

O'Mega: Optimal Monte-Carlo Event Generation Amplitudes

Thorsten Ohl*

Institut für Theoretische Physik und Astrophysik
Julius-Maximilians-Universität Würzburg
Am Hubland, 97074 Würzburg, Germany

Jürgen Reuter†

Physikalisches Institut
Albert-Ludwigs-Universität Freiburg
Hermann-Herder-Str. 3, 79104 Freiburg, Germany

Wolfgang Kilian^{c,‡}

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

with contributions from Christian Schwinn et al.

unpublished draft, printed 02/03/2010, 01:57

Abstract

...

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

†juergen.reuter@physik.uni-freiburg.de

‡kilian@hep.physik.uni-siegen.de

Copyright © 1999-2009 by

- Wolfgang Kilian <kilian@hep.physik.uni-siegen.de>
- Thorsten Ohl <ohl@physik.uni-wuerzburg.de>
- Jürgen Reuter <juergen.reuter@physik.uni-freiburg.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.

Revision Control

CONTENTS

1	INTRODUCTION	1
1.1	<i>Complexity</i>	1
1.2	<i>Ancestors</i>	2
1.3	<i>Architecture</i>	2
1.3.1	<i>General purpose libraries</i>	2
1.3.2	<i>O'Mega</i>	2
1.3.3	<i>Abstract interfaces</i>	4
1.3.4	<i>Models</i>	4
1.3.5	<i>Targets</i>	4
1.3.6	<i>Applications</i>	4
1.4	<i>The Big To Do Lists</i>	5
1.4.1	<i>Required</i>	5
1.4.2	<i>Useful</i>	5
1.4.3	<i>Future Features</i>	5
1.4.4	<i>Science Fiction</i>	6
2	TUPLES AND POLYTUPLES	7
2.1	<i>Interface of Tuple</i>	7
2.2	<i>Implementation of Tuple</i>	10
2.2.1	<i>Typesafe Combinatorics</i>	11
2.2.2	<i>Pairs</i>	11
2.2.3	<i>Triples</i>	13
2.2.4	<i>Pairs and Triples</i>	14
2.2.5	<i>... and All The Rest</i>	16
3	TOPOLOGIES	20
3.1	<i>Interface of Topology</i>	20
3.1.1	<i>Diagnostics: Counting Diagrams and Factorizations for $\sum_n \lambda_n \phi^n$</i>	21
3.1.2	<i>Emulating HELAC</i>	22
3.2	<i>Implementation of Topology</i>	22
3.2.1	<i>Factorizing Diagrams for ϕ^3</i>	23
3.2.2	<i>Factorizing Diagrams for $\sum_n \lambda_n \phi^n$</i>	26
3.2.3	<i>Factorizing Diagrams for ϕ^4</i>	28
3.2.4	<i>Factorizing Diagrams for $\phi^3 + \phi^4$</i>	28
3.2.5	<i>Diagnostics: Counting Diagrams and Factorizations for $\sum_n \lambda_n \phi^n$</i>	29
3.2.6	<i>Emulating HELAC</i>	35

4	DIRECTED ACYCLICAL GRAPHS	37
4.1	<i>Interface of DAG</i>	37
4.1.1	<i>Forests</i>	37
4.1.2	<i>DAGs</i>	39
4.1.3	<i>Graded Sets, Forests & DAGs</i>	41
4.2	<i>Implementation of DAG</i>	42
4.2.1	<i>The Forest Functor</i>	44
4.2.2	<i>Gradings</i>	44
4.2.3	<i>The DAG Functor</i>	46
5	MOMENTA	51
5.1	<i>Interface of Momentum</i>	51
5.1.1	<i>Scattering Kinematics</i>	53
5.2	<i>Implementation of Momentum</i>	54
5.2.1	<i>Lists of Integers</i>	55
5.2.2	<i>Bit Fiddlings</i>	60
5.2.3	<i>Whizard</i>	65
5.2.4	<i>Suggesting a Default Implementation</i>	66
6	CASCADES	67
6.1	<i>Interface of Cascade_syntax</i>	67
6.2	<i>Implementation of Cascade_syntax</i>	67
6.3	<i>Lexer</i>	69
6.4	<i>Parser</i>	70
6.5	<i>Interface of Cascade</i>	71
6.6	<i>Implementation of Cascade</i>	72
7	COLOR	77
7.1	<i>Interface of Color</i>	77
7.1.1	<i>Quantum Numbers</i>	77
7.1.2	<i>Color Flows</i>	77
7.1.3	<i>Evaluation</i>	78
7.1.4	<i>Color Flow Representation</i>	79
7.2	<i>Implementation of Color</i>	80
7.2.1	<i>Quantum Numbers</i>	80
7.2.2	<i>Color Flows</i>	80
7.2.3	<i>Evaluation</i>	83
7.2.4	<i>Color Flow Representation</i>	84
7.2.5	<i>Evaluation Revisited</i>	92
8	FUSIONS	98
8.1	<i>Interface of Fusion</i>	98
8.1.1	<i>Multiple Colored Amplitudes</i>	101
8.1.2	<i>Tags</i>	102
8.2	<i>Implementation of Fusion</i>	103
8.2.1	<i>Fermi Statistics</i>	104
8.2.2	<i>Dirac Fermions</i>	104
8.2.3	<i>Tags</i>	106
8.2.4	<i>The Fusion.Make Functor</i>	107
8.2.5	<i>Fusions with Majorana Fermions</i>	125

8.2.6	<i>Multiple Colored Amplitudes</i>	127
9	LORENTZ REPRESENTATIONS, COUPLINGS, MODELS AND TARGETS	132
10	COLORIZATION	133
10.1	<i>Interface of Colorize</i>	133
10.1.1	<i>...</i>	133
10.2	<i>Implementation of Colorize</i>	133
10.2.1	<i>(Statically) Colorizing a Monochrome Model</i>	133
10.2.2	<i>(Statically) Colorizing a Monochrome Gauge Model</i>	157
10.2.3	<i>(Dynamically) Colorizing a Monochrome Model</i>	159
11	VERTICES	167
12	MODELS	168
13	HARDCODED MODELS	169
13.1	<i>Implementation of Models2</i>	169
14	COMPHEP MODELS	170
14.1	<i>Interface of Comphep-syntax</i>	170
14.2	<i>Implementation of Comphep-syntax</i>	170
14.3	<i>Lexer</i>	172
14.4	<i>Parser</i>	173
14.5	<i>Interface of Comphep</i>	174
14.6	<i>Implementation of Comphep</i>	174
15	HARDCODED TARGETS	184
15.1	<i>Interface of Targets</i>	184
15.1.1	<i>Supported Targets</i>	184
15.1.2	<i>Potential Targets</i>	184
15.2	<i>Implementation of Targets</i>	184
15.2.1	<i>Fortran 90/95</i>	185
15.2.2	<i>O'Mega Virtual Machine</i>	242
15.2.3	<i>C</i>	242
15.2.4	<i>O'Caml</i>	242
15.2.5	<i>L^AT_EX</i>	242
15.3	<i>Interface of Targets_Kmatrix</i>	244
15.4	<i>Implementation of Targets_Kmatrix</i>	244
16	PHASE SPACE	255
16.1	<i>Interface of Phasespace</i>	255
16.2	<i>Implementation of Phasespace</i>	256
16.2.1	<i>Tools</i>	256
16.2.2	<i>Phase Space Parameterization Trees</i>	256
17	WHIZARD	262
17.1	<i>Interface of Whizard</i>	262
17.2	<i>Implementation of Whizard</i>	262
17.2.1	<i>Building Trees</i>	263
17.2.2	<i>Merging Homomorphic Trees</i>	264

17.2.3	<i>Printing Trees</i>	265
17.2.4	<i>Process Dispatcher</i>	267
17.2.5	<i>Makefile</i>	269
18	APPLICATIONS	271
18.1	<i>Sample</i>	271
18.2	<i>Interface of Omega</i>	271
18.3	<i>Implementation of Omega</i>	271
18.3.1	<i>Parsing Process Descriptions</i>	273
18.3.2	<i>Main Program</i>	275
18.4	<i>Implementation of Omega_QED</i>	278
18.5	<i>Implementation of Omega_QCD</i>	278
18.6	<i>Implementation of Omega_SM</i>	282
19	O’GIGA: O’MEGA GRAPHICAL INTERFACE FOR GENERATION AND ANALYSIS	283
A	REVISION CONTROL	286
A.1	<i>Interface of RCS</i>	286
A.2	<i>Implementation of RCS</i>	287
B	TEXTUAL OPTIONS	289
B.1	<i>Interface of Options</i>	289
B.2	<i>Implementation of Options</i>	289
C	PROGRESS REPORTS	291
C.1	<i>Interface of Progress</i>	291
C.2	<i>Implementation of Progress</i>	291
D	CACHE FILES	294
D.1	<i>Interface of Cache</i>	294
D.2	<i>Implementation of Cache</i>	294
E	MORE ON LISTS	297
E.1	<i>Interface of ThoList</i>	297
E.2	<i>Implementation of ThoList</i>	298
F	MORE ON ARRAYS	302
F.1	<i>Interface of ThoArray</i>	302
F.2	<i>Implementation of ThoArray</i>	302
G	POLYMORPHIC MAPS	305
G.1	<i>Interface of Pmap</i>	305
G.2	<i>Implementation of Pmap</i>	306
H	TRIES	316
H.1	<i>Interface of Trie</i>	316
H.1.1	<i>Monomorphically</i>	316
H.1.2	<i>New in O’Caml 3.08</i>	316
H.1.3	<i>O’Mega customization</i>	317
H.1.4	<i>Polymorphically</i>	317

H.1.5	<i>O'Mega customization</i>	317
H.2	<i>Implementation of Trie</i>	318
H.2.1	<i>Monomorphically</i>	318
H.2.2	<i>O'Mega customization</i>	320
H.2.3	<i>Polymorphically</i>	321
H.2.4	<i>O'Mega customization</i>	323
I	TENSOR PRODUCTS	324
I.1	<i>Interface of Product</i>	324
I.1.1	<i>Lists</i>	324
I.1.2	<i>Sets</i>	324
I.2	<i>Implementation of Product</i>	325
I.2.1	<i>Lists</i>	325
I.2.2	<i>Sets</i>	326
J	COMBINATORICS	327
J.1	<i>Interface of Combinatorics</i>	327
J.1.1	<i>Simple Combinatorial Functions</i>	327
J.1.2	<i>Partitions</i>	327
J.1.3	<i>Choices</i>	329
J.1.4	<i>Permutations</i>	329
J.2	<i>Implementation of Combinatorics</i>	329
J.2.1	<i>Simple Combinatorial Functions</i>	330
J.2.2	<i>Partitions</i>	331
J.2.3	<i>Choices</i>	334
J.2.4	<i>Permutations</i>	335
K	PARTITIONS	337
K.1	<i>Interface of Partition</i>	337
K.2	<i>Implementation of Partition</i>	337
L	TREES	339
L.1	<i>Interface of Tree</i>	340
L.1.1	<i>Abstract Data Type</i>	340
L.1.2	<i>Homomorphisms</i>	341
L.1.3	<i>Output</i>	341
L.2	<i>Implementation of Tree</i>	342
L.2.1	<i>Abstract Data Type</i>	342
L.2.2	<i>Homomorphisms</i>	344
L.2.3	<i>Output</i>	344
L.2.4	<i>Least Squares Layout</i>	348
M	CONSISTENCY CHECKS	353
N	COMPLEX NUMBERS	354
N.1	<i>Interface of Complex</i>	354
N.2	<i>Implementation of Complex</i>	355
N.2.1	<i>Sparse Representation</i>	358
N.2.2	<i>Suggesting A Default</i>	358

O	ALGEBRA	359
	<i>O.1 Interface of Algebra</i>	359
	<i>O.1.1 Coefficients</i>	359
	<i>O.1.2 Naive Rational Arithmetic</i>	359
	<i>O.1.3 Expressions: Terms, Rings and Linear Combinations</i>	360
	<i>O.2 Implementation of Algebra</i>	362
	<i>O.2.1 Coefficients</i>	362
	<i>O.2.2 Naive Rational Arithmetic</i>	362
	<i>O.2.3 Expressions: Terms, Rings and Linear Combinations</i>	363
P	SIMPLE LINEAR ALGEBRA	369
	<i>P.1 Interface of Linalg</i>	369
	<i>P.2 Implementation of Linalg</i>	369
	<i>P.2.1 LU Decomposition</i>	370
Q	TALK TO THE WHIZARD . . .	374
R	WIDGET LIBRARY AND CLASS HIERARCHY FOR O’GIGA	375
	<i>R.1 Architecture</i>	375
	<i>R.1.1 Inheritance vs. Aggregation</i>	375
S	O’MEGA VIRTUAL MACHINE	377
T	FORTTRAN LIBRARIES	378
	<i>T.1 Trivia</i>	378
	<i>T.1.1 Inner Product</i>	378
	<i>T.1.2 Spinor Vector Space</i>	379
	<i>T.1.3 Norm</i>	383
	<i>T.2 Spinors Revisited</i>	383
	<i>T.2.1 Spinor Vector Space</i>	384
	<i>T.2.2 Norm</i>	387
	<i>T.3 Vectorspinors</i>	387
	<i>T.3.1 Vectorspinor Vector Space</i>	388
	<i>T.3.2 Norm</i>	392
	<i>T.4 Vectors and Tensors</i>	392
	<i>T.4.1 Constructors</i>	393
	<i>T.4.2 Inner Products</i>	394
	<i>T.4.3 Not Entirely Inner Products</i>	395
	<i>T.4.4 Outer Products</i>	396
	<i>T.4.5 Vector Space</i>	397
	<i>T.4.6 Norm</i>	404
	<i>T.4.7 Conjugation</i>	404
	<i>T.4.8 ϵ-Tensors</i>	405
	<i>T.4.9 Utilities</i>	408
	<i>T.5 Polarization vectors</i>	408
	<i>T.6 Polarization vectors revisited</i>	412
	<i>T.7 Symmetric Tensors</i>	414
	<i>T.7.1 Vector Space</i>	415
	<i>T.8 Symmetric Polarization Tensors</i>	419
	<i>T.9 Couplings</i>	421

T.9.1	Triple Gauge Couplings	425
T.9.2	Quadruple Gauge Couplings	426
T.9.3	Scalar Current	426
T.9.4	Triple Vector Couplings	427
T.10	Graviton Couplings	430
T.11	Tensor Couplings	432
T.12	Scalar-Vector Dim-5 Couplings	432
T.13	Spinor Couplings	434
T.13.1	Fermionic Vector and Axial Couplings	435
T.13.2	Fermionic Scalar and Pseudo Scalar Couplings	444
T.13.3	On Shell Wave Functions	448
T.13.4	Off Shell Wave Functions	450
T.13.5	Propagators	452
T.14	Spinor Couplings Revisited	455
T.14.1	Fermionic Vector and Axial Couplings	455
T.14.2	Fermionic Scalar and Pseudo Scalar Couplings	461
T.14.3	Couplings for BRST Transformations	463
T.14.4	Gravitino Couplings	470
T.14.5	Gravitino 4-Couplings	492
T.14.6	On Shell Wave Functions	503
T.14.7	Off Shell Wave Functions	506
T.14.8	Propagators	507
T.15	Polarization vectorspinors	509
T.16	Utilities	513
T.16.1	Helicity Selection Rule Heuristics	513
T.16.2	Diagnostics	514
T.16.3	Summation & Density Matrices	519
T.16.4	Obsolescent Summation	528
T.17	omega95	531
T.18	omega95 Revisited	531
T.19	Testing	531
T.20	O'Mega Virtual Machine	545
T.20.1	Memory Layout	545
T.20.2	Instruction Set	546
U	INDEX	549

—1—

INTRODUCTION

1.1 Complexity

There are

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

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

$$F(n) = (2n - 5)!! = (2n - 5) \cdot (2n - 7) \cdot \dots \cdot 3 \cdot 1 \quad (1.2)$$

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

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

while the 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 builds 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 E, p. 297)

Product : efficient tensor products for lists and sets (section I, p. 324)

Combinatorics : combinatorical formulae, sets of subsets, etc. (section J, p. 327)

1.3.2 O’Mega

The non-trivial algorithms that constitute O’Mega:

DAG : Directed Acyclical Graphs (section 4, p. 37)

Topology : unusual enumerations of unflavored tree diagrams (section 3, p. 20)

Momentum : finite sums of external momenta (section 5, p. 51)

Fusion : off shell wave functions (section 8, p. 98)

OVM : O’Mega Virtual Machine (not implemented yet) (section S, p. 377)

Omega : functor constructing an application from a model and a target
(section 18, p. 271)

1.3.3 Abstract interfaces

The domains and co-domains of functors (section 9, p. 132)

Coupling : all possible couplings (not comprehensive yet)

Model : physical models

Target : target programming languages

1.3.4 Models

(section 13, p. 169)

Models.QED : Quantum Electrodynamics

Models.QCD : Quantum Chromodynamics (not complete yet)

Models.SM : Minimal Standard Model (not complete yet)

Other models will be supported by a convenient concrete syntax for langrangians in text files.

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. 184).

Targets.Fortran : Fortran95 language implementation, calling subroutines

Targets.Fortran_Inlined : Fortran language implementation, self contained

Targets.Helas : Fortran language implementation calling HELAS [5] subroutines

Other targets will come in the future: C, C++, O’Caml itself, symbolic manipulation languages, etc.

1.3.6 Applications

(section 18, p. 271)

1.4 The Big To Do Lists

1.4.1 Required

All features planned for a first release are in place.

1.4.2 Useful

1. complete standard model in R_ξ -gauge
2. provide `omega77`, a Fortran77 library equivalent to `omega95` (i.e. a more orthogonal HELAS clone)
3. groves (the simple method of cloned generations works)
4. color factors for a “few” colored particles, maybe one can separate color “eigenamplitudes”
5. color factors for many colored particles: Mangano, Moretti et al.
6. select allowed helicity combinations for massless fermions
7. Weyl-Van der Waerden spinors
8. speed up helicity sums by using discrete symmetries
9. general triple and quartic vector couplings
10. complete MSSM
11. diagnostics: count corresponding Feynman diagrams more efficiently for more than ten external lines
12. recognize potential cascade decays (τ , b , etc.)
 - warn the user to add additional
 - kill fusions (at runtime), that contribute to a cascade

1.4.3 Future Features

1. investigate if unpolarized squared matrix elements can be calculated faster as traces of density matrices. Unfortunately, the answer appears 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 amplitudes 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 other target as well, because native code appears to become too 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  $\alpha$  t

  val arity :  $\alpha$  t  $\rightarrow$  int
  val max_arity : int

  val compare : ( $\alpha \rightarrow \alpha \rightarrow$  int)  $\rightarrow$   $\alpha$  t  $\rightarrow$   $\alpha$  t  $\rightarrow$  int

  val for_all : ( $\alpha \rightarrow$  bool)  $\rightarrow$   $\alpha$  t  $\rightarrow$  bool

  val map : ( $\alpha \rightarrow \beta$ )  $\rightarrow$   $\alpha$  t  $\rightarrow$   $\beta$  t
  val iter : ( $\alpha \rightarrow$  unit)  $\rightarrow$   $\alpha$  t  $\rightarrow$  unit
  val fold_left : ( $\alpha \rightarrow \beta \rightarrow \alpha$ )  $\rightarrow$   $\alpha \rightarrow \beta$  t  $\rightarrow$   $\alpha$ 
  val fold_right : ( $\alpha \rightarrow \beta \rightarrow \beta$ )  $\rightarrow$   $\alpha$  t  $\rightarrow$   $\beta \rightarrow \beta$ 
```

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

```
val fold_left_internal : ( $\alpha \rightarrow \alpha \rightarrow \alpha$ )  $\rightarrow$   $\alpha$  t  $\rightarrow$   $\alpha$ 
val fold_right_internal : ( $\alpha \rightarrow \alpha \rightarrow \alpha$ )  $\rightarrow$   $\alpha$  t  $\rightarrow$   $\alpha$ 

val map2 : ( $\alpha \rightarrow \beta \rightarrow \gamma$ )  $\rightarrow$   $\alpha$  t  $\rightarrow$   $\beta$  t  $\rightarrow$   $\gamma$  t

val split : ( $\alpha \times \beta$ ) t  $\rightarrow$   $\alpha$  t  $\times$   $\beta$  t
```

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

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

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

```
val product :  $\alpha$  list t  $\rightarrow$   $\alpha$  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.:

$$power [x_1; x_2] = product ([x_1; x_2], [x_1; x_2]) = [(x_1, x_1); (x_1, x_2); (x_2, x_1); (x_2, x_2)] \quad (2.2)$$

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

$$power [x_1; x_2] = product ([x_1; x_2], [x_1; x_2]) \cup product ([x_1; x_2], [x_1; x_2], [x_1; x_2]) \quad (2.3)$$

For tuples and polytuples with bounded arity, the *power* and *power_fold* functions terminate. In polytuples with unbounded arity, 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.:

$$sym_power [x_1; x_2] = [(x_1, x_1); (x_1, x_2); (x_2, x_2)] \quad (2.4)$$

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 $\phi : X \rightarrow \mathbf{N}$, 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'Cam1 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 :  $\alpha \rightarrow \alpha \rightarrow \alpha t$ 
  end
module Binary : Binary

module type Ternary =
  sig
    include Mono
    val of3 :  $\alpha \rightarrow \alpha \rightarrow \alpha \rightarrow \alpha t$ 
  end
module Ternary : Ternary

type  $\alpha$  pair_or_triple = T2 of  $\alpha \times \alpha$  | T3 of  $\alpha \times \alpha \times \alpha$ 

module type Mixed23 =
  sig
    include Poly
    val of2 :  $\alpha \rightarrow \alpha \rightarrow \alpha t$ 
    val of3 :  $\alpha \rightarrow \alpha \rightarrow \alpha \rightarrow \alpha t$ 
  end
module Mixed23 : Mixed23

module type Nary =
  sig
    include Poly
    val of2 :  $\alpha \rightarrow \alpha \rightarrow \alpha t$ 
    val of3 :  $\alpha \rightarrow \alpha \rightarrow \alpha \rightarrow \alpha t$ 
    val of_list :  $\alpha \text{ list} \rightarrow \alpha t$ 
  end
module Unbounded_Nary : Nary

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

```



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

- val length : $\alpha t \rightarrow int$
- val hd : $\alpha t \rightarrow \alpha$
- val nth : $\alpha t \rightarrow int \rightarrow \alpha$
- val rev : $\alpha t \rightarrow \alpha t$

- `val rev_map : ($\alpha \rightarrow \beta$) $\rightarrow \alpha\ t \rightarrow \beta\ t$`
- `val iter2 : ($\alpha \rightarrow \beta \rightarrow \text{unit}$) $\rightarrow \alpha\ t \rightarrow \beta\ t \rightarrow \text{unit}$`
- `val rev_map2 : ($\alpha \rightarrow \beta \rightarrow \gamma$) $\rightarrow \alpha\ t \rightarrow \beta\ t \rightarrow \gamma\ t$`
- `val fold_left2 : ($\alpha \rightarrow \beta \rightarrow \gamma \rightarrow \alpha$) $\rightarrow \alpha \rightarrow \beta\ t \rightarrow \gamma\ t \rightarrow \alpha$`
- `val fold_right2 : ($\alpha \rightarrow \beta \rightarrow \gamma \rightarrow \gamma$) $\rightarrow \alpha\ t \rightarrow \beta\ t \rightarrow \gamma \rightarrow \gamma$`
- `val exists : ($\alpha \rightarrow \text{bool}$) $\rightarrow \alpha\ t \rightarrow \text{bool}$`
- `val for_all2 : ($\alpha \rightarrow \beta \rightarrow \text{bool}$) $\rightarrow \alpha\ t \rightarrow \beta\ t \rightarrow \text{bool}$`
- `val exists2 : ($\alpha \rightarrow \beta \rightarrow \text{bool}$) $\rightarrow \alpha\ t \rightarrow \beta\ t \rightarrow \text{bool}$`
- `val mem : $\alpha \rightarrow \alpha\ t \rightarrow \text{bool}$`
- `val memq : $\alpha \rightarrow \alpha\ t \rightarrow \text{bool}$`
- `val find : ($\alpha \rightarrow \text{bool}$) $\rightarrow \alpha\ t \rightarrow \alpha$`
- `val find_all : ($\alpha \rightarrow \text{bool}$) $\rightarrow \alpha\ t \rightarrow \alpha\ \text{list}$`
- `val assoc : $\alpha \rightarrow (\alpha \times \beta)\ t \rightarrow \beta$`
- `val assq : $\alpha \rightarrow (\alpha \times \beta)\ t \rightarrow \beta$`
- `val mem_assoc : $\alpha \rightarrow (\alpha \times \beta)\ t \rightarrow \text{bool}$`
- `val mem_assq : $\alpha \rightarrow (\alpha \times \beta)\ t \rightarrow \text{bool}$`
- `val combine : $\alpha\ t \rightarrow \beta\ t \rightarrow (\alpha \times \beta)\ t$`
- `val sort : ($\alpha \rightarrow \alpha \rightarrow \text{int}$) $\rightarrow \alpha\ t \rightarrow \alpha\ t$`
- `val stable_sort : ($\alpha \rightarrow \alpha \rightarrow \text{int}$) $\rightarrow \alpha\ t \rightarrow \alpha\ t$`

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 = "$Revision: 759$";
  RCS.date = "$Date: 2009-06-10 11:38:07 +0200 (Wed, 10 Jun 2009)$";
  RCS.author = "$Author: ohl$";
  RCS.source
    = "$URL: svn+ssh://jr-reuter@login.hepforge.org/hepforge/svn/whizard/trunk/src/omeg";
module type Mono =
sig
  type  $\alpha\ t$ 
  val arity :  $\alpha\ t \rightarrow \text{int}$ 
  val max_arity :  $\text{int}$ 
  val compare : ( $\alpha \rightarrow \alpha \rightarrow \text{int}$ )  $\rightarrow \alpha\ t \rightarrow \alpha\ t \rightarrow \text{int}$ 
  val for_all : ( $\alpha \rightarrow \text{bool}$ )  $\rightarrow \alpha\ t \rightarrow \text{bool}$ 
  val map : ( $\alpha \rightarrow \beta$ )  $\rightarrow \alpha\ t \rightarrow \beta\ t$ 
  val iter : ( $\alpha \rightarrow \text{unit}$ )  $\rightarrow \alpha\ t \rightarrow \text{unit}$ 
  val fold_left : ( $\alpha \rightarrow \beta \rightarrow \alpha$ )  $\rightarrow \alpha \rightarrow \beta\ t \rightarrow \alpha$ 
  val fold_right : ( $\alpha \rightarrow \beta \rightarrow \beta$ )  $\rightarrow \alpha\ t \rightarrow \beta \rightarrow \beta$ 
  val fold_left_internal : ( $\alpha \rightarrow \alpha \rightarrow \alpha$ )  $\rightarrow \alpha\ t \rightarrow \alpha$ 
  val fold_right_internal : ( $\alpha \rightarrow \alpha \rightarrow \alpha$ )  $\rightarrow \alpha\ t \rightarrow \alpha$ 
```

```

val map2 : ( $\alpha \rightarrow \beta \rightarrow \gamma$ )  $\rightarrow \alpha\ t \rightarrow \beta\ t \rightarrow \gamma\ t$ 
val split : ( $\alpha \times \beta$ )  $t \rightarrow \alpha\ t \times \beta\ t$ 
val product :  $\alpha\ list\ t \rightarrow \alpha\ t\ list$ 
val product_fold : ( $\alpha\ t \rightarrow \beta \rightarrow \beta$ )  $\rightarrow \alpha\ list\ t \rightarrow \beta \rightarrow \beta$ 
val power :  $\alpha\ list \rightarrow \alpha\ t\ list$ 
val power_fold : ( $\alpha\ t \rightarrow \beta \rightarrow \beta$ )  $\rightarrow \alpha\ list \rightarrow \beta \rightarrow \beta$ 
type  $\alpha\ graded = \alpha\ list\ array$ 
val graded_sym_power :  $int \rightarrow \alpha\ graded \rightarrow \alpha\ t\ list$ 
val graded_sym_power_fold :  $int \rightarrow (\alpha\ t \rightarrow \beta \rightarrow \beta) \rightarrow \alpha\ graded \rightarrow \beta \rightarrow \beta$ 
val to_list :  $\alpha\ t \rightarrow \alpha\ list$ 
val of2_kludge :  $\alpha \rightarrow \alpha \rightarrow \alpha\ t$ 
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 combinatorial 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]  $\rightarrow (x, y) \mid - \rightarrow impossible\ "choose2"$ )
    (Combinatorics.choose 2 set)

let choose3 set =
  List.map (function [x; y; z]  $\rightarrow (x, y, z) \mid - \rightarrow impossible\ "choose3"$ )
    (Combinatorics.choose 3 set)

```

2.2.2 Pairs

```

module type Binary =
sig
  sig
    include Poly (* should become Mono! *)
    val of2 :  $\alpha \rightarrow \alpha \rightarrow \alpha\ t$ 
  end
end

module Binary =

```

```

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

  type  $\alpha$  t =  $\alpha \times \alpha$ 

  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  $\neq$  0 then
      cx
    else
      cmp y1 y2

  let for_all p (x, y) = p x  $\wedge$  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  $\rightarrow$  (x, y)) lx ly
  let product_fold f (lx, ly) init =
    Product.fold2 (fun x y  $\rightarrow$  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  $\alpha$  graded =  $\alpha$  list array

let fuse2 f set (i, j) acc =
  if i = j then
    List.fold_right (fun (x, y)  $\rightarrow$  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  $\rightarrow$  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  $\rightarrow$  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 :  $\alpha \rightarrow \alpha \rightarrow \alpha \rightarrow \alpha t$ 
  end

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

    type  $\alpha t = \alpha \times \alpha \times \alpha$ 

    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  $\neq$  0 then
        cx
      else
        let cy = cmp y1 y2 in
        if cy  $\neq$  0 then
          cy
        else
          cmp z1 z2

    let for_all p (x, y, z) = p x  $\wedge$  p y  $\wedge$  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  $\rightarrow$  (x, y, z)) lx ly lz
    let product_fold f (lx, ly, lz) init =
      Product.fold3 (fun x y z  $\rightarrow$  f (x, y, z)) lx ly lz init
  end

```

```

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

type  $\alpha$  graded =  $\alpha$  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)  $\rightarrow$  f x y z) (choose3 set.(pred i)) acc
    else
      Product.fold2 (fun (x, y) z  $\rightarrow$  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)  $\rightarrow$  f x y z)
        set.(pred i) (choose2 set.(pred j)) acc
    else
      Product.fold3 (fun x y z  $\rightarrow$  f x y z)
        set.(pred i) set.(pred j) set.(pred k) acc
  end
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  $\rightarrow$  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  $\rightarrow$  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  $\alpha$  pair_or_triple = T2 of  $\alpha \times \alpha$  | T3 of  $\alpha \times \alpha \times \alpha$ 

module type Mixed23 =
  sig
    include Poly
    val of2 :  $\alpha \rightarrow \alpha \rightarrow \alpha t$ 
    val of3 :  $\alpha \rightarrow \alpha \rightarrow \alpha \rightarrow \alpha t$ 
  end

module Mixed23 =
  struct
    let rcs = RCS.rename rcs_file "Tuple.Mixed23"
      ["Mixed_pairs_and_triples"]

    type  $\alpha t$  =  $\alpha$  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  $\alpha$  graded =  $\alpha$  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 :  $\alpha \rightarrow \alpha \rightarrow \alpha t$ 
  val of3 :  $\alpha \rightarrow \alpha \rightarrow \alpha \rightarrow \alpha t$ 
  val of_list :  $\alpha list \rightarrow \alpha t$ 
end

module Nary (A : sig val max_arity : int end) =
struct
  let rcs = RCS.rename rcs_file "Tuple.Nary"
    ["Tuples_of_indefinite_arity"]

  type  $\alpha t = \alpha \times \alpha list$ 

  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  $\alpha$  graded =  $\alpha$  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 tuples or lists.

type *partition*

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

$$\sum_{i=1}^d n_i = n \quad (3.1)$$

for d from 3 to d_{\max} , where d_{\max} is a fixed number for each module implementing T . In particular, if `type partition = int × int × 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 → 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

$$\sum_{3 \leq k \leq d_{\max}} \lambda_k \phi^k \quad (3.2)$$

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

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

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

val *keystones* : α list → (α list × α 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 combinatorial factors. Therefore we code a functor that can use arbitrary 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

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

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

val *diagrams* : ? $f : (integer \rightarrow bool) \rightarrow integer \rightarrow integer \rightarrow integer$
val *diagrams_via_keystones* : $integer \rightarrow integer \rightarrow integer$

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

val *keystones* : $integer\ list \rightarrow integer$

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

val *diagrams_per_keystone* : $integer \rightarrow integer\ list \rightarrow integer$

end

module *Count* : functor ($I : Integer$) $\rightarrow Count$ with type *integer* = $I.t$

3.1.2 Emulating HELAC

We can also proceed á la [2].

module *Helac* : functor ($B : Tuple.Bound$) \rightarrow
(T with type $\alpha\ children = \alpha\ Tuple.Nary(B).t$)



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

module *Helac_Binary* : T with type $\alpha\ children = \alpha\ Tuple.Binary.t$

3.2 Implementation of Topology

```
let rcs_file = RCS.parse "Topology" ["Topologies"]
{ RCS.revision = "$Revision: 759$";
  RCS.date = "$Date: 2009-06-10 11:38:07 +0200 (Wed, 10 Jun 2009)$";
  RCS.author = "$Author: ohl$";
  RCS.source
    = "$URL: svn+ssh://jr-reuter@login.hepforge.org/hepforge/svn/whizard/trunk/src/omeg
```

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 .

```

module type  $T$  =
  sig
    type partition
    val partitions : int → partition list
    type  $\alpha$  children
    val keystones :  $\alpha$  list → ( $\alpha$  list ×  $\alpha$  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]
  end

```

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

$$partitions : n \rightarrow \{(n_1, n_2, n_3) \mid n_1 + n_2 + n_3 = n \wedge n_1 \leq n_2 \leq n_3 \leq \lfloor n/2 \rfloor\} \quad (3.5)$$

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 $n_1 \leq \lfloor n/3 \rfloor$:

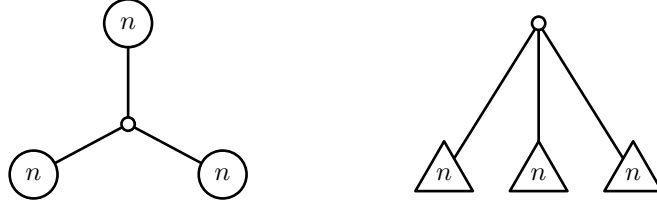


Figure 3.1: Topologies with a blatant three-fold permutation symmetry, if the number of external lines is a multiple of three

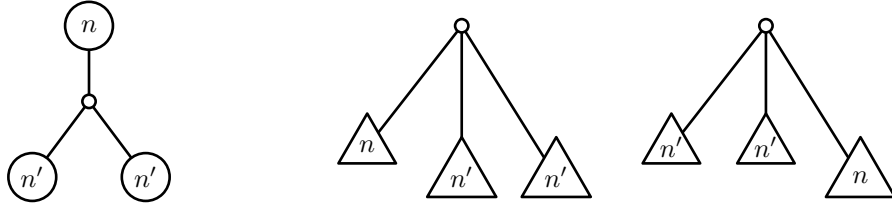


Figure 3.2: Topologies with a blatant two-fold symmetry.

```

let rec partitions' n n1 =
  if n1 > n / 3 then
    []
  else
    List.map (fun (n2, n3) → (n1, n2, n3))
      (Partition.pairs (n - n1) n1 (n / 2)) @ partitions' n (succ n1)
let partitions n = partitions' n 1
    
```

```

type α children = α 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.

$$\{1\}\{2,3\}\{4,5,6\} \mapsto \{\{4\}\{5,6\}\{1,2,3\}; \{5\}\{4,6\}\{1,2,3\}; \{6\}\{4,5\}\{1,2,3\}\} \quad (3.6)$$

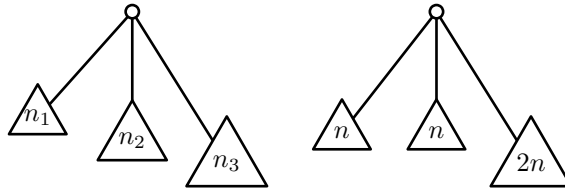


Figure 3.3: If $n_3 = n_1 + n_2$, the apparently asymmetric topologies on the left hand side have a non obvious two-fold symmetry, that exchanges the two halves. Therefore, the topologies on the right hand side have a four fold symmetry.

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 .

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 $n\beta$.

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

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

where the symmetry factor from the above arguments is

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

Indeed, the sum of all Feynman diagrams

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

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*.

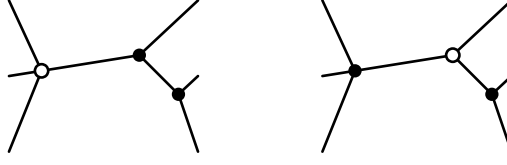


Figure 3.4: Degenerate (1, 1, 1, 3) and (1, 2, 3).

```

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

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
end

```

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

```

module Tuple = Tuple.Nary(B)
type  $\alpha$  children =  $\alpha$  Tuple.t

let keystones' l =
  let n = List.length l in
  ThoList.flatmap (fun p  $\rightarrow$  Combinatorics.factorized_keystones p l)
    (partitions n)

let keystones l =
  List.map (fun (bra, kets)  $\rightarrow$  (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 $\sqcup$ topology"]
  let rcs = rcs_file
  type partition = int  $\times$  int  $\times$  int  $\times$  int
  let inspect_partition (n1, n2, n3, n4) = [n1; n2; n3; n4]
  type  $\alpha$  children =  $\alpha$  Tuple.Ternary.t
  let collect4 acc = function
    | [x; y; z; u]  $\rightarrow$  (x, y, z, u) :: acc
    | _  $\rightarrow$  acc
  let partitions n =
    List.fold_left collect4 [] (Nary4.partitions n)
  let collect3 acc = function
    | [x; y; z]  $\rightarrow$  Tuple.Ternary.of3 x y z :: acc
    | _  $\rightarrow$  acc
  let keystones l =
    List.map (fun (bra, kets)  $\rightarrow$  (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 $\sqcup$ + $\sqcup$ phi**4 $\sqcup$ 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 [M](#), 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
  end

```

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:

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

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

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

```
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:

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

```
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  $d \geq three$  then
  try
     $IMap.find\ d\ cl.v$ 
  with
    |  $Not\_found \rightarrow zero$ 
  else
     $invalid\_arg\ "multiplicity:\_d\leq 3"$ 

```

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 of 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:

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

```

let rec class_size  $cl =$ 
  if  $cl.ext = two \vee cl.prop = zero$  then
    one
  else
     $IMap.fold\ (fun\ d\ s \rightarrow class\_size\_n\ cl\ d + s)\ cl.v\ (class\_size\_3\ cl)$ 

```

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 $\{n_3, n_4, \dots, n_d\}$ with

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

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 $\{n_3, n_4, \dots, n_d\}$ with

$$E(\{n_3, n_4, \dots, n_d\}) = \text{sum} \quad (3.14)$$

```

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 $N_{d,n} = \{n_1, n_2, \dots, n_d\}$ with $n = n_1 + n_2 + \dots + n_d$ and

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

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

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

```

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 α children = α 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

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

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

$$T(n, D) = \{\text{tree}(e_i, t_i, t'_i) \mid (e_i, t_i, t'_i) \in \{e_1\} \times T(n_1, D) \times T(n'_1, D) \cup \dots \cup \{e_k\} \times T(n_k, D) \times T(n'_k, D)\} \quad (4.2)$$

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 semantics of *compare* are compatible with *Pervasives.compare*:

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

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, in 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”).

$$F : N \rightarrow (E \times 2^N) \cup \{\perp\}$$

$$n \mapsto \begin{cases} (e, \{n'_1, n'_2, \dots\}) \\ \perp \end{cases} \quad (4.4)$$

The nodes are ordered so that cycles can be detected

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

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 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

$$f(x_1, f(x_2, \dots f(x_n, acc))) \quad (4.6)$$

where the *children* are $\{x_1, x_2, \dots, x_n\}$. 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 probably 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 iter_nodes : (node → unit) → t → unit
val map_nodes : (node → node) → t → t
val fold_nodes : (node → α → α) → t → α → α

```

... 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 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

```

Return the DAG as a list of lists.

```
val lists : t → (node × (edge × children) list) list
```

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

```
val dependencies : t → node → node Tree2.t
```

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

```
val harvest : t → node → t → t
```

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

```
val size : t → int
```

eval f mul_edge mul_nodes add null unit root dag

```

val eval : (node → α) → (node → edge → β → γ) →
  (α → β → β) → (γ → α → α) → α → β → node → t → α
val eval_memoized : (node → α) → (node → edge → β → γ) →
  (α → β → β) → (γ → α → α) → α → β → node → t → α

```

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

```
val harvest_list : t → node list → t
```

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.

```
val count_trees : node → t → int
```

forest root dag

```

val forest : node → t → (node × edge option, node) Tree.t list
val forest_memoized : node → t → (node × edge option, node) Tree.t list
val rcs : RCS.t
end

```

module *Make* (*F* : *Forest*) :

```

  T with type node = F.node and type edge = F.edge
  and type children = F.children

```

4.1.3 Graded Sets, Forests & DAGs

A graded ordered¹ 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.

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

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.

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

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



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
```

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

```

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 = "$Revision: 1340$";
  RCS.date = "$Date: 2009-12-03 00:45:04 +0100 (Thu, 03 Dec 2009)$";
  RCS.author = "$Author: ohl$";
  RCS.source
    = "$URL: svn+ssh://jr-reuter@login.hepforge.org/hepforge/svn/whizard/trunk/src/omeg
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 Tree2.t
val harvest : t → node → t → t
val size : t → int
val eval : (node → α) → (node → edge → β → γ) →
  (α → β → β) → (γ → α → α) → α → β → node → t → α
val eval_memoized : (node → α) → (node → edge → β → γ) →
  (α → β → β) → (γ → α → α) → α → β → node → t → α
val harvest_list : t → node list → t
val count_trees : node → t → int
val forest : node → t → (node × edge option, node) Tree.t list
val forest_memoized : node → t → (node × edge option, node) Tree.t list
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.rank")
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
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  $\alpha$  t
    val empty :  $\alpha$  t
    val add : key  $\rightarrow$   $\alpha$   $\rightarrow$   $\alpha$  t  $\rightarrow$   $\alpha$  t
    val find : key  $\rightarrow$   $\alpha$  t  $\rightarrow$   $\alpha$ 
    val mem : key  $\rightarrow$   $\alpha$  t  $\rightarrow$  bool
    val iter : (key  $\rightarrow$   $\alpha$   $\rightarrow$  unit)  $\rightarrow$   $\alpha$  t  $\rightarrow$  unit
    val fold : (key  $\rightarrow$   $\alpha$   $\rightarrow$   $\beta$   $\rightarrow$   $\beta$ )  $\rightarrow$   $\alpha$  t  $\rightarrow$   $\beta$   $\rightarrow$   $\beta$ 
    val ranks :  $\alpha$  t  $\rightarrow$  rank list
    val min_max_rank :  $\alpha$  t  $\rightarrow$  rank  $\times$  rank
    val ranked : rank  $\rightarrow$   $\alpha$  t  $\rightarrow$  key list
  end

module type Graded_Map_Maker = functor (O : Graded_Ord)  $\rightarrow$ 
  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
  end

```

```

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 explicitly!

```

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 =
  Offspring.is_empty (Parents.find node dag)

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

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 →
            (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

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

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

$$\bar{p}_n = \begin{cases} -p_n & \text{for } 1 \leq n \leq n_{\text{in}} \\ p_n & \text{for } n_{\text{in}} + 1 \leq n \leq k \end{cases} \quad (5.2)$$

where momentum is conserved

$$\sum_{n=1}^k \bar{p}_n = 0 \quad (5.3)$$

below, we need the notion of ‘rank’ and ‘dimension’:

$$\dim(p) = k \quad (5.4a)$$

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

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

module type $T =$

sig
type t

Constructor: $(k, N) \rightarrow p = \sum_{n \in N} \bar{p}_n$ and $k = \dim(p)$ is the *overall* number of independent momenta, while $\text{rank}(p) = |N|$ 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 $of_ints : int \rightarrow int\ list \rightarrow t$

No two indices may be the same. Implementations 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 operations 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 → int list
val dim    : t → int
val rank   : t → int
```

Shortcuts: $singleton\ d\ p = of_ints\ d\ [p]$ and $zero\ d = of_ints\ d\ []$:

```
val singleton : int → int → t
val zero      : int → t
```

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

```
val compare : t → t → int
```

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

```
val neg : t → t
```

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

$$abs\ p = abs\ q \iff p = p \vee p = -q \quad (5.5)$$

for momenta with $rank = dim/2$.

```
val abs : t → t
```

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

```
val add : t → t → t
val sub : t → t → t
```

Once more, but not raising exceptions this time:

```
val try_add : t → t → t option
val try_sub : t → t → t option
```

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

```
val less : t → t → bool
val lesseq : t → t → bool
```

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

$$\text{val } \text{try_fusion} : t \rightarrow t \rightarrow t \rightarrow (bool \times bool) \text{ option}$$

A textual representation for debugging:

$$\text{val } \text{to_string} : t \rightarrow \text{string}$$

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

$$e^+(p_1)e^-(p_2) \rightarrow W^-(p_3)\nu_e(p_4)e^+(p_5) \quad (5.6a)$$

$$W^-(p_3) \rightarrow d(p'_3)\bar{u}(p'_4) \quad (5.6b)$$

to

$$e^+(p_1)e^-(p_2) \rightarrow d(p_3)\bar{u}(p_4)\nu_e(p_5)e^+(p_6) \quad (5.7)$$

in narrow width approximation for the W^- .

$$\text{val } \text{split} : \text{int} \rightarrow \text{int} \rightarrow t \rightarrow t$$

5.1.1 Scattering Kinematics

From here on, we assume scattering kinematics $\{1, 2\} \rightarrow \{3, 4, \dots\}$, i. e. $n_{\text{in}} = 2$.



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!

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

$$\text{val } \text{incoming} : t \rightarrow bool$$

$$p = \bar{p}_3 \vee p = \bar{p}_4 \vee \dots$$

$$\text{val } \text{outgoing} : t \rightarrow bool$$

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

$$\text{val } \text{timelike} : t \rightarrow bool$$

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

$$\text{val } \text{spacelike} : t \rightarrow bool$$

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

$$\text{val } \text{s_channel_in} : t \rightarrow bool$$

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

$$\text{val } \text{s_channel_out} : t \rightarrow bool$$

$$p = \bar{p}_1 + \bar{p}_2 \vee p = \bar{p}_3 + \bar{p}_4 + \dots + \bar{p}_n$$

```

    val s_channel : t → bool
 $\bar{p}_1 + \bar{p}_2 \rightarrow \bar{p}_3 + \bar{p}_4 + \dots + \bar{p}_n$ 
    val flip_s_channel_in : t → t
    val rcs : RCS.t
end
module Lists : T
module Bits : T
module Default : T

```

Wolfgang's funny tree codes:

$$(2^n, 2^{n-1}) \rightarrow (1, 2, 4, \dots, 2^{n-2}) \quad (5.8)$$

```

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 = "$Revision: 759$";
  RCS.date = "$Date: 2009-06-10 11:38:07 +0200 (Wed, 10 Jun 2009)$";
  RCS.author = "$Author: ohl$";
  RCS.source
    = "$URL: svn+ssh://jr-reuter@login.hepforge.org/hepforge/svn/whizard/trunk/src/omeg
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
  end

```

```

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
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
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 than 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

```

```

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

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
  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 information 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
  end

```

Bits 1...21 are used as a bitvector, indicating whether a particular momentum is included. Bits 22...26 represent the numbers of bits set in bits 1...21 and bits 27...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...51, 52...57, and 58...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

let incoming p =
  let p' = bits p in
  p' = mask_in1 ∨ p' = mask_in2

let outgoing p =
  rank p = 1 ∧ ¬ (incoming p)

```



```

let timelike p = (mask_in1 land p) = ((mask_in2 land p) lsr 1)
let spacelike p = ¬ (timelike p)

let s_channel_in p = bits p = 3
let s_channel_out p = ((mask (dim p)) land (lnot p)) = 3
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

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
  end

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 × '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

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

val to_string : ('flavor → string) → ('p → string) → ('flavor, 'p) t →
  string

exception Syntax_Error of string × int × 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 f p ps* returns **true** iff either the flavor *f* and momentum *p* match or *all* combinations of the momenta in *ps* are compatible, i.e. $\pm \sum p_i \leq q$

```

  val select_wf : selectors → (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)
```

Diagnostics:

```
val description : selectors → string option
```

end

```
module Make (M : Model.T) (P : Momentum.T) :
```

```
  T with type flavor = M.flavor_sans_color 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 → (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 description : selectors → string option
```

```
  end
```

```
module Make (M : Model.T) (P : Momentum.T) :
```

```
  (T with type flavor = M.flavor_sans_color and type p = P.t) =
```

```
  struct
```

```
    module CS = Cascade_syntax
```

```
    type flavor = M.flavor_sans_color
```

```
    type p = P.t
```

Since we have

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

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

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

```

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)

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_sans_color_of_string f, P.of_ints dim p)
  | CS.On_shell_not (f, p) →
    On_shell_not (List.map M.flavor_sans_color_of_string f, P.of_ints dim p)
  | CS.Off_shell (fs, p) →
    Off_shell (List.map M.flavor_sans_color_of_string fs, P.of_ints dim p)
  | CS.Off_shell_not (fs, p) →
    Off_shell_not (List.map M.flavor_sans_color_of_string fs, P.of_ints dim p)
  | CS.Gauss (f, p) →
    Gauss (List.map M.flavor_sans_color_of_string f, P.of_ints dim p)
  | CS.Gauss_not (f, p) →
    Gauss (List.map M.flavor_sans_color_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_sans_color_to_string fs))

let rec to_string = function
  | True →
    "true"
  | False →
    "false"
  | On_shell (fs, p) →
    P.to_string p ^ "□=□" ^ flavors_to_string fs
  | On_shell_not (fs, p) →
    P.to_string p ^ "□=□!" ^ flavors_to_string fs
  | Off_shell (fs, p) →
    P.to_string p ^ "□~□" ^ flavors_to_string fs
  | Off_shell_not (fs, p) →
    P.to_string p ^ "□~□!" ^ flavors_to_string fs
  | Gauss (fs, p) →
    P.to_string p ^ "□#□" ^ flavors_to_string fs
  | Gauss_not (fs, p) →
    P.to_string p ^ "□#□!" ^ flavors_to_string fs
  | Any_flavor p →
    P.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 : flavor → p → p list → bool;

```

```

    on_shell : flavor → p → bool;
    is_gauss : flavor → p → bool;
    description : string option }

let no_cascades =
  { select_p = (fun _ _ → true);
    select_wf = (fun _ _ _ → true);
    on_shell = (fun _ _ → false);
    is_gauss = (fun _ _ → false);
    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 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 f p p_in =
  let f' = M.conjugate_sans_color f in
  let rec to_select_wf' = function
    | True → true
    | False → false
    | On_shell (flavors, momentum)
    | Off_shell (flavors, momentum)
    | Gauss (flavors, momentum) →
      if p = momentum ∨ p = P.neg momentum then
        List.mem f flavors ∨ List.mem f' flavors
      else
        one_compatible p momentum ∧ all_compatible p p_in momentum
    | On_shell_not (flavors, momentum)
    | Off_shell_not (flavors, momentum)
    | Gauss_not (flavors, momentum) →
      if p = momentum ∨ p = P.neg momentum then
        ¬ (List.mem f flavors ∨ List.mem f' flavors)
      else
        one_compatible p momentum ∧ all_compatible p p_in momentum
    | 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_sans_color 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_sans_color 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

let to_selectors = function
  | True → no_cascades
  | c → { select_p = to_select_p c;
          select_wf = to_select_wf c;
          on_shell = to_on_shell c;
          is_gauss = to_gauss c;
          description = Some (to_string c) }

```

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 $SU(N_C)$ for the moment and use the $SU(N_C)$ completeness relation¹

$$T_{ij}^a T_{kl}^a = \frac{1}{2} \delta_{il} \delta_{jk} - \frac{1}{2N_C} \delta_{ij} \delta_{kl} \quad (7.1)$$

for the conventional normalization

$$\text{tr}(T_a T_b) = \frac{1}{2} \delta_{ab} \quad (7.2)$$

Therefore, particles are either color singlets or live in the defining representation of $SU(N)$: $SUN(|n|)$, its conjugate $SUN(-|n|)$ or in the adjoint representation of $SU(N)$: $AdjSUN(n)$.

```
type t = Singlet | SUN of int | AdjSUN of int
val conjugate : t → t
```

```
module type NC =
sig
  val nc : int
end
```

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
```

¹The corresponding formulae for the other Lie algebras (except E_8) can be found in [15].

```

    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
  end
module Flow : Flow

```

Case of Few Color Flows

Iff there are only few contributing color flows, it is more efficient to perform all calculations directly in a color flow basis.

7.1.3 *Evaluation*

For further processing we can either reduce internal gluons via the completeness relation or construct a trace of the square in one step. The first approach reduces part of the complexity from N^2 to N .

Algebraic Infrastructure

Allow for different implementations (symbolic and numeric) of the coefficient ring.

```

module type Ring =
  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_float : t → float
    val to_string : t → string
  end

module type Rational =
  sig
    include Ring
    val is_null : t → bool
    val make : int → int → t
  end
end

```

The coefficient ring required evaluating traces in $SU(N_C)$ and $SO(N_C)$ is the ring generated by the rational numbers \mathbf{Q} and the atoms N_C and N_C^{-1} . It is generated almost freely, but we take into account that $N_C \cdot N_C^{-1} = 1$.

```

module type Coeff =
  sig
    include Ring
    val is_null : t → bool

```

atom p creates the power N_C^p and *coef n d* creates the rational coefficient n/d . All possible coefficients can be generated by ring operations from these two.

```

    val atom : int → t
    val coeff : int → int → t
  end

```

The *Sum* signature describes a variant of rings. The main difference is that α *Sum.t* is polymorphic. One could think of using functors to implement a monomorphic *Sum.t*, but this would make dealing with *Sum.map* much harder. Therefore, we refrain from casting *Sum* into the *Ring* mold.

```

module type Sum =
  sig
    module C : Coeff
    type  $\alpha$  t
    val zero :  $\alpha$  t
    val atom :  $\alpha$  →  $\alpha$  t
    val scale : C.t →  $\alpha$  t →  $\alpha$  t
    val add :  $\alpha$  t →  $\alpha$  t →  $\alpha$  t
    val sub :  $\alpha$  t →  $\alpha$  t →  $\alpha$  t
    val mul : ( $\alpha$  →  $\beta$  →  $\gamma$ ) →  $\alpha$  t →  $\beta$  t →  $\gamma$  t
    val mulx : ( $\alpha$  →  $\beta$  →  $\gamma$  t) →  $\alpha$  t →  $\beta$  t →  $\gamma$  t →  $\gamma$  t
    val mulc : ( $\alpha$  →  $\beta$  → C.t) →  $\alpha$  t →  $\beta$  t → C.t → C.t
    val map : ( $\alpha$  →  $\beta$ ) →  $\alpha$  t →  $\beta$  t
    val eval : ( $\alpha$  → C.t) →  $\alpha$  t → C.t
    val to_string : ( $\alpha$  → string) →  $\alpha$  t → string
    val terms :  $\alpha$  t →  $\alpha$  list
  end

```

7.1.4 Color Flow Representation

```

module type Flows =
  sig
    module C : Coeff
    type  $\alpha$  t
    type  $\alpha$  wf
    val to_string : ( $\alpha$  → string) →  $\alpha$  t → string

```

of_amplitude root a calculates the sum of color flows corresponding to amplitude *a* with *root* as label for the particle at the root.

```

    val of_amplitude :  $\varepsilon$  → ( $\alpha$ ,  $\varepsilon$ ) amplitude →  $\varepsilon$  t

```

square_flip a1 a2 calculates the product of the colorflows *a1* and *a2*, where the duplicate gluon labels are flipped by *flip*.


```

val square : ( $\alpha \rightarrow \alpha$ )  $\rightarrow$   $\alpha$   $t \rightarrow \alpha$   $t \rightarrow \alpha$   $t$ 
val eval :  $\alpha$   $t \rightarrow C.t$ 
val eval_square : ( $\alpha \rightarrow \alpha$ )  $\rightarrow \alpha$   $t \rightarrow \alpha$   $t \rightarrow C.t$ 

type  $\alpha$  hash
val make_hash : unit  $\rightarrow \alpha$  hash
val eval_memoized :  $\alpha$  hash  $\rightarrow \alpha$   $t \rightarrow C.t$ 
val eval_square_memoized :  $\alpha$  hash  $\rightarrow (\alpha \rightarrow \alpha) \rightarrow \alpha$   $t \rightarrow \alpha$   $t \rightarrow C.t$ 

end

module Make_Flows ( $S : Sum$ ) : Flows with module  $C = S.C$ 
module Flows : Flows
i  $\times$  )

```

7.2 Implementation of *Color*

7.2.1 Quantum Numbers

```

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

let conjugate = function
| Singlet  $\rightarrow$  Singlet
| SUN  $n \rightarrow$  SUN ( $-n$ )
| AdjSUN  $n \rightarrow$  AdjSUN  $n$ 

module type NC =
sig
val nc : int
end

module NC3 = struct let nc = 3 end

```

7.2.2 Color Flows

```

module type Flow =
sig
type color
type  $t = color$  list  $\times$  color list
val rank :  $t \rightarrow int$ 
val of_list : int list  $\rightarrow$  color
val ghost : unit  $\rightarrow$  color
val to_lists :  $t \rightarrow int$  list list
val in_to_lists :  $t \rightarrow int$  list list
val out_to_lists :  $t \rightarrow int$  list list
val ghost_flags :  $t \rightarrow bool$  list
val in_ghost_flags :  $t \rightarrow bool$  list

```

```

    val out_ghost_flags : t → bool list
  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
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

```

Printing

```

let to_string_fold_functions fmt fmt_ext =
  let outer pfx s =
    (* pfx ^ ":" ^ *) s
  and ext pfx tag ext_tag =
    "<" ^ pfx ^ fmt_ext ext_tag ^ ">"
  and fuse pfx tag children =
    "<" ^ pfx ^ fmt tag ^ ">(" ^ String.concat ", " children ^ ")" in
  let fuse1 pfx tag child children =
    fuse pfx tag (child :: children)
  and fuse2 pfx tag child1 child2 children =
    fuse pfx tag (child1 :: child2 :: children)
  in
  { s_ext = ext "S";
    s_of_s = fuse "S";
    s_of_fc = fuse2 "S";
    s_final = outer "S";
    f_ext = ext "F";
    f_of_f = fuse1 "F";
    f_of_fa = fuse2 "F";
    f_final = outer "F";
    c_ext = ext "C";
    c_of_c = fuse1 "C";
    c_of_ca = fuse2 "C";
    c_final = outer "C";
    a_ext = ext "A";
    a_of_a = fuse1 "A";
    a_of_aa = fuse2 "A";
    a_of_fc = fuse2 "A";
    a_final = outer "A" }

```

```
let to_string fmt fmt_ext = fold (to_string_fold_functions fmt fmt_ext)
```

7.2.3 Evaluation

```
module type Ring =
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_float : t → float
  val to_string : t → string
end

module type Rational =
sig
  include Ring
  val is_null : t → bool
  val make : int → int → t
end

module type Coeff =
sig
  include Ring
  val is_null : t → bool
  val atom : int → t
  val coeff : int → int → t
end

module type Sum =
sig
  module C : Coeff
  type α t
  val zero : α t
  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 → β → γ) → α t → β t → γ t
  val mulx : (α → β → γ t) → α t → β t → γ t → γ t
  val mulc : (α → β → C.t) → α t → β t → C.t → C.t
  val map : (α → β) → α t → β t
  val eval : (α → C.t) → α t → C.t
  val to_string : (α → string) → α t → string
  val terms : α t → α list
end
```

7.2.4 *Color Flow Representation*

```

module type Flows =
  sig
    module C : Coeff
    type  $\alpha$  t
    type  $\alpha$  wf
    val to_string : ( $\alpha \rightarrow \text{string}$ )  $\rightarrow$   $\alpha$  t  $\rightarrow$  string
    val of_amplitude :  $\varepsilon \rightarrow (\alpha, \varepsilon)$  amplitude  $\rightarrow$   $\varepsilon$  t
    val square : ( $\alpha \rightarrow \alpha$ )  $\rightarrow$   $\alpha$  t  $\rightarrow$   $\alpha$  t  $\rightarrow$   $\alpha$  t
    val eval :  $\alpha$  t  $\rightarrow$  C.t
    val eval_square : ( $\alpha \rightarrow \alpha$ )  $\rightarrow$   $\alpha$  t  $\rightarrow$   $\alpha$  t  $\rightarrow$  C.t
    type  $\alpha$  hash
    val make_hash : unit  $\rightarrow$   $\alpha$  hash
    val eval_memoized :  $\alpha$  hash  $\rightarrow$   $\alpha$  t  $\rightarrow$  C.t
    val eval_square_memoized :  $\alpha$  hash  $\rightarrow$  ( $\alpha \rightarrow \alpha$ )  $\rightarrow$   $\alpha$  t  $\rightarrow$   $\alpha$  t  $\rightarrow$  C.t
  end

module Make_Flows (S : Sum) : Flows with module C = S.C =
  struct
    module C = S.C

    let one = S.C.unit
    let minus_one = S.C.neg one
    let two = S.C.coeff 2 1
    let half = S.C.coeff 1 2
    let nc = S.C.atom 1
    let minus_one_over_two_nc = S.C.neg (S.C.mul half (S.C.atom (-1)))

    type  $\alpha$  lines = ( $\alpha \times \alpha$ ) list

    let canonicalize_lines = List.sort compare lines
  end

```



expand_sum (and the functions using it) assumes too much about the physical representation of singlet terms!

```
let expand_sum sum = List.fold_left (S.mul (@)) (S.atom []) sum
```

The first member of these pairs is always the list of tags of external gluons contained in the amplitude. This information must be maintained in order to avoid duplicate application of the completeness relation for them.

```

type  $\alpha$  sng =  $\alpha$  list  $\times$  ( $\alpha$  lines) S.t
type  $\alpha$  fnd =  $\alpha$  list  $\times$  ( $\alpha \times \alpha$  lines) S.t
type  $\alpha$  cjg =  $\alpha$  list  $\times$  ( $\alpha \times \alpha$  lines) S.t
type  $\alpha$  adj =  $\alpha$  list  $\times$  ( $\alpha \times \alpha \times \alpha$  lines) S.t

type  $\alpha$  t =  $\alpha$  sng

type  $\alpha$  wf =
  | S of  $\alpha$  sng
  | F of  $\alpha$  fnd

```

```

| C of  $\alpha$  cjg
| A of  $\alpha$  adj

let ext_s tag' tag = ([], S.atom [])
let ext_f tag' tag = ([], S.atom (tag, []))
let ext_c tag' tag = ([], S.atom (tag, []))
let ext_a tag' tag = ([tag], S.atom (tag, tag, []))

```

Things are trivial, as long as there are no colors at all or all colors are coupled to singlets.

```

let merge_s tag sngs =
  let gluons, sum = List.split sngs in
  (List.concat gluons, expand_sum sum)

let mul_s_fc (f, lf) (c, lc) = canonicalize ((f, c) :: lf @ lc)

let merge_s_fc tag (gluons_f, sum_f) (gluons_c, sum_c) sngs =
  let gluons, sum = List.split sngs in
  (gluons_f @ gluons_c @ List.concat gluons,
   expand_sum (S.mul mul_s_fc sum_f sum_c :: sum))

```

Things remain simple, as long as colored particles emit and absorb only colorless particles:

```

let merge1 mul (gluons1, sum1) sngs =
  let gluons, sum = List.split sngs in
  (gluons1 @ List.concat gluons, S.mul mul sum1 (expand_sum sum))

let mul_f (f, lf) l = (f, canonicalize (lf @ l))
let mul_c (c, lc) l = (c, canonicalize (lc @ l))
let mul_a (f, c, la) l = (f, c, canonicalize (la @ l))

let merge_f tag f sngs = merge1 mul_f f sngs
let merge_c tag c sngs = merge1 mul_c c sngs
let merge_a tag a sngs = merge1 mul_a a sngs

```

We have only one way to emit a gluon from a quark anti-quark current:

```

let mul_a_fc (f, lf) (c, lc) = (f, c, canonicalize (lf @ lc))

let merge_a_fc tag (gluons_f, sum_f) (gluons_c, sum_c) sngs =
  let gluons, sum = List.split sngs in
  (gluons_f @ gluons_c @ List.concat gluons,
   S.mul mul_a (S.mul mul_a_fc sum_f sum_c) (expand_sum sum))

```

Using the $SU(N_C)$ completeness relation

$$T_{ij}^a T_{kl}^a = \frac{1}{2} \delta_{il} \delta_{jk} - \frac{1}{2N_C} \delta_{ij} \delta_{kl} \quad (7.3)$$

for the conventional normalization

$$\text{tr}(T_a T_b) = \frac{1}{2} \delta_{ab} \quad (7.4)$$

```

let merge2 mul mulx (gluons1, sum1) (gluons2, sum2) sngs =
  let gluons, sum = List.split sngs in

```

```

(gluons1 @ gluons2 @ List.concat gluons,
 S.mul mul (S.mulx mulx sum1 sum2 S.zero) (expand_sum sum))

let absorb_glue mul1 mul2 q (af, ac, _ as a) =
  let q' = S.atom q
  and a' = S.atom a in
  if af = ac then
    S.mul mul1 q' a'
  else
    S.add
      (S.scale half (S.mul mul1 q' a'))
      (S.scale minus_one_over_two_nc (S.mul mul2 q' a'))

let mul_fa_1 (f, lf) (af, ac, la) = (af, canonicalize ((f, ac) :: lf @ la))
let mul_fa_2 (f, lf) (af, ac, la) = (f, canonicalize ((af, ac) :: lf @ la))

let mul_ca_1 (c, lc) (af, ac, la) = (ac, canonicalize ((af, c) :: lc @ la))
let mul_ca_2 (c, lc) (af, ac, la) = (c, canonicalize ((af, ac) :: lc @ la))

let merge_fa tag f a sngs =
  merge2 mul_f (absorb_glue mul_fa_1 mul_fa_2) f a sngs

let merge_ca tag c a sngs =
  merge2 mul_c (absorb_glue mul_ca_1 mul_ca_2) c a sngs
    
```

The fun starts here, but the completeness relation turns out to be surprisingly simple. The $1/N_C$ -terms cancel always due to the antisymmetry ...

```

let mul_aa_1 (f1, c1, l1) (f2, c2, l2) =
  (f1, c2, canonicalize ((f2, c1) :: l1 @ l2))
let mul_aa_2 (f1, c1, l1) (f2, c2, l2) =
  (f2, c1, canonicalize ((f1, c2) :: l1 @ l2))
    
```

... and only the prefactor changes if we do apply the completeness relation for internal gluons or don't for external gluons:

$$f_{abc} = -2i \operatorname{tr} ([T_a, T_b] T_c) \quad (7.5a)$$

$$= 2 \left(\text{diagram 1} \right) - 2 \left(\text{diagram 2} \right) \quad (7.5b)$$

Both incoming gluons external, i.e. the completeness relation is never applied

$$= 2 \left(\text{diagram 1} \right) - 2 \left(\text{diagram 2} \right) \quad (7.6)$$

The right incoming gluon is external and the left internal, i.e. the completeness relation is only applied to the left

$$\begin{aligned}
 & \text{Diagram: A vertex with one external gluon line (wavy) and two internal gluon lines (curly).} \\
 &= \left(\text{Diagram: Triangle loop with two internal gluon lines and one external gluon line.} \right) - \frac{1}{N_C} \left(\text{Diagram: Triangle loop with two internal gluon lines and one external gluon line, with a self-energy insertion on the right internal line.} \right) \\
 &- \left(\text{Diagram: Triangle loop with two internal gluon lines and one external gluon line, with a self-energy insertion on the left internal line.} \right) + \frac{1}{N_C} \left(\text{Diagram: Triangle loop with two internal gluon lines and one external gluon line, with a self-energy insertion on the right internal line.} \right) \\
 &= \left(\text{Diagram: Triangle loop with two internal gluon lines and one external gluon line.} \right) - \left(\text{Diagram: Triangle loop with two internal gluon lines and one external gluon line, with a self-energy insertion on the right internal line.} \right) \quad (7.7)
 \end{aligned}$$

The left incoming gluon is external and the right internal, i. e. the completeness relation is only applied to the right

$$\begin{aligned}
 & \text{Diagram: A vertex with two external gluon lines (wavy) and one internal gluon line (curly).} \\
 &= \left(\text{Diagram: Triangle loop with two internal gluon lines and one external gluon line.} \right) - \frac{1}{N_C} \left(\text{Diagram: Triangle loop with two internal gluon lines and one external gluon line, with a self-energy insertion on the right internal line.} \right) \\
 &- \left(\text{Diagram: Triangle loop with two internal gluon lines and one external gluon line, with a self-energy insertion on the left internal line.} \right) + \frac{1}{N_C} \left(\text{Diagram: Triangle loop with two internal gluon lines and one external gluon line, with a self-energy insertion on the right internal line.} \right) \\
 &= \left(\text{Diagram: Triangle loop with two internal gluon lines and one external gluon line.} \right) - \left(\text{Diagram: Triangle loop with two internal gluon lines and one external gluon line, with a self-energy insertion on the right internal line.} \right) \quad (7.8)
 \end{aligned}$$

Both incoming gluons internal, i. e. the completeness relation is applied twice and as we have seen it corresponds to a factor of $1/2$ each time.

$$\begin{aligned}
 & \text{Diagram: A vertex with three internal gluon lines (curly).} \\
 &= \frac{1}{2} \left(\text{Diagram: Triangle loop with two internal gluon lines and one external gluon line.} \right) - \frac{1}{2} \left(\text{Diagram: Triangle loop with two internal gluon lines and one external gluon line, with a self-energy insertion on the right internal line.} \right) \quad (7.9)
 \end{aligned}$$


```

let fuse_aa (a1f, a1c, _ as a1) (a2f, a2c, _ as a2) =
  let a1 = S.atom a1
  and a2 = S.atom a2 in
  let aa = S.sub (S.mul mul_aa_1 a1 a2) (S.mul mul_aa_2 a1 a2) in
  match (a1f = a1c), (a2f = a2c) with
  | true, true → S.scale two aa
  | false, false → S.scale half aa
  | true, false | false, true → aa

let merge_aa tag a1 a2 snags = merge2 mul_a fuse_aa a1 a2 snags

let finalize_s s' t = t

let finalize_f f' (gluons, sum) =
  (gluons, S.map (fun (f, l) → canonicalize ((f, f') :: l)) sum)

let finalize_c c' (gluons, sum) =
  (gluons, S.map (fun (c, l) → canonicalize ((c', c) :: l)) sum)

let finalize_a a' (gluons, sum) =
  (a' :: gluons,
   S.map (fun (f, c, l) → canonicalize ((a', c) :: (f, a') :: l)) sum)

let of_amplitude_fold_functions root =
  { s_ext = ext_s;
    s_of_s = merge_s;
    s_of_fc = merge_s_fc;
    s_final = finalize_s root;
    f_ext = ext_f;
    f_of_f = merge_f;
    f_of_fa = merge_fa;
    f_final = finalize_f root;
    c_ext = ext_c;
    c_of_c = merge_c;
    c_of_ca = merge_ca;
    c_final = finalize_c root;
    a_ext = ext_a;
    a_of_a = merge_a;
    a_of_aa = merge_aa;
    a_of_fc = merge_a_fc;
    a_final = finalize_a root }

let of_amplitude root a =
  fold (of_amplitude_fold_functions root) a

exception Open_flow

```

It is crucial to apply the completeness relation

$$T_{ij}^a (T_{kl}^a)^* = T_{ij}^a T_{lk}^a = \frac{1}{2} \delta_{ik} \delta_{jl} - \frac{1}{2N_C} \delta_{ij} \delta_{kl} \quad (7.10)$$

either to the flow or the conjugated flow. It was appealing to apply it to the product, but this results in a disastrous quadratic behaviour!

$$T : (f, g)(g, c) \rightarrow (f, g)(g', c) \quad (7.11a)$$

```
let flip_fc_left flip is_gluon sum =
  S.map (List.map (fun (f, c) → ((if is_gluon f then flip f else f), c))) sum
```

$$T^* : (f, c) \rightarrow (c, f)$$

$$T^* : (f, g)(g, c) \rightarrow (c, g)(g, f) \rightarrow \frac{1}{2}(c, g')(g, f) - \frac{1}{2N_C}(g, g')(c, f) \quad (7.11b)$$

```
let find_gluon gluon pairs =
  let rec find_gluon_cf seen = function
    | [] → raise Not_found
    | (c, f as cf) :: cfs →
        if c = gluon then
          find_gluon_f f seen cfs
        else if f = gluon then
          find_gluon_c c seen cfs
        else
          find_gluon_cf (cf :: seen) cfs
  and find_gluon_f f seen = function
    | [] → invalid_arg "incomplete_gluon"
    | (c', f' as cf) :: cfs →
        if f' = gluon then
          ((c', f), List.rev_append seen cfs)
        else if c' = gluon then
          invalid_arg "duplicate_gluon"
        else
          find_gluon_f f (cf :: seen) cfs
  and find_gluon_c c seen = function
    | [] → invalid_arg "incomplete_gluon"
    | (c', f' as cf) :: cfs →
        if c' = gluon then
          ((c, f'), List.rev_append seen cfs)
        else if f' = gluon then
          invalid_arg "duplicate_gluon"
        else
          find_gluon_c c (cf :: seen) cfs in
  find_gluon_cf [] pairs

let flip1_cf_right flip gluon sum =
  let gluon' = flip gluon in
  S.add
    (S.scale half
     (S.map (List.map (fun (c, f) →
       (c, if f = gluon then gluon' else f))) sum))
    (S.scale minus_one_over_two_nc
     (S.map (fun pairs →
       let cf', cfs' = find_gluon gluon pairs in
       (gluon, gluon') :: cf' :: cfs') sum))

let flip_fc_right flip gluons sum =
  List.fold_right (flip1_cf_right flip) gluons
```

```
(S.map (List.map (fun (f, c) → (c, f))) sum)
```



Possible further optimizations:

- count and consume all *non-gluon* cycles *before* applying the completeness relation in *flip_fc*, then apply the completeness relation and count and consume the processed cycles

```
let square flip (gluons1, sum1) (gluons2, sum2) =
  assert (List.sort compare gluons1 = List.sort compare gluons2);
  ([,
    S.mul (fun l1 l2 → l1 @ l2)
      (flip_fc_left flip (fun g → List.mem g gluons1) sum1)
      (flip_fc_right flip gluons2 sum2))
```



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.

However ...



... (much to my surprise), the most expensive (i. e. inefficient) operation turned out to be an inefficient implementation of *square*.

```
let consume_cycle f0 c0 lines =
  let rec consume_cycle' c' seen = function
    | [] → raise Open_flow
    | (f, c) :: fc →
      if c = f0 then
        (f, c') :: List.rev_append seen fc
      else if f = c' then
        consume_cycle' c [] (List.rev_append seen fc)
      else
        consume_cycle' c' ((f, c) :: seen) fc in
  consume_cycle' c0 [] lines

let count_cycles lines =
  let rec count_cycles' acc = function
    | [] → acc
    | (f, c) :: fc →
      if f = c then
        count_cycles' (S.C.mul acc nc) fc
      else
        count_cycles' acc (consume_cycle f c fc) in
  count_cycles' one lines

let eval (gluons, sum) =
  assert (List.length gluons = 0);
  S.eval count_cycles sum
```

This deforestation is very helpful and conserves *a lot* of memory!

```
let eval_square_flip (gluons1, sum1) (gluons2, sum2) =
  assert (List.sort compare gluons1 = List.sort compare gluons2);
  S.mulc (fun l1 l2 → count_cycles (l1 @ l2))
    (flip_fc_left flip (fun g → List.mem g gluons1) sum1)
    (flip_fc_right flip gluons2 sum2)
  S.C.null
```

Memoization is precisely as useful as the lookup is efficient. Empirically, more than 99% of all lookups will be successful in complicated applications. However, the naive use of *Hashtbl* leads to *terrible* results which are more than an order of magnitude slower than naive evaluation.



