 457, 457, 458, 459, 461, 462, 462, 476, 479, ??, 498, 500, 503, 503, 504, 505, 505, 506, 507, 507, 508, 514, 529, 531, 534, 535, 535, 537, 537, 538, 538, 539, 539, 539, 539, 540, 540, 549, 275, 277, 281, 281, 281, 281, 281, 282, 282, 282, 282, 283, 290, 291, 297, 299, 302, 303, 303, 303, 303, 303, 304, 304, 304, 304, 312, 312, 319, 320, 323, 323, 323, 323, 324, 325, 325, 325, 327, 333, 335, 338, 338, 338, 338, 338, 338, 340, 344, 347, 350, 350, 350, 351, 351, 351, 353, 363, 365, 368, 368, 368, 368, 370, 375, 376, 379, 379, 379, 379, 379, 381, 385, 386, 389, 389, 389, 389, 390, 391, 411, 412, 417, 417, 417, 417, 417, 418, 418, 418, 422, 428, 432, 433, 433, 433, 434, 434, 434, 434, 435, 435, 435, 435, 435, 435, 436, 436, 437, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??
u1_gauged, **273**, **273**, **273**, **274**, **272**, used: 275, 281, 281, 282, 283, 283, 284, 285, 285, 286, 287, 291, 298, 303, 303, 304, 304, 305, 305, 306, 307, 307, 308, 312
Ubar, **230**, used: 230, 230, 231, 232
UED (module), **343**, **272**, used: ??
Unbounded_Nary (module), **17**,

8, used:
unbounded_power, **17**, used: 17
unbounded_power_fold, **17**, used: 17
uncolored_colored, **711**, used: 712
uncolored_only, **711**, used: 712
uncolorize_wf, **117**, used: 121
uncompress, **749**, **748**, used: 750
uncompress2, **750**, **748**, used:
uncons, **755**, **758**, **763**, **754**, used:
uncons_opt, **755**, **758**, **763**, **754**, used:
uncross_colored, **206**, used: 207
unfinished_decays (type), **695**, used:
unfinished_decays_of_momenta, **696**, used: 697, 697
unfold, **701**, used: 701
unfold1, **700**, used: 701
unfold_tree, **701**, used: 701
Uninitialized (exn), **222**, used:
union, **755**, **760**, **764**, **784**, **784**, **754**, **783**, used: 760, 764, 746, 784, 786, 786, 78, 214, 215
uniq, **744**, **748**, **709**, **740**, **748**, used: 830, 832, 133, 709, 709, 714
uniq (field), **748**, used: 748, 749, 749, 750
uniq′, **744**, used: 744, 744
uniq2, **749**, **748**, used:
uniq2 (field), **748**, used: 749, 749, 750
unique_final_state, **211**, used:
unique_flavors, **211**, used: 211
unit, **826**, **826**, **827**, **827**, **828**, **829**, **823**, **824**, **824**, used:
Unit, **241**, **450**, **??**, **??**, **278**, **300**, **321**, **337**, **349**, **367**, **378**, **388**, **414**, **428**, **??**, **??**, used: **??**, 576, 98, 128, 712, 713, 236, 256, 445, **??**, **??**, **??**, **??**, 496, 526, 274, 297, 319, 333, 344, 363, 374, 384, 396, 410, 428, 435, **??**, **??**
Unitarity_Gauge (sig), **177**, used:
unitialized, **222**, used: 222
unit_tension, **816**, used:
Unordered (exn), **692**, **693**, **691**, used:
unpack_map, **700**, used: 701
unphysical_of_flavor, **131**, used: 131
unphysical_of_flavors, **131**, used: 131
unphysical_of_flavors1, **131**, used: 131
unphysical_of_lorentz, **130**, used: 131
unphysical_polarization, **713**, used: 713, 715
unquote, **??**, **70**, used: **??**, 70
Uodd, **297**, used: 297, 298
update, **755**, **759**, **762**, **754**, used: 759, 759, 760, 762, 762
upper (camllex regexpr), **70**, **566**, used: 70, 566
user_cache_dir, **729**, **729**, used: 737
use_channel, **733**, used: 734, 734, 734
use_fudged_width, **236**, **445**, **??**, **??**, **496**, **526**, **274**, **297**, **319**, **333**, **344**, **363**, **374**, **384**, **410**, **428**, **??**, **??**, used: 236, 239, 445, 449, **??**, **??**, **??**, **??**, 496, 499, 526, 531, 274, 276, 297, 299, 319, 320, 333, 334, 344, 346, 363, 365, 374, 376, 384, 386, 410, 412, 428, 428, **??**, **??**, **??**, **??**
Using_Arrays (module), **797**, **796**, used: 798
Using_Lists (module), **797**, **796**, used:
U_K1_L, **344**, used: 344, 347, 353
U_K1_R, **344**, used: 344, 347, 353
U_K2_L, **344**, used: 344, 347, 353