On the other hand, using a polymorphic *Trie* doesn't slow down things significantly, but it doesn't appear to speed them up either.

```
module PT = Trie.MakePoly (Pmap.Tree)
type α hash = (α × α, S.C.t) PT.t ref
let make_hash () =
  ref PT.empty
let count_cycles_memoized hash lines =
  try
    PT.find compare lines !hash
  with
  | Not_found →
    let result = count_cycles lines in
    hash := PT.add compare lines result !hash;
    result
let eval_memoized hash (gluons, sum) =
  assert (List.length gluons = 0);
  S.eval (count_cycles_memoized hash) sum
let eval_square_memoized hash flip (gluons1, sum1) (gluons2, sum2) =
  assert (List.sort compare gluons1 = List.sort compare gluons2);
  S.mulc (fun l1 l2 → count_cycles_memoized hash (l1 @ l2))
    (flip_fc_left flip (fun g → List.mem g gluons1) sum1)
    (flip_fc_right flip gluons2 sum2)
  S.C.null
```

Printing Revisited

```
let gluons_to_string fmt = function
  | [] → ""
  | gluons → "<glue=" ^ String.concat "," (List.map fmt gluons) ^ ">"
let sng_to_string fmt lines =
  String.concat "/" (List.map (fun (f, c) → fmt f ^ ":" ^ fmt c) lines)
```

```

let to_string fmt (gluons, sum) =
  gluons_to_string fmt gluons ^ S.to_string (sng_to_string fmt) sum
end

```

7.2.5 Evaluation Revisited

```

module Make_Sum_Simple (C : Coeff) : Sum =
struct
  module C = C
  let one = C.coeff 1 1
  let minus_one = C.coeff (-1) 1
  type  $\alpha$  summand = { coeff : C.t; term :  $\alpha$  }

```



This implementation does not combine identical terms.

```

type  $\alpha$  t =  $\alpha$  summand list
let zero :  $\alpha$  t = []
let atom1 t = { coeff = one; term = t }
let atom t = [atom1 t]
let add x y = x @ y
let mul1 mul_term x y =
  { coeff = C.mul x.coeff y.coeff; term = mul_term x.term y.term }
let mul mul_term x y =
  Product.list2 (mul1 mul_term) x y
let scale c x =
  List.map (fun t → { t with coeff = C.mul c t.coeff }) x
let sub x y = x @ (scale minus_one y)
let mulx mul_term x y acc =
  Product.fold2 (fun x' y' →
    add (scale (C.mul x'.coeff y'.coeff) (mul_term x'.term y'.term))) x y acc
let multic mul_term x y acc =
  Product.fold2 (fun x' y' →
    C.add (C.mul (C.mul x'.coeff y'.coeff) (mul_term x'.term y'.term))) x y acc
let map f sum =
  List.map (fun t → { t with term = f t.term }) sum
let eval to_coeff x =
  List.fold_right (fun t → C.add (C.mul t.coeff (to_coeff t.term))) x C.null
let to_string fmt sum =
  "(" ^ String.concat "⊕"

```

```

      (List.map (fun s →
        C.to_string s.coeff ^ "*" [ ^ fmt s.term ^ "]" ) sum) ^ ") "
module M = Pmap.Tree
let terms sum =
  List.map fst
    (M.elements (List.fold_left (fun acc s → M.add compare s.term () acc) M.empty sum))
end
module Make_Sum (C : Coeff) : Sum =
struct
  module C = C
  let one = C.coeff 1 1
  type α summand = { coeff : C.t; term : α }
  module M = Pmap.Tree
  type α t = (α, C.t) M.t
  let zero = M.empty
  let atom t = M.singleton 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
  let fold2 f x y =
    M.fold (fun tx cx → M.fold (f tx cx) y) x
  let mul mul_term x y =
    fold2 (fun tx cx ty cy → insert1 C.add (mul_term tx ty) (C.mul cx cy))
      x y zero
  let mulx mul_term x y acc =
    fold2 (fun tx cx ty cy → add (scale (C.mul cx cy) (mul_term tx ty))) x y acc
  let mulc mul_term x y acc =
    fold2 (fun tx cx ty cy → C.add (C.mul (C.mul cx cy) (mul_term tx ty))) x y acc
  let map f sum = M.fold (fun t → insert1 C.add (f t)) sum M.empty
  let eval to_coeff x =
    M.fold (fun t c → C.add (C.mul c (to_coeff t))) x C.null
  let to_string fmt sum =
    "(" ^ String.concat "⊔⊔"
      (M.fold (fun t c acc →

```

```

      (C.to_string c ^ "[" ^ fmt t ^ "]" ) :: acc) sum [] ^ "]"
let terms sum =
  List.map fst (M.elements (M.fold (fun s _ → M.add compare s ()) sum M.empty))
end

```

Floating Point Arithmetic

Floatig point arithmetic for a fixed N_C is of course the fastest approach and it appears to be reasonable accurate in most cases.

```

module SUN_Float (NC : NC) : Coeff =
struct
  type t = float
  let nc = float NC.nc
  let is_null x = (x = 0.0)
  let null = 0.0
  let unit = 1.0
  let atom p = nc ** (float p)
  let coeff n d = float n /. float d
  let mul = ( *. )
  let add = ( +. )
  let sub = ( -. )
  let neg c = -. c
  let to_float c = c
  let to_string = string_of_float
end

```

Rational Arithmetic

```

module SUN_Rational (R : Rational) (NC : NC) : Coeff =
struct
  type t = R.t
  let null = R.null
  let unit = R.unit

  let is_null = R.is_null

  let nc = R.make NC.nc 1
  let one_over_nc = R.make 1 NC.nc

  let rec pow n p =
    if p < 0 then
      invalid_arg "pow"
    else if p = 0 then
      unit
    else
      R.mul n (pow n (pred p))

```

```

let atom p =
  if p < 0 then
    pow one_over_nc (-p)
  else if p = 0 then
    unit
  else
    pow nc p
let coeff = R.make
let mul = R.mul
let add = R.add
let sub = R.sub
let neg = R.neg
let to_float = R.to_float
let to_string = R.to_string
end

```

Symbolic Arithmetic

```

module SUN_Coeff (R : Rational) (NC : NC) : Coeff =
struct
  module IMap = Map.Make (struct type t = int let compare = compare end)
  type t = R.t IMap.t
  let null = IMap.empty
  let unit = IMap.add 0 R.unit null
  let is_null c = (c = IMap.empty)
  let atom p = IMap.add p R.unit null
  let coeff n d = IMap.add 0 (R.make n d) null
  let neg = IMap.map R.neg
  let insert1 binop p r c =
    let r' = binop (try IMap.find p c with Not_found → R.null) r in
    if R.is_null r' then
      IMap.remove p c
    else
      IMap.add p r' c
  let add x y = IMap.fold (insert1 R.add) x y
  let sub x y = IMap.fold (insert1 R.sub) y x
  let insert2 p1 r1 p2 r2 c =
    insert1 R.add (p1 + p2) (R.mul r1 r2) c
  let mul1 c2 p1 r1 c = IMap.fold (insert2 p1 r1) c2 c
  let mul c1 c2 = IMap.fold (mul1 c2) c1 IMap.empty
  let to_list c = IMap.fold (fun p r acc → (p, r) :: acc) c []
end

```



```

let to_string c =
  "(" ^ String.concat "⊕"
    (List.map
      (fun (p, r) →
        if p = 0 then
          R.to_string r
        else
          Printf.sprintf "%s*N^{%d}" (R.to_string r) p)
      (List.sort
        (fun (p1, _) (p2, _) → compare p2 p1)
        (to_list c))) ^ ")"

let nc = float NC.nc
let to_float c =
  IMap.fold (fun p r acc → (R.to_float r) *. nc ** (float p) +. acc) c 0.0
end

```

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 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_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

```

```

        else
            Printf.sprintf "(%d/%d)" n d
        end
module Flows = Make_Flows(Make_Sum(SUN_Coeff(Small_Rational)(NC3)))
slightly faster, but noticeably less precise: module Flows = Make_Flows(Make_Sum(SUN_Float(NC3)))
i × )

```

—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
```

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

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

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 (constant and Lorentz structure) and wave functions.

```

type rhs
type  $\alpha$  children
val sign : rhs  $\rightarrow$  int
val coupling : rhs  $\rightarrow$  constant Coupling.t
val coupling_tag : rhs  $\rightarrow$  string option

```

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



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

```

val children : rhs  $\rightarrow$  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  $\rightarrow$  wf
val rhs : fusion  $\rightarrow$  rhs list

```

and products at the keystones:

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

```

type braket
val bra : braket  $\rightarrow$  wf
val ket : braket  $\rightarrow$  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 amplitude : bool  $\rightarrow$  selectors  $\rightarrow$  flavor list  $\rightarrow$  flavor list  $\rightarrow$  amplitude
val dependencies : amplitude  $\rightarrow$  wf  $\rightarrow$  wf 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

$$\langle f_3, p_3, f_4, p_4, \dots | T | f_1, p_1, f_2, p_2 \rangle \quad (8.1a)$$

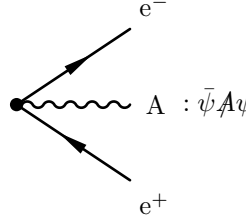
directly, O'Mega calculates the—equivalent, but more symmetrical—crossed amplitude

$$\langle \bar{f}_1, -p_1, \bar{f}_2, -p_2, f_3, p_3, f_4, p_4, \dots | T | 0 \rangle \quad (8.1b)$$

Internally, all flavors are represented by their charge conjugates

$$A(f_1, -p_1, f_2, -p_2, \bar{f}_3, p_3, \bar{f}_4, p_4, \dots) \quad (8.1c)$$

The correspondence of vertex and term in the lagrangian



$$(8.2)$$

suggests to denote the *outgoing* particle by the flavor of the *antiparticle* and the *outgoing antiparticle* by the flavor of the *particle*, since this choice allows to represent the vertex by a triple

$$\bar{\psi} A \psi : (e^+, A, e^-) \quad (8.3)$$

which is more intuitive than the alternative (e^-, A, e^+) . Also, when thinking in terms of building wavefunctions from the outside in, the outgoing *antiparticle* is represented by a *particle* propagator and vice versa¹. *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 brackets : amplitude → bracket 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
```

Diagnostics

```
val count_fusions : amplitude → int
val count_propagators : amplitude → int
val count_diagrams : amplitude → int
type coupling
```

¹Even 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 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 = 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 Colored Amplitudes

```

module type Colored =
  sig
    exception Mismatch
    val options : Options.t

    type flavor
    type amplitude
    type selectors
    type amplitudes

```

Construct all possible color flow amplitudes for a given process.

```

val amplitudes : bool → selectors → (flavor list × flavor list) list →
  amplitudes

val initialize_cache : string → unit

```

The list of all combinations of incoming and outgoing particles with a non-vanishing scattering amplitude.

val flavors : amplitudes → (flavor list × flavor list) list

The list of all combinations of incoming and outgoing particles that lead to a vanishing scattering amplitude.

val vanishing_flavors : amplitudes → (flavor list × flavor list) list

The list of all color flows with a nonvanishing scattering amplitude.

val color_flows : amplitudes → Color.Flow.t list

The list of all color flows that lead to a vanishing scattering amplitude.

val vanishing_color_flows : amplitudes → Color.Flow.t list

The list of all valid helicity combinations.

val helicities : amplitudes → (int list × int list) list

The rectangular nested list of all scattering amplitudes.

val processes : amplitudes → amplitude list list

A description of optional diagram selectors.

val constraints : amplitudes → string option

end

```
module type Colored_Maker = functor (Fusion_Maker : Maker) →
  functor (P : Momentum.T) →
    functor (Colorized_Model : Model.Colorized) →
      Colored with type flavor = Colorized_Model.M.flavor
      and type amplitude = Fusion_Maker(P)(Colorized_Model).amplitude
      and type selectors = Fusion_Maker(P)(Colorized_Model).selectors
```

```
module Colored : Colored_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  $\alpha$  children =  $\alpha$  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 = 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 = "$Revision: 1564$";
    RCS.date = "$Date: 2010-01-21 19:19:23 +0100 (Thu, 21 Jan 2010)$";
    RCS.author = "$Author: ohl$";
    RCS.source
      = "$URL: svn+ssh://jr-reuter@login.hepforge.org/hepforge/svn/whizard/trunk/src/omeg";
  }

module type T =
  sig
    val options : Options.t
    type wf
    type flavor
    val flavor : wf → flavor
    type p
    val momentum : wf → p
    val momentum_list : wf → int list
    val wf_tag : wf → string option
    type constant
    type rhs
    type  $\alpha$  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 amplitude : bool → selectors → flavor list → flavor list → amplitude
    val dependencies : amplitude → wf → wf Tree2.t
    val incoming : amplitude → flavor list
    val outgoing : amplitude → flavor list
  end

```



```

val externals : amplitude → wf list
val variables : amplitude → wf list
val fusions : amplitude → fusion list
val brackets : amplitude → bracket 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 count_fusions : amplitude → int
val count_propagators : amplitude → int
val count_diagrams : amplitude → int
type coupling
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 = 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) =

```

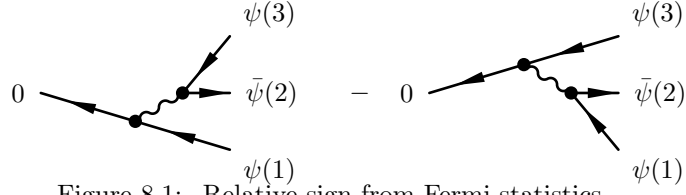


Figure 8.1: Relative sign from Fermi statistics.

```

struct
  let rcs = RCS.rename rcs_file "Fusion.Stat.Dirac()"
    [ "Fermi_statistics_for_Dirac_fermions" ]
  type flavor = M.flavor


$$\gamma_\mu \psi(1) G^{\mu\nu} \bar{\psi}(2) \gamma_\nu \psi(3) - \gamma_\mu \psi(3) G^{\mu\nu} \bar{\psi}(2) \gamma_\nu \psi(1) \quad (8.4)$$


  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 =
    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

```

$$\epsilon(\{(0,1), (2,3)\}) = -\epsilon(\{(0,3), (2,1)\}) \quad (8.5)$$

```

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 = 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  $\alpha$  children =  $\alpha$  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 The *Fusion.Make* Functor

```

module Tagged (Tagger : Tagger) (PT : Tuple.Poly)
  (Stat : Stat_Maker) (T : Topology.T with type  $\alpha$  children =  $\alpha$  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

  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" ]

  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 flavor = M.flavor
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*.

```

module Tags = Tagger(PT)

type p = P.t
type wf =
  { flavor : flavor;
    momentum : p;
    wf_tag : Tags.wf }

let flavor wf = wf.flavor
let flavor_sans_color wf = M.flavor_sans_color wf.flavor
let momentum wf = wf.momentum
let momentum_list wf = P.to_ints wf.momentum
let wf_tag_raw wf = wf.wf_tag
let wf_tag wf = Tags.wf_to_string (wf_tag_raw wf)

```

Operator insertions can be fused only if they are external.

```

let is_source wf =
  match M.propagator wf.flavor with
  | Only_Insertion → P.rank wf.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 future, we might want to have *Coupling* among the functor arguments. However, for the moment, *Coupling* is assumed to be comprehensive.

```

type sign = int
type coupling =
  { sign : sign;
    coupling : constant Coupling.t;
    coupling_tag : Tags.coupling }

type α children = α PT.t

```

This *must* be a pair matching the *edge* × *node children* pairs of *DAG.Forest*!

```

type rhs = coupling × wf children
let sign ({ sign = s }, _) = s
let coupling ({ coupling = c }, _) = c
let coupling_tag_raw ({ coupling_tag = t }, _) = t
let coupling_tag rhs = Tags.coupling_to_string (coupling_tag_raw rhs)
let children (_, wfs) = PT.to_list wfs

```



In the end, *PT.to_list* should become redundant!

```
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 = 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 = (flavor × (constant Coupling.t × 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 =
  ((flavor × flavor × flavor) × constant Coupling.vertex3 ×
   constant) list
  × ((flavor × flavor × flavor × flavor) × constant Coupling.vertex4 ×
   constant) list
  × (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 ())
let filename = Config.cache_prefix
let vertices_max_degree_flavors : vertices =
  match !vertices_cache with
  | None →
    begin match !cache_option with
    | Cache_Use →
      begin match VCache.maybe_read hash filename with
      | None →
        Printf.eprintf
          "␣>>>␣Initializing␣lookup␣table.␣␣This␣may␣take␣some␣time␣...␣";
        flush stderr;
        let result = vertices_nocache max_degree_flavors in
          VCache.write hash filename (M.rcs, result);
          vertices_cache := Some result;
          Printf.eprintf "done.␣<<<␣\n";
          flush stderr;
          result
        | Some (rcs, result) → result
      end
    | Cache_Overwrite →
        Printf.eprintf
          "␣>>>␣Overwriting␣lookup␣table.␣␣This␣may␣take␣some␣time␣...␣";
        flush stderr;
        let result = vertices_nocache max_degree_flavors in
          VCache.write hash filename (M.rcs, result);
          vertices_cache := Some result;
          Printf.eprintf "done.␣<<<␣\n";
          flush stderr;
          result
        | Cache_Ignore →
            Printf.eprintf
              "␣>>>␣Ignoring␣lookup␣table.␣␣This␣may␣take␣some␣time␣...␣";
            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
  end

```

Partitions

Vertices that are not crossing invariant need special treatment so that they're only generated for the correct combinations of momenta.



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 crossing 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.timelike (P.add q1 q2),
                       P.timelike (P.add q2 q3),
                       P.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
    end
  end

```

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) =
  ({ flavor = f1;
    momentum = P.of_ints dim p1;

```



```

wf_tag = Tags.null_wf },
Product.fold2 (fun (c, f) p acc →
  try
    if select_p
      (P.of_ints dim p1)
      (PT.to_list (PT.map (P.of_ints dim) p)) then begin
    if crossing c (PT.map (P.of_ints dim) p) then
      (c, PT.map2 (fun f' p' → { flavor = f';
                                momentum = P.of_ints dim p';
                                wf_tag = Tags.null_wf }) f p) :: acc
      else
        acc
    end 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 $2 \rightarrow 4$ 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

```

Once more, but without duplicates this time.

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

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

let wavefunctions t =
  let module WF =
    Set.Make (struct type t = wf let compare = order_wf end) in
  WF.elements (List.fold_left (fun set (wf1, wf23) →
    WF.add wf1 (List.fold_left (fun set' (_, wfs) →
      PT.fold_right WF.add wfs set') set wf23)) WF.empty 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 : (wf × stat × 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 flavor wfs
      and momenta = PT.map momentum wfs
      in
      let p = PT.fold_left_internal P.add momenta in
      List.fold_left
        (fun acc (f, c) →
          if select_wf (M.flavor_sans_color f) p (PT.to_list momenta)
            ∧ crossing 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
    ({ flavor = f;
      momentum = p;
      wf_tag = Tags.null_wf }, s,
    ({ sign = flip;
      coupling = c;
      coupling_tag = Tags.null_coupling }, wfs)) :: acc
  else
    acc)
  [] (fuse_rhs flavors)
with
  | P.Duplicate _ | S.Impossible → []
else
  []
module D = DAG.Make
  (DAG.Forest(PT)
   (struct type t = wf let compare = order_wf end)
   (struct type t = coupling let compare = compare end))

```



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* contains 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 × stat` is *not* a good idea because the variable *stat* makes it impossible to test for the existence of a particular *wf* in a *DAG*.



In summary, it seems that $(wf \times stat) \text{ list array} \times D.t$ should be replaced by $(wf \rightarrow stat) \times D.t$.

```

module GF =
  struct
    module Nodes =
      struct
        type t = wf
        module G = struct type t = int let compare = compare end
        let compare = order_wf
        let rank wf = P.rank (momentum wf)
      end
    module Edges = struct type t = 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
  end

```

```

    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)

  module Stat = Map.Make (struct type t = wf let compare = order_wf end)

```

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.

$$\begin{aligned}
 & \left[\{ \phi_1(p_1), \phi_2(p_2), \phi_3(p_3), \dots \}, \right. \\
 & \quad \{ \phi_{12}(p_1 + p_2), \phi'_{12}(p_1 + p_2), \dots, \phi_{13}(p_1 + p_3), \dots, \phi_{23}(p_2 + p_3), \dots \}, \\
 & \quad \dots \\
 & \quad \left. \{ \phi_{1\dots n}(p_1 + \dots + p_n), \phi'_{1\dots n}(p_1 + \dots + p_n), \dots \} \right] \quad (8.6)
 \end{aligned}$$

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) =
    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 : (wf × stat) list array × 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 tower' dag

```

```
(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, _) → D.add_node wf m) D.empty wfs
let mixed_fold_left f acc lists =
  Array.fold_left (List.fold_left f) acc lists
let fusion_tower height select_wf wfs : (wf → stat) × 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.add wf s m) Stat.empty tower in
  ((fun wf → Stat.find wf stats), dag)
```

Calculate the minimal tower of fusions that suffices for calculating the amplitude.

```
let minimal_fusion_tower n select_wf wfs : (wf → stat) × 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 : (wf → stat) × 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 significantly, because most fusions (including Fermi signs!) will be calculated before being rejected by *DAD().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 =
  D.fold_nodes (fun wf dag' →
    match M.goldstone wf.flavor with
    | Some (g, _) →
      let wf' = { wf with flavor = g } in
      if D.is_node wf' tower then begin
        D.harvest tower wf' dag'
      end else begin
        dag'
      end
  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 (flavor wf1))))
      (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 ∧ 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 ∧ D.is_node wf1 dag then
    match List.filter (test_rhs dag) pairs with
    | [] → acc
    | pairs' → (wf1, List.map (fun (c, wfs) →
      ({ sign = stat_keystone stats wf1 wfs;
        coupling = c;
        coupling_tag = Tags.null_coupling },
        wfs)) pairs') :: acc
  else
    acc
```

Amplitudes

```
type fusion = wf × rhs list

let lhs (l, _) = l
let rhs (_, r) = r

type bracket = wf × rhs list

let bra (b, _) = b
let ket (_, k) = k
```

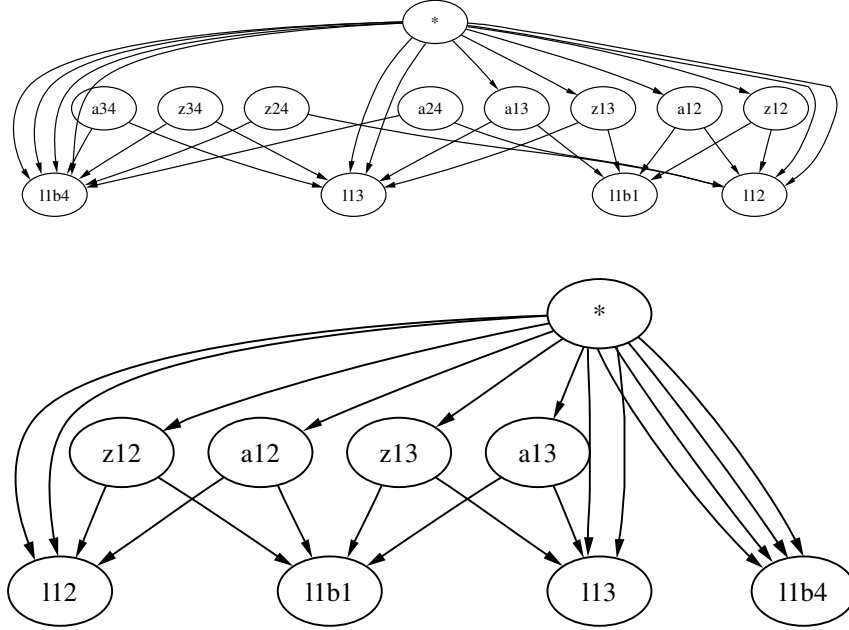


Figure 8.2: The DAGs for Bhabha scattering before and after weeding out unused nodes. The blatant asymmetry of these DAGs is caused by our prescription for removing doubling counting for an even number of external lines.

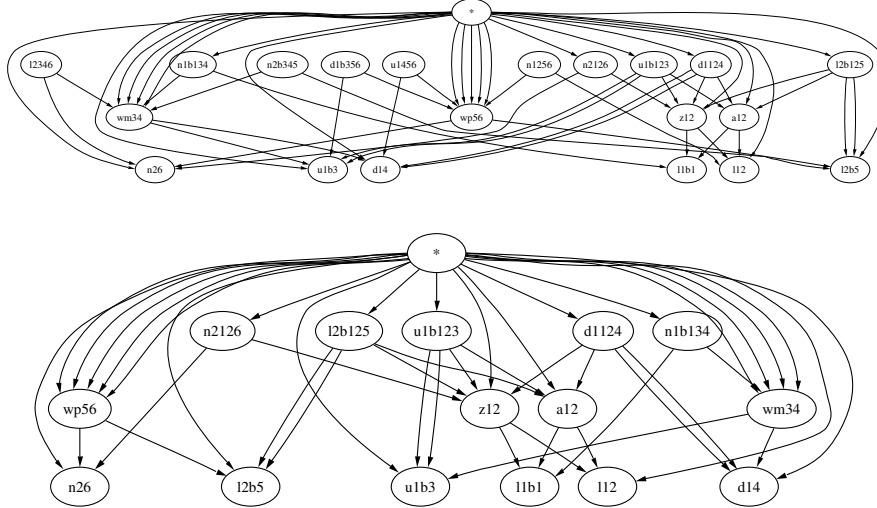


Figure 8.3: The DAGs for $e^+e^- \rightarrow u\bar{d}\mu^-\bar{\nu}_\mu$ before and after weeding out unused nodes.

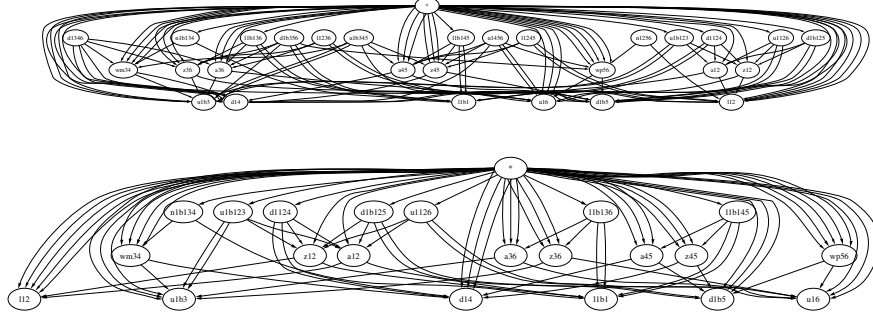


Figure 8.4: The DAGs for $e^+e^- \rightarrow u\bar{d}d\bar{u}$ before and after weeding out unused nodes.

```

type amplitude =
  { fusions : fusion list;
    brackets : bracket 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 Tree2.t;
    fusion_tower : D.t;
    fusion_dag : D.t }

```

```

module C = Cascade.Make(M)(P)
type selectors = C.selectors

let incoming a = a.incoming
let outgoing a = a.outgoing
let externals a = a.externals
let fusions a = a.fusions
let brackets a = a.brackets
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 allowed amplitude =
  match brackets amplitude with
  | [] → false
  | _ → true

let external_wfs n particles =
  List.map (fun (f, p) →

```



```
({ flavor = f;
  momentum = P.singleton n p;
  wf_tag = Tags.null_wf },
 stat f p)) particles
```

Main Function

```
module WFFMap = Map.Make (struct type t = wf let compare = compare end)

map_amplitude_wfs f a applies the function  $f : wf \rightarrow wf$  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) in
  and map_fusion (lhs, rhs) = (f lhs, List.map map_rhs rhs) in
  let map_dag = D.map f (fun node rhs → map_rhs rhs) in
  let tower = map_dag a.fusion_tower
  and dag = map_dag a.fusion_dag in
  let dependencies_map =
    D.fold (fun wf _ → WFFMap.add wf (D.dependencies dag wf)) dag WFFMap.empty in
  { fusions = List.map map_fusion a.fusions;
    brackets = List.map map_braket a.brackets;
    on_shell = a.on_shell;
    is_gauss = a.is_gauss;
    constraints = a.constraints;
    incoming = a.incoming;
    outgoing = a.outgoing;
    externals = List.map f a.externals;
    symmetry = a.symmetry;
    dependencies = (fun wf → WFFMap.find wf dependencies_map);
    fusion_tower = tower;
    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_wf = C.select_wf selectors in
let select_p = C.select_p selectors 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 brackets =
  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
    D.harvest_list tower (wavefunctions brackets) in

```

Remove the leaf nodes of the DAG, corresponding to external lines.

```

let fusions =
  List.filter (function (_, []) → false | _ → true) (D.lists dag) in

```

Calculate the symmetry factor for identical particles in the final state.

```

let symmetry =
  Combinatorics.symmetry (List.map M.flavor_sans_color fouts) in

```

```

let dependencies_map =
  D.fold (fun wf _ → WFFMap.add wf (D.dependencies dag wf)) dag WFFMap.empty in

```

Finally: package the results:

```

{ fusions = fusions;
  brackets = brackets;
  on_shell = (fun wf → C.on_shell selectors (flavor_sans_color wf) (momentum wf));
  is_gauss = (fun wf → C.is_gauss selectors (flavor_sans_color wf) (momentum wf));
  constraints = C.description selectors;
  incoming = fin;
  outgoing = fout;
  externals = List.map fst wfs;
  symmetry = symmetry;
  dependencies = (fun wf → WFFMap.find wf dependencies_map);
  fusion_tower = tower;
  fusion_dag = dag }

```

```

let initialize_cache dir =
  Printf.eprintf ">>>Initializing lookup table. This may take some time...\n";
  flush stderr;
  VCache.write_dir hash dir filename
  (M.rcs, vertices_nocache (M.max_degree ()) (M.flavors()));
  Printf.eprintf "done.<<<\n"

```

Diagnostics

```

let count_propagators a =
  List.length a.fusions

let count_fusions a =
  List.fold_left (fun n (_, a) → n + List.length a) 0 a.fusions
  + List.fold_left (fun n (_, t) → n + List.length t) 0 a.brackets
  + List.length a.brackets

```



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 + D.count_trees wf1 a.fusion_dag ×
    (List.fold_left (fun n' (_, wfs) →
      n' + PT.fold_left (fun n'' wf →
        n'' × D.count_trees wf a.fusion_dag) 1 wfs) 0 wf23))
    0 a.brackets

exception Impossible

```



We still need to perform the appropriate charge conjugations so that we get the correct flavors for the external tree representation.

```

let forest' a =
  let below wf = D.forest_memoized wf a.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.brackets

let cross wf =
  { flavor = M.conjugate wf.flavor;
    momentum = P.neg wf.momentum;
    wf_tag = wf.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 =
  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.fusion_dag in
    (ThoList.flatmap (fun (_, wfs) →
      Product.list List.flatten
        (PT.to_list (PT.map (fun wf →
          poles_wf1 @ poles_beneath wf a.fusion_dag) wfs)))
      wf23))
    a.brackets

let s_channel a =
  let module WF =
    Set.Make (struct type t = wf let compare = order_wf end) in
    WF.elements (ThoList.fold_right2
      (fun wf wfs →
        if P.timelike (momentum wf) then
          WF.add wf wfs
        else
          wfs) (poles a) WF.empty)

```



This should be much faster! Is it correct? Is it faster indeed?

```

let poles' a =
  List.map lhs a.fusions

let s_channel a =
  let module WF =
    Set.Make (struct type t = wf let compare = order_wf end) in
    WF.elements (List.fold_right
      (fun wf wfs →
        if P.timelike (momentum wf) then
          WF.add wf wfs
        else
          wfs) (poles' a) WF.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 =
  M.flavor_symbol (flavor wf) ^
  String.concat "" (List.map p2s (momentum_list wf))

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)
    (D.fold_nodes (fun wf → add_to_list (P.rank (momentum wf)) wf)
      dag Int.empty) []

let dag_to_dot ch brackets dag =
  Printf.fprintf ch "digraph_Ω_{\n";
  D.iter_nodes (fun wf →
    Printf.fprintf ch "_{\n" [label=wf];\n"
      (variable wf) (variable wf)) dag;
  List.iter (fun (_, wfs) →
    Printf.fprintf ch "{rank=same;\n"
      List.iter (fun n →
        Printf.fprintf ch "\n" (variable n)) wfs;
        Printf.fprintf ch "};\n") (classify_nodes dag);
    List.iter (fun n →
      Printf.fprintf ch "\n->\n" (variable n))
      (flatten_keystones brackets);
    D.iter (fun n (_, ns) →
      let p = variable n in
      PT.iter (fun n' →
        Printf.fprintf ch "\n->\n" (variable n')) ns) dag;
      Printf.fprintf ch "}\n"

let tower_to_dot ch a =
  dag_to_dot ch a.brackets a.fusion_tower

let amplitude_to_dot ch a =
  dag_to_dot ch a.brackets a.fusion_dag

let rcs_list = [D.rcs; T.rcs; P.rcs; rcs]

end

module Make = Tagged(No_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.Dirac()"
    [ "Fermi_statistics_for_Dirac_fermions" ]

  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 additional one due to the permutation 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 Colored Amplitudes

```

module type Colored =
  sig
    exception Mismatch
    val options : Options.t
    type flavor
    type amplitude
    type selectors
    type amplitudes
    val amplitudes : bool → selectors → (flavor list × flavor list) list →
    amplitudes
    val initialize_cache : string → unit
    val flavors : amplitudes → (flavor list × flavor list) list
    val vanishing_flavors : amplitudes → (flavor list × flavor list) list
    val color_flows : amplitudes → Color.Flow.t list
    val vanishing_color_flows : amplitudes → Color.Flow.t list
    val helicities : amplitudes → (int list × int list) list
    val processes : amplitudes → amplitude list list
    val constraints : amplitudes → string option
  end

module type Colored_Maker = functor (Fusion_Maker : Maker) →
  functor (P : Momentum.T) →
    functor (Colorized_Model : Model.Colorized) →
      Colored with type flavor = Colorized_Model.M.flavor
      and type amplitude = Fusion_Maker(P)(Colorized_Model).amplitude
      and type selectors = Fusion_Maker(P)(Colorized_Model).selectors

module Colored (Fusion_Maker : Maker) (P : Momentum.T) (CM : Model.Colorized) =
  struct
    exception Mismatch

    type progress_mode =
      | Quiet

```

```

    | Channel of out_channel
    | File of string

let progress_option = ref Quiet

module F = Fusion_Maker(P)(CM)
module C = Cascade.Make(CM)(P)

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 = CM.flavor_sans_color
type p = F.p
type amplitude = F.amplitude
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 : (flavor list × flavor list) list;
    vanishing_flavors : (flavor list × flavor list) list;
    color_flows : Color.Flow.t list;
    vanishing_color_flows : Color.Flow.t list;
    helicities : (int list × int list) list;
    processes : amplitude list list;
    constraints : string option }

let flavors a = a.flavors
let vanishing_flavors a = a.vanishing_flavors
let color_flows a = a.color_flows
let vanishing_color_flows a = a.vanishing_color_flows
let helicities a = a.helicities
let processes a = a.processes
let constraints a = a.constraints

let sans_colors f =
  List.map CM.flavor_sans_color f

let colors (fin, fout) =
  List.map CM.M.color (fin @ fout)

let process_to_string fin fout =
  String.concat "␣" (List.map CM.flavor_to_string fin)
  ^ "␣->␣" ^ String.concat "␣" (List.map CM.flavor_to_string fout)

let count_processes colored_processes =
  List.fold_left (+) 0 (List.map List.length colored_processes)

```

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)

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 (CM.M.lorentz f)

let hs_of_flavors (fin, fout) =
  (List.map hs_of_flavor fin, List.map hs_of_flavor fout)

let helicity_table flavors =
  let hs = List.map hs_of_flavors flavors 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))

```

Calculate All The Amplitudes

```

let amplitudes goldstones select_wf processes =
  if ¬ (ThoList.homogeneous (List.map hs_of_flavors processes)) then
    invalid_arg "Fusion.Colored.amplitudes: incompatible helicities";
  if ¬ (ThoList.homogeneous (List.map colors processes)) then
    invalid_arg "Fusion.Colored.amplitudes: incompatible color representations";
  let colored_processes =
    List.map (fun (fi, fo) → CM.amplitude fi fo) processes in
  let progress =
    match !progress_option with
    | Quiet → Progress.dummy
    | Channel oc → Progress.channel oc (count_processes colored_processes)
    | File name → Progress.file name (count_processes colored_processes) in
  let all =
    List.map
      (List.map (fun (fi, fo) →
        Progress.begin_step progress (process_to_string fi fo);
        let amp = F.amplitude goldstones select_wf fi fo in
        Progress.end_step progress "done";
        amp)) colored_processes in
    Progress.summary progress "all processes done";
  let allowed_flows, forbidden_flows =
    List.partition (List.exists F.allowed) (ThoList.transpose all) in
  let allowed_flows =
    ThoList.transpose allowed_flows
  and forbidden_flows =
    ThoList.transpose forbidden_flows in
  let color_flows =
    match allowed_flows with
    | [] → []
    | flows :: _ → List.map (fun a → CM.flow (F.incoming a) (F.outgoing a)) flows
  and vanishing_color_flows =
    match forbidden_flows with
    | [] → []
    | flows :: _ → List.map (fun a → CM.flow (F.incoming a) (F.outgoing a)) flows in
  let allowed, forbidden =
    List.partition (List.exists F.allowed) allowed_flows in
  let flavors =
    List.map
      (fun a →
        let a' = List.hd a in
        (sans_colors (F.incoming a'), sans_colors (F.outgoing a')))
      allowed
  and vanishing_flavors =

```

```

    List.map
      (fun a →
        let a' = List.hd a in
        (sans_colors (F.incoming a'), sans_colors (F.outgoing a')))
      forbidden in

let helicities =
  helicity_table flavors in

{ flavors = flavors;
  vanishing_flavors = vanishing_flavors;
  color_flows = color_flows;
  vanishing_color_flows = vanishing_color_flows;
  helicities = helicities;
  processes = allowed;
  constraints = C.description select_wf }


let initialize_cache = F.initialize_cache


end


```

—9—

LORENTZ REPRESENTATIONS, COUPLINGS,
MODELS AND TARGETS

 *Interface `coupling.mli` unavailable!*

 *Interface `model.mli` unavailable!*

 *Interface `target.mli` unavailable!*

—10—

COLORIZATION

10.1 *Interface of Colorize*

10.1.1 ...

```
module type Flows =
  sig
    val max_lines : int
  end

module It (F : Flows) (M : Model.T) :
  Model.Colorized with module M = M

module Gauge (F : Flows) (M : Model.Gauge) :
  Model.Colorized_Gauge with module M = M

module Dynamical (M : Model.T) :
  Model.Colorized with module M = M
```



Also implement module *Trivial* (*M* : *Model.T*) : *Model.Colorized* with module *M* = *M* for handling completely colorless models more efficiently.

10.2 *Implementation of Colorize*

10.2.1 *(Statically) Colorizing a Monochrome Model*

```
module type Flows =
  sig
    val max_lines : int
  end

module It (F : Flows) (M : Model.T) =
  struct
    module M = M
    open Coupling
    module C = Color
```

```

let incomplete s =
  failwith ("Colorize.It()." ^ s ^ "not done yet!")

let incomplete s =
  Printf.eprintf "WARNING: Colorize.It().%snot done yet!\n" s;
  []

let su0 s =
  invalid_arg ("Colorize.It()." ^ s ^ ": found SU(0)!")

let colored_vertex s =
  invalid_arg ("Colorize.It()." ^ s ^ ": colored vertex!")

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 if c > F.max_lines then
    invalid_arg ("Colorize.It()." ^ s ^ ": color flow # too large")
  else
    c

type flavor =
| White of M.flavor
| CF_in of M.flavor × int
| CF_out of M.flavor × int
| CF_io of M.flavor × int × int
| 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

```

For the propagator we cannot use pullback because we have to add the case of the color singlet propagator by hand.

```

let colorize_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. *)
| Prop_Col_Scalar | Prop_Col_Feynman | Prop_Col_Majorana |
Prop_Col_Unitarity
  → failwith ("Colorize.It().colorize_propagator: already colored particle!")
  | _ → failwith ("Colorize.It().colorize_propagator: impossible!")

let propagator = function
| CF_aux f → colorize_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 permute_triple (a, b, c) =
  List.map
    (function
      | [a'; b'; c'] → (a', b', c')
      | _ → failwith "Colorize.It().permute_triple:␣internal␣error")
    (Combinatorics.permute [a; b; c])

let permute_quadruple (a, b, c, d) =
  List.map
    (function
      | [a'; b'; c'; d'] → (a', b', c', d')
      | _ → failwith "Colorize.It().permute_quadruple:␣internal␣error")
    (Combinatorics.permute [a; b; c; d])

let max_degree = M.max_degree

let color_flows =
  ThoList.range 1 F.max_lines

let color_flow_pairs =
  ThoList.flatmap
    (function
      | [c1; c2] → [(c1, c2); (c2, c1)]
      | _ → failwith "Colorize.It().color_flow_pairs:␣internal␣error")
    (Combinatorics.choose 2 color_flows)

let color_flow_triples =
  List.map
    (function
      | [c1; c2; c3] → (c1, c2, c3)
      | _ → failwith "Colorize.It().color_flow_triples:␣internal␣error")
    (Combinatorics.choose 3 color_flows)

let color_flow_quadruples =
  List.map
    (function
      | [c1; c2; c3; c4] → (c1, c2, c3, c4)
      | _ → failwith "Colorize.It().color_flow_quadruples:␣internal␣error")
    (Combinatorics.choose 4 color_flows)

let colorize_flavor f =
  match M.color f with
  | C.Singlet → [White f]
  | C.SUN nc →
    if nc > 0 then
      List.map (fun c → CF_in (f, c)) color_flows
    else if nc < 0 then
      List.map (fun c → CF_out (f, c)) color_flows
    else
      su0 "colorize_flavor"
  | C.AdjSUN _ →
    CF_aux f :: (List.map (fun (c1, c2) → CF_io (f, c1, c2)) color_flow_pairs)

```

```

let flavors () =
  ThoList.flatmap colorize_flavor (M.flavors ())

let external_flavors () =
  List.map
    (fun (name, flist) → (name, ThoList.flatmap colorize_flavor flist))
    (M.external_flavors ())

let parameters = M.parameters

module Fusion = Modeltools.Fusions (struct
  type f = flavor
  type c = constant
  let compare = compare
  let conjugate = conjugate
end)

```

Auxiliary functions

```

let mult_vertex3 fac = function
| FBF (c, fb, coup, f) → FBF ((fac × c), fb, coup, f)
| PBP (c, fb, coup, f) → PBP ((fac × c), fb, coup, f)
| BBB (c, fb, coup, f) → BBB ((fac × c), fb, coup, f)
| GBG (c, fb, coup, f) → GBG ((fac × c), fb, coup, f)
| Gauge_Gauge_Gauge c → Gauge_Gauge_Gauge (fac × c)
| Aux_Gauge_Gauge c → Aux_Gauge_Gauge (fac × c)
| Scalar_Vector_Vector c → Scalar_Vector_Vector (fac × c)
| Aux_Vector_Vector c → Aux_Vector_Vector (fac × c)
| Aux_Scalar_Vector c → Aux_Scalar_Vector (fac × c)
| Scalar_Scalar_Scalar c → Scalar_Scalar_Scalar (fac × c)
| Aux_Scalar_Scalar c → Aux_Scalar_Scalar (fac × c)
| Vector_Scalar_Scalar c → Vector_Scalar_Scalar (fac × c)
| Graviton_Scalar_Scalar c → Graviton_Scalar_Scalar (fac × c)
| Graviton_Vector_Vector c → Graviton_Vector_Vector (fac × c)
| Graviton_Spinor_Spinor c → Graviton_Spinor_Spinor (fac × c)
| Dim4_Vector_Vector_Vector_T c → Dim4_Vector_Vector_Vector_T (fac ×
c)
| Dim4_Vector_Vector_Vector_L c → Dim4_Vector_Vector_Vector_L (fac ×
c)
| Dim4_Vector_Vector_Vector_T5 c → Dim4_Vector_Vector_Vector_T5 (fac ×
c)
| Dim4_Vector_Vector_Vector_L5 c → Dim4_Vector_Vector_Vector_L5 (fac ×
c)
| Dim6_Gauge_Gauge_Gauge c → Dim6_Gauge_Gauge_Gauge (fac ×
c)
| Dim6_Gauge_Gauge_Gauge_5 c → Dim6_Gauge_Gauge_Gauge_5 (fac ×
c)
| Aux_DScalar_DScalar c → Aux_DScalar_DScalar (fac × c)
| Aux_Vector_DScalar c → Aux_Vector_DScalar (fac × c)
| Dim5_Scalar_Gauge2 c → Dim5_Scalar_Gauge2 (fac × c)

```

```

| Dim5_Scalar_Gauge2_Skew c → Dim5_Scalar_Gauge2_Skew (fac ×
c)
| Dim5_Scalar_Vector_Vector_T c → Dim5_Scalar_Vector_Vector_T (fac ×
c)
| Dim6_Vector_Vector_Vector_T c → Dim6_Vector_Vector_Vector_T (fac ×
c)
| Tensor_2_Vector_Vector c → Tensor_2_Vector_Vector (fac × c)
| Dim5_Tensor_2_Vector_Vector_1 c → Dim5_Tensor_2_Vector_Vector_1 (fac ×
c)
| Dim5_Tensor_2_Vector_Vector_2 c → Dim5_Tensor_2_Vector_Vector_2 (fac ×
c)
| Dim7_Tensor_2_Vector_Vector_T c → Dim7_Tensor_2_Vector_Vector_T (fac ×
c)

let mult_vertex4 fac = function
| Scalar4 c → Scalar4 (fac × c)
| Scalar2_Vector2 c → Scalar2_Vector2 (fac × c)
| Vector4 ic4_list → Vector4 (List.map (fun (c, icl) → ((fac ×
c), icl)) ic4_list)
| DScalar4 ic4_list → DScalar4 (List.map (fun (c, icl) → ((fac ×
c), icl)) ic4_list)
| DScalar2_Vector2 ic4_list → DScalar2_Vector2 (List.map (fun (c, icl) →
((fac × c), icl)) ic4_list)
| GBBG (c, fb, b2, f) → GBBG ((fac × c), fb, b2, f)
| Vector4_K_Matrix_tho (c, ch2_list) → Vector4_K_Matrix_tho ((fac ×
c), ch2_list)
| Vector4_K_Matrix_jr (c, ch2_list) → Vector4_K_Matrix_jr ((fac ×
c), ch2_list)

```

Cubic Vertices

```
let vertices3, vertices4, verticesn = M.vertices ()
```

Important: In the following, we don't test that the $SU(N)$ groups match and that $N > 0$, since we can assume that *colorize_flavor* would have thrown an exception.

```
let colorize_vertex3 ((f1, f2, f3), v, g) =
  match M.color f1, M.color f2, M.color f3 with
```

The trivial case.

```

| C.Singlet, C.Singlet, C.Singlet →
  [(White f1, White f2, White f3), v, g]

```

Coupling a quark, an anti-quark and a colorless particle: all particles are *guaranteed* to be different and no nontrivial symmetry can arise.

```

| C.SUN nc1, C.SUN nc2, C.Singlet →
  if nc1 ≠ - nc2 then
    colored_vertex "colored_vertex3"
  else if nc1 > 0 then

```

```

      List.map
        (fun c → ((CF_in (f1, c), CF_out (f2, c), White f3), v, g))
        color_flows
    else
      List.map
        (fun c → ((CF_out (f1, c), CF_in (f2, c), White f3), v, g))
        color_flows
  | C.SUN nc1, C.Singlet, C.SUN nc3 →
    if nc1 ≠ - nc3 then
      colored_vertex "colored_vertex3"
    else if nc1 > 0 then
      List.map
        (fun c → ((CF_in (f1, c), White f2, CF_out (f3, c)), v, g))
        color_flows
    else
      List.map
        (fun c → ((CF_out (f1, c), White f2, CF_in (f3, c)), v, g))
        color_flows
  | C.Singlet, C.SUN nc2, C.SUN nc3 →
    if nc2 ≠ - nc3 then
      colored_vertex "colored_vertex3"
    else if nc2 > 0 then
      List.map
        (fun c → ((White f1, CF_in (f2, c), CF_out (f3, c)), v, g))
        color_flows
    else
      List.map
        (fun c → ((White f1, CF_out (f2, c), CF_in (f3, c)), v, g))
        color_flows

```

Coupling a quark, an anti-quark and a gluon: all particles are again *guaranteed* to be different and no nontrivial symmetry can arise.

```

  | C.SUN nc1, C.SUN nc2, C.AdjSUN _ →
    if nc1 ≠ - nc2 then
      colored_vertex "colored_vertex3"
    else if nc1 > 0 then
      List.map
        (fun (c1, c2) →
          ((CF_in (f1, c1), CF_out (f2, c2), CF_io (f3, c2, c1)), v, g))
        color_flow_pairs
      @ (List.map
        (fun c → ((CF_in (f1, c), CF_out (f2, c), CF_aux f3), v, g))
        color_flows)
    else
      List.map
        (fun (c1, c2) →
          ((CF_out (f1, c2), CF_in (f2, c1), CF_io (f3, c2, c1)), v, g))
        color_flow_pairs
      @ (List.map

```

```

      (fun c → ((CF_out (f1, c), CF_in (f2, c), CF_aux f3), v, g))
      color_flows)
| C.SUN nc1, C.AdjSUN -, C.SUN nc3 →
  if nc1 ≠ - nc3 then
    colored_vertex "colored_vertex3"
  else if nc1 > 0 then
    List.map
      (fun (c1, c3) →
        ((CF_in (f1, c1), CF_io (f2, c3, c1), CF_out (f3, c3)), v, g))
        color_flow_pairs
      @ (List.map
        (fun c → ((CF_in (f1, c), CF_aux f2, CF_out (f3, c)), v, g))
          color_flows)
    else
      List.map
        (fun (c1, c3) →
          ((CF_out (f1, c1), CF_io (f2, c1, c3), CF_in (f3, c3)), v, g))
          color_flow_pairs
        @ (List.map
          (fun c → ((CF_out (f1, c), CF_aux f2, CF_in (f3, c)), v, g))
            color_flows)
| C.AdjSUN -, C.SUN nc2, C.SUN nc3 →
  if nc2 ≠ - nc3 then
    colored_vertex "colored_vertex3"
  else if nc2 > 0 then
    List.map
      (fun (c2, c3) →
        ((CF_io (f1, c3, c2), CF_in (f2, c2), CF_out (f3, c3)), v, g))
        color_flow_pairs
      @ (List.map
        (fun c → ((CF_aux f1, CF_in (f2, c), CF_out (f3, c)), v, g))
          color_flows)
    else
      List.map
        (fun (c2, c3) →
          ((CF_io (f1, c2, c3), CF_out (f2, c2), CF_in (f3, c3)), v, g))
          color_flow_pairs
        @ (List.map
          (fun c → ((CF_aux f1, CF_out (f2, c), CF_in (f3, c)), v, g))
            color_flows)

```

Coupling two gluons with a colorless particle:

To make the color algebra correct, we need to introduce the vertex with the two ghost gluons with a relative factor of -3.

```

| C.AdjSUN -, C.AdjSUN -, C.Singlet →
  List.map (fun (c1, c2) → ((CF_io (f1, c1, c2), CF_io (f2, c2, c1), White f3), v, g))
    color_flow_pairs
  @
  [((CF_aux f1, CF_aux f2, White f3), (mult_vertex3 (-3) v), g)]

```

```

| C.AdjSUN -, C.Singlet, C.AdjSUN - →
  List.map (fun (c1, c3) → ((CF_io (f1, c1, c3), White f2, CF_io (f3, c3, c1)), v, g))
    color_flow_pairs
  @
  [((CF_aux f1, White f2, CF_aux f3), (mult_vertex3 (-3) v), g)]
| C.Singlet, C.AdjSUN -, C.AdjSUN - →
  List.map (fun (c2, c3) → ((White f1, CF_io (f2, c2, c3), CF_io (f3, c3, c2)), v, g))
    color_flow_pairs
  @
  [((White f1, CF_aux f2, CF_aux f3), (mult_vertex3 (-3) v), g)]

```

Coupling three gluons:

```

| C.AdjSUN -, C.AdjSUN -, C.AdjSUN - →
  if f1 = f2 ∧ f2 = f3 then
    ThoList.flatmap
      (fun (c1, c2, c3) →
        [((CF_io (f1, c1, c3), CF_io (f2, c2, c1), CF_io (f3, c3, c2)), v, g);
          ((CF_io (f1, c1, c2), CF_io (f2, c3, c1), CF_io (f3, c2, c3)), v, g)])
        color_flow_triples
  else
    ThoList.flatmap
      (fun (c1, c2, c3) →
        (List.map (fun (c1', c2', c3') →
          ((CF_io (f1, c1', c3'), CF_io (f2, c2', c1'), CF_io (f3, c3', c2')), v, g))
            (permute_triple (c1, c2, c3))))
        color_flow_triples

```

The rest is *verboden*!

JR mildly protests, because in principle a diquark coupling like in the baryon-number violating superpotential of three (s)quarks might be allowed. Might be an interesting task to work out the color flow combinations.

Tho concedes that he forgot the special case of a SU(3)-baryon-like coupling
...

```

| C.SUN -, (C.Singlet | C.AdjSUN -), (C.Singlet | C.AdjSUN -)
| (C.Singlet | C.AdjSUN -), C.SUN -, (C.Singlet | C.AdjSUN -)
| (C.Singlet | C.AdjSUN -), (C.Singlet | C.AdjSUN -), C.SUN - →
  colored_vertex "colored_vertex3:_single_quark/anti-quark"
| C.SUN -, C.SUN -, C.SUN - →
  colored_vertex "colored_vertex3:_three_quarks/anti-quarks"
| C.Singlet, C.Singlet, C.AdjSUN -
| C.Singlet, C.AdjSUN -, C.Singlet
| C.AdjSUN -, C.Singlet, C.Singlet →
  colored_vertex "colored_vertex3:_single_gluon"

```

Quartic Vertices

```
let colorize_vertex4 ((f1, f2, f3, f4), v, g) =
```

match *M.color f1*, *M.color f2*, *M.color f3*, *M.color f4* with

The trivial case.

```
| C.Singlet, C.Singlet, C.Singlet, C.Singlet →
  [(White f1, White f2, White f3, White f4), v, g]
```

Coupling a quark, an anti-quark and two colorless particles:

```
| C.SUN nc1, C.SUN nc2, C.Singlet, C.Singlet →
  if nc1 ≠ - nc2 then
    colored_vertex "colorize_vertex4"
  else if nc1 > 0 then
    List.map
      (fun c → ((CF_in (f1, c), CF_out (f2, c), White f3, White f4), v, g))
      color_flows
  else
    List.map
      (fun c → ((CF_out (f1, c), CF_in (f2, c), White f3, White f4), v, g))
      color_flows
| C.SUN nc1, C.Singlet, C.SUN nc3, C.Singlet →
  if nc1 ≠ - nc3 then
    colored_vertex "colorize_vertex4"
  else if nc1 > 0 then
    List.map
      (fun c → ((CF_in (f1, c), White f2, CF_out (f3, c), White f4), v, g))
      color_flows
  else
    List.map
      (fun c → ((CF_out (f1, c), White f2, CF_in (f3, c), White f4), v, g))
      color_flows
| C.SUN nc1, C.Singlet, C.Singlet, C.SUN nc4 →
  if nc1 ≠ - nc4 then
    colored_vertex "colorize_vertex4"
  else if nc1 > 0 then
    List.map
      (fun c → ((CF_in (f1, c), White f2, White f3, CF_out (f4, c)), v, g))
      color_flows
  else
    List.map
      (fun c → ((CF_out (f1, c), White f2, White f3, CF_in (f4, c)), v, g))
      color_flows
| C.Singlet, C.SUN nc2, C.SUN nc3, C.Singlet →
  if nc2 ≠ - nc3 then
    colored_vertex "colorize_vertex4"
  else if nc2 > 0 then
    List.map
      (fun c → ((White f1, CF_in (f2, c), CF_out (f3, c), White f4), v, g))
      color_flows
  else
    List.map
```

```

      (fun c → (( White f1, CF_out (f2, c), CF_in (f4, c), White f4), v, g))
      color_flows
| C.Singlet, C.SUN nc2, C.Singlet, C.SUN nc4 →
  if nc2 ≠ - nc4 then
    colored_vertex "colorize_vertex4"
  else if nc2 > 0 then
    List.map
      (fun c → (( White f1, CF_in (f2, c), White f3, CF_out (f4, c)), v, g))
      color_flows
  else
    List.map
      (fun c → (( White f1, CF_out (f2, c), White f3, CF_in (f4, c)), v, g))
      color_flows
| C.Singlet, C.Singlet, C.SUN nc3, C.SUN nc4 →
  if nc3 ≠ - nc4 then
    colored_vertex "colorize_vertex4"
  else if nc3 > 0 then
    List.map
      (fun c → (( White f1, White f2, CF_in (f3, c), CF_out (f4, c)), v, g))
      color_flows
  else
    List.map
      (fun c → (( White f1, White f2, CF_out (f3, c), CF_in (f4, c)), v, g))
      color_flows

```

Coupling two quarks and two anti-quarks requires additional colorflow specification: better use an auxiliary field here!:

```

| C.SUN -, C.SUN -, C.SUN -, C.SUN - →
  color_flow_ambiguous "colorize_vertex4:four quarks/anti-quarks"

```

Coupling a quark, an anti-quark, a gluon and a colorless particle: all particles are again *guaranteed* to be different and no nontrivial symmetry can arise.

```

| C.SUN nc1, C.SUN nc2, C.AdjSUN -, C.Singlet →
  if nc1 ≠ - nc2 then
    colored_vertex "colorize_vertex4"
  else if nc1 > 0 then
    List.map
      (fun (c1, c2) → ((CF_in (f1, c1), CF_out (f2, c2), CF_io (f3, c2, c1), White f4), v, g))
      color_flow_pairs
    @ (List.map
      (fun c → ((CF_in (f1, c), CF_out (f2, c), CF_aux f3, White f4), v, g))
      color_flows)
  else
    List.map
      (fun (c1, c2) → ((CF_out (f1, c2), CF_in (f2, c1), CF_io (f3, c2, c1), White f4), v, g))
      color_flow_pairs
    @ (List.map
      (fun c → ((CF_out (f1, c), CF_in (f2, c), CF_aux f3, White f4), v, g))
      color_flows)

```



```

| C.SUN nc1, C.SUN nc2, C.Singlet, C.AdjSUN _ →
  if nc1 ≠ - nc2 then
    colored_vertex "colorize_vertex4"
  else if nc1 > 0 then
    List.map
      (fun (c1, c2) → ((CF_in (f1, c1), CF_out (f2, c2), White f3, CF_io (f4, c2, c1)), v, g))
    color_flow_pairs
    @ (List.map
      (fun c → ((CF_in (f1, c), CF_out (f2, c), White f3, CF_aux f4), v, g))
      color_flows)
  else
    List.map
      (fun (c1, c2) → ((CF_out (f1, c2), CF_in (f2, c1), White f3, CF_io (f4, c2, c1)), v, g))
    color_flow_pairs
    @ (List.map
      (fun c → ((CF_out (f1, c), CF_in (f2, c), White f3, CF_aux f4), v, g))
      color_flows)

| C.SUN nc1, C.AdjSUN _, C.SUN nc3, C.Singlet →
  if nc1 ≠ - nc3 then
    colored_vertex "colorize_vertex4"
  else if nc1 > 0 then
    List.map
      (fun (c1, c3) → ((CF_in (f1, c1), CF_io (f2, c3, c1), CF_out (f3, c3), White f4), v, g))
    color_flow_pairs
    @ (List.map
      (fun c → ((CF_in (f1, c), CF_aux f2, CF_out (f3, c), White f4), v, g))
      color_flows)
  else
    List.map
      (fun (c1, c3) → ((CF_out (f1, c3), CF_io (f2, c3, c1), CF_in (f3, c1), White f4), v, g))
    color_flow_pairs
    @ (List.map
      (fun c → ((CF_out (f1, c), CF_aux f2, CF_in (f3, c), White f4), v, g))
      color_flows)

| C.SUN nc1, C.Singlet, C.SUN nc3, C.AdjSUN _ →
  if nc1 ≠ - nc3 then
    colored_vertex "colorize_vertex4"
  else if nc1 > 0 then
    List.map
      (fun (c1, c3) → ((CF_in (f1, c1), White f2, CF_out (f3, c3), CF_io (f4, c3, c1)), v, g))
    color_flow_pairs
    @ (List.map
      (fun c → ((CF_in (f1, c), White f2, CF_out (f3, c), CF_aux f4), v, g))
      color_flows)
  else
    List.map
      (fun (c1, c3) → ((CF_out (f1, c3), White f2, CF_in (f3, c1), CF_io (f4, c3, c1)), v, g))
    color_flow_pairs
    @ (List.map

```

```

      (fun c → ((CF_out (f1, c), White f2, CF_in (f3, c), CF_aux f4), v, g))
      color_flows)
| C.SUN nc1, C.AdjSUN –, C.Singlet, C.SUN nc4 →
  if nc1 ≠ – nc4 then
    colored_vertex "colorize_vertex4"
  else if nc1 > 0 then
    List.map
      (fun (c1, c4) → ((CF_in (f1, c1), CF_io (f2, c4, c1), White f3, CF_out (f4, c4)), v, g))
      color_flow_pairs
    @ (List.map
      (fun c → ((CF_in (f1, c), CF_aux f2, White f3, CF_out (f4, c)), v, g))
      color_flows)
  else
    List.map
      (fun (c1, c4) → ((CF_out (f1, c4), CF_io (f2, c4, c1), White f3, CF_in (f4, c1)), v, g))
      color_flow_pairs
    @ (List.map
      (fun c → ((CF_out (f1, c), CF_aux f2, White f3, CF_in (f4, c)), v, g))
      color_flows)
| C.SUN nc1, C.Singlet, C.AdjSUN –, C.SUN nc4 →
  if nc1 ≠ – nc4 then
    colored_vertex "colorize_vertex4"
  else if nc1 > 0 then
    List.map
      (fun (c1, c4) → ((CF_in (f1, c1), White f2, CF_io (f3, c4, c1), CF_out (f4, c4)), v, g))
      color_flow_pairs
    @ (List.map
      (fun c → ((CF_in (f1, c), White f2, CF_aux f3, CF_out (f4, c)), v, g))
      color_flows)
  else
    List.map
      (fun (c1, c4) → ((CF_out (f1, c4), White f2, CF_io (f3, c4, c1), CF_in (f4, c1)), v, g))
      color_flow_pairs
    @ (List.map
      (fun c → ((CF_out (f1, c), White f2, CF_aux f3, CF_in (f4, c)), v, g))
      color_flows)
| C.AdjSUN nc1, C.SUN nc2, C.SUN nc3, C.Singlet →
  if nc2 ≠ – nc3 then
    colored_vertex "colorize_vertex4"
  else if nc2 > 0 then
    List.map
      (fun (c2, c3) → ((CF_io (f1, c3, c2), CF_in (f2, c2), CF_out (f3, c3), White f4), v, g))
      color_flow_pairs
    @ (List.map
      (fun c → ((CF_aux f1, CF_in (f2, c), CF_out (f3, c), White f4), v, g))
      color_flows)
  else
    List.map
      (fun (c2, c3) → ((CF_io (f1, c3, c2), CF_out (f2, c3), CF_in (f3, c2), White f4), v, g))

```

```

      color_flow_pairs
    @ (List.map
      (fun c → ((CF_aux f1, CF_out (f2, c), CF_in (f3, c), White f4), v, g))
      color_flows)
  | C.Singlet, C.SUN nc2, C.SUN nc3, C.AdjSUN _ →
    if nc2 ≠ - nc3 then
      colored_vertex "colorize-vertex4"
    else if nc2 > 0 then
      List.map
        (fun (c2, c3) → ((White f1, CF_in (f2, c2), CF_out (f3, c3), CF_io (f4, c3, c2)), v, g))
        color_flow_pairs
      @ (List.map
        (fun c → ((White f1, CF_in (f2, c), CF_out (f3, c), CF_aux f4), v, g))
        color_flows)
    else
      List.map
        (fun (c2, c3) → ((White f1, CF_out (f2, c3), CF_in (f3, c2), CF_io (f4, c3, c2)), v, g))
        color_flow_pairs
      @ (List.map
        (fun c → ((White f1, CF_out (f2, c), CF_in (f3, c), CF_aux f4), v, g))
        color_flows)
  | C.AdjSUN _, C.SUN nc2, C.Singlet, C.SUN nc4 →
    if nc2 ≠ - nc4 then
      colored_vertex "colorize-vertex4"
    else if nc2 > 0 then
      List.map
        (fun (c2, c4) → ((CF_io (f1, c4, c2), CF_in (f2, c2), White f3, CF_out (f4, c4)), v, g))
        color_flow_pairs
      @ (List.map
        (fun c → ((CF_aux f1, CF_in (f2, c), White f3, CF_out (f4, c)), v, g))
        color_flows)
    else
      List.map
        (fun (c2, c4) → ((CF_io (f1, c4, c2), CF_out (f2, c4), White f3, CF_in (f4, c2)), v, g))
        color_flow_pairs
      @ (List.map
        (fun c → ((CF_aux f1, CF_out (f2, c), White f3, CF_in (f4, c)), v, g))
        color_flows)
  | C.Singlet, C.SUN nc2, C.AdjSUN _, C.SUN nc4 →
    if nc2 ≠ - nc4 then
      colored_vertex "colorize-vertex4"
    else if nc2 > 0 then
      List.map
        (fun (c2, c4) → ((White f1, CF_in (f2, c2), CF_io (f3, c4, c2), CF_out (f4, c4)), v, g))
        color_flow_pairs
      @ (List.map
        (fun c → ((White f1, CF_in (f2, c), CF_aux f3, CF_out (f4, c)), v, g))
        color_flows)
    else

```

```

      List.map
      (fun (c2, c4) → ((White f1, CF_out (f2, c4), CF_io (f3, c4, c2), CF_in (f4, c2)), v, g))
      color_flow_pairs
    @ (List.map
      (fun c → ((White f1, CF_out (f2, c), CF_aux f3, CF_in (f4, c)), v, g))
      color_flows)
  | C.AdjSUN -, C.Singlet, C.SUN nc3, C.SUN nc4 →
    if nc3 ≠ - nc4 then
      colored_vertex "colorize_vertex4"
    else if nc3 > 0 then
      List.map
      (fun (c3, c4) → ((CF_io (f1, c4, c3), White f2, CF_in (f3, c3), CF_out (f4, c4)), v, g))
      color_flow_pairs
    @ (List.map
      (fun c → ((CF_aux f1, White f2, CF_in (f3, c), CF_out (f4, c)), v, g))
      color_flows)
    else
      List.map
      (fun (c3, c4) → ((CF_io (f1, c4, c3), White f2, CF_out (f2, c4), CF_in (f4, c3)), v, g))
      color_flow_pairs
    @ (List.map
      (fun c → ((CF_aux f1, White f2, CF_out (f2, c), CF_in (f4, c)), v, g))
      color_flows)
  | C.Singlet, C.AdjSUN -, C.SUN nc3, C.SUN nc4 →
    if nc3 ≠ - nc4 then
      colored_vertex "colorize_vertex4"
    else if nc3 > 0 then
      List.map
      (fun (c3, c4) → ((White f1, CF_io (f2, c4, c3), CF_in (f3, c3), CF_out (f4, c4)), v, g))
      color_flow_pairs
    @ (List.map
      (fun c → ((White f1, CF_aux f2, CF_in (f3, c), CF_out (f4, c)), v, g))
      color_flows)
    else
      List.map
      (fun (c3, c4) → ((White f1, CF_io (f2, c4, c3), CF_out (f2, c4), CF_in (f4, c3)), v, g))
      color_flow_pairs
    @ (List.map
      (fun c → ((White f1, CF_aux f2, CF_out (f2, c), CF_in (f4, c)), v, g))
      color_flows)

```

Coupling a quark, an anti-quark and two gluons. For two different octets (is there a realistic situation for this we need twelve combinations as well as two combinations for the rest.

```

  | C.SUN nc1, C.SUN nc2, C.AdjSUN -, C.AdjSUN - →
    if (compare f3 f4) ≠ 0 then
      incomplete "colorize_vertex4"
    else
      if nc1 ≠ - nc2 then

```

```

    colored_vertex "colorize_vertex4"
  else if nc1 > 0 then
    ThoList.flatmap
      (fun (c1, c2, c3) → List.map (fun (c1', c2', c3') →
        ((CF_in (f1, c1'), CF_out (f2, c2'), CF_io (f3, c2', c3'), CF_io (f4, c3', c1')), v, g)
        (permute_triple (c1, c2, c3))) color_flow_triples
      @ List.map (fun (c1, c2) →
        ((CF_in (f1, c1), CF_out (f2, c2), CF_io (f3, c2, c1), CF_aux f4),
        (mult_vertex4 2 v), g)) color_flow_pairs
      @ (List.map (fun c →
        ((CF_in (f1, c), CF_out (f2, c), CF_aux f3, CF_aux f4), (mult_vertex4 2 v), g))
        color_flows)
    else
      ThoList.flatmap
        (fun (c1, c2, c3) → List.map (fun (c1', c2', c3') →
          ((CF_out (f1, c2'), CF_in (f2, c1'), CF_io (f3, c2', c3'), CF_io (f4, c3', c1')), v, g)
          (permute_triple (c1, c2, c3))) color_flow_triples
        @ (List.map (fun (c1, c2) →
          ((CF_out (f1, c2), CF_in (f2, c1), CF_io (f3, c2, c1), CF_aux f4),
          (mult_vertex4 2 v), g)) color_flow_pairs)
        @ (List.map (fun c →
          ((CF_out (f1, c), CF_in (f2, c), CF_aux f3, CF_aux f4), (mult_vertex4 2 v), g))
          color_flows)
      | C.SUN nc1, C.AdjSUN -, C.SUN nc3, C.AdjSUN - →
        if (compare f2 f4) ≠ 0 then
          incomplete "colorize_vertex4"
        else
          if nc1 ≠ - nc3 then
            colored_vertex "colorize_vertex4"
          else if nc1 > 0 then
            ThoList.flatmap
              (fun (c1, c2, c3) → List.map (fun (c1', c2', c3') →
                ((CF_in (f1, c1'), CF_io (f2, c2', c3'), CF_out (f3, c2'), CF_io (f4, c3', c1')), v, g)
                (permute_triple (c1, c2, c3))) color_flow_triples
              @ List.map (fun (c1, c2) →
                ((CF_in (f1, c1), CF_io (f2, c2, c1), CF_out (f3, c2), CF_aux f4),
                (mult_vertex4 2 v), g)) color_flow_pairs
              @ (List.map (fun c →
                ((CF_in (f1, c), CF_aux f2, CF_out (f3, c), CF_aux f4), (mult_vertex4 2 v), g))
                color_flows)
            else
              ThoList.flatmap
                (fun (c1, c2, c3) → List.map (fun (c1', c2', c3') →
                  ((CF_out (f1, c2'), CF_io (f2, c2', c3'), CF_in (f3, c1'), CF_io (f4, c3', c1')), v, g)
                  (permute_triple (c1, c2, c3))) color_flow_triples
                @ (List.map (fun (c1, c2) →
                  ((CF_out (f1, c2), CF_io (f2, c2, c1), CF_in (f3, c1), CF_aux f4),
                  (mult_vertex4 2 v), g)) color_flow_pairs)
                @ (List.map (fun c →

```

```

((CF_out (f1, c), CF_aux f2, CF_in (f3, c), CF_aux f4), (mult_vertex4 2 v), g))
  color_flows)
| C.SUN nc1, C.AdjSUN -, C.AdjSUN -, C.SUN nc4 →
  if (compare f2 f3) ≠ 0 then
    incomplete "colorize_vertex4"
  else
    if nc1 ≠ - nc4 then
      colored_vertex "colorize_vertex4"
    else if nc1 > 0 then
      ThoList.flatmap
        (fun (c1, c2, c3) → List.map (fun (c1', c2', c3') →
          ((CF_in (f1, c1'), CF_io (f2, c2', c3'), CF_io (f3, c3', c1'), CF_out (f4, c2')), v, g)
          (permute_triple (c1, c2, c3))) color_flow_triples
        @ List.map (fun (c1, c2) →
          ((CF_in (f1, c1), CF_io (f2, c2, c1), CF_aux f3, CF_out (f4, c2)),
            (mult_vertex4 2 v), g)) color_flow_pairs
        @ (List.map (fun c →
          ((CF_in (f1, c), CF_aux f2, CF_aux f3, CF_out (f4, c)), (mult_vertex4 2 v), g))
          color_flows)
      else
        ThoList.flatmap
          (fun (c1, c2, c3) → List.map (fun (c1', c2', c3') →
            ((CF_out (f1, c2'), CF_io (f2, c2', c3'), CF_io (f3, c3', c1'), CF_in (f4, c1')), v, g)
            (permute_triple (c1, c2, c3))) color_flow_triples
          @ (List.map (fun (c1, c2) →
            ((CF_out (f1, c2), CF_io (f2, c2, c1), CF_aux f3, CF_in (f4, c1)),
              (mult_vertex4 2 v), g)) color_flow_pairs)
          @ (List.map (fun c →
            ((CF_out (f1, c), CF_aux f2, CF_aux f3, CF_in (f4, c)), (mult_vertex4 2 v), g))
            color_flows)
    | C.AdjSUN -, C.SUN nc2, C.SUN nc3, C.AdjSUN - →
      if (compare f1 f4) ≠ 0 then
        incomplete "colorize_vertex4"
      else
        if nc2 ≠ - nc3 then
          colored_vertex "colorize_vertex4"
        else if nc2 > 0 then
          ThoList.flatmap
            (fun (c1, c2, c3) → List.map (fun (c1', c2', c3') →
              ((CF_io (f1, c2', c3'), CF_in (f2, c1'), CF_out (f3, c2'), CF_io (f4, c3', c1')), v, g)
              (permute_triple (c1, c2, c3))) color_flow_triples
            @ List.map (fun (c1, c2) →
              ((CF_io (f1, c2, c1), CF_in (f2, c1), CF_out (f3, c2), CF_aux f4),
                (mult_vertex4 2 v), g)) color_flow_pairs
            @ (List.map (fun c →
              ((CF_aux f1, CF_in (f2, c), CF_out (f3, c), CF_aux f4), (mult_vertex4 2 v), g))
              color_flows)
          else
            ThoList.flatmap