U_K2_R, **344**, used: 344, 347, 353
v (field), **29**, used: 29, 30, 30, 30
V, **137**, used: 229, 231, 245, 245, 256, 455, 455, 457, 462, 462, ??, ??, ??, ??, 503, 503, 505, 508, 534, 535, 536, 539, 542, 542, 542, 281, 281, 281, 302, 303, 303, 323, 323, 325, 338, 338, 350, 351, 368, 369, 379, 379, 389, 389, 403, 403, 404, 417, 417, 428, 435, ??, ??, ??, ??, 720
V2, **137**, used: 476
V2LR, **137**, used: 476, 476
v3 (field), **219**, used: 219, 220, 221
V3, **140**, used: 220, 190, 191, 227
v4 (field), **219**, used: 219, 220, 221
V4, **140**, used: 221, 190, 191, 227
VA, **137**, used: 246, 455, 457, 457, ??, ??, 503, 505, 535, 536, 536, 281, 281, 281, 282, 303, 303, 303, 304, 323, 338, 350, 368, 379, 389, 417, 428, ??, ??, ??
VA2, **137**, used: 403, 403, 404, 404, 404
VA3, **137**, used:
VA3M, **137**, used:
value (field), **737**, used:
value (type), **737**, **737**, **736**, used: 737, 737, 736, 736
Value (sig), **737**, **736**, used: 737, 736
VAM, **137**, used: 254, 254
vanilla, **813**, **806**, used: 813
Vanishing, **137**, used: 236, 445, ??, ??, 496, 526, 274, 297, 319, 333, 344, 363, 374, 384, 396, 410, 428, ??, ??
Vanishing_couplings (exn), **98**, used:
vanishing_flavors, **127**, **129**, **92**, used:
vanishing_flavors (field), **129**, used: 129, 134
variable, **124**, **709**, used: 124, 716
variable (type), **171**, used: 171
variable′, **709**, used: 712, 712, 712
variables, **94**, **101**, **103**, **122**, **90**, used:
variables_file, **576**, used: 576, 576
variable_array (type), **171**, used: 171
VA_ILC_ttA, **241**, used: 254
VA_ILC_ttZ, **241**, used: 254
vc (type), **450**, **501**, **532**, used: 450, 501, 532
VCache (module), **105**, used: 105, 105, 105
vector (type), **566**, used:
Vector, **135**, used: 574, 228, 230, 237, 238, 448, ??, ??, 499, 530, 275, 298, 319, 334, 345, 364, 375, 385, 399, 411, 428, ??, ??, 719
Vector4, **139**, used: 189, 191, 232, 248, 249, 463, ??, ??, 508, 542, 284, 305, 326, 338, 351, 369, 380, 390, 405, 420, 428, ??, ??, 721
Vector4_K_Matrix_jr, **139**, used: 189, 191, 250, ??, ??, 428
Vector4_K_Matrix_tho, **139**, used: 189
Vectorspinor, **135**, used: 448, 364
vector_keyword, **566**, used:
Vector_Scalar_Scalar, **138**, used: 188, 455, 456, 456, 456, 462, 463, 464, 503, 504, 504, 505, 508, 508, 534, 535, 536, 536, 542, 542, 542, 542, 543, 286, 307, 327, 420, 720
version, **712**, used: 713
vertex3 (type), **138**, used: 218, 105, 140, 174, 175, 217
vertex4 (type), **139**, used: 218, 105, 140, 174, 175, 217
vertexn (type), **140**, used: 218, 105, 140, 174, 175, 217

vertex_table (type), **105**, used: 105
vertices, **222**, **571**, **191**, **207**, **105**, **225**, **227**, **229**, **232**, **259**, **266**, **270**, **479**, ??, ??, **514**, **549**, **291**, **312**, **327**, **340**, **353**, **370**, **381**, **391**, **406**, **422**, **437**, ??, ??, **721**, **174**, used: 571, 205, 207, 105, 117, 122, 225, 229, 232, 259, 266, 267, 270, 270, 479, ??, ??, 514, 549, 291, 312, 327, 340, 353, 370, 381, 391, 406, 422, 437, ??, ??, 721
vertices (label), **223**, **175**, used:
vertices (type), **104**, used: 104, 105, 105
vertices3, **232**, **258**, **478**, ??, ??, **513**, **548**, **291**, **312**, **327**, **340**, **353**, **370**, **380**, **391**, **422**, **436**, ??, ??, **721**, used: 232, 259, 479, ??, ??, 514, 549, 291, 312, 327, 340, 353, 370, 381, 391, 422, 437, ??, ??, 721
vertices3′, **478**, used: 478
vertices3″, **477**, used: 478
vertices4, **232**, **259**, **478**, ??, ??, **514**, **549**, **291**, **312**, **327**, **340**, **353**, **370**, **381**, **391**, **422**, **437**, ??, ??, **721**, used: 232, 259, 479, ??, ??, 514, 549, 291, 312, 327, 340, 353, 370, 381, 391, 422, 437, ??, ??, 721
vertices4′, **478**, used: 478
vertices4″, **478**, used: 478
vertices4‴, **478**, used: 478
vertices_aaww, **405**, used: 406
vertices_all, **403**, used: 406
vertices_aqq, **403**, used: 406
vertices_aww, **404**, used: 406
vertices_cache, **105**, used: 105
vertices_ggg, **405**, used: 406
vertices_gggg, **406**, used: 406
vertices_gqq, **404**, used: 406
vertices_nocache, **104**, used: 105, 105
vertices_wll, **403**, used: 403, 406
vertices_wll_diet, **403**, used: 406
vertices_wqq, **404**, used: 406
vertices_wqq_no_ckm, **403**, used: 404, 406
vertices_wqq_no_ckm_diet, **404**, used: 406
vertices_wwww, **405**, used: 406
vertices_wwza, **405**, used: 406
vertices_wwzz, **405**, used: 406
vertices_zll, **404**, used: 406
vertices_zqq, **404**, used: 406
vertices_zww, **405**, used: 406
Vev, **241**, **450**, ??, ??, **278**, **300**, **321**, **337**, **349**, **367**, **378**, **388**, **414**, **428**, ??, ??, used: 244, ??, ??, 428
VHeavy, **278**, **300**, **321**, used:
VL, **137**, used: 246, 246, 246, 257, 257, 257, 456, 456, ??, ??, ??, ??, ??, ??, ??, ??, 504, 504, 535, 535, 281, 281, 282, 282, 303, 303, 303, 304, 323, 325, 338, 350, 368, 379, 389, 417, 428, 435, 435, 436, ??, ??
VLR, **137**, used: 256, 257, 257, 456, 462, ??, ??, ??, ??, ??, ??, 504, 505, 535, 324, 435, 435, 435
VLRM, **137**, used: 254, 255, ??, ??, ??, ??, 433, 433
VM (module), **578**, used: ??, ??, ??, ??, ??, ??, ??, ??, ??
VMOM, **137**, used:
vn (field), **219**, used: 219, 220, 221
Vn, **140**, used: 190, 191
VR, **137**, used: 281, 303
VSet (module), **104**, used: 104, 104
V_0, **450**, used:
V_CKM, **450**, used:
V_P, **450**, used:
w, **397**, used: 398
W, **450**, **397**, used: 397, 400, 403, 403, 404, 404, 405, 405, 405, 405, 405, 407
W (module), **708**, used: 716