```

```

    (fun (c1, c2, c3) → List.map (fun (c1', c2', c3') →
      ((CF_io (f1, c2', c3'), CF_out (f2, c2'), CF_in (f3, c1'), CF_io (f4, c3', c1')), v, g)
      (permute_triple (c1, c2, c3))) color_flow_triples
    @ (List.map (fun (c1, c2) →
      ((CF_io (f1, c2, c1), CF_out (f2, c2), CF_in (f3, c1), CF_aux f4),
        (mult_vertex4 2 v), g)) color_flow_pairs)
    @ (List.map (fun c →
      ((CF_aux f1, CF_out (f2, c), CF_in (f3, c), CF_aux f4), (mult_vertex4 2 v), g))
        color_flows)

| C.AdjSUN -, C.SUN nc2, C.AdjSUN -, C.SUN nc4 →
  if (compare f1 f3) ≠ 0 then
    incomplete "colorize-vertex4"
  else
    if nc2 ≠ - nc4 then
      colored_vertex "colorize-vertex4"
    else if nc2 > 0 then
      ThoList.flatmap
        (fun (c1, c2, c3) → List.map (fun (c1', c2', c3') →
          ((CF_io (f1, c2', c3'), CF_in (f2, c1'), CF_io (f3, c3', c1'), CF_out (f4, c2')), v, g)
          (permute_triple (c1, c2, c3))) color_flow_triples
        @ List.map (fun (c1, c2) →
          ((CF_io (f1, c2, c1), CF_in (f2, c1), CF_aux f3, CF_out (f4, c2)),
            (mult_vertex4 2 v), g)) color_flow_pairs
        @ (List.map (fun c →
          ((CF_aux f1, CF_in (f2, c), CF_aux f3, CF_out (f4, c)), (mult_vertex4 2 v), g))
            color_flows)
    else
      ThoList.flatmap
        (fun (c1, c2, c3) → List.map (fun (c1', c2', c3') →
          ((CF_io (f1, c2', c3'), CF_out (f2, c2'), CF_io (f3, c3', c1'), CF_in (f4, c1')), v, g)
          (permute_triple (c1, c2, c3))) color_flow_triples
        @ (List.map (fun (c1, c2) →
          ((CF_io (f1, c2, c1), CF_out (f2, c2), CF_aux f3, CF_in (f4, c1)),
            (mult_vertex4 2 v), g)) color_flow_pairs)
        @ (List.map (fun c →
          ((CF_aux f1, CF_out (f2, c), CF_aux f3, CF_in (f4, c)), (mult_vertex4 2 v), g))
            color_flows)

| C.AdjSUN -, C.AdjSUN -, C.SUN nc3, C.SUN nc4 →
  if (compare f1 f2) ≠ 0 then
    incomplete "colorize-vertex4"
  else
    if nc3 ≠ - nc4 then
      colored_vertex "colorize-vertex4"
    else if nc3 > 0 then
      ThoList.flatmap
        (fun (c1, c2, c3) → List.map (fun (c1', c2', c3') →
          ((CF_io (f1, c2', c3'), CF_io (f2, c3', c1'), CF_in (f3, c1'), CF_out (f4, c2')), v, g)
          (permute_triple (c1, c2, c3))) color_flow_triples
        @ List.map (fun (c1, c2) →
          ((CF_io (f1, c2, c1), CF_in (f2, c2), CF_in (f3, c1), CF_out (f4, c2)),
            (mult_vertex4 2 v), g)) color_flow_pairs)

```

```

      ((CF_io (f1, c2, c1), CF_aux f2, CF_in (f3, c1), CF_out (f4, c2)),
       (mult_vertex4 2 v), g)) color_flow_pairs
    @ (List.map (fun c →
      ((CF_aux f1, CF_aux f2, CF_in (f3, c), CF_out (f4, c)), (mult_vertex4 2 v), g))
      color_flows)
  else
    ThoList.flatmap
      (fun (c1, c2, c3) → List.map (fun (c1', c2', c3') →
        ((CF_io (f1, c2', c3'), CF_io (f2, c3', c1'), CF_out (f3, c2'), CF_in (f4, c1')), v, g)
        (permute_triple (c1, c2, c3))) color_flow_triples
      @ (List.map (fun (c1, c2) →
        ((CF_io (f1, c2, c1), CF_aux f2, CF_out (f3, c2), CF_in (f4, c1)),
         (mult_vertex4 2 v), g)) color_flow_pairs)
      @ (List.map (fun c →
        ((CF_aux f1, CF_aux f2, CF_out (f3, c), CF_in (f4, c)), (mult_vertex4 2 v), g))
        color_flows)

```

Coupling two gluons and two colorless particles.

```

| C.AdjSUN -, C.AdjSUN -, C.Singlet, C.Singlet →
  List.map (fun (c1, c2) → ((CF_io (f1, c1, c2), CF_io (f2, c2, c1), White f3, White f4), v, g)
    color_flow_pairs
  @
  [((CF_aux f1, CF_aux f2, White f3, White f4), (mult_vertex4 (-3) v), g)]
| C.AdjSUN -, C.Singlet, C.AdjSUN -, C.Singlet →
  List.map (fun (c1, c3) → ((CF_io (f1, c1, c3), White f2, CF_io (f3, c3, c1), White f4), v, g)
    color_flow_pairs
  @
  [((CF_aux f1, White f2, CF_aux f3, White f4), (mult_vertex4 (-3) v), g)]
| C.AdjSUN -, C.Singlet, C.Singlet, C.AdjSUN - →
  List.map (fun (c1, c4) → ((CF_io (f1, c1, c4), White f2, White f3, CF_io (f4, c4, c1)), v, g)
    color_flow_pairs
  @
  [((CF_aux f1, White f2, White f3, CF_aux f4), (mult_vertex4 (-3) v), g)]
| C.Singlet, C.AdjSUN -, C.AdjSUN -, C.Singlet →
  List.map (fun (c2, c3) → ((White f1, CF_io (f2, c2, c3), CF_io (f3, c3, c2), White f4), v, g)
    color_flow_pairs
  @
  [((White f1, CF_aux f2, CF_aux f3, White f4), (mult_vertex4 (-3) v), g)]
| C.Singlet, C.AdjSUN -, C.Singlet, C.AdjSUN - →
  List.map (fun (c2, c4) → ((White f1, CF_io (f2, c2, c4), White f3, CF_io (f4, c4, c2)), v, g)
    color_flow_pairs
  @
  [((White f1, CF_aux f2, White f3, CF_aux f4), (mult_vertex4 (-3) v), g)]
| C.Singlet, C.Singlet, C.AdjSUN -, C.AdjSUN - →
  List.map (fun (c3, c4) → ((White f1, White f2, CF_io (f3, c3, c4), CF_io (f4, c4, c3)), v, g)
    color_flow_pairs
  @
  [((White f1, White f2, CF_aux f3, CF_aux f4), (mult_vertex4 (-3) v), g)]

```


Coupling tree gluons and a colorless particle.

```

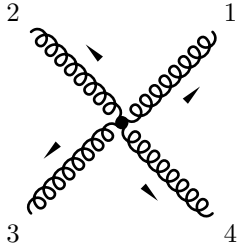
| C.AdjSUN -, C.AdjSUN -, C.AdjSUN -, C.Singlet →
  ThoList.flatmap
    (fun (c1, c2, c3) →
      [((CF_io (f1, c1, c3), CF_io (f2, c2, c1), CF_io (f3, c3, c2), White f4), v, g);
        ((CF_io (f1, c1, c2), CF_io (f2, c3, c1), CF_io (f3, c2, c3), White f4), v, g)])
      color_flow_triples

| C.AdjSUN -, C.AdjSUN -, C.Singlet, C.AdjSUN - →
  ThoList.flatmap
    (fun (c1, c2, c4) →
      [((CF_io (f1, c1, c4), CF_io (f2, c2, c1), White f3, CF_io (f4, c4, c2)), v, g);
        ((CF_io (f1, c1, c2), CF_io (f2, c4, c1), White f3, CF_io (f4, c2, c4)), v, g)])
      color_flow_triples

| C.AdjSUN -, C.Singlet, C.AdjSUN -, C.AdjSUN - →
  ThoList.flatmap
    (fun (c1, c3, c4) →
      [((CF_io (f1, c1, c4), White f2, CF_io (f3, c3, c1), CF_io (f4, c4, c3)), v, g);
        ((CF_io (f1, c1, c3), White f2, CF_io (f3, c4, c1), CF_io (f4, c3, c4)), v, g)])
      color_flow_triples

| C.Singlet, C.AdjSUN -, C.AdjSUN -, C.AdjSUN - →
  ThoList.flatmap
    (fun (c2, c3, c4) →
      [((White f1, CF_io (f2, c2, c4), CF_io (f3, c3, c2), CF_io (f4, c4, c3)), v, g);
        ((White f1, CF_io (f2, c2, c3), CF_io (f3, c4, c2), CF_io (f4, c3, c4)), v, g)])
      color_flow_triples
    
```

Coupling four gluons. Tho still has concerns about symmetry factors for KK gluons. It's the same problem that already appears for the gluino-gluon-gluino vertex.



$$\begin{aligned}
 &= -ig^2 f_{a_1 a_2 b} f_{a_3 a_4 b} (g_{\mu_1 \mu_3} g_{\mu_4 \mu_2} - g_{\mu_1 \mu_4} g_{\mu_2 \mu_3}) \\
 &\quad -ig^2 f_{a_1 a_3 b} f_{a_4 a_2 b} (g_{\mu_1 \mu_4} g_{\mu_2 \mu_3} - g_{\mu_1 \mu_2} g_{\mu_3 \mu_4}) \\
 &\quad -ig^2 f_{a_1 a_4 b} f_{a_2 a_3 b} (g_{\mu_1 \mu_2} g_{\mu_3 \mu_4} - g_{\mu_1 \mu_3} g_{\mu_4 \mu_2})
 \end{aligned} \tag{10.1}$$

```

| C.AdjSUN -, C.AdjSUN -, C.AdjSUN -, C.AdjSUN - →
  if f1 = f2 ∧ f2 = f3 ∧ f3 = f4 then
  ThoList.flatmap
    (fun (c1, c2, c3, c4) →
      let c1' = c1 in
      List.map (fun (c2', c3', c4') →
        ((CF_io (f1, c1', c2'), CF_io (f2, c3', c1'),
          CF_io (f3, c4', c3'), CF_io (f4, c2', c4')), v, g))
        (permute_triple (c2, c3, c4)))
      color_flow_quadruples
    
```

```

else
  ThoList.flatmap
    (fun (c1, c2, c3, c4) →
      List.map (fun (c1', c2', c3', c4') →
        ((CF_io (f1, c1', c2'), CF_io (f2, c3', c1'),
          CF_io (f3, c4', c3'), CF_io (f4, c2', c4')), v, g))
        (permute_quadruple (c1, c2, c3, c4)))
      color_flow_quadruples

```

The rest is *verboden*!

```

| C.SUN -, (C.Singlet | C.AdjSUN -), (C.Singlet | C.AdjSUN -), (C.Singlet |
C.AdjSUN -)
| (C.Singlet | C.AdjSUN -), C.SUN -, (C.Singlet | C.AdjSUN -), (C.Singlet |
C.AdjSUN -)
| (C.Singlet | C.AdjSUN -), (C.Singlet | C.AdjSUN -), C.SUN -, (C.Singlet |
C.AdjSUN -)
| (C.Singlet | C.AdjSUN -), (C.Singlet | C.AdjSUN -), (C.Singlet |
C.AdjSUN -), C.SUN - →
  colored_vertex "colorize_vertex4:␣single␣quark/anti-quark"
| C.SUN -, C.SUN -, C.SUN -, (C.Singlet | C.AdjSUN -)
| C.SUN -, C.SUN -, (C.Singlet | C.AdjSUN -), C.SUN -
| C.SUN -, (C.Singlet | C.AdjSUN -), C.SUN -, C.SUN -
| (C.Singlet | C.AdjSUN -), C.SUN -, C.SUN -, C.SUN - →
  colored_vertex "colorize_vertex4:␣three␣quarks/anti-quarks"
| C.Singlet, C.Singlet, C.Singlet, C.AdjSUN -
| C.Singlet, C.Singlet, C.AdjSUN -, C.Singlet
| C.Singlet, C.AdjSUN -, C.Singlet, C.Singlet
| C.AdjSUN -, C.Singlet, C.Singlet, C.Singlet →
  colored_vertex "colorize_vertex4:␣single␣gluon"

```

Higher Vertices

```

let colorize_vertexn (flist, v, g) =
  if List.for_all
    (fun f → match M.color f with C.Singlet → true | _ → false)
    flist
  then
    [(List.map (fun f → White f) flist, v, g)]
  else
    incomplete "colorize_vertexn"

```

Discuss with *tho*: Is there possibly a functor that could take a vertex structure and add a singlet ???

```

let vertices () =
  (ThoList.flatmap colorize_vertex3 vertices3,
   ThoList.flatmap colorize_vertex4 vertices4,
   ThoList.flatmap colorize_vertexn verticesn)

```

```

let table = Fusion.of_vertices (vertices ())
let fuse2 = Fusion.fuse2 table
let fuse3 = Fusion.fuse3 table
let fuse = Fusion.fuse table
let max_degree = M.max_degree

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_sans_color_of_string = M.flavor_of_string

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_sans_color_to_string = M.flavor_to_string

```

```

let flavor_to_TeX = function
| White f →
    M.flavor_to_TeX f
| CF_in (f, c) →
    "{" ^ M.flavor_to_TeX f ^ "}" ^ "c" ^ string_of_int c
| CF_out (f, c) →
    "{" ^ M.flavor_to_TeX f ^ "}" ^ "a" ^ string_of_int c
| CF_io (f, c1, c2) →
    "{" ^ M.flavor_to_TeX f ^ "}" ^ "c" ^ string_of_int c1 ^ "a" ^ string_of_int c2 ^ "}"
| CF_aux f →
    "{" ^ M.flavor_to_TeX f ^ "}" ^ "0"

let flavor_sans_color_to_TeX = M.flavor_to_TeX

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 flavor_sans_color_symbol = M.flavor_symbol

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

```

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
    invalid_arg
      "Colorize.It().external_color_flows:␣crossed␣amplitude␣not␣a␣singlet!"
  else if n_in > F.max_lines then
    invalid_arg
      "Colorize.It().external_color_flows:␣too␣few␣color␣lines!"
  else
    let color_strings = ThoList.range 1 n_in in
    List.map
      (fun permutation → (color_strings, permutation))
      (Combinatorics.permute color_strings)
let rec colorize_crossed_amplitude1 f_list (ecf_in, ecf_out) =
  match f_list with
  | f :: rest →
    begin match M.color f with
    | C.Singlet →
      White f :: colorize_crossed_amplitude1 rest (ecf_in, ecf_out)
    | C.SUN nc →
      if nc > 0 then
        CF_in (f, List.hd ecf_in) ::
          colorize_crossed_amplitude1 rest (List.tl ecf_in, ecf_out)
      else if nc < 0 then
        CF_out (f, List.hd ecf_out) ::
          colorize_crossed_amplitude1 rest (ecf_in, List.tl ecf_out)
      else
        su0 "colorize_flavor"
    | C.AdjSUN _ →
      let ecf_in' = List.hd ecf_in
      and ecf_out' = List.hd ecf_out in
      if ecf_in' = ecf_out' then
        CF_aux f ::
          colorize_crossed_amplitude1 rest (List.tl ecf_in, List.tl ecf_out)
      else
        CF_io (f, ecf_in', ecf_out') ::
          colorize_crossed_amplitude1 rest (List.tl ecf_in, List.tl ecf_out)
    end
  | [] →
    begin match ecf_in, ecf_out with
    | [], [] → []
    | _ → invalid_arg "colorize_crossed_amplitude1"
    end
let colorize_crossed_amplitude p_list =

```

```

List.map (colorize_crossed_amplitude1 p_list) (external_color_flows p_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)

let rcs =
  RCS.rename M.rcs
    ("Colorize.It(" ^ string_of_int F.max_lines ^ ", " ^ RCS.name M.rcs ^ ")")
    [String.concat "␣" (RCS.description M.rcs @ ["(statically_colorized)"])]

end

```

10.2.2 (Statically) Colorizing a Monochrome Gauge Model

```

module Gauge (F : Flows) (M : Model.Gauge) =
struct

  module M = M

  module CM = It (F) (M)

  type flavor = CM.flavor
  type flavor_sans_color = CM.flavor_sans_color
  type gauge = CM.gauge
  type constant = CM.constant

  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 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 flavor_sans_color_of_string = CM.flavor_sans_color_of_string
let flavor_sans_color_to_string = CM.flavor_sans_color_to_string
let flavor_sans_color_to_TeX = CM.flavor_sans_color_to_TeX
let flavor_sans_color_symbol = CM.flavor_sans_color_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
let rcs =
  RCS.rename M.rcs
  ("Colorize.Gauge(" ^ string_of_int F.max_lines ^ ", " ^ RCS.name M.rcs ^ ")")
  [String.concat "_" (RCS.description M.rcs @ ["(statically_colorized)"])]
end

```

10.2.3 (Dynamically) Colorizing a Monochrome Model

```

module Dynamical (M : Model.T) =
struct
  module M = M
  open Coupling
  module C = Color

  let incomplete s =
    failwith ("Colorize.It()." ^ s ^ "_not_done_yet!")

  let incomplete s =
    Printf.eprintf "WARNING: _Colorize.It().%s_not_done_yet!\n" s;
    []

  let su0 s =
    invalid_arg ("Colorize.It()." ^ s ^ ": _found_SU(0)!")

  let colored_vertex s =
    invalid_arg ("Colorize.It()." ^ s ^ ": _colored_vertex!")

  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 flavor =
    | White of M.flavor
    | CF_in of M.flavor × int
    | CF_out of M.flavor × int
    | CF_io of M.flavor × int × int
    | 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

```

For the propagator we cannot use pullback because we have to add the case of the color singlet propagator by hand.

```

let colorize_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. *)
| Prop_Col_Scalar | Prop_Col_Feynman | Prop_Col_Majorana |
Prop_Col_Unitarity
→ failwith ("Colorize.It().colorize_propagator: already colored particle!")
| _ → failwith ("Colorize.It().colorize_propagator: impossible!")

let propagator = function
| CF_aux f → colorize_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

```



That's the tricky part: the current implementation of *Fusion.Tagged* needs a list of all flavors. For this we need the list of all color lines ...

```

let flavors () =
  incomplete "flavors"

let external_flavors () =
  incomplete "external_flavors"

let parameters = M.parameters

module Fusion = Modeltools.Fusions (struct
  type f = flavor
  type c = constant
  let compare = compare
  let conjugate = conjugate
end)

```

Vertices



vertices are *only* used by functor applications and for indexing a cache of precomputed fusion rules.

```

let vertices () =
  failwith "Colorize.Dynamical().vertices: no longer supported";
  ([], [], [])

```

Cubic Vertices

The following pattern matches will eventually become quite long. The O’Caml compiler will hopefully optimize them aggressively (<http://pauillac.inria.fr/~maranget/papers/opat/>). If this doesn’t turn out to be the case, there might be an intermediate way using hashtables of functions mapping color flow lines.

```
let colorize_fusion2 f1 f2 (f, v) =
  match M.color f, f1, f2 with
  | C.Singlet, White -, White - → [(White f, v)]
  | C.Singlet, CF_in (_, -), White -
  | C.Singlet, White -, CF_in (_, -) → []
  | C.Singlet, CF_in (_, c1), CF_out (_, c2) →
    if c1 = c2 then [(White f, v)] else []
  | C.SUN -, White f1, White f2 → []
  | C.SUN -, CF_in (_, c1), White -
  | C.SUN -, White -, CF_in (_, c1) → [(CF_in (f, c1), v)]
  | C.SUN -, CF_out (_, c1), White -
  | C.SUN -, White -, CF_out (_, c1) → [(CF_out (f, c1), v)]
  | _ → incomplete "colorize_fusion2"
```

Quartic Vertices

```
let colorize_fusion3 f1 f2 f3 (f, v) =
  match M.color f, f1, f2, f3 with
  | C.Singlet, White f1, White f2, White f3 → [(White f, v)]
  | C.SUN -, White f1, White f2, White f3 → []
  | _ → incomplete "colorize_fusion3"
```

Quintic and Higher Vertices

```
let is_white = function
  | White - → true
  | _ → false

let colorize_fusionn flist (f, v) =
  match M.color f, List.for_all is_white flist with
  | C.Singlet, true → [(White f, v)]
  | C.SUN -, true → []
  | _ → incomplete "colorize_fusionn"

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.Dynamical().fuse"
| [f1; f2] → fuse2 f1 f2
| [f1; f2; f3] → fuse3 f1 f2 f3
| flist → fuse_list flist

let max_degree = M.max_degree

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_sans_color_of_string = M.flavor_of_string

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_sans_color_to_string = M.flavor_to_string
let flavor_to_TeX = function
| White f →
  M.flavor_to_TeX f
| CF_in (f, c) →
  "{" ^ M.flavor_to_TeX f ^ "}" ^ string_of_int c
| CF_out (f, c) →
  "{" ^ M.flavor_to_TeX f ^ "}" ^ string_of_int c
| CF_io (f, c1, c2) →
  "{" ^ M.flavor_to_TeX f ^ "}" ^ string_of_int c1 ^ string_of_int c2
| CF_aux f →
  "{" ^ M.flavor_to_TeX f ^ "}" ^ "0"

let flavor_sans_color_to_TeX = M.flavor_to_TeX
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 flavor_sans_color_symbol = M.flavor_symbol
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

```

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
    invalid_arg
    "Colorize.Dynamical().external_color_flows: crossed amplitude not a singlet!"
  else
    let color_strings = ThoList.range 1 n_in in
    List.map
      (fun permutation → (color_strings, permutation))
      (Combinatorics.permute color_strings)
let rec colorize_crossed_amplitude1 f_list (ecf_in, ecf_out) =
  match f_list with
  | f :: rest →
    begin match M.color f with
    | C.Singlet →
      White f :: colorize_crossed_amplitude1 rest (ecf_in, ecf_out)
    | C.SUN nc →
      if nc > 0 then
        CF_in (f, List.hd ecf_in) ::
          colorize_crossed_amplitude1 rest (List.tl ecf_in, ecf_out)
      else if nc < 0 then
        CF_out (f, List.hd ecf_out) ::
          colorize_crossed_amplitude1 rest (ecf_in, List.tl ecf_out)
      else
        su0 "colorize_flavor"
    | C.AdjSUN _ →
      let ecf_in' = List.hd ecf_in
      and ecf_out' = List.hd ecf_out in
      if ecf_in' = ecf_out' then
        CF_aux f ::
          colorize_crossed_amplitude1 rest (List.tl ecf_in, List.tl ecf_out)
      else
        CF_io (f, ecf_in', ecf_out') ::
          colorize_crossed_amplitude1 rest (List.tl ecf_in, List.tl ecf_out)
    end
  end

```

```

    end
  | [] →
    begin match ecf_in, ecf_out with
    | [], [] → []
    | _ → invalid_arg "colorize-crossed-amplitude1"
    end

let colorize_crossed_amplitude p_list =
  List.map (colorize_crossed_amplitude1 p_list) (external_color_flows p_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)

let rscs =
  RCS.rename M.rscs
    ("Colorize.Dynamical(" ^ RCS.name M.rscs ^ ")")
    [String.concat "⊔" (RCS.description M.rscs @ ["(dynamically_colorized)"])]

end

```

—11—

VERTICES



Temporarily disabled, until, we implement some conditional weaving...

—12—


MODELS





Temporarily disabled, until, we implement some conditional weaving...

—13—


HARDCODED MODELS


 *Interface `models.mli` unavailable!*

 *Implementation `models.ml` unavailable!*

 *Interface `models2.mli` unavailable!*

13.1 *Implementation of Models2*

 *Interface `models2.mli` unavailable!*

 *Implementation `models2.ml` unavailable!*

—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 = "$Revision: 810$";
    RCS.date = "$Date: 2009-06-13 23:56:23 +0200 (Sat, 13 Jun 2009)$";
    RCS.author = "$Author: ohl$";
    RCS.source
      = "$URL: svn+ssh://jr-reuter@login.hepforge.org/hepforge/svn/whizard/trunk/src/omeg
```

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 flavor_sans_color = flavor
  let flavor_sans_color f = f
  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 conjugate_sans_color = 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 flavor_sans_color_of_string = flavor_of_string
  let flavor_sans_color_to_string = flavor_to_string
  let flavor_sans_color_to_TeX = flavor_to_TeX
  let flavor_sans_color_symbol = 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

```

```

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_lagrangian_row = function
| p1 :: p2 :: p3 :: p4 :: c :: t :: _ →

```

```

      ((p1, p2, p3, p4), parse_expr c, parse_expr t)
    | _ → invalid_arg "parse-lagrangian-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_lagrangian_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.create (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 "prtc1s1.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
```

15.1.2 Potential Targets

```
module VM : Target.Maker
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 rsc_file = RCS.parse "Targets" ["Code_Generation"]
  { RCS.revision = "$Revision: 1655$";
    RCS.date = "$Date: 2010-02-02 16:50:32 +0100 (Tue, 02 Feb 2010)$";
    RCS.author = "$Author: ohl$";
    RCS.source
      = "$URL: svn+ssh://jr_reuter@login.hepforge.org/hepforge/svn/whizard/trunk/src/omeg";
  }
module Dummy (F : Fusion.Maker) (P : Momentum.T) (CM : Model.Colorized) =
  struct
    let rsc_list = []
    type amplitudes = Fusion.Colored(F)(P)(CM).amplitudes
```

```

type diagnostic = All | Arguments | Momenta | Gauge
let options = Options.empty
let amplitudes_to_channel cmdline oc amplitudes = failwith "Targets.Dummy"
let parameters_to_channel oc = failwith "Targets.Dummy"
end

```

15.2.1 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_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 =
  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 → 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_current = function
| coeff, Psibar, VA, Psi → print_fermion_current2 coeff "va"
| coeff, Psibar, VA2, Psi → print_fermion_current2 coeff "va2"
| 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"
| coeff, 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_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_2009_06_A"; "omega_spinor_cpls_2009_06_A"]
end

```

Main Functor

```

module Make_Fortran (Fermions : Fermions)
  (Fusion_Maker : Fusion.Maker) (P : Momentum.T) (CM : Model.Colorized) =
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_2009_06_A"; "omega_polarizations_2009_06_A";
      "omega_couplings_2009_06_A"; "omega_utils_2009_06_A" ]

  module F = Fusion_Maker(P)(CM)
  type amplitude = F.amplitude

  module CF = Fusion.Colored(Fusion_Maker)(P)(CM)
  type amplitudes = CF.amplitudes

  open Coupling
  open Format

  let line_length = ref 80
  let kind = ref "default"
  let fortran95 = ref true
  let module_name = ref "omega_amplitude"
  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 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), "approx. line length";
  "module", Arg.String (fun s → module_name := s), "module name";
  "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"]

```

Fortran style line continuation:

```

let continuing = ref true

let nl () =
  continuing := false;
  print_newline ();
  continuing := true

let wrap_newline () =
  let out, flush, newline, space = get_all_formatter_output_functions () in
  let newline' () = if !continuing then out "&" 0 2; newline () in
  set_all_formatter_output_functions out flush newline' space

let print_list = function
| [] → ()
| a :: rest →
  print_string a;
  List.iter (fun s → printf ",@_%s" s) rest

```

Variables and Declarations

```

let flavors_symbol flavors =
  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 WFSets = Set.Make (struct type t = F.wf let compare = compare end)
module WFSets2 = Set.Make (struct type t = F.wf × F.wf Tree2.t let compare = compare end)
module WFMap = Map.Make (struct type t = F.wf let compare = compare end)
module WFMap2 = Map.Make (struct type t = F.wf × F.wf Tree2.t let compare = compare end)
module WFTSet = Set.Make (struct type t = F.wf Tree2.t let compare = compare end)

let add_tag wf name =
  match F.wf_tag wf with
  | None → name
  | Some tag → name ^ "_" ^ tag

let variable wf =
  add_tag wf (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 wf i =
  variable wf ^ "_X" ^ string_of_int i

let multiple_variable amplitude dictionary wf =
  try
    format_multiple_variable wf (WFMap2.find (wf, F.dependencies amplitude wf) dictionary)
  with
  | Not_found → variable wf

let multiple_variables multiplicities wf =
  try
    List.map
      (format_multiple_variable wf)
      (ThoList.range 1 (WFMap.find wf multiplicities))
  with
  | Not_found → [variable wf]

let declare_list multiplicities t = function
  | [] → ()
  | wfs →
    printf "UUUU@ [<2>%sU: :U" t;
    print_list (ThoList.flatmap (multiple_variables multiplicities) wfs); nl ()

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  $\alpha$  parameters =
  { real_singles :  $\alpha$  list;
    real_arrays : ( $\alpha \times \text{int}$ ) list;
    complex_singles :  $\alpha$  list;
    complex_arrays : ( $\alpha \times \text{int}$ ) list }

let rec classify_singles acc = function
| []  $\rightarrow$  acc
| Real p :: rest  $\rightarrow$  classify_singles
  { acc with real_singles = p :: acc.real_singles } rest
| Complex p :: rest  $\rightarrow$  classify_singles
  { acc with complex_singles = p :: acc.complex_singles } rest

let rec classify_arrays acc = function
| []  $\rightarrow$  acc
| (Real_Array p, rhs) :: rest  $\rightarrow$  classify_arrays
  { acc with real_arrays =
    (p, List.length rhs) :: acc.real_arrays } rest
| (Complex_Array p, rhs) :: rest  $\rightarrow$  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

let rec schisma n l =
  if List.length l  $\leq$  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  $\leq$  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 "%s@(<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 "%s@(<2>%s(kind=%s),_dimension(%d),_public,_save_::" t !kind n (CM.constant_symbol p); nl ()

let default_parameter (x, v) =
  printf "%s@s=%s" (CM.constant_symbol x) v !kind

let declare_default_parameters t = function
| [] → ()
| p :: plist →
  printf "%s@(<2>%s(kind=%s),_public,_save_::" t !kind;
  default_parameter p;
  List.iter (fun p' → printf ","; default_parameter p') plist;
  nl ()

let rec 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 "%s,("; eval_parameter' x;
  List.iter (fun x → printf "%s,+"; eval_parameter' x) xs;
  printf ")"
| Diff (x, y) →
  printf "%s,("; eval_parameter' x;
  printf "%s-"; eval_parameter' y; printf ")"
| Neg x → printf "%s,(_"; eval_parameter' x; printf ")"
| Prod [] → printf "1.0-_%s" !kind
| Prod [x] → eval_parameter' x
| Prod (x :: xs) →
  printf "%s,("; eval_parameter' x;

```

```

    List.iter (fun x → printf "%s" x; eval_parameter' x) xs;
    printf ")"
  | Quot (x, y) →
    printf "@,("; eval_parameter' x;
    printf "%s/%s"; eval_parameter' y; printf ")")
  | Rec x →
    printf "@,%(1.0-%s/%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 "%s,%s"; eval_parameter' x; printf ")")
  | Conj x → printf "@,conj("; 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 "%%s@(<2>%s=%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 "%%s@(<2>%s(%d)%s=%s" 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 =
  set_formatter_out_channel oc;
  set_margin !line_length;
  wrap_newline ();
  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 "!_0'Mega_revision_control_information:"; nl ();
List.iter (fun s → printf "!!!!_s" s; nl ())
  (ThoList.flatmap RCS.summary (CM.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 = function
| [] → ()
| momenta →
    printf "uuuu@ [<2>type(momentum)u::u";
    print_list (List.map format_momentum momenta); nl ()

let declare_wavefunctions multiplicities wfs =
let wfs' = classify_wfs wfs in
declare_list multiplicities ("complex(kind=" ^ !kind ^ ")")
(wfs'.scalars @ wfs'.brs_scalars);
declare_list multiplicities ("type(" ^ Fermions.psi_type ^ ")")
(wfs'.spinors @ wfs'.brs_spinors);
declare_list multiplicities ("type(" ^ Fermions.psibar_type ^ ")")
(wfs'.conjspinors @ wfs'.brs_conjspinors);
declare_list multiplicities ("type(" ^ Fermions.chi_type ^ ")")
(wfs'.realspinors @ wfs'.brs_realspinors @ wfs'.ghostspinors);
declare_list multiplicities ("type(" ^ Fermions.grav_type ^ ")") wfs'.vectorspinors;
declare_list multiplicities "type(vector)" (wfs'.vectors @ wfs'.massive_vectors @
wfs'.brs_vectors @ wfs'.brs_massive_vectors @ wfs'.ward_vectors);
declare_list multiplicities "type(tensor2odd)" wfs'.tensors_1;
declare_list multiplicities "type(tensor)" wfs'.tensors_2

let flavors a = F.incoming a @ F.outgoing a

let declare_brackets = function
| [] → ()

```

```

| amplitudes →
  printf "UUUU@ [<2>complex(kind=%s)U::U" !kind;
  print_list (List.map (fun a → flavors_symbol (flavors a)) amplitudes); nl ()
let print_declarations2 multiplicities dictionary amplitudes =
  declare_momenta
    (PSet.elements
      (ThoList.fold_left2
        (fun set a →
          PSet.union set (List.fold_right
            (fun wf → PSet.add (F.momentum_list wf))
            (F.externals a) PSet.empty))
        PSet.empty amplitudes));
  declare_wavefunctions multiplicities
    (WFSet.elements
      (ThoList.fold_left2
        (fun set a →
          WFSet.union set (List.fold_right WFSet.add (F.externals a) WFSet.empty))
          WFSet.empty amplitudes));
  declare_momenta
    (PSet.elements
      (ThoList.fold_left2
        (fun set a →
          PSet.union set (List.fold_right
            (fun wf → PSet.add (F.momentum_list wf))
            (F.variables a) PSet.empty))
        PSet.empty amplitudes));
  declare_wavefunctions multiplicities
    (WFSet.elements
      (ThoList.fold_left2
        (fun set a →
          WFSet.union set (List.fold_right WFSet.add (F.variables a) WFSet.empty))
          WFSet.empty amplitudes));
  declare_brackets (List.flatten amplitudes)

```

`print_current` is the most important function that has to match the functions in `omega95` (see appendix T). 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:Ucan't happen"

let children3 rhs =
  match F.children rhs with
  | [wf1; wf2; wf3] → (wf1, wf2, wf3)
  | - → invalid_arg "Targets.children3:Ucan'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 automatically.

```

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 $\bar{\psi}A\psi$ 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) →
    Fermions.print_current (coeff, fb, b, f) c wf1 wf2 fusion
  | 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 ?? is a bit misleading, since it 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

$$(V_2 \wedge V_3) \cdot T_1 = \begin{cases} V_2 \cdot (T_1 \cdot V_3) = -V_2 \cdot (V_3 \cdot T_1) \\ V_3 \cdot (V_2 \cdot T_1) = -V_3 \cdot (T_1 \cdot V_2) \end{cases} \quad (15.1)$$

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_kv v(%s,%s,%s,%s,%s)" c wf1 p1 wf2 p2
  | F21 → printf "tv_kv v(%s,%s,%s,%s,%s)" c wf2 p2 wf1 p1
  | F13 → printf "(-1)*tv_kv v(%s,%s,%s,%s,%s)" c wf1 p1 wf2 p2
  | F31 → printf "(-1)*tv_kv v(%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_kv v(%s,%s,%s,%s,%s)" c wf1 p1 wf2
  | F21 | F31 → printf "lv_kv v(%s,%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_kv v(%s,%s,%s,%s,%s)" c wf1 p1 wf2 p2
  | F21 | F31 → printf "t5v_kv v(%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_kv v(%s,%s,%s,%s,%s)" c wf1 p1 wf2
  | F21 → printf "l5v_kv v(%s,%s,%s,%s,%s)" c wf2 p2 wf1
  | F13 → printf "(-1)*l5v_kv v(%s,%s,%s,%s,%s)" c wf1 p1 wf2
  | F31 → printf "(-1)*l5v_kv v(%s,%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

| 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"
    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

| 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

| 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" c wf1 wf2 wf3
  | F314 | F413 | F324 | F423 →
    printf "%s*%s*%s" c wf2 wf1 wf3
  | F341 | F431 | F342 | F432 →
    printf "%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 (disc, 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)@,%*"
    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, p_4 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,%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 "("
| 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 xi →
    invalid_arg "no⊔on-shell⊔massless⊔gauge⊔propagator!"
  | Prop_Rxi xi →
    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 "("
  | 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,%s," Fermions.psi_projector p m gamma
| Prop_ConjSpinor →
    printf "%s(%s,%s,%s,%s," Fermions.psibar_projector p m gamma
| Prop_Majorana →
    printf "%s(%s,%s,%s,%s," Fermions.chi_projector p m gamma
| Prop_Col_Majorana →
    printf "%s*%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 xi →
    invalid_arg "no_on-shell_massless_gauge_propagator!"
| Prop_Rxi xi →
    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 "UUUUUU@(<2>%s=" v;
        List.iter (print_current amplitude dictionary) (F.rhs fusion); nl();
        begin match CM.goldstone f with
        | None →
            printf "UUUUUUcall_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 "UUUUUUcall_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 dependencies = F.dependencies amplitude lhs in

```

```

let f = F.flavor lhs in
printf "uuuuu@ [<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 "uuu%s=" (momentum wf)
        (String.concat "++"
          (List.map momentum (F.children rhs1))); nl ()
    end;
    PSet.add p seen)
  seen_momenta (F.fusions amplitude)

```

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))
    (List.flatten 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
        WFTSet.add wf (WFTSet.add dependencies set) map)
      WFTSet.empty fusions in
  let multiplicities =
    WFTSet.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 =
    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
    (multiplicities, dictionary)
let print_fusions dictionary seen_wfs amplitude =
  List.fold_left (fun seen f →
    let wf = F.lhs f in
    let dependencies = F.dependencies amplitude wf in
    let p = F.momentum_list wf in
    if ¬ (WFTSet2.mem (wf, dependencies) seen) then begin
      print_fusion_diagnostics amplitude dictionary f;
      print_fusion amplitude dictionary f;
    end;
    WFTSet2.add (wf, dependencies) seen)
  seen_wfs (F.fusions amplitude)
let print_braket amplitude dictionary name braket =
  let bra = F.bra braket
  and ket = F.ket braket in
  printf "UUUUUU@ [<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 ()

```

$$iT = i^{\# \text{vertices}} i^{\# \text{propagators}} \dots = i^{n-2} i^{n-3} \dots = -i(-1)^n \dots \quad (15.2)$$



tho : we write some brackets twice using different names. Is it useful to cache them?

```

let print_brackets dictionary amplitude =
  let name = flavors_symbol (flavors amplitude) in
  printf "~~~~~%s=0" name; nl ();
  List.iter (print_braket amplitude dictionary name) (F.brackets amplitude);
  let n = List.length (F.externals amplitude) in
  if n mod 2 = 0 then begin
    printf "~~~~~%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 "~~~~~%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 "conj⊔(eps⊔(%s,⊔%s,⊔%s))" m p s
| BRS Vector | BRS Massive_Vector → printf
    "(0,1)⊔*(%s*%s-%s**2)⊔*(conj⊔(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 "conj⊔(eps2⊔(%s,⊔%s,⊔%s))" m p s
| BRS _ → invalid_arg "no⊔such⊔BRST⊔transformations"

let twice_spin wf =
match CM.lorentz (F.flavor wf) with
| Scalar | BRS Scalar → "0"
| Spinor | ConjSpinor | Majorana | Maj_Ghost | Vectorspinor
| BRS Spinor | BRS ConjSpinor | BRS Majorana | BRS Vectorspinor →
"1"
| Vector | BRS Vector | Massive_Vector | BRS Massive_Vector →
"2"
| Tensor_1 → "2"
| Tensor_2 → "4"
| BRS _ → invalid_arg "Targets.twice_spin:⊔no⊔such⊔BRST⊔transformation"

let print_argument_diagnostics amplitude =
let externals = (F.externals amplitude) in
let n = List.length externals
and masses =
List.map (fun wf → CM.mass_symbol (F.flavor wf)) externals
and spins = List.map twice_spin externals in
if warn diagnose_arguments then begin
printf "⊔⊔⊔⊔call⊔omega_check_arguments-%s⊔(%d,⊔k)"
(suffix diagnose_arguments) n; nl ()

```

```

end;
if warn diagnose_momenta then begin
  printf "%%@ [<2> call_omega_check_momenta_%s (/ "
    (suffix diagnose_momenta);
  print_list masses;
  printf "%k)"; nl ()
end
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 "%%s=" (variable wf);
      (if incoming then print_incoming else print_outgoing) wf; nl ()
    end;
    WFSet.add wf seen) seen_wfs externals
let flavors_to_string flavors =
  String.concat " " (List.map CM.flavor_to_string flavors)
let process_to_string amplitude =
  flavors_to_string (F.incoming amplitude) ^ "→" ^
  flavors_to_string (F.outgoing amplitude)
let flavors_sans_color_to_string flavors =
  String.concat " " (List.map CM.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 mark_one_table_entry flavor color amplitude =
    printf "~~~~~amp (%d, h, %d) = %s"
        color flavor (flavors_symbol (flavors amplitude));
    nl ()

let print_amplitudes amplitudes =
    printf "~~@ [<5>"; if !fortran95 then printf "pure_";
    printf "subroutine_calculate_amplitudes (amp, k, hel_mask)"; nl ();
    printf "~~~~~complex (kind = default), dimension (:, :, :), intent (out) _ :: _amp"; nl ();
    printf "~~~~~real (kind = default), dimension (0:3, *), intent (in) _ :: _k"; nl ();
    printf "~~~~~logical, dimension (:), intent (in) _ :: _hel_mask"; nl ();
    printf "~~~~~integer, dimension (n_prt) _ :: _s"; nl ();
    printf "~~~~~integer _ :: _h"; nl ();
    let multiplicities, dictionary = disambiguate_fusions (CF.processes amplitudes) in
    print_declarations2 multiplicities dictionary (CF.processes amplitudes);
    List.iter (List.iter print_argument_diagnostics) (CF.processes amplitudes);
    begin match CF.processes amplitudes with
    | (p :: _) :: _ → print_external_momenta p
    | _ → ()
    end;
    ignore (ThoList.fold_left2 print_momenta PSet.empty (CF.processes amplitudes));
    printf "~~~~~do_h = 1, _hel"; nl ();
    printf "~~~~~if (hel_mask(h)) then"; nl ();
    printf "~~~~~s = table_spin_states (:, h)"; nl ();
    ignore (ThoList.fold_left2 print_externals WSet.empty (CF.processes amplitudes));
    ignore (ThoList.fold_left2
        (print_fusions dictionary)
        WSet2.empty (CF.processes amplitudes));
    List.iter (List.iter (print_brackets dictionary)) (CF.processes amplitudes);
    List.iter (List.iter print_fudge_factor) (CF.processes amplitudes);
    ThoList.iteri2 mark_one_table_entry 1 1 (CF.processes amplitudes);
    printf "~~~~~else"; nl ();
    printf "~~~~~amp (:, h, :) = 0"; nl ();
    printf "~~~~~end_if"; nl ();
    printf "~~~~~end_do"; nl ();
    printf "~~end_subroutine_calculate_amplitudes"; nl ();
    nl ()

```

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 not 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 thrice the same algorithm.

```

let print_integer_parameter name value =
  printf "%u@ [<2>integer, parameter, private_u: :u%s_u=%u%d" name value; nl ()

let print_logical_parameter name value =
  printf "%u@ [<2>logical, parameter, private_u: :u%s_u=.%.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 cflows =
  List.length cflows

let num_color_indices_default = 2 (* Standard model *)

let num_color_indices cflows =
  try CFlow.rank (List.hd cflows) 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 print_spin_table abbrev name = function
  | [] →

```

```

    printf "%u@ [<2>integer, dimension(n_prt,0), private_::";
    printf "@table_spin_%s" name; nl ()
| - :: tuples' as tuples →
    ignore (List.fold_left (fun i (tuple1, tuple2) →
        printf "%u@ [<2>integer, dimension(n_prt), parameter, private_::";
        printf "@_s%04d=_(/_s_/" abbrev i
            (String.concat ",_" (List.map (Printf.sprintf "%2d") (tuple1 @ tuple2)));
        nl (); succ i) 1 tuples);
    printf
        "%u@ [<2>integer, dimension(n_prt, n_hel), parameter, private_::";
    printf "@table_spin_%s=@_reshape_(/_/" name;
    printf "@_s%04d" abbrev 1;
    ignore (List.fold_left (fun i tuple →
        printf ",@_s%04d" abbrev i; succ i) 2 tuples');
    printf "@_/,_(/_n_prt,_n_hel_/_)" ; nl ()

let print_spin_tables amplitudes =
    print_spin_table "s" "states" (CF.helicities amplitudes);
    nl ()

let num_helicities amplitudes =
    List.length (CF.helicities amplitudes)

let print_flavor_table n abbrev name = function
| [] →
    printf "%u@ [<2>integer, dimension(n_prt,0), private_::";
    printf "@table_flavor_%s" name; nl ()
| - :: tuples' as tuples →
    ignore (List.fold_left (fun i tuple →
        printf
            "%u@ [<2>integer, dimension(n_prt), parameter, private_::";
        printf "@_s%04d=_(/_s_/_)!_s" abbrev i
            (String.concat ",_"
                (List.map (fun f → Printf.sprintf "%3d" (CM.M.pdg f)) tuple))
            (String.concat "_" (List.map CM.M.flavor_to_string tuple)));
        nl (); succ i) 1 tuples);
    printf
        "%u@ [<2>integer, dimension(n_prt, n_flv), parameter, private_::";
    printf "@table_flavor_%s=@_reshape_(/_/" name;
    printf "@_s%04d" abbrev 1;
    ignore (List.fold_left (fun i tuple →
        printf ",@_s%04d" abbrev i; succ i) 2 tuples');
    printf "@_/,_(/_n_prt,_n_flv_/_)" ; nl ()

let print_flavor_tables amplitudes =
    let n = num_particles amplitudes in
    print_flavor_table n "f" "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 abbrev = function
| [] →
  printf "%d@[%d>integer,dimension(n_cindex,n_prt,n_cflow),parameter,private_:";
  printf "@table_color_flows"; nl ()
| - :: tuples' as tuples →
  ignore (List.fold_left (fun i tuple →
    printf
      "%d@[%d>integer,dimension(n_cindex,n_prt),parameter,private_:";
      printf "@s%04d=_reshape_(_/_" abbrev i;
      begin match CFlow.to_lists tuple with
      | [] → ()
      | cf1 :: cf_n →
        printf "@s" (String.concat " " (List.map string_of_int cf1));
        List.iter (function cf →
          printf ",@s" (String.concat " " (List.map string_of_int cf))) cf_n
        end;
        printf "@_/_),@_(_/_n_cindex,n_prt/_/_)");
        nl (); succ i) 1 tuples);
  printf
    "%d@[%d>integer,dimension(n_cindex,n_prt,n_cflow),parameter,private_:";
    printf "@table_color_flows=_reshape_(_/_";
    printf "@s%04d" abbrev 1;
    ignore (List.fold_left (fun i tuple →
      printf ",@s%04d" abbrev i; succ i) 2 tuples');
    printf "@_/_),@_(_/_n_cindex,n_prt,n_cflow/_/_)"); nl ()

let print_ghost_flags_table abbrev = function
| [] →
  printf "%d@[%d>logical,dimension(n_prt,n_cflow),parameter,private_:";
  printf "@table_ghost_flags"; nl ()
| - :: tuples' as tuples →
  ignore (List.fold_left (fun i tuple →
    printf
      "%d@[%d>logical,dimension(n_prt),parameter,private_:";
      printf "@s%04d=_(_/_" abbrev i;
      begin match CFlow.ghost_flags tuple with
      | [] → ()
      | gf1 :: gf_n →
        printf "@s" (if gf1 then "T" else "F");
        List.iter (function gf → printf ",@s" (if gf then "T" else "F")) gf_n
        end;
        printf "@_/_)";
        nl (); succ i) 1 tuples);
  printf
    "%d@[%d>logical,dimension(n_prt,n_cflow),parameter,private_:";
    printf "@table_ghost_flags=_reshape_(_/_";
    printf "@s%04d" abbrev 1;
    ignore (List.fold_left (fun i tuple →
      printf ",@s%04d" abbrev i; succ i) 2 tuples');
    printf "@_/_),@_(_/_n_prt,n_cflow/_/_)"); nl ()

```

```

let print_color_tables cflows =
  print_color_flows_table "c" cflows;
  print_ghost_flags_table "g" cflows;
  nl (); nl ()

let print_amplitude_table () =
  printf
    "%u@ [<2>complex(kind=default),_dimension(n_cflow,_n_hel,_n_flg),_private,_save:::_"
  nl ();
  nl ()

let print_helicity_selection_table () =
  printf "%u@ [<2>logical,_dimension(n_hel),_private,_save:::_"
  printf "hel_is_allowed=_true."; nl ();
  printf "%u@ [<2>real(kind=default),_dimension(n_hel),_private,_save:::_"
  printf "hel_max_abs=_0"; nl ();
  printf "%u@ [<2>real(kind=default),_private,_save:::_"
  printf "hel_sum_abs=_0,_";
  printf "hel_threshold=_1E10"; nl ();
  printf "%u@ [<2>integer,_private,_save:::_"
  printf "hel_count=_0,_";
  printf "hel_cutoff=_100"; nl ();
  nl ()

```

Optional MD5 sum function

```

let print_md5sum_functions () =
  match !md5sum with
  | Some s →
    begin
      printf "%u@ [<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 ()
    end
  | None → ()

```

Maintenance & Inquiry Functions

```

let print_maintenance_functions () =
  if !whizard then begin
    printf "%u_subroutine_init_(par)"; nl ();
    printf "!!!!real(default),_dimension(*),_intent(in)_::_par"; nl ();
    printf "!!!!call_import_from_whizard_(par)"; nl ();
  end

```

```

    printf "end_subroutine_init"; nl ();
    nl ();
    printf "subroutine_final()"; nl ();
    printf "if(hel_threshold.gt..0)then"; nl ();
    printf "call_omega_report_helicity_selection";
    printf "(hel_is_allowed,table_spin_states,hel_threshold)"; nl ();
    printf "end_if"; nl ();
    printf "end_subroutine_final"; nl ();
    nl ();
    printf "subroutine_update_alpha_s(alpha_s)"; nl ();
    printf "real(default),intent(in)::alpha_s"; nl ();
    printf "call_model_update_alpha_s(alpha_s)"; nl ();
    printf "end_subroutine_update_alpha_s"; nl ();
    nl ();
end

let print_inquiry_function_declarations name =
    printf "@[<2>public::number_%s,@%s" name name;
    nl ()

let print_numeric_inquiry_functions () =
    printf "@[<5>"; if !fortran95 then printf "pure";
    printf "function_number_particles_in()result(n)"; nl ();
    printf "integer::n"; nl ();
    printf "n=n_in"; nl ();
    printf "end_function_number_particles_in"; nl ();
    nl ();
    printf "@[<5>"; if !fortran95 then printf "pure";
    printf "function_number_particles_out()result(n)"; nl ();
    printf "integer::n"; nl ();
    printf "n=n_out"; nl ();
    printf "end_function_number_particles_out"; 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 "uuuuu_n=_size_(table_color_flows,_dim=1)"; nl ();
    printf "uuend_function_number_color_indices"; nl ();
    nl ();
    printf "uu@ [<5>"; if !fortran95 then printf "pure_";
    printf "function_number_color_flows_( )_result_(n)"; nl ();
    printf "uuuuinteger_::_n"; nl ();
    printf "uuuuu_n=_size_(table_color_flows,_dim=3)"; nl ();
    printf "uuend_function_number_color_flows"; nl ();
    nl ();
    printf "uu@ [<5>"; if !fortran95 then printf "pure_";
    printf "subroutine_color_flows_(a,_g)"; nl ();
    printf "uuuuinteger,_dimension(:, :, :),_intent(out)_::_a"; nl ();
    printf "uuuulogical,_dimension(:, :),_intent(out)_::_g"; nl ();
    printf "uuuuu_a=_table_color_flows"; nl ();
    printf "uuuuu_g=_table_ghost_flags"; nl ();
    printf "uuend_subroutine_color_flows"; nl ();
    nl ();

let print_dispatch_functions () =
    printf "uu@ [<5>";
    printf "subroutine_new_event_(p)"; nl ();
    printf "uuuuu_real(kind=default),_dimension(0:3,*),_intent(in)_::_p"; nl ();
    printf "uuuuu_call_calculate_amplitudes_(amp,_p,_hel_is_allowed)"; nl ();
    printf "uuuuu_if_(hel_threshold_.gt._0)_.and._(hel_count_.le._hel_cutoff))_then"; nl ();
    printf "uuuuuuu_call_omega_update_helicity_selection_(hel_count,_amp,_";
    printf "hel_max_abs,_hel_sum_abs,_hel_is_allowed,_hel_threshold,_hel_cutoff)"; nl ();
    printf "uuuuu_end_if"; nl ();
    printf "uuend_subroutine_new_event"; nl ();
    nl ();
    printf "uu@ [<5>";
    printf "subroutine_reset_helicity_selection_(threshold,_cutoff)"; nl ();
    printf "uuuuu_real(kind=default),_intent(in)_::_threshold"; nl ();
    printf "uuuuu_integer,_intent(in)_::_cutoff"; nl ();
    printf "uuuuu_hel_is_allowed=_true."; nl ();
    printf "uuuuu_hel_max_abs=_0"; nl ();
    printf "uuuuu_hel_sum_abs=_0"; nl ();
    printf "uuuuu_hel_count=_0"; nl ();
    printf "uuuuu_hel_threshold=_threshold"; nl ();
    printf "uuuuu_hel_cutoff=_cutoff"; nl ();
    printf "uuend_subroutine_reset_helicity_selection"; nl ();
    nl ();
    printf "uu@ [<5>"; if !fortran95 then printf "pure_";
    printf "function_is_allowed_(flv,_hel,_col)"; nl ();
    printf "uuuuu_logical_::_is_allowed"; nl ();
    printf "uuuuu_integer,_intent(in)_::_flv,_hel,_col"; nl ();
    printf "uuuuu_is_allowed=_hel_is_allowed_(hel)"; nl ();
    printf "uuend_function_is_allowed"; nl ();
    nl ();
    printf "uu@ [<5>"; if !fortran95 then printf "pure_";
    printf "function_get_amplitude_(flv,_hel,_col)_result_(amp_result)"; nl ();

```

```

printf "UUUUcomplex(kind=default)U::Uamp_result"; nl ();
printf "UUUUinteger,Uintent(in)U::Uflv,Uhel,Ucol"; nl ();
printf "UUUUamp_resultU=Uamp(col,Uhel,Uflv)"; nl ();
printf "UUend_Ufunction_Uget_amplitude"; nl ();
nl ()

```

Main Function

```

let print_description cmdline amplitudes =
  printf "!_File_generated_automatically_by_0'Mega"; nl();
  printf "!"; nl();
  printf "UUU%s" cmdline; nl();
  printf "!"; nl();
  printf "!_with_all_scattering_amplitudes_for_the_process(es)"; nl ();
  printf "!"; nl ();
  printf "UUUcontributing:"; nl ();
  printf "!"; nl ();
  List.iter
    (fun process →
      printf "UUUUU%s" (process_sans_color_to_string process); nl ())
    (CF.flavors amplitudes);
  printf "!"; nl ();
  List.iter
    (fun cflow → printf "UUUUU%s" (cflow_to_string cflow); nl ())
    (CF.color_flows amplitudes);
  printf "!"; nl ();
  printf "UUUvanishing:"; nl ();
  printf "!"; nl ();
  List.iter (fun process →
    printf "UUUUU%s" (process_sans_color_to_string process); nl ())
    (CF.vanishing_flavors amplitudes);
  printf "!"; nl ();
  List.iter
    (fun cflow → printf "UUUUU%s" (cflow_to_string cflow); nl ())
    (CF.vanishing_color_flows amplitudes);
  printf "!"; nl ();
  begin
    match CF.constraints amplitudes with
    | None → ()
    | Some s →
      printf
        "UUUUdiagram_selection_(MIGHT_BREAK_GAUGE_INVARIANCE!!!):"; nl ();
      printf "!"; nl ();
      printf "UUUUU%s" s; nl ();
      printf "!"; nl ()
  end;
  begin match RCS.description CM.rcs with
  | line1 :: lines →

```

```

    printf "!in%s" line1; nl ();
    List.iter (fun s → printf "!%s" s; nl ()) lines
  | [] → printf "!in%s" (RCS.name CM.rcs); nl ()
end;
printf "!"; nl ()

let print_version () =
  printf "!0'Mega_revision_control_information:"; nl ();
  List.iter (fun s → printf "!%s" s; nl ())
    (ThoList.flatmap RCS.summary (CM.rcs :: rcs_list @ F.rcs_list))

let print_public = function
  | name1 :: names →
    printf "%s@(<2>public::%s" name1;
    List.iter (fun n → printf ",@%s" n) names; nl ()
  | [] → ()

let print_public_interface generic procedures =
  printf "%spublic::%s" generic; nl ();
  begin match procedures with
  | name1 :: names →
    printf "%sinterface%s" generic; nl ();
    printf "%s@(<2>module_procedure%s" name1;
    List.iter (fun n → printf ",@%s" n) names; nl ();
    printf "%send_interface"; nl ();
    print_public procedures
  | [] → ()
  end

let print_module_header amplitudes =
  let cflows = CF.color_flows amplitudes in
  printf "module%s" !module_name; nl (); nl ();
  List.iter (fun s → printf "%s" s; nl ())
    (["kinds"; Fermions.use_module] @
     !use_modules); nl ();
  if ((String.length !parameter_module) > 0) then
    printf "%s" !parameter_module; nl (); nl ();
    printf "%simplicit_none"; nl ();
    printf "%sprivate"; nl (); nl ();
    begin match !md5sum with
    | Some _ → print_public ["md5sum"]
    | None → ()
    end;
  print_public ["number_particles_in"; "number_particles_out"];
  List.iter print_inquiry_function_declarations
    ["spin_states"; "flavor_states"; "color_flows"];
  print_public ["number_color_indices"];
  if !whizard then
    print_public ["init"; "final"; "update_alpha_s"];
  print_public
    ["new_event"; "reset_helicity_selection"; "is_allowed"; "get_amplitude"]; nl ();
  printf "%s!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)
[ ("n_prt", num_particles amplitudes);
  ("n_in", num_particles_in amplitudes);
  ("n_out", num_particles_out amplitudes);
  ("n_cflow", num_color_flows cflows); (* Number of different color
amplitudes. *)
  ("n_cindex", num_color_indices cflows); (* Maximum rank of color
tensors. *)
  ("n_flv", num_flavors amplitudes); (* Number of different flavor
amplitudes. *)
  ("n_hel", num_helicities amplitudes) (* Number of different helicity
amplitudes. *) ];
nl ();

```

Abbreviations.

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 cflows;
print_amplitude_table ();
print_helicity_selection_table ();
printf "contains"; nl (); nl ();
print_md5sum_functions ();
print_maintenance_functions ();
print_numeric_inquiry_functions ();
List.iter print_inquiry_functions
["spin_states", "flavor_states"];
print_color_flows ();
print_dispatch_functions ()

let print_module_footer () =
  printf "end_module_%s" !module_name; nl ()

let amplitudes_to_channel cmdline oc diagnostics amplitudes =
  set_formatter_out_channel oc;
  set_margin !line_length;
  wrap_newline ();

```

```

    parse_diagnostics diagnostics;
    print_description cmdline amplitudes;
    print_module_header amplitudes;
    if !km_write ∨ !km_pure then
        Targets_Kmatrix.Fortran.print !km_pure;
    print_amplitudes amplitudes;
    print_module_footer ();
    print_version ();
    print_flush ()

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,%s)" f1 c wf1 wf2
| F31 → printf "%s_ff(-(s),%s,%s,%s)" f2 c wf1 wf2
| F23 → printf "f_-%sf(%s,%s,%s,%s)" f1 c wf1 wf2
| F32 → printf "f_-%sf(%s,%s,%s,%s)" f1 c wf2 wf1
| F12 → printf "f_-%sf(-(s),%s,%s,%s)" f2 c wf2 wf1
| F21 → printf "f_-%sf(-(s),%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"
| coeff, -, -, - → 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"
| coeff, -, -, - → 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"
| coeff, -, -, - → 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 distinguish 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,%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,%s)" f c wf1 wf2 p1
  | F32 → printf "f_%sf(%s,%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,%s)" f c wf1 wf2 p12
  | F31 → printf "%s_fgr(%s,%s,%s,%s,%s)" f c wf1 wf2 p12
  | F23 → printf "gr_%sf(%s,%s,%s,%s,%s)" f c wf1 wf2 p1
  | F32 → printf "gr_%sf(%s,%s,%s,%s,%s)" f c wf2 wf1 p2
  | F12 → printf "f_%sgr(%s,%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 p1 p2 p12 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 p1 p2 p12 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"
| coeff, -, -, - → invalid_arg
"Targets.Fortran-Majorana-Fermions: not used 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 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 "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_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,%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,%s)" f c f1 f2 f3
  | (F423 | F243 | F432 | F234 | F342 | F324) →
    printf "f_%sgr(%s,%s,%s,%s,%s)" f c f1 f2 f3
  | (F134 | F143 | F314) → printf "%s%s_fgr(%s,%s,%s,%s,%s)" f d1 c f1 f2 f3
  | (F124 | F142 | F214) → printf "%s%s_fgr(%s,%s,%s,%s,%s)" f d2 c f1 f2 f3
  | (F413 | F431 | F341) → printf "%s%s_grf(%s,%s,%s,%s,%s)" f d1 c f1 f2 f3
  | (F241 | F412 | F421) → printf "%s%s_grf(%s,%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"
| coeff, 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"
| coeff, _, _, 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"
| coeff, _, V2LR, _ → invalid_arg "Targets.print_current: not available"
let reverse_braket _ = false
let use_module = "omega95_bispinors"
let require_library =
  ["omega_bispinors.2009-06-A"; "omega_bispinor_cpls.2009-06-A"]

```

```

end
module Fortran_Majorana = Make_Fortran(Fortran_Majorana_Fermions)

```

FORTRAN 77

```

module Fortran77 = Dummy

```

15.2.2 *O'Mega Virtual Machine*

```

module VM = 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 *L^AT_EX*

```

module LaTeX = Dummy
  List.iter print_current (F.rhs fusion);
  let propagate code =
    printi { code = code; sign = 0; coupl = 0;
             lhs = int_of_string (format_p lhs);
             rhs1 = abs (M.pdg f); rhs2 = abs (M.pdg f) } in
  match M.propagator f with

```

```

| Prop_Scalar → propagate ovm_PROPAGATE_SCALAR
| Prop_Col_Scalar →
  failwith "print-fusion:␣Prop_Col_Scalar␣not␣implemented␣yet!"
| Prop_Ghost →
  failwith "print-fusion:␣Prop_Ghost␣not␣implemented␣yet!"
| Prop_Spinor → propagate ovm_PROPAGATE_SPINOR
| Prop_ConjSpinor → propagate ovm_PROPAGATE_CONJSPINOR
| Prop_Majorana | Prop_Col_Majorana →
  failwith "print-fusion:␣Prop_Majorana␣not␣implemented␣yet!"
| Prop_Unitarity → propagate ovm_PROPAGATE_UNITARITY
| Prop_Col_Unitarity →
  failwith "print-fusion:␣Prop_Col_Unitarity␣not␣implemented␣yet!"
| Prop_Feynman → propagate ovm_PROPAGATE_FEYNMAN
| Prop_Col_Feynman →
  failwith "print-fusion:␣Prop_Col_Feynman␣not␣implemented␣yet!"
| Prop_Gauge xi →
  failwith "print-fusion:␣Prop_Gauge␣not␣implemented␣yet!"
| Prop_Rxi xi →
  failwith "print-fusion:␣Prop_Rxi␣not␣implemented␣yet!"
| Prop_Vectorspinor →
  failwith "print-fusion:␣Prop_Vectorspinor␣not␣implemented␣yet!"
| Prop_Tensor_2 → propagate ovm_PROPAGATE_TENSOR2
| Aux_Scalar | Aux_Spinor | Aux_ConjSpinor | Aux_Majorana
| Aux_Vector | Aux_Tensor_1 → ()
| Only_Insertion → ()

module P = Set.Make (struct type t = int list let compare = compare end)

let rec add_momenta lhs = function
| [] | [-] → invalid_arg "add_momenta"
| [rhs1; rhs2] →
  printi { code = ovm_ADD_MOMENTA; sign = 0; coupl = 0;
    lhs = int_of_string (format_p lhs);
    rhs1 = int_of_string (format_p rhs1);
    rhs2 = int_of_string (format_p rhs2) }
| rhs1 :: rhs →
  add_momenta lhs rhs;
  add_momenta lhs [lhs; rhs1]

let print_fusions amplitude =
  printf "@\n@ [<2>BEGIN_FUSIONS";
  let momenta =
    List.fold_left (fun seen f →
      let wf = F.lhs f in
      let p = F.momentum_list wf in
      let momentum = format_p wf in
      if ¬ (P.mem p seen) then
        add_momenta wf (F.children (List.hd (F.rhs f)));
      print_fusion f;
      P.add p seen) P.empty (F.fusions amplitude)
  in
  printf "@]\n@<2>END_FUSIONS"

```

```

let print_brackets amplitude =
  printf "@\n@ [<2>BEGIN_BRACKETS";
  printf "@\n!!!_not_implemented_yet_!!!";
  printf "@]\nEND_BRACKETS"

let print_fudge_factor amplitude =
  printf "@\n@ [<2>BEGIN_FUDGE";
  printf "@\n!!!_not_implemented_yet_!!!";
  printf "@]\nEND_FUDGE"

let amplitude_to_channel oc diagnostics amplitude =
  set_formatter_out_channel oc;
  printf "@\n@ [<2>BEGIN_AMPLITUDE_%s" (format_process amplitude);
  print_externals amplitude;
  print_fusions amplitude;
  print_brackets amplitude;
  print_fudge_factor amplitude;
  printf "@]\nEND_AMPLITUDE"

let amplitudes_to_channel oc diagnostics amplitudes =
  List.iter (amplitude_to_channel oc diagnostics) (MF.allowed amplitudes)

let parameters_to_channel oc =
  set_formatter_out_channel oc;
  printf "@ [<2>BEGIN_PARAMETERS@\n";
  printf "!!!_not_implemented_yet_!!!@\n";
  printf "END_PARAMETERS@\n"

end

i × )

```

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 = "$Revision: 774$";
  RCS.date = "$Date: 2009-06-11 19:42:04 +0200 (Thu, 11 Jun 2009)$";
  RCS.author = "$Author: ohl$";
  RCS.source
    = "$URL: svn+ssh://jr-reuter@login.hepforge.org/hepforge/svn/whizard/trunk/src/omeg
module Fortran =
struct
  open Printf
  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 "%s\n" "!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!"; nl ();
  printf "%s\n" "Special_Kmatrix_functions"; nl ();
  printf "%s\n" "!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!"; nl ();
  nl ();
  printf "%s\n" "%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)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=3.*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 "%s\n" "end_function_width_res"; nl ();
  nl ();
  printf "%s\n" "%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*_s*(6*m**2+_s3*s)&"; nl();
printf "%%%%%%%%_m**4/s**3*_s*(6*m**4+_s6*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_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=_s1+_s*(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=_s*(m**2+_s2*s)/s**2*_s*(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 "endif"; nl ();
printf "endfunction_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)+6*(2*m**2+s)*(m**4+6*m**2*s+6*s**2))/6/s**2"; nl ();
printf "endif"; nl ();
printf "endfunction_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(1:5),intent(in):m,cc"; nl ();
printf "real(kind=default):a00_0,a00_1"; nl ();
printf "complex(kind=default),dimension(1:6):a00"; nl ();
printf "complex(kind=default):ii,jj,amp_00"; nl ();
printf "ii=cplx(0.0,1.0/32.0/Pi,default)"; nl ();
printf "jj=s**2/vev**4*ii"; nl ();
printf "!!!Scalar_isosinglet"; nl ();
printf "if(cc(1).ne.0)then"; nl ();
printf "if(fudge_km.ne.0)then"; nl ();
printf "a00(1)=vev**4/s**2*fudge_km"; nl ();
printf "cplx(0.0,32.0*Pi,default)*(1.0+"; nl ();
printf "(s-m(1)**2)/(ii*cc(1)**2/vev**2*(3.0*s**2+"; nl ();
printf "(s-m(1)**2)*2.0*s0stu(s,m(1))-(s-m(1)**2))"; nl ();
printf "else"; nl ();
printf "a00(1)=vev**2/s**2*cc(1)**2"; nl ();
printf "(3.0*s**2/cplx(s-m(1)**2,m(1)*width_res(w_res,1,&"; nl ();
printf "wkm(1),m(1),cc(1)))+2.0*s0stu(s,m(1))"; nl ();
printf "endif"; nl ();
printf "else"; nl ();
printf "a00(1)=0"; nl ();
printf "endif"; nl ();
printf "!!!Scalar_isoquintet"; nl ();
printf "a00(2)=5.0*cc(2)**2/vev**2*s0stu(s,m(2))/3.0"; nl ();
printf "a00(2)=vev**4/s**2*a00(2)"; nl ();
printf "(1.0-default-fudge_km*ii*a00(2))"; nl ();
printf "!!!Vector_isotriplet"; nl ();
printf "a00(3)=cc(3)**2*(4.0*p0stu(s,m(3))+3.0*s/m(3)**2)"; nl ();
printf "a00(3)=vev**4/s**2*a00(3)"; nl ();
printf "(1.0-default-fudge_km*ii*a00(3))"; nl ();

```



```

printf "!!!!_Tensor_isosinglet"; nl ();
printf "!!!!a00(4)=cc(4)**2/vev**2*_u(d0stu(s,m(4))_u&"; nl ();
printf "!!!!!!!!!!!!/3.0+_u11.0*s**2/m(4)**2/36.0"; nl ();
printf "!!!!a00(4)=vev**4/s**2*a00(4)/&"; nl ();
printf "!!!!!!!!!!!!(1.0-default-_ufudge_km*ii*a00(4))"; nl ();
printf "!!!!_Tensor_isoquintet"; nl ();
printf "!!!!a00(5)=5.0*cc(5)**2/vev**2*(d0stu(s,m(5))&"; nl ();
printf "!!!!!!!!!!!!/3.0+_us**2/m(5)**2/18.0)/6.0"; nl ();
printf "!!!!a00(5)=vev**4/s**2*a00(5)/&"; nl ();
printf "!!!!!!!!!!!!(1.0-default-_ufudge_km*ii*a00(5))"; nl ();
printf "!!!!_Low_energy_theory_alphas"; nl ();
printf "!!!!a00_0=_u2*fudge_higgs*vev**2/s+_u8*(7*a4+_u11*a5)/3"; nl ();
printf "!!!!a00_1=_u25*log(lam_reg**2/s)/9+_u11./54.0-default"; nl ();
printf "!!!!a00(6)=a00_0_!!!!+_ua00_1/16/Pi**2"; nl ();
printf "!!!!a00(6)=fudge_km*jj*a00(6)**2/_u(1.0-default-_ujj*a00(6))"; nl ();
printf "!!!!amp_00=_u sum(a00)"; nl ();
printf "!!end_function_da00"; nl ();
nl ();
printf "!!_sfunction_da02(cc,_us,_um)_result(amp_02)" pure; nl ();
printf "!!!!real(kind=default),_intent(in)_: _us"; nl ();
printf "!!!!real(kind=default),_dimension(5),_intent(in)_: _um,_ucc"; nl ();
printf "!!!!real(kind=default)_: _ua02_0,_ua02_1"; nl ();
printf "!!!!complex(kind=default),_dimension(1:6)_: _ua02"; nl ();
printf "!!!!complex(kind=default)_: _uii,_ujj,_uamp_02"; nl ();
printf "!!!!ii=_ucmplx(0.0,1.0/32.0/Pi,default)"; nl ();
printf "!!!!jj=_us**2/vev**4*ii"; nl ();
printf "!!!!_Scalar_isosinglet"; nl ();
printf "!!!!a02(1)=2.0*cc(1)**2/vev**2*_us2stu(s,m(1))"; nl ();
printf "!!!!a02(1)=vev**4/s**2*a02(1)/&"; nl ();
printf "!!!!!!!!!!!!(1.0-default-_ufudge_km*ii*a02(1))"; nl ();
printf "!!!!_Scalar_isoquintet"; nl ();
printf "!!!!a02(2)=5.0*cc(2)**2/vev**2*_us2stu(s,m(2))/_u3.0"; nl ();
printf "!!!!a02(2)=vev**4/s**2*a02(2)/&"; nl ();
printf "!!!!!!!!!!!!(1.0-default-_ufudge_km*ii*a02(2))"; 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 "!!!!a02(3)=vev**4/s**2*a02(3)/&"; nl ();
printf "!!!!!!!!!!!!(1.0-default-_ufudge_km*ii*a02(3))"; nl ();
printf "!!!!_Tensor_isosinglet"; nl ();
printf "!!!!if(cc(4).ne.0)_then"; nl ();
printf "!!!!if(fudge_km.ne.0)_then"; nl ();
printf "!!!!a02(4)=vev**4/s**2*_ufudge_km*_u&"; nl ();
printf "!!!!!!!!!!!!cmplx(0.0,32.0*Pi,default)*(1.0+_u&"; nl ();
printf "!!!!!!!!!!!!(s-m(4)**2)/(ii*cc(4)**2/vev**2*(s**2/10.0+_u&"; nl ();
printf "!!!!!!!!!!!!(s-m(4)**2)*((1.0+6.0*s/m(4)**2+6.0*_u&"; nl ();
printf "!!!!!!!!!!!!s**2/m(4)**4)*_us2stu(s,m(4))/3.0&"; nl ();
printf "!!!!!!!!!!!!+_us**2/m(4)**2/180.0))/_u(s-m(4)**2))"; nl ();
printf "!!!!else"; nl ();
printf "!!!!a02(4)=vev**2/s**2*_ucc(4)**2*_u(s**2/_u&"; nl ();
printf "!!!!!!!!!!!!cmplx(s-m(4)**2,m(4)*width_res(w_res,4,wkm(4),&"; nl ();

```

```

printf "aaaaaaaaaaaaaaaaa m(4),cc(4))/10.0_&"; nl ();
printf "aaaaaaaaaaaaaaaaa (1.+6.*s/m(4)**2+6.*s**2/m(4)**4)*s2stu(s,m(4))/_&"; nl ();
printf "aaaaaaaaaaaaaaaaa 3._+s**2/m(4)**2/180.)"; nl ();
printf "aaaaaaa end_if"; nl ();
printf "aaaaaaa else"; nl ();
printf "aaaaaaa a02(4)=_0"; nl ();
printf "aaaaaaa end_if"; nl ();
printf "aaaaaaa !!!Tensor_isoquintet"; nl ();
printf "aaaaaaa a02(5)=_cc(5)**2/vev**2*(5.0*(1.0+6.0*_&"; nl ();
printf "aaaaaaaaaaaaaaaaa s/m(5)**2+6.0*s**2/m(5)**4)*s2stu(s,m(5))/3.0_&"; nl ();
printf "aaaaaaaaaaaaaaaaa +s**2/m(5)**2/216.0)/6.0"; nl ();
printf "aaaaaaa a02(5)=_vev**4/s**2*a02(5)/_&"; nl ();
printf "aaaaaaaaaaaaaaaaa (1.0_default+_fudge_km*ii*a02(5))"; nl ();
printf "aaaaaaa !!!Low_energy_theory_alphas"; nl ();
printf "aaaaaaa a02_0=_8*(2*a4+_a5)/15"; nl ();
printf "aaaaaaa a02_1=_log(lam_reg**2/s)/9-_7./135.0_default"; nl ();
printf "aaaaaaa a02(6)=_a02_0_!!!+_a02_1/16/Pi**2"; nl ();
printf "aaaaaaa a02(6)=_fudge_km*jj*a02(6)**2/_ (1.0_default+_jj*a02(6))"; nl ();
printf "aaaaaaa amp_02=_sum(a02)"; nl ();
printf "aa end_function_da02"; nl ();
nl ();
printf "aa %sfunction_da11_(cc,s,m)_result_(amp_11)" pure; nl ();
printf "aaaaaaa real(kind=default),_intent(in)_:_s"; nl ();
printf "aaaaaaa real(kind=default),_dimension(5),_intent(in)_:_m,_cc"; nl ();
printf "aaaaaaa real(kind=default)_:_a11_0,_a11_1"; nl ();
printf "aaaaaaa complex(kind=default),_dimension(1:6)_:_a11"; nl ();
printf "aaaaaaa complex(kind=default)_:_ii,_jj,_amp_11"; nl ();
printf "aaaaaaa ii=_cmplx(0.0,1.0/32.0/Pi,default)"; nl ();
printf "aaaaaaa jj=_s**2/vev**4*ii"; nl ();
printf "aaaaaaa !!!Scalar_isosinglet"; nl ();
printf "aaaaaaa a11(1)=_2.0*cc(1)**2/vev**2*_s1stu(s,m(1))"; nl ();
printf "aaaaaaa a11(1)=_vev**4/s**2*a11(1)/_&"; nl ();
printf "aaaaaaaaaaaaaaaaa (1.0_default+_fudge_km*ii*a11(1))"; nl ();
printf "aaaaaaa !!!Scalar_isoquintet"; nl ();
printf "aaaaaaa a11(2)=_5.0*cc(2)**2/vev**2*_s1stu(s,m(2))/_6.0"; nl ();
printf "aaaaaaa a11(2)=_vev**4/s**2*a11(2)/_&"; nl ();
printf "aaaaaaaaaaaaaaaaa (1.0_default+_fudge_km*ii*a11(2))"; nl ();
printf "aaaaaaa !!!Vector_isotriplet"; nl ();
printf "aaaaaaa if(cc(3).ne.0)_then"; nl ();
printf "aaaaaaa if(_fudge_km.ne.0)_then"; nl ();
printf "aaaaaaa a11(3)=_vev**4/s**2*_fudge_km*_&"; nl ();
printf "aaaaaaaaaaaaaaaaa cmplx(0.0,32.0*Pi,default)*(1.0+_ (s-m(3)**2)_&"; nl ();
printf "aaaaaaaaaaaaaaaaa /(ii*cc(3)**2*(2.0*s/3.0+_ (s-m(3)**2)&"; nl ();
printf "aaaaaaaaaaaaaaaaa *(s/m(3)**2+2.0*p1stu(s,m(3))))_-(s-m(3)**2))"; nl ();
printf "aaaaaaa else"; nl ();
printf "aaaaaaa a11(3)=_vev**4/s**2*_cc(3)**2*_ (2.*s/_&"; nl ();
printf "aaaaaaaaaaaaaaaaa cmplx(s-m(3)**2,m(3)*width_res(w_res,3,wkm(3),m(3),_&"; nl ();
printf "aaaaaaaaaaaaaaaaa cc(3))/3._+s/m(3)**2+_2.*p1stu(s,m(3))"; nl ();
printf "aaaaaaa end_if"; nl ();
printf "aaaaaaa else"; nl ();

```