was_successful, **??**, used: **??**, **??**, **??**
wavefunctions, **100**, **103**, used: 117, 120
wf (type), **94**, **97**, **97**, **97**, **98**, **100**, **101**, **121**, **127**, **128**, **88**, **91**, **93**, used: 94, 97, 100, 100, 100, 100, 100, 100, 100, 100, 101, 101, 103, 103, 103, 103, 103, 103, 108, 109, 110, 111, 111, 111, 111, 116, 118, 119, 121, 123, 127, 128, 128, 129, 131, 88, 88, 88, 88, 89, 89, 89, 89, 90, 90, 91, 92, 92, 93, 93
wf1, **709**, used: 709
WFMap (module), **116**, **131**, used: 116, 117, 117, 131
WFMap2 (module), **131**, used: 131
WFSet (module), **103**, **123**, used: 103, 123, 124
WFSet2 (module), **131**, used: 132, 132
WFTSet (module), **131**, used: 131
wf_tag, **94**, **100**, **101**, **121**, **88**, used: 121
wf_tag (field), **100**, **101**, used: 101, 101, 102, 108, 108, 112, 117, 117, 118, 123
wf_tag_raw, **100**, **101**, used: 108
wf_to_string, **97**, **97**, **97**, **98**, **93**, used: 101
wf_to_TeX, **712**, used: 712, 712, 712
white (camllex regexpr), **70**, used: 70
White, **184**, used: 185, 186, 186, 192, 192, 192, 192, 195, 195, 195, 196, 196, 196, 203, 205
Whizard (module), **698**, **698**, used: 708
Whizard (sig), **66**, **53**, used: 699, 53, 698
whizard_tree, **701**, used:
whizard_tree_debug, **701**, used:
WHm, **274**, **297**, used: 275, 277, 281, 282, 283, 284, 285, 285, 286, 287, 291, 298, 299, 303, 303, 305, 305, 306, 307, 307, 308, 312
WHp, **274**, **297**, used: 275, 277, 281, 282, 283, 284, 285, 285, 286, 287, 291, 298, 299, 303, 303, 305, 305, 306, 307, 307, 308, 312
width, **222**, **571**, **185**, **207**, **225**, **226**, **228**, **230**, **239**, **266**, **267**, **449**, **??**, **??**, **499**, **531**, **276**, **299**, **320**, **334**, **346**, **365**, **376**, **386**, **400**, **412**, **428**, **??**, **??**, **720**, **174**, used: 571, 185, 207, 266, 267
width (label), **223**, **175**, used: 575
width (type), **137**, used: 174, 175
Width, **241**, **??**, **??**, **278**, **300**, **321**, **337**, **349**, **367**, **378**, **388**, **414**, **428**, **??**, **??**, used:
width_symbol, **222**, **571**, **188**, **207**, **226**, **227**, **229**, **233**, **264**, **267**, **267**, **487**, **??**, **??**, **522**, **560**, **294**, **316**, **331**, **343**, **362**, **373**, **384**, **394**, **409**, **425**, **441**, **??**, **??**, **723**, **175**, used: 571, 188, 207, 267, 267
width_symbol (label), **223**, **175**, used: 575
with_infinity, **810**, used: 811
with_supremum (type), **809**, used:
with_supremum_or_infinity (type), **810**, used:
Wm, **236**, **447**, **??**, **??**, **498**, **529**, **274**, **297**, **319**, **333**, **344**, **363**, **374**, **384**, **411**, **428**, **??**, **??**, used: 237, 239, 246, 246, 246, 246, 247, 248, 249, 250, 251, 251, 252, 253, 255, 255, 256, 256, 256, 259, 447, 449, 456, 456, 456, 456, 456, 462, 463, 463, 463, 464,

464, 465, 465, 466, 466, 476, 476, 476, 476, 479, ??, 498, 500, 504, 504, 504, 504, 504, 508, 508, 508, 509, 509, 509, 509, 510, 510, 510, 510, 510, 511, 511, 511, 514, 529, 531, 535, 535, 535, 535, 536, 542, 542, 543, 544, 544, 544, 544, 544, 544, 544, 544, 545, 545, 545, 545, 549, 275, 277, 281, 282, 283, 283, 284, 284, 285, 285, 285, 286, 287, 287, 290, 291, 298, 299, 303, 303, 304, 305, 305, 305, 305, 306, 306, 307, 307, 308, 308, 312, 312, 319, 320, 323, 325, 325, 325, 326, 326, 326, 326, 327, 327, 333, 335, 338, 338, 338, 339, 339, 339, 340, 340, 344, 347, 350, 351, 351, 352, 352, 352, 353, 353, 363, 365, 368, 368, 369, 369, 369, 370, 370, 375, 376, 379, 380, 380, 380, 380, 380, 381, 385, 386, 389, 390, 390, 390, 390, 391, 391, 411, 412, 417, 419, 420, 420, 420, 422, 422, 428, 428, 429, 430, 431, 431, 431, 432, 432, 432, 432, 432, 433, 434, 434, 435, 435, 437, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??

Wm1, **344**, used: 344, 347, 351, 353

Wm2, **344**, used: 344, 347, 353

Wp, **236**, **447**, ??, ??, **498**, **529**, **274**, **297**, **319**, **333**, **344**, **363**, **374**, **384**, **411**, **428**, ??, ??, used: 237, 239, 244, 246, 246, 246, 246, 247, 248, 249, 250, 251, 251, 252, 253, 255, 255, 256, 256, 256, 259, 447, 449, 456, 456, 456, 456, 456, 462, 463, 463, 463, 464, 464, 465, 465, 466, 466, 476, 476, 476, 476, 479, ??, 498, 500, 504, 504, 504, 504, 504, 508, 508, 508, 509, 509, 509, 509, 510, 510, 510, 510, 510, 511, 511, 511, 514, 529, 531, 535, 535, 535, 535, 536, 542, 542, 543, 544, 544, 544, 544, 544, 544, 544, 544, 545, 545, 545, 545, 549, 275, 277, 281, 282, 283, 283, 284, 284, 285, 285, 285, 286, 287, 287, 290, 291, 298, 299, 303, 303, 304, 305, 305, 305, 305, 306, 306, 307, 307, 308, 308, 312, 312, 319, 320, 323, 325, 325, 325, 326, 326, 326, 326, 327, 327, 333, 335, 338, 338, 338, 339, 339, 339, 340, 340, 344, 347, 350, 351, 351, 352, 352, 352, 353, 353, 363, 365, 368, 368, 369, 369, 369, 370, 370, 375, 376, 379, 380, 380, 380, 380, 380, 381, 385, 386, 389, 390, 390, 390, 390, 391, 391, 411, 412, 417, 419, 420, 420, 420, 422, 422, 428, 428, 429, 430, 431, 431, 431, 432, 432, 432, 432, 432, 433, 434, 434, 435, 435, 437, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??