```

printf "aaaaaaaaa11(3)=0"; nl ();
printf "aaaaaaend_if"; nl ();
printf "aaaaaa!!!Tensor_isosinglet"; nl ();
printf "aaaaaa11(4)=cc(4)**2/vev**2*(d1stu(s,m(4)))&"; nl ();
printf "aaaaaaaaaaaaaaaa/3.0-s**2/m(4)**2/36.0"; nl ();
printf "aaaaaa11(4)=vev**4/s**2*a11(4)/&"; nl ();
printf "aaaaaaaaaaaaaaaa(1.0-default-fudge_km*ii*a11(4))"; nl ();
printf "aaaaaa!!!Tensor_isoquintet"; nl ();
printf "aaaaaa11(5)=5.0*cc(5)**2/vev**2*(-d1stu(s,m(5)))&"; nl ();
printf "aaaaaaaaaaaaaaaa+s**2/m(5)**2/12.0)/36.0"; nl ();
printf "aaaaaa11(5)=vev**4/s**2*a11(5)/&"; nl ();
printf "aaaaaaaaaaaaaaaa(1.0-default-fudge_km*ii*a11(5))"; nl ();
printf "aaaaaa!!!Low_energy_theory_alphas"; nl ();
printf "aaaaaa11_0=fudge_higgs*vev**2/3/s+4*(a4-2*a5)/3"; nl ();
printf "aaaaaa11_1=-1.0/54.0-default"; nl ();
printf "aaaaaa11(6)=a11_0!!!+a11_1/16/Pi**2"; nl ();
printf "aaaaaa11(6)=fudge_km*jj*a11(6)**2/(1.0-default-ujj*a11(6))"; nl ();
printf "aaaaaaamp_11=sum(a11)"; nl ();
printf "aaend_function_da11"; nl ();
nl ();
printf "%%sfunction_da20(cc,s,m)result(amp_20)" pure; nl ();
printf "aaaaaa real(kind=default),intent(in):s"; nl ();
printf "aaaaaa real(kind=default),dimension(1:5),intent(in):m,cc"; nl ();
printf "aaaaaa real(kind=default):a20_0,a20_1"; nl ();
printf "aaaaaa complex(kind=default),dimension(1:6):a20"; nl ();
printf "aaaaaa complex(kind=default):ii,jj,amp_20"; nl ();
printf "aaaaaa ii=cplx(0.0,1.0/32.0/Pi,default)"; nl ();
printf "aaaaaa jj=s**2/vev**4*ii"; nl ();
printf "aaaaaa!!!Scalar_isosinglet"; nl ();
printf "aaaaaa a20(1)=2.0*cc(1)**2/vev**2*s0stu(s,m(1))"; nl ();
printf "aaaaaa a20(1)=vev**4/s**2*a20(1)/&"; nl ();
printf "aaaaaaaaaaaaaaaa(1.0-default-fudge_km*ii*a20(1))"; nl ();
printf "aaaaaa!!!Scalar_isoquintet"; nl ();
printf "aaaaaa if(cc(2).ne.0)then"; nl ();
printf "aaaaaa if(fudge_km.ne.0)then"; nl ();
printf "aaaaaa a20(2)=vev**4/s**2*fudge_km*&"; nl ();
printf "aaaaaaaaaaaaaaaa cplx(0.0,32.0*Pi,default)*(1.0+&"; nl ();
printf "aaaaaaaaaaaaaaaa (s-m(2)**2)/(ii*cc(2)**2/vev**2*(s**2/2.0+&"; nl ();
printf "aaaaaaaaaaaaaaaa (s-m(2)**2)*s0stu(s,m(2))/6.0)-(s-m(2)**2))"; nl ();
printf "aaaaaa else"; nl ();
printf "aaaaaa a20(2)=vev**2/s**2*cc(2)**2*(s**2/&"; nl ();
printf "aaaaaaaaaaaaaaaa cplx(s-m(2)**2,m(2)*width_res(w_res,2,wkm(2),&"; nl ();
printf "aaaaaaaaaaaaaaaa m(2),cc(2))/2.+s0stu(s,m(2))/6.)"; nl ();
printf "aaaaaa end_if"; nl ();
printf "aaaaaa else"; nl ();
printf "aaaaaa a20(2)=0"; nl ();
printf "aaaaaa end_if"; nl ();
printf "aaaaaa!!!Vector_isotriplet"; nl ();
printf "aaaaaa a20(3)=cc(3)**2*(2.0*p0stu(s,m(3))+3.0*s/m(3)**2)"; nl ();
printf "aaaaaa a20(3)=vev**4/s**2*a20(3)/&"; nl ();

```

```

printf "aaaaaaaaaaaaaaaaaaaa(1.0-default-ufudge_km*ii*a20(3)); nl ();
printf "aaaaaaa!!!_Tensor_isosinglet"; nl ();
printf "aaaaaaa_a20(4)=_cc(4)**2/vev**2*(d1stu(s,m(4))_&"; nl ();
printf "aaaaaaa/3.0+_s**2/m(4)**2/18.0"; nl ();
printf "aaaaaaa_a20(4)=_vev**4/s**2*a20(4)/&"; nl ();
printf "aaaaaaaaaaaaaaaaaaaa(1.0-default-ufudge_km*ii*a20(4)); nl ();
printf "aaaaaaa!!!_Tensor_isoquintet"; nl ();
printf "aaaaaaa_a20(5)=_cc(5)**2/vev**2*(d0stu(s,m(5))_&"; nl ();
printf "aaaaaaa+_5.0*s**2/m(4)**2/3.0)/36.0"; nl ();
printf "aaaaaaa_a20(5)=_vev**4/s**2*a20(5)/&"; nl ();
printf "aaaaaaaaaaaaaaaaaaaa(1.0-default-ufudge_km*ii*a20(5)); nl ();
printf "aaaaaaa!!!_Low_energy_theory_alphas"; nl ();
printf "aaaaaaa_a20_0=_-fudge_higgs*vev**2/s+_16*(2*a4+_a5)/3"; nl ();
printf "aaaaaaa_a20_1=_10*log(lam_reg**2/s)/9+_25/108.0-default"; nl ();
printf "aaaaaaa_a20(6)=_a20_0!!!+_a20_1/16/Pi**2"; nl ();
printf "aaaaaaa_a20(6)=_fudge_km*jj*a20(6)**2/_(1.0-default-_jj*a20(6)); nl ();
printf "aaaaaaa_amp_20=_sum(a20); nl ();
printf "end_function_da20"; nl ();
nl ();
printf "%sfunction_da22(cc,s,m)_result(amp_22)" pure; nl ();
printf "aaaaaaa_real(kind=default),_intent(in)_:_s"; nl ();
printf "aaaaaaa_real(kind=default),_dimension(1:5),_intent(in)_:_m,_cc"; nl ();
printf "aaaaaaa_real(kind=default)_:_a22_0,_a22_1"; nl ();
printf "aaaaaaa_complex(kind=default),_dimension(1:6)_:_a22"; nl ();
printf "aaaaaaa_complex(kind=default)_:_ii,_jj,_amp_22"; nl ();
printf "aaaaaaa_ii=_cplx(0.0,1.0/32.0/Pi,default); nl ();
printf "aaaaaaa_jj=_s**2/vev**4*ii"; nl ();
printf "aaaaaaa!!!_Scalar_isosinglet"; nl ();
printf "aaaaaaa_a22(1)=_2.0*cc(1)**2/vev**2*_s2stu(s,m(1)); nl ();
printf "aaaaaaa_a22(1)=_vev**4/s**2*a22(1)/&"; nl ();
printf "aaaaaaaaaaaaaaaaaaaa(1.0-default-ufudge_km*ii*a22(1)); nl ();
printf "aaaaaaa!!!_Scalar_isoquintet"; nl ();
printf "aaaaaaa_a22(2)=_cc(2)**2/vev**2*_s2stu(s,m(2))/_6.0"; nl ();
printf "aaaaaaa_a22(2)=_vev**4/s**2*a22(2)/&"; nl ();
printf "aaaaaaaaaaaaaaaaaaaa(1.0-default-ufudge_km*ii*a22(2)); nl ();
printf "aaaaaaa!!!_Vector_triplet"; nl ();
printf "aaaaaaa_a22(3)=_2.0*cc(3)**2*(2*s+m(3)**2)*s2stu(s,m(3))/m(3)**4"; nl ();
printf "aaaaaaa_a22(3)=_vev**4/s**2*a22(3)/&"; nl ();
printf "aaaaaaaaaaaaaaaaaaaa(1.0-default-ufudge_km*ii*a22(3)); nl ();
printf "aaaaaaa!!!_Tensor_isosinglet"; nl ();
printf "aaaaaaa_a22(4)=_cc(4)**2/vev**2*((1.0+_6.0*s/m(4)**2+6.0*_&"; nl ();
printf "aaaaaaa_s**2/m(4)**4)*s2stu(s,m(4))/3.0+_s**2/m(4)**2/180.0"; nl ();
printf "aaaaaaa_a22(4)=_vev**4/s**2*a22(4)/&"; nl ();
printf "aaaaaaaaaaaaaaaaaaaa(1.0-default-ufudge_km*ii*a22(4)); nl ();
printf "aaaaaaa!!!_Tensor_isoquintet"; nl ();
printf "aaaaaaa_if(cc(5).ne.0)_then"; nl ();
printf "aaaaaaa_if(fudge_km.ne.0)_then"; nl ();
printf "aaaaaaa_a22(5)=_vev**4/_s**2*_&"; nl ();
printf "aaaaaaaaaaaaaaaaaaaa_cplx(0.0,32.0*Pi,default)*(1.0+_&"; nl ();
printf "aaaaaaaaaaaaaaaaaaaa(s-m(5)**2)/(ii*cc(5)**2/vev**2*(s**2/60.0+_&"; nl ();

```

```

printf "aaaaaaaaaaaaaaaa(s-m(5)**2)*((1.0+6.0*s/m(5)**2+6.0*_&"; nl ();
printf "aaaaaaaaaaaaaaaas**2/m(5)**4)*s2stu(s,m(5))/36.0*&"; nl ();
printf "aaaaaaaaaaaaaaaa+_s**2/m(5)**2/2160.0))_-(s-m(5)**2))"; nl ();
printf "aaaaaaaaelse"; nl ();
printf "aaaaaaaaa22(5)=_vev**2/s**2*_cc(5)**2*_(_s**2/_&"; nl ();
printf "aaaaaaaaaaaaaaaacmplx(s-m(5)**2,m(5)*width_res(w_res,5,wkm(5),&"; nl ();
printf "aaaaaaaaaaaaaaaam(5),cc(5)))/80._+(1.0+6.0*_&"; nl ();
printf "aaaaaaaaaaaaaaaas/m(5)**2+6.0*s**2/m(5)**4)*s2stu(s,m(5))/36.0+_&"; nl ();
printf "aaaaaaaaaaaaaaaas**2/m(5)**2/2160.0)"; nl ();
printf "aaaaaaaaend_if"; nl ();
printf "aaaaaaaaelse"; nl ();
printf "aaaaaaaaa22(5)=_0"; nl ();
printf "aaaaaaaaend_if"; nl ();
printf "aaaaaaa!!!_Low_energy_theory_alphas"; nl ();
printf "aaaaaaa_a22_0=_4*(a4+_2*a5)/15"; nl ();
printf "aaaaaaa_a22_1=_2*log(lam_reg**2/s)/45-_247/5400.0_default"; nl ();
printf "aaaaaaa_a22(6)=_a22_0_!!!+_a22_1/16/Pi**2"; nl ();
printf "aaaaaaa_a22(6)=_fudge_km*jj*a22(6)**2/_+(1.0_default-_jj*a22(6))"; nl ();
printf "aaaaaaa_amp_22=_sum(a22)"; nl ();
printf "aaend_function_da22"; nl ();
nl ();
printf "aa%sfunction_dalzz0_s_(cc,m,k)_result_(alzz0_s)" pure; nl ();
printf "aaaaaaa_type(momentum),_intent(in):_k"; nl ();
printf "aaaaaaa_real(kind=default),_dimension(1:5),_intent(in):_cc,_m"; nl ();
printf "aaaaaaa_complex(kind=default):_alzz0_s"; nl ();
printf "aaaaaaa_real(kind=default):_s"; nl ();
printf "aaaaaaa_s=_k*k"; nl ();
printf "aaaaaaa_alzz0_s=_2*g**4/costhw**2*((da00(cc,s,m)_&"; nl ();
printf "aaaaaaa_da20(cc,s,m))/24*&"; nl ();
printf "aaaaaaa_5*(da02(cc,s,m)-_da22(cc,s,m))/12)"; nl ();
printf "aaend_function_dalzz0_s"; nl ();
nl ();
printf "aa%sfunction_dalzz0_t_(cc,m,k)_result_(alzz0_t)" pure; nl ();
printf "aaaaaaa_type(momentum),_intent(in):_k"; nl ();
printf "aaaaaaa_real(kind=default),_dimension(1:5),_intent(in):_cc,_m"; nl ();
printf "aaaaaaa_complex(kind=default):_alzz0_t"; nl ();
printf "aaaaaaa_real(kind=default):_s"; nl ();
printf "aaaaaaa_s=_k*k"; nl ();
printf "aaaaaaa_alzz0_t=_5*g**4/costhw**2*(da02(cc,s,m)-_&"; nl ();
printf "aaaaaaa_da22(cc,s,m))/4"; nl ();
printf "aaend_function_dalzz0_t"; nl ();
nl ();
printf "aa%sfunction_dalzz1_s_(cc,m,k)_result_(alzz1_s)" pure; nl ();
printf "aaaaaaa_type(momentum),_intent(in):_k"; nl ();
printf "aaaaaaa_real(kind=default),_dimension(1:5),_intent(in):_cc,_m"; nl ();
printf "aaaaaaa_complex(kind=default):_alzz1_s"; nl ();
printf "aaaaaaa_real(kind=default):_s"; nl ();
printf "aaaaaaa_s=_k*k"; nl ();
printf "aaaaaaa_alzz1_s=_g**4/costhw**2*(da20(cc,s,m)/8*&"; nl ();
printf "aaaaaaa_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(1:5),intent(in)::cc,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 "%%%%%%%%+15*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(1:5),intent(in)::cc,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 "%%%%%%%%+15*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(1:5),intent(in)::cc,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(1:5),intent(in)::cc,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*(10*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(1:5),intent(in)::cc,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*(10*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(1:5),_intent(in)_::_cc,_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)-_10*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(1:5),_intent(in)_::_cc,_m"; nl ();
printf "%%%%%%%%complex(kind=default)_::_alww2_t"; nl ();
printf "%%%%%%%%real(kind=default)_::_s"; nl ();
printf "%%%%%%%%s=_k*k"; nl ();
printf "%%%%%%%%alww2_t=_15*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(1:5),_intent(in)_::_cc,_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 "%%sfunction_dalz4_t(cc,m,k)_result_(alz4_t)" pure; nl ();
printf "%%%%%%%%type(momentum),_intent(in)_::_k"; nl ();
printf "%%%%%%%%real(kind=default),_dimension(1:5),_intent(in)_::_cc,_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  $\alpha$  t
    type  $\alpha$  decay
  end

```

Sort individual decays and complete phasespaces in a canonical order to determine topological equivalence classes.

```

val sort : ( $\alpha \rightarrow \alpha \rightarrow \text{int}$ )  $\rightarrow \alpha$  t  $\rightarrow \alpha$  t
val sort_decay : ( $\alpha \rightarrow \alpha \rightarrow \text{int}$ )  $\rightarrow \alpha$  decay  $\rightarrow \alpha$  decay

```

Functionals:

```

val map : ( $\alpha \rightarrow \beta$ )  $\rightarrow \alpha$  t  $\rightarrow \beta$  t
val map_decay : ( $\alpha \rightarrow \beta$ )  $\rightarrow \alpha$  decay  $\rightarrow \beta$  decay

val eval : ( $\alpha \rightarrow \beta$ )  $\rightarrow (\alpha \rightarrow \beta) \rightarrow (\alpha \rightarrow \beta \rightarrow \beta \rightarrow \beta) \rightarrow \alpha$  t  $\rightarrow \beta$  t
val eval_decay : ( $\alpha \rightarrow \beta$ )  $\rightarrow (\alpha \rightarrow \beta \rightarrow \beta \rightarrow \beta) \rightarrow \alpha$  decay  $\rightarrow \beta$  decay

```

of_momenta *f1* *f2* *plist* constructs the phasespace parameterization for a process $f_1 f_2 \rightarrow X$ with flavor decoration from pairs of outgoing momenta and flavors *plist* and initial flavors *f1* and *f2*

```

val of_momenta :  $\alpha \rightarrow \alpha \rightarrow (\text{momentum} \times \alpha)$  list  $\rightarrow (\text{momentum} \times \alpha)$  t
val decay_of_momenta : ( $\text{momentum} \times \alpha$ ) list  $\rightarrow (\text{momentum} \times \alpha)$  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  $\alpha$  t
  type  $\alpha$  decay
  val sort : ( $\alpha \rightarrow \alpha \rightarrow \text{int}$ )  $\rightarrow \alpha$  t  $\rightarrow \alpha$  t
  val sort_decay : ( $\alpha \rightarrow \alpha \rightarrow \text{int}$ )  $\rightarrow \alpha$  decay  $\rightarrow \alpha$  decay
  val map : ( $\alpha \rightarrow \beta$ )  $\rightarrow \alpha$  t  $\rightarrow \beta$  t
  val map_decay : ( $\alpha \rightarrow \beta$ )  $\rightarrow \alpha$  decay  $\rightarrow \beta$  decay
  val eval : ( $\alpha \rightarrow \beta$ )  $\rightarrow (\alpha \rightarrow \beta) \rightarrow (\alpha \rightarrow \beta \rightarrow \beta \rightarrow \beta) \rightarrow \alpha$  t  $\rightarrow \beta$  t
  val eval_decay : ( $\alpha \rightarrow \beta$ )  $\rightarrow (\alpha \rightarrow \beta \rightarrow \beta \rightarrow \beta) \rightarrow \alpha$  decay  $\rightarrow \beta$  decay
  val of_momenta :  $\alpha \rightarrow \alpha \rightarrow (\text{momentum} \times \alpha)$  list  $\rightarrow (\text{momentum} \times \alpha)$  t
  val decay_of_momenta : ( $\text{momentum} \times \alpha$ ) list  $\rightarrow (\text{momentum} \times \alpha)$  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  $\alpha$  decay =
| Leaf of  $\alpha$ 
| Branch of  $\alpha \times \alpha$  decay  $\times \alpha$  decay

```



Trees of type $(\text{momentum} \times \alpha \text{ option}) \text{ decay}$ can be build easily and mapped to $(\text{momentum} \times \alpha) \text{ decay}$ later, once all the α slots are filled. A more elegant functor operating on $\beta \text{ decay}$ directly (with *Momentum* style functions defined for β) would not allow holes in the $\beta \text{ decay}$ 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 $p > p_D \vee p = p_D \vee p < p_D$, 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

```

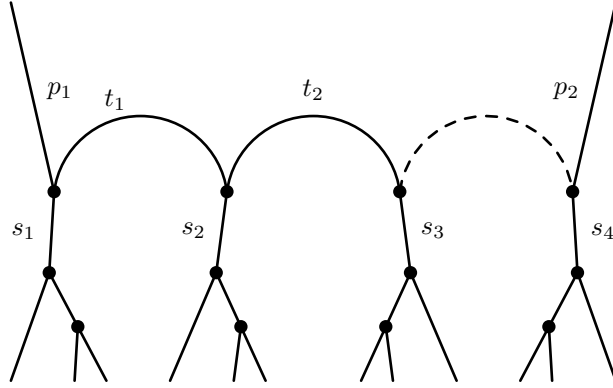


Figure 16.1: Phasespace parameterization for $2 \rightarrow n$ scattering by a sequence of cascade decays.

```
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"
```

$2 \rightarrow n$ Scattering

A general $2 \rightarrow n$ 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  $\alpha$  t = ( $\alpha \times \alpha$  decay  $\times \alpha$ ) list
```



let *topology* = *map snd* has type $(\text{momentum} \times \alpha) t \rightarrow \alpha t$ 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
```

¹Not to be confused with gauge invariant classes of Feynman diagrams [12].

```

if M.lesseq p' p  $\vee$  M.less p p' then
  embed_in_decay pf d' :: rest
else
  d' :: embed_in_decays pf rest

```

Collecting Ingredients

```

type  $\alpha$  unfinished_decays =
  { n : int;
    t_channel : (momentum  $\times$   $\alpha$  option) list;
    decays : (momentum  $\times$   $\alpha$  option) decay list }

let empty n = { n = n; t_channel = []; decays = [] }

let insert_in_unfinished_decays (p, f as pf) d =
  if M.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 ( $\lambda (p', f') \rightarrow (M.flip\_s\_channel\_in\ p', f')$ ) 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 _  $\rightarrow$  true
  | None  $\rightarrow$  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  $\rightarrow$  (p, acc, tlist, dlist)
  | Some ((t, - as td), (tlist', dlist'))  $\rightarrow$ 
    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" ["WhizardInterface"]
{ RCS.revision = "$Revision: 759$";
  RCS.date = "$Date: 2009-06-10 11:38:07 +0200 (Wed, 10 Jun 2009)$";
  RCS.author = "$Author: ohl$";
  RCS.source
    = "$URL: svn+ssh://jr-reuter@login.hepforge.org/hepforge/svn/whizard/trunk/src/omeg";
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 × M.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 : M.flavor;
      in2 : M.flavor;
      out : M.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.timelike c ∧ ¬ (P.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.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.flip_s_channel_in (F.momentum wf), [F.flavor wf])) poles in

```



```

    add_tree (count_maps n tree) tree trees
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 = M.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" | "d̄" → "D"
  | "u" → "u" | "ū" → "U"
  | "s" → "s" | "s̄" → "S"
  | "c" → "c" | "c̄" → "C"
  | "b" → "b" | "b̄" → "B"
  | "t" → "t" | "t̄" → "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.timelike c then begin
      if P.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 =
  fprintf ch "!_whizard_trees_by_0'Mega\n\n";
  fprintf ch "!_s_s->s\n"
    (M.flavor_to_string t.in1) (M.flavor_to_string t.in2)
    (String.concat "_" (List.map M.flavor_to_string t.out));
  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!_0'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"
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_prefix 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_prefix 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_prefix 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_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_tot\n"

let write_other_declarations ch =
fprintf ch "public::n_in,n_out,n_tot,updg_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:\omega_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*


```
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
```

 *max_lines* = 8 is plenty, since amplitudes with 8 gluons still take several *days* to construct.

```
  module CM = Colorize.It(struct let max_lines = Config.max_color_lines end)(M')
  module M = CM.M
  type flavor = M.flavor
```

 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)(CM)
  module CF = Fusion.Colored(Fusion_Maker)(P)(CM)
  module T = Target_Maker(Fusion_Maker)(P)(CM)
  module W = Whizard.Make(Fusion_Maker)(P)(P_Whizard)(CM)
  module C = Cascade.Make(CM)(P)

  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 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 = CM.flavor_to_string (F.flavor wf) ^ "[" ^ format_p wf ^ "]"
  let variable' wf = CM.flavor_symbol (F.flavor wf) ^ "[" ^ format_p wf ^ "]"
```

18.3.1 Parsing Process Descriptions

```

type  $\alpha$  bag =  $\alpha$  list
type decay = flavor bag  $\times$  flavor bag list
type scattering = flavor bag  $\times$  flavor bag  $\times$  flavor bag list
type process =
  | Any of flavor bag list
  | Decay of decay
  | Scattering of scattering

```

parse_process decodes process descriptions

$$\text{"a b c d"} \Rightarrow \text{Any } [a; b; c; d] \quad (18.1a)$$

$$\text{"a -> b c d"} \Rightarrow \text{Decay } (a, [b; c; d]) \quad (18.1b)$$

$$\text{"a b -> c d"} \Rightarrow \text{Scattering } (a, b, [c; d]) \quad (18.1c)$$

where each word is split into a bag of flavors separated by ‘:’s.

```

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  $\rightarrow$  Any (List.rev flavors :: l)
    | Decay (i, f)  $\rightarrow$  Decay (i, List.rev flavors :: f)
    | Scattering (i1, i2, f)  $\rightarrow$  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'  $\rightarrow$  scan_list so_far n'
      | '-'  $\rightarrow$  scan_gtr so_far n'
      | c  $\rightarrow$  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'  $\rightarrow$ 
        scan_list (add_flavors (flavor w (n - w) :: flavors) so_far) n'
      | ':'  $\rightarrow$  scan_flavors so_far (flavor w (n - w) :: flavors) n' n'
      | _  $\rightarrow$  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)

```

Force interpretation as decay and punt on an explicit scattering "a b -> c d".

```

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"

```

Force interpretation as scattering and punt on an explicit decay "a -> b c".

```

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.list
    (function
      | fin1' :: fin2' :: fout' → ([fin1'; fin2'], fout')
      | [-] | [] → failwith "Omega.expand_scatterings:_can't_happen")
    (fin1 :: fin2 :: fout)) scatterings

let expand_decays decays =
  ThoList.flatmap
  (function (fin, fout) →
    Product.list
    (function
      | fin' :: fout' → ([fin'], fout')
      | [] → failwith "Omega.expand_decays:_can't_happen")
    (fin :: fout)) decays

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

```

18.3.2 Main Program

```

let main () =
  let usage =
    "usage: " ^ Sys.argv.(0) ^
    "[options] " ^ String.concat " | " (List.map M.flavor_to_string (M.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 feynmf = ref None
  and feynmf_tex = 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
      [ "", "arguments", T.All, checks;
        "a", "#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),
  "write_to_given_file_instead_of_dev/stdout");
  ("-scatter", Arg.String (fun s → rev_scatterings := s :: !rev_scatterings),
  "in1_in2->out1_out2...");
  ("-scatter_file",
  Arg.String (fun s → rev_scatterings := read_lines_rev s @ !rev_scatterings),
  "in1_in2->out1_out2...");
  ("-decay", Arg.String (fun s → rev_decays := s :: !rev_decays),
  "in->out1_out2...");
  ("-decay_file", Arg.String (fun s → rev_decays := read_lines_rev s @ !rev_decays),
  "in->out1_out2...");
  ("-cascade", Arg.String (fun s → cascades := s :: !cascades),
  "select_diagrams");
  ("-initialize", Arg.String (fun s → cache_option := Cache_Initialize s),
  "precompute_large_lookup_table(s) and store them in the directory");
  ("-template", Arg.Set template,
  "write_a_template_for_using_handwritten_amplitudes_with_WHIZARD");
  ("-forest", Arg.Set print_forest, "Diagrammatic_expansion");
  ("-feynmf", Arg.String (fun s → feynmf := Some s), "print_feynmf/mp_output");
  ("-feynmf_tex", Arg.Set feynmf_tex, "print_feynmf/mp/LaTeX_output");
  ("-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;
begin match !cache_option with
| Cache_Initialize dir → CF.initialize_cache dir
| _ → ()
end;
let output_channel =
  match !output_file with
  | None → stdout
  | Some name → open_out name in
let processes =
  ThoList.uniq
  (List.sort compare
    (match List.rev !rev_scatterings, List.rev !rev_decays with
    | [], [] → []
    | scatterings, [] → expand_scatterings (List.map parse_scattering scatterings)
    | [], decays → expand_decays (List.map parse_decay decays)
    | scatterings, decays → invalid_arg "mixed_scattering_and_decay!")) in
begin match processes with
| [] → exit 0
| _ → ()
end;

```

```

let selectors =
  let fin, fout = List.hd processes in
  C.to_selectors (C.of_string_list (List.length fin + List.length fout) !cascades) in
if !params then
  T.parameters_to_channel output_channel
else
  let amplitudes = CF.amplitudes (include_goldstones !checks) selectors processes in
  begin match CF.flavors amplitudes with
  | [] → failwith "No allowed processes"
  | _ → ()
  end;
  if !write then
    T.amplitudes_to_channel
      (String.concat " " (List.map ThoString.quote (Array.to_list Sys.argv)))
    output_channel !checks amplitudes;
  if ¬ !quiet then begin
    List.iter
      (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
        (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 (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 (List.iter (F.amplitude_to_dot ch)) (CF.processes amplitudes);
        close_out ch
      | None → ()
      end;
      end;
      if !print_forest then
        List.iter
          (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 !output_file with
| None → ()
| Some name → close_out output_channel
end;
exit 0

```



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 =
  let a = F.amplitude false C.no_cascades [in1; in2] out in
  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_QCD*

```

let rcs_file = RCS.parse "F90_QCD" ["QCD"]
{ RCS.revision = "$Revision: 1340$";
  RCS.date = "$Date: 2009-12-03 00:45:04 +0100 (Thu, 03 Dec 2009)$";
  RCS.author = "$Author: ohl$";
  RCS.source
    = "$Source: /home/sources/ohl/ml/omega/src/omega_QCD.ml,v$" }

```

QCD with colors.

```

module M : Model.T =
struct
  let rcs = rcs_file

```

```

open Coupling

let options = Options.empty

type flavor =
  | U | Ubar | D | Dbar
  | C | Cbar | S | Sbar
  | T | Tbar | B | Bbar
  | Gl

type flavor_sans_color = flavor
let flavor_sans_color f = f

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

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 conjugate_sans_color = conjugate

let fermion = function
  | U | D | C | S | T | B → 1
  | Ubar | Dbar | Cbar | Sbar | Tbar | Bbar → -1
  | Gl → 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, 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 "Models.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 flavor_sans_color_of_string = flavor_of_string
let flavor_sans_color_to_string = flavor_to_string
let flavor_sans_color_to_TeX = flavor_to_TeX
let flavor_sans_color_symbol = flavor_symbol

let gauge_symbol () =
  failwith "Models.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

module O = Omega.Make(Fusion.Mixed23)(Targets.Fortran)(M)
let _ = O.main ()

```

18.6 Implementation of *Omega_SM*

```

module O = Omega.Make(Fusion.Mixed23)(Targets.Fortran)
  (Modellib_SM.SM(Modellib_SM.SM_no_anomalous))
let _ = O.main ()

```

—19—

O’GIGA: O’MEGA GRAPHICAL INTERFACE FOR GENERATION AND ANALYSIS

NB: The code in this chapter must be compiled with `-labels`, since `lablgtk` doesn’t appear to work in classic mode.



Keep in mind that `ocamlweb` doesn’t work properly with O’Caml 3 yet. The colons in label declarations are typeset with erroneous white space.



Application `ogiga.ml` unavailable!

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 → string list → raw → 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 → string → string list → t`

Access individual parts of the revision control information:

`val name : t → string`

`val description : t → string list`

`val revision : t → string`

`val date : t → string`

`val author : t → string`

This one tries **URL** (svn), **Source** (CVS) and **Id**, in that order, for the filename.

`val source : t → 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 → 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

let parse_source s =
  let s = strip_dollars s in

```

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.


```

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—

TEXTUAL OPTIONS

B.1 Interface of Options

```
type t
val empty : 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
```

B.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

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
```

—C—

PROGRESS REPORTS

C.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
```

C.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] 644 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 %s, time: %s]\n" msg
        (time_to_string (now - . state.created)));
    close_channel state

```

—D—

CACHE FILES

D.1 Interface of Cache

```
module type T =
  sig
    type key
    type hash = string
    type value

    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 option
  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
```

D.2 Implementation of Cache

```
let suffix = Config.cache_suffix
let search_path =
```

```

[ Filename.current_dir_name;
  Config.user_cache_dir;
  Config.system_cache_dir ]

module type T =
sig
  type key
  type hash = string
  type value

  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 option
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 file name =
    name ^ suffix

  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
    in
    find_first' path

```



```

        else
            find_first' path
    in
        find_first' path
let find hash name =
    try Some (find_first search_path (file 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 filename = file name in
    let oc, _ = open_out_bin_last search_path filename 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 (file name)) in
    Marshal.to_channel oc { tag = hash; value = value } [];
    close_out oc
exception Mismatch of string × string × string
let read hash name =
    let filename = file name in
    let ic, dir = open_in_bin_first search_path filename in
    let { tag = tag; value = value } = Marshal.from_channel ic in
    close_in ic;
    if tag = hash then
        value
    else
        raise (Mismatch (filename, hash, tag))
let maybe_read hash name =
    try Some (read hash name) with Not_found → None
end

```

—E—

MORE ON LISTS

E.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* : $int \rightarrow \alpha\ list \rightarrow \alpha\ list \times \alpha\ list$

of_subarray $n\ m\ a$ is $[a.(n); a.(n+1); \dots; a.(m)]$. 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 $[n; n+s; n+2s; \dots; m - ((m-n) \bmod s)]$

val *range* : $?stride : int \rightarrow int \rightarrow int \rightarrow int\ list$

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 $List.flatten \circ (List.map\ f)$, but more efficient, because no intermediate lists are built.

```

val flatmap : ( $\alpha \rightarrow \beta \text{ list}$ )  $\rightarrow \alpha \text{ list} \rightarrow \beta \text{ list}$ 
val clone   :  $\text{int} \rightarrow \alpha \rightarrow \alpha \text{ list}$ 
val multiply :  $\text{int} \rightarrow \alpha \text{ list} \rightarrow \alpha \text{ list}$ 

```



Invent other names to avoid confusions with *List.fold_left2* and *List.fold_right2*.

```

val fold_right2 : ( $\alpha \rightarrow \beta \rightarrow \beta$ )  $\rightarrow \alpha \text{ list list} \rightarrow \beta \rightarrow \beta$ 
val fold_left2  : ( $\beta \rightarrow \alpha \rightarrow \beta$ )  $\rightarrow \beta \rightarrow \alpha \text{ list list} \rightarrow \beta$ 

iteri f n [a; b; c] evaluates f n a, f (n+1) b and f (n+2) c.

val iteri : ( $\text{int} \rightarrow \alpha \rightarrow \text{unit}$ )  $\rightarrow \text{int} \rightarrow \alpha \text{ list} \rightarrow \text{unit}$ 

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 : ( $\text{int} \rightarrow \text{int} \rightarrow \alpha \rightarrow \text{unit}$ )  $\rightarrow \text{int} \rightarrow \text{int} \rightarrow \alpha \text{ list list} \rightarrow \text{unit}$ 

val transpose :  $\alpha \text{ list list} \rightarrow \alpha \text{ list list}$ 

```

E.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

```

```

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

let rec flatmap f = function
| [] → []
| x :: rest → f x @ flatmap f rest

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

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 "t1" → 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  $c = \text{cmp } n1 \ n2$  in
      if  $c \neq 0$  then
         $c$ 
      else
         $\text{compare}' \ r1 \ r2$ 
in
   $\text{compare}' \ l1 \ l2$ 
let rec  $\text{uniq}' \ x = \text{function}$ 
  |  $[] \rightarrow []$ 
  |  $x' :: \text{rest} \rightarrow$ 
    if  $x' = x$  then
       $\text{uniq}' \ x \ \text{rest}$ 
    else
       $x' :: \text{uniq}' \ x' \ \text{rest}$ 
let  $\text{uniq} = \text{function}$ 
  |  $[] \rightarrow []$ 
  |  $x :: \text{rest} \rightarrow x :: \text{uniq}' \ x \ \text{rest}$ 
let rec  $\text{homogeneous} = \text{function}$ 
  |  $[] \mid [-] \rightarrow \text{true}$ 
  |  $a1 :: (a2 :: \_ \text{ as } \text{rest}) \rightarrow$ 
    if  $a1 \neq a2$  then
      false
    else
       $\text{homogeneous} \ \text{rest}$ 

```

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  $\text{classify} \ l =$ 
  let rec  $\text{add\_to\_class} \ a = \text{function}$ 
    |  $[] \rightarrow [1, a]$ 
    |  $(n, a') :: \text{rest} \rightarrow$ 
      if  $a = a'$  then
         $(\text{succ } n, a) :: \text{rest}$ 
      else
         $(n, a') :: \text{add\_to\_class} \ a \ \text{rest}$ 
  in
  let rec  $\text{classify}' \ cl = \text{function}$ 
    |  $[] \rightarrow cl$ 
    |  $a :: \text{rest} \rightarrow \text{classify}' \ (\text{add\_to\_class} \ a \ cl) \ \text{rest}$ 
  in
     $\text{classify}' \ [] \ l$ 
let rec  $\text{factorize} \ l =$ 
  let rec  $\text{add\_to\_class} \ x \ y = \text{function}$ 
    |  $[] \rightarrow [(x, [y])]$ 
    |  $(x', ys) :: \text{rest} \rightarrow$ 
      if  $x = x'$  then
         $(x, y :: ys) :: \text{rest}$ 
      else
         $(x', y) :: \text{rest}$ 

```

```

      (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 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

```

—F—

MORE ON ARRAYS

F.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  $\alpha$  compressed
val uniq :  $\alpha$  compressed  $\rightarrow$   $\alpha$  array
val embedding :  $\alpha$  compressed  $\rightarrow$  int array
```

These two are inverses of each other:

```
val compress :  $\alpha$  array  $\rightarrow$   $\alpha$  compressed
val uncompress :  $\alpha$  compressed  $\rightarrow$   $\alpha$  array
```

One can play the same game for matrices.

```
type  $\alpha$  compressed2
val uniq2 :  $\alpha$  compressed2  $\rightarrow$   $\alpha$  array array
val embedding1 :  $\alpha$  compressed2  $\rightarrow$  int array
val embedding2 :  $\alpha$  compressed2  $\rightarrow$  int array
```

Again, these two are inverses of each other:

```
val compress2 :  $\alpha$  array array  $\rightarrow$   $\alpha$  compressed2
val uncompress2 :  $\alpha$  compressed2  $\rightarrow$   $\alpha$  array array
```

F.2 Implementation of ThoArray

```
type  $\alpha$  compressed =
  { uniq :  $\alpha$  array;
    embedding : int array }

let uniq a = a.uniq
let embedding a = a.embedding

type  $\alpha$  compressed2 =
  { uniq2 :  $\alpha$  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 })

```

—G—

POLYMORPHIC MAPS

From [9].

G.1 Interface of Pmap

Module *Pmap*: association tables over a polymorphic type¹.

```

module type T =
  sig
    type ('key,  $\alpha$ ) t
    val empty : ('key,  $\alpha$ ) t
    val is_empty : ('key,  $\alpha$ ) t  $\rightarrow$  bool
    val singleton : 'key  $\rightarrow$   $\alpha$   $\rightarrow$  ('key,  $\alpha$ ) t
    val add : ('key  $\rightarrow$  'key  $\rightarrow$  int)  $\rightarrow$  'key  $\rightarrow$   $\alpha$   $\rightarrow$  ('key,  $\alpha$ ) t  $\rightarrow$ 
      ('key,  $\alpha$ ) t
    val update : ('key  $\rightarrow$  'key  $\rightarrow$  int)  $\rightarrow$  ( $\alpha$   $\rightarrow$   $\alpha$   $\rightarrow$   $\alpha$ )  $\rightarrow$ 
      'key  $\rightarrow$   $\alpha$   $\rightarrow$  ('key,  $\alpha$ ) t  $\rightarrow$  ('key,  $\alpha$ ) t
    val cons : ('key  $\rightarrow$  'key  $\rightarrow$  int)  $\rightarrow$  ( $\alpha$   $\rightarrow$   $\alpha$   $\rightarrow$   $\alpha$  option)  $\rightarrow$ 
      'key  $\rightarrow$   $\alpha$   $\rightarrow$  ('key,  $\alpha$ ) t  $\rightarrow$  ('key,  $\alpha$ ) t
    val find : ('key  $\rightarrow$  'key  $\rightarrow$  int)  $\rightarrow$  'key  $\rightarrow$  ('key,  $\alpha$ ) t  $\rightarrow$   $\alpha$ 
    val find_opt : ('key  $\rightarrow$  'key  $\rightarrow$  int)  $\rightarrow$  'key  $\rightarrow$  ('key,  $\alpha$ ) t  $\rightarrow$   $\alpha$  option
    val choose : ('key,  $\alpha$ ) t  $\rightarrow$  'key  $\times$   $\alpha$ 
    val choose_opt : ('key,  $\alpha$ ) t  $\rightarrow$  ('key  $\times$   $\alpha$ ) option
    val uncons : ('key,  $\alpha$ ) t  $\rightarrow$  'key  $\times$   $\alpha$   $\times$  ('key,  $\alpha$ ) t
    val uncons_opt : ('key,  $\alpha$ ) t  $\rightarrow$  ('key  $\times$   $\alpha$   $\times$  ('key,  $\alpha$ ) t) option
    val elements : ('key,  $\alpha$ ) t  $\rightarrow$  ('key  $\times$   $\alpha$ ) list
    val mem : ('key  $\rightarrow$  'key  $\rightarrow$  int)  $\rightarrow$  'key  $\rightarrow$  ('key,  $\alpha$ ) t  $\rightarrow$  bool
    val remove : ('key  $\rightarrow$  'key  $\rightarrow$  int)  $\rightarrow$  'key  $\rightarrow$  ('key,  $\alpha$ ) t  $\rightarrow$  ('key,  $\alpha$ ) t
    val union : ('key  $\rightarrow$  'key  $\rightarrow$  int)  $\rightarrow$  ( $\alpha$   $\rightarrow$   $\alpha$   $\rightarrow$   $\alpha$ )  $\rightarrow$ 
      ('key,  $\alpha$ ) t  $\rightarrow$  ('key,  $\alpha$ ) t  $\rightarrow$  ('key,  $\alpha$ ) t
    val compose : ('key  $\rightarrow$  'key  $\rightarrow$  int)  $\rightarrow$  ( $\alpha$   $\rightarrow$   $\alpha$   $\rightarrow$   $\alpha$  option)  $\rightarrow$ 
      ('key,  $\alpha$ ) t  $\rightarrow$  ('key,  $\alpha$ ) t  $\rightarrow$  ('key,  $\alpha$ ) t
    val iter : ('key  $\rightarrow$   $\alpha$   $\rightarrow$  unit)  $\rightarrow$  ('key,  $\alpha$ ) t  $\rightarrow$  unit
    val map : ( $\alpha$   $\rightarrow$   $\beta$ )  $\rightarrow$  ('key,  $\alpha$ ) t  $\rightarrow$  ('key,  $\beta$ ) t
    val mapi : ('key  $\rightarrow$   $\alpha$   $\rightarrow$   $\beta$ )  $\rightarrow$  ('key,  $\alpha$ ) t  $\rightarrow$  ('key,  $\beta$ ) t
    val fold : ('key  $\rightarrow$   $\alpha$   $\rightarrow$   $\beta$   $\rightarrow$   $\beta$ )  $\rightarrow$  ('key,  $\alpha$ ) t  $\rightarrow$   $\beta$   $\rightarrow$   $\beta$ 
  end

```

¹Extension 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*

G.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 → 'key → 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,  $\alpha$ ) t =
| Empty
| Node of ('key,  $\alpha$ ) t  $\times$  'key  $\times$   $\alpha$   $\times$  ('key,  $\alpha$ ) t  $\times$  int

let empty = Empty

let is_empty = function
| Empty  $\rightarrow$  true
| _  $\rightarrow$  false

let singleton k d =
Node (Empty, k, d, Empty, 1)

let height = function
| Empty  $\rightarrow$  0
| Node (_, _, _, _, h)  $\rightarrow$  h

let create l x d r =
let hl = height l and hr = height r in
Node (l, x, d, r, (if hl  $\geq$  hr then hl + 1 else hr + 1))

let bal l x d r =
let hl = match l with Empty  $\rightarrow$  0 | Node (_, _, _, _, h)  $\rightarrow$  h in
let hr = match r with Empty  $\rightarrow$  0 | Node (_, _, _, _, h)  $\rightarrow$  h in
if hl > hr + 2 then begin
match l with
| Empty  $\rightarrow$  invalid_arg "Map.bal"
| Node (ll, lv, ld, lr, _)  $\rightarrow$ 
if height ll  $\geq$  height lr then
create ll lv ld (create lr x d r)
else begin
match lr with
| Empty  $\rightarrow$  invalid_arg "Map.bal"
| Node (lrl, lrv, lrd, lrr, _)  $\rightarrow$ 
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  $\rightarrow$  invalid_arg "Map.bal"
| Node (rl, rv, rd, rr, _)  $\rightarrow$ 
if height rr  $\geq$  height rl then
create (create l x d rl) rv rd rr
else begin
match rl with
| Empty  $\rightarrow$  invalid_arg "Map.bal"
| Node (rll, rlv, rld, rlr, _)  $\rightarrow$ 
create (create l x d rll) rlv rld (create rlr rv rd rr)
end
end else
Node (l, x, d, r, (if hl  $\geq$  hr then hl + 1 else hr + 1))

let rec join l x d r =
match bal l x d r with
| Empty  $\rightarrow$  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 $\text{height } t1 - \text{height } t2 \leq 2$.

```

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
      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
      end
    end
  end

```

```

      | 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
      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
      end
    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 v 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

```

—H—

TRIES

From [4], extended for [9].

H.1 Interface of Trie

H.1.1 Monomorphically

module type *T* =

sig

type *key*

type $(+\alpha)$ *t*

val *empty* : α *t*

val *is_empty* : α *t* \rightarrow *bool*

Standard trie interface:

val *add* : *key* \rightarrow $\alpha \rightarrow \alpha$ *t* \rightarrow α *t*

val *find* : *key* \rightarrow α *t* \rightarrow α

Functionals:

val *remove* : *key* \rightarrow α *t* \rightarrow α *t*

val *mem* : *key* \rightarrow α *t* \rightarrow *bool*

val *map* : $(\alpha \rightarrow \beta) \rightarrow \alpha$ *t* \rightarrow β *t*

val *mapi* : $(key \rightarrow \alpha \rightarrow \beta) \rightarrow \alpha$ *t* \rightarrow β *t*

val *iter* : $(key \rightarrow \alpha \rightarrow unit) \rightarrow \alpha$ *t* \rightarrow *unit*

val *fold* : $(key \rightarrow \alpha \rightarrow \beta \rightarrow \beta) \rightarrow \alpha$ *t* \rightarrow $\beta \rightarrow \beta$

Try to match a longest prefix and return the unmatched rest.

val *longest* : *key* \rightarrow α *t* \rightarrow α *option* \times *key*

Try to match a shortest prefix and return the unmatched rest.

val *shortest* : *key* \rightarrow α *t* \rightarrow α *option* \times *key*

H.1.2 New in O’Caml 3.08

val *compare* : $(\alpha \rightarrow \alpha \rightarrow int) \rightarrow \alpha$ *t* \rightarrow α *t* \rightarrow *int*

```
val equal : ( $\alpha \rightarrow \alpha \rightarrow \text{bool}$ )  $\rightarrow \alpha\ t \rightarrow \alpha\ t \rightarrow \text{bool}$ 
```

H.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.

```
val export : ( $\text{int} \rightarrow \text{unit}$ )  $\rightarrow (\text{int} \rightarrow \text{unit}) \rightarrow$   

  ( $\text{int} \rightarrow \text{key} \rightarrow \text{unit}$ )  $\rightarrow (\text{int} \rightarrow \text{key} \rightarrow \alpha \rightarrow \text{unit}) \rightarrow \alpha\ t \rightarrow \text{unit}$   

end  

module Make ( $M : \text{Map}.S$ ) :  $T$  with type key =  $M.\text{key list}$   

module MakeMap ( $M : \text{Map}.S$ ) :  $\text{Map}.S$  with type key =  $M.\text{key list}$ 
```

H.1.4 Polymorphically

```
module type Poly =  
  sig  
    type ( $\alpha, \beta$ )  $t$   
    val empty : ( $\alpha, \beta$ )  $t$ 
```

Standard trie interface:

```
val add : ( $\alpha \rightarrow \alpha \rightarrow \text{int}$ )  $\rightarrow \alpha\ \text{list} \rightarrow \beta \rightarrow (\alpha, \beta)\ t \rightarrow (\alpha, \beta)\ t$   

val find : ( $\alpha \rightarrow \alpha \rightarrow \text{int}$ )  $\rightarrow \alpha\ \text{list} \rightarrow (\alpha, \beta)\ t \rightarrow \beta$ 
```

Functionals:

```
val remove : ( $\alpha \rightarrow \alpha \rightarrow \text{int}$ )  $\rightarrow \alpha\ \text{list} \rightarrow (\alpha, \beta)\ t \rightarrow (\alpha, \beta)\ t$   

val mem : ( $\alpha \rightarrow \alpha \rightarrow \text{int}$ )  $\rightarrow \alpha\ \text{list} \rightarrow (\alpha, \beta)\ t \rightarrow \text{bool}$   

val map : ( $\beta \rightarrow \gamma$ )  $\rightarrow (\alpha, \beta)\ t \rightarrow (\alpha, \gamma)\ t$   

val mapi : ( $\alpha\ \text{list} \rightarrow \beta \rightarrow \gamma$ )  $\rightarrow (\alpha, \beta)\ t \rightarrow (\alpha, \gamma)\ t$   

val iter : ( $\alpha\ \text{list} \rightarrow \beta \rightarrow \text{unit}$ )  $\rightarrow (\alpha, \beta)\ t \rightarrow \text{unit}$   

val fold : ( $\alpha\ \text{list} \rightarrow \beta \rightarrow \gamma \rightarrow \gamma$ )  $\rightarrow (\alpha, \beta)\ t \rightarrow \gamma \rightarrow \gamma$ 
```

Try to match a longest prefix and return the unmatched rest.

```
val longest : ( $\alpha \rightarrow \alpha \rightarrow \text{int}$ )  $\rightarrow \alpha\ \text{list} \rightarrow (\alpha, \beta)\ t \rightarrow \beta\ \text{option} \times \alpha\ \text{list}$ 
```

Try to match a shortest prefix and return the unmatched rest.

```
val shortest : ( $\alpha \rightarrow \alpha \rightarrow \text{int}$ )  $\rightarrow \alpha\ \text{list} \rightarrow (\alpha, \beta)\ t \rightarrow \beta\ \text{option} \times \alpha\ \text{list}$ 
```

H.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 : ( $\text{int} \rightarrow \text{unit}$ )  $\rightarrow (\text{int} \rightarrow \text{unit}) \rightarrow$ 
```

```

      (int →  $\alpha$  list → unit) → (int →  $\alpha$  list →  $\beta$  → unit) → ( $\alpha$ ,  $\beta$ ) t →
unit
end
module MakePoly (M : Pmap.T) : Poly

```

H.2 Implementation of *Trie*

H.2.1 Monomorphically

```

module type T =
sig
  type key
  type (+ $\alpha$ ) t
  val empty :  $\alpha$  t
  val is_empty :  $\alpha$  t → bool
  val add : key →  $\alpha$  →  $\alpha$  t →  $\alpha$  t
  val find : key →  $\alpha$  t →  $\alpha$ 
  val remove : key →  $\alpha$  t →  $\alpha$  t
  val mem : key →  $\alpha$  t → bool
  val map : ( $\alpha$  →  $\beta$ ) →  $\alpha$  t →  $\beta$  t
  val mapi : (key →  $\alpha$  →  $\beta$ ) →  $\alpha$  t →  $\beta$  t
  val iter : (key →  $\alpha$  → unit) →  $\alpha$  t → unit
  val fold : (key →  $\alpha$  →  $\beta$  →  $\beta$ ) →  $\alpha$  t →  $\beta$  →  $\beta$ 
  val longest : key →  $\alpha$  t →  $\alpha$  option × key
  val shortest : key →  $\alpha$  t →  $\alpha$  option × key
  val compare : ( $\alpha$  →  $\alpha$  → int) →  $\alpha$  t →  $\alpha$  t → int
  val equal : ( $\alpha$  →  $\alpha$  → bool) →  $\alpha$  t →  $\alpha$  t → bool
  val export : (int → unit) → (int → unit) →
    (int → key → unit) → (int → key →  $\alpha$  → unit) →  $\alpha$  t → unit
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  $\alpha$  t = Trie of  $\alpha$  option ×  $\alpha$  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

```

H.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)

```

H.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

```

H.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

```

— I —

TENSOR PRODUCTS

From [9].

I.1 Interface of Product

I.1.1 Lists

Since April 2001, we preserve lexicographic ordering.

```
val fold2 : ( $\alpha \rightarrow \beta \rightarrow \gamma \rightarrow \gamma$ )  $\rightarrow \alpha$  list  $\rightarrow \beta$  list  $\rightarrow \gamma \rightarrow \gamma$ 
val fold3 : ( $\alpha \rightarrow \beta \rightarrow \gamma \rightarrow \delta \rightarrow \delta$ )  $\rightarrow \alpha$  list  $\rightarrow \beta$  list  $\rightarrow \gamma$  list  $\rightarrow \delta \rightarrow \delta$ 
val fold : ( $\alpha$  list  $\rightarrow \beta \rightarrow \beta$ )  $\rightarrow \alpha$  list list  $\rightarrow \beta \rightarrow \beta$ 

val list2 : ( $\alpha \rightarrow \beta \rightarrow \gamma$ )  $\rightarrow \alpha$  list  $\rightarrow \beta$  list  $\rightarrow \gamma$  list
val list3 : ( $\alpha \rightarrow \beta \rightarrow \gamma \rightarrow \delta$ )  $\rightarrow \alpha$  list  $\rightarrow \beta$  list  $\rightarrow \gamma$  list  $\rightarrow \delta$  list
val list : ( $\alpha$  list  $\rightarrow \beta$ )  $\rightarrow \alpha$  list list  $\rightarrow \beta$  list

val power : int  $\rightarrow \alpha$  list  $\rightarrow \alpha$  list list
val thread :  $\alpha$  list list  $\rightarrow \alpha$  list list
```

I.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 α set

```
type ( $\alpha$ , 'a_set,  $\beta$ ) fold = ( $\alpha \rightarrow \beta \rightarrow \beta$ )  $\rightarrow$  'a_set  $\rightarrow \beta \rightarrow \beta$ 
```

```
type ( $\alpha$ , 'a_set,  $\beta$ , 'b_set,  $\gamma$ ) fold2 =
  ( $\alpha \rightarrow \beta \rightarrow \gamma \rightarrow \gamma$ )  $\rightarrow$  'a_set  $\rightarrow$  'b_set  $\rightarrow \gamma \rightarrow \gamma$ 
```

```
val outer : ( $\alpha$ , 'a_set,  $\gamma$ ) fold  $\rightarrow$  ( $\beta$ , 'b_set,  $\gamma$ ) fold  $\rightarrow$ 
```

```
  ( $\alpha$ , 'a_set,  $\beta$ , 'b_set,  $\gamma$ ) fold2
```

```
val outer_self : ( $\alpha$ , 'a_set,  $\beta$ ) fold  $\rightarrow$  ( $\alpha$ , 'a_set,  $\alpha$ , 'a_set,  $\beta$ ) fold2
```

I.2 Implementation of *Product*

I.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:

$$[[a_1; \dots; a_k]; [b_1; \dots; b_k]; [c_1; \dots; c_k]; \dots] \rightarrow [[a_1; b_1; c_1; \dots]; [a_2; b_2; c_2; \dots]; \dots] \quad (\text{I.1})$$



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)
| [] → []

```

I.2.2 Sets

The implementation is amazingly simple:

```
type α set
```

```
type (α, 'a_set, β) fold = (α → β → β) → 'a_set → β → β
```

```
type (α, 'a_set, β, 'b_set, γ) fold2 =
  (α → β → γ → γ) → 'a_set → 'b_set → γ → γ
```

```
let outer fold1 fold2 f l1 l2 = fold1 (fun x1 → fold2 (f x1) l2) l1
```

```
let outer_self fold f l1 l2 = fold (fun x1 → fold (f x1) l2) l1
```

—J—

COMBINATORICS

J.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`

J.1.1 Simple Combinatorial Functions

The functions

$$\text{factorial} : n \rightarrow n! \quad (\text{J.1a})$$

$$\text{binomial} : (n, k) \rightarrow \binom{n}{k} = \frac{n!}{k!(n-k)!} \quad (\text{J.1b})$$

$$\text{multinomial} : [n_1; n_2; \dots; n_k] \rightarrow \binom{n_1 + n_2 + \dots + n_k}{n_1, n_2, \dots, n_k} = \frac{(n_1 + n_2 + \dots + n_k)!}{n_1! n_2! \dots n_k!} \quad (\text{J.1c})$$

have not been optimized. They can quickly run out of the range of native integers.

`val factorial : int → int`
`val binomial : int → int → int`
`val multinomial : int list → 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 → int`

J.1.2 Partitions

partitions $[n_1; n_2; \dots; n_k] [x_1; x_2; \dots; x_n]$, where $n = n_1 + n_2 + \dots + n_k$, returns all inequivalent partitions of $[x_1; x_2; \dots; x_n]$ into parts of size n_1, n_2, \dots, n_k . The order of the n_i is not respected. There are

$$\frac{1}{S(n_1, n_2, \dots, n_k)} \binom{n_1 + n_2 + \dots + n_k}{n_1, n_2, \dots, n_k} \quad (\text{J.2})$$

such partitions, where the symmetry factor $S(n_1, n_2, \dots, n_k)$ is the size of the permutation group of $[n_1; n_2; \dots; n_k]$ 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

$$\binom{n_1 + n_2 + \dots + n_k}{n_1, n_2, \dots, n_k} \quad (J.3)$$

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 though 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

$$\frac{1}{S(|l| - k, k)} \binom{|l|}{k} \quad (J.4)$$

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

$$\binom{|l|}{k} \quad (J.5)$$

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 : $int \rightarrow int \rightarrow \alpha\ seq \rightarrow (\alpha\ seq\ list \times \alpha\ seq)\ list$

J.1.3 Choices

choose n $[x_1; x_2; \dots; x_n]$ returns the list of all n -element subsets of $[x_1; x_2; \dots; x_n]$.
choose n is equivalent to $(\text{map fst}) \circ (\text{ordered_split } n)$.

`val choose : int → α seq → α seq list`

multi_choose n k is equivalent to $(\text{map fst}) \circ (\text{multi_split } n \ k)$.

`val multi_choose : int → int → α seq → α seq list list`

`val ordered_multi_choose : int → int → α seq → α seq list list`

J.1.4 Permutations

`val permute : α seq → α seq list`

Graded Permutations

`val permute_signed : α seq → (int × α seq) list`

`val permute_even : α seq → α seq list`

`val permute_odd : α seq → α seq list`

Tensor Products of Permutations

In other words: permutations which respect compartmentalization.

`val permute_tensor : α seq list → α seq list list`

`val permute_tensor_signed : α seq list → (int × α seq list) list`

`val permute_tensor_even : α seq list → α seq list list`

`val permute_tensor_odd : α seq list → α seq list list`

Sorting

`val sort_signed : (α → α → int) → α list → int × α list`

J.2 Implementation of Combinatorics

`type α seq = α list`

J.2.1 Simple Combinatorial Functions

```

let rec factorial' fn n =
  if n < 1 then
    fn
  else
    factorial' (n × fn) (pred n)

let factorial n =
  if 0 ≤ n ∧ n ≤ 12 then
    factorial' 1 n
  else
    invalid_arg "Combinatorics.factorial"

```

$$\begin{aligned}
\binom{n}{k} &= \frac{n!}{k!(n-k)!} = \frac{n(n-1)\cdots(n-k+1)}{k(k-1)\cdots 1} \\
&= \frac{n(n-1)\cdots(k+1)}{(n-k)(n-k-1)\cdots 1} = \begin{cases} B_{n-k+1}(n, k) & \text{for } k \leq \lfloor n/2 \rfloor \\ B_{k+1}(n, n-k) & \text{for } k > \lfloor n/2 \rfloor \end{cases} \quad (\text{J.6})
\end{aligned}$$

where

$$B_{n_{\min}}(n, k) = \begin{cases} nB_{n_{\min}}(n-1, k) & \text{for } n \geq n_{\min} \\ \frac{1}{k}B_{n_{\min}}(n, k-1) & \text{for } k > 1 \\ 1 & \text{otherwise} \end{cases} \quad (\text{J.7})$$

```

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:

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1} \quad (\text{J.8})$$

```

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)

```

J.2.2 Partitions

The inner steps of the recursion (i.e. $n = 1$) are expanded as follows

$$\begin{aligned}
 \text{split}'(1, [p_k; p_{k-1}; \dots; p_1], [x_l; x_{l-1}; \dots; x_1], [x_{l+1}; x_{l+2}; \dots; x_m]) = \\
 & ([p_1; \dots; p_k; x_{l+1}], [x_1; \dots; x_l; x_{l+2}; \dots; x_m]); \\
 & ([p_1; \dots; p_k; x_{l+2}], [x_1; \dots; x_l; x_{l+1}; x_{l+3}; \dots; x_m]); \dots; \\
 & ([p_1; \dots; p_k; x_m], [x_1; \dots; x_l; x_{l+1}; \dots; x_{m-1}]) \quad (\text{J.9})
 \end{aligned}$$

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 n th level we have

$$\begin{aligned}
 \text{split}'(n, [p_k; p_{k-1}; \dots; p_1], [x_l; x_{l-1}; \dots; x_1], [x_{l+1}; x_{l+2}; \dots; x_m]) = \\
 & ([p_1; \dots; p_k; x_{l+1}; x_{l+2}; \dots; x_{l+n}], [x_1; \dots; x_l; x_{l+n+1}; \dots; x_m]); \dots; \\
 & ([p_1; \dots; p_k; x_{m-n+1}; x_{m-n+2}; \dots; x_m], [x_1; \dots; x_l; x_{l+1}; \dots; x_{m-n}]) \quad (\text{J.10})
 \end{aligned}$$

where the order of the $[x_1; x_2; \dots; x_m]$ 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 $n \in \{0, |l|\}$ explicitly. 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
end

```

If we chop off parts repeatedly, we can either keep permutations or suppress them. Generically, *attach_to_fst* has type

$$(\alpha \times \beta) \text{ list} \rightarrow \alpha \text{ list} \rightarrow (\alpha \text{ list} \times \beta) \text{ list} \rightarrow (\alpha \text{ list} \times \beta) \text{ list}$$

and semantics

$$\begin{aligned} \text{attach_to_fst}([(a_1, b_1), (a_2, b_2), \dots, (a_m, b_m)], [a'_1, a'_2, \dots]) = \\ [[a_1, a'_1, \dots], b_1], [[a_2, a'_1, \dots], b_2], \dots, [[a_m, a'_1, \dots], b_m] \end{aligned} \quad (\text{J.11})$$

(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 second 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.htdl"
| 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 reflections consistently. For this to work, the lengths of the parts *must not* be reordered arbitrarily. Ordering with monotonously falling 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))

```

J.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  $n = \text{abs\_}l$  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  $l$ )

```

J.2.4 Permutations

```

let rec insert  $x$  = function
| [] → [[ $x$ ]]
|  $h :: t$  as  $l$  → ( $x :: l$ ) :: List.map (fun  $l'$  →  $h :: l'$ ) (insert  $x$   $t$ )
let permute  $l$  =
  List.fold_left (fun acc  $x$  → ThoList.flatmap (insert  $x$ ) acc) [[]]  $l$ 

```

Graded Permutations

```

let rec insert_signed  $x$  = function
| ( $\text{eps}$ , []) → [( $\text{eps}$ , [ $x$ ])]
| ( $\text{eps}$ ,  $h :: t$ ) → ( $\text{eps}$ ,  $x :: h :: t$ ) ::
  (List.map (fun ( $\text{eps}'$ ,  $l'$ ) → ( $-\text{eps}'$ ,  $h :: l'$ )) (insert_signed  $x$  ( $\text{eps}$ ,  $t$ )))
let rec permute_signed' = function
| ( $\text{eps}$ , []) → [( $\text{eps}$ , [])]
| ( $\text{eps}$ ,  $h :: t$ ) → ThoList.flatmap (insert_signed  $h$ ) (permute_signed' ( $\text{eps}$ ,  $t$ ))
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 ( $\text{eps}$ , _) →  $\text{eps} = s$ )  $l$ )
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

```

—K—

PARTITIONS

K.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 $[n_1; n_2; \dots; n_d]$ with $n_{\min} \leq n_1 \leq n_2 \leq \dots \leq n_d \leq n_{\max}$ and

$$\sum_{i=1}^d n_i = n \quad (\text{K.1})$$

`val tuples : int → int → int → int → int list list`
`val rcs : RCS.t`

K.2 Implementation of Partition

```
let rcs = RCS.parse "Partition" ["Partitions"]
  { RCS.revision = "$Revision: 759$";
    RCS.date = "$Date: 2009-06-10 11:38:07 +0200 (Wed, 10 Jun 2009)$";
    RCS.author = "$Author: ohl$";
    RCS.source
      = "$URL: svn+ssh://jr-reuter@login.hepforge.org/hepforge/svn/whizard/trunk/src/omeg
```

All unordered pairs of integers with the same sum n in a given range $\{n_1, \dots, n_2\}$:

$$\text{pairs} : (n, n_1, n_2) \rightarrow \{(i, j) \mid i + j = n \wedge n_1 \leq i \leq j \leq n_2\} \quad (\text{K.2})$$

```
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)

```

—L— TREES

From [10]: Trees with one root admit a straightforward recursive definition

$$T(N, L) = L \cup N \times T(N, L) \times T(N, L) \quad (\text{L.1})$$

that is very well adapted to mathematical reasoning. Such recursive definitions are useful because they allow us to prove properties of elements by induction

$$\begin{aligned} \forall l \in L : p(l) \wedge (\forall n \in N : \forall t_1, t_2 \in T(N, L) : p(t_1) \wedge p(t_2) \Rightarrow p(n \times t_1 \times t_2)) \\ \implies \forall t \in T(N, L) : p(t) \end{aligned} \quad (\text{L.2})$$

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

$$p(1) \wedge (\forall n \in \mathbf{N} : p(n) \Rightarrow p(n+1)) \implies \forall n \in \mathbf{N} : p(n) \quad (\text{L.3})$$

The recursive definition (L.1) is mirrored by the two tree construction functions¹

$$\text{leaf} : \nu \times \lambda \rightarrow (\nu, \lambda)T \quad (\text{L.4a})$$

$$\text{node} : \nu \times (\nu, \lambda)T \times (\nu, \lambda)T \rightarrow (\nu, \lambda)T \quad (\text{L.4b})$$

Renaming leaves and nodes leaves the structure of the tree invariant. Therefore, morphisms $L \rightarrow L'$ and $N \rightarrow N'$ of the sets of leaves and nodes induce natural homomorphisms $T(N, L) \rightarrow T(N', L')$ of trees

$$\text{map} : (\nu \rightarrow \nu') \times (\lambda \rightarrow \lambda') \times (\nu, \lambda)T \rightarrow (\nu', \lambda')T \quad (\text{L.5})$$

The homomorphisms constructed by *map* are trivial, but ubiquitous. More interesting are the morphisms

$$\begin{aligned} \text{fold} : (\nu \times \lambda \rightarrow \alpha) \times (\nu \times \alpha \times \alpha \rightarrow \alpha) \times (\nu, \lambda)T &\rightarrow \alpha \\ (f_1, f_2, l \in L) &\mapsto f_1(l) \\ (f_1, f_2, (n, t_1, t_2)) &\mapsto f_2(n, \text{fold}(f_1, f_2, t_1), \text{fold}(f_1, f_2, t_2)) \end{aligned} \quad (\text{L.6})$$

¹To 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 $\alpha \rightarrow \beta \rightarrow \gamma$ for $\alpha \times \beta \rightarrow \gamma$.

and

$$\begin{aligned}
fan : (\nu \times \lambda \rightarrow \{\alpha\}) \times (\nu \times \alpha \times \alpha \rightarrow \{\alpha\}) \times (\nu, \lambda)T &\rightarrow \{\alpha\} \\
(f_1, f_2, l \in L) &\mapsto f_1(l) \\
(f_1, f_2, (n, t_1, t_2)) &\mapsto f_2(n, fold(f_1, f_2, t_1) \otimes fold(f_1, f_2, t_2))
\end{aligned} \tag{L.7}$$

where the tensor product notation means that f_2 is applied to all combinations of list members in the argument:

$$\phi(\{x\} \otimes \{y\}) = \{\phi(x, y) | x \in \{x\} \wedge y \in \{y\}\} \tag{L.8}$$

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

$$\begin{aligned}
map : (\nu \rightarrow \nu') \times (\lambda \rightarrow \lambda') \times (\nu, \lambda)T &\rightarrow (\nu', \lambda')T \\
(f, g, t) &\mapsto fold(leaf \circ (f \times g), node \circ (f \times id \times id), t)
\end{aligned} \tag{L.9}$$

$fold$ 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 $\{(\nu, \{p_1, \dots, p_n\})T\}$ of a given topology $t \in (\emptyset, \{1, \dots, n\})T$.



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.

L.1 Interface of *Tree*

This module provides utilities for generic decorated trees, such as FeynMF output.

L.1.1 Abstract Data Type

type $(\nu, \lambda) t$

$leaf\ n\ l$ returns a tree consisting of a single leaf of type n connected to l .

val $leaf : \nu \rightarrow \lambda \rightarrow (\nu, \lambda) t$

$cons\ n\ ch$ returns a tree node.

val $cons : \nu \rightarrow (\nu, \lambda) t\ list \rightarrow (\nu, \lambda) t$

node *t* returns the top node of the tree *t*.

val *node* : (ν , λ) *t* \rightarrow ν

leafs *t* returns a list of all leafs *in order*.

val *leafs* : (ν , λ) *t* \rightarrow λ *list*

nodes *t* returns a list of all nodes in post-order. This guarantees that the root node can be stripped from the result by *List.tl*.

val *nodes* : (ν , λ) *t* \rightarrow ν *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 $\lambda \rightarrow ((\nu, \lambda) t \rightarrow \text{bool}) \rightarrow (\nu, \lambda) t \text{ list} \rightarrow (\nu, \lambda) t$

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* : ($\lambda \rightarrow \lambda \rightarrow \text{bool}$) \rightarrow (ν , λ) *t* \rightarrow (ν , λ) *t*

L.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 ($\nu \rightarrow \alpha \text{ list} \rightarrow \alpha$) \rightarrow (ν , λ) *t* \rightarrow α

val *fan* : ($\nu \rightarrow \lambda \rightarrow \alpha \text{ list}$) \rightarrow ($\nu \rightarrow \alpha \text{ list} \rightarrow \alpha \text{ list}$) \rightarrow
 (ν , λ) *t* \rightarrow $\alpha \text{ list}$

L.1.3 Output

val *to_string* : (*string*, *string*) *t* \rightarrow *string*

Feynmf



style : (*string* \times *string*) *option* should be replaced by *style* : *string option*; *tex_label* : *string option*

type *feynmf* =
 { *style* : (*string* \times *string*) *option*;
 rev : *bool*;
 label : *string option*;
 tension : *float option* }

val *vanilla* : *feynmf*

```
val sty : (string × string) × bool × string → feynmf
```

to_feynmf file to_string i2 t write the trees in the list *t* to the file named *file*. The leaf *i2* is used as the second incoming particle and *to_string* is used to convert leaf labels to L^AT_EX-strings.

```
val to_feynmf : bool ref → string → (λ → string) → λ → (feynmf, λ) t list → unit
```

Least Squares Layout

A general graph with edges of type ε , internal nodes of type ν , and external nodes of type $'ext$.

```
type (ε, ν, 'ext) graph
```

```
val graph_of_tree : (ν → ν → ε) → (ν → ν) → ν → (ν, ν) t → (ε, ν, ν) graph
```

A general graph with the layout of the external nodes fixed.

```
type (ε, ν, 'ext) ext_layout
```

```
val left_to_right : int → (ε, ν, 'ext) graph → (ε, ν, '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
```

L.2 Implementation of *Tree*

L.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 leaves *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

```

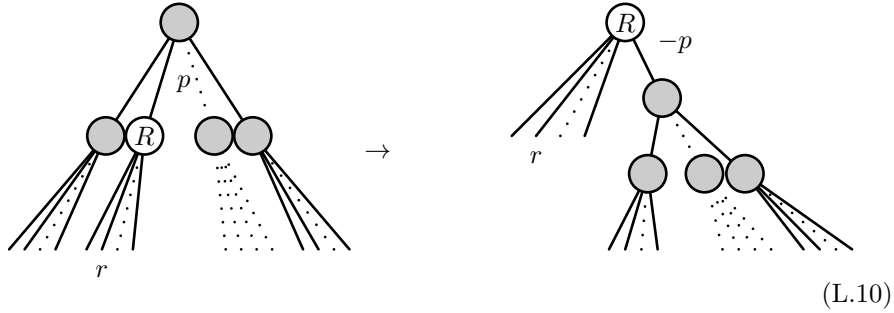
first_match p list returns $(x, list')$, where x is the first element of *list* for which $p\ x = \text{true}$ and *list'* is *list* sans x .

```

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:



```

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 (L.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 ( $\alpha$ ,  $\beta$ ) with_supremum = { sup :  $\alpha$ ; data :  $\beta$  }
```

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
```

L.2.2 Homomorphisms

Isomorphisms 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) []
```

L.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  $\alpha$  supremum_or_infinity = Infinity | Sup of  $\alpha$ 
type ( $\alpha$ ,  $\beta$ ) with_supremum_or_infinity =
  { sup :  $\alpha$  supremum_or_infinity; data :  $\beta$  }
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}\n" s lab io leaf

```

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_string 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_string leaf) (tex_lbl prop) prop.label ;
  fprintf tex "\fmfdot{v%d}\n" n;
  if rev then
    fprintf tex "\fmf{s}{s,v%d}\n"
      (species prop) tension io (to_string leaf) n
  else
    fprintf tex "\fmf{s}{v%d,s}\n"
      (species prop) tension n io (to_string leaf)

and int_node tex to_string 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_string i2 n' t

and leaf_or_int_node tex to_string i2 n n' = function
| Leaf (prop, l) → leaf_node tex to_string i2 n prop l
| Node (prop, _) as t → int_node tex to_string i2 n n' prop t

and edges_feynmf' tex to_string i2 n = function
| Leaf (prop, l) → leaf_node tex to_string i2 n prop l
| Node (_, ch) →
  ignore (List.fold_right
    (fun t' n' →
      leaf_or_int_node tex to_string i2 n n' t';
      succ n') ch (4 × n))

let edges_feynmf tex to_string i2 t =
  let n = 1 in
  begin match t with
  | Leaf _ → ()
  | Node (prop, _) →

```

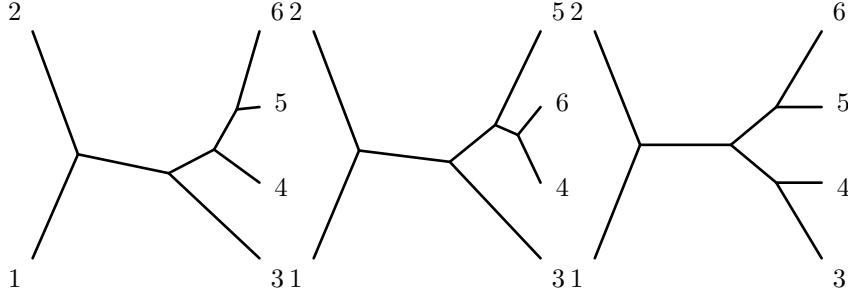


Figure L.1: Note that this is subtly different ...