Wp1, **344**, used: 344, 347, 351, 353

Wp2, **344**, used: 344, 347, 351, 353

write, **737**, **738**, **698**, **702**, **736**, **698**, used: ??, 105
write_dir, **737**, **738**, **736**, used: 105
write_interface, **704**, **698**, used:
write_interface_function, **703**, used: 704
write_interface_subroutine, **703**, used: 704
write_makefile, **705**, **698**, used:
write_makefile_processes, **705**, **698**, used:
write_other_declarations, **704**, used: 704
write_other_interface_functions, **704**, used: 704
X0, **319**, used: 319, 320, 323, 325, 326, 327, 327
Xdim (module), **332**, **272**, used: ??
XiA, **266**, used: 266
XiW, **266**, used: 266
XiZ, **266**, used: 266
Xm, **319**, used: 319, 320, 323, 325, 326, 326, 327
Xp, **319**, used: 319, 320, 323, 325, 326, 326, 327
xy_currents, **323**, used: 327
Y0, **319**, used: 319, 320, 323, 325, 327
yukawa, **246**, **282**, **304**, **325**, **338**, **351**, **368**, **379**, **390**, **417**, **428**, ??, ??, used: 258, 291, 312, 327, 340, 353, 370, 380, 391, 422, 436, ??, ??
yukawa_add, **283**, **304**, **325**, used: 291, 312, 327
yukawa_add', **282**, **304**, used: 283, 304
yukawa_c, **461**, **507**, **538**, used: 477, 513, 548
yukawa_cq, **461**, **507**, **539**, used: 477, 513, 548
yukawa_cq', **461**, **507**, **539**, used: 461, 507, 539
yukawa_cq'', **461**, used: 461
yukawa_c_1, **461**, used: 461
yukawa_c_2, **461**, **507**, **538**, used: 461, 507, 538
yukawa_c_3, **461**, **507**, **538**, used: 461, 507, 538
yukawa_goldstone, **457**, used: 478
yukawa_goldstone_2, **458**, used: 478
yukawa_goldstone_2', **458**, used: 458
yukawa_goldstone_2'', **458**, used: 458
yukawa_goldstone_quark, **458**, used: 478
yukawa_higgs, **457**, **505**, **537**, used: 477, 513, 548
yukawa_higgs_2, **458**, **506**, **537**, used: 477, 513, 548
yukawa_higgs_2', **458**, used: 458
yukawa_higgs_2'', **458**, used: 458
yukawa_higgs_FFP, **505**, **536**, used: 505, 537
yukawa_higgs_FFS, **505**, **536**, used: 505, 537
yukawa_higgs_NLC, **505**, **537**, used: 505, 537
yukawa_higgs_quark, **457**, **505**, **537**, used: 477, 513, 548
yukawa_n, **459**, **507**, **538**, used: 477, 513, 548
yukawa_n_1, **459**, used: 459
yukawa_n_2, **459**, **506**, **538**, used: 459, 507, 538
yukawa_n_3, **459**, **506**, **538**, used: 459, 507, 538
yukawa_n_4, **459**, used: 459
yukawa_n_5, **459**, **507**, **538**, used: 459, 507, 538
yukawa_phiggs_2, **506**, **537**, used: 506, 537
yukawa_shiggs_2, **506**, **537**, used: 506, 537
yukawa_v, **457**, **505**, **536**, used: 477, 513, 548
yuk_lqino_ec_su1, **540**, used: 548
yuk_lqino_ec_su1', **540**, used: 540
yuk_lqino_ec_su2, **540**, used: 548
yuk_lqino_ec_su2', **540**, used: 540