```

leaf_label tex "i" "1" (tex_lbl prop) prop.label;
if prop.rev then
  fprintf tex "\fmf{%s}{i1,v%d}\n" (species prop) n
else
  fprintf tex "\fmf{%s}{v%d,i1}\n" (species prop) n
end;
fprintf tex "\fmfdot{v%d}\n" n;
edges_feynmf' tex to_string i2 n t

let to_feynmf_channel tex to_string i2 t =
  let t' = sort_2i (≤) i2 t in
  let out = List.map to_string (List.filter (fun a → i2 ≠ a) (leaves t')) in
  fprintf tex "\fmfframe(6,7)(6,6){%%\n";
  fprintf tex "\begin{fmfgraph*}(35,30)\n";
  fprintf tex "\fmfpen{.1pt}\n";
  fprintf tex "\fmfset{arrow_len}{2mm}\n";
  fprintf tex "\fmfleft{i1,i%s}\n" (to_string i2);
  fprintf tex "\fmfright{o%s}\n" (String.concat ",o" out);
  List.iter (fun s → fprintf tex "\fmflabel{${s$}{i%s}\n" s s)
    ["1"; (to_string i2)];
  List.iter (fun s → fprintf tex "\fmflabel{${s$}{o%s}\n" s s) out;
  edges_feynmf tex to_string i2 t';
  fprintf tex "\end{fmfgraph*}}\n"

let to_feynmf_latex file to_string i2 t =
  if !latex then
    let tex = open_out (file ^ ".tex") in
    fprintf tex "\documentclass[10pt]{article}\n";
    fprintf tex "\usepackage{feynmp}\n";
    fprintf tex "\textwidth18.5cm\n";
    fprintf tex "\evensidemargin1.5cm\n";
    fprintf tex "\oddsidemargin1.5cm\n";
    fprintf tex "\setlength{\unitlength}{1mm}\n";
    fprintf tex "\begin{document}\n";
    fprintf tex "\begin{fmffile}{%s.fmf}\n" file;
    List.iter (to_feynmf_channel tex to_string i2) t;

```

```

    fprintf tex "\n";
    fprintf tex "\\end{fmffile}\n";
    fprintf tex "\\end{document}\n";
    close_out tex
  else
    let tex = open_out file in
      List.iter (to_feynmf_channel tex to_string i2) t;
      close_out tex
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 }

```

L.2.4 Least Squares Layout

$$L = \frac{1}{2} \sum_{i' \neq i} T_{ii'} (x_i - x_{i'})^2 + \frac{1}{2} \sum_{i,j} T'_{ij} (x_i - e_j)^2 \quad (\text{L.11})$$

and thus

$$0 = \frac{\partial L}{\partial x_i} = \sum_{i' \neq i} T_{ii'} (x_i - x_{i'}) + \sum_j T'_{ij} (x_i - e_j) \quad (\text{L.12})$$

or

$$\left(\sum_{i' \neq i} T_{ii'} + \sum_j T'_{ij} \right) x_i - \sum_{i' \neq i} T_{ii'} x_{i'} = \sum_j T'_{ij} e_j \quad (\text{L.13})$$

where we can assume that

$$T_{ii'} = T_{i'i} \quad (\text{L.14a})$$

$$T_{ii} = 0 \quad (\text{L.14b})$$

```

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 leaves_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 :  $\alpha$  array →
( $\alpha$ , 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 (L.13)

$$Ax = b_x \quad (\text{L.15a})$$

$$Ay = b_y \quad (\text{L.15b})$$

with

$$A_{ii'} = \left(\sum_{i'' \neq i} T_{ii''} + \sum_j T'_{ij} \right) \delta_{ii'} - T_{ii'} \quad (\text{L.16a})$$

$$(b_{x/y})_i = \sum_j T'_{ij} (e_{x/y})_j \quad (\text{L.16b})$$

```

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"

```

—M—

CONSISTENCY CHECKS



Application count.ml unavailable!

—N—

COMPLEX NUMBERS

N.1 Interface of Complex

```
module type T =
  sig
    type t
    val null : t
    val one : t

    val real : t → float
    val imag : t → float

    val conj : t → t
    val neg : t → t
    val inv : t → t

    val add : t → t → t
    val sub : t → t → t
    val mul : t → t → t
    val div : t → t → t

    val abs : t → float
    val arg : t → float

    val sqrt : t → t
    val exp : t → t
    val log : t → t

    val of_float2 : float → float → t
    val of_int2 : int → int → t
    val to_float2 : t → float × float
    val to_int2 : t → int × int

    val of_float : float → t
    val of_int : int → t
    val to_float : t → float
    val to_int : t → int

    val to_string : t → string
    val of_string : α → β
  end
```

```

module Dense : T
module Default : T

```

N.2 Implementation of Complex

```

module type T =
  sig
    type t

    val null : t
    val one : t

    val real : t → float
    val imag : t → float

    val conj : t → t
    val neg : t → t
    val inv : t → t

    val add : t → t → t
    val sub : t → t → t
    val mul : t → t → t
    val div : t → t → t

    val abs : t → float
    val arg : t → float

    val sqrt : t → t
    val exp : t → t
    val log : t → t

    val of_float2 : float → float → t
    val of_int2 : int → int → t
    val to_float2 : t → float × float
    val to_int2 : t → int × int

    val of_float : float → t
    val of_int : int → t
    val to_float : t → float
    val to_int : t → int

    val to_string : t → string
    val of_string : α → β
  end

```

The hairier formulae are “inspired” by [14].

```

module Dense =
  struct

    type t = { re : float; im : float }
    let null = { re = 0.0; im = 0.0 }
    let one = { re = 1.0; im = 0.0 }

    let real z = z.re

```

```

let imag z = z.im
let conj z = {re = z.re; im = ~-. (z.im) }

let neg z = {re = ~-. (z.re); im = ~-. (z.im) }
let add z1 z2 = {re = z1.re +. z2.re; im = z1.im +. z2.im }
let sub z1 z2 = {re = z1.re -. z2.re; im = z1.im -. z2.im }

```

Save one multiplication with respect to the standard formula

$$(x + iy)(u + iv) = [xu - yv] + i[(x + u)(y + v) - xu - yv] \quad (\text{N.1})$$

at the expense of one addition and two subtractions.

```

let mul z1 z2 =
  let re12 = z1.re *. z2.re
  and im12 = z1.im *. z2.im in
  { re = re12 -. im12;
    im = (z1.re +. z1.im) *. (z2.re +. z2.im) -. re12 -. im12 }

```

$$\frac{x + iy}{u + iv} = \begin{cases} \frac{[x+y(v/u)]+i[y-x(v/u)]}{u+v(v/u)} & \text{for } |u| \geq |v| \\ \frac{[x(u/v)+y]+i[y(u/v)-x]}{u(u/v+v)} & \text{for } |u| < |v| \end{cases} \quad (\text{N.2})$$

```

let div z1 z2 =
  if abs_float z2.re ≥ abs_float z2.im then
    let r = z2.im /. z2.re in
    let den = z2.re +. r *. z2.im in
    { re = (z1.re +. r *. z1.im) /. den;
      im = (z1.im -. r *. z1.re) /. den }
  else
    let r = z2.re /. z2.im in
    let den = z2.im +. r *. z2.re in
    { re = (r *. z1.re +. z1.im) /. den;
      im = (r *. z1.im -. z1.re) /. den }

let inv = div one

```

$$|x + iy| = \begin{cases} |x|\sqrt{1 + (y/x)^2} & \text{for } |x| \geq |y| \\ |y|\sqrt{1 + (x/y)^2} & \text{for } |x| < |y| \end{cases} \quad (\text{N.3})$$

```

let abs z =
  let absr = abs_float z.re
  and absi = abs_float z.im in
  if absr = 0.0 then
    absi
  else if absi = 0.0 then
    absr
  else if absr > absi then
    let q = absi /. absr in
    absr *. sqrt (1.0 +. q *. q)
  else
    let q = absr /. absi in

```

```
absi *. sqrt (1.0 +. q *. q)
```

```
let arg z = atan2 z.im z.re
```

Square roots are trickier:

$$\sqrt{x + iy} = \begin{cases} 0 & \text{for } w = 0 \\ w + i \left(\frac{y}{2w} \right) & \text{for } w \neq 0, x \geq 0 \\ \left(\frac{|y|}{2w} \right) + iw & \text{for } w \neq 0, x < 0, y \geq 0 \\ \left(\frac{|y|}{2w} \right) - iw & \text{for } w \neq 0, x < 0, y < 0 \end{cases} \quad (\text{N.4})$$

where

$$w = \begin{cases} 0 & \text{for } x = y = 0 \\ \sqrt{|x|} \sqrt{\frac{1 + \sqrt{1 + (y/x)^2}}{2}} & \text{for } |x| \geq |y| \\ \sqrt{|y|} \sqrt{\frac{|x/y| + \sqrt{1 + (x/y)^2}}{2}} & \text{for } |x| < |y| \end{cases} \quad (\text{N.5})$$

Equation (N.4) is encoded in *cont w*.

```
let sqrt z =
  if z.re = 0.0 ∧ z.im = 0.0 then
    { re = 0.0; im = 0.0 }
  else
    let absr = abs_float z.re
    and absi = abs_float z.im
    and cont w =
      if z.re ≥ 0.0 then
        { re = w; im = z.im /. (2. *. w) }
      else
        let im = if z.im ≥ 0.0 then w else ~-. w in
        { re = z.im /. (2. *. im); im = im }
    in
    if absr ≥ absi then
      let q = absi /. absr in
      cont ((sqrt absr) *. sqrt (0.5 *. (1.0 +. sqrt (1.0 +. q *. q))))
    else
      let q = absr /. absi in
      cont ((sqrt absi) *. sqrt (0.5 *. (q +. sqrt (1.0 +. q *. q))))

let exp z =
  let er = exp z.re in
  { re = er *. (cos z.im); im = er *. (sin z.im) }

let log z = { re = log (abs z); im = arg z }

let of_float2 r i = { re = r; im = i }
let of_int2 r i = { re = float r; im = float i }
let to_float2 z = (z.re, z.im)
let to_int2 z = (truncate z.re, truncate z.im)
let of_float r = { re = r; im = 0.0 }
let of_int r = { re = float r; im = 0.0 }
let to_float z = z.re
```

```

let to_int z = truncate z.re
let to_string z =
  if z.re ≠ 0.0 ∧ z.im ≠ 0.0 then
    Printf.sprintf "%g+%gi" (* starting from 3.04: "%g+%gi" *) z.re z.im
  else if z.re ≠ 0.0 then
    Printf.sprintf "%g" z.re
  else if z.im ≠ 0.0 then
    Printf.sprintf "%gi" z.im
  else
    "0"
let of_string z = failwith "Complex.of_string not implemented yet!"
end

```

N.2.1 Sparse Representation

If the numbers are very likely to be either purely real or imaginary, a different representation can reduce the load from the floating point unit.

```

module Sparse =
struct
  module C = Dense

  type t =
    | Real of float
    | Imag of float
    | Complex of C.t

  let null = Real 0.0
  let one = Real 1.0

  let real = function
    | Real x → x
    | Imag y → 0.0
    | Complex z → C.real z

  let imag = function
    | Real x → 0.0
    | Imag y → y
    | Complex z → C.imag z
end

```

N.2.2 Suggesting A Default

There's no real choice here (yet) ...

```

module Default = Dense

```

—O—

ALGEBRA

O.1 Interface of Algebra

O.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
```

O.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
```


0.1.3 Expressions: Terms, Rings and Linear Combinations

The tensor algebra will be spanned by an abelian monoid:

module type *Term* =

```
sig
  type  $\alpha$  t
  val unit : unit  $\rightarrow$   $\alpha$  t
  val is_unit :  $\alpha$  t  $\rightarrow$  bool
  val atom :  $\alpha \rightarrow \alpha$  t
  val power : int  $\rightarrow$   $\alpha$  t  $\rightarrow$   $\alpha$  t
  val mul :  $\alpha$  t  $\rightarrow$   $\alpha$  t  $\rightarrow$   $\alpha$  t
  val map : ( $\alpha \rightarrow \beta$ )  $\rightarrow$   $\alpha$  t  $\rightarrow$   $\beta$  t
  val to_string : ( $\alpha \rightarrow$  string)  $\rightarrow$   $\alpha$  t  $\rightarrow$  string
```

The derivative of a term is *not* a term, but a sum of terms instead:

$$D(f_1^{p_1} f_2^{p_2} \dots f_n^{p_n}) = \sum_i (Df_i) p_i f_1^{p_1} f_2^{p_2} \dots f_i^{p_i-1} \dots f_n^{p_n} \quad (\text{O.1})$$

The function returns the sum as a list of triples $(Df_i, p_i, f_1^{p_1} f_2^{p_2} \dots f_i^{p_i-1} \dots f_n^{p_n})$. 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 : ( $\alpha \rightarrow \beta$  option)  $\rightarrow$   $\alpha$  t  $\rightarrow$  ( $\beta \times$  int  $\times$   $\alpha$  t) list
```

convenience function

```
val product :  $\alpha$  t list  $\rightarrow$   $\alpha$  t
val atoms :  $\alpha$  t  $\rightarrow$   $\alpha$  list
```

end

module type *Ring* =

```
sig
  module C : Rational
  type  $\alpha$  t
  val null : unit  $\rightarrow$   $\alpha$  t
  val unit : unit  $\rightarrow$   $\alpha$  t
  val is_null :  $\alpha$  t  $\rightarrow$  bool
  val is_unit :  $\alpha$  t  $\rightarrow$  bool
  val atom :  $\alpha \rightarrow \alpha$  t
  val scale : C.t  $\rightarrow$   $\alpha$  t  $\rightarrow$   $\alpha$  t
  val add :  $\alpha$  t  $\rightarrow$   $\alpha$  t  $\rightarrow$   $\alpha$  t
  val sub :  $\alpha$  t  $\rightarrow$   $\alpha$  t  $\rightarrow$   $\alpha$  t
  val mul :  $\alpha$  t  $\rightarrow$   $\alpha$  t  $\rightarrow$   $\alpha$  t
  val neg :  $\alpha$  t  $\rightarrow$   $\alpha$  t
```

Again

$$D(f_1^{p_1} f_2^{p_2} \dots f_n^{p_n}) = \sum_i (Df_i) p_i f_1^{p_1} f_2^{p_2} \dots f_i^{p_i-1} \dots f_n^{p_n} \quad (\text{O.2})$$

but, iff Df_i can be identified with a f' , we know how to perform the sum.

```
val derive_inner : ( $\alpha \rightarrow \alpha t$ )  $\rightarrow \alpha t \rightarrow \alpha t$  (* this? *)
val derive_inner' : ( $\alpha \rightarrow \alpha t option$ )  $\rightarrow \alpha t \rightarrow \alpha t$  (* 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 α t :

```
val derive_outer : ( $\alpha \rightarrow \beta option$ )  $\rightarrow \alpha t \rightarrow (\beta \times \alpha t) list$ 
```

convenience functions

```
val sum :  $\alpha t list \rightarrow \alpha t$ 
```

```
val product :  $\alpha t list \rightarrow \alpha t$ 
```

The list of all generators appearing in an expression:

```
val atoms :  $\alpha t \rightarrow \alpha list$ 
```

```
val to_string : ( $\alpha \rightarrow string$ )  $\rightarrow \alpha t \rightarrow string$ 
```

end

module type *Linear* =

sig

module *C* : *Ring*

type (α, γ) *t*

val *null* : *unit* $\rightarrow (\alpha, \gamma) t$

val *atom* : $\alpha \rightarrow (\alpha, \gamma) t$

val *singleton* : $\gamma C.t \rightarrow \alpha \rightarrow (\alpha, \gamma) t$

val *scale* : $\gamma C.t \rightarrow (\alpha, \gamma) t \rightarrow (\alpha, \gamma) t$

val *add* : (α, γ) *t* $\rightarrow (\alpha, \gamma) t \rightarrow (\alpha, \gamma) t$

val *sub* : (α, γ) *t* $\rightarrow (\alpha, \gamma) t \rightarrow (\alpha, \gamma) t$

A partial derivative w.r.t. a vector maps from a coefficient ring to the dual vector space.

```
val partial : ( $\gamma \rightarrow (\alpha, \gamma) t$ )  $\rightarrow \gamma C.t \rightarrow (\alpha, \gamma) t$ 
```

A linear combination of vectors

$$linear[(v_1, c_1); (v_2, c_2); \dots; (v_n, c_n)] = \sum_{i=1}^n c_i \cdot v_i \quad (O.3)$$

```
val linear : (( $\alpha, \gamma$ ) t  $\times \gamma C.t$ ) list  $\rightarrow (\alpha, \gamma) t$ 
```

Some convenience functions

```
val map : ( $\alpha \rightarrow \gamma C.t \rightarrow (\beta, \delta) t$ )  $\rightarrow (\alpha, \gamma) t \rightarrow (\beta, \delta) t$ 
```

```
val sum : ( $\alpha, \gamma$ ) t list  $\rightarrow (\alpha, \gamma) t$ 
```

The list of all generators and the list of all generators of coefficients appearing in an expression:

```
val atoms : ( $\alpha, \gamma$ ) t  $\rightarrow \alpha list \times \gamma list$ 
```

```
val to_string : ( $\alpha \rightarrow string$ )  $\rightarrow (\gamma \rightarrow string) \rightarrow (\alpha, \gamma) t \rightarrow string$ 
```

end

module *Term* : *Term*

module *Make_Ring* (*C* : *Rational*) (*T* : *Term*) : *Ring*

module *Make_Linear* (*C* : *Ring*) : *Linear* with module *C* = *C*

O.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
```

O.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
```

O.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
end

```

O.2.3 Expressions: Terms, Rings and Linear Combinations

The tensor algebra will be spanned by an abelian monoid:

```

module type Term =
  sig
    type  $\alpha$  t
    val unit : unit →  $\alpha$  t
    val is_unit :  $\alpha$  t → bool
    val atom :  $\alpha$  →  $\alpha$  t
    val power : int →  $\alpha$  t →  $\alpha$  t
    val mul :  $\alpha$  t →  $\alpha$  t →  $\alpha$  t
    val map : ( $\alpha$  →  $\beta$ ) →  $\alpha$  t →  $\beta$  t
    val to_string : ( $\alpha$  → string) →  $\alpha$  t → string
    val derive : ( $\alpha$  →  $\beta$  option) →  $\alpha$  t → ( $\beta$  × int ×  $\alpha$  t) list
    val product :  $\alpha$  t list →  $\alpha$  t
    val atoms :  $\alpha$  t →  $\alpha$  list
  end

module type Ring =
  sig
    module C : Rational
    type  $\alpha$  t
  end

```

```

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 too 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

```

—P—

SIMPLE LINEAR ALGEBRA

P.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
```

P.2 Implementation of Linalg

This is not a functional implementations, but uses imperative array in Fortran style for maximum 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

```

P.2.1 LU Decomposition

$$A = LU \quad (\text{P.1a})$$

In more detail

$$\begin{pmatrix} a_{00} & a_{01} & \dots & a_{0(n-1)} \\ a_{10} & a_{11} & \dots & a_{1(n-1)} \\ \vdots & \vdots & \ddots & \vdots \\ a_{(n-1)0} & a_{(n-1)1} & \dots & a_{(n-1)(n-1)} \end{pmatrix} =
 \begin{pmatrix} 1 & 0 & \dots & 0 \\ l_{10} & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{(n-1)0} & l_{(n-1)1} & \dots & 1 \end{pmatrix}
 \begin{pmatrix} u_{00} & u_{01} & \dots & u_{0(n-1)} \\ 0 & u_{11} & \dots & u_{1(n-1)} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & u_{(n-1)(n-1)} \end{pmatrix} \quad (\text{P.1b})$$

Rewriting (P.1) in block matrix notation

$$\begin{pmatrix} a_{00} & a_{0\cdot} \\ a_{\cdot 0} & A \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ l_{\cdot 0} & L \end{pmatrix} \begin{pmatrix} u_{00} & u_{0\cdot} \\ 0 & U \end{pmatrix} = \begin{pmatrix} u_{00} & u_{0\cdot} \\ l_{\cdot 0} u_{00} & l_{\cdot 0} \otimes u_{0\cdot} + LU \end{pmatrix} \quad (\text{P.2})$$

we can solve it easily

$$u_{00} = a_{00} \quad (\text{P.3a})$$

$$u_{0\cdot} = a_{0\cdot} \quad (\text{P.3b})$$

$$l_{\cdot 0} = \frac{a_{\cdot 0}}{a_{00}} \quad (\text{P.3c})$$

$$LU = A - \frac{a_{\cdot 0} \otimes a_{0\cdot}}{a_{00}} \quad (\text{P.3d})$$

and (P.3c) and (P.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
```

—Q—
TALK TO THE WHIZARD ...

Talk to [\[11\]](#).



Temporarily disabled, until, we implement some conditional weaving...

—R—

WIDGET LIBRARY AND CLASS HIERARCHY FOR O’GIGA

NB: The code in this chapter must be compiled with `-labels`, since `lablgtk` doesn’t appear to work in classic mode.



Keep in mind that `ocamlweb` doesn’t work properly with O’Caml 3 yet. The colons in label declarations are typeset with erroneous white space.

R.1 Architecture

In `lablgtk`, O’Caml objects are typically constructed in parallel to constructors for GTK+ widgets. The objects provide inheritance and all that, while the constructors implement the semantics.

R.1.1 Inheritance vs. Aggregation

We have two mechanisms for creating new widgets: inheritance and aggregation. Inheritance makes it easy to extend a given widget with new methods or to combine orthogonal widgets (*multiple inheritance*). Aggregation is more suitable for combining non-orthogonal widgets (e.g. multiple instances of the same widget).

The problem with inheritance in `lablgtk` is, that it is a *bad* idea to implement the semantics in the objects. In a multi-level inheritance hierarchy, O’Caml can evaluate class functions more than once. Since functions accessing GTK+ change the state of GTK+, we could accidentally violate invariants. Therefore inheritance forces us to use the two-tiered approach of `lablgtk` ourselves. It is not really complicated, but tedious and it appears to be a good idea to use aggregation whenever in doubt.

Nevertheless, there are examples (like `ThoGButton.mutable_button` below, where just one new method is added), that cry out for inheritance for the benefit of the application developer.




Interface `thoGWindow.mli` unavailable!





Implementation `thoGWindow.ml` unavailable!





Interface `thoGButton.mli` unavailable!

 *Implementation `thoGButton.ml` unavailable!*

 *Interface `thoGMenu.mli` unavailable!*

 *Implementation `thoGMenu.ml` unavailable!*

 *Interface `thoGDraw.mli` unavailable!*

 *Implementation `thoGDraw.ml` unavailable!*

—S—
O’MEGA VIRTUAL MACHINE



Interface oVM.mli unavailable!



Implementation oVM.ml unavailable!

—T—

FORTRAN LIBRARIES

T.1 Trivia

```
<omega_spinors.f90>≡  
  <Cotypeleft>  
  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_2009_06_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>≡
```

T.1.1 Inner Product

```
<Declaration of operations for spinors>≡
```

```

interface operator (*)
  module procedure conjspinor_spinor
end interface
private :: conjspinor_spinor

```

$$\bar{\psi}\psi' \quad (T.1)$$

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

```

T.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

```

T.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 (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 (dot_product (psibar%a, psibar%a))
end function abs_conjspinor

```

T.2 Spinors Revisited

```

<omega_bispinors.f90>≡
<Copleft>
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_2009_06_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

```

$$\bar{\psi}\psi' \quad (T.2)$$

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

```

T.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

```

T.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 (dot_product (psi%a, psi%a))
end function abs_bispinor

```

T.3 Vectorspinors

```

<omega_vectorspinors.f90>≡
<Copleft>
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_2009_06_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

```

$$\bar{\psi}^{\mu}\psi'_{\mu} \quad (T.3)$$

(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

```

T.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

```

```

    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

```

T.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 (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

```

T.4 Vectors and Tensors

Condensed representation of antisymmetric rank-2 tensors:

$$\begin{pmatrix} T^{00} & T^{01} & T^{02} & T^{03} \\ T^{10} & T^{11} & T^{12} & T^{13} \\ T^{20} & T^{21} & T^{22} & T^{23} \\ T^{30} & T^{31} & T^{32} & T^{33} \end{pmatrix} = \begin{pmatrix} 0 & T_e^1 & T_e^2 & T_e^3 \\ -T_e^1 & 0 & T_b^3 & -T_b^2 \\ -T_e^2 & -T_b^3 & 0 & T_b^1 \\ -T_e^3 & T_b^2 & -T_b^1 & 0 \end{pmatrix} \quad (T.4)$$

```

<omega_vectors.f90>=
<Cotypeleft>
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_2009_06_A = 0
contains
    <Implementation of operations for vectors>
end module omega_vectors

```

T.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

```

```

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

```

T.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:

$$\frac{1}{2}T_{\mu\nu}U^{\mu\nu} = \frac{1}{2}(-T^{0i}U^{0i} - T^{i0}U^{i0} + T^{ij}U^{ij}) = T_b^k U_b^k - T_e^k U_e^k \quad (\text{T.5})$$

```

<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

```

T.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

```

$$y^\nu = x_\mu T^{\mu\nu} : y^0 = -x^i T^{i0} = x^i T^{0i} \quad (\text{T.6a})$$

$$y^1 = x^0 T^{01} - x^2 T^{21} - x^3 T^{31} \quad (\text{T.6b})$$

$$y^2 = x^0 T^{02} - x^1 T^{12} - x^3 T^{32} \quad (\text{T.6c})$$

$$y^3 = x^0 T^{03} - x^1 T^{13} - x^2 T^{23} \quad (\text{T.6d})$$

```

<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

```

$$y^\mu = T^{\mu\nu} x_\nu : y^0 = -T^{0i} x^i \quad (\text{T.7a})$$

$$y^1 = T^{10} x^0 - T^{12} x^2 - T^{13} x^3 \quad (\text{T.7b})$$

$$y^2 = T^{20} x^0 - T^{21} x^1 - T^{23} x^3 \quad (\text{T.7c})$$

$$y^3 = T^{30} x^0 - T^{31} x^1 - T^{32} x^2 \quad (\text{T.7d})$$

```

<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

```

T.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

```

T.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

```

T.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 (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 (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 (dot_product (x%e, x%e) + dot_product (x%b, x%b))
end function abs_tensor2odd

```

T.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

```

T.4.8 ϵ -Tensors

$$\epsilon_{0123} = 1 = -\epsilon^{0123} \quad (\text{T.8})$$

in particular

$$\epsilon(p_1, p_2, p_3, p_4) = \epsilon_{\mu_1 \mu_2 \mu_3 \mu_4} p_1^{\mu_1} p_2^{\mu_2} p_3^{\mu_3} p_4^{\mu_4} = p_1^0 p_2^1 p_3^2 p_4^3 \pm \dots \quad (\text{T.9})$$

(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

```

$$\epsilon_{\mu}(p_1, p_2, p_3) = \epsilon_{\mu\mu_1\mu_2\mu_3} p_1^{\mu_1} p_2^{\mu_2} p_3^{\mu_3} \quad (\text{T.10})$$

i. e.

$$\epsilon_0(p_1, p_2, p_3) = p_1^1 p_2^2 p_3^3 \pm \dots \quad (\text{T.11a})$$

$$\epsilon_1(p_1, p_2, p_3) = p_1^2 p_2^3 p_3^0 \pm \dots \quad (\text{T.11b})$$

$$\epsilon_2(p_1, p_2, p_3) = -p_1^3 p_2^0 p_3^1 \pm \dots \quad (\text{T.11c})$$

$$\epsilon_3(p_1, p_2, p_3) = p_1^0 p_2^1 p_3^2 \pm \dots \quad (\text{T.11d})$$

```

<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

```

T.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

```

T.5 Polarization vectors

<omega_polarizations.f90>≡

<Copleft>

```

module omega_polarizations
  use kinds
  use constants
  use omega_vectors
  implicit none
  private
  <Declaration of polarization vectors>
  integer, parameter, public :: omega_polarizations_2009_06_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.

$$\epsilon_1^\mu(k) = \frac{1}{|\vec{k}| \sqrt{k_x^2 + k_y^2}} (0; k_z k_x, k_y k_z, -k_x^2 - k_y^2) \quad (\text{T.12a})$$

$$\epsilon_2^\mu(k) = \frac{1}{\sqrt{k_x^2 + k_y^2}} (0; -k_y, k_x, 0) \quad (\text{T.12b})$$

$$\epsilon_3^\mu(k) = \frac{k_0}{m|\vec{k}|} \left(\vec{k}^2/k_0; k_x, k_y, k_z \right) \quad (\text{T.12c})$$

and

$$\epsilon_\pm^\mu(k) = \frac{1}{\sqrt{2}} (\epsilon_1^\mu(k) \pm i\epsilon_2^\mu(k)) \quad (\text{T.13a})$$

$$\epsilon_0^\mu(k) = \epsilon_3^\mu(k) \quad (\text{T.13b})$$

i. e.

$$\epsilon_+^\mu(k) = \frac{1}{\sqrt{2}\sqrt{k_x^2 + k_y^2}} \left(0; \frac{k_z k_x}{|\vec{k}|} - ik_y, \frac{k_y k_z}{|\vec{k}|} + ik_x, -\frac{k_x^2 + k_y^2}{|\vec{k}|} \right) \quad (\text{T.14a})$$

$$\epsilon_-^\mu(k) = \frac{1}{\sqrt{2}\sqrt{k_x^2 + k_y^2}} \left(0; \frac{k_z k_x}{|\vec{k}|} + ik_y, \frac{k_y k_z}{|\vec{k}|} - ik_x, -\frac{k_x^2 + k_y^2}{|\vec{k}|} \right) \quad (\text{T.14b})$$

$$\epsilon_0^\mu(k) = \frac{k_0}{m|\vec{k}|} \left(\vec{k}^2/k_0; k_x, k_y, k_z \right) \quad (\text{T.14c})$$

Determining the mass from the momenta is a numerically haphazardous for light particles. Therefore, we accept some redundancy and pass the mass explicitly.

(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 if

```

```

end function eps
!!! OLD VERSION !!!!!
!!! 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
!!!   integer, parameter :: x = 2, y = 3, z = 1
!!!   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(x)**2 + k%x(y)**2)
!!!       e%x(x) = cmplx ( k%x(z)*k%x(x)/kabs, &
!!!         - k%x(y), kind=default) / kt / sqrt2
!!!       e%x(y) = cmplx ( k%x(y)*k%x(z)/kabs, &
!!!         k%x(x), kind=default) / kt / sqrt2
!!!       e%x(z) = - kt / kabs / sqrt2
!!!     case (-1)
!!!       kt = sqrt (k%x(x)**2 + k%x(y)**2)
!!!       e%x(x) = cmplx ( k%x(z)*k%x(x)/kabs, &
!!!         k%x(y), kind=default) / kt / sqrt2
!!!       e%x(y) = cmplx ( k%x(y)*k%x(z)/kabs, &
!!!         - k%x(x), kind=default) / kt / sqrt2
!!!       e%x(z) = - kt / kabs / sqrt2
!!!     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
!!!     select case (s)
!!!     case (1)
!!!       e%x(x) = cmplx ( 1, 0, kind=default) / sqrt2
!!!       e%x(y) = cmplx ( 0, 1, kind=default) / sqrt2
!!!     case (-1)
!!!       e%x(x) = cmplx ( 1, 0, kind=default) / sqrt2
!!!       e%x(y) = cmplx ( 0, - 1, kind=default) / sqrt2
!!!     case (0)
!!!       if (m > 0) then

```

```

!!!          e%x(z) = 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
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

```

T.6 Polarization vectors revisited

```

(omega_polarizations_madgraph.f90)≡
<Cotypeleft>
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_2009_06_A = 0
contains
  <Implementation of polarization vectors for madgraph>
end module omega_polarizations_madgraph

```

This set of polarization vectors is compatible with HELAS [5]:

$$\epsilon_1^\mu(k) = \frac{1}{|\vec{k}|\sqrt{k_x^2 + k_y^2}} (0; k_z k_x, k_y k_z, -k_x^2 - k_y^2) \quad (\text{T.15a})$$

$$\epsilon_2^\mu(k) = \frac{1}{\sqrt{k_x^2 + k_y^2}} (0; -k_y, k_x, 0) \quad (\text{T.15b})$$

$$\epsilon_3^\mu(k) = \frac{k_0}{m|\vec{k}|} (\vec{k}^2/k_0; k_x, k_y, k_z) \quad (\text{T.15c})$$

and

$$\epsilon_\pm^\mu(k) = \frac{1}{\sqrt{2}} (\mp \epsilon_1^\mu(k) - i \epsilon_2^\mu(k)) \quad (\text{T.16a})$$

$$\epsilon_0^\mu(k) = \epsilon_3^\mu(k) \quad (\text{T.16b})$$

i. e.

$$\epsilon_+^\mu(k) = \frac{1}{\sqrt{2}\sqrt{k_x^2 + k_y^2}} \left(0; -\frac{k_z k_x}{|\vec{k}|} + i k_y, -\frac{k_y k_z}{|\vec{k}|} - i k_x, \frac{k_x^2 + k_y^2}{|\vec{k}|} \right) \quad (\text{T.17a})$$

$$\epsilon_-^\mu(k) = \frac{1}{\sqrt{2}\sqrt{k_x^2 + k_y^2}} \left(0; \frac{k_z k_x}{|\vec{k}|} + i k_y, \frac{k_y k_z}{|\vec{k}|} - i k_x, -\frac{k_x^2 + k_y^2}{|\vec{k}|} \right) \quad (\text{T.17b})$$

$$\epsilon_0^\mu(k) = \frac{k_0}{m|\vec{k}|} \left(\vec{k}^2/k_0; k_x, k_y, k_z \right) \quad (\text{T.17c})$$

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 explicitly.

(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

```

T.7 Symmetric Tensors

Spin-2 polarization tensors are symmetric, transversal and traceless

$$\epsilon_m^{\mu\nu}(k) = \epsilon_m^{\nu\mu}(k) \quad (\text{T.18a})$$

$$k_\mu \epsilon_m^{\mu\nu}(k) = k_\nu \epsilon_m^{\mu\nu}(k) = 0 \quad (\text{T.18b})$$

$$\epsilon_{m,\mu}^\mu(k) = 0 \quad (\text{T.18c})$$

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_2009_06_A = 0
contains
  <Implementation of operations for tensors>
end module omega_tensors

```

T.7.1 Vector Space

Scalar Multiplication

```

<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

```

T.8 Symmetric Polarization Tensors

$$\epsilon_{+2}^{\mu\nu}(k) = \epsilon_{+}^{\mu}(k)\epsilon_{+}^{\nu}(k) \quad (\text{T.19a})$$

$$\epsilon_{+1}^{\mu\nu}(k) = \frac{1}{\sqrt{2}} (\epsilon_+^\mu(k)\epsilon_0^\nu(k) + \epsilon_0^\mu(k)\epsilon_+^\nu(k)) \quad (\text{T.19b})$$

$$\epsilon_0^{\mu\nu}(k) = \frac{1}{\sqrt{6}} (\epsilon_+^\mu(k)\epsilon_-^\nu(k) + \epsilon_-^\mu(k)\epsilon_+^\nu(k) - 2\epsilon_0^\mu(k)\epsilon_0^\nu(k)) \quad (\text{T.19c})$$

$$\epsilon_{-1}^{\mu\nu}(k) = \frac{1}{\sqrt{2}} (\epsilon_-^\mu(k)\epsilon_0^\nu(k) + \epsilon_0^\mu(k)\epsilon_-^\nu(k)) \quad (\text{T.19d})$$

$$\epsilon_{-2}^{\mu\nu}(k) = \epsilon_-^\mu(k)\epsilon_-^\nu(k) \quad (\text{T.19e})$$

Note that $\epsilon_{\pm 2, \mu}^\mu(k) = \epsilon_\pm^\mu(k)\epsilon_{\pm, \mu}(k) \propto \epsilon_\pm^\mu(k)\epsilon_{\mp, \mu}^*(k) = 0$ and that the sign in $\epsilon_0^{\mu\nu}(k)$ insures its tracelessness¹.

```

⟨omega_tensor_polarizations.f90⟩≡
⟨Cotypeleft⟩
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_2009_06_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)

```

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 $\epsilon_{-,0,+}^\mu$ with $|-,0,+\rangle$.

```

    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)
    em = eps (m, k, -1)
    t = em.tprod.em
end select
end function eps2

```

T.9 Couplings

```

(omega_couplings.f90)≡
  <Cotypeleft>
  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_2009_06_A = 0
  contains
    <Implementation of couplings>
    <Implementation of propagators>
  end module omega_couplings

  <Declaration of propagators>≡
    public :: wd_tl

  <Declaration of propagators>+≡
    public :: gauss

```

$$\Theta(p^2)\Gamma \quad (\text{T.20})$$

```

  <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

```

$$\frac{i}{p^2 - m^2 + im\Gamma} \phi \quad (\text{T.21})$$

```

<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 / cplx (p*p - m**2, m*w, kind=default)) * phi
end function pr_phi

```

$$\sqrt{\frac{\pi}{MT}} \phi \quad (\text{T.22})$$

```

<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

```

$$\frac{i}{p^2 - m^2 + im\Gamma} \left(-g_{\mu\nu} + \frac{p_\mu p_\nu}{m^2} \right) \epsilon^\nu(p) \quad (\text{T.23})$$

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
```

$$\sqrt{\frac{\pi}{M\Gamma}} \left(-g_{\mu\nu} + \frac{p_\mu p_\nu}{m^2} \right) \epsilon^\nu(p) \quad (\text{T.24})$$

(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
```

$$\frac{-i}{p^2} \epsilon^\nu(p) \quad (\text{T.25})$$

(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
```

$$\frac{i}{p^2} \left(-g_{\mu\nu} + (1 - \xi) \frac{p_\mu p_\nu}{p^2} \right) \epsilon^\nu(p) \quad (\text{T.26})$$

(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

```

$$\frac{i}{p^2 - m^2 + im\Gamma} \left(-g_{\mu\nu} + (1 - \xi) \frac{p_\mu p_\nu}{p^2 - \xi m^2} \right) \epsilon^\nu(p) \quad (\text{T.27})$$

(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

```

$$\frac{iP^{\mu\nu,\rho\sigma}(p, m)}{p^2 - m^2 + im\Gamma} T_{\rho\sigma} \quad (\text{T.28a})$$

with

$$P^{\mu\nu,\rho\sigma}(p, m) = \frac{1}{2} \left(g^{\mu\rho} - \frac{p^\mu p^\rho}{m^2} \right) \left(g^{\nu\sigma} - \frac{p^\nu p^\sigma}{m^2} \right) + \frac{1}{2} \left(g^{\mu\sigma} - \frac{p^\mu p^\sigma}{m^2} \right) \left(g^{\nu\rho} - \frac{p^\nu p^\rho}{m^2} \right) - \frac{1}{3} \left(g^{\mu\nu} - \frac{p^\mu p^\nu}{m^2} \right) \left(g^{\rho\sigma} - \frac{p^\rho p^\sigma}{m^2} \right) \quad (\text{T.28b})$$

Be careful with raising and lowering of indices:

$$g^{\mu\nu} - \frac{k^\mu k^\nu}{m^2} = \begin{pmatrix} 1 - k^0 k^0 / m^2 & -k^0 \vec{k} / m^2 \\ -\vec{k} k^0 / m^2 & -\mathbf{1} - \vec{k} \otimes \vec{k} / m^2 \end{pmatrix} \quad (\text{T.29a})$$

$$g^\mu{}_\nu - \frac{k^\mu k_\nu}{m^2} = \begin{pmatrix} 1 - k^0 k^0 / m^2 & k^0 \vec{k} / m^2 \\ -\vec{k} k^0 / m^2 & \mathbf{1} + \vec{k} \otimes \vec{k} / m^2 \end{pmatrix} \quad (\text{T.29b})$$

(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

```

T.9.1 Triple Gauge Couplings

(Declaration of couplings)≡

```
public :: g_gg
```

According to (??)

$$\begin{aligned}
A^{a,\mu}(k_1 + k_2) = & -ig((k_1^\mu - k_2^\mu)A^{a_1}(k_1) \cdot A^{a_2}(k_2) \\
& + (2k_2 + k_1) \cdot A^{a_1}(k_1)A^{a_2,\mu}(k_2) - A^{a_1,\mu}(k_1)A^{a_2}(k_2) \cdot (2k_1 + k_2)) \quad (\text{T.30})
\end{aligned}$$

(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

```

T.9.2 Quadruple Gauge Couplings

(Declaration of couplings)+≡

```
public :: x_gg, g_gx
```

$$T^{a,\mu\nu}(k_1 + k_2) = g(A^{a_1,\mu}(k_1)A^{a_2,\nu}(k_2) - A^{a_1,\nu}(k_1)A^{a_2,\mu}(k_2)) \quad (\text{T.31})$$

(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
```

$$A^{a,\mu}(k_1 + k_2) = gA_\nu^{a_1}(k_1)T^{a_2,\nu\mu}(k_2) \quad (\text{T.32})$$

(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
```

T.9.3 Scalar Current

(Declaration of couplings)+≡

```
public :: v_ss, s_vs
```

$$V^\mu(k_1 + k_2) = g(k_1^\mu - k_2^\mu)\phi_1(k_1)\phi_2(k_2) \quad (\text{T.33})$$

(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
```

$$\phi(k_1 + k_2) = g(k_1^\mu + 2k_2^\mu)V_\mu(k_1)\phi(k_2) \quad (\text{T.34})$$

(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
```

T.9.4 Triple Vector Couplings

(Declaration of couplings)+≡

```
public :: tkv_vv, lkv_vv, tv_kv, lv_kv, kg_kgkg
public :: t5kv_vv, l5kv_vv, t5v_kv, l5v_kv, kg5_kgkg, kg_kg5kg
```

$$V^\mu(k_1 + k_2) = ig(k_1 - k_2)^\mu V_1^\nu(k_1) V_{2,\nu}(k_2) \quad (\text{T.35})$$

(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
```

$$V^\mu(k_1 + k_2) = ig\epsilon^{\mu\nu\rho\sigma}(k_1 - k_2)_\nu V_{1,\rho}(k_1) V_{2,\sigma}(k_2) \quad (\text{T.36})$$

(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
```

$$V^\mu(k_1 + k_2) = ig(k_1 + k_2)^\mu V_1^\nu(k_1) V_{2,\nu}(k_2) \quad (\text{T.37})$$

(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
```

$$V^\mu(k_1 + k_2) = ig\epsilon^{\mu\nu\rho\sigma}(k_1 + k_2)_\nu V_{1,\rho}(k_1) V_{2,\sigma}(k_2) \quad (\text{T.38})$$

(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
```

$$V^\mu(k_1 + k_2) = ig(k_2 - k)^\nu V_{1,\nu}(k_1) V_2^\mu(k_2) = ig(2k_2 + k_1)^\nu V_{1,\nu}(k_1) V_2^\mu(k_2) \quad (\text{T.39})$$

using $k = -k_1 - k_2$

(Implementation of couplings) +=

```
pure function tv_kv (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_kv
```

$$V^\mu(k_1 + k_2) = ig\epsilon^{\mu\nu\rho\sigma}(2k_2 + k_1)_\nu V_{1,\rho}(k_1) V_{2,\sigma}(k_2) \quad (\text{T.40})$$

(Implementation of couplings) +=

```
pure function t5v_kv (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_kv
```

$$V^\mu(k_1 + k_2) = -igk_1^\nu V_{1,\nu}(k_1) V_2^\mu(k_2) \quad (\text{T.41})$$

using $k = -k_1 - k_2$

(Implementation of couplings) +=

```
pure function lv_kv (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_kv
```

$$V^\mu(k_1 + k_2) = -ig\epsilon^{\mu\nu\rho\sigma} k_{1,\nu} V_{1,\rho}(k_1) V_{2,\sigma}(k_2) \quad (\text{T.42})$$

(Implementation of couplings) +=

```
pure function l5v_kv (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_kv
```

$$A^\mu(k_1 + k_2) = igk^\nu \left(F_{1,\nu}{}^\rho(k_1)F_{2,\rho\mu}(k_2) - F_{1,\mu}{}^\rho(k_1)F_{2,\rho\nu}(k_2) \right) \quad (\text{T.43})$$

with $k = -k_1 - k_2$, i. e.

$$\begin{aligned} A^\mu(k_1 + k_2) = -ig \Big(& [(kk_2)(k_1A_2) - (k_1k_2)(kA_2)]A_1^\mu \\ & + [(k_1k_2)(kA_1) - (kk_1)(k_2A_1)]A_2^\mu \\ & + [(k_2A_1)(kA_2) - (kk_2)(A_1A_2)]k_1^\mu \\ & + [(kk_1)(A_1A_2) - (kA_1)(k_1A_2)]k_2^\mu \Big) \quad (\text{T.44}) \end{aligned}$$

(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
```

$$A^\mu(k_1 + k_2) = ig\epsilon^{\mu\nu\rho\sigma}k_\nu F_{1,\rho}{}^\lambda(k_1)F_{2,\lambda\sigma}(k_2) \quad (\text{T.45})$$

with $k = -k_1 - k_2$, i. e.

$$\begin{aligned} A^\mu(k_1 + k_2) = -2ig\epsilon^{\mu\nu\rho\sigma}k_\nu \Big(& (k_2A_1)k_{1,\rho}A_{2,\sigma} + (k_1A_2)A_{1,\rho}k_{2,\sigma} \\ & - (A_1A_2)k_{1,\rho}k_{2,\sigma} - (k_1k_2)A_{1,\rho}A_{2,\sigma} \Big) \quad (\text{T.46}) \end{aligned}$$

(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

```

$$A^\mu(k_1 + k_2) = igk_\nu \left(\epsilon^{\mu\rho\lambda\sigma} F_{1,\rho}{}^\nu - \epsilon^{\nu\rho\lambda\sigma} F_{1,\rho}{}^\mu \right) \frac{1}{2} F_{1,\lambda\sigma} \quad (\text{T.47})$$

with $k = -k_1 - k_2$, i.e.

$$A^\mu(k_1+k_2) = -ig \left(\epsilon^{\mu\rho\lambda\sigma} (k k_2) A_{2,\rho} - \epsilon^{\mu\rho\lambda\sigma} (k A_2) k_{2,\rho} - k_2^\mu \epsilon^{\nu\rho\lambda\sigma} k_n u A_{2,\rho} + A_2^\mu \epsilon^{\nu\rho\lambda\sigma} k_n u k_{2,\rho} \right) k_{1,\lambda} A_{1,\sigma} \quad (\text{T.48})$$



This is not the most efficient 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

```

T.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%(0,0) - t%(1,1) - t%(2,2) - 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

```

T.11 Tensor Couplings

(Declaration of couplings)+≡
 public :: t2_vv, v_t2v

T.12 Scalar-Vector Dim-5 Couplings

(Declaration of couplings)+≡
 public :: phi_vv, v_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 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

<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

```

T.13 Spinor Couplings

```

<omega_spinor_couplings.f90>=
<Cotypeleft>
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_cppls_2009_06_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 T.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.

T.13.1 Fermionic Vector and Axial Couplings

There's more than one chiral representation. This one is compatible with HELAS [5].

$$\gamma^0 = \begin{pmatrix} 0 & \mathbf{1} \\ \mathbf{1} & 0 \end{pmatrix}, \gamma^i = \begin{pmatrix} 0 & \sigma^i \\ -\sigma^i & 0 \end{pmatrix}, \gamma_5 = i\gamma^0\gamma^1\gamma^2\gamma^3 = \begin{pmatrix} -\mathbf{1} & 0 \\ 0 & \mathbf{1} \end{pmatrix} \quad (\text{T.49})$$

Therefore

$$g_S + g_P\gamma_5 = \begin{pmatrix} g_S - g_P & 0 & 0 & 0 \\ 0 & g_S - g_P & 0 & 0 \\ 0 & 0 & g_S + g_P & 0 \\ 0 & 0 & 0 & g_S + g_P \end{pmatrix} \quad (\text{T.50a})$$

$$g_V\gamma^0 - g_A\gamma^0\gamma_5 = \begin{pmatrix} 0 & 0 & g_V - g_A & 0 \\ 0 & 0 & 0 & g_V - g_A \\ g_V + g_A & 0 & 0 & 0 \\ 0 & g_V + g_A & 0 & 0 \end{pmatrix} \quad (\text{T.50b})$$

$$g_V\gamma^1 - g_A\gamma^1\gamma_5 = \begin{pmatrix} 0 & 0 & 0 & g_V - g_A \\ 0 & 0 & g_V - g_A & 0 \\ 0 & -g_V - g_A & 0 & 0 \\ -g_V - g_A & 0 & 0 & 0 \end{pmatrix} \quad (\text{T.50c})$$

$$g_V\gamma^2 - g_A\gamma^2\gamma_5 = \begin{pmatrix} 0 & 0 & 0 & -i(g_V - g_A) \\ 0 & 0 & i(g_V - g_A) & 0 \\ 0 & i(g_V + g_A) & 0 & 0 \\ -i(g_V + g_A) & 0 & 0 & 0 \end{pmatrix} \quad (\text{T.50d})$$

$$g_V\gamma^3 - g_A\gamma^3\gamma_5 = \begin{pmatrix} 0 & 0 & g_V - g_A & 0 \\ 0 & 0 & 0 & -g_V + g_A \\ -g_V - g_A & 0 & 0 & 0 \\ 0 & g_V + g_A & 0 & 0 \end{pmatrix} \quad (\text{T.50e})$$

(Declaration of spinor currents)≡

```
public :: va_ff, v_ff, a_ff, vl_ff, vr_ff, vlr_ff, grav_ff, va2_ff
```

(Implementation of spinor 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)
```

$\bar{\psi}(g_V\gamma^\mu - g_A\gamma^\mu\gamma_5)\psi$	<code>va_ff</code> ($g_V, g_A, \bar{\psi}, \psi$)
$g_V\bar{\psi}\gamma^\mu\psi$	<code>v_ff</code> ($g_V, \bar{\psi}, \psi$)
$g_A\bar{\psi}\gamma_5\gamma^\mu\psi$	<code>a_ff</code> ($g_A, \bar{\psi}, \psi$)
$g_L\bar{\psi}\gamma^\mu(1 - \gamma_5)\psi$	<code>vl_ff</code> ($g_L, \bar{\psi}, \psi$)
$g_R\bar{\psi}\gamma^\mu(1 + \gamma_5)\psi$	<code>vr_ff</code> ($g_R, \bar{\psi}, \psi$)
$\bar{\psi}(g_V - g_A\gamma_5)\psi$	<code>f_vaf</code> (g_V, g_A, V, ψ)
$g_V\bar{\psi}\psi$	<code>f_vf</code> (g_V, V, ψ)
$g_A\gamma_5\bar{\psi}\psi$	<code>f_af</code> (g_A, V, ψ)
$g_L\bar{\psi}(1 - \gamma_5)\psi$	<code>f_vlf</code> (g_L, V, ψ)
$g_R\bar{\psi}(1 + \gamma_5)\psi$	<code>f_vrf</code> (g_R, V, ψ)
$\bar{\psi}\bar{V}(g_V - g_A\gamma_5)$	<code>f_fva</code> ($g_V, g_A, \bar{\psi}, V$)
$g_V\bar{\psi}\bar{V}$	<code>f_fv</code> ($g_V, \bar{\psi}, V$)
$g_A\bar{\psi}\gamma_5\bar{V}$	<code>f_fa</code> ($g_A, \bar{\psi}, V$)
$g_L\bar{\psi}\bar{V}(1 - \gamma_5)$	<code>f_fvl</code> ($g_L, \bar{\psi}, V$)
$g_R\bar{\psi}\bar{V}(1 + \gamma_5)$	<code>f_fvr</code> ($g_R, \bar{\psi}, V$)

Table T.1: Mnemonically abbreviated names of Fortran functions implementing fermionic vector and axial currents.

$\bar{\psi}(g_S + g_P\gamma_5)\psi$	<code>sp_ff</code> ($g_S, g_P, \bar{\psi}, \psi$)
$g_S\bar{\psi}\psi$	<code>s_ff</code> ($g_S, \bar{\psi}, \psi$)
$g_P\bar{\psi}\gamma_5\psi$	<code>p_ff</code> ($g_P, \bar{\psi}, \psi$)
$g_L\bar{\psi}(1 - \gamma_5)\psi$	<code>sl_ff</code> ($g_L, \bar{\psi}, \psi$)
$g_R\bar{\psi}(1 + \gamma_5)\psi$	<code>sr_ff</code> ($g_R, \bar{\psi}, \psi$)
$\phi(g_S + g_P\gamma_5)\psi$	<code>f_spf</code> (g_S, g_P, ϕ, ψ)
$g_S\phi\psi$	<code>f_sf</code> (g_S, ϕ, ψ)
$g_P\phi\gamma_5\psi$	<code>f_pf</code> (g_P, ϕ, ψ)
$g_L\phi(1 - \gamma_5)\psi$	<code>f_slf</code> (g_L, ϕ, ψ)
$g_R\phi(1 + \gamma_5)\psi$	<code>f_srf</code> (g_R, ϕ, ψ)
$\bar{\psi}\phi(g_S + g_P\gamma_5)$	<code>f_fsp</code> ($g_S, g_P, \bar{\psi}, \phi$)
$g_S\bar{\psi}\phi$	<code>f_fs</code> ($g_S, \bar{\psi}, \phi$)
$g_P\bar{\psi}\phi\gamma_5$	<code>f_fp</code> ($g_P, \bar{\psi}, \phi$)
$g_L\bar{\psi}\phi(1 - \gamma_5)$	<code>f_fsl</code> ($g_L, \bar{\psi}, \phi$)
$g_R\bar{\psi}\phi(1 + \gamma_5)$	<code>f_fsr</code> ($g_R, \bar{\psi}, \phi$)

Table T.2: Mnemonically abbreviated names of Fortran functions implementing fermionic scalar and pseudo scalar “currents”.

```

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

```

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

```

$$g_L \gamma_\mu (1 - \gamma_5) + g_R \gamma_\mu (1 + \gamma_5) = (g_L + g_R) \gamma_\mu - (g_L - g_R) \gamma_\mu \gamma_5 = g_V \gamma_\mu - g_A \gamma_\mu \gamma_5 \quad (\text{T.51})$$

... 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

$$\not{p} - \not{p} \gamma_5 = \begin{pmatrix} 0 & 0 & v_- - a_- & -v^* + a^* \\ 0 & 0 & -v + a & v_+ - a_+ \\ v_+ + a_+ & v^* + a^* & 0 & 0 \\ v + a & v_- + a_- & 0 & 0 \end{pmatrix} \quad (\text{T.52})$$

with $v_\pm = v_0 \pm v_3$, $a_\pm = a_0 \pm a_3$, $v = v_1 + iv_2$, $v^* = v_1 - iv_2$, $a = a_1 + ia_2$, and $a^* = a_1 - ia_2$. 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

<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_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

<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_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

```

T.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

```

$$g_L(1 - \gamma_5) + g_R(1 + \gamma_5) = (g_R + g_L) + (g_R - g_L)\gamma_5 = g_S + g_P\gamma_5 \quad (\text{T.53})$$

```

<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

```

T.13.3 On Shell Wave Functions

(Declaration of spinor on shell wave functions)≡

```

public :: u, ubar, v, vbar
private :: chi_plus, chi_minus

```

$$\chi_+(\vec{p}) = \frac{1}{\sqrt{2|\vec{p}|(|\vec{p}| + p_3)}} \begin{pmatrix} |\vec{p}| + p_3 \\ p_1 + ip_2 \end{pmatrix} \quad (\text{T.54a})$$

$$\chi_-(\vec{p}) = \frac{1}{\sqrt{2|\vec{p}|(|\vec{p}| + p_3)}} \begin{pmatrix} -p_1 + ip_2 \\ |\vec{p}| + p_3 \end{pmatrix} \quad (\text{T.54b})$$

(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
    !!! OLD VERSION !!!!!!
    !!! if (1 + p%x(3) / pabs <= epsilon (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
!!! OLD VERSION !!!!!!!!!!!!!
!!! if (1 + p%x(3) / pabs <= epsilon (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

```

$$u_{\pm}(p) = \begin{pmatrix} \sqrt{p_0 \mp |\vec{p}|} \cdot \chi_{\pm}(\vec{p}) \\ \sqrt{p_0 \pm |\vec{p}|} \cdot \chi_{\pm}(\vec{p}) \end{pmatrix} \quad (\text{T.55})$$

Determining the mass from the momenta is a numerically haphazardous for light particles. Therefore, we accept some redundancy and pass the mass explicitly. 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 (m, p, s) result (psi)
  type(spinor) :: psi
  real(kind=default), intent(in) :: m
  type(momentum), intent(in) :: p
  integer, intent(in) :: s
  complex(kind=default), dimension(2) :: chi
  real(kind=default) :: pabs
  pabs = sqrt (dot_product (p%x, p%x))
  select case (s)
  case (1)
    chi = chi_plus (p)
    psi%a(1:2) = sqrt (max (p%t - pabs, 0.0_default)) * 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) = sqrt (max (p%t - pabs, 0.0_default)) * chi
  case default
    pabs = m ! make the compiler happy and use m
    psi%a = 0
  end select
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

```

$$v_{\pm}(p) = \begin{pmatrix} \mp \sqrt{p_0 \pm |\vec{p}|} \cdot \chi_{\mp}(\vec{p}) \\ \pm \sqrt{p_0 \mp |\vec{p}|} \cdot \chi_{\mp}(\vec{p}) \end{pmatrix} \quad (\text{T.56})$$

(Implementation of spinor on shell wave functions)+≡

```

pure function v (m, p, s) result (psi)
  type(spinor) :: psi
  real(kind=default), intent(in) :: m
  type(momentum), intent(in) :: p
  integer, intent(in) :: s
  complex(kind=default), dimension(2) :: chi
  real(kind=default) :: pabs
  pabs = sqrt (dot_product (p%x, p%x))
  select case (s)
  case (1)
    chi = chi_minus (p)
    psi%a(1:2) = - sqrt (p%t + pabs) * chi
    psi%a(3:4) = sqrt (max (p%t - pabs, 0.0_default)) * chi
  case (-1)
    chi = chi_plus (p)
    psi%a(1:2) = sqrt (max (p%t - pabs, 0.0_default)) * 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
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

```

T.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:

$$\langle \text{Out} | d^\dagger | \text{In} \rangle = i \int d^4 x \bar{v} e^{-ikx} (i \not{\partial} - m) \langle \text{Out} | \psi | \text{In} \rangle \quad (\text{T.57a})$$

$$\langle \text{Out} | b | \text{In} \rangle = -i \int d^4 x \bar{u} e^{ikx} (i \not{\partial} - m) \langle \text{Out} | \psi | \text{In} \rangle \quad (\text{T.57b})$$

$$\langle \text{Out} | d | \text{In} \rangle = i \int d^4x \langle \text{Out} | \bar{\psi} | \text{In} \rangle (-i \overleftarrow{\not{\partial}} - m) v e^{ikx} \quad (\text{T.57c})$$

$$\langle \text{Out} | b^\dagger | \text{In} \rangle = -i \int d^4x \langle \text{Out} | \bar{\psi} | \text{In} \rangle (-i \overleftarrow{\not{\partial}} - m) u e^{-ikx} \quad (\text{T.57d})$$

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:

$$brsu(p) = (-i)(\not{p} - m)u(p) \quad (\text{T.58})$$

(Implementation of spinor off shell wave functions) \equiv

```
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=cplx(0.0,-1.0)*(f_vf(one,vp,psi)-m*psi)
end function brs_u
```

$$brsv(p) = i(\not{p} + m)v(p) \quad (\text{T.59})$$

(Implementation of spinor off shell wave functions) $+\equiv$

```
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=cplx(0.0,1.0)*(f_vf(one,vp,psi)+m*psi)
end function brs_v
```

$$brs\bar{u}(p) = (-i)\bar{u}(p)(\not{p} - m) \quad (\text{T.60})$$

(Implementation of spinor off shell wave functions) $+\equiv$

```
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=cplx(0.0,-1.0)*(f_fv(one,psibar,vp)-m*psibar)
end function brs_ubar
```

$$brs\bar{v}(p) = (i)\bar{v}(p)(\not{p} + m) \quad (T.61)$$

(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=cplx(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.

T.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

```

$$\frac{i(-\not{p} + m)}{p^2 - m^2 + im\Gamma}\psi \quad (T.62)$$

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 / cplx (p*p - m**2, m*w, kind=default)) &
    * (- f_vf (one, vp, psi) + m * psi)
end function pr_psi

```

$$\sqrt{\frac{\pi}{M\Gamma}}(-\not{p} + m)\psi \quad (T.63)$$

(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

```

$$\bar{\psi} \frac{i(\not{p} + m)}{p^2 - m^2 + im\Gamma} \quad (\text{T.64})$$

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

```

$$\sqrt{\frac{\pi}{M\Gamma}} \bar{\psi}(\not{p} + m) \quad (\text{T.65})$$

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
pppsibar = gauss (p*p, m, w) * (f_fv (one, psibar, vp) + m * psibar)
end function pg_psibar

```

$$\frac{i(-\not{p} + m)}{p^2 - m^2 + im\Gamma} \sum_n \psi_n \otimes \bar{\psi}_n \quad (\text{T.66})$$

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

```

$$\sum_n \psi_n \otimes \bar{\psi}_n \frac{i(\not{p} + m)}{p^2 - m^2 + im\Gamma} \quad (\text{T.67})$$

```

<(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))
  psipsibar%a(i,:) = psibar(i)%a
end do
end function pr_dyadright

```

T.14 Spinor Couplings Revisited

```

(omega_bispinor_couplings.f90)≡
  <Cotypeleft>
  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_2009_06_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 T.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.

T.14.1 Fermionic Vector and Axial Couplings

There's more than one chiral representation. This one is compatible with HELAS [5].

$$\gamma^0 = \begin{pmatrix} 0 & \mathbf{1} \\ \mathbf{1} & 0 \end{pmatrix}, \quad \gamma^i = \begin{pmatrix} 0 & \sigma^i \\ -\sigma^i & 0 \end{pmatrix}, \quad \gamma_5 = i\gamma^0\gamma^1\gamma^2\gamma^3 = \begin{pmatrix} -\mathbf{1} & 0 \\ 0 & \mathbf{1} \end{pmatrix}, \quad (\text{T.68a})$$

$$C = \begin{pmatrix} \epsilon & 0 \\ 0 & -\epsilon \end{pmatrix}, \quad \epsilon = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}. \quad (\text{T.68b})$$

Therefore

$$g_S + g_P \gamma_5 = \begin{pmatrix} g_S - g_P & 0 & 0 & 0 \\ 0 & g_S - g_P & 0 & 0 \\ 0 & 0 & g_S + g_P & 0 \\ 0 & 0 & 0 & g_S + g_P \end{pmatrix} \quad (\text{T.69a})$$

$$g_V \gamma^0 - g_A \gamma^0 \gamma_5 = \begin{pmatrix} 0 & 0 & g_V - g_A & 0 \\ 0 & 0 & 0 & g_V - g_A \\ g_V + g_A & 0 & 0 & 0 \\ 0 & g_V + g_A & 0 & 0 \end{pmatrix} \quad (\text{T.69b})$$

$$g_V \gamma^1 - g_A \gamma^1 \gamma_5 = \begin{pmatrix} 0 & 0 & 0 & g_V - g_A \\ 0 & 0 & g_V - g_A & 0 \\ 0 & -g_V - g_A & 0 & 0 \\ -g_V - g_A & 0 & 0 & 0 \end{pmatrix} \quad (\text{T.69c})$$

$$g_V \gamma^2 - g_A \gamma^2 \gamma_5 = \begin{pmatrix} 0 & 0 & 0 & -i(g_V - g_A) \\ 0 & 0 & i(g_V - g_A) & 0 \\ 0 & i(g_V + g_A) & 0 & 0 \\ -i(g_V + g_A) & 0 & 0 & 0 \end{pmatrix} \quad (\text{T.69d})$$

$$g_V \gamma^3 - g_A \gamma^3 \gamma_5 = \begin{pmatrix} 0 & 0 & g_V - g_A & 0 \\ 0 & 0 & 0 & -g_V + g_A \\ -g_V - g_A & 0 & 0 & 0 \\ 0 & g_V + g_A & 0 & 0 \end{pmatrix} \quad (\text{T.69e})$$

and

$$C(g_S + g_P \gamma_5) = \begin{pmatrix} 0 & g_S - g_P & 0 & 0 \\ -g_S + g_P & 0 & 0 & 0 \\ 0 & 0 & 0 & -g_S - g_P \\ 0 & 0 & g_S + g_P & 0 \end{pmatrix} \quad (\text{T.70a})$$

$$C(g_V \gamma^0 - g_A \gamma^0 \gamma_5) = \begin{pmatrix} 0 & 0 & 0 & g_V - g_A \\ 0 & 0 & -g_V + g_A & 0 \\ 0 & -g_V - g_A & 0 & 0 \\ g_V + g_A & 0 & 0 & 0 \end{pmatrix} \quad (\text{T.70b})$$

$$C(g_V \gamma^1 - g_A \gamma^1 \gamma_5) = \begin{pmatrix} 0 & 0 & g_V - g_A & 0 \\ 0 & 0 & 0 & -g_V + g_A \\ g_V + g_A & 0 & 0 & 0 \\ 0 & -g_V - g_A & 0 & 0 \end{pmatrix} \quad (\text{T.70c})$$

$$C(g_V \gamma^2 - g_A \gamma^2 \gamma_5) = \begin{pmatrix} 0 & 0 & i(g_V - g_A) & 0 \\ 0 & 0 & 0 & i(g_V - g_A) \\ i(g_V + g_A) & 0 & 0 & 0 \\ 0 & i(g_V + g_A) & 0 & 0 \end{pmatrix} \quad (\text{T.70d})$$

$$C(g_V \gamma^3 - g_A \gamma^3 \gamma_5) = \begin{pmatrix} 0 & 0 & 0 & -g_V + g_A \\ 0 & 0 & -g_V + g_A & 0 \\ 0 & -g_V - g_A & 0 & 0 \\ -g_V - g_A & 0 & 0 & 0 \end{pmatrix} \quad (\text{T.70e})$$

(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

```

$$\not{p} - \not{p}\gamma_5 = \begin{pmatrix} 0 & 0 & v_- - a_- & -v^* + a^* \\ 0 & 0 & -v + a & v_+ - a_+ \\ v_+ + a_+ & v^* + a^* & 0 & 0 \\ v + a & v_- + a_- & 0 & 0 \end{pmatrix} \quad (\text{T.71})$$

with $v_{\pm} = v_0 \pm v_3$, $a_{\pm} = a_0 \pm a_3$, $v = v_1 + iv_2$, $v^* = v_1 - iv_2$, $a = a_1 + ia_2$, and $a^* = a_1 - ia_2$. 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

```

T.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

```

T.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)
    end function f_vvf
    
```

```

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_svr, 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

```

T.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 :: fgvg, fgvg5gr, fggvvgr, grkgf, grkggf, grkkggf, &
        fgkgr, fg5gkgr, grvgf, grg5vgf, grkgggf, fggkgggr

<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(bispinor) :: psibar_r
    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:

$$\not{k} = \begin{pmatrix} 0 & 0 & k_- & -k^* \\ 0 & 0 & -k & k_+ \\ k_+ & k^* & 0 & 0 \\ k & k_- & 0 & 0 \end{pmatrix}, \quad \not{k}\gamma^5 = \begin{pmatrix} 0 & 0 & k_- & -k^* \\ 0 & 0 & -k & k_+ \\ -k_+ & -k^* & 0 & 0 \\ -k & -k_- & 0 & 0 \end{pmatrix} \quad (\text{T.72})$$

with $k_{\pm} = k_0 \pm k_3$, $k = k_1 + ik_2$, $k^* = k_1 - ik_2$. But note that \cdot^* is *not* complex conjugation for complex k_{μ} .

$$\gamma^0 \not{k} = \begin{pmatrix} k_+ & k^* & 0 & 0 \\ k & k_- & 0 & 0 \\ 0 & 0 & k_- & -k^* \\ 0 & 0 & -k & k_+ \end{pmatrix}, \quad \gamma^0 \not{k}\gamma^5 = \begin{pmatrix} -k_+ & -k^* & 0 & 0 \\ -k & -k_- & 0 & 0 \\ 0 & 0 & k_- & -k^* \\ 0 & 0 & -k & k_+ \end{pmatrix} \quad (\text{T.73a})$$

$$\gamma^1 \not{k} = \begin{pmatrix} k & k_- & 0 & 0 \\ k_+ & k^* & 0 & 0 \\ 0 & 0 & k & -k_+ \\ 0 & 0 & -k_- & k^* \end{pmatrix}, \quad \gamma^1 \not{k}\gamma^5 = \begin{pmatrix} -k & -k_- & 0 & 0 \\ -k_+ & -k^* & 0 & 0 \\ 0 & 0 & k & -k_+ \\ 0 & 0 & -k_- & k^* \end{pmatrix} \quad (\text{T.73b})$$

$$\gamma^2 \not{k} = \begin{pmatrix} -ik & -ik_- & 0 & 0 \\ ik_+ & ik^* & 0 & 0 \\ 0 & 0 & -ik & ik_+ \\ 0 & 0 & -ik_- & ik^* \end{pmatrix}, \quad \gamma^2 \not{k}\gamma^5 = \begin{pmatrix} ik & ik_- & 0 & 0 \\ -ik_+ & -ik^* & 0 & 0 \\ 0 & 0 & -ik & ik_+ \\ 0 & 0 & -ik_- & ik^* \end{pmatrix} \quad (\text{T.73c})$$

$$\gamma^3 \not{k} = \begin{pmatrix} k_+ & k^* & 0 & 0 \\ -k & -k_- & 0 & 0 \\ 0 & 0 & -k_- & k^* \\ 0 & 0 & -k & k_+ \end{pmatrix}, \quad \gamma^3 \not{k}\gamma^5 = \begin{pmatrix} -k_+ & -k^* & 0 & 0 \\ k & k_- & 0 & 0 \\ 0 & 0 & -k_- & k^* \\ 0 & 0 & -k & k_+ \end{pmatrix} \quad (\text{T.73d})$$

and

$$\not{k}\gamma^0 = \begin{pmatrix} k_- & -k^* & 0 & 0 \\ -k & k_+ & 0 & 0 \\ 0 & 0 & k_+ & k^* \\ 0 & 0 & k & k_- \end{pmatrix}, \quad \not{k}\gamma^0\gamma^5 = \begin{pmatrix} -k_- & k^* & 0 & 0 \\ k & -k_+ & 0 & 0 \\ 0 & 0 & k_+ & k^* \\ 0 & 0 & k & k_- \end{pmatrix} \quad (\text{T.74a})$$

$$\not{k}\gamma^1 = \begin{pmatrix} k^* & -k_- & 0 & 0 \\ -k_+ & k & 0 & 0 \\ 0 & 0 & k^* & k_+ \\ 0 & 0 & k_- & k \end{pmatrix}, \quad \not{k}\gamma^1\gamma^5 = \begin{pmatrix} -k^* & k_- & 0 & 0 \\ k_+ & -k & 0 & 0 \\ 0 & 0 & k^* & k_+ \\ 0 & 0 & k_- & k \end{pmatrix} \quad (\text{T.74b})$$

$$\not{k}\gamma^2 = \begin{pmatrix} ik^* & ik_- & 0 & 0 \\ -ik_+ & -ik & 0 & 0 \\ 0 & 0 & ik^* & -ik_+ \\ 0 & 0 & ik_- & -ik \end{pmatrix}, \quad \not{k}\gamma^2\gamma^5 = \begin{pmatrix} -ik^* & -ik_- & 0 & 0 \\ ik_+ & ik & 0 & 0 \\ 0 & 0 & ik^* & -ik_+ \\ 0 & 0 & ik_- & -ik \end{pmatrix} \quad (\text{T.74c})$$

$$\not{k}\gamma^3 = \begin{pmatrix} -k_- & -k^* & 0 & 0 \\ k & k_+ & 0 & 0 \\ 0 & 0 & k_+ & -k^* \\ 0 & 0 & k & -k_- \end{pmatrix}, \quad \not{k}\gamma^3\gamma^5 = \begin{pmatrix} k_- & k^* & 0 & 0 \\ -k & -k_+ & 0 & 0 \\ 0 & 0 & k_+ & -k^* \\ 0 & 0 & k & -k_- \end{pmatrix} \quad (\text{T.74d})$$

and

$$C\gamma^0\not{k} = \begin{pmatrix} k & k_- & 0 & 0 \\ -k_+ & -k^* & 0 & 0 \\ 0 & 0 & k & -k_+ \\ 0 & 0 & k_- & -k^* \end{pmatrix}, \quad C\gamma^0\not{k}\gamma^5 = \begin{pmatrix} -k & -k_- & 0 & 0 \\ k_+ & k^* & 0 & 0 \\ 0 & 0 & k & -k_+ \\ 0 & 0 & k_- & -k^* \end{pmatrix} \quad (\text{T.75a})$$

$$C\gamma^1\not{k} = \begin{pmatrix} k_+ & k^* & 0 & 0 \\ -k & -k_- & 0 & 0 \\ 0 & 0 & k_- & -k^* \\ 0 & 0 & k & -k_+ \end{pmatrix}, \quad C\gamma^1\not{k}\gamma^5 = \begin{pmatrix} -k_+ & -k^* & 0 & 0 \\ k & k_- & 0 & 0 \\ 0 & 0 & k_- & -k^* \\ 0 & 0 & k & -k_+ \end{pmatrix} \quad (\text{T.75b})$$

$$C\gamma^2\not{k} = \begin{pmatrix} ik_+ & ik^* & 0 & 0 \\ ik & ik_- & 0 & 0 \\ 0 & 0 & ik_- & -ik^* \\ 0 & 0 & -ik & ik_+ \end{pmatrix}, \quad C\gamma^2\not{k}\gamma^5 = \begin{pmatrix} -ik_+ & -ik^* & 0 & 0 \\ -ik & -ik_- & 0 & 0 \\ 0 & 0 & ik_- & -ik^* \\ 0 & 0 & -ik & ik_+ \end{pmatrix} \quad (\text{T.75c})$$

$$C\gamma^3\not{k} = \begin{pmatrix} -k & -k_- & 0 & 0 \\ -k_+ & -k^* & 0 & 0 \\ 0 & 0 & k & -k_+ \\ 0 & 0 & -k_- & k^* \end{pmatrix}, \quad C\gamma^3\not{k}\gamma^5 = \begin{pmatrix} k & k_- & 0 & 0 \\ k_+ & k^* & 0 & 0 \\ 0 & 0 & k & -k_+ \\ 0 & 0 & -k_- & k^* \end{pmatrix} \quad (\text{T.75d})$$

and

$$C\not{k}\gamma^0 = \begin{pmatrix} -k & k^+ & 0 & 0 \\ -k_- & k^* & 0 & 0 \\ 0 & 0 & -k & -k_- \\ 0 & 0 & k_+ & k^* \end{pmatrix}, \quad C\not{k}\gamma^0\gamma^5 = \begin{pmatrix} k & -k_+ & 0 & 0 \\ k_- & -k^* & 0 & 0 \\ 0 & 0 & -k & -k_- \\ 0 & 0 & k_+ & k^* \end{pmatrix} \quad (\text{T.76a})$$

$$C\not{k}\gamma^1 = \begin{pmatrix} -k_+ & k & 0 & 0 \\ -k^* & k_- & 0 & 0 \\ 0 & 0 & -k_- & -k \\ 0 & 0 & k^* & k_+ \end{pmatrix}, \quad C\not{k}\gamma^1\gamma^5 = \begin{pmatrix} k_+ & -k & 0 & 0 \\ k^* & -k_- & 0 & 0 \\ 0 & 0 & -k_- & -k \\ 0 & 0 & k^* & k_+ \end{pmatrix} \quad (\text{T.76b})$$

$$C\not{k}\gamma^2 = \begin{pmatrix} -ik_+ & -ik & 0 & 0 \\ -ik^* & -ik_- & 0 & 0 \\ 0 & 0 & -ik_- & ik \\ 0 & 0 & ik^* & -ik_+ \end{pmatrix}, \quad C\not{k}\gamma^2\gamma^5 = \begin{pmatrix} ik_+ & ik & 0 & 0 \\ ik^* & ik_- & 0 & 0 \\ 0 & 0 & -ik_- & ik \\ 0 & 0 & ik^* & -ik_+ \end{pmatrix} \quad (\text{T.76c})$$

$$C\!\!\!/\!\!k\gamma^3 = \begin{pmatrix} k & k_+ & 0 & 0 \\ k_- & k^* & 0 & 0 \\ 0 & 0 & -k & k_- \\ 0 & 0 & k_+ & -k^* \end{pmatrix}, \quad C\!\!\!/\!\!k\gamma^3\gamma^5 = \begin{pmatrix} -k & -k_+ & 0 & 0 \\ -k_- & -k^* & 0 & 0 \\ 0 & 0 & -k & k_- \\ 0 & 0 & k_+ & -k^* \end{pmatrix} \quad (\text{T.76d})$$

(Implementation of bispinor currents)+≡

```
pure function fgvr (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 fgvr
```

(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) * fgvr (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 = fgvr (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 $k_{\pm} = k_0 \pm k_3$, $k = k_1 + ik_2$, $k^* = k_1 - ik_2$ 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 in-

dices have the usual meaning of antisymmetrizing with respect to the indices inside the brackets, here without a factor two in the denominator)

$$a_+ = k_+ v_- + k v^* - k_- v_+ - k^* v = 2(k_{[3} v_{0]} + i k_{[2} v_{1]}) \quad (\text{T.77a})$$

$$a_- = k_- v_+ + k v^* - k_+ v_- - k^* v = 2(-k_{[3} v_{0]} + i k_{[2} v_{1]}) \quad (\text{T.77b})$$

$$b_+ = 2(k_+ v - k v_+) = 2(k_{[0} v_{1]} + k_{[3} v_{1]} + i k_{[0} v_{2]} + i k_{[3} v_{2]}) \quad (\text{T.77c})$$

$$b_- = 2(k_- v - k v_-) = 2(k_{[0} v_{1]} - k_{[3} v_{1]} + i k_{[0} v_{2]} - i k_{[3} v_{2]}) \quad (\text{T.77d})$$

$$b_{+*} = 2(k_+ v^* - k^* v_+) = 2(k_{[0} v_{1]} + k_{[3} v_{1]} - i k_{[0} v_{2]} - i k_{[3} v_{2]}) \quad (\text{T.77e})$$

$$b_{-*} = 2(k_- v^* - k^* v_-) = 2(k_{[0} v_{1]} - k_{[3} v_{1]} - i k_{[0} v_{2]} + i k_{[3} v_{2]}) \quad (\text{T.77f})$$

Of course, one could introduce a more advanced notation, but we don't want to become confused.

$$[k, \gamma^0] = \begin{pmatrix} -2k_3 & -2k^* & 0 & 0 \\ -2k & 2k_3 & 0 & 0 \\ 0 & 0 & 2k_3 & 2k^* \\ 0 & 0 & 2k & -2k_3 \end{pmatrix} \quad (\text{T.78a})$$

$$[k, \gamma^1] = \begin{pmatrix} -2ik_2 & -2k_- & 0 & 0 \\ -2k_+ & 2ik_2 & 0 & 0 \\ 0 & 0 & -2ik_2 & 2k_+ \\ 0 & 0 & 2k_- & 2ik_2 \end{pmatrix} \quad (\text{T.78b})$$

$$[k, \gamma^2] = \begin{pmatrix} 2ik_1 & 2ik_- & 0 & 0 \\ -2ik_+ & -2ik_1 & 0 & 0 \\ 0 & 0 & 2ik_1 & -2ik_+ \\ 0 & 0 & 2ik_- & -2ik_1 \end{pmatrix} \quad (\text{T.78c})$$

$$[k, \gamma^3] = \begin{pmatrix} -2k_0 & -2k^* & 0 & 0 \\ 2k & 2k_0 & 0 & 0 \\ 0 & 0 & 2k_0 & -2k^* \\ 0 & 0 & 2k & -2k_0 \end{pmatrix} \quad (\text{T.78d})$$

$$[k, V] = \begin{pmatrix} a_- & b_{-*} & 0 & 0 \\ b_+ & -a_- & 0 & 0 \\ 0 & 0 & a_+ & -b_{+*} \\ 0 & 0 & -b_- & -a_+ \end{pmatrix} \quad (\text{T.78e})$$

$$\gamma^5 \gamma^0 [k, V] = \begin{pmatrix} 0 & 0 & -a_+ & b_{+*} \\ 0 & 0 & b_- & a_+ \\ a_- & b_{-*} & 0 & 0 \\ b_+ & -a_- & 0 & 0 \end{pmatrix} \quad (\text{T.78f})$$

$$\gamma^5 \gamma^1 [k, V] = \begin{pmatrix} 0 & 0 & b_- & a_+ \\ 0 & 0 & -a_+ & b_{+*} \\ -b_+ & a_- & 0 & 0 \\ -a_- & -b_{-*} & 0 & 0 \end{pmatrix} \quad (\text{T.78g})$$

$$\gamma^5 \gamma^2 [\not{k}, V] = \begin{pmatrix} 0 & 0 & -ib_- & -ia_+ \\ 0 & 0 & -ia_+ & ib_{+*} \\ ib_+ & -ia_- & 0 & 0 \\ -ia_- & -ib_{-*} & 0 & 0 \end{pmatrix} \quad (\text{T.78h})$$

$$\gamma^5 \gamma^3 [\not{k}, V] = \begin{pmatrix} 0 & 0 & -a_+ & b_{+*} \\ 0 & 0 & -b_- & -a_+ \\ -a_- & -b_{-*} & 0 & 0 \\ b_+ & -a_- & 0 & 0 \end{pmatrix} \quad (\text{T.78i})$$

and

$$[\not{k}, V] \gamma^0 \gamma^5 = \begin{pmatrix} 0 & 0 & a_- & b_{-*} \\ 0 & 0 & b_+ & -a_- \\ -a_+ & b_{+*} & 0 & 0 \\ b_- & a_+ & 0 & 0 \end{pmatrix} \quad (\text{T.79a})$$

$$[\not{k}, V] \gamma^1 \gamma^5 = \begin{pmatrix} 0 & 0 & b_{-*} & a_- \\ 0 & 0 & -a_- & b_+ \\ -b_{+*} & a_+ & 0 & 0 \\ -a_+ & -b_- & 0 & 0 \end{pmatrix} \quad (\text{T.79b})$$

$$[\not{k}, V] \gamma^2 \gamma^5 = \begin{pmatrix} 0 & 0 & ib_{-*} & -ia_- \\ 0 & 0 & -ia_- & -ib_+ \\ -ib_{+*} & -ia_+ & 0 & 0 \\ -ia_+ & ib_- & 0 & 0 \end{pmatrix} \quad (\text{T.79c})$$

$$[\not{k}, V] \gamma^3 \gamma^5 = \begin{pmatrix} 0 & 0 & a_- & -b_{-*} \\ 0 & 0 & b_+ & a_- \\ a_+ & b_{+*} & 0 & 0 \\ -b_- & a_+ & 0 & 0 \end{pmatrix} \quad (\text{T.79d})$$

In what follows l always means twice the value of k , e.g. $l_+ = 2k_+$. We use the abbreviation $C^{\mu\nu} \equiv C[\not{k}, \gamma^\mu] \gamma^\nu \gamma^5$.

$$C^{00} = \begin{pmatrix} 0 & 0 & -l & -l_3 \\ 0 & 0 & l_3 & l^* \\ l & -l_3 & 0 & 0 \\ -l_3 & -l^* & 0 & 0 \end{pmatrix}, \quad C^{20} = \begin{pmatrix} 0 & 0 & -il_+ & -il_1 \\ 0 & 0 & -il_1 & -il_- \\ il_- & -il_1 & 0 & 0 \\ -il_1 & il_+ & 0 & 0 \end{pmatrix} \quad (\text{T.80a})$$

$$C^{01} = \begin{pmatrix} 0 & 0 & l_3 & -l \\ 0 & 0 & l^* & l_3 \\ l_3 & -l & 0 & 0 \\ l^* & l_3 & 0 & 0 \end{pmatrix}, \quad C^{21} = \begin{pmatrix} 0 & 0 & -il_1 & -il_+ \\ 0 & 0 & -il_- & -il_1 \\ il_1 & -il_- & 0 & 0 \\ -il_+ & il_1 & 0 & 0 \end{pmatrix} \quad (\text{T.80b})$$

$$C^{02} = \begin{pmatrix} 0 & 0 & il_3 & il \\ 0 & 0 & il^* & -il_3 \\ il_3 & il & 0 & 0 \\ il^* & -il_3 & 0 & 0 \end{pmatrix}, \quad C^{22} = \begin{pmatrix} 0 & 0 & l_1 & -l_+ \\ 0 & 0 & l_- & -l_1 \\ -l_1 & -l_- & 0 & 0 \\ l_+ & l_1 & 0 & 0 \end{pmatrix} \quad (\text{T.80c})$$

$$C^{03} = \begin{pmatrix} 0 & 0 & -l & -l_3 \\ 0 & 0 & l_3 & -l^* \\ -l & -l_3 & 0 & 0 \\ l_3 & -l^* & 0 & 0 \end{pmatrix}, \quad C^{23} = \begin{pmatrix} 0 & 0 & -il_+ & il_1 \\ 0 & 0 & -il_1 & il_- \\ -il_- & -il_1 & 0 & 0 \\ il_1 & il_+ & 0 & 0 \end{pmatrix} \quad (\text{T.80d})$$

$$C^{10} = \begin{pmatrix} 0 & 0 & -l_+ & il_2 \\ 0 & 0 & il_2 & l_- \\ l_- & il_2 & 0 & 0 \\ il_2 & -l_+ & 0 & 0 \end{pmatrix}, \quad C^{30} = \begin{pmatrix} 0 & 0 & l & l_0 \\ 0 & 0 & l_0 & l^* \\ l & -l_0 & 0 & 0 \\ -l_0 & l^* & 0 & 0 \end{pmatrix} \quad (\text{T.80e})$$

$$C^{11} = \begin{pmatrix} 0 & 0 & il_2 & -l_+ \\ 0 & 0 & l_- & il_2 \\ -il_2 & -l_- & 0 & 0 \\ l_+ & -il_2 & 0 & 0 \end{pmatrix}, \quad C^{31} = \begin{pmatrix} 0 & 0 & l_0 & l \\ 0 & 0 & l^* & l_0 \\ l_0 & -l & 0 & 0 \\ -l^* & l_0 & 0 & 0 \end{pmatrix} \quad (\text{T.80f})$$

$$C^{12} = \begin{pmatrix} 0 & 0 & -l_2 & il_+ \\ 0 & 0 & il_- & l_2 \\ l_2 & il_- & 0 & 0 \\ il_+ & -l_2 & 0 & 0 \end{pmatrix}, \quad C^{32} = \begin{pmatrix} 0 & 0 & il_0 & -il \\ 0 & 0 & il^* & -il_0 \\ il_0 & il & 0 & 0 \\ -il^* & -il_0 & 0 & 0 \end{pmatrix} \quad (\text{T.80g})$$

$$C^{13} = \begin{pmatrix} 0 & 0 & -l_+ & -il_2 \\ 0 & 0 & il_2 & -l_- \\ -l_- & il_2 & 0 & 0 \\ -il_2 & -l_+ & 0 & 0 \end{pmatrix}, \quad C^{33} = \begin{pmatrix} 0 & 0 & l & -l_0 \\ 0 & 0 & l_0 & -l^* \\ -l & -l_0 & 0 & 0 \\ l_0 & l^* & 0 & 0 \end{pmatrix} \quad (\text{T.80h})$$

and, with the abbreviation $\tilde{C}^{\mu\nu} \equiv C\gamma^5\gamma^\nu[k, \gamma^\mu]$ (note the reversed order of the indices!)

$$\tilde{C}^{00} = \begin{pmatrix} 0 & 0 & -l & l_3 \\ 0 & 0 & l_3 & l^* \\ l & -l_3 & 0 & 0 \\ -l_3 & -l^* & 0 & 0 \end{pmatrix}, \quad \tilde{C}^{20} = \begin{pmatrix} 0 & 0 & -il_- & il_1 \\ 0 & 0 & il_1 & -il_+ \\ il_+ & il_1 & 0 & 0 \\ il_1 & il_- & 0 & 0 \end{pmatrix} \quad (\text{T.81a})$$

$$\tilde{C}^{01} = \begin{pmatrix} 0 & 0 & -l_3 & -l^* \\ 0 & 0 & l & -l_3 \\ -l_3 & -l^* & 0 & 0 \\ l & -l_3 & 0 & 0 \end{pmatrix}, \quad \tilde{C}^{21} = \begin{pmatrix} 0 & 0 & -il_1 & il_+ \\ 0 & 0 & il_- & -il_1 \\ il_1 & il_- & 0 & 0 \\ il_+ & il_1 & 0 & 0 \end{pmatrix} \quad (\text{T.81b})$$

$$\tilde{C}^{02} = \begin{pmatrix} 0 & 0 & -il_3 & -il^* \\ 0 & 0 & -il & il_3 \\ -il_3 & -il^* & 0 & 0 \\ -il & il_3 & 0 & 0 \end{pmatrix}, \quad \tilde{C}^{22} = \begin{pmatrix} 0 & 0 & l_1 & -l_+ \\ 0 & 0 & l_- & -l_1 \\ -l_1 & -l_- & 0 & 0 \\ l_+ & l_1 & 0 & 0 \end{pmatrix} \quad (\text{T.81c})$$

$$\tilde{C}^{03} = \begin{pmatrix} 0 & 0 & l & -l_3 \\ 0 & 0 & l_3 & l^* \\ l & -l_3 & 0 & 0 \\ l_3 & l^* & 0 & 0 \end{pmatrix}, \quad \tilde{C}^{23} = \begin{pmatrix} 0 & 0 & il_- & -il_1 \\ 0 & 0 & il_1 & -il_+ \\ il_+ & il_1 & 0 & 0 \\ -il_1 & -il_- & 0 & 0 \end{pmatrix} \quad (\text{T.81d})$$

$$\tilde{C}^{10} = \begin{pmatrix} 0 & 0 & -l_- & -il_2 \\ 0 & 0 & -il_2 & l_+ \\ l_+ & -il_2 & 0 & 0 \\ -il_2 & -l_- & 0 & 0 \end{pmatrix}, \quad \tilde{C}^{30} = \begin{pmatrix} 0 & 0 & -l & l_0 \\ 0 & 0 & l_0 & -l^* \\ -l & -l_0 & 0 & 0 \\ -l_0 & -l^* & 0 & 0 \end{pmatrix} \quad (\text{T.81e})$$

$$\tilde{C}^{11} = \begin{pmatrix} 0 & 0 & il_2 & -l_+ \\ 0 & 0 & l_- & il_2 \\ -il_2 & -l_- & 0 & 0 \\ l_+ & -il_2 & 0 & 0 \end{pmatrix}, \quad \tilde{C}^{31} = \begin{pmatrix} 0 & 0 & -l_0 & l^* \\ 0 & 0 & l & -l_0 \\ -l_0 & -l^* & 0 & 0 \\ -l & -l_0 & 0 & 0 \end{pmatrix} \quad (\text{T.81f})$$

$$\tilde{C}^{12} = \begin{pmatrix} 0 & 0 & -l_2 & -il_+ \\ 0 & 0 & -il_- & l_2 \\ l_2 & -il_- & 0 & 0 \\ -il_+ & -l_2 & 0 & 0 \end{pmatrix}, \quad \tilde{C}^{32} = \begin{pmatrix} 0 & 0 & -il_0 & il^* \\ 0 & 0 & -il & il_0 \\ -il_0 & -il^* & 0 & 0 \\ il & il_0 & 0 & 0 \end{pmatrix} \quad (\text{T.81g})$$

$$\tilde{C}^{13} = \begin{pmatrix} 0 & 0 & l_- & il_2 \\ 0 & 0 & -il_2 & l_+ \\ l_+ & -il_2 & 0 & 0 \\ il_2 & l_- & 0 & 0 \end{pmatrix}, \quad \tilde{C}^{33} = \begin{pmatrix} 0 & 0 & l & -l_0 \\ 0 & 0 & l_0 & -l^* \\ -l & -l_0 & 0 & 0 \\ l_0 & l^* & 0 & 0 \end{pmatrix} \quad (\text{T.81h})$$

(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_vlrg (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_vlrg

<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(bispinor) :: psi_r
  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 grkggf (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 grkggf

```

(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 grkkggf (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_psil0, c_psil1, c_psil2, c_psil3
    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_psil0%psi(1)%a(1) = (-k%x(3)) * psir%a(3) + (-k12s) * psir%a(4)
    c_psil0%psi(1)%a(2) = (-k12) * psir%a(3) + k%x(3) * psir%a(4)
    c_psil0%psi(1)%a(3) = (-k%x(3)) * psir%a(1) + (-k12s) * psir%a(2)
    c_psil0%psi(1)%a(4) = (-k12) * psir%a(1) + k%x(3) * psir%a(2)
    c_psil0%psi(2)%a(1) = (-k12s) * psir%a(3) + (-k%x(3)) * psir%a(4)
    c_psil0%psi(2)%a(2) = k%x(3) * psir%a(3) + (-k12) * psir%a(4)
    c_psil0%psi(2)%a(3) = k12s * psir%a(1) + k%x(3) * psir%a(2)
    c_psil0%psi(2)%a(4) = (-k%x(3)) * psir%a(1) + k12 * psir%a(2)
    c_psil0%psi(3)%a(1) = (0,1) * ((-k12s) * psir%a(3) + k%x(3) * psir%a(4))
    c_psil0%psi(3)%a(2) = (0,1) * (k%x(3) * psir%a(3) + k12 * psir%a(4))
    c_psil0%psi(3)%a(3) = (0,1) * (k12s * psir%a(1) + (-k%x(3)) * psir%a(2))
    c_psil0%psi(3)%a(4) = (0,1) * ((-k%x(3)) * psir%a(1) + (-k12) * psir%a(2))
    c_psil0%psi(4)%a(1) = (-k%x(3)) * psir%a(3) + k12s * psir%a(4)
    c_psil0%psi(4)%a(2) = (-k12) * psir%a(3) + (-k%x(3)) * psir%a(4)
    c_psil0%psi(4)%a(3) = k%x(3) * psir%a(1) + (-k12s) * psir%a(2)

```

```

c_psi0%psi(4)%a(4) = k12 * psir%a(1) + k%x(3) * psir%a(2)
!!!
c_psi1%psi(1)%a(1) = (-ik2) * psir%a(3) + (-km) * psir%a(4)
c_psi1%psi(1)%a(2) = (-kp) * psir%a(3) + ik2 * psir%a(4)
c_psi1%psi(1)%a(3) = ik2 * psir%a(1) + (-kp) * psir%a(2)
c_psi1%psi(1)%a(4) = (-km) * psir%a(1) + (-ik2) * psir%a(2)
c_psi1%psi(2)%a(1) = (-km) * psir%a(3) + (-ik2) * psir%a(4)
c_psi1%psi(2)%a(2) = ik2 * psir%a(3) + (-kp) * psir%a(4)
c_psi1%psi(2)%a(3) = kp * psir%a(1) + (-ik2) * psir%a(2)
c_psi1%psi(2)%a(4) = ik2 * psir%a(1) + km * psir%a(2)
c_psi1%psi(3)%a(1) = ((0,-1) * km) * psir%a(3) + (-k%x(2)) * psir%a(4)
c_psi1%psi(3)%a(2) = (-k%x(2)) * psir%a(3) + ((0,1) * kp) * psir%a(4)
c_psi1%psi(3)%a(3) = ((0,1) * kp) * psir%a(1) + (-k%x(2)) * psir%a(2)
c_psi1%psi(3)%a(4) = (-k%x(2)) * psir%a(1) + ((0,-1) * km) * psir%a(2)
c_psi1%psi(4)%a(1) = (-ik2) * psir%a(3) + km * psir%a(4)
c_psi1%psi(4)%a(2) = (-kp) * psir%a(3) + (-ik2) * psir%a(4)
c_psi1%psi(4)%a(3) = (-ik2) * psir%a(1) + (-kp) * psir%a(2)
c_psi1%psi(4)%a(4) = km * psir%a(1) + (-ik2) * psir%a(2)
!!!
c_psi2%psi(1)%a(1) = (0,1) * (k%x(1) * psir%a(3) + km * psir%a(4))
c_psi2%psi(1)%a(2) = (0,-1) * (kp * psir%a(3) + k%x(1) * psir%a(4))
c_psi2%psi(1)%a(3) = (0,1) * ((-k%x(1)) * psir%a(1) + kp * psir%a(2))
c_psi2%psi(1)%a(4) = (0,1) * ((-km) * psir%a(1) + k%x(1) * psir%a(2))
c_psi2%psi(2)%a(1) = (0,1) * (km * psir%a(3) + k%x(1) * psir%a(4))
c_psi2%psi(2)%a(2) = (0,-1) * (k%x(1) * psir%a(3) + kp * psir%a(4))
c_psi2%psi(2)%a(3) = (0,-1) * (kp * psir%a(1) + (-k%x(1)) * psir%a(2))
c_psi2%psi(2)%a(4) = (0,-1) * (k%x(1) * psir%a(1) + (-km) * psir%a(2))
c_psi2%psi(3)%a(1) = (-km) * psir%a(3) + k%x(1) * psir%a(4)
c_psi2%psi(3)%a(2) = k%x(1) * psir%a(3) + (-kp) * psir%a(4)
c_psi2%psi(3)%a(3) = kp * psir%a(1) + k%x(1) * psir%a(2)
c_psi2%psi(3)%a(4) = k%x(1) * psir%a(1) + km * psir%a(2)
c_psi2%psi(4)%a(1) = (0,1) * (k%x(1) * psir%a(3) + (-km) * psir%a(4))
c_psi2%psi(4)%a(2) = (0,1) * ((-kp) * psir%a(3) + k%x(1) * psir%a(4))
c_psi2%psi(4)%a(3) = (0,1) * (k%x(1) * psir%a(1) + kp * psir%a(2))
c_psi2%psi(4)%a(4) = (0,1) * (km * psir%a(1) + k%x(1) * psir%a(2))
!!!
c_psi3%psi(1)%a(1) = (-k%t) * psir%a(3) - k12s * psir%a(4)
c_psi3%psi(1)%a(2) = k12 * psir%a(3) + k%t * psir%a(4)
c_psi3%psi(1)%a(3) = (-k%t) * psir%a(1) + k12s * psir%a(2)
c_psi3%psi(1)%a(4) = (-k12) * psir%a(1) + k%t * psir%a(2)
c_psi3%psi(2)%a(1) = (-k12s) * psir%a(3) + (-k%t) * psir%a(4)
c_psi3%psi(2)%a(2) = k%t * psir%a(3) + k12 * psir%a(4)
c_psi3%psi(2)%a(3) = (-k12s) * psir%a(1) + k%t * psir%a(2)
c_psi3%psi(2)%a(4) = (-k%t) * psir%a(1) + k12 * psir%a(2)
c_psi3%psi(3)%a(1) = (0,-1) * (k12s * psir%a(3) + (-k%t) * psir%a(4))
c_psi3%psi(3)%a(2) = (0,1) * (k%t * psir%a(3) + (-k12) * psir%a(4))
c_psi3%psi(3)%a(3) = (0,-1) * (k12s * psir%a(1) + k%t * psir%a(2))
c_psi3%psi(3)%a(4) = (0,-1) * (k%t * psir%a(1) + k12 * psir%a(2))
c_psi3%psi(4)%a(1) = (-k%t) * psir%a(3) + k12s * psir%a(4)
c_psi3%psi(4)%a(2) = k12 * psir%a(3) + (-k%t) * psir%a(4)
c_psi3%psi(4)%a(3) = k%t * psir%a(1) + k12s * psir%a(2)
c_psi3%psi(4)%a(4) = k12 * psir%a(1) + k%t * psir%a(2)
j%t    = 2 * (psil * c_psi0)
j%x(1) = 2 * (psil * c_psi1)
    
```

```

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_psi0%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_psi0%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_psi1%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_psi1%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_psi1%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_psi1%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_psi2%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_psi2%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_psi2%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_psi2%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_psi3%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_psi3%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_psi3%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_psi3%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_psi0%a)
j%x(1)   = 2 * dot_product (conjg (psil%a), c_psi1%a)
j%x(2)   = 2 * dot_product (conjg (psil%a), c_psi2%a)
j%x(3)   = 2 * dot_product (conjg (psil%a), c_psi3%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

```

T.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_v2lgr

<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, phidum
  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) * fgvggr (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_v2lrg (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_v2lrg

<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(bispinor) :: psi_r
  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) * grkggf (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
    
```

$$C\gamma^0\gamma^0 = -C\gamma^1\gamma^1 = -C\gamma^2\gamma^2 = C\gamma^3\gamma^3 = C = \begin{pmatrix} 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \quad (\text{T.82a})$$

$$C\gamma^0\gamma^1 = -C\gamma^1\gamma^0 = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (\text{T.82b})$$

$$C\gamma^0\gamma^2 = -C\gamma^2\gamma^0 = \begin{pmatrix} -i & 0 & 0 & 0 \\ 0 & -i & 0 & 0 \\ 0 & 0 & -i & 0 \\ 0 & 0 & 0 & -i \end{pmatrix} \quad (\text{T.82c})$$

$$C\gamma^0\gamma^3 = -C\gamma^3\gamma^0 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \quad (\text{T.82d})$$

$$C\gamma^1\gamma^2 = -C\gamma^2\gamma^1 = \begin{pmatrix} 0 & i & 0 & 0 \\ i & 0 & 0 & 0 \\ 0 & 0 & 0 & -i \\ 0 & 0 & -i & 0 \end{pmatrix} \quad (\text{T.82e})$$

$$C\gamma^1\gamma^3 = -C\gamma^3\gamma^1 = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (\text{T.82f})$$

$$C\gamma^2\gamma^3 = -C\gamma^3\gamma^2 = \begin{pmatrix} -i & 0 & 0 & 0 \\ 0 & i & 0 & 0 \\ 0 & 0 & i & 0 \\ 0 & 0 & 0 & -i \end{pmatrix} \quad (\text{T.82g})$$

(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

```

T.14.6 On Shell Wave Functions

(Declaration of bispinor on shell wave functions)≡

```

public :: u, v, ghost

```

$$\chi_+(\vec{p}) = \frac{1}{\sqrt{2|\vec{p}|(|\vec{p}| + p_3)}} \begin{pmatrix} |\vec{p}| + p_3 \\ p_1 + ip_2 \end{pmatrix} \quad (\text{T.83a})$$

$$\chi_-(\vec{p}) = \frac{1}{\sqrt{2|\vec{p}|(|\vec{p}| + p_3)}} \begin{pmatrix} -p_1 + ip_2 \\ |\vec{p}| + p_3 \end{pmatrix} \quad (\text{T.83b})$$

$$u_{\pm}(p) = \begin{pmatrix} \sqrt{p_0 \mp |\vec{p}|} \cdot \chi_{\pm}(\vec{p}) \\ \sqrt{p_0 \pm |\vec{p}|} \cdot \chi_{\pm}(\vec{p}) \end{pmatrix} \quad (\text{T.84})$$

(Implementation of bispinor on shell wave functions)≡

```

pure function u (m, p, s) result (psi)
  type(bispinor) :: psi
  real(kind=default), intent(in) :: m
  type(momentum), intent(in) :: p
  integer, intent(in) :: s
  complex(kind=default), dimension(2) :: chip, chim
  real(kind=default) :: pabs, norm
  pabs = sqrt (dot_product (p%x, p%x))
  if (pabs + p%x(3) <= 1000 * epsilon (pabs) * pabs) then
    !!! OLD VERSION !!!!!
    !!! if (1 + p%x(3) / pabs <= epsilon (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 (max (p%t - pabs, 0.0_default)) * chip
        psi%a(3:4) = sqrt (p%t + pabs) * chip
    else
        psi%a(1:2) = sqrt (p%t + pabs) * chim
        psi%a(3:4) = sqrt (max (p%t - pabs, 0.0_default)) * chim
    end if
    pabs = m ! make the compiler happy and use m
end function u
!pure function u (m, p, s) result (psi)
! type(bispinor) :: psi
! real(kind=default), intent(in) :: m
! type(momentum), intent(in) :: p
! integer, intent(in) :: s
! complex(kind=default), dimension(2) :: chip, chim
! real(kind=default) :: pabs, norm
! pabs = sqrt (dot_product (p%x, p%x))
! if (p%x(3) <= epsilon(p%x(3))) 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
!     if (1 + p%x(3) / pabs <= epsilon (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
! end if
! if (s > 0) then
!     psi%a(1:2) = sqrt (max (p%t - pabs, 0.0_default)) * chip
!     psi%a(3:4) = sqrt (p%t + pabs) * chip
! else
!     psi%a(1:2) = sqrt (p%t + pabs) * chim
!     psi%a(3:4) = sqrt (max (p%t - pabs, 0.0_default)) * chim
! end if
! pabs = m ! make the compiler happy and use m
!end function u

```

$$v_{\pm}(p) = \begin{pmatrix} \mp \sqrt{p_0 \pm |\vec{p}|} \cdot \chi_{\mp}(\vec{p}) \\ \pm \sqrt{p_0 \mp |\vec{p}|} \cdot \chi_{\mp}(\vec{p}) \end{pmatrix} \quad (\text{T.85})$$

(Implementation of bispinor on shell wave functions)+≡

```

pure function v (m, p, s) result (psi)
  type(bispinor) :: psi
  real(kind=default), intent(in) :: m
  type(momentum), intent(in) :: p
  integer, intent(in) :: s
  complex(kind=default), dimension(2) :: chip, chim
  real(kind=default) :: pabs, norm
  pabs = sqrt(dot_product(p%x, p%x))
  if (pabs + p%x(3) <= 1000 * epsilon(pabs) * pabs) then
!!! OLD VERSION !!!!!
!!! if (1 + p%x(3) / pabs <= epsilon(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) = sqrt (max (p%t - pabs, 0.0_default)) * chim
  else
      psi%a(1:2) = sqrt (max (p%t - pabs, 0.0_default)) * chip
      psi%a(3:4) = - sqrt (p%t + pabs) * chip
  end if
  pabs = m ! make the compiler happy and use m
end function v

!pure function v (m, p, s) result (psi)
! type(bispinor) :: psi
! real(kind=default), intent(in) :: m
! type(momentum), intent(in) :: p
! integer, intent(in) :: s
! complex(kind=default), dimension(2) :: chip, chim
! real(kind=default) :: pabs, norm
! pabs = sqrt(dot_product(p%x, p%x))
! if (p%x(3) <= epsilon(p%x(3))) then
!   chip = (/ cmplx ( 1.0, 0.0, kind=default), &
!             cmplx ( 0.0, 0.0, kind=default) /)
!   chim = (/ cmplx ( 0.0, 0.0, kind=default), &
!             cmplx ( 1.0, 0.0, kind=default) /)
! else
!   if (1 + p%x(3) / pabs <= epsilon(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) /)
!   end if
! end if

```

```

!               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
! end if
! if (s > 0) then
!     psi%a(1:2) = - sqrt (p%t + pabs) * chim
!     psi%a(3:4) =  sqrt (max (p%t - pabs, 0.0_default)) * chim
! else
!     psi%a(1:2) =  sqrt (max (p%t - pabs, 0.0_default)) * chip
!     psi%a(3:4) = - sqrt (p%t + pabs) * chip
! end if
! pabs = m ! make the compiler happy and use m
!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

```

T.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:

$$brsu(p) = (-i)(\not{p} - m)u(p) \quad (T.86)$$

(Implementation of bispinor off shell wave functions) \equiv

```
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=cplx(0.0,-1.0)*(f_vf(one,vp,psi)-m*psi)
end function brs_u
```

$$brsv(p) = i(\not{p} + m)v(p) \quad (T.87)$$

(Implementation of bispinor off shell wave functions) $+\equiv$

```
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=cplx(0.0,1.0)*(f_vf(one,vp,psi)+m*psi)
end function brs_v
```

T.14.8 Propagators

(Declaration of bispinor propagators) \equiv

```
public :: pr_psi, pr_grav
public :: pj_psi, pg_psi
```

$$\frac{i(-\not{p} + m)}{p^2 - m^2 + im\Gamma}\psi \quad (T.88)$$

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) \equiv

```
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

```

$$\sqrt{\frac{\pi}{M\Gamma}}(-\not{p} + m)\psi \quad (\text{T.89})$$

(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

```

$$i \frac{\left\{ (-\not{p} + m) \left(-\eta_{\mu\nu} + \frac{p_\mu p_\nu}{m^2} \right) + \frac{1}{3} \left(\gamma_\mu - \frac{p_\mu}{m} \right) (\not{p} + m) \left(\gamma_\nu - \frac{p_\nu}{m} \right) \right\}}{p^2 - m^2 + im\Gamma} \psi^\nu \quad (\text{T.90})$$

(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%y(1) * grav%psi(3) - p%z(1) * 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 / cmlpx (p*p - m**2, m*w, kind=default)) * &
            (etagrav + pppgrav + (1/3.0_default) * gg_grav)
end function pr_grav

```

T.15 Polarization vectorspinors

Here we construct the wavefunctions for (massive) gravitinos out of the wavefunctions of (massive) vectorbosons and (massive) Majorana fermions.

$$\psi_{(u;3/2)}^\mu(k) = \epsilon_+^\mu(k) \cdot u(k, +) \quad (\text{T.91a})$$

$$\psi_{(u;1/2)}^\mu(k) = \sqrt{\frac{1}{3}} \epsilon_+^\mu(k) \cdot u(k, -) + \sqrt{\frac{2}{3}} \epsilon_0^\mu(k) \cdot u(k, +) \quad (\text{T.91b})$$

$$\psi_{(u;-1/2)}^\mu(k) = \sqrt{\frac{2}{3}} \epsilon_0^\mu(k) \cdot u(k, -) + \sqrt{\frac{1}{3}} \epsilon_-^\mu(k) \cdot u(k, +) \quad (\text{T.91c})$$

$$\psi_{(u;-3/2)}^\mu(k) = \epsilon_-^\mu(k) \cdot u(k, -) \quad (\text{T.91d})$$

and in the same manner for $\psi_{(v;s)}^\mu$ 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

$$\gamma_\mu \psi_{(u/v;s)}^\mu = 0. \quad (\text{T.92})$$

```

(omega_vspinor_polarizations.f90)≡
<Cotypeleft>
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_2009_06_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.

$$\epsilon_+^\mu(k) = \frac{-e^{+i\phi}}{\sqrt{2}} (0; \cos\theta \cos\phi - i \sin\phi, \cos\theta \sin\phi + i \cos\phi, -\sin\theta) \quad (\text{T.93a})$$

$$\epsilon_-^\mu(k) = \frac{e^{-i\phi}}{\sqrt{2}} (0; \cos\theta \cos\phi + i \sin\phi, \cos\theta \sin\phi - i \cos\phi, -\sin\theta) \quad (\text{T.93b})$$

$$\epsilon_0^\mu(k) = \frac{1}{m} \left(|\vec{k}|; k^0 \sin\theta \cos\phi, k^0 \sin\theta \sin\phi, k^0 \cos\theta \right) \quad (\text{T.93c})$$

Determining the mass from the momenta is a numerically haphazardous for light particles. Therefore, we accept some redundancy and pass the mass explicitly. 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 (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) :: kabs, kabs2, sqrt2
    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)
    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
    end if

```

```

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

```

T.16 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, private :: REPEAT = 5, SAMPLE = 10
    integer, parameter, public :: omega_utils_2009_06_A = 0
contains
    <Implementation of utility functions>
end module omega_utils

```

T.16.1 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)
        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(out) :: mask
real(kind=default), intent(in) :: threshold
integer, intent(in) :: cutoff
integer :: h
real(kind=default) :: avg
if (threshold .gt. 0) then
  count = count + 1
  if (count .le. cutoff) then
    forall (h = lbound (amp, 2) : ubound (amp, 2))
      max_abs(h) = max (max_abs(h), maxval (abs (amp(:,h,:))))
    end forall
    sum_abs = sum_abs + sum (abs (amp))
    if (count .eq. cutoff) then
      avg = sum_abs / size (amp) / cutoff
      mask = max_abs .ge. threshold * epsilon (avg) * avg
    end if
  end if
end if
end subroutine omega_update_helicity_selection

```

T.16.2 Diagnostics

(Declaration of utility functions)+≡

```
public :: omega_report_helicity_selection
```

(Implementation of utility functions)+≡

```

subroutine omega_report_helicity_selection (mask, spin_states, threshold)
  logical, dimension(:), intent(in) :: mask
  integer, dimension(:,:), intent(in) :: spin_states
  real(kind=default), intent(in) :: threshold
  integer :: h, i
  write (unit = *, &
    fmt = "('| ', 'Contributing Helicity Combinations: ', I5, ' of ', I5)") &
    count (mask), size (mask)
  write (unit = *, &
    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 = *, fmt = "('| ', I4, ': ', 20I4)") i, spin_states (:, h)
    end if
  end do
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 these 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_ek > 1000 * epsilon (abs_ek_abs_e)) then
            print *, "O'Mega: warning: non-transverse vector field: ", &
                name, ":", abs_ek / 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_ek, abs_ek_abs_e
    ek = eps (m, k, 4)
    abs_ek = abs (ek * e - phi)
    abs_ek_abs_e = abs (ek) * abs (e)
    if (abs_ek > 1000 * epsilon (abs_ek_abs_e)) then
        print *, "O'Mega: panic: non-transverse vector field: ", &
            name, ":", abs_ek / 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

```

T.16.3 Summation & Density Matrices

```

<Declaration of utility functions>+≡
    public :: omega_spin_sum_sqme_1, omega_sum_sqme

<Implementation of utility functions>+≡
    pure function omega_spin_sum_sqme_1 &
        (amplitude_1, k, f, s_max, smask) result (amp2)
    real(kind=default), dimension(0:,:), intent(in) :: k
    integer, intent(in) :: f, s_max
    logical, dimension(:), intent(in), optional :: smask
    real(kind=default) :: amp2
    <Interface amplitude_1>
    complex(kind=default) :: amp
    integer :: s
    amp2 = 0
    if (present (smask)) then
        do s = 1, s_max
            if (smask(s)) then
                amp = amplitude_1 (k, s, f)
                amp2 = amp2 + amp * conjg (amp)
            end if
        end do
    end if
end function omega_spin_sum_sqme_1

```



```

else
  do s = 1, s_max
    amp = amplitude_1 (k, s, f)
    amp2 = amp2 + amp * conjg (amp)
  end do
end if
end function omega_spin_sum_sqme_1

<Interface amplitude_1>≡
interface
  pure function amplitude_1 (k, s, f) result (amp)
    use kinds
    implicit none
    real(kind=default), dimension(0:,:), intent(in) :: k
    integer, intent(in) :: s, f
    complex(kind=default) :: amp
  end function amplitude_1
end interface

<Implementation of utility functions>+≡
pure function omega_sum_sqme &
  (amplitude_1, k, s_max, f_max, mult, smask, fmask) result (amp2)
  real(kind=default), dimension(0:,:), intent(in) :: k
  integer, intent(in) :: s_max, f_max
  integer, dimension(:), intent(in) :: mult
  logical, dimension(:), intent(in), optional :: smask, fmask
  real(kind=default) :: amp2
  <Interface amplitude_1>
  complex(kind=default) :: amp
  integer :: s, f
  amp2 = 0
  if (present (smask)) then
    if (present (fmask)) then
      do s = 1, s_max
        if (smask(s)) then
          do f = 1, f_max
            if (fmask(f)) then
              amp = amplitude_1 (k, s, f)
              amp2 = amp2 + amp * conjg (amp) / mult(f)
            end if
          end do
        end do
      end if
    end do
  else
    do s = 1, s_max
      if (smask(s)) then
        do f = 1, f_max
          amp = amplitude_1 (k, s, f)
          amp2 = amp2 + amp * conjg (amp) / mult(f)
        end do
      end if
    end do
  end if
else
  if (present (fmask)) then

```

```

do f = 1, f_max
  if (fmask(f)) then
    do s = 1, s_max
      amp = amplitude_1 (k, s, f)
      amp2 = amp2 + amp * conjg (amp) / mult(f)
    end do
  end if
end do
else
  do s = 1, s_max
    do f = 1, f_max
      amp = amplitude_1 (k, s, f)
      amp2 = amp2 + amp * conjg (amp) / mult(f)
    end do
  end do
end if
end if
end function omega_sum_sqme

```

(Declaration of utility functions)+≡

```

public :: omega_spin_sum_sqme_1_nonzero, omega_sum_sqme_nonzero

```

(Implementation of utility functions)+≡

```

pure subroutine omega_spin_sum_sqme_1_nonzero &
  (amplitude_1, amp2, k, f, zero, n, smask)
  real(kind=default), intent(out) :: amp2
  real(kind=default), dimension(0:,:), intent(in) :: k
  integer, intent(in) :: f
  integer, dimension(:,:), intent(inout) :: zero
  integer, intent(in) :: n
  logical, dimension(:), intent(in), optional :: smask
  <Interface amplitude_1>
  complex(kind=default) :: amp
  real(kind=default) :: dummy
  integer :: s, i
  if (n <= SAMPLE) then
    call omega_sum_sqme_nonzero &
      (amplitude_1, dummy, k, (/ (1, i = 1, size(zero,dim=2)) /), zero, n)
  end if
  amp2 = 0
  if (present (smask)) then
    do s = 1, size(zero,dim=1)
      if (smask(s)) then
        if (zero(s,f) <= REPEAT) then
          amp = amplitude_1 (k, s, f)
          amp2 = amp2 + amp * conjg (amp)
        end if
      end if
    end do
  else
    do s = 1, size(zero,dim=1)
      if (zero(s,f) <= REPEAT) then
        amp = amplitude_1 (k, s, f)
        amp2 = amp2 + amp * conjg (amp)
      end if
    end do
  end if
end if

```

```

        end do
    end if
end subroutine omega_spin_sum_sqme_1_nonzero
<Implementation of utility functions>+≡
pure subroutine omega_sum_sqme_nonzero &
    (amplitude_1, amp2, k, mult, zero, n, smask, fmask)
    real(kind=default), intent(out) :: amp2
    real(kind=default), dimension(0:,:), intent(in) :: k
    integer, dimension(:), intent(in) :: mult
    integer, dimension(:,:), intent(inout) :: zero
    integer, intent(in) :: n
    logical, dimension(:), intent(in), optional :: smask, fmask
    <Interface amplitude_1>
    complex(kind=default) :: amp
    integer :: s, f
    if (n <= SAMPLE) then
        do s = 1, size(zero,dim=1)
            do f = 1, size(zero,dim=2)
                if (zero(s,f) <= REPEAT) then
                    amp = amplitude_1 (k, s, f)
                    if (real (amp * conjg (amp), kind=default) &
                        <= tiny (1.0_default)) then
                        zero(s,f) = zero(s,f) + 1
                    end if
                end if
            end do
        end do
    end if
    amp2 = 0
    if (present (smask)) then
        if (present (fmask)) then
            do s = 1, size(zero,dim=1)
                if (smask(s)) then
                    do f = 1, size(zero,dim=2)
                        if (fmask(f)) then
                            if (zero(s,f) <= REPEAT) then
                                amp = amplitude_1 (k, s, f)
                                amp2 = amp2 + amp * conjg (amp) / mult(f)
                            end if
                        end if
                    end do
                end if
            end do
        else
            do s = 1, size(zero,dim=1)
                if (smask(s)) then
                    do f = 1, size(zero,dim=2)
                        if (zero(s,f) <= REPEAT) then
                            amp = amplitude_1 (k, s, f)
                            amp2 = amp2 + amp * conjg (amp) / mult(f)
                        end if
                    end do
                end if
            end do
        end if
    end do
end do

```

```

        end if
    else
        if (present (fmask)) then
            do f = 1, size(zero,dim=2)
                if (fmask(f)) then
                    do s = 1, size(zero,dim=1)
                        if (zero(s,f) <= REPEAT) then
                            amp = amplitude_1 (k, s, f)
                            amp2 = amp2 + amp * conjg (amp) / mult(f)
                        end if
                    end do
                end if
            end do
        else
            do s = 1, size(zero,dim=1)
                do f = 1, size(zero,dim=2)
                    if (zero(s,f) <= REPEAT) then
                        amp = amplitude_1 (k, s, f)
                        amp2 = amp2 + amp * conjg (amp) / mult(f)
                    end if
                end do
            end do
        end if
    end if
end subroutine omega_sum_sqme_nonzero

<Declaration of utility functions>+≡
public :: omega_amplitude_1_nonzero, omega_amplitude_2_nonzero

<Implementation of utility functions>+≡
pure subroutine omega_amplitude_1_nonzero &
    (amplitude_1, amp, k, s, f, zero, n)
    complex(kind=default), intent(out) :: amp
    real(kind=default), dimension(0:,:), intent(in) :: k
    integer, intent(in) :: s, f
    integer, dimension(:,:), intent(inout) :: zero
    integer, intent(in) :: n
    <Interface amplitude_1>
    integer :: i
    real(kind=default) :: dummy
    if (n <= SAMPLE) then
        call omega_sum_sqme_nonzero &
            (amplitude_1, dummy, k, (/ (1, i = 1, size(zero,dim=2)) /), zero, n)
    end if
    if (zero(s,f) < REPEAT) then
        amp = amplitude_1 (k, s, f)
    else
        amp = 0
    end if
end subroutine omega_amplitude_1_nonzero

<Implementation of utility functions>+≡
pure subroutine omega_amplitude_2_nonzero &
    (amplitude_2, amp, k, s_in, f_in, s_out, f_out, zero, n)
    complex(kind=default), intent(out) :: amp
    real(kind=default), dimension(0:,:), intent(in) :: k

```

```

integer, intent(in) :: s_in, f_in, s_out, f_out
integer, dimension(:,:,:), intent(inout) :: zero
integer, intent(in) :: n
<Interface amplitude_2>
integer :: si, fi, so, fo
if (n <= SAMPLE) then
  do si = 1, size(zero,dim=1)
    do fi = 1, size(zero,dim=2)
      do so = 1, size(zero,dim=3)
        do fo = 1, size(zero,dim=4)
          if (zero(si,fi,so,fo) <= REPEAT) then
            amp = amplitude_2 (k, si, fi, so, fo)
            if (real (amp * conjg (amp), kind=default) &
              <= tiny (1.0_default)) then
              zero(si,fi,so,fo) = zero(si,fi,so,fo) + 1
            end if
          end if
        end do
      end do
    end do
  end do
end if
if (zero(s_in,f_in,s_out,f_out) < REPEAT) then
  amp = amplitude_2 (k, s_in, f_in, s_out, f_out)
else
  amp = 0
end if
end subroutine omega_amplitude_2_nonzero

```

$$\rho \rightarrow \rho' = T \rho T^\dagger \quad (\text{T.94})$$

I.e.

$$\rho'_{ff'} = \sum_{ii'} T_{fi} \rho_{ii'} T_{i'f'}^* \quad (\text{T.95})$$

<Declaration of utility functions>+≡

```
public :: omega_scatter, omega_scatter_nonzero
```

<Implementation of utility functions>+≡

```

pure subroutine omega_scatter (amplitude_2, k, rho_in, rho_out, mult)
real(kind=default), dimension(0:,:), intent(in) :: k
complex(kind=default), dimension(:,:,:), intent(in) :: rho_in
complex(kind=default), dimension(:,:,:), intent(inout) :: rho_out
integer, dimension(:), intent(in) :: mult
<Interface amplitude_2>
integer :: s_in1, s_in2, f_in1, f_in2, s_out1, s_out2, f_out1, f_out2
complex(kind=default), &
  dimension(size(rho_in,dim=1),size(rho_in,dim=2),&
    size(rho_out,dim=1),size(rho_out,dim=2)) :: a
do s_in1 = 1, size(rho_in,dim=1)
  do f_in1 = 1, size(rho_in,dim=2)
    do s_out1 = 1, size(rho_out,dim=1)
      do f_out1 = 1, size(rho_out,dim=2)
        a(s_in1,f_in1,s_out1,f_out1) = &
          amplitude_2 (k, s_in1, f_in1, s_out1, f_out1) &

```

```

        / sqrt (real (mult(f_out1), kind=default))
    end do
end do
end do
do s_out1 = 1, size(rho_out,dim=1)
  do f_out1 = 1, size(rho_out,dim=2)
    do s_out2 = 1, size(rho_out,dim=3)
      do f_out2 = 1, size(rho_out,dim=4)
        rho_out(s_out1,f_out1,s_out2,f_out2) = 0
        do s_in1 = 1, size(rho_in,dim=1)
          do f_in1 = 1, size(rho_in,dim=2)
            do s_in2 = 1, size(rho_in,dim=3)
              do f_in2 = 1, size(rho_in,dim=4)
                rho_out(s_out1,f_out1,s_out2,f_out2) = &
                  rho_out(s_out1,f_out1,s_out2,f_out2) &
                  + a(s_in1,f_in1,s_out1,f_out1) &
                  * rho_in(s_in1,f_in1,s_in2,f_in2) &
                  * conjg (a(s_in2,f_in2,s_out2,f_out2))
              end do
            end do
          end do
        end do
      end do
    end do
  end do
end do
end subroutine omega_scatter

<Interface amplitude_2>≡
interface
  pure function amplitude_2 (k, s_in, f_in, s_out, f_out) result (amp)
    use kinds
    implicit none
    real(kind=default), dimension(0:,:), intent(in) :: k
    integer, intent(in) :: s_in, f_in, s_out, f_out
    complex(kind=default) :: amp
  end function amplitude_2
end interface

<Implementation of utility functions>+≡
pure subroutine omega_scatter_nonzero &
  (amplitude_2, k, rho_in, rho_out, mult, zero, n)
  real(kind=default), dimension(0:,:), intent(in) :: k
  complex(kind=default), dimension(:,:,:), intent(in) :: rho_in
  complex(kind=default), dimension(:,:,:), intent(inout) :: rho_out
  integer, dimension(:), intent(in) :: mult
  integer, dimension(:,:,:), intent(inout) :: zero
  integer, intent(in) :: n
  <Interface amplitude_2 (non zero)>
  integer :: s_in1, s_in2, f_in1, f_in2, s_out1, s_out2, f_out1, f_out2
  complex(kind=default), &
    dimension(size(rho_in,dim=1),size(rho_in,dim=2),&
      size(rho_out,dim=1),size(rho_out,dim=2)) :: a
  do s_in1 = 1, size(rho_in,dim=1)

```

```

do f_in1 = 1, size(rho_in,dim=2)
  do s_out1 = 1, size(rho_out,dim=1)
    do f_out1 = 1, size(rho_out,dim=2)
      call amplitude_2 (a(s_in1,f_in1,s_out1,f_out1), &
                       k, s_in1, f_in1, s_out1, f_out1, zero, n)
      a(s_in1,f_in1,s_out1,f_out1) = &
        a(s_in1,f_in1,s_out1,f_out1) &
        / sqrt (real (mult(f_out1), kind=default))
    end do
  end do
end do
do s_out1 = 1, size(rho_out,dim=1)
  do f_out1 = 1, size(rho_out,dim=2)
    do s_out2 = 1, size(rho_out,dim=3)
      do f_out2 = 1, size(rho_out,dim=4)
        rho_out(s_out1,f_out1,s_out2,f_out2) = 0
        do s_in1 = 1, size(rho_in,dim=1)
          do f_in1 = 1, size(rho_in,dim=2)
            do s_in2 = 1, size(rho_in,dim=3)
              do f_in2 = 1, size(rho_in,dim=4)
                rho_out(s_out1,f_out1,s_out2,f_out2) = &
                  rho_out(s_out1,f_out1,s_out2,f_out2) &
                  + a(s_in1,f_in1,s_out1,f_out1) &
                    * rho_in(s_in1,f_in1,s_in2,f_in2) &
                    * conjg (a(s_in2,f_in2,s_out2,f_out2))
              end do
            end do
          end do
        end do
      end do
    end do
  end do
end do
end subroutine omega_scatter_nonzero
<Interface amplitude_2 (non zero)>≡
interface
  pure subroutine amplitude_2 (amp, k, s_in, f_in, s_out, f_out, zero, n)
    use kinds
    implicit none
    complex(kind=default), intent(out) :: amp
    real(kind=default), dimension(0:,:), intent(in) :: k
    integer, intent(in) :: s_in, f_in, s_out, f_out
    integer, dimension(:,:,:), intent(inout) :: zero
    integer, intent(in) :: n
  end subroutine amplitude_2
end interface

```

$$\rho'_f = \sum_i T_{fi} \rho_i T_{if}^* = \sum_i |T_{fi}|^2 \rho_i \quad (\text{T.96})$$

```

<Declaration of utility functions>+≡
public :: omega_scatter_diagonal, omega_scatter_diagonal_nonzero

```

(Implementation of utility functions)+≡

```

pure subroutine omega_scatter_diagonal &
  (amplitude_2, k, rho_in, rho_out, mult)
  real(kind=default), dimension(0:,:), intent(in) :: k
  real(kind=default), dimension(:,:), intent(in) :: rho_in
  real(kind=default), dimension(:,:), intent(inout) :: rho_out
  integer, dimension(:), intent(in) :: mult
  (Interface amplitude_2)
  integer :: s_in, f_in, s_out, f_out
  complex(kind=default) :: a
  do s_out = 1, size(rho_out,dim=1)
    do f_out = 1, size(rho_out,dim=2)
      rho_out(s_out,f_out) = 0
      do s_in = 1, size(rho_in,dim=1)
        do f_in = 1, size(rho_in,dim=2)
          a = amplitude_2 (k, s_in, f_in, s_out, f_out)
          rho_out(s_out,f_out) = rho_out(s_out,f_out) &
            + rho_in(s_in,f_in) * real (a*conjg(a), kind=default) &
              / mult(f_out)
        end do
      end do
    end do
  end do
end subroutine omega_scatter_diagonal

```

(Implementation of utility functions)+≡

```

pure subroutine omega_scatter_diagonal_nonzero &
  (amplitude_2, k, rho_in, rho_out, mult, zero, n)
  real(kind=default), dimension(0:,:), intent(in) :: k
  real(kind=default), dimension(:,:), intent(in) :: rho_in
  real(kind=default), dimension(:,:), intent(inout) :: rho_out
  integer, dimension(:), intent(in) :: mult
  integer, dimension(:,:,:), intent(inout) :: zero
  integer, intent(in) :: n
  (Interface amplitude_2 (non zero))
  integer :: s_in, f_in, s_out, f_out
  complex(kind=default) :: a
  do s_out = 1, size(rho_out,dim=1)
    do f_out = 1, size(rho_out,dim=2)
      rho_out(s_out,f_out) = 0
      do s_in = 1, size(rho_in,dim=1)
        do f_in = 1, size(rho_in,dim=2)
          call amplitude_2 (a, k, s_in, f_in, s_out, f_out, zero, n)
          rho_out(s_out,f_out) = rho_out(s_out,f_out) &
            + rho_in(s_in,f_in) * real (a*conjg(a), kind=default) &
              / mult(f_out)
        end do
      end do
    end do
  end do
end subroutine omega_scatter_diagonal_nonzero

```


Flavor Summation

Interface to WHIZARD here ...

<Declaration of utility functions>+≡

<Implementation of utility functions>+≡

*T.16.4 Obsolescent Summation**Spin/Helicity Summation*

<Declaration of utility functions>+≡

```
public :: omega_sum, omega_sum_nonzero, omega_nonzero
private :: state_index
```

<Implementation of 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 $(-1, \dots, -1, -1)$, $(-1, \dots, -1, 0)$, $(-1, \dots, -1, 1)$, $(-1, \dots, 0, -1)$, ..., $(1, \dots, 1, 0)$, $(1, \dots, 1, 1)$:

<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 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

```

(Implementation of utility functions)+≡

```

pure subroutine omega_sum_nonzero (sigma, omega, p, zero, n, states, fixed)
    real(kind=default), intent(out) :: sigma
    real(kind=default), dimension(0:,:), intent(in) :: p
    integer, dimension(:), intent(inout) :: zero
    integer, intent(in) :: n

```

```

integer, dimension(:), intent(in), optional :: states, fixed
<interface for O'Mega Amplitude>
integer, dimension(size(p,dim=2)) :: s, nstates
integer :: j, k
complex(kind=default) :: a
real(kind=default) :: a2
if (present (states)) then
    nstates = states
else
    nstates = 2
end if
sigma = 0
s = -1
k = 1
sum_spins: do
    if (zero (k) < REPEAT) then
        if (present (fixed)) then
            a = omega (p, merge (fixed, s, mask = nstates == 0))
        else
            a = omega (p, s)
        end if
        a2 = a * conjg(a)
        if (n <= SAMPLE .and. a2 <= tiny (1.0_default)) then
            zero (k) = zero (k) + 1
        end if
        sigma = sigma + a2
    end if
    k = k + 1
    <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 subroutine omega_sum_nonzero

<Declaration of utility functions>+≡
public :: num_states

<Implementation of 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

<Implementation of utility functions>+≡
pure subroutine omega_nonzero (a, omega, p, s, zero, n, states)
complex(kind=default), intent(out) :: a
real(kind=default), dimension(0:,:), intent(in) :: p
integer, dimension(:), intent(in) :: s
integer, dimension(:), intent(inout) :: zero
integer, intent(in) :: n
integer, dimension(:), intent(in), optional :: states

```

```

<interface for O'Mega Amplitude>
real(kind=default) :: dummy
if (n < SAMPLE) then
    call omega_sum_nonzero (dummy, omega, p, zero, n, states)
end if
if (zero (state_index (s, states)) < REPEAT) then
    a = omega (p, s)
else
    a = 0
end if
end subroutine omega_nonzero

```

T.17 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_utils
    public
end module omega95

```

T.18 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_utils
    public
end module omega95_bispinors

```

T.19 Testing

```

<omega_testtools.f90>≡
<Copyleft>

```

```

module omega_testtools
  use kinds
  implicit none
  private
  public :: print_matrix
  public :: expect
  real(kind=default), parameter, private :: TOLERANCE = 1.0e8
  <Declare expect>
contains
  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
  <Implement expect>
end module omega_testtools

<Declare expect>≡
interface expect
  module procedure expect_integer, expect_real, expect_complex, &
    expect_double_integer, expect_complex_integer, expect_complex_real
end interface
private :: expect_integer, expect_real, expect_complex, &
  expect_double_integer, expect_complex_integer, expect_complex_real

<Implement expect>≡
subroutine expect_integer (x, x0, msg)
  integer, intent(in) :: x, x0
  character(len=*), intent(in) :: msg
  if (x == x0) then
    print *, msg, " passed"
  else
    print *, msg, " FAILED: expected ", x0, " got ", x
  end if
end subroutine expect_integer

<Implement expect>+≡
subroutine expect_real (x, x0, msg)
  real(kind=default), intent(in) :: x, x0
  character(len=*), intent(in) :: msg
  if (x == x0) then
    print *, msg, " passed exactly"
  else if (abs (x - x0) <= epsilon (x)) then
    print *, msg, " passed at machine precision"
  else if (abs (x - x0) <= TOLERANCE * epsilon (x)) then
    print *, msg, " passed at", &
      ceiling (abs (x - x0) / epsilon (x)), "* machine precision"
  else
    print *, msg, " FAILED: expected ", x0, " got ", x, " (", &
      (x - x0) / epsilon (x), " epsilon)"
  end if
end subroutine expect_real

<Implement expect>+≡

```

```

subroutine expect_complex (x, x0, msg)
  complex(kind=default), intent(in) :: x, x0
  character(len=*), intent(in) :: msg
  if (x == x0) then
    print *, msg, " passed exactly"
  else if (abs (x - x0) <= epsilon (real(x))) then
    print *, msg, " passed at machine precision"
  else if (abs (x - x0) <= TOLERANCE * epsilon (real(x))) then
    print *, msg, " passed at", &
      ceiling (abs (x - x0) / epsilon (real(x))), "* machine precision"
  else
    print *, msg, " FAILED: expected ", x0, " got ", x, " (", &
      (x - x0) / epsilon (real(x)), " epsilon)"
  end if
end subroutine expect_complex

<Implement expect>+≡
subroutine expect_double_integer (x, x0, msg)
  real(kind=default), intent(in) :: x
  integer, intent(in) :: x0
  character(len=*), intent(in) :: msg
  call expect_real (x, real (x0, kind=default), msg)
end subroutine expect_double_integer

<Implement expect>+≡
subroutine expect_complex_integer (x, x0, msg)
  complex(kind=default), intent(in) :: x
  integer, intent(in) :: x0
  character(len=*), intent(in) :: msg
  call expect_complex (x, cmplx (x0, kind=default), msg)
end subroutine expect_complex_integer

<Implement expect>+≡
subroutine expect_complex_real (x, x0, msg)
  complex(kind=default), intent(in) :: x
  real(kind=default), intent(in) :: x0
  character(len=*), intent(in) :: msg
  call expect_complex (x, cmplx (x0, kind=default), msg)
end subroutine expect_complex_real

<test_omega.f90>≡
<Copyleft>
program test_omega
  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) :: one
  type(momentum) :: p, q
  type(vector) :: vp, vq, vtest
  type(tensor) :: ttest
  integer, dimension(8) :: date_time
  integer :: rsize
  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
one = 1
m = 13
pabs = 42
qabs = 137
call random_number (r)
vtest%t = cmplx (10.0_default * r(0))
vtest%x(1:3) = cmplx (10.0_default * r(1:3))
ttest = vtest.tprod.vtest
call random_momentum (p, pabs, m)
call random_momentum (q, qabs, m)
vp = p
vq = q
<Test omega95>
end program test_omega

<Test omega95>≡
print *, "*** Checking the equations of motion ***:"
call expect (abs(f_vf(one, vp, u(m, p, +1)) - m * u(m, p, +1)), 0, "| [p-m] u(+) |=0")
call expect (abs(f_vf(one, vp, u(m, p, -1)) - m * u(m, p, -1)), 0, "| [p-m] u(-) |=0")
call expect (abs(f_vf(one, vp, v(m, p, +1)) + m * v(m, p, +1)), 0, "| [p+m] v(+) |=0")
call expect (abs(f_vf(one, vp, v(m, p, -1)) + m * v(m, p, -1)), 0, "| [p+m] v(-) |=0")
call expect (abs(f_fv(one, ubar(m, p, +1), vp) - m * ubar(m, p, +1)), 0, "| ubar(+) [p-m] |=0")
call expect (abs(f_fv(one, ubar(m, p, -1), vp) - m * ubar(m, p, -1)), 0, "| ubar(-) [p-m] |=0")
call expect (abs(f_fv(one, vbar(m, p, +1), vp) + m * vbar(m, p, +1)), 0, "| vbar(+) [p+m] |=0")
call expect (abs(f_fv(one, vbar(m, p, -1), vp) + m * vbar(m, p, -1)), 0, "| vbar(-) [p+m] |=0")

<Test omega95>+≡
print *, "*** Checking the normalization ***:"
call expect (ubar(m, p, +1) * u(m, p, +1), +2 * m, "ubar(+) * u(+) = +2m")
call expect (ubar(m, p, -1) * u(m, p, -1), +2 * m, "ubar(-) * u(-) = +2m")
call expect (vbar(m, p, +1) * v(m, p, +1), -2 * m, "vbar(+) * v(+) = -2m")
call expect (vbar(m, p, -1) * v(m, p, -1), -2 * m, "vbar(-) * v(-) = -2m")
call expect (ubar(m, p, +1) * v(m, p, +1), 0, "ubar(+) * v(+) = 0")
call expect (ubar(m, p, -1) * v(m, p, -1), 0, "ubar(-) * v(-) = 0")
call expect (vbar(m, p, +1) * u(m, p, +1), 0, "vbar(+) * u(+) = 0")
call expect (vbar(m, p, -1) * u(m, p, -1), 0, "vbar(-) * u(-) = 0")

<Test omega95>+≡
print *, "*** Checking the currents ***:"
call expect (abs(v_ff(one, ubar(m, p, +1), u(m, p, +1)) - 2 * vp), 0, "ubar(+).V.u(+) = 2p")
call expect (abs(v_ff(one, ubar(m, p, -1), u(m, p, -1)) - 2 * vp), 0, "ubar(-).V.u(-) = 2p")
call expect (abs(v_ff(one, vbar(m, p, +1), v(m, p, +1)) - 2 * vp), 0, "vbar(+).V.v(+) = 2p")
call expect (abs(v_ff(one, vbar(m, p, -1), v(m, p, -1)) - 2 * vp), 0, "vbar(-).V.v(-) = 2p")

<Test omega95>+≡
print *, "*** Checking current conservation ***:"
call expect ((vp - vq) * v_ff(one, ubar(m, p, +1), u(m, q, +1)), 0, "d(ubar(+).V.u(+)) = 0")
call expect ((vp - vq) * v_ff(one, ubar(m, p, -1), u(m, q, -1)), 0, "d(ubar(-).V.u(-)) = 0")
call expect ((vp - vq) * v_ff(one, vbar(m, p, +1), v(m, q, +1)), 0, "d(vbar(+).V.v(+)) = 0")
call expect ((vp - vq) * v_ff(one, vbar(m, p, -1), v(m, q, -1)), 0, "d(vbar(-).V.v(-)) = 0")

<Test omega95>+≡
if (m == 0) then
  print *, "*** Checking axial current conservation ***:"

```

```

    call expect ((vp-vq)*a_ff(one,ubar(m,p,+1),u(m,q,+1)), 0, "d(ubar(+).A.u(+))=0")
    call expect ((vp-vq)*a_ff(one,ubar(m,p,-1),u(m,q,-1)), 0, "d(ubar(-).A.u(-))=0")
    call expect ((vp-vq)*a_ff(one,vbar(m,p,+1),v(m,q,+1)), 0, "d(vbar(+).A.v(+))=0")
    call expect ((vp-vq)*a_ff(one,vbar(m,p,-1),v(m,q,-1)), 0, "d(vbar(-).A.v(-))=0")
end if

<Test omega95>+≡
print *, "*** Checking polarisation vectors: ***"
call expect (conjg(eps(m,p, 1))*eps(m,p, 1), -1, "e( 1).e( 1)=-1")
call expect (conjg(eps(m,p, 1))*eps(m,p,-1), 0, "e( 1).e(-1)= 0")
call expect (conjg(eps(m,p,-1))*eps(m,p, 1), 0, "e(-1).e( 1)= 0")
call expect (conjg(eps(m,p,-1))*eps(m,p,-1), -1, "e(-1).e(-1)=-1")
call expect (
    p*eps(m,p, 1), 0, "    p.e( 1)= 0")
call expect (
    p*eps(m,p,-1), 0, "    p.e(-1)= 0")
if (m > 0) then
    call expect (conjg(eps(m,p, 1))*eps(m,p, 0), 0, "e( 1).e( 0)= 0")
    call expect (conjg(eps(m,p, 0))*eps(m,p, 1), 0, "e( 0).e( 1)= 0")
    call expect (conjg(eps(m,p, 0))*eps(m,p, 0), -1, "e( 0).e( 0)=-1")
    call expect (conjg(eps(m,p, 0))*eps(m,p,-1), 0, "e( 0).e(-1)= 0")
    call expect (conjg(eps(m,p,-1))*eps(m,p, 0), 0, "e(-1).e( 0)= 0")
    call expect (
        p*eps(m,p, 0), 0, "    p.e( 0)= 0")
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)" )
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)" )
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)" )
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)" )
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)" )
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)" )
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'" )

```

$$\frac{1}{2}[x \wedge y]_{\mu\nu}^*[x \wedge y]^{\mu\nu} = \frac{1}{2}(x_{\mu}^*y_{\nu}^* - x_{\nu}^*y_{\mu}^*)(x^{\mu}y^{\nu} - x^{\nu}y^{\mu}) = (x^*x)(y^*y) - (x^*y)(y^*x) \quad (\text{T.97})$$

```

<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")
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")

```

i. e.

$$\frac{1}{2}[p \wedge \epsilon(p, i)]_{\mu\nu}^*[p \wedge \epsilon(p, j)]^{\mu\nu} = -p^2\delta_{ij} \quad (\text{T.98})$$

```

<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")
call expect (conjg(p.wedge.eps(m,p, 1))*(p.wedge.eps(m,p,-1)), 0, &
    "[p,e( 1)].[p,e(-1)]=0")
call expect (conjg(p.wedge.eps(m,p,-1))*(p.wedge.eps(m,p, 1)), 0, &
    "[p,e(-1)].[p,e( 1)]=0")
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")
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")
    call expect (conjg(p.wedge.eps(m,p, 0))*(p.wedge.eps(m,p, 1)), 0, &
        "[p,e( 0)].[p,e( 1)]=0")
    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")
    call expect (conjg(p.wedge.eps(m,p, 0))*(p.wedge.eps(m,p,-1)), 0, &
        "[p,e( 1)].[p,e(-1)]=0")
    call expect (conjg(p.wedge.eps(m,p,-1))*(p.wedge.eps(m,p, 0)), 0, &
        "[p,e(-1)].[p,e( 0)]=0")
end if
also

```

$$[x \wedge y]_{\mu\nu} z^\nu = x_\mu(yz) - y_\mu(xz) \quad (\text{T.99})$$

$$z_\mu[x \wedge y]^{\mu\nu} = (zx)y^\nu - (zy)x^\nu \quad (\text{T.100})$$

$\langle \text{Test omega95} \rangle + \equiv$

```

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)")
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)")
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)")
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)")
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)")
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)")

```

$\langle \text{Test omega95} \rangle + \equiv$

```

print *, "*** Checking polarisation tensors: ***"
call expect (conjg(eps2(m,p, 2))*eps2(m,p, 2), 1, "e2( 2).e2( 2)=1")
call expect (conjg(eps2(m,p, 2))*eps2(m,p,-2), 0, "e2( 2).e2(-2)=0")
call expect (conjg(eps2(m,p,-2))*eps2(m,p, 2), 0, "e2(-2).e2( 2)=0")
call expect (conjg(eps2(m,p,-2))*eps2(m,p,-2), 1, "e2(-2).e2(-2)=1")
if (m > 0) then
    call expect (conjg(eps2(m,p, 2))*eps2(m,p, 1), 0, "e2( 2).e2( 1)=0")
    call expect (conjg(eps2(m,p, 2))*eps2(m,p, 0), 0, "e2( 2).e2( 0)=0")
    call expect (conjg(eps2(m,p, 2))*eps2(m,p,-1), 0, "e2( 2).e2(-1)=0")
    call expect (conjg(eps2(m,p, 1))*eps2(m,p, 2), 0, "e2( 1).e2( 2)=0")
    call expect (conjg(eps2(m,p, 1))*eps2(m,p, 1), 1, "e2( 1).e2( 1)=1")
    call expect (conjg(eps2(m,p, 1))*eps2(m,p, 0), 0, "e2( 1).e2( 0)=0")
    call expect (conjg(eps2(m,p, 1))*eps2(m,p,-1), 0, "e2( 1).e2(-1)=0")
    call expect (conjg(eps2(m,p, 1))*eps2(m,p,-2), 0, "e2( 1).e2(-2)=0")
    call expect (conjg(eps2(m,p, 0))*eps2(m,p, 2), 0, "e2( 0).e2( 2)=0")

```

```

call expect (conjg(eps2(m,p, 0))*eps2(m,p, 1), 0, "e2( 0).e2( 1)=0")
call expect (conjg(eps2(m,p, 0))*eps2(m,p, 0), 1, "e2( 0).e2( 0)=1")
call expect (conjg(eps2(m,p, 0))*eps2(m,p,-1), 0, "e2( 0).e2(-1)=0")
call expect (conjg(eps2(m,p, 0))*eps2(m,p,-2), 0, "e2( 0).e2(-2)=0")
call expect (conjg(eps2(m,p,-1))*eps2(m,p, 2), 0, "e2(-1).e2( 2)=0")
call expect (conjg(eps2(m,p,-1))*eps2(m,p, 1), 0, "e2(-1).e2( 1)=0")
call expect (conjg(eps2(m,p,-1))*eps2(m,p, 0), 0, "e2(-1).e2( 0)=0")
call expect (conjg(eps2(m,p,-1))*eps2(m,p,-1), 1, "e2(-1).e2(-1)=1")
call expect (conjg(eps2(m,p,-1))*eps2(m,p,-2), 0, "e2(-1).e2(-2)=0")
call expect (conjg(eps2(m,p,-2))*eps2(m,p, 1), 0, "e2(-2).e2( 1)=0")
call expect (conjg(eps2(m,p,-2))*eps2(m,p, 0), 0, "e2(-2).e2( 0)=0")
call expect (conjg(eps2(m,p,-2))*eps2(m,p,-1), 0, "e2(-2).e2(-1)=0")
end if

<Test omega95>+=
call expect (      abs(p*eps2(m,p, 2) ), 0, " |p.e2( 2)| =0")
call expect (      abs(eps2(m,p, 2)*p), 0, " |e2( 2).p|=0")
call expect (      abs(p*eps2(m,p,-2) ), 0, " |p.e2(-2)| =0")
call expect (      abs(eps2(m,p,-2)*p), 0, " |e2(-2).p|=0")
if (m > 0) then
  call expect (      abs(p*eps2(m,p, 1) ), 0, " |p.e2( 1)| =0")
  call expect (      abs(eps2(m,p, 1)*p), 0, " |e2( 1).p|=0")
  call expect (      abs(p*eps2(m,p, 0) ), 0, " |p.e2( 0)| =0")
  call expect (      abs(eps2(m,p, 0)*p), 0, " |e2( 0).p|=0")
  call expect (      abs(p*eps2(m,p,-1) ), 0, " |p.e2(-1)| =0")
  call expect (      abs(eps2(m,p,-1)*p), 0, " |e2(-1).p|=0")
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")
call expect (abs(p * eps2(m,p,1)), 0, "p.e(+1)=0")
call expect (abs(p * eps2(m,p,0)), 0, "p.e( 0)=0")
call expect (abs(p * eps2(m,p,-1)), 0, "p.e(-1)=0")
call expect (abs(p * eps2(m,p,-2)), 0, "p.e(-2)=0")
call expect (abs(trace(eps2 (m,p,2))), 0, "Tr[e(+2)]=0")
call expect (abs(trace(eps2 (m,p,1))), 0, "Tr[e(+1)]=0")
call expect (abs(trace(eps2 (m,p,0))), 0, "Tr[e( 0)]=0")
call expect (abs(trace(eps2 (m,p,-1))), 0, "Tr[e(-1)]=0")
call expect (abs(trace(eps2 (m,p,-2))), 0, "Tr[e(-2)]=0")
call expect (abs(eps2(m,p,2) * eps2(m,p,2)), 1, &
  "e(2).e(2) = 1")
call expect (abs(eps2(m,p,2) * eps2(m,p,1)), 0, &
  "e(2).e(1) = 0")
call expect (abs(eps2(m,p,2) * eps2(m,p,0)), 0, &
  "e(2).e(0) = 0")
call expect (abs(eps2(m,p,2) * eps2(m,p,-1)), 0, &
  "e(2).e(-1) = 0")
call expect (abs(eps2(m,p,2) * eps2(m,p,-2)), 0, &
  "e(2).e(-2) = 0")
call expect (abs(eps2(m,p,1) * eps2(m,p,1)), 1, &
  "e(1).e(1) = 1")
call expect (abs(eps2(m,p,1) * eps2(m,p,0)), 0, &
  "e(1).e(0) = 0")
call expect (abs(eps2(m,p,1) * eps2(m,p,-1)), 0, &