yuk_lqino_nc_sd1, **540**, used: 548
yuk_lqino_nc_sd2, **540**, used: 548
yuk_lqino_phiggs, **541**, used: 548
yuk_lqino_se_uc1, **539**, used: 548
yuk_lqino_se_uc1', **539**, used: 539
yuk_lqino_se_uc2, **540**, used: 548
yuk_lqino_se_uc2', **539**, used: 540
yuk_lqino_shiggs, **541**, used: 548
yuk_lqino_sn_dc1, **540**, used: 548
yuk_lqino_sn_dc2, **540**, used: 548
yuk_lq_ec_uc, **540**, used: 548
yuk_lq_ec_uc', **540**, used: 540
yuk_lq_ec_uc2, **540**, used: 548
yuk_lq_ec_uc2', **540**, used: 540
yuk_lq_nc_dc, **540**, used: 548
yuk_lq_nc_dc2, **540**, used: 548
yyact, **??**, **??**, used: **??**, **??**
yycheck, **??**, **??**, used: **??**, **??**
yydefred, **??**, **??**, used: **??**, **??**
yydgoto, **??**, **??**, used: **??**, **??**
yygindex, **??**, **??**, used: **??**, **??**
yylen, **??**, **??**, used: **??**, **??**
yylhs, **??**, **??**, used: **??**, **??**
yynames_block, **??**, **??**, used: **??**, **??**
yynames_const, **??**, **??**, used: **??**, **??**
yyrindex, **??**, **??**, used: **??**, **??**
yysindex, **??**, **??**, used: **??**, **??**
yytable, **??**, **??**, used: **??**, **??**
yytables, **??**, **??**, used: **??**, **??**
yytablesize, **??**, **??**, used: **??**, **??**
yytransl_block, **??**, **??**, used: **??**, **??**
yytransl_const, **??**, **??**, used: **??**, **??**
z, **397**, used:
Z, **236**, **447**, **??**, **??**, **498**, **529**, **274**, **297**, **319**, **333**, **344**, **363**, **374**, **384**, **397**, **411**, **428**, **??**, **??**, used: 237, 239, 243, 244, 246, 246, 247, 248, 249, 250, 251, 251, 252, 253, 253, 254, 254, 254, 256, 259, 447, 449, 455, 456, 457, 457, 462, 463, 463, 463, 464, 464, 465, 465, 466, 466, 476, 476, 476, 476, 479, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, 498, 500, 503, 505, 505, 505, 508, 508, 508, 509, 509, 509, 509, 509, 510, 510, 510, 511, 511, 511, 514, 529, 531, 535, 536, 536, 536, 542, 542, 542, 542, 542, 542, 543, 543, 543, 543, 543, 544, 544, 544, 545, 545, 545, 546, 549, 275, 277, 281, 281, 283, 283, 284, 284, 285, 285, 285, 286, 287, 287, 290, 290, 291, 298, 299, 303, 303, 305, 305, 305, 305, 306, 306, 307, 307, 308, 308, 311, 312, 312, 319, 320, 323, 323, 325, 325, 326, 326, 326, 326, 326, 326, 327, 327, 327, 333, 335, 338, 338, 338, 339, 339, 339, 339, 340, 340, 344, 347, 350, 351, 351, 352, 352, 352, 352, 353, 353, 363, 365, 368, 368, 369, 369, 369, 369, 370, 370, 375, 376, 379, 380, 380, 380, 380, 380, 381, 385, 386, 389, 390, 390, 390, 390, 391, 391, 391, 397, 398, 404, 404, 405, 405, 405, 407, 411, 412, 417, 419, 420, 420, 420, 422, 422, 428, 428, 429, 430, 430, 431, 431, 431, 432, 432, 432, 432, 432, 433, 433, 434, 437, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**, **??**
Z (module), **181**, **180**, used:

Z1, **344**, used: 344, 347, 351, 353
Z2, **344**, used: 344, 347, 351, 353
zero, **27**, **27**, **54**, **55**, **62**, **83**, **84**, **19**, **51**, **81**, used: 29, 29, 29, 30, 30, 30, 30, 31, 31, 32, 32, 32, 74, 134, 134
Zero (exn), **29**, used:
ZH, **274**, **297**, **319**, **??**, used: 275, 277, 281, 281, 283, 284, 285, 285, 286, 287, 291, 298, 299, 303, 303, 305, 305, 306, 307, 307, 308, 312, 319, 320, 324, 325, 326, 326, 327, 327, **??**, **??**, **??**, **??**
Zprime (module), **??**, **??**, used: **??**, **??**
ZZ (module), **181**, **180**, used: 228, 231, 224
__ocaml_lex_tables, **??**, **??**, used: **??**, **??**
__ocaml_lex_token_rec, **??**, **??**, used: **??**, **??**, **??**, **??**