```

```

    "e(1).e(-1) = 0")
call expect (abs(eps2(m,p,1) * eps2(m,p,-2)), 0, &
    "e(1).e(-2) = 0")
call expect (abs(eps2(m,p,0) * eps2(m,p,0)), 1, &
    "e(0).e(0) = 1")
call expect (abs(eps2(m,p,0) * eps2(m,p,-1)), 0, &
    "e(0).e(-1) = 0")
call expect (abs(eps2(m,p,0) * eps2(m,p,-2)), 0, &
    "e(0).e(-2) = 0")
call expect (abs(eps2(m,p,-1) * eps2(m,p,-1)), 1, &
    "e(-1).e(-1) = 1")
call expect (abs(eps2(m,p,-1) * eps2(m,p,-2)), 0, &
    "e(-1).e(-2) = 0")
call expect (abs(eps2(m,p,-2) * eps2(m,p,-2)), 1, &
    "e(-2).e(-2) = 1")

<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)")
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)")
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)")
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)")
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)")
call expect (abs(p * (cmplx (p*p - m**2, m*w, kind=default) * &
    pr_tensor(p,m,w,ttest))), 0, "p.pr.ttest")

<test_omega_bispinors.f90>=
<Copyleft>
program test_omega_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) :: one, two
    type(momentum) :: p, q, t, z, p_0
    type(vector) :: vp, vq, vt, vz
    type(vectorspinor) :: testv
    call random_seed ()
    one = 1
    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(1) = (0.0_default, 2.0_default)
testv%psi(4)%a(1) = (1.0_default, 0.0_default)
testv%psi(4)%a(1) = (3.0_default, 0.0_default)
vp = p
vq = q
vt = t
vz = z
<Test omega95_bispinors>
end program test_omega_bispinors

<Test omega95_bispinors>≡
print *, "*** Checking the equations of motion ***:"
call expect (abs(f_vf(one, vp, u(m, p, +1)) - m * u(m, p, +1)), 0, "[p-m]u(+)=0")
call expect (abs(f_vf(one, vp, u(m, p, -1)) - m * u(m, p, -1)), 0, "[p-m]u(-)=0")
call expect (abs(f_vf(one, vp, v(m, p, +1)) + m * v(m, p, +1)), 0, "[p+m]v(+)=0")
call expect (abs(f_vf(one, vp, v(m, p, -1)) + m * v(m, p, -1)), 0, "[p+m]v(-)=0")

<Test omega95_bispinors>+≡
print *, "*** Checking the normalization ***:"
call expect (s_ff(one, v(m, p, +1), u(m, p, +1)), +2*m, "ubar(+)*u(+)=2m")
call expect (s_ff(one, v(m, p, -1), u(m, p, -1)), +2*m, "ubar(-)*u(-)=2m")
call expect (s_ff(one, u(m, p, +1), v(m, p, +1)), -2*m, "vbar(+)*v(+)=2m")
call expect (s_ff(one, u(m, p, -1), v(m, p, -1)), -2*m, "vbar(-)*v(-)=2m")
call expect (s_ff(one, v(m, p, +1), v(m, p, +1)), 0, "ubar(+)*v(+)=0")
call expect (s_ff(one, v(m, p, -1), v(m, p, -1)), 0, "ubar(-)*v(-)=0")
call expect (s_ff(one, u(m, p, +1), u(m, p, +1)), 0, "vbar(+)*u(+)=0")
call expect (s_ff(one, u(m, p, -1), u(m, p, -1)), 0, "vbar(-)*u(-)=0")

<Test omega95_bispinors>+≡
print *, "*** Checking the currents ***:"
call expect (abs(v_ff(one, v(m, p, +1), u(m, p, +1)) - 2*vp), 0, "ubar(+).V.u(+)=2p")
call expect (abs(v_ff(one, v(m, p, -1), u(m, p, -1)) - 2*vp), 0, "ubar(-).V.u(-)=2p")
call expect (abs(v_ff(one, u(m, p, +1), v(m, p, +1)) - 2*vp), 0, "vbar(+).V.v(+)=2p")
call expect (abs(v_ff(one, u(m, p, -1), v(m, p, -1)) - 2*vp), 0, "vbar(-).V.v(-)=2p")

<Test omega95_bispinors>+≡
print *, "*** Checking current conservation ***:"
call expect ((vp-vq)*v_ff(one, v(m, p, +1), u(m, q, +1)), 0, "d(ubar(+).V.u(+))=0")
call expect ((vp-vq)*v_ff(one, v(m, p, -1), u(m, q, -1)), 0, "d(ubar(-).V.u(-))=0")
call expect ((vp-vq)*v_ff(one, u(m, p, +1), v(m, q, +1)), 0, "d(vbar(+).V.v(+))=0")
call expect ((vp-vq)*v_ff(one, u(m, p, -1), v(m, q, -1)), 0, "d(vbar(-).V.v(-))=0")

```

```

<Test omega95_bispinors>+≡
  if (m == 0) then
    print *, "*** Checking axial current conservation ***:"
    call expect ((vp-vq)*a_ff(one,v(m,p,+1),u(m,q,+1)), 0, "d(ubar(+).A.u(+))=0")
    call expect ((vp-vq)*a_ff(one,v(m,p,-1),u(m,q,-1)), 0, "d(ubar(-).A.u(-))=0")
    call expect ((vp-vq)*a_ff(one,u(m,p,+1),v(m,q,+1)), 0, "d(vbar(+).A.v(+))=0")
    call expect ((vp-vq)*a_ff(one,u(m,p,-1),v(m,q,-1)), 0, "d(vbar(-).A.v(-))=0")
  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")
  call expect (conjg(eps(m,p, 1))*eps(m,p,-1), 0, "e( 1).e(-1)= 0")
  call expect (conjg(eps(m,p,-1))*eps(m,p, 1), 0, "e(-1).e( 1)= 0")
  call expect (conjg(eps(m,p,-1))*eps(m,p,-1), -1, "e(-1).e(-1)=-1")
  call expect (p*eps(m,p, 1), 0, "p.e( 1)= 0")
  call expect (p*eps(m,p,-1), 0, "p.e(-1)= 0")
  if (m > 0) then
    call expect (conjg(eps(m,p, 1))*eps(m,p, 0), 0, "e( 1).e( 0)= 0")
    call expect (conjg(eps(m,p, 0))*eps(m,p, 1), 0, "e( 0).e( 1)= 0")
    call expect (conjg(eps(m,p, 0))*eps(m,p, 0), -1, "e( 0).e( 0)=-1")
    call expect (conjg(eps(m,p, 0))*eps(m,p,-1), 0, "e( 0).e(-1)= 0")
    call expect (conjg(eps(m,p,-1))*eps(m,p, 0), 0, "e(-1).e( 0)= 0")
    call expect (p*eps(m,p, 0), 0, "p.e( 0)= 0")
  end if

<Test omega95_bispinors>+≡
  print *, "*** Checking polarization vectorspinors: ***"
  call expect (abs(p * ueps(m, p, 2)), 0, "p.ueps ( 2)= 0")
  call expect (abs(p * ueps(m, p, 1)), 0, "p.ueps ( 1)= 0")
  call expect (abs(p * ueps(m, p, -1)), 0, "p.ueps (-1)= 0")
  call expect (abs(p * ueps(m, p, -2)), 0, "p.ueps (-2)= 0")
  call expect (abs(p * veps(m, p, 2)), 0, "p.veps ( 2)= 0")
  call expect (abs(p * veps(m, p, 1)), 0, "p.veps ( 1)= 0")
  call expect (abs(p * veps(m, p, -1)), 0, "p.veps (-1)= 0")
  call expect (abs(p * veps(m, p, -2)), 0, "p.veps (-2)= 0")
  print *, "*** in the rest frame ***"
  call expect (abs(p_0 * ueps(m, p_0, 2)), 0, "p0.ueps ( 2)= 0")
  call expect (abs(p_0 * ueps(m, p_0, 1)), 0, "p0.ueps ( 1)= 0")
  call expect (abs(p_0 * ueps(m, p_0, -1)), 0, "p0.ueps (-1)= 0")
  call expect (abs(p_0 * ueps(m, p_0, -2)), 0, "p0.ueps (-2)= 0")
  call expect (abs(p_0 * veps(m, p_0, 2)), 0, "p0.veps ( 2)= 0")
  call expect (abs(p_0 * veps(m, p_0, 1)), 0, "p0.veps ( 1)= 0")
  call expect (abs(p_0 * veps(m, p_0, -1)), 0, "p0.veps (-1)= 0")
  call expect (abs(p_0 * veps(m, p_0, -2)), 0, "p0.veps (-2)= 0")

<Test omega95_bispinors>+≡
  print *, "*** Checking the irreducibility condition: ***"
  call expect (abs(f_potgr (one, one, ueps(m, p, 2))), 0, "g.ueps ( 2)")
  call expect (abs(f_potgr (one, one, ueps(m, p, 1))), 0, "g.ueps ( 1)")
  call expect (abs(f_potgr (one, one, ueps(m, p, -1))), 0, "g.ueps (-1)")
  call expect (abs(f_potgr (one, one, ueps(m, p, -2))), 0, "g.ueps (-2)")
  call expect (abs(f_potgr (one, one, veps(m, p, 2))), 0, "g.veps ( 2)")
  call expect (abs(f_potgr (one, one, veps(m, p, 1))), 0, "g.veps ( 1)")
  call expect (abs(f_potgr (one, one, veps(m, p, -1))), 0, "g.veps (-1)")
  call expect (abs(f_potgr (one, one, veps(m, p, -2))), 0, "g.veps (-2)")

```

```

print *, "*** in the rest frame ***"
call expect (abs(f_potgr (one, one, ueps(m, p_0, 2))), 0, "g.ueps ( 2)")
call expect (abs(f_potgr (one, one, ueps(m, p_0, 1))), 0, "g.ueps ( 1)")
call expect (abs(f_potgr (one, one, ueps(m, p_0, -1))), 0, "g.ueps (-1)")
call expect (abs(f_potgr (one, one, ueps(m, p_0, -2))), 0, "g.ueps (-2)")
call expect (abs(f_potgr (one, one, veps(m, p_0, 2))), 0, "g.veps ( 2)")
call expect (abs(f_potgr (one, one, veps(m, p_0, 1))), 0, "g.veps ( 1)")
call expect (abs(f_potgr (one, one, veps(m, p_0, -1))), 0, "g.veps (-1)")
call expect (abs(f_potgr (one, one, veps(m, p_0, -2))), 0, "g.veps (-2)")

<Test omega95_bispinors>+≡
print *, "*** Testing vectorspinor normalization ***"
call expect (veps(m,p, 2)*ueps(m,p, 2), -2*m, "ueps( 2).ueps( 2)= -2m")
call expect (veps(m,p, 1)*ueps(m,p, 1), -2*m, "ueps( 1).ueps( 1)= -2m")
call expect (veps(m,p,-1)*ueps(m,p,-1), -2*m, "ueps(-1).ueps(-1)= -2m")
call expect (veps(m,p,-2)*ueps(m,p,-2), -2*m, "ueps(-2).ueps(-2)= -2m")
call expect (ueps(m,p, 2)*veps(m,p, 2), 2*m, "veps( 2).veps( 2)= +2m")
call expect (ueps(m,p, 1)*veps(m,p, 1), 2*m, "veps( 1).veps( 1)= +2m")
call expect (ueps(m,p,-1)*veps(m,p,-1), 2*m, "veps(-1).veps(-1)= +2m")
call expect (ueps(m,p,-2)*veps(m,p,-2), 2*m, "veps(-2).veps(-2)= +2m")
call expect (ueps(m,p, 2)*ueps(m,p, 2), 0, "ueps( 2).veps( 2)= 0")
call expect (ueps(m,p, 1)*ueps(m,p, 1), 0, "ueps( 1).veps( 1)= 0")
call expect (ueps(m,p,-1)*ueps(m,p,-1), 0, "ueps(-1).veps(-1)= 0")
call expect (ueps(m,p,-2)*ueps(m,p,-2), 0, "ueps(-2).veps(-2)= 0")
call expect (veps(m,p, 2)*veps(m,p, 2), 0, "veps( 2).ueps( 2)= 0")
call expect (veps(m,p, 1)*veps(m,p, 1), 0, "veps( 1).ueps( 1)= 0")
call expect (veps(m,p,-1)*veps(m,p,-1), 0, "veps(-1).ueps(-1)= 0")
call expect (veps(m,p,-2)*veps(m,p,-2), 0, "veps(-2).ueps(-2)= 0")
print *, "*** in the rest frame ***"
call expect (veps(m,p_0, 2)*ueps(m,p_0, 2), -2*m, "ueps( 2).ueps( 2)= -2m")
call expect (veps(m,p_0, 1)*ueps(m,p_0, 1), -2*m, "ueps( 1).ueps( 1)= -2m")
call expect (veps(m,p_0,-1)*ueps(m,p_0,-1), -2*m, "ueps(-1).ueps(-1)= -2m")
call expect (veps(m,p_0,-2)*ueps(m,p_0,-2), -2*m, "ueps(-2).ueps(-2)= -2m")
call expect (ueps(m,p_0, 2)*veps(m,p_0, 2), 2*m, "veps( 2).veps( 2)= +2m")
call expect (ueps(m,p_0, 1)*veps(m,p_0, 1), 2*m, "veps( 1).veps( 1)= +2m")
call expect (ueps(m,p_0,-1)*veps(m,p_0,-1), 2*m, "veps(-1).veps(-1)= +2m")
call expect (ueps(m,p_0,-2)*veps(m,p_0,-2), 2*m, "veps(-2).veps(-2)= +2m")
call expect (ueps(m,p_0, 2)*ueps(m,p_0, 2), 0, "ueps( 2).veps( 2)= 0")
call expect (ueps(m,p_0, 1)*ueps(m,p_0, 1), 0, "ueps( 1).veps( 1)= 0")
call expect (ueps(m,p_0,-1)*ueps(m,p_0,-1), 0, "ueps(-1).veps(-1)= 0")
call expect (ueps(m,p_0,-2)*ueps(m,p_0,-2), 0, "ueps(-2).veps(-2)= 0")
call expect (veps(m,p_0, 2)*veps(m,p_0, 2), 0, "veps( 2).ueps( 2)= 0")
call expect (veps(m,p_0, 1)*veps(m,p_0, 1), 0, "veps( 1).ueps( 1)= 0")
call expect (veps(m,p_0,-1)*veps(m,p_0,-1), 0, "veps(-1).ueps(-1)= 0")
call expect (veps(m,p_0,-2)*veps(m,p_0,-2), 0, "veps(-2).ueps(-2)= 0")

<Test omega95_bispinors>+≡
print *, "*** Majorana properties of gravitino vertices: ***"
call expect (abs(u (m,q,1) * f_sgr (one, one, ueps(m,p,2), t) + &
    ueps(m,p,2) * gr_sf(one,one,u(m,q,1),t)), 0, "f_sgr + gr_sf = 0")
!!! call expect (abs(u (m,q,-1) * f_sgr (one, one, ueps(m,p,2), t) + &
    ueps(m,p,2) * gr_sf(one,one,u(m,q,-1),t)), 0, "f_sgr + gr_sf = 0")
!!! call expect (abs(u (m,q,1) * f_sgr (one, one, ueps(m,p,1), t) + &
    ueps(m,p,1) * gr_sf(one,one,u(m,q,1),t)), 0, "f_sgr + gr_sf = 0")
!!! call expect (abs(u (m,q,-1) * f_sgr (one, one, ueps(m,p,1), t) + &
    ueps(m,p,1) * gr_sf(one,one,u(m,q,-1),t)), 0, "f_sgr + gr_sf = 0")

```

```

!!! ueps(m,p,1) * gr_sf(one,one,u(m,q,-1),t)), 0, "f_sgr + gr_sf = 0")
!!! call expect (abs(u (m,q,1) * f_sgr (one, one, ueps(m,p,-1), t) + &
!!! ueps(m,p,-1) * gr_sf(one,one,u(m,q,1),t))), 0, "f_sgr + gr_sf = 0")
!!! call expect (abs(u (m,q,-1) * f_sgr (one, one, ueps(m,p,-1), t) + &
!!! ueps(m,p,-1) * gr_sf(one,one,u(m,q,-1),t))), 0, "f_sgr + gr_sf = 0")
!!! call expect (abs(u (m,q,1) * f_sgr (one, one, ueps(m,p,-2), t) + &
!!! ueps(m,p,-2) * gr_sf(one,one,u(m,q,1),t))), 0, "f_sgr + gr_sf = 0")
!!! call expect (abs(u (m,q,-1) * f_sgr (one, one, ueps(m,p,-2), t) + &
!!! ueps(m,p,-2) * gr_sf(one,one,u(m,q,-1),t))), 0, "f_sgr + gr_sf = 0")
call expect (abs(u (m,q,1) * f_slgr (one, one, ueps(m,p,2), t) + &
ueps(m,p,2) * gr_slf(one,one,u(m,q,1),t))), 0, "f_slgr + gr_slf = 0")
call expect (abs(u (m,q,1) * f_srgr (one, one, ueps(m,p,2), t) + &
ueps(m,p,2) * gr_srf(one,one,u(m,q,1),t))), 0, "f_srgr + gr_srf = 0")
call expect (abs(u (m,q,1) * f_slrgr (one, two, one, ueps(m,p,2), t) + &
ueps(m,p,2) * gr_slrf(one,two,one,u(m,q,1),t))), 0, "f_slrgr + gr_slrf = 0")
call expect (abs(u (m,q,1) * f_pgr (one, one, ueps(m,p,2), t) + &
ueps(m,p,2) * gr_pf(one,one,u(m,q,1),t))), 0, "f_pgr + gr_pf = 0")
call expect (abs(u (m,q,1) * f_vgr (one, vt, ueps(m,p,2), p+q) + &
ueps(m,p,2) * gr_vf(one,vt,u(m,q,1),p+q))), 0, "f_vgr + gr_vf = 0")
call expect (abs(u (m,q,1) * f_vlgr (one, two, vt, ueps(m,p,2), p+q) + &
ueps(m,p,2) * gr_vlrf(one,two,vt,u(m,q,1),p+q))), 0, "f_vlgr + gr_vlrf = 0")
!!! call expect (abs(u (m,q,-1) * f_vgr (one, vt, ueps(m,p,2), p+q) + &
!!! ueps(m,p,2) * gr_vf(one,vt,u(m,q,-1),p+q))), 0, "f_vgr + gr_vf = 0")
!!! call expect (abs(u (m,q,1) * f_vgr (one, vt, ueps(m,p,1), p+q) + &
!!! ueps(m,p,1) * gr_vf(one,vt,u(m,q,1),p+q))), 0, "f_vgr + gr_vf = 0")
!!! call expect (abs(u (m,q,-1) * f_vgr (one, vt, ueps(m,p,1), p+q) + &
!!! ueps(m,p,1) * gr_vf(one,vt,u(m,q,-1),p+q))), 0, "f_vgr + gr_vf = 0")
!!! call expect (abs(u (m,q,1) * f_vgr (one, vt, ueps(m,p,-1), p+q) + &
!!! ueps(m,p,-1) * gr_vf(one,vt,u(m,q,1),p+q))), 0, "f_vgr + gr_vf = 0")
!!! call expect (abs(u (m,q,-1) * f_vgr (one, vt, ueps(m,p,-1), p+q) + &
!!! ueps(m,p,-1) * gr_vf(one,vt,u(m,q,-1),p+q))), 0, "f_vgr + gr_vf = 0")
!!! call expect (abs(v (m,q,1) * f_vgr (one, vt, ueps(m,p,-2), p+q) + &
!!! ueps(m,p,-2) * gr_vf(one,vt,v(m,q,1),p+q))), 0, "f_vgr + gr_vf = 0")
!!! call expect (abs(u (m,q,-1) * f_vgr (one, vt, ueps(m,p,-2), p+q) + &
!!! ueps(m,p,-2) * gr_vf(one,vt,u(m,q,-1),p+q))), 0, "f_vgr + gr_vf = 0")
call expect (abs(s_grf (one, ueps(m,p,2), u(m,q,1),t) + &
s_fgr(one,u(m,q,1),ueps(m,p,2),t))), 0, "s_grf + s_fgr = 0")
call expect (abs(sl_grf (one, ueps(m,p,2), u(m,q,1),t) + &
sl_fgr(one,u(m,q,1),ueps(m,p,2),t))), 0, "sl_grf + sl_fgr = 0")
call expect (abs(sr_grf (one, ueps(m,p,2), u(m,q,1),t) + &
sr_fgr(one,u(m,q,1),ueps(m,p,2),t))), 0, "sr_grf + sr_fgr = 0")
call expect (abs(slr_grf (one, two, ueps(m,p,2), u(m,q,1),t) + &
slr_fgr(one,two,u(m,q,1),ueps(m,p,2),t))), 0, "slr_grf + slr_fgr = 0")
call expect (abs(p_grf (one, ueps(m,p,2), u(m,q,1),t) + &
p_fgr(one,u(m,q,1),ueps(m,p,2),t))), 0, "p_grf + p_fgr = 0")
call expect (abs(v_grf (one, ueps(m,p,2), u(m,q,1),t) + &
v_fgr(one,u(m,q,1),ueps(m,p,2),t))), 0, "v_grf + v_fgr = 0")
call expect (abs(vlr_grf (one, two, ueps(m,p,2), u(m,q,1),t) + &
vlr_fgr(one,two,u(m,q,1),ueps(m,p,2),t))), 0, "vlr_grf + vlr_fgr = 0")
call expect (abs(u(m,p,1) * f_potgr (one,one,testv) - testv * gr_potf &
(one,one,u (m,p,1))), 0, "f_potgr - gr_potf = 0")
call expect (abs (pot_fgr (one,u(m,p,1),testv) - pot_grf(one, &
testv,u(m,p,1))), 0, "pot_fgr - pot_grf = 0")
call expect (abs(u(m,p,1) * f_s2gr (one,one,one,testv) - testv * gr_s2f &

```

```

    (one,one,one,u (m,p,1))), 0, "f_s2gr - gr_s2f = 0")
call expect (abs (s2_fgr (one,u(m,p,1),one,testv) - s2_grf(one, &
    testv,one,u(m,p,1))), 0, "s2_fgr - s2_grf = 0")
call expect (abs(u (m,q,1) * f_svgr (one, one, vt, ueps(m,p,2)) + &
    ueps(m,p,2) * gr_svf(one,one,vt,u(m,q,1))), 0, "f_svgr + gr_svf = 0")
call expect (abs(u (m,q,1) * f_slvgr (one, one, vt, ueps(m,p,2)) + &
    ueps(m,p,2) * gr_slvf(one,one,vt,u(m,q,1))), 0, "f_slvgr + gr_slvf = 0")
call expect (abs(u (m,q,1) * f_srvgr (one, one, vt, ueps(m,p,2)) + &
    ueps(m,p,2) * gr_srvf(one,one,vt,u(m,q,1))), 0, "f_srvgr + gr_srvf = 0")
call expect (abs(u (m,q,1) * f_slrvgr (one, two, one, vt, ueps(m,p,2)) + &
    ueps(m,p,2) * gr_slrvf(one,two,one,vt,u(m,q,1))), 0, "f_slrvgr + gr_slrvf = 0")
call expect (abs (sv1_fgr (one,u(m,p,1),vt,ueps(m,q,2)) + sv1_grf(one, &
    ueps(m,q,2),vt,u(m,p,1))), 0, "sv1_fgr + sv1_grf = 0")
call expect (abs (sv2_fgr (one,u(m,p,1),one,ueps(m,q,2)) + sv2_grf(one, &
    ueps(m,q,2),one,u(m,p,1))), 0, "sv2_fgr + sv2_grf = 0")
call expect (abs (slv1_fgr (one,u(m,p,1),vt,ueps(m,q,2)) + slv1_grf(one, &
    ueps(m,q,2),vt,u(m,p,1))), 0, "slv1_fgr + slv1_grf = 0")
call expect (abs (srv2_fgr (one,u(m,p,1),one,ueps(m,q,2)) + srv2_grf(one, &
    ueps(m,q,2),one,u(m,p,1))), 0, "srv2_fgr + srv2_grf = 0")
call expect (abs (slrv1_fgr (one,two,u(m,p,1),vt,ueps(m,q,2)) + slrv1_grf(one,two, &
    ueps(m,q,2),vt,u(m,p,1))), 0, "slrv1_fgr + slrv1_grf = 0")
call expect (abs (slrv2_fgr (one,two,u(m,p,1),one,ueps(m,q,2)) + slrv2_grf(one, &
    two,ueps(m,q,2),one,u(m,p,1))), 0, "slrv2_fgr + slrv2_grf = 0")
call expect (abs(u (m,q,1) * f_pvgr (one, one, vt, ueps(m,p,2)) + &
    ueps(m,p,2) * gr_pvf(one,one,vt,u(m,q,1))), 0, "f_pvgr + gr_pvf = 0")
call expect (abs (pv1_fgr (one,u(m,p,1),vt,ueps(m,q,2)) + pv1_grf(one, &
    ueps(m,q,2),vt,u(m,p,1))), 0, "pv1_fgr + pv1_grf = 0")
call expect (abs (pv2_fgr (one,u(m,p,1),one,ueps(m,q,2)) + pv2_grf(one, &
    ueps(m,q,2),one,u(m,p,1))), 0, "pv2_fgr + pv2_grf = 0")
call expect (abs(u (m,q,1) * f_v2gr (one, vt, vz, ueps(m,p,2)) + &
    ueps(m,p,2) * gr_v2f(one,vt,vz,u(m,q,1))), 0, "f_v2gr + gr_v2f = 0")
call expect (abs(u (m,q,1) * f_v2lrgr (one, two, vt, vz, ueps(m,p,2)) + &
    ueps(m,p,2) * gr_v2lrf(one,two,vt,vz,u(m,q,1))), 0, "f_v2lrgr + gr_v2lrf = 0")
call expect (abs (v2_fgr (one,u(m,p,1),vt,ueps(m,q,2)) + v2_grf(one, &
    ueps(m,q,2),vt,u(m,p,1))), 0, "v2_fgr + v2_grf = 0")
call expect (abs (v2lr_fgr (one,two,u(m,p,1),vt,ueps(m,q,2)) + v2lr_grf(one, two, &
    ueps(m,q,2),vt,u(m,p,1))), 0, "v2lr_fgr + v2lr_grf = 0")

<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")
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)")
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)")
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)")
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)")
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)")
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)")

```

```

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)")
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)")
print *, "Irreducibility:"
call expect (abs(f_potgr (one, one, (cmplx (p*p - m**2, m*w, &
    kind=default) * pr_grav(p,m,w,testv)))), 0, "g.pr.test")
call expect (abs(f_potgr (one, one, (cmplx (p*p - m**2, m*w, &
    kind=default) * pr_grav(p,m,w,ueps(m,p,2)))))), 0, &
    "g.pr.ueps ( 2)")
call expect (abs(f_potgr (one, one, (cmplx (p*p - m**2, m*w, &
    kind=default) * pr_grav(p,m,w,ueps(m,p,1)))))), 0, &
    "g.pr.ueps ( 1)")
call expect (abs(f_potgr (one, one, (cmplx (p*p - m**2, m*w, &
    kind=default) * pr_grav(p,m,w,ueps(m,p,-1)))))), 0, &
    "g.pr.ueps (-1)")
call expect (abs(f_potgr (one, one, (cmplx (p*p - m**2, m*w, &
    kind=default) * pr_grav(p,m,w,ueps(m,p,-2)))))), 0, &
    "g.pr.ueps (-2)")
call expect (abs(f_potgr (one, one, (cmplx (p*p - m**2, m*w, &
    kind=default) * pr_grav(p,m,w,veps(m,p,2)))))), 0, &
    "g.pr.veps ( 2)")
call expect (abs(f_potgr (one, one, (cmplx (p*p - m**2, m*w, &
    kind=default) * pr_grav(p,m,w,veps(m,p,1)))))), 0, &
    "g.pr.veps ( 1)")
call expect (abs(f_potgr (one, one, (cmplx (p*p - m**2, m*w, &
    kind=default) * pr_grav(p,m,w,veps(m,p,-1)))))), 0, &
    "g.pr.veps (-1)")
call expect (abs(f_potgr (one, one, (cmplx (p*p - m**2, m*w, &
    kind=default) * pr_grav(p,m,w,veps(m,p,-2)))))), 0, &
    "g.pr.veps (-2)")

<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>

```

T.20 O'Mega Virtual Machine

```

(omegavm95.f90)≡
  <Copleft>
  module omevavm95
    use kinds
    use omega95
    ! use omega95_bispinors
    implicit none
    private
    <OVM Procedure Declarations>
    <OVM Data Declarations>
    <OVM Instructions>
  contains
    <OVM Procedure Implementations>
  end module omevavm95

```

T.20.1 Memory Layout

On one hand, we need a memory pool for all the intermediate results

```

<OVM Data Declarations>≡
  type, public :: ovm
  private
    complex(kind=default) :: amp
    type(momentum), dimension(:), pointer :: p
    complex(kind=default), dimension(:), pointer :: phi
    type(spinor), dimension(:), pointer :: psi
    type(conjspinor), dimension(:), pointer :: psibar
    ! type(bispinor), dimension(:), pointer :: chi
    type(vector), dimension(:), pointer :: v
  end type ovm

<OVM Procedure Declarations>≡
  public :: alloc

<OVM Procedure Implementations>≡
  subroutine alloc (vm, momenta, scalars, spinors, conjspinors, vectors)
    type(ovm), intent(inout) :: vm
    integer, intent(in) :: momenta, scalars, spinors, conjspinors, vectors
    allocate (vm%p(momenta))
    allocate (vm%phi(scalars))
    allocate (vm%psi(spinors))
    allocate (vm%psibar(conjspinors))
    allocate (vm%v(vectors))
  end subroutine alloc

```

and on the other hand, we need to access coupling parameters that define the environment

```

<OVM Data Declarations>+≡
  type, public :: ovm_env
  private
    real(kind=default), dimension(:), pointer :: gr
    real(kind=default), dimension(:, :), pointer :: gr2
    complex(kind=default), dimension(:), pointer :: gc
    complex(kind=default), dimension(:, :), pointer :: gc2

```

```
end type ovm_env
```

NB: during, execution, the type of the coupling constant is implicit in the instruction.



How to load coupling constants? Is brute force linear lookup good enough?

T.20.2 Instruction Set

```
<OVM Data Declarations>+≡
integer, parameter, private :: MAX_RHS = 3
type, public :: instr
  private
    integer :: code, sign, coupl, lhs
    integer, dimension(MAX_RHS) :: rhs
  end type instr

<OVM Procedure Declarations>+≡
public :: eval

<OVM Procedure Implementations>+≡
pure subroutine eval (vm, amp, env, amplitude, p, s)
  type(ovm), intent(inout) :: vm
  complex(kind=default), intent(out) :: amp
  type(ovm_env), intent(in) :: env
  type(instr), dimension(:), intent(in) :: amplitude
  real(kind=default), dimension(0:,:), intent(in) :: p
  integer, dimension(:), intent(in) :: s
  integer :: code, sign, coupl, lhs
  integer, dimension(MAX_RHS) :: rhs
  integer :: i, pc
  vm%p(1) = - p(:,1)
  vm%p(2) = - p(:,2)
  do i = 3, size (p, dim = 2)
    vm%p(i) = p(:,i)
  end do
  do pc = 1, size (amplitude)
    code = amplitude(pc)%code
    sign = amplitude(pc)%sign
    coupl = amplitude(pc)%coupl
    lhs = amplitude(pc)%lhs
    rhs = amplitude(pc)%rhs
    select case (code)
      <cases of code>
    end select
  end do
  amp = vm%amp
end subroutine eval
```

Loading External states

```
<OVM Instructions>≡
integer, public, parameter :: OVM_LOAD_SCALAR = 1
integer, public, parameter :: OVM_LOAD_U = 2
integer, public, parameter :: OVM_LOAD_UBAR = 3
```

```

integer, public, parameter :: OVM_LOAD_V = 4
integer, public, parameter :: OVM_LOAD_VBAR = 5
integer, public, parameter :: OVM_LOAD_VECTOR = 6

<cases of code>≡
case (OVM_LOAD_SCALAR)
  vm%phi(lhs) = 1
case (OVM_LOAD_U)
  if (lhs <= 2) then
    vm%psi(lhs) = u (env%gr(coupl), - vm%p(rhs(1)), s(rhs(2)))
  else
    vm%psi(lhs) = u (env%gr(coupl), vm%p(rhs(1)), s(rhs(2)))
  end if
case (OVM_LOAD_UBAR)
  if (lhs <= 2) then
    vm%psibar(lhs) = ubar (env%gr(coupl), - vm%p(rhs(1)), s(rhs(2)))
  else
    vm%psibar(lhs) = ubar (env%gr(coupl), vm%p(rhs(1)), s(rhs(2)))
  end if
case (OVM_LOAD_V)
  if (lhs <= 2) then
    vm%psi(lhs) = v (env%gr(coupl), - vm%p(rhs(1)), s(rhs(2)))
  else
    vm%psi(lhs) = v (env%gr(coupl), vm%p(rhs(1)), s(rhs(2)))
  end if
case (OVM_LOAD_VBAR)
  if (lhs <= 2) then
    vm%psibar(lhs) = vbar (env%gr(coupl), - vm%p(rhs(1)), s(rhs(2)))
  else
    vm%psibar(lhs) = vbar (env%gr(coupl), vm%p(rhs(1)), s(rhs(2)))
  end if
case (OVM_LOAD_VECTOR)
  if (lhs <= 2) then
    vm%v(lhs) = eps (env%gr(coupl), - vm%p(rhs(1)), s(rhs(2)))
  else
    vm%v(lhs) = eps (env%gr(coupl), vm%p(rhs(1)), s(rhs(2)))
  end if

<OVM Instructions>+≡
integer, public, parameter :: OVM_ADD_MOMENTA = 10

<cases of code>+≡
case (OVM_ADD_MOMENTA)
  vm%p(lhs) = vm%p(rhs(1)) + vm%p(rhs(2))

<OVM Instructions>+≡
integer, public, parameter :: OVM_PROPAGATE_SCALAR = 11
integer, public, parameter :: OVM_PROPAGATE_SPINOR = 12

<cases of code>+≡
case (OVM_PROPAGATE_SCALAR)
  vm%phi(lhs) = pr_phi (vm%p(lhs), env%gr(rhs(1)), env%gr(rhs(2)), vm%phi(lhs))
case (OVM_PROPAGATE_SPINOR)
  vm%psi(lhs) = pr_psi (vm%p(lhs), env%gr(rhs(1)), env%gr(rhs(2)), vm%psi(lhs))

<OVM Instructions>+≡
integer, public, parameter :: OVM_FUSE_VECTOR_PSIBAR_PSI = 21

```

```

integer, public, parameter :: OVM_FUSE_PSI_VECTOR_PSI = 22
integer, public, parameter :: OVM_FUSE_PSIBAR_PSIBAR_VECTOR = 23

<cases of code>+≡
  case (OVM_FUSE_VECTOR_PSIBAR_PSI)
    vm%v(lhs) = &
      v_ff (sign*env%gc(coupl), vm%psibar(rhs(1)), vm%psi(rhs(2)))
  case (OVM_FUSE_PSI_VECTOR_PSI)
    vm%psi(lhs) = &
      f_vf (sign*env%gc(coupl), vm%v(rhs(1)), vm%psi(rhs(2)))
  case (OVM_FUSE_PSIBAR_PSIBAR_VECTOR)
    vm%psibar(lhs) = &
      f_fw (sign*env%gc(coupl), vm%psibar(rhs(1)), vm%v(rhs(2)))

<Copyleft>≡
! $Id: omegalib.nw 1655 2010-02-02 15:50:32Z ohl $
!
! Copyright (C) 1999-2009 by
!   Wolfgang Kilian <kilian@hep.physik.uni-siegen.de>
!   Thorsten Ohl <ohl@physik.uni-wuerzburg.de>
!   Juergen Reuter <juergen.reuter@physik.uni-freiburg.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.
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

```

—U—

INDEX

This index has been generated automatically and might not be 100%ly accurate. In particular, hyperlinks have been observed to by of by one page.

*, **29, 30, 21**, used: 362, 363,
364, 330, 330, 331, 332,
336, 336, 346, 30, 31, 32,
33, 33, 34, 34, 34, 35, 49,
56, 62, 171, 137, 138, 105,
113, 117, 122, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??

+, **29, 30, 21**, used: 307, 307,
299, ??, 363, 365, 330,
330, 331, 333, 333, 334,
30, 31, 31, 32, 32, 32, 33,
34, 34, 35, 35, 49, 57, 57,
60, 62, 62, 64, 264, 171,
174, 178, 181, 107, 120,
122, 122, 123, 128, 272,
273, 275, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
190, 193, 196, 219, ??, ??

-, **29, 30, 21**, used: 307, 303,
??, ??, ??, 287, 287, 370,
371, 363, 330, 331, 337,
338, 350, 23, 30, 31, 31,
32, 32, 33, 33, 33, 35, 36,
56, 57, 58, 60, 61, 62, 62,
63, 64, 65, 65, 174, 175,
175, 176, 181, ??, 154,
163, 116, 123, 272, 273,
273, ??, ??, ??, ??, ??, ??,
187, 190, 215, 230, 69

/, **29, 30, 21**, used: 362, 363,
330, 330, 331, 331, 23, 26,
26, 28, 30, 33, 34, 73

<, **29, 30, 21**, used: 307, 308,
308, 308, 309, 309, 309,
310, 312, 313, 313, 313,
314, 314, 315, 315, 298,
301, 301, ??, 363, 330,
330, 331, 332, 332, 334,
334, 336, 350, 350, 26, 30,
31, 32, 33, 33, 35, 47, 55,
56, 57, 57, 58, 60, 174,
175, 69, 134, 136, 155,
156, 159, 164, 165, 105,
125, ??, ??, 194, 194, 202,
207

<=, **29, 30, 21**, used: 298, 298,
298, 299, 330, 331, 332,
337, 338, 347, 348, 30, 30,
48, 49, 55, 56, 61, 63, 257,
181, 73, 123, 272, ??, ??,
??, ??, ??, ??, ??, ??,
190, 193, 193, 213

<>, **29, 30, 21**, used: 312, 299,
300, ??, ??, ??, 320, 323,
371, 372, 363, 365, 333,
333, 334, 347, 348, 350,
12, 13, 15, 17, 30, 44, 56,
62, 63, 64, 358, ??, ??,
263, 265, 174, 174, 175,
138, 139, 139, 139, 140,
140, 142, 142, 142, 142,
143, 143, 143, 144, 144,
144, 145, 145, 145, 146,
146, 146, 147, 147, 147,
148, 149, 149, 150, 150,
156, 165, 112, 113, 128,
129

=, **29, 30, 21**, used: 308, 308,
308, 309, 309, 309, 310,
310, 311, 312, 313, 313,
313, 314, 314, 314, 315,
315, 299, 300, 300, 300,
301, 301, ??, 296, 318,
319, 322, 362, 363, 364,
365, 365, 330, 331, 331,
332, 334, 334, 335, 336,
343, 344, 344, 345, 346,
349, 350, 12, 14, 30, 31,
32, 32, 33, 33, 34, 34, 56,
56, 56, 57, 57, 57, 57, 58,
58, 58, 58, 58, 58, 59, 59,
59, 59, 59, 61, 62, 62, 62,
62, 63, 63, 63, 63, 64, 64,
64, 64, 65, 258, 356, 357,
??, ??, ??, 171, 171, 171,
171, 171, 175, 175, 176,
176, 180, 181, 75, 76, 76,
141, 152, 156, 162, 165,
105, 108, 108, 125, ??, ??,
??, ??, ??, 215, 235, 236

>, **29, 30, 21**, used: 307, 307,
299, 299, 303, ??, ??, ??,
291, 292, 371, 330, 330,
332, 332, 333, 334, 337,
338, 338, 18, 23, 30, 33,
55, 56, 56, 59, 60, 62, 64,
65, 356, 174, 176, 134,

136, 138, 139, 139, 139, 139,
140, 140, 142, 142, 142,
142, 143, 143, 143, 144,
144, 144, 145, 145, 145,
146, 146, 146, 147, 147,
147, 148, 149, 149, 150,
150, 154, 155, 156, 156,
163, 164, 165, 273, 273,
273, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, 215,
226, ??, ??, ??, ??, ??, ??,
??, ??, ??

\geq , **29, 30, 21**, used: 307, 307,
310, 311, ??, ??, 372, 330,
338, 30, 31, 57, 58, 62, 63,
356, 357, 174, 175, 175,
181, 115, 123, 272, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, 190, ??, ??, ??, ??

?cmp (label), **299, 297**, used:
?f (label), **30, 33, 22**, used:
?stride (label), **299, 297**, used:
A, ??, ??, ??, used: ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??

A (module), **289**, used: 289, 289,
289, 289

abs, **54, 56, 62, 355, 356, 52**,
354, used: 362, 261, 357,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, 282, 282, ??,
??, ??, ??, ??, ??

actions (field), **289**, used: 289,
289, 289, ??, ??

AD, ??, used:
add, **306, 310, 313, 318, 318**,
321, 321, 362, 363, 363,
364, 366, 367, 32, 45,
46, 50, 54, 57, 62, 355,
356, 171, 305, 316, 317,
359, 360, 361, 52, 354,
170, used: 312, 303, 318,
319, 321, 322, 364, 366,
366, 366, 366, 366, 367,
367, 367, 367, 367, 289,
349, 349, 31, 32, 32, 33,
46, 46, 46, 47, 47, 47, 47,
50, ??, ??, ??, ??, ??,
263, 264, 264, 264, 266,
171, ??, 181, 73, 109, 111,
113, 113, 116, 120, 121,
123, 123, 124, 199, 213,
213, 214, 217, 173

add', **56**, used: 56, 57
add1, **171**, used: 171, 171, 171
addn, **171**, used: 171, 171
add_degree, **31**, used: 31
add_derivatives, **366**, used: 366,
366
add_flavors, **264**, used: 264
add_fusion2, ??, used: ??
add_fusion3, ??, used: ??
add_fusionnn, ??, used:
add_node, **42, 47, 39**, used: 47,
48, 116
add_non_empty, **319, 322**, used:
319, 322
add_offspring, **42, 47, 115, 39**,
used: 115, 115
add_offspring_unsafe, **42, 47**,
39, used: 47, 48
add_permute3, ??, used: ??
add_permute4, ??, used: ??
add_tag, **191**, used: 191, 212
add_to_list, **124**, used: 124
add_tree, **263**, used: 263
add_unique, **349**, used: 349
add_vertex3, ??, used: ??
add_vertex4, ??, used: ??
add_vertices, **109**, used: 109
AdjSUN, **80, 77**, used: 80, 180,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
279, ??, ??

- atom*, **363, 363, 364, 364, 366, 367, 360, 360, 361**,
 used: **366**
Atom, ??, used: ??, ??, ??, ??,
 ??, ??, ??
atoms, **363, 363, 364, 365, 367, 368, 360, 361, 361**,
 used: **367, 368**
Atpsi, ??, ??, ??, used:
attach_to_sorted, **333**, used:
333
attach_to_fst_unsorted, **332**,
 used: **333, 333, 333**
AU, ??, used:
author, **287, 286**, used: **288**
author (field), **287, 286**, used:
288, 337, 10, 22, 42, 54,
 ??, **262, 174, 103, ??, ??,**
 ??, ??, ??, **244, 184, 278,**
 ??, ??
Aux_ConjSpinor, ??, used:
Aux_DScalar_DScalar, ??, used:
137, ??, ??
Aux_Gauge_Gauge, ??, used:
137, ??, ??
Aux_Majorana, ??, used:
Aux_Scalar, ??, used: ??
Aux_Scalar_Scalar, ??, used:
137, ??, ??
Aux_Scalar_Vector, ??, used:
137
Aux_Spinor, ??, used:
Aux_Tensor_1, ??, used: ??, ??
Aux_Vector, ??, used:
Aux_Vector_DScalar, ??, used:
137, ??
Aux_Vector_Vector, ??, used:
137, ??, ??, ??
A_0, ??, used:
A_P, ??, used:
B, **279**, used: **279, 279, 280, 280**
B1, ??, used: ??, ??, ??
B2, ??, used: ??, ??, ??
bag (type), **273**, used: **273**
bal, **307**, used: **307, 308, 309,**
309, 310
Bbar, **279**, used: **279, 279, 280,**
280
BBB, ??, used: **137, ??, ??, ??,**
 ??, ??, ??, ??, ??
bcdi_of_flavor, ??, used: ??, ??
Be, ??, used:
begin_step, **293, 291**, used: **130**
Binary (module), **11, 23, 124, 9,**
21, 101, used: **16, 24, 25,**
36, 36, 124, 126, 278, 21,
22
Binary (sig), **11, 9**, used: **9**
Binary_Majorana (module), **126,**
101, used:
binomial, **330, 327**, used:
binomial', **330**, used: **330, 330**
bits, **61**, used: **62, 62, 62, 62, 62,**
63, 63, 63, 64, 64
Bits (module), **60, 54**, used: **65,**
66, 66, 54
bits0, **61**, used: **61**
BitsW (module), **65, 54**, used:
66, 66
boson (type), ??, ??, used: ??,
185, ??
Boson, **105, 125, ??**, used: **105,**
105, 125, 125, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??
boson2 (type), ??, used: **185, ??**
Bound (sig), **19, 9**, used: **26, 35,**
126, 127, 9, 21, 22, 101,
101
bounded_power, **18**, used: **18**
bounded_power_fold, **18**, used:
18, 18
bra, **103, 117, 99**, used: **214**
braket (type), **103, 117, 99**,
 used: **103, 117, 99, 100**
brakets, **103, 119, 100**, used:
119, 215
brakets (field), **117**, used: **119,**
120, 122, 122, 122, 123,
124, 124
Branch, **257**, used: **257, 257, 257,**
258
Broken_Gauge (sig), ??, used:
broken_is_unit, **366**, used:
BRS, ??, used: **176**
brs_chi_incoming, **185, 186,**
229, used: **215**
brs_chi_outgoing, **185, 186, 229,**
 used: **215**

- brs_conjspinors* (field), **191**,
used: **192, 192, 198**
- brs_massive_vectors* (field), **191**,
used: **192, 192, 198**
- brs_psibar_incoming*, **185, 186**,
229, used: **215**
- brs_psibar_outgoing*, **185, 186**,
229, used: **215**
- brs_psi_incoming*, **185, 186**,
229, used: **215**
- brs_psi_outgoing*, **185, 186, 229**,
used: **215**
- brs_realspinors* (field), **191**, used:
192, 192, 198
- brs_scalars* (field), **191**, used:
192, 192, 198
- brs_spinors* (field), **191**, used:
192, 192, 198
- brs_vectors* (field), **191**, used:
192, 192, 198
- brs_vectorspinors* (field), **191**,
used: **192, 192**
- BSM_anom* (module), ??, ??,
used: ??
- BSM_bsm* (module), ??, ??,
used: ??, ??, ??, ??, ??, ??,
??, ??
- BSM_flags* (sig), ??, ??, used:
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??
- BSM_ungauged* (module), ??,
??, used: ??
- build_forest*, **49**, used: **49, 49**
- c* (type), ??, ??, ??, ??, **177**,
181, 137, 161, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, **280**, ??, ??,
??, ??, ??, used: ??, ??,
??, ??, ??, ??, ??, ??
- C*, **279**, used: **279, 279, 280, 280**
- C* (module), **363, 364, 365, 367**,
358, 133, 159, 119, 128,
272, 242, 360, 361, 184,
used: **363, 364, 358, 358**,
358, 119, 120, 121, 131,
275, 278, 360, 361, 361,
361, 361
- C1*, ??, ??, ??, used: ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
- ??, ??, ??, ??, ??, ??, ??,
- C1c*, ??, ??, ??, used: ??, ??, ??,
??, ??, ??, ??, ??, ??, ??
- C2*, ??, ??, ??, used: ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??
- C2c*, ??, ??, ??, used: ??, ??, ??,
??, ??, ??, ??, ??, ??, ??
- C3*, ??, used: ??, ??, ??
- C3c*, ??, used: ??, ??, ??
- C4*, ??, used: ??, ??, ??
- C4c*, ??, used: ??, ??, ??
- Cache* (module), **294, 294**, used:
109
- Cache_Default*, **275**, used: **275**
- Cache_Ignore*, **107**, used: **107**
- Cache_Initialize*, **275**, used: **275**
- cache_mode* (type), **107, 275**,
used:
- cache_option*, **107, 275**, used:
107, 110, 275
- Cache_Overwrite*, **107**, used: **107**
- cache_prefix*, ??, ??, used: **110**
- cache_suffix*, ??, ??, used: **294**
- Cache_Use*, **107**, used: **107**
- call_function*, **267**, used: **267**
- call_subroutine*, **267**, used: **267**
- canonicalize*, **306, 312, 315**,
305, used:
- cascade* (camlyacc non-terminal),
70, used: **70**
- Cascade* (module), **72, 71**, used:
104, 119, 128, 272, 101
- cascades* (camlyacc non-terminal),
70, used: **70, 70**
- Cascade_lexer* (module), ??, ??,
69, used: **73**
- Cascade_parser* (module), ??, ??,
70, used: ??, **73**, ??, **69**
- Cascade_syntax* (module), **67**,
67, used: ??, ??, **72**, ??,
70, 70
- Cbar*, **279**, used: **279, 279, 280**,
280
- CF* (module), **272, 189**, used:
275, 189, 218, 219, 219,
219, 220, 220, 220, 220,
225, 226

- CFlow* (module), **219**, used: **219**,
219, 220, 221
- cflow_to_string*, **219**, used: **225**
- CF_aux*, **134**, **159**, used: **135**,
135, 136, 139, 140, 140,
140, 141, 141, 143, 144,
144, 144, 145, 145, 145,
146, 146, 146, 147, 147,
147, 148, 149, 149, 150,
150, 151, 151, 151, 151,
151, 151, 154, 156, 160,
161, 163, 165
- CF_in*, **134**, **159**, used: **135**, **135**,
136, 138, 139, 139, 139,
140, 140, 142, 142, 142,
142, 143, 143, 143, 144,
144, 144, 145, 145, 145,
146, 146, 146, 147, 147,
147, 148, 149, 149, 150,
150, 154, 156, 160, 161,
162, 163, 165
- CF_io*, **134**, **159**, used: **135**, **135**,
136, 139, 140, 140, 140,
141, 141, 141, 143, 144,
144, 144, 145, 145, 145,
146, 146, 146, 147, 147,
147, 148, 149, 149, 150,
150, 151, 151, 151, 151,
151, 151, 152, 152, 152,
152, 152, 154, 156, 160,
161, 163, 165
- CF_out*, **134**, **159**, used: **135**,
135, 136, 138, 139, 139,
139, 140, 140, 142, 142,
142, 142, 143, 143, 143,
144, 144, 144, 145, 145,
145, 146, 146, 146, 147,
147, 147, 148, 149, 149,
150, 150, 154, 156, 160,
161, 162, 163, 165
- channel*, **292**, **291**, used: **130**
- channel* (field), **291**, used: **292**,
292
- channel* (type), **291**, used: **291**
- Channel*, **291**, **127**, used: **292**,
128
- Chaotic* (module), **44**, **41**, used:
- char* (type), ??, ??, ??, used:
- char* (camllex regexp), **69**, used:
- charged_chargino_currents*, ??,
??, ??, used: ??, ??, ??
- charged_currents*, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, used: ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??
- charged_currents'*, ??, ??, used:
??, ??, ??
- charged_currents''*, ??, used: ??
- charged_currents_ckm*, ??, used:
??
- charged_currents_triv*, ??, used:
??
- charged_heavy_currents*, ??,
used: ??
- charged_quark_currents*, ??, ??,
??, used: ??, ??, ??
- charged_slepton_currents*, ??, ??,
??, used: ??, ??, ??
- charged_squark_currents*, ??, ??,
??, used: ??, ??, ??
- charged_squark_currents'*, ??,
??, ??, used: ??, ??, ??
- charged_up_currents*, ??, used:
??
- charged_Z*, ??, ??, ??, used: ??,
??, ??
- Chargino*, ??, ??, ??, used: ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??
- charlist*, ??, used: ??
- check*, **55**, used: **55**, **55**
- chg*, ??, ??, used: ??, ??, ??, ??
- Chi*, ??, used: ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??
- Chibar*, ??, used: ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??
- chiggs* (type), ??, used: ??
- CHiggs*, ??, used: ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,

- ??, ??, ??, ??, ??, ??, ??
children, **103, 109, 99**, used:
 199, 199, 213
children (type), **22, 24, 26, 28,**
 28, 35, 36, 42, 42, 43,
 44, 45, 46, 103, 106,
 106, 107, 108, 114, 20,
 38, 39, 41, 98, 102,
 used: 22, 42, 42, 43, 43,
 44, 45, 46, 106, 106, 107,
 108, 114, 20, 21, 22, 22,
 38, 38, 39, 39, 39, 40, 40,
 40, 41, 41, 42, 102, 103
children2, **199**, used: 202
children3, **199**, used: 207
chi_gauss, **185, 187, 229**, used:
chi_incoming, **185, 186, 229**,
 used: 215
chi_outgoing, **185, 186, 229**,
 used: 215
chi_projector, **185, 186, 229**,
 used: 211, 211
chi_propagator, **185, 186, 229**,
 used: 210
chi_type, **185, 186, 228**, used:
 198
choose, **306, 309, 313, 334, 305,**
 329, used: 11, 11, 18, 18,
 73, 136, 136, 136, ??, ??
choose', **334**, used: 334, 334
choose2, **11, ??, ??**, used: 12, 14,
 ??, ??
choose3, **11**, used: 14
choose_opt, **306, 309, 313, 305**,
 used:
CKM_12, ??, used:
CKM_13, ??, used:
CKM_23, ??, used:
ckm_present, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, used: ??,
 ??, ??, ??, ??, ??, ??, ??
classify, **300, 34, 297**, used: 331,
 333, 334, 18, 18, 34
classify_arrays, **193**, used: 193,
 193
classify_nodes, **124**, used: 124
classify_parameters, **193**, used:
 196
classify_singles, **193**, used: 193,
 193
classify_wfs, **192**, used: 198
classify_wfs', **192**, used: 192, 192
class_size, **32**, used: 32, 33, 33
class_size_3, **32**, used: 32
class_size_n, **33**, used: 32
clist, ??, ??, used: ??, ??
clone, **301, ??, 298**, used: 301,
 325, 18, ??, ??
clone3, ??, used: ??
clone4, ??, used: ??
clonen, ??, used: ??
cloop, ??, ??, used: ??, ??, ??,
 ??
cloop_kk, ??, used: ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??
cloop_kk2, ??, used: ??
close_channel, **292**, used: 293
CM (module), **157, 272**, used:
 157, 157, 158, 159, 272,
 272, 272, 272
cmdline, **290, 289**, used: 275
collect23, **28**, used: 28
collect3, **28**, used: 28
collect34, **28**, used: 28
collect4, **28**, used: 28
collect_derivatives, **366**, used:
 366
collect_vertices, **109**, used: 109
COLON, ??, ??, used: ??, 69
COLON (camlyacc token), **70**,
 used: 70, 71
color, ??, **177, 134, 157, 160,**
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, **279**,
 ??, ??, ??, used: 177,
 134, 136, 138, 141, 153,
 154, 155, 156, 157, 160,
 162, 162, 162, 163, 164,
 165, 128, ??, ??
color (label), ??, ??, used: 181
color (type), **80, 81, 81, 77**,
 used: 80, 81, 82, 77
Color (module), **80, 77**, used:
 178, 178, 178, 179, 180,
 133, 157, 159, 166, 127,
 128, 128, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,

- ??, ??, ??, ??, ??, ??,
219, 279, ??, ??, 102, 102,
??, ??, ??, ??
- Colored* (module), 127, 102,
used: 272, 184, 189, ??
- Colored* (sig), 127, 101, used:
127, 102
- Colored_Maker* (sig), 127, 102,
used: 102
- colored_vertex*, 134, 159, used:
138, 139, 139, 139, 140,
140, 141, 141, 141, 142,
142, 142, 142, 143, 143,
143, 144, 144, 144, 145,
145, 145, 146, 146, 146,
147, 147, 147, 148, 149,
149, 150, 150, 153, 153,
153
- Colorize* (module), 133, 133,
used: 272
- Colorized* (sig), ??, used: 127,
127, 184, 189, 133, 133,
102, ??
- Colorized_Gauge* (sig), ??, used:
133
- colorize_crossed_amplitude*, 156,
166, used: 157, 166
- colorize_crossed_amplitude1*,
156, 165, used: 156, 156,
165, 166
- colorize_flavor*, 136, used: 137,
137
- colorize_fusion2*, 162, used: 162
- colorize_fusion3*, 162, used: 163
- colorize_fusionn*, 162, used: 163
- colorize_propagator*, 134, 160,
used: 135, 160
- colorize_vertex3*, 138, used: 153
- colorize_vertex4*, 141, used: 153
- colorize_vertexn*, 153, used: 153
- colors*, 128, used: 130
- colors* (type), 128, used:
- color_current*, 280, used: 280
- color_currents*, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, used:
??, ??, ??, ??, ??, ??, ??,
??, ??
- color_flows*, 136, 127, 128, 102,
used: 136, 136, 136, 136,
138, 139, 139, 139, 140,
140, 142, 142, 142, 142,
143, 143, 143, 144, 144,
144, 145, 145, 145, 146,
146, 146, 147, 147, 147,
148, 149, 149, 150, 150,
225, 226
- color_flows* (field), 128, used:
128
- color_flow_ambiguous*, 134, 159,
used: 143
- color_flow_of_string*, 134, 159,
used: 154, 163
- color_flow_pairs*, 136, used: 136,
139, 140, 140, 140, 141,
141, 143, 144, 144, 144,
145, 145, 145, 146, 146,
146, 147, 147, 147, 148,
149, 149, 150, 150, 151,
151, 151, 151, 151, 151
- color_flow_quadruples*, 136,
used: 152
- color_flow_triples*, 136, used:
141, 147, 148, 149, 149,
150, 150, 152, 152, 152,
152
- color_to_string*, 219, used: 219
- colsymm*, ??, used:
- columns*, 175, used: 176
- column_tabs*, 175, used: 176
- col_currents*, ??, ??, ??, used:
??, ??, ??
- col_lqino_currents*, ??, used: ??
- col_lq_currents*, ??, used: ??
- col_sfermion_currents*, ??, ??,
??, used: ??, ??, ??
- Combinatorics* (module), 329,
327, used: 11, 11, 18, 18,
26, 28, 30, 35, 36, 73, 136,
136, 136, 136, 136, 156,
165, 105, 121, 126, ??, ??
- combine_decays*, 261, used: 261,
261
- commute_proj*, 232, used: 233,
235, 236
- compare*, 306, 312, 314, 299,
318, 320, 289, 10, 12,
13, 15, 17, 29, 30, 31,
42, 42, 43, 44, 44, 44,
45, 54, 56, 62, ??, ??,
??, ??, 263, 264, 266,

compare', **312**, used: **312**, **312**
Comphep (module), **174**, **174**,
 used:
Comphep_lexer (module), **??**, **??**,
172, used: **179**
Comphep_parser (module), **??**,
??, **173**, used: **??**, **179**,
??, **172**
Comphep_syntax (module), **170**,
170, used: **??**, **??**, **178**,
178, **181**, **??**, **173**, **173**
complete_fusion_tower, **116**,
 used: **121**
Complex, **358**, **??**, used: **??**, **??**
Complex (module), **355**, **354**,
 used:
Complex_Array, **??**, used:
complex_arrays (field), **193**,
 used: **193**, **193**, **196**
complex_singles (field), **193**,
 used: **193**, **193**, **196**
compose, **306**, **310**, **315**, **305**,
 used: **310**, **315**
compress, **303**, **302**, used: **303**
compress2, **303**, **302**, used:
compressed (type), **302**, **302**,
 used: **302**, **302**

- ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, **279**, ??, ??,
 ??, used: **75, 76, 76, 157**
- conj_char*, ??, ??, ??, used: ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??
- conj_chiggs*, ??, used: ??
conj_iso, ??, used: ??, ??, ??
conj_symbol, ??, ??, ??, used:
 ??, ??, ??
- cons*, **306, 309, 312**, ??, **342**,
305, ??, **340**, used: **309**,
310, 312, 48, 49
- Const*, ??, used: **181**, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??
- constant* (type), ??, **177, 134**,
157, 160, 103, 108, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, **279**, ??,
 ??, **98**, ??, used: **177**,
181, 134, 137, 157, 160,
161, 103, 104, 106, 108,
108, 109, 109, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, **280**, ??,
 ??, ??, **98, 101, 103**, ??,
 ??, ??, ??, ??
- Constant*, ??, used: **181**, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??
- constant_symbol*, ??, **177, 155**,
157, 164, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, **282**, ??, ??, ??, used:
177, 155, 157, 164, ??, ??,
194, 194, 194, 194, 195,
195, 195, 195, 202, 207
- constant_symbol* (label), ??, ??,
 used: **181**
- constraints*, **103, 119, 127, 128**,
100, 102, used: **225**
- constraints* (field), **117, 128**,
 used: **119, 120, 121, 128**,
131
- continuing*, **190**, used: **190, 190**
- contract4* (type), ??, used: ??, ??
- copy_matrix*, **369, 369**, used:
372, 373, 373
- Cos*, ??, ??, used:
- Cos2al*, ??, used:
- Cos2am2b*, ??, used:
- Cos2be*, ??, used:
- Cos4be*, ??, used:
- Cosamb*, ??, used:
- Cosapb*, ??, used:
- Cospsi*, ??, ??, ??, used:
- Costhw*, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, used:
 ??, ??, ??, ??
- Cot*, ??, used:
- Count* (module), **30, 22**, used:
- Count* (sig), **30, 21**, used: **22**
- count_color_strings*, **155, 164**,
 used: **156, 165**
- count_diagrams*, **103, 122, 100**,
 used: **275**
- count_flavors*, **178**, used: **181**
- count_fusions*, **103, 122, 100**,
 used: **275**
- count_maps*, **263**, used: **263**
- count_non_zero*, **65**, used: **65**
- count_non_zero'*, **65**, used: **65**,
65
- count_processes*, **128**, used: **130**
- count_propagators*, **103, 122**,
100, used: **275**
- count_trees*, **42, 49, 40**, used:
122
- coupling*, **103, 109, 98**, used: **202**
- coupling* (field), **108**, used: **113**,
117
- coupling* (type), **103, 106, 106**,
107, 108, 100, 102,
 used: **103, 106, 108, 108**,
114, 114, 100, 102
- Coupling* (module), ??, used: ??,
 ??, ??, **176, 178, 178, 178**,
178, 180, 181, 181, 133,
159, 103, 107, 108, 109,
109, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,

`??, ??, ??, ??, ??, ??, ??, ??,`
`??, ??, ??, ??, ??, ??, ??, ??,`
`??, ??, ??, ??, 185, 186,`
`189, 228, 279, ??, ??, ??,`
`??, ??, 98, ??, ??, ??, ??,`
`??, ??, ??`
coupling_tag, **103, 109, 99**, used:
coupling_tag (field), **108**, used:
113, 117
coupling_tag_raw, **109**, used: **109**
coupling_to_string, **106, 106,**
107, 102, used: **109**
Cpp (module), **242, 184**, used:
create, **307, 292, 289, 289**, used:
307, 292, 292, 292, ??,
181, 181, 183, 107, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, 190, ??, ??
created (field), **291**, used: **292,**
293
create_class, **31**, used: **33**
CRing (sig), **362, 359**, used: **362,**
359
cross, **122**, used: **122**
crossing, **111**, used: **111, 113**
cross_uncolored, **157, 166**, used:
157, 166
CS (module), **72**, used: **74**
csign (type), **??**, used: **??, ??**
Custom, **??**, used: **??, ??, ??, ??,**
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??
Cycle (exn), **42, 47, 39**, used:
C_12_34, **??**, used: **??, ??, ??,**
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, 280, ??, ??
C_13_42, **??**, used: **??, ??, ??,**
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, 280, ??, ??
C_14_23, **??**, used: **??, ??, ??,**
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, 280, ??, ??
d (field), **55**, used: **56, 56, 56, 56,**
56, 56, 57, 57, 57, 58, 58,
58, 59, 59, 59, 60, 66
D, **??, ??, ??, ??, ??, ??, ??, ??,**
??, ??, ??, ??, 279, ??,
??, used: **??, ??, ??, ??,**
??, ??, ??, ??, ??, ??, ??,

`??`, `??`, `??`, `??`, `??`, `??`, `??`,
`??`, `??`, `??`, `??`, `??`, `??`, `??`,
`??`, `??`, `??`, `??`, `??`, `??`, `??`,
`??`, `??`, `??`, `??`, `??`, `??`, `??`,
`??`, `??`, `??`, `??`, `??`, `??`, `??`,
`??`, `??`, `??`, `??`, `??`, `??`, `??`,
`??`, `??`, `??`, `??`, `??`, `??`, `??`,
`??`, `??`, `??`, `??`, `??`, `??`, `??`,
`??`, `??`, `??`, `??`, `??`, `??`, `??`,
`??`, `??`, `??`, `??`, `??`, `??`, `??`,
`??`, `??`, `??`, `??`, `??`, `??`, `??`,
`??`, `??`, `??`, `??`, `??`, `??`, `??`,
`??`, `??`, `??`, `??`, `??`, `??`, `??`,
`??`, `??`, `??`, `??`, `??`, `??`, `??`,
`??`, `??`, `??`, `??`, `??`, `??`, `??`,
`??`, `??`, `??`

D (module), **114**, used: **115**, **115**,
116, **116**, **116**, **116**, **116**,
117, **117**, **117**, **120**, **121**,
121, **121**, **122**, **122**, **122**,
124, **124**, **124**

D' (module), **115**, used: **115**

DAG (module), **42**, **37**, used:
114, **114**, **115**

dag_to_dot, **124**, used: **124**, **124**

data (field), **344**, **345**, used: **344**,
344, **345**, **345**

date, **287**, **286**, used: **288**

date (field), **287**, **286**, used: **288**,
337, **10**, **22**, **42**, **54**, `??`,
262, **174**, **103**, `??`, `??`, `??`,
`??`, `??`, **244**, **184**, **278**, `??`,
`??`

Dbar, **279**, used: **279**, **279**, **280**,
280

debug, **272**, used: **275**

decay (type), **256**, **257**, **273**,
255, used: **256**, **257**, **259**,
260, **273**, **255**, **255**, **255**,
255

Decay, **273**, used: **273**, **273**, **274**

decays (field), **260**, used: **260**,
260, **260**, **261**

decay_of_momenta, **256**, **259**,
255, used:

declarations (type), **191**, used:
declare, **??**, used: **??**
declare_argument, **267**, used:
267, 267
declare_brackets, **198**, used: **199**
declare_default_parameters, **194**,
used: **196**
declare_list, **191**, used: **198**
declare_momenta, **198**, used: **199**
declare_parameters, **194**, used:
196
declare_parameters', **194**, used:
194
declare_parameter_array, **194**,
used: **196**
declare_wavefunctions, **198**,
used: **199**
decode, **278**, used: **278**
Default (module), **66, 358, 54,**
355, used: **271, 271, 278,**
54, 271
DefaultW (module), **66, 54**,
used: **271**
default_function, **267**, used: **267**
default_parameter, **194**, used:
194
default_subroutine, **267**, used:
267
default_width, **??, ??, ??, ??, ??,**
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, used: ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??
Delta, **??**, used:
Dense (module), **355, 355**, used:
358, 358
dependencies, **42, 48, 103, 119,**
40, 99, used: **120, 121,**
191, 212, 213, 214
dependencies (field), **117**, used:
119, 120, 121
derive, **363, 365, 360**, used: **366,**
366, 366
derived (field), **??**, used: **181, ??,**
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??

derived_arrays (field), ??, used:
181, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, **193**,
196, **280**, ??, ??
derived_parameters, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, used: ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??
derived_parameter_arrays, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, used: ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??
derive_inner, **363**, **366**, **361**,
 used:
derive_inner', **363**, **366**, **361**,
 used:
derive_outer, **363**, **366**, **361**,
 used: **367**
descend, **319**, **319**, **319**, **320**,
322, **322**, **322**, **323**,
 used: **319**, **319**, **319**, **320**,
322, **322**, **322**, **323**
description, **287**, **30**, **72**, **75**,
286, **72**, used: **288**, **55**,
60, **157**, **159**, **166**, **121**,
131, ??, **225**
description (field), **287**, **74**, used:
287, **75**, **75**, **76**
DH, ??, used:
diag, ??, ??, used: ??, ??, ??, ??
diagnose_arguments, **197**, used:
198, **216**
diagnose_gauge, **197**, used: **198**,
212
diagnose_momenta, **197**, used:
198, **216**
diagnostic (type), **184**, **197**, ??,
 used: ??
diagnostic_mode (type), **197**,
 used:
diagrams, **30**, **33**, **271**, **278**, **278**,
22, **271**, used: **34**
diagrams_per_keystone, **30**, **34**,
22, used: **35**
diagrams_via_keystones, **30**, **35**,
22, used:
diagram_class (type), **31**, used:

- diet*, ??, ??, ??, ??, ??, ??, ??, ??, used: ??, ??
- Diff*, ??, used: ??, ??, ??, ??
- digit* (camllex regexpr), **69**, **172**, used: **69**, **172**, **172**
- digits*, **291**, used: **292**, **293**
- digits* (field), **291**, used: **292**, **293**, **293**
- dim*, **54**, **56**, **61**, **52**, used: **62**, **62**, **62**, **64**, **64**, **65**, **261**, **261**, **73**
- dim0*, **61**, used: **61**, **62**, **62**, **62**, **62**, **63**, **63**
- Dim4_Vector_Vector_Vector_L*, ??, used: **137**, ??, ??
- Dim4_Vector_Vector_Vector_L5*, ??, used: **137**, ??, ??
- Dim4_Vector_Vector_Vector_T*, ??, used: **137**, ??, ??
- Dim4_Vector_Vector_Vector_T5*, ??, used: **137**, ??, ??
- Dim5_Scalar_Gauge2*, ??, used: **137**, ??, ??, ??, ??, ??
- Dim5_Scalar_Gauge2_Skew*, ??, used: **137**, ??, ??, ??
- Dim5_Scalar_Vector_Vector_T*, ??, used: **137**
- Dim5_Tensor_2_Vector_Vector_1*, ??, used: **137**
- Dim5_Tensor_2_Vector_Vector_2*, ??, used: **137**
- Dim6_Gauge_Gauge_Gauge*, ??, used: **137**, ??, ??
- Dim6_Gauge_Gauge_Gauge_5*, ??, used: **137**, ??, ??
- Dim6_Vector_Vector_Vector_T*, ??, used: **137**
- Dim7_Tensor_2_Vector_Vector_T*, ??, used: **137**
- disambiguate_fusions*, **213**, used: **218**
- Discrete* (module), **44**, **41**, used: **33**
- distribute_degrees*, **33**, used: **33**
- distribute_degrees'*, **33**, used: **33**, **33**
- distribute_degrees''*, **33**, used: **33**, **33**
- div*, **355**, **356**, **354**, used: **356**
- DIV*, ??, ??, used: ??, **172**
- DIV* (camlyacc token), **173**, used: **173**, **173**
- divide*, **171**, **170**, used: ??, **173**
- division*, **350**, used: **350**
- Dodd*, ??, used: ??
- dot*, **170**, **170**, used: ??, **173**
- DOT*, ??, ??, used: ??, **172**
- DOT* (camlyacc token), **173**, used: **173**, **173**
- Dotproduct*, **170**, **170**, used: **170**
- drb*, **61**, used: **61**, **61**, **61**, **62**, **64**, **65**
- drb0*, **61**, used: **62**, **62**, **62**, **63**
- DScalar2_Vector2*, ??, used: **138**
- DScalar4*, ??, used: **138**
- dummy*, **292**, **291**, used: **130**
- Dummy* (module), **184**, **184**, used: **242**, **242**, **242**, **242**, **242**, **242**
- dummy_flavor*, **178**, used: **181**
- dump*, **352**, **342**, used: **258**, **51**, **255**, used: **258**, **51**, **255**
- Dynamical* (module), **159**, **133**, used: ??
- D_Alpha_WWWW0_S*, ??, used: ??
- D_Alpha_WWWW0_T*, ??, used: ??
- D_Alpha_WWWW0_U*, ??, used: ??
- D_Alpha_WWWW2_S*, ??, used: ??
- D_Alpha_WWWW2_T*, ??, used: ??
- D_Alpha_ZZWW0_S*, ??, used: ??
- D_Alpha_ZZWW0_T*, ??, used: ??
- D_Alpha_ZZWW1_S*, ??, used: ??
- D_Alpha_ZZWW1_T*, ??, used: ??
- D_Alpha_ZZWW1_U*, ??, used: ??
- D_Alpha_ZZZZ_S*, ??, used: ??
- D_Alpha_ZZZZ_T*, ??, used: ??
- D_K1_L*, ??, used: ??, ??, ??
- D_K1_R*, ??, used: ??, ??, ??
- D_K2_L*, ??, used: ??, ??, ??

- D_K2_R*, ??, used: ??, ??, ??
d_p, **235**, used: **235**, **239**, **239**,
240, **240**, **240**
e (camlyacc non-terminal), **173**,
used: **173**, **173**
E, ??, ??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??,
used: ??, ??
edge (type), **42**, **42**, **43**, **44**, **45**,
46, **114**, **38**, **39**, **41**, used:
42, **42**, **43**, **43**, **44**, **45**, **46**,
114, **38**, **38**, **39**, **39**, **39**, **40**,
40, **40**, **40**, **40**, **41**, **41**, **42**
Edges (module), **114**, used: **114**
edges_feynmf, **346**, used: **347**
edges_feynmf', **346**, used: **346**,
346
Eidelta, ??, used:
electromagnetic_currents, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, used: ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??
electromagnetic_currents', ??,
used: ??
electromagnetic_currents_2, ??,
??, ??, used: ??, ??, ??
electromagnetic_currents_3, ??,
??, ??, used: ??, ??, ??
electromagnetic_sfermion_currents,
??, ??, ??, used: ??, ??,
??
Electron, ??, used: ??, ??, ??, ??
elements, **306**, **312**, **314**, **305**,
used: **365**, **366**, **367**, **368**,
46, **48**, **264**, **113**, **123**, **123**,
199
elements', **312**, used: **312**, **312**
embedding, **302**, **302**, used:
embedding (field), **302**, used: **302**,
303, **303**, **304**
embedding1, **303**, **302**, used:
embedding1 (field), **302**, used:
303, **303**, **304**
embedding2, **303**, **302**, used:
embedding2 (field), **302**, used:
303, **303**, **304**
embed_in_decay, **258**, used: **258**,
259, **259**
embed_in_decays, **259**, used:
259, **260**
empty, **306**, **307**, **312**, **318**, **318**,
321, **321**, **289**, **42**, **45**,
46, **47**, **50**, **260**, **305**,
316, **317**, **289**, **39**, used:
312, **303**, **318**, **318**, **318**,
319, **320**, **321**, **321**, **322**,
323, **364**, **365**, **365**, **367**,
367, **289**, **289**, **349**, **349**,
31, **46**, **46**, **46**, **47**, **47**, **47**,
47, **47**, **48**, **49**, **49**, **50**, **260**,
??, ??, ??, **264**, **264**, **264**,
266, **109**, **113**, **116**, **116**,
120, **121**, **123**, **123**, **124**,
??, ??, ??, ??, **184**, **199**,
213, **218**, **279**
Empty, **306**, used: **307**, **307**, **308**,
309, **309**, **310**, **311**, **311**,
312
em_lqino_currents, ??, used: ??
em_lq_currents, ??, used: ??
em_up_type_currents, ??, used:
??
END, ??, ??, ??, ??, used: ??,
??, **69**, **172**
END (camlyacc token), **70**, **173**,
used: **70**, **173**
end_step, **293**, **291**, used: **130**
equal, **318**, **321**, ??, ??, ??, **316**,
used:
Eta, ??, ??, ??, used: ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??
eta_higgs_gauge, ??, used: ??
eval, **42**, **48**, **256**, **259**, **40**, **255**,
used: **49**, **49**
eval_decay, **256**, **257**, **255**, used:
257, **259**
eval_memoized, **42**, **49**, **40**, used:
49, **122**
eval_memoized', **49**, used: **49**
eval_offspring, **49**, used: **49**
eval_parameter, **195**, used: **195**
eval_parameter', **194**, used: **194**,
195, **195**
eval_parameter_pair, **195**, used:
195
eval_para_list, **195**, used: **196**

eval_para_pair_list, **195**, used:
 196

exists, **295, 296, 294**, used: **34,**
 130, 130

exp, **355, 357, 354**, used: **357**

expand_decays, **274**, used: **275**

expand_scatterings, **274**, used:
 275

export, **318, 320, 321, 323, 317,**
 317, used:

export', **320, 323**, used: **320, 320,**
 323, 323

expr, **??, ??**, used: **179**

expr (type), **??**, used: **??, ??, ??,**
 ??

expr (camlyacc non-terminal),
 173, used: **173**

ext (field), **31**, used: **31, 32, 32,**
 32, 32

extend, **289, 289**, used: **289, 128**

externals, **103, 119, 100**, used:
 264, 275, 278, 199, 215,
 216, 217, 217

externals (field), **117**, used: **119,**
 120

external_color_flows, **156, 165,**
 used: **156, 166**

external_edges, **349**, used: **349**

external_edges_from, **349**, used:
 349, 349

external_flavors, **??, 177, 137,**
 157, 161, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, 279, ??, ??, ??, used:
 177, 137, 157, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, 279, ??, ??

external_flavors', **??**, used: **??**

external_flavors'', **??**, used: **??**

external_wfs, **119**, used: **120**

ExtMSSM (module), **??, ??**,
 used: **??**

extMSSM_flags (sig), **??, ??**,
 used: **??, ??, ??, ??**

ext_edges (field), **349**, used: **349,**
 350, 351

ext_incidence, **350**, used: **351**

ext_layout (type), **350, 342**,
used: **350, 342, 342**
ext_momentum, **191**, used: **191**,
217
ext_nodes (field), **349**, used: **350**,
350, 351, 351, 352, 352,
352
f (type), **??, ??, ??, ??, 177**,
181, 137, 161, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, 280, ??, ??,
??, ??, ??, used: **??, ??**,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??
F (module), **263, 181, 114, 128**,
272, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
189, 280, ??, ??, used:
263, 263, 264, 266, 181,
114, 128, 128, 130, 130,
130, 130, 130, 131, 272,
272, 272, 275, 278, 278,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, 189, 191,
191, 191, 191, 191, 191,
191, 192, 198, 199, 199,
199, 202, 207, 212, 212,
213, 213, 214, 214, 215,
215, 215, 216, 216, 217,
217, 217, 217, 226, 280,
??, ??
F12, **??**, used: **??**
F123, **??**, used: **??**
F124, **??**, used: **??**
F13, **??**, used: **??**
F132, **??**, used: **??**
F134, **??**, used: **??**
F142, **??**, used: **??**
F143, **??**, used: **??**
F2 (module), **??**, used: **??, ??**
F2' (module), **??**, used: **??, ??**
F21, **??**, used: **??**
F213, **??**, used: **??**
F214, **??**, used: **??**
F23, **??**, used: **??, ??**
F231, **??**, used: **??**
F234, **??**, used: **??, ??**

[illegible]

[illegible][illegible]

- flavors*, ??, **177**, **137**, **157**, **161**,
127, **128**, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, **198**, **279**, ??, ??,
102, ??, used: **177**, **183**,
137, **157**, **121**, **121**, **275**,
 ??, ??, ??, **198**, **215**, **217**,
218, **219**, **219**, **219**, **220**,
220, **225**
- flavors* (field), **128**, used: **128**
- flavors* (label), ??, ??, used:
- flavors* (type), **128**, used:
- flavors_of_particles*, **181**, used:
182
- flavors_sans_color_to_string*,
217, used: **217**
- flavors_symbol*, **190**, used: **198**,
215, **217**, **218**
- flavors_to_string*, **265**, **74**, **217**,
 used: **265**, **265**, **266**, **74**,
217
- flavor_keystone*, **111**, used: **112**
- flavor_keystones*, **112**, used: **121**
- flavor_list* (camlyacc
 non-terminal), **71**, used:
70, **71**
- flavor_of_antiparticle*, **179**, used:
181
- flavor_of_b*, ??, used: ??, ??
- flavor_of_f*, ??, used: ??, ??
- flavor_of_particle*, **179**, used:
181
- flavor_of_string*, ??, **177**, **154**,
157, **163**, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, **281**, ??, ??
- flavor_of_string* (label), ??, ??,
 used:
- flavor_sans_color*, ??, **177**, **134**,
157, **159**, **108**, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, **279**, ??, ??,
- ??, used: **134**, **157**, **160**,
162, **163**, **163**, **108**, **113**,
121, **121**, **128**, ??
- flavor_sans_color* (type), ??,
177, **134**, **157**, **159**, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, **279**, ??,
 ??, ??, used: **72**, **72**, **157**,
128, ??, **72**, ??, ??, ??,
 ??, ??, ??, ??
- flavor_sans_color_of_string*, ??,
177, **154**, **157**, **163**, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, **281**, ??,
 ??, ??, used: **73**, **157**, ??
- flavor_sans_color_symbol*, ??,
177, **155**, **157**, **164**, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, **281**, ??,
 ??, ??, used: **157**, ??
- flavor_sans_color_to_string*, ??,
177, **154**, **157**, **164**, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, **281**, ??,
 ??, ??, used: **157**, ??
- flavor_sans_color_to_TeX*, ??,
177, **155**, **157**, **164**, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, **281**, ??,
 ??, ??, used: **157**, ??
- Flavor_Set* (module), **264**, used:
264, **264**, **264**
- flavor_symbol*, ??, **177**, **155**,
157, **164**, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, **281**, ??, ??, ??,
 used: **177**, **155**, **155**, **157**,
164, **164**, **124**, **272**, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
190, **191**, **212**, **281**, ??, ??,
 ??, ??

- flavor_symbol* (label), ??, ??,
used: 181
- flavor_to_string*, ??, 177, 154,
157, 163, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, 281, ??, ??, ??, used:
265, 266, 177, 183, 154,
154, 157, 163, 164, 128,
272, 275, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, 217, 217, 220,
281, ??, ??
- flavor_to_string* (label), ??, ??,
used: 181
- flavor_to_TeX*, ??, 177, 154,
157, 164, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, 281, ??, ??, ??, used:
177, 154, 155, 157, 164,
164, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, 281, ??, ??
- flavor_to_TeX* (label), ??, ??,
used: 181
- flip_incoming*, 260, used: 260
- flip_s_channel_in*, 54, 59, 64,
54, used: 260, 263
- flow*, 157, 159, 166, ??, ??,
used: 159, 130
- Flow* (module), 81, 77, used:
157, 166, 127, 128, 128,
219, 102, 102, ??, ??
- Flow* (sig), 80, 77, used: 81, 77
- Flows* (sig), 133, 133, used: 133,
157, 133, 133
- Fn* (module), ??, used: ??
- fold*, 306, 311, 315, 318, 319,
321, 322, 325, 344, 42,
42, 43, 44, 45, 45, 46,
48, 50, 114, 305, 316,
317, 324, 341, 38, 40,
41, used: 311, 312, 315,
319, 322, 365, 365, 365,
365, 366, 366, 366, 366,
366, 366, 367, 367, 367,
368, 325, 344, 344, 345,
17, 18, 18, 32, 45, 46, 46,
46, 47, 47, 47, 48, 48, 48,
48, 48, 48, 48, 49, 49, 50,
264, 109, 114, 120, 121,
124, 213
- fold* (type), 326, 324, used: 324
- fold'*, 319, 322, used: 319, 322
- fold2*, 366, 325, 324, used: 366,
325, 325, 12, 12, 14, 16,
49, 111, 112, 122
- fold2* (type), 326, 324, used: 324
- fold2_rev*, 325, used: 325
- fold3*, 325, 324, used: 325, 13,
14, 16
- fold3_rev*, 325, used: 325
- fold_left*, 10, 12, 13, 15, 17, 7,
used: 299, 299, 365, 366,
366, 366, 366, 367, 367,
289, 325, 325, 325, 325,
331, 331, 332, 332, 332,
333, 333, 333, 333, 333,
334, 335, 336, 336, 349,
349, 351, 17, 28, 28, 31,
31, 31, 33, 33, 34, 34, 34,
34, 35, 48, 49, 61, ??, ??,
263, 264, 264, 178, 181,
181, 73, 107, 113, 113,
115, 116, 116, 117, 122,
122, 128, 195, 213, 213,
214, 217, 219, 220, 220,
221
- fold_left2*, 299, 298, used: 199,
218
- fold_left_internal*, 10, 12, 13,
15, 17, 7, used: 113
- fold_lines*, 175, used: 176
- fold_nodes*, 42, 48, 39, used:
116, 124
- fold_rev*, 325, used: 325, 325
- fold_right*, 10, 12, 13, 15, 17, 7,
used: 299, 338, 346, 12,
12, 14, 14, 16, 17, 18, 18,
44, 259, 260, 264, 264,
266, 171, 176, 74, 109,
113, 123, 199
- fold_right2*, 299, 298, used: 123
- fold_right_internal*, 10, 12, 13,
15, 17, 7, used: 107
- forest*, 42, 49, 103, 122, 40,
100, used: 275, 278

- Forest* (module), **44, 38**, used:
114, 114
- Forest* (sig), **42, 38**, used: 43, 44,
45, 50, 38, 40, 41
- forest'*, **122**, used: 122
- Forest_Grader* (sig), **43, 41**,
used: 41
- forest_memoized*, **42, 49, 40**,
used: 122
- format_coeff*, **200**, used: 200,
200, 201, 201
- format_constant*, **194**, used: 212
- format_coupling*, **187, 200, 229**,
used: 187, 202, 203, 203,
203, 203, 203, 203, 204,
204, 204, 204, 205, 205,
205, 205, 205, 205, 206,
206, 206, 206, 206, 207,
207, 207, 207, 207, 207,
230, 230, 231, 233, 233,
234, 234, 234, 234, 234,
235, 236, 238, 239, 239,
240, 240
- format_coupling_2*, **187, 229**,
used: 188, 230, 230, 231,
236, 237, 237, 238, 238,
239, 240
- format_coupling_mom*, **232**,
used: 232, 233, 233
- format_flavor*, **266**, used: 266
- format_generation*, ??, used: ??,
??
- format_maps*, **265**, used: 266
- format_momentum*, **191**, used:
198
- format_multiple_variable*, **191**,
used: 191, 191
- format_p*, **272, 191**, used: 272,
191, 191, 212
- Fortran* (module), **244, 228**,
244, 184, used: 227, 278,
282, 282, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??
- Fortran77* (module), **242, 184**,
used:
- fortran95*, **189**, used: 190, 198,
218, 222, 223, 223, 223,
224
- Fortran_Fermions* (module),
186, used: 228
- Fortran_Majorana* (module),
242, 184, used: ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??
- Fortran_Majorana_Fermions*
(module), **228**, used:
242
- for_all*, **10, 12, 13, 15, 17, 42**,
43, 44, 45, 114, 7, 38,
41, used: 299, 17, 33, 44,
45, 47, 73, 75, 75, 76, 76,
153, 162, 113, 114, 117
- four_gluon*, **280**, used: 280
- fspec_of_gen*, ??, used: ??
- fspec_of_iso*, ??, used: ??
- fspec_of_kk2*, ??, used: ??
- fspec_of_kkmode*, ??, used: ??
- Fudged*, ??, used: ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??
- functions_file*, **182**, used: 182,
183
- fuse*, **343**, ??, ??, ??, ??, **177, 154**,
157, 163, 106, 106, 107,
113, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
280, ??, ??, **341**, ??,
102, ??, used: 177, 181,
154, 157, 163, 109, 115,
122, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
280, ??, ??
- fuse* (label), ??, ??, used: 181
- fuse2*, **12**, ??, ??, ??, **177, 154**,
157, 162, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, **280**, ??, ??, ??, ??,
used: 12, 16, ??, ??, 177,
181, 154, 157, 162, 163,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, 280,
??, ??
- fuse2* (type), ??, used: 185, ??
- fuse3*, **14**, ??, ??, ??, **177, 154**,
157, 163, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,

Ga, ??, ??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??, ??,
used: ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??, ??,
gauge (type), ??, **177**, **134**, **157**,
160, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??, ??,
279, ??, ??, ??, used:
177, **178**, **134**, **157**, **160**,
??, ??, ??, ??, ??
Gauge, **158**, ??, ??, ??, ??, ??, ??,
??, ??, ??, **184**, **197**, ??,
??, ??, ??, used: **275**, ??,
??, ??, ??, ??, ??, ??, ??, ??,
198, ??, ??
Gauge (module), **157**, **133**, used:

Gauge (sig), ??, used: **157**, ??,
133, ??, ??, ??, ??, ??, ??, ??
gauge4, ??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
280, ??, ??, used: ??, ??,
??, ??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
280, ??, ??
gauge_boson, **158**, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??, ??,
??, used: ??, ??, ??, ??,

??, ??, ??, ??, ??, ??, ??
gauge_boson (type), **158**, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, used: **158**, **158**,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??
Gauge_Gauge_Gauge, ??, used:
137, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
280, ??, ??, ??
gauge_higgs, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, used: ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??
gauge_higgs4, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, used: ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??
gauge_higgs4-GaGaCC, ??, ??,
 used: ??, ??
gauge_higgs4-GaWPC, ??, ??,
 used: ??, ??
gauge_higgs4-GaWSC, ??, ??,
 used: ??, ??
gauge_higgs4-WWCC, ??, ??,
 used: ??, ??
gauge_higgs4-WWPP, ??, ??,
 used: ??, ??
gauge_higgs4-WWSS, ??, ??,
 used: ??, ??
gauge_higgs4-ZGaCC, ??, ??,
 used: ??, ??
gauge_higgs4-ZWPC, ??, ??,
 used: ??, ??
gauge_higgs4-ZWSC, ??, ??,
 used: ??, ??
gauge_higgs4-ZZCC, ??, ??,
 used: ??, ??
gauge_higgs4-ZZPP, ??, ??,
 used: ??, ??
gauge_higgs4-ZZSS, ??, ??,
 used: ??, ??
gauge_higgs-GaCC, ??, ??, used:
 ??, ??
gauge_higgs-gold, ??, used: ??

gauge_higgs_gold4, ??, used: ??
gauge_higgs_WPC, ??, ??, used:
 ??, ??
gauge_higgs_WSC, ??, ??, used:
 ??, ??
gauge_higgs_WWS, ??, ??, used:
 ??, ??
gauge_higgs_ZCC, ??, ??, used:
 ??, ??
gauge_higgs_ZSP, ??, ??, used:
 ??, ??
gauge_higgs_ZZS, ??, ??, used:
 ??, ??
gauge_sferrnion4, ??, ??, ??,
 used: ??, ??, ??
gauge_sferrnion4', ??, ??, ??,
 used: ??, ??, ??
gauge_sferrnion4'', ??, ??, ??,
 used: ??, ??, ??
gauge_squark4, ??, ??, ??, used:
 ??, ??, ??
gauge_squark4', ??, ??, ??, used:
 ??, ??, ??
gauge_symbol, ??, **177**, **155**,
157, **164**, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, **282**, ??, ??, ??,
 used: **177**, **155**, **157**, **164**,
 ??, **210**
gauge_symbol (label), ??, ??,
 used: **181**
Gauss, **68**, **73**, **67**, used: **68**, **73**
GAUSS, ??, ??, used: ??, **69**
GAUSS (camlyacc token), **70**,
 used: **70**
Gauss_not, **68**, **73**, **67**, used: **68**
GBBG, ??, used: **138**, ??, ??, ??,
 ??, ??, ??, ??, ??
GBG, ??, used: **137**, ??, ??, ??,
 ??, ??, ??, ??
gcd, **362**, used: **362**, **362**, **363**
gen (type), ??, ??, ??, used: ??,
 ??, ??
Gen0, ??, used: ??
Gen1, ??, used: ??
Gen2, ??, used: ??
General_Flow (module), **81**,
 used:

[illegible]

- gluon_gauge_squark'*, ??, ??, ??,
 used: ??, ??, ??
gluon_gauge_squark'', ??, ??, ??,
 used: ??, ??, ??
gluon_w_squark, ??, ??, ??,
 used: ??, ??, ??
gluon_w_squark', ??, ??, ??,
 used: ??, ??, ??
Gl_K1, ??, used: ??, ??, ??, ??,
 ??
Gl_K2, ??, used: ??, ??, ??, ??,
 ??
gobble_arrows, **175**, used: **176**
gobble_white, **175**, used: **175**,
175, **176**
goldstone, ??, **177**, **135**, **157**,
160, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??, ??,
279, ??, ??, ??, ??, used:
177, **135**, **157**, **160**, **108**,
116, ??, ??, **212**
goldstone (label), ??, ??, used:
181
goldstone (type), ??, used: ??, ??
Goldstone, ??, used:
goldstone4, ??, used: ??
goldstone_charg_neutr, ??, used:
 ??
goldstone_neutr, ??, used: ??
goldstone_neutr', ??, used: ??
goldstone_neutr'', ??, used: ??
goldstone_sfermion, ??, used: ??
goldstone_sfermion', ??, used: ??
goldstone_sfermion'', ??, used:
 ??
goldstone_sfermion''', ??, used:
 ??
goldstone_squark, ??, used: ??
goldstone_squark', ??, used: ??
goldstone_squark_a, ??, used: ??
goldstone_squark_b, ??, used: ??
goldstone_vertices, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??, ??,
 ??, used: ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??, ??
graded (type), **10**, **12**, **14**, **16**,
18, **8**, used: **10**, **8**
Graded (module), **49**, **42**, used:
115
- Graded* (sig), **49**, **41**, used: **42**
Graded_Forest (sig), **43**, **41**,
 used: **43**, **46**, **49**, **41**, **42**
Graded_Map (module), **45**, used:
49
Graded_Map (sig), **45**, used: **45**,
45, **50**
Graded_Map_Maker (sig), **45**,
 used: **46**
Graded_Ord (sig), **43**, **41**, used:
43, **43**, **45**, **45**, **50**, **41**, **41**
graded_sym_power, **10**, **12**, **14**,
16, **18**, **8**, used:
14, **16**, **18**, **8**, used: **12**,
14, **16**, **18**, **115**
Grader (sig), **43**, **41**, used: **43**,
45, **41**, **41**
Grade_Forest (module), **45**, **41**,
 used: **50**
graph (type), **349**, **342**, used:
350, **342**, **342**
graph_of_tree, **349**, **342**, used:
Grav, ??, ??, ??, used: ??, ??,
 ??, ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??
Gravbar, ??, used: ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??, ??,
 ??, ??
gravitino, ??, ??, ??, ??, ??, ??, ??,
 ??, used: ??, ??, ??
gravitino_coup, ??, used: ??
gravitino_gauge, ??, used: ??
gravitino_gaugino_3, ??, used:
 ??
Graviton_Scalar_Scalar, ??,
 used: **137**, ??, ??
Graviton_Spinor_Spinor, ??,
 used: **137**, ??, ??
Graviton_Vector_Vector, ??,
 used: **137**, ??, ??
gravity_currents, ??, ??, used:
 ??, ??
gravity_gauge, ??, ??, used: ??,
 ??
gravity_higgs, ??, ??, used: ??,
 ??
GravTest (module), ??, ??, used:
 ??

- grav_gauss*, 185, 187, 229, used:
- grav_incoming*, 185, 186, 229,
used: 215, 215
- grav_outgoing*, 185, 186, 229,
used:
- grav_projector*, 185, 186, 229,
used:
- grav_propagator*, 185, 186, 229,
used:
- grav_type*, 185, 186, 228, used:
198
- Grino*, ??, ??, used: ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??
- Groves* (module), ??, ??, used:
??
- grow*, 115, used: 115
- Gs*, ??, ??, ??, ??, ??, ??, ??,
??, ??, ??, 279, ??, used:
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, 280, 280,
280, ??
- G_AASFSF*, ??, used: ??
- G_aaww*, ??, used: ??, ??
- G_AAWW*, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, used:
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??
- G_AG0SFSF*, ??, used: ??, ??
- G_AGSNSL*, ??, used: ??
- G_AGSUSD*, ??, used: ??
- G_AHPsip*, ??, ??, used: ??, ??
- G_AHTHT*, ??, ??, used: ??, ??
- G_AHTHTH*, ??, ??, used: ??,
??
- G_AHTT*, ??, ??, used: ??, ??
- G_ASFSF*, ??, used: ??
- G_auw*, ??, used: ??, ??
- G_AZWW*, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, used:
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??
- G_a_lep*, ??, used: ??, ??
- g_a_quark*, ??, used: ??
- G_a_quark*, ??, used: ??, ??
- G_CAC*, ??, used: ??
- G_CC*, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, used: ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??
- G_CCQ*, ??, ??, ??, ??, ??, ??,
??, used: ??, ??, ??, ??
- G_CCtop*, ??, ??, used: ??, ??
- G_CC_heavy*, ??, ??, used: ??,
??
- G_CC_W*, ??, ??, used: ??, ??
- G_CC_WH*, ??, ??, used: ??, ??
- G_CGC*, ??, used: ??
- G_CGN*, ??, used: ??
- G_CH1C*, ??, used: ??
- G_CH2C*, ??, used: ??
- G_CHN*, ??, used: ??
- G_CICIA*, ??, used: ??
- G_CICIG*, ??, used: ??
- G_CICIH1*, ??, used: ??
- G_CICIH2*, ??, used: ??
- G_CICIP*, ??, ??, used: ??, ??
- G_CICIS*, ??, ??, used: ??, ??
- G_CPC*, ??, ??, used: ??, ??
- G_CSC*, ??, ??, used: ??, ??
- G_CWN*, ??, ??, ??, used: ??,
??, ??
- G_CZC*, ??, ??, ??, used: ??,
??, ??
- G_DH2W2*, ??, used: ??
- G_DH2Z2*, ??, used: ??
- G_DHW2*, ??, used: ??
- G_DHZ2*, ??, used: ??
- G_Ebb*, ??, ??, ??, used: ??, ??,
??
- G_EGaGa*, ??, ??, ??, used: ??,
??, ??
- G_EGaZ*, ??, ??, ??, used: ??,
??, ??
- G_EGIGl*, ??, ??, ??, used: ??,
??, ??
- G_Etht*, ??, ??, ??, used: ??, ??,
??
- G_Ethth*, ??, ??, ??, used:
- G_Ett*, ??, ??, ??, used: ??, ??,
??
- G_G0G0SFSF*, ??, used: ??, ??
- G_GG4*, ??, used: ??
- G_GGSFSF*, ??, used: ??, ??

G_GGSNSL, ??, used: ??
G_GGSUSD, ??, used: ??
G_GH, ??, used: ??, ??
G_GH4, ??, used: ??
G_GH4_GaGaCC, ??, ??, used: ??, ??
G_GH4_GaWPC, ??, ??, used: ??, ??
G_GH4_GaWSC, ??, ??, used: ??, ??
G_GH4_WWCC, ??, ??, used: ??, ??
G_GH4_WWPP, ??, ??, used: ??, ??
G_GH4_WWSS, ??, ??, used: ??, ??
G_GH4_ZGaCC, ??, ??, used: ??, ??
G_GH4_ZWPC, ??, ??, used: ??, ??
G_GH4_ZWSC, ??, ??, used: ??, ??
G_GH4_ZZCC, ??, ??, used: ??, ??
G_GH4_ZZPP, ??, ??, used: ??, ??
G_GH4_ZZSS, ??, ??, used: ??, ??
G_GHGo, ??, used: ??
G_GHGo4, ??, used: ??
G_GH_GaCC, ??, ??, used: ??, ??
G_GH_WPC, ??, ??, used: ??, ??
G_GH_WSC, ??, ??, used: ??, ??
G_GH_WWS, ??, ??, used: ??, ??
G_GH_ZCC, ??, ??, used: ??, ??
G_GH_ZSP, ??, ??, used: ??, ??
G_GH_ZZS, ??, ??, used: ??, ??
G_GIGILQLQ, ??, used: ??
G_GIGISQSQ, ??, ??, ??, used: ??, ??, ??
G_GIPSQSQ, ??, ??, ??, used: ??, ??, ??
G_GIWSUSD, ??, ??, ??, used: ??, ??, ??
G_GIZSFSF, ??, ??, ??, used: ??, ??, ??
G_GoSFSF, ??, used: ??
G_GoSNSL, ??, used: ??
G_Gr4A_Sd, ??, used: ??
G_Gr4A_Sdc, ??, used: ??
G_Gr4A_Sl, ??, used: ??
G_Gr4A_Slc, ??, used: ??
G_Gr4A_Su, ??, used: ??
G_Gr4A_Suc, ??, used: ??
G_Gr4Gl_Sd, ??, used: ??
G_Gr4Gl_Sdc, ??, used: ??
G_Gr4Gl_Su, ??, used: ??
G_Gr4Gl_Suc, ??, used: ??
G_Gr4W_Sd, ??, used: ??
G_Gr4W_Sdc, ??, used: ??
G_Gr4W_Sl, ??, used: ??
G_Gr4W_Slc, ??, used: ??
G_Gr4W_Sn, ??, used: ??
G_Gr4W_Snc, ??, used: ??
G_Gr4W_Su, ??, used: ??
G_Gr4W_Suc, ??, used: ??
G_Gr4Z_Sd, ??, used: ??
G_Gr4Z_Sdc, ??, used: ??
G_Gr4Z_Sl, ??, used: ??
G_Gr4Z_Slc, ??, used: ??
G_Gr4Z_Sn, ??, used: ??
G_Gr4Z_Snc, ??, used: ??
G_Gr4Z_Su, ??, used: ??
G_Gr4Z_Suc, ??, used: ??
G_Gr4_A_Ch, ??, used: ??
G_Gr4_H_A, ??, used: ??
G_Gr4_H_Z, ??, used: ??
G_Gr4_Neu, ??, used: ??
G_Gr4_W_H, ??, used: ??
G_Gr4_W_Hc, ??, used: ??
G_Gr4_Z_Ch, ??, used: ??
G_Gr4_Z_H1, ??, used: ??
G_Gr4_Z_H2, ??, used: ??
G_Gr4_Z_H3, ??, used: ??
G_Grav, ??, ??, ??, ??, used: ??, ??, ??, ??, ??, ??, ??, ??
G_GravGl, ??, used: ??
G_Grav_D, ??, used: ??
G_Grav_Dc, ??, used: ??
G_Grav_L, ??, used: ??
G_Grav_Lc, ??, used: ??
G_Grav_N, ??, used: ??
G_Grav_U, ??, used: ??

- G_Grav_Uc*, ??, used: ??
G_Gr_A_Neu, ??, used: ??
G_Gr_Ch, ??, used: ??
G_Gr_H1_Neu, ??, used: ??
G_Gr_H2_Neu, ??, used: ??
G_Gr_H3_Neu, ??, used: ??
G_Gr_H_Ch, ??, used: ??
G_Gr_Z_Neu, ??, used: ??
G_GSUSD, ??, used: ??
G_H1GSNSL, ??, used: ??
G_H1GSUSD, ??, used: ??
G_H1H1SFSF, ??, used: ??, ??
G_H1H2SFSF, ??, used: ??, ??
G_H1SFSF, ??, used: ??, ??
G_H2GSNSL, ??, used: ??
G_H2GSUSD, ??, used: ??
G_H2H2SFSF, ??, used: ??, ??
G_H2SFSF, ??, used: ??, ??
G_H3, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, used: ??, ??, ??, ??, ??, ??, ??, ??
G_H3_SCC, ??, ??, used: ??, ??
G_H3_SPP, ??, ??, used: ??, ??
G_H3_SSS, ??, ??, used: ??, ??
G_H4, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, used: ??, ??, ??, ??, ??, ??, ??, ??
G_HAHAAH, ??, ??, used: ??, ??
G_HAHZ, ??, ??, used: ??, ??
G_HASLSN, ??, used: ??
G_HASUSD, ??, used: ??
G_Hbb, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, used: ??, ??, ??, ??, ??, ??, ??, ??
G_Hcc, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, used: ??, ??, ??, ??, ??, ??, ??, ??
G_heavy_HVV, ??, used: ??
G_heavy_HWW, ??, used: ??
G_heavy_HZZ, ??, used: ??
G_HGaGa, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, used: ??, ??, ??, ??, ??
G_HGaZ, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, used: ??, ??, ??, ??, ??
G_Hgg, ??, ??, ??, ??, ??, ??, ??, ??, used: ??, ??, ??, ??, ??, ??
G_HGo3, ??, used: ??
G_HGo4, ??, used: ??
G_HGSFSF, ??, used: ??, ??
G_HGSNSL, ??, used: ??
G_HGSUSD, ??, used: ??
G_HH1SLSN, ??, used: ??
G_HH1SUSD, ??, used: ??
G_HH2SLSN, ??, used: ??
G_HH2SUSD, ??, used: ??
G_HHAA, ??, ??, used: ??, ??
G_HHAHZ, ??, ??, used: ??, ??
G_HHSFSF, ??, used: ??, ??
G_HHtht, ??, ??, used: ??, ??
G_HHthth, ??, ??, ??, used: ??, ??, ??
G_HHtt, ??, ??, used: ??, ??
G_HHWW, ??, ??, used: ??, ??
G_HHWW, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, used: ??, ??, ??, ??, ??, ??, ??, ??
G_HHZHAAH, ??, ??, used: ??, ??
G_HHZHZ, ??, ??, used: ??, ??
G_HHZZ, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, used: ??, ??, ??, ??, ??, ??, ??, ??
G_HHZZH, ??, used: ??
G_Hmm, ??, ??, ??, used: ??, ??, ??
G_HPsi0AAAH, ??, ??, used: ??, ??
G_HPsi0WHW, ??, ??, used: ??, ??
G_HPsi0WW, ??, ??, used: ??, ??
G_HPsi0ZAAH, ??, ??, used: ??, ??
G_HPsi0ZAAH, ??, ??, used: ??, ??
G_HPsi0ZHZ, ??, ??, used: ??, ??
G_HPsi0ZHZH, ??, ??, used: ??, ??

G_HPsi0ZZ, ??, ??, used: ??, ??
G_HPsippWHW, ??, ??, used:
 ??, ??
G_HPsippWHWH, ??, ??, used:
 ??, ??
G_HPsippWW, ??, ??, used: ??,
 ??
G_HPsipWA, ??, ??, used: ??,
 ??
G_HPsipWAH, ??, ??, used: ??,
 ??
G_HPsipWHA, ??, ??, used: ??,
 ??
G_HPsipWHAH, ??, ??, used:
 ??, ??
G_HPsipWHZ, ??, ??, used: ??,
 ??
G_HPsipWHZH, ??, ??, used:
 ??, ??
G_HPsipWZ, ??, ??, used: ??,
 ??
G_HPsipWZH, ??, ??, used: ??,
 ??
G_Hghq, ??, used: ??
G_HSF31, ??, used:
G_HSF32, ??, used:
G_HSF41, ??, used:
G_HSF42, ??, used:
G_HSNSL, ??, ??, ??, used: ??,
 ??, ??
G_HSUSD, ??, ??, ??, used: ??,
 ??, ??
G_Htautau, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??,
 used: ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??
G_Htht, ??, ??, ??, used: ??, ??,
 ??
G_Hthth, ??, ??, ??, used: ??,
 ??
G_Htt, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, used:
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??
G_HWHW, ??, ??, used: ??, ??
G_HWHWH, ??, ??, used: ??,
 ??
G_HWW, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, used:

$??, ??, ??, ??, ??, ??, ??,$
 $??, ??, ??, ??$
 $G_HZH\bar{A}H, \quad ??, ??, \text{used: } ??, ??$
 $G_HZHZ, \quad ??, ??, \text{used: } ??, ??$
 $G_HZZ, \quad ??, ??, ??, ??, ??, ??,$
 $??, ??, ??, ??, ??, \text{used:}$
 $??, ??, ??, ??, ??, ??, ??,$
 $??, ??, ??, ??$
 $G_LQ_EC_UC, \quad ??, \text{used: } ??,$
 $??, ??, ??, ??, ??, ??, ??,$
 $??, ??, ??, ??$
 $G_LQ_GG, \quad ??, \text{used: } ??$
 $G_LQ_NEU, \quad ??, \text{used: } ??, ??$
 $G_LQ_P, \quad ??, \text{used: } ??$
 $G_LQ_S, \quad ??, \text{used: } ??$
 $G_LQ_SSD, \quad ??, \text{used: } ??$
 $G_LQ_SSU, \quad ??, \text{used: } ??$
 $G_NCHt, \quad ??, \text{used: } ??$
 $G_NCH_D, \quad ??, \text{used: } ??$
 $G_NCH_N, \quad ??, \text{used: } ??$
 $G_NCH_U, \quad ??, \text{used: } ??$
 $G_NC_down, \quad ??, ??, ??, ??, ??,$
 $??, ??, ??, ??, ??, ??, ??,$
 $??, ??, \text{used: } ??, ??, ??,$
 $??, ??, ??, ??, ??, ??, ??,$
 $??, ??, ??, ??, ??, ??$
 $G_NC_H, \quad ??, \text{used: } ??$
 $G_NC_heavy, \quad ??, ??, \text{used: } ??,$
 $??$
 $G_NC_h_bot, \quad ??, \text{used: } ??$
 $G_NC_h_down, \quad ??, ??, ??, ??,$
 $\text{used: } ??, ??, ??, ??$
 $G_NC_h_lepton, \quad ??, ??, ??, ??,$
 $\text{used: } ??, ??, ??, ??$
 $G_NC_h_neutrino, \quad ??, ??, ??,$
 $??, \text{used: } ??, ??, ??, ??$
 $G_NC_h_top, \quad ??, \text{used: } ??$
 $G_NC_h_up, \quad ??, ??, ??, ??,$
 $\text{used: } ??, ??, ??, ??$
 $G_NC_lepton, \quad ??, ??, ??, ??,$
 $??, ??, ??, ??, ??, ??, ??,$
 $??, ??, ??, \text{used: } ??, ??,$
 $??, ??, ??, ??, ??, ??, ??,$
 $??, ??, ??, ??, ??, ??, ??$
 $G_NC_neutrino, \quad ??, ??, ??, ??,$
 $??, ??, ??, ??, ??, ??, ??,$
 $??, ??, ??, \text{used: } ??, ??,$
 $??, ??, ??, ??, ??, ??, ??,$
 $??, ??, ??, ??, ??, ??, ??$
 $G_NC_top, \quad ??, \text{used: } ??$

- G_NC_up*, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, used: ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??
G_NC_X, ??, used: ??
G_NC_X_t, ??, used: ??
G_NC_Y, ??, used: ??
G_NC_Y_t, ??, used: ??
G_NGC, ??, used: ??
G_NHC, ??, ??, ??, used: ??,
 ??, ??
G_NLQC, ??, used: ??
G_NWC, ??, ??, ??, used: ??,
 ??, ??
G_NZN, ??, ??, ??, used: ??,
 ??, ??
g_over_2_costh, ??, ??, used:
 ??, ??
G_PGLQLQ, ??, used: ??
G_PPLQLQ, ??, used: ??
G_PPSFSF, ??, ??, ??, used:
 ??, ??, ??
G_PPWW, ??, ??, ??, used: ??,
 ??, ??
G_Psi00AH, ??, ??, used: ??, ??
G_Psi00ZH, ??, ??, used: ??, ??
G_Psi00ZHAH, ??, ??, used: ??,
 ??
G_Psi01AH, ??, ??, used: ??, ??
G_Psi01Z, ??, ??, used: ??, ??
G_Psi01ZH, ??, ??, used: ??, ??
G_Psi0bb, ??, ??, used: ??, ??
G_Psi0cc, ??, ??, used: ??, ??
G_Psi0ppWHW, ??, ??, used:
 ??, ??
G_Psi0ppWHWH, ??, ??, used:
 ??, ??
G_Psi0ppWW, ??, ??, used: ??,
 ??
G_Psi0pWA, ??, ??, used: ??, ??
G_Psi0pWAH, ??, ??, used: ??,
 ??
G_Psi0pWHA, ??, ??, used: ??,
 ??
G_Psi0pWHAH, ??, ??, used:
 ??, ??
G_Psi0pWHZ, ??, ??, used: ??,
 ??
G_Psi0pWHZH, ??, ??, used:
 ??, ??
G_Psi0pWZ, ??, ??, used: ??,
 ??
G_Psi0pWZH, ??, ??, used: ??,
 ??
G_Psi0tautau, ??, ??, used: ??,
 ??
G_Psi0tt, ??, ??, used: ??, ??
G_Psi0th, ??, ??, used: ??, ??
G_Psi0W, ??, ??, used: ??, ??
G_Psi0WH, ??, ??, used: ??, ??
G_Psi1bb, ??, ??, used: ??, ??
G_Psi1cc, ??, ??, used: ??, ??
G_Psi1HAH, ??, ??, used: ??,
 ??
G_Psi1HZ, ??, ??, used: ??, ??
G_Psi1HZH, ??, ??, used: ??,
 ??
G_Psi1tautau, ??, ??, used: ??,
 ??
G_Psi1tt, ??, ??, used: ??, ??
G_Psi1th, ??, ??, used: ??, ??
G_Psi1W, ??, ??, used: ??, ??
G_Psi1WH, ??, ??, used: ??, ??
G_PsiAHAH, ??, ??, used: ??,
 ??
G_PsiAHW, ??, ??, used: ??, ??
G_PsiAHWH, ??, ??, used: ??,
 ??
G_PsiccAAH, ??, ??, used: ??,
 ??
G_PsiccAZ, ??, ??, used: ??, ??
G_PsiccAZH, ??, ??, used: ??,
 ??
G_PsiccZAH, ??, ??, used: ??,
 ??
G_PsiccZZ, ??, ??, used: ??, ??
G_PsiccZZH, ??, ??, used: ??,
 ??
G_PsiHW, ??, ??, used: ??, ??
G_PsiHWH, ??, ??, used: ??, ??
G_Psibpth, ??, ??, used: ??, ??
G_Psipl3, ??, ??, used: ??, ??
G_PsippAAH, ??, ??, used: ??,
 ??
G_PsippAZ, ??, ??, used: ??, ??
G_PsiPPW, ??, ??, used: ??, ??
G_PsippWA, ??, ??, used: ??, ??

G_PsippWAH, ??, ??, used: ??,
 ??
G_PsiPPWH, ??, ??, used: ??,
 ??
G_PsippWHA, ??, ??, used: ??,
 ??
G_PsippWHAH, ??, ??, used:
 ??, ??
G_PsippWHW, ??, ??, used: ??,
 ??
G_PsippWHWH, ??, ??, used:
 ??, ??
G_PsippWHZ, ??, ??, used: ??,
 ??
G_PsippWHZH, ??, ??, used:
 ??, ??
G_PsippWW, ??, ??, used: ??,
 ??
G_PsippWZ, ??, ??, used: ??,
 ??
G_PsippWZH, ??, ??, used: ??,
 ??
G_PsippZAH, ??, ??, used: ??,
 ??
G_PsippZHZH, ??, ??, used: ??,
 ??
G_PsippZZ, ??, ??, used: ??, ??
G_Psipq2, ??, ??, used: ??, ??
G_Psipq3, ??, ??, used: ??, ??
G_PsiWHW, ??, ??, used: ??,
 ??
G_PsiWW, ??, ??, used: ??, ??
G_PsiZAH, ??, ??, used: ??, ??
G_PsiZHAAH, ??, ??, used: ??,
 ??
G_PsiZHW, ??, ??, used: ??, ??
G_PsiZHWH, ??, ??, used: ??,
 ??
G_PsiZHZ, ??, ??, used: ??, ??
G_PsiZHZH, ??, ??, used: ??,
 ??
G_PsiZW, ??, ??, used: ??, ??
G_PsiZWH, ??, ??, used: ??, ??
G_PsiZZ, ??, ??, used: ??, ??
G_PZWW, ??, ??, ??, used: ??,
 ??, ??
G_s, ??, used: ??
G_s2, ??, used: ??
G_SD4, ??, used: ??
G_SD4_2, ??, used: ??
G_SF4, ??, used:
G_SF4_3, ??, used:
G_SF4_4, ??, used:
G_SFSFP, ??, ??, used: ??, ??
G_SFSFS, ??, ??, used: ??, ??,
 ??, ??
G_SL2SQ2, ??, used: ??
G_SL4, ??, used: ??
G_SL4_2, ??, used: ??
G_SLSNW, ??, ??, ??, used: ??,
 ??, ??
G_SN2SL2_1, ??, used: ??
G_SN2SL2_2, ??, used: ??
G_SN2SQ2, ??, used: ??
G_SN4, ??, used: ??
G_SS, ??, ??, ??, used: ??, ??,
 ??
G_strong, ??, ??, ??, ??, ??,
 used: ??, ??
G_SU2SD2, ??, used: ??
G_SU4, ??, used: ??
G_SU4_2, ??, used: ??
G_SUSDSNSL, ??, used: ??
G_SWS, ??, used: ??
G-S-Sqrt, ??, used: ??
G_weak, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, used:
 ??, ??, ??, ??
G_WH3W, ??, ??, used: ??, ??
G_WH4, ??, ??, used: ??, ??
G_WHWAAH, ??, ??, used: ??,
 ??
G_WHWAZ, ??, ??, used: ??,
 ??
G_WHWAZH, ??, ??, used: ??,
 ??
G_WHWHAAH, ??, ??, used:
 ??, ??
G_WHWHAZH, ??, ??, used:
 ??, ??
G_WHWHWW, ??, ??, used:
 ??, ??
G_WHWHZAH, ??, ??, used:
 ??, ??
G_WHWHZHAAH, ??, ??, used:
 ??, ??
G_WHWHZZH, ??, ??, used:
 ??, ??
G_WHWWWW, ??, ??, used: ??,
 ??

- G_WHWZAH*, ??, ??, used: ??,
 ??
G_WHWZHAH, ??, ??, used:
 ??, ??
G_WHWZHZH, ??, ??, used:
 ??, ??
G_WHWZZ, ??, ??, used: ??, ??
G_WHWZZH, ??, ??, used: ??,
 ??
G_WPSLSN, ??, ??, ??, used:
 ??, ??, ??
G_WPSUSD, ??, ??, ??, used:
 ??, ??, ??
G_WSQ, ??, ??, used: ??, ??
G_WWAHAH, ??, ??, used: ??,
 ??
G_WWAZH, ??, ??, used: ??,
 ??
G_WWSFSF, ??, ??, ??, used:
 ??, ??, ??, ??, ??
g_www, ??, used: ??
G_www, ??, used: ??, ??
G_WWWWW, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, used: ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??
g_wwz, ??, used: ??
G_wwz, ??, used: ??, ??
g_wwza, ??, used: ??
G_wwza, ??, used: ??, ??
G_WWZAH, ??, ??, used: ??,
 ??
G_WWZHAH, ??, ??, used: ??,
 ??
g_wwzz, ??, used: ??
G_wwzz, ??, used: ??, ??
G_WWZZH, ??, ??, used: ??,
 ??
G_WZSLSN, ??, ??, ??, used:
 ??, ??, ??
G_WZSUSD, ??, ??, ??, used:
 ??, ??, ??
g_w-lep, ??, used: ??
G_w-lep, ??, used: ??, ??
g_w-quark, ??, used: ??
G_w-quark, ??, used: ??, ??, ??
G_YUK, ??, used: ??, ??
G_YUK_1, ??, used: ??
G_YUK_2, ??, used: ??
G_YUK_3, ??, used: ??
G_YUK_4, ??, used: ??
G_YUK_C, ??, ??, ??, used: ??,
 ??, ??
G_YUK_DCU, ??, ??, used: ??,
 ??
G_YUK_FFP, ??, ??, used: ??,
 ??
G_YUK_FFS, ??, ??, used: ??,
 ??
G_YUK_G, ??, ??, ??, used: ??,
 ??, ??
G_YUK_LCN, ??, ??, used: ??,
 ??
G_YUK_LQ_P, ??, used: ??
G_YUK_LQ_S, ??, used: ??
G_YUK_N, ??, ??, ??, used:
 ??, ??, ??
G_YUK_Q, ??, ??, ??, used: ??,
 ??, ??
G_YUK_UCD, ??, ??, used: ??,
 ??
G_Z, ??, ??, ??, used:
G_ZEH, ??, used: ??
G_ZGLQLQ, ??, used: ??
G_ZHEH, ??, used: ??
G_ZHPSipp, ??, ??, used: ??, ??
G_ZHTHT, ??, ??, used: ??, ??
G_zhthth, ??, used:
G_ZLQ, ??, used: ??
G_ZPLQLQ, ??, used: ??
G_ZPSFSF, ??, ??, ??, used:
 ??, ??, ??
G_ZPsip, ??, ??, used: ??, ??
G_ZPsipp, ??, ??, used: ??, ??
G_ZSF, ??, ??, ??, used: ??, ??,
 ??
G_ZTHT, ??, ??, used: ??, ??
G_ZZLQLQ, ??, used: ??
G_ZZSFSF, ??, ??, ??, used:
 ??, ??, ??, ??, ??
G_ZZWW, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, used: ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??
g_z-lep, ??, used: ??
G_z-lep, ??, used: ??, ??
g_z-quark, ??, used: ??
G_z-quark, ??, used: ??, ??

581

- ??, ??, ??, ??, ??, ??, ??, ??
??
- higgs_anom*, ??, ??, ??, ??, ??, ??, ??, ??, used: ??, ??, ??, ??, ??, ??
- higgs_charg_neutr*, ??, ??, ??, used: ??, ??, ??
- higgs_ch_gravitino*, ??, used: ??
- higgs_gold*, ??, used: ??
- higgs_gold4*, ??, used: ??
- higgs_gold_sf*ermion, ??, used: ??
- higgs_gold_sf*ermion', ??, used: ??
- higgs_gold_sneutrino*, ??, used: ??
- higgs_gold_sneutrino'*, ??, used: ??
- higgs_gold_squark*, ??, used: ??
- higgs_gold_squark'*, ??, used: ??
- higgs_neutr*, ??, ??, ??, used: ??, ??, ??
- higgs_neutr'*, ??, used: ??
- higgs_neutr''*, ??, used: ??
- higgs_neu_gravitino*, ??, used: ??
- higgs_SCC*, ??, ??, used: ??, ??
- higgs_sf*ermion, ??, ??, ??, used: ??, ??, ??
- higgs_sf*ermion', ??, ??, ??, used: ??, ??, ??
- higgs_sf*ermion'', ??, ??, ??, used: ??, ??, ??
- higgs_sf*ermion4, ??, used: ??
- higgs_sf*ermion4', ??, used: ??
- higgs_sf*ermion-P, ??, ??, used: ??, ??
- higgs_sf*ermion-S, ??, ??, used: ??, ??
- higgs_sneutrino*, ??, ??, ??, used: ??, ??, ??
- higgs_sneutrino'*, ??, ??, ??, used: ??, ??, ??
- higgs_sneutrino''*, ??, ??, ??, used: ??, ??, ??
- higgs_sneutrino4*, ??, used: ??
- higgs_sneutrino4'*, ??, used: ??
- higgs_SPP*, ??, ??, used: ??, ??
- higgs_squark*, ??, ??, ??, used: ??, ??, ??
- higgs_squark'*, ??, ??, ??, used: ??, ??, ??
- higgs_squark4*, ??, used: ??
- higgs_squark4'*, ??, used: ??
- higgs_squark-a*, ??, ??, ??, used: ??, ??, ??
- higgs_squark-b*, ??, ??, ??, used: ??, ??, ??
- higgs_SSS*, ??, ??, used: ??, ??
- higgs_triangle*, ??, ??, ??, ??, ??, ??, ??, ??, used: ??, ??
- higgs_triangle-vertices*, ??, ??, used: ??, ??
- Hm*, ??, ??, used: ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??
- Hn* (module), ??, used: ??, ??, ??, ??
- homogeneous*, **300**, **297**, used: **300**, **129**, **130**, **130**
- Hp*, ??, ??, used: ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??
- hs_of_flavor*, **129**, used: **129**
- hs_of_flavors*, **129**, used: **129**, **130**
- hs_of_lorentz*, **129**, used: **129**, **129**
- H_Heavy*, ??, used: ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??
- H_Light*, ??, used: ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??
- I*, **170**, ??, **170**, ??, ??, used: **170**, ??, ??, ??, **172**
- I* (camlyacc token), **173**, used: **173**
- id*, **129**, used: **129**
- IG-s*, ??, used: ??

im (field), **355**, used: **355**, **355**,
 356, **356**, **356**, **356**, **357**,
 357, **357**, **357**, **357**, **358**

imag, **355**, **355**, **358**, **170**, **354**,
 170, used: **358**, ??, **173**

Imag, **358**, used:

IMap (module), **31**, used: **31**, **31**,
 31, **31**, **32**, **32**, **32**

import, **73**, used: **74**

import_pfxixed, **267**, used: **267**,
 267

impossible, **338**, **11**, used: **338**,
 11, **11**

Impossible (exn), **338**, **11**, **25**,
 44, **50**, **104**, **105**, **122**,
 125, used:

in1 (field), **263**, used: **266**

in2 (field), **263**, used: **266**

include_anomalous, ??, ??, ??,
 ??, ??, ??, used:

include_ckm, ??, ??, ??, ??, ??,
 ??, ??, ??, used: ??, ??

include_four, ??, ??, ??, ??, ??,
 ??, ??, used: ??

include_gluons, ??, ??, ??, ??,
 ??, ??, used: ??, ??

include_goldstone, ??, ??, ??, ??,
 ??, ??, ??, used: ??, ??,
 ??, ??

include_goldstones, **272**, used:
 272, **275**

include_hf, ??, ??, ??, ??, ??,
 ??, ??, ??, used: ??, ??,
 ??

incoming, **54**, **59**, **63**, **103**, **119**,
 53, **100**, used: **63**, **130**,
 130, **198**, **217**, **217**, **217**

incoming (field), **117**, used: **119**,
 120, **121**

incomplete, **133**, **134**, **158**, **159**,
 159, used: **147**, **148**, **149**,
 149, **150**, **150**, **153**, **158**,
 158, **158**, **158**, **161**, **161**,
 162, **162**, **162**

Incomplete (exn), **256**, **258**, **255**,
 used:

Index, **172**, used: **172**

Index', **172**, used: **172**

index_string, ??, ??, used: **287**,
 287

indices, **157**, **166**, used: **157**, **166**

Infinity, **345**, used: **345**

init, **181**, used: **369**, **371**, **351**,
 182, **115**

initialize_cache, **103**, **121**, **127**,
 131, **100**, **101**, used: **131**,
 275

inject, ??, used: ??, ??, ??, ??,
 ??, ??

injectl, ??, used: ??, ??

input (field), ??, used: **181**, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, **196**, **280**, ??, ??

input_functions, **180**, used: **180**,
 182

input_lagrangian, **180**, used: **180**,
 182

input_model, **180**, used:

input_parameters, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 used: ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??

input_particles, **180**, used: **180**,
 182

input_table, **176**, used: **180**

input_variables, **180**, used: **180**,
 182

insert, **335**, used: **335**, **335**

insert1, **364**, **366**, **367**, used:
 365, **366**, **366**, **366**, **367**

insert_inorder_signed, **336**, used:
 336

insert_in_unfinished_decays,
 260, used: **260**

insert_signed, **335**, used: **335**,
 335

inspect_partition, **22**, **23**, **26**, **28**,
 28, **35**, **36**, **21**, used:

Int (module), **30**, **124**, **21**, used:
 124, **124**

INT, ??, ??, ??, ??, used: ??,
 ??, **69**, **172**

INT (camlyacc token), **70**, **173**,
 used: **71**, **173**

integer, **170**, **170**, used: ??, **178**,
 181, **173**

integer (type), **30**, **30**, **21**, used:
 30, **31**, **31**, **22**, **22**, **22**, **22**

- integer* (camllex regexp), **172**,
 used: **172**
Integer, **170, 170**, used: **170**
Integer (sig), **29, 21**, used: **30**,
30, 21, 22
internal_edges, **348**, used: **349**
internal_edges_from, **348**, used:
348, 348
int_edges (field), **349**, used: **349**,
350, 351
int_incidence, **350**, used: **351**
int_list_to_string, **69**, used:
int_node, **346**, used: **346**
int_nodes (field), **349**, used: **350**,
350, 351, 351, 352, 352
int_of_char, **??, ??, ??**, used:
??, ??, ??, ??, ??, ??, ??, ??
??, ??, ??, ??, ??, ??
int_of_csign, **??**, used: **??, ??**
int_of_gen, **??**, used: **??**
inv, **355, 356, 354**, used:
Invalid (exn), **289, 289**, used:
invert_array, **349**, used: **349**
invert_array_unsafe, **349**, used:
in_ghost_flags, **80, 81, 77**, used:
in_to_lists, **80, 81, 77**, used: **219**
isospin (type), **??**, used: **??, ??**
Iso_down, **??**, used: **??, ??, ??**
Iso_up, **??**, used: **??, ??**
is_empty, **306, 307, 312, 318**,
318, 305, 316, used: **364**,
365, 367, 47, 48
is_gauss, **72, 75, 103, 119, 72**,
100, used: **121, 212**
is_gauss (field), **74, 117**, used:
75, 75, 76, 119, 120, 121
is_goldstone_of, **108**, used:
is_node, **42, 47, 39**, used: **116**,
117, 117
is_null, **362, 363, 363, 365**,
367, 359, 360, used: **366**,
366, 367, 367, 367, 368
is_offspring, **42, 47, 39**, used: **48**
is_source, **108**, used: **113, 117**,
117
is_sterile, **42, 47, 39**, used: **48**,
49
is_unit, **362, 363, 363, 363**,
364, 365, 359, 360, 360,
 used: **366, 367, 368**
is_white, **162**, used: **162**
It (module), **133, 133**, used: **157**,
272
iter, **306, 311, 314, 318, 319**,
321, 322, 10, 12, 13, 15,
17, 42, 45, 46, 47, 50,
305, 316, 317, 7, 40,
 used: **311, 314, 319, 320**,
322, 323, 347, 347, 350,
350, 351, 352, 352, 17, 46,
47, 47, 50, ??, 266, 266,
267, 267, 268, 269, 124,
272, 275, 190, 194, 194,
194, 195, 195, 196, 198,
207, 209, 209, 212, 212,
214, 215, 217, 217, 218,
220, 221, 225, 226, 226,
226, 226, 227, 227, 227
iter', **319, 322**, used: **319, 322**
iteri, **299, 298**, used: **299**
iteri2, **299, 298**, used: **218**
iter_edges, **351, 342**, used:
iter_incoming, **352, 342**, used:
iter_internal, **352, 342**, used:
iter_nodes, **42, 47, 39**, used: **124**
iter_outgoing, **352, 342**, used:
I_Alpha_WWWWW0, **??**, used: **??**
I_Alpha_WWWWW2, **??**, used: **??**
I_Alpha_ZZWW0, **??**, used: **??**
I_Alpha_ZZWW1, **??**, used: **??**
I_Alpha_ZZZZ, **??**, used: **??**
I_G1_AWW, **??, ??**, used: **??**,
??
I_G1_minus_kappa_minus_G4_AWW,
??, ??, used: **??, ??**
I_G1_minus_kappa_minus_G4_ZWW,
??, ??, used: **??, ??**
I_G1_minus_kappa_plus_G4_AWW,
??, ??, used: **??, ??**
I_G1_minus_kappa_plus_G4_ZWW,
??, ??, used: **??, ??**
I_G1_plus_kappa_minus_G4_AWW,
??, ??, used: **??, ??**
I_G1_plus_kappa_minus_G4_ZWW,
??, ??, used: **??, ??**
I_G1_plus_kappa_plus_G4_AWW,
??, ??, used: **??, ??**
I_G1_plus_kappa_plus_G4_ZWW,
??, ??, used: **??, ??**

- I_G1_ZWW*, ??, ??, used: ??,
 ??
I_Gs, ??, ??, ??, ??, ??, ??, ??,
279, ??, used: ??, ??, ??,
 ??, ??, ??, ??, ??
I_GsRt2, ??, used: ??
I_G_AHWWH, ??, ??, used:
 ??, ??
I_G_AHWWH, ??, ??, used:
 ??, ??
I_G_AHWW, ??, ??, used: ??,
 ??
I_G_CC, ??, used: ??
I_G_DH4, ??, used: ??
I_G_Psi0ppWHW, ??, ??, used:
 ??, ??
I_G_Psi0ppWHWH, ??, ??,
 used: ??, ??
I_G_Psi0ppWW, ??, ??, used:
 ??, ??
I_G_Psi0pWA, ??, ??, used: ??,
 ??
I_G_Psi0pWAH, ??, ??, used:
 ??, ??
I_G_Psi0pWHA, ??, ??, used:
 ??, ??
I_G_Psi0pWHAH, ??, ??, used:
 ??, ??
I_G_Psi0pWHZ, ??, ??, used:
 ??, ??
I_G_Psi0pWHZH, ??, ??, used:
 ??, ??
I_G_Psi0pWZ, ??, ??, used: ??,
 ??
I_G_Psi0pWZH, ??, ??, used:
 ??, ??
I_G_S, ??, ??, ??, used: ??, ??,
 ??
I_G_WWW, ??, ??, ??, ??, ??,
 used: ??, ??, ??, ??
I_G_Z1, ??, used: ??
I_G_Z2, ??, used: ??
I_G_Z3, ??, used: ??
I_G_Z4, ??, used: ??
I_G_Z5, ??, used: ??
I_G_Z6, ??, used: ??
I_G_ZHWWH, ??, ??, used:
 ??, ??
I_G_ZHWW, ??, ??, used: ??,
 ??
I_G_ZHWH, ??, ??, used: ??,
 ??
I_G_ZWW, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, used: ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??
I_G_ZWW_K1, ??, used: ??
I_G_ZWW_K2, ??, used: ??
I_G_ZWW_K3, ??, used: ??
I_kappa5_AWW, ??, ??, used:
 ??, ??
I_kappa5_ZWW, ??, ??, used:
 ??, ??
I_lambda5_AWW, ??, ??, used:
 ??, ??
I_lambda5_ZWW, ??, ??, used:
 ??, ??
I_lambda_AWW, ??, ??, used:
 ??, ??
I_lambda_ZWW, ??, ??, used:
 ??, ??
I_Q_H, ??, ??, ??, used:
I_Q_W, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, used: ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??
I_Q_W_K, ??, used: ??
I_Q_ZH, ??, used: ??
Java (module), **242**, **184**, used:
join, **307**, used: **307**, **308**, **308**,
310, **311**
join_signs, **336**, used: **336**
ket, **103**, **117**, **99**, used: **214**
key (type), **295**, **295**, **318**, **318**,
45, **45**, **50**, **294**, **316**,
 used: **295**, **318**, **318**, **318**,
321, **45**, **45**, **45**, **50**, **294**,
294, **316**, **316**, **316**, **316**,
317, **317**
Key (sig), **295**, **294**, used: **295**,
294
keys, **46**, used: **46**
keystone, **26**, used: **26**

[illegible]

List (module), **312, 306**, used:

298, 299, 299, 299, 299,
300, 301, 301, 303, 288,
296, 373, ??, ??, ??, 362,
365, 365, 366, 366, 366,
366, 367, 367, 367, 368,
289, 290, 290, 325, 325,
325, 325, 325, 325, 325,
331, 331, 331, 331, 332,
332, 332, 332, 332, 333,
333, 333, 333, 333, 334,
334, 334, 334, 334, 334,
335, 335, 335, 335, 335,
335, 336, 336, 336, 336,
336, 337, 338, 338, 343,
344, 344, 344, 344, 345,
346, 347, 347, 348, 349,
350, 350, 350, 351, 351,
351, 11, 11, 12, 12, 14, 14,
16, 17, 17, 17, 17, 17, 18,
18, 18, 18, 23, 26, 26, 28,
28, 28, 28, 31, 31, 31, 33,
33, 34, 34, 34, 34, 35, 35,
35, 35, 36, 36, 36, 48, 49,
49, 55, 56, 57, 58, 60, 61,
61, 62, 256, 256, 258, 259,
259, 260, 260, 261, 261,
81, 81, 81, 81, 81, 81, 82,
??, ??, ??, 263, 263, 264,
264, 264, 264, 264, 264,
264, 265, 265, 265, 265,
266, 266, 266, 267, 267,
267, 268, 269, 171, 171,
171, 175, 175, 176, 176,
178, 180, 181, 181, 183,
68, 69, 73, 73, 74, 74, 74,
75, 75, 76, 76, 136, 136,
136, 136, 136, 137, 138,
138, 139, 139, 139, 140,
140, 140, 141, 141, 141,
142, 142, 142, 142, 143,
143, 143, 144, 144, 144,
145, 145, 145, 146, 146,
146, 147, 147, 147, 148,
149, 149, 150, 150, 151,
151, 151, 151, 151, 151,
152, 153, 156, 156, 156,
157, 157, 157, 157, 162,
163, 165, 165, 166, 166,
166, 166, 166, 105, 109,

[illegible]

- ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??
list2, **325, 324**, used: **12, 16**,
122, 129, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??
list3, **325, 324**, used: **13, 16**, ??,
 ??, ??, ??, ??, ??, ??, ??
lists, **42, 48, 40**, used: **121**
Lists (module), **55, 54**, used: **66**,
54
ListsW (module), **66, 54**, used:
Littlest (module), ??, ??, used:
 ??, ??
littlest_gauge_higgs4, ??, ??,
 used:
Littlest_Tpar (module), ??, ??,
 used: ??
LMOM, ??, used:
load, **182**, used: **183**
Lodd, ??, used: ??
log, **355, 357, 354**, used: **357**
longest, **318, 320, 321, 322**,
316, 317, used:
longest', **320, 322**, used: **320, 322**
longest_first, **334**, used: **334**
loop_cs, ??, used: ??, ??
loop_gen, ??, used: ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??
loop_iso, ??, used: ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??
loop_kk, ??, used: ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??
loop_kk2, ??, used: ??, ??, ??,
 ??
Loop_Tags (module), **107**, used:
lorentz, ??, **177, 134, 157, 160**,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, **279**,
 ??, ??, ??, used: **177**,
134, 157, 160, 112, 125,
129, ??, ??, 192, 214, 215,
215, 216
lorentz (label), ??, ??, used: **181**
lorentz (type), ??, used: **178**,
178, 185, ??, ??, ??
lorentz_ordering, **112**, used: **112**
lower (camllex regexp), **69, 172**,
 used: **69, 172**
LPAREN, ??, ??, ??, ??, used:
 ??, ??, **69, 172**
LPAREN (camlyacc token), **70**,
173, used: **70, 173**
LQ, ??, used: ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??
LQino, ??, used: ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??
lqino_lq_gg, ??, used: ??
lqino_lq_gg', ??, used: ??
lqino_lq_neu, ??, used: ??
lqino_lq_neu', ??, used: ??
lqino_lq_neu2, ??, used: ??
lqino_lq_neu2', ??, used: ??
lq_gauge4, ??, used: ??
lq_gauge4', ??, used: ??
lq_gg_gauge2, ??, used: ??
lq_gg_gauge2', ??, used: ??
lq_neutr_Z, ??, used: ??
lq_phiggs, ??, used: ??
lq_phiggs', ??, used: ??
lq_se_su, ??, used: ??
lq_se_su', ??, used: ??
lq_shiggs, ??, used: ??
lq_shiggs', ??, used: ??
lq_snu_sd, ??, used: ??
lq_snu_sd', ??, used: ??
lu_backsubstitute, **372**, used: **372**,
373
lu_decompose, **372, 369**, used:
lu_decompose_in_place, **371**,
 used: **372, 372, 373**
lu_decompose_split, **371**, used:
372
L_CN, ??, used:
L_CNG, ??, used:
L_K1_L, ??, used: ??, ??, ??

[illegible]

M (module), **364, 365, 367,**
349, 50, 177, 133, 157,
159, 272, 278, ??, ??,
 used: **364, 364, 364, 364,**
364, 365, 365, 365, 365,
365, 365, 365, 366, 366,
366, 366, 366, 366, 366,
366, 366, 367, 367, 367,
367, 367, 367, 367, 368,
349, 349, 349, 349, 50,
177, 181, 183, 127, 128,
129, 272, 272, 273, 275,
217, 220, 282, 102, ??, ??

[illegible]

M1 (module), **45**, used: **45**, **46**,
46, **46**

[illegible]

??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??

$M2$ (module), **45**, used: **45**, **46**,
46

```
main, ??, 271, 275, ??, 271,
    used: 73, 278, 282, 282,
    ??, ??, ??, ??, ??, ??, ??,
    ??, ??, ??, ??, ??, ??, ??,
    ??, ??, ??, ??, ??, ??, ??,
    ??, ??
```

```
main (camlyacc non-terminal),
    70, used: 70
```

Majorana, **125**, ??, used: **125**,
125, ??, ??, ??, ??

Maj-Ghost, ??, used:

make, **362, 363, 359**, used: 303,
370, 371, 363, 366, 366,
123, 272, 190

Make (module), **295, 318, 50,**
257, 263, 72, 124, 272,
294, 317, 40, 255, 262,
72, 271, used: 321, 289,
 31, 45, 46, 46, 46, 50, ??,
 ??, ??, 263, 264, 266, 104,
 109, 109, 113, 114, 115,
 119, 120, 123, 123, 124,
 124, 126, 126, 127, 127,
 128, 272, 191, 278, 282,
 282, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, 101

MakeMap (module), **321, 317**,
used:

MakePoly (module), **321, 318**,
used:

Maker (sig), **104, 101, ??**, used:
263, 127, 127, 272, 184,
189, 262, 101, 101, 101,
101, 102, 271, ??, 184,
184, 184

make_external_dag, 116, used:
116

Make_Fortran (module), [189](#),
used: [228](#), [242](#)

Massive, **??**, **??**, used:

??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??

massless, *??*, *??*, used:

Massless, `??`, `??`, used:

mass_symbol (label), ??, ??,
used: 181

matmul, **369, 369**, used:

Matter, 158, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??,
 used: ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??

```
matter_field (type), 158, ??, ??,
    ??, ??, ??, ??, ??, ??, ??,
    ??, ??, used: 158, 158,
    ??, ??, ??, ??, ??, ??, ??,
    ??, ??, ??, ??, ??, ??, ??,
    ??, ??, ??, ??, ??, ??, ??
```

max_arity, 10, 12, 13, 14, 17,
17, 19, 28, 35, 7, 9, used:
17, 18, 18, 18, 26, 35

max_degree, ??, **177**, **136**, **154**,
157, **161**, **163**, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,

max_degree (label), ??, ??, used:

max_lines, **133, 272, 133**, used:
134, 136, 156, 157, 159

Maybe-Graded (module), [46](#),
used: [49](#), [50](#)

md5sum, **189**, used: **190**, **222**,
226

merge, **308, 262, 265, 262**, used:
308, 309, 309, 309, 309,
275

mgm, ??, ??, ??, ??, ??, ??, ??,
 ??, ??, used: ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??

$min,$ **30**, used: **299**, **174**

MINUS, ??, ??, used: ??, 172

*minus-gauge*₄, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, used: ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??

- min_max_rank*, **45, 46, 47, 49, 50, 41**, used: **47, 115**
mismatch, **56, 61**, used: **56, 56, 57, 57, 57, 58, 58, 58, 59, 62, 62, 62, 62, 63, 63, 64**
Mismatch (exn), **295, 296, 54, 56, 61, 127, 127, 294, 52, 101**, used:
Mismatched_arity (exn), **11, 12, 13, 16, 17, 9**, used:
Mixed23 (module), **14, 28, 127, 9, 21, 101**, used: **28, 127, 282, 282, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, 21**
Mixed23 (sig), **14, 9**, used: **9**
Mixed23_Majorana (module), **127, 101**, used: **??, ??, ??, ??, ??, ??, ??, ??**
mixed_fold_left, **116**, used: **116**
mk_and, **68, 67**, used: **??, 74, 70**
mk_any_flavor, **68, 67**, used: **??, 70**
mk_false, **68, 67**, used:
mk_gauss, **68, 67**, used: **??, 70**
mk_gauss_not, **68, 67**, used: **??, 70**
mk_off_shell, **68, 67**, used: **??, 70**
mk_off_shell_not, **68, 67**, used: **??, 70**
mk_on_shell, **68, 67**, used: **??, 70**
mk_on_shell_not, **68, 67**, used: **??, 70**
mk_or, **68, 67**, used: **??, 70**
mk_true, **68, 67**, used: **??, 70**
Model (module), **177, 174, ??**, used: **263, 72, 133, 157, 159, 104, 104, 106, 107, 125, 127, 127, 272, ??, 184, 189, 278, ??, 262, 174, 72, 133, 133, 133, 101, 102, 103, 271, ??, ??, ??, ??, ??, ??, ??, ??**
Modellib_BSM (module), **??, ??**, used: **??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??**
Modellib_MSSM (module), **??, ??**, used: **??, ??, ??, ??**
Modellib_NMSSM (module), **??, ??**, used: **??, ??**
Modellib_PSSSM (module), **??, ??**, used: **??**
Modellib_SM (module), **??, ??**, used: **278, 282, ??, ??, ??, ??**
Modeltools (module), **??, ??**, used: **177, 181, 137, 161, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, 280, ??, ??**
modname, **??**, used: **??, ??, ??, ??**
module_name, **189**, used: **190, 226, 227**
mod_float2, **291**, used: **292**
mom, **??, ??, ??, ??, ??**, used: **??, ??, ??, ??, ??**
MOM, **??**, used:
MOM5, **??**, used:
Momenta, **184, 197, ??**, used: **275, 198**
momentum, **103, 108, 191, 98**, used: **263, 113, 114, 121, 123, 123, 124, 278, 202, 207, 212, 212, 213, 215, 215, 217, 217**
momentum (field), **108**, used: **108, 108, 111, 113, 113, 119, 122**
momentum (type), **256, 257, 255**, used: **256, 258, 258, 260, 255, 255, 255**
momentum (camlyacc non-terminal), **71**, used: **71**
Momentum, **172**, used: **172**
Momentum (module), **54, 51**, used: **257, 263, 72, 104, 106, 107, 127, 127, 271, 271, 278, 184, 189, 255, 262, 72, 101, 102, 103, 271, ??**
momentum_list, **103, 108, 98**, used: **124, 272, 191, 191, 199, 213, 214**

momentum_list (camlyacc
non-terminal), **71**, used:
70, 71

MOML, **??**, used:

MOMR, **??**, used:

Mono (sig), **10, 7**, used: **11, 13**,
9, 9

MSSM (module), **??, ??**, used:
??, ??, ??

MSSM_flags (sig), **??, ??**, used:
??, ??, ??, ??, ??, ??, ??

MSSM_goldstone (module), **??**,
??, used:

MSSM_Grav (module), **??, ??**,
used: **??**

MSSM_no_4 (module), **??, ??**,
used: **??**

MSSM_no_4_ckm (module), **??**,
??, used: **??**

MSSM_no_goldstone (module),
??, ??, used:

Mu, **??, ??**, used:

mul, **362, 363, 363, 363, 365**,
366, 355, 356, 359, 360,
360, 354, used: **365, 366**,
366, 366, 366, 366, 367

mul1, **365**, used: **365, 365, 365**

MULT, **??, ??**, used: **??, 172**

MULT (camlyacc token), **173**,
used: **173, 173**

multinomial, **331, 327**, used:

multiple_variable, **191**, used:
202, 207, 212, 214

multiple_variables, **191**, used:
191

multiplicity, **31**, used: **32, 32, 33**

multiply, **301, 171, 298, 170**,
used: **171, ??, 173**

multi_choose, **335, 329**, used:

multi_split, **333, 328**, used: **333**,
335

multi_split', **332**, used: **332, 333**,
333

mult_vertex3, **137**, used: **140**,
141, 141

mult_vertex4, **138**, used: **147**,
148, 149, 149, 150, 150,
151, 151, 151, 151, 151,
151

Muon, **??**, used: **??, ??, ??, ??**

Mutable (module), ??, ??, used:
177

Mutable (sig), ??, used: ??

M_N, ??, used:

M_SF, ??, used:

M_U, ??, used:

M_V, ??, used:

n (field), 260, used:

N, ??, ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??,
 used: ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,

N1, ??, ??, ??, used: ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??

N10, ??, used: ??, ??

N11, ??, used: ??, ??

N2, ??, ??, ??, used: ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??

N3, ??, ??, ??, used: ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??

N4, ??, ??, ??, used: ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??

N5, ??, ??, used: ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??

N6, ??, used: ??, ??

N7, ??, used: ??, ??

N8, ??, used: ??, ??

N9, ??, used: ??, ??

neutral_up_type_currents, ??,
used: ??
neutral_Z, ??, ??, used: ??, ??
neutral_Z_1, ??, used: ??
neutral_Z_2, ??, used: ??
neutr_lqino_current, ??, used:
??
NH, ??, used: ??, ??, ??, ??, ??
nl, **244, 190**, used: **245, 191, 194,**
194, 194, 195, 195, 195,
195, 195, 195, 196, 198,
198, 212, 212, 213, 214,
215, 216, 217, 217, 217,
218, 218, 219, 219, 219,
220, 220, 220, 220, 221,
221, 222, 222, 222, 222,
223, 223, 223, 223, 224,
225, 226, 226, 226, 226,
227, 227, 227, 227
nlist, ??, used: ??, ??
NMSSM (module), ??, ??, used:
??
NMSSM_CKM (module), ??, ??,
used: ??
NMSSM_flags (sig), ??, ??,
used: ??, ??, ??, ??
NMSSM_func (module), ??, ??,
used: ??, ??
Nodd, ??, used: ??
node, **343, 341**, used:
node (field), **348**, used: **348**
node (type), **42, 42, 43, 44, 45,**
46, 114, 38, 39, 41,
used: **42, 42, 43, 43, 44,**
46, 49, 38, 38, 38, 39, 39,
39, 39, 39, 39, 39, 40, 40,
40, 40, 40, 40, 40, 40,
41, 41, 41, 42
Node, **306, ??, 342**, used: **307,**
307, 307, 309, 310, 311,
311, 312, ??, 342, 343,
343, 344, 344, 345
nodes, **343, 341**, used: **343, 348**
Nodes (module), **42, 43, 44, 45,**
114, 38, 41, used: **42, 43,**
43, 45, 46, 46, 47, 47, 48,
49, 114, 38, 41, 41, 42
node_to_string, **344**, used: **345**
node_with_tension (type), **348**,
used:

[illegible]

??, ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??
O (module), **278, 282, 282**, ??,
 ??, ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??, ??,
 ??, used: **278, 282, 282**,
 ??, ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??, ??,
 ??, ??
Ocaml (module), **242, 184**, used:
of2, **11, 12, 14, 15, 16, 17, 9, 9**,
9, used: **12, 12, 16, 16, 18**,
25, 28, 36
of2_kludge, **10, 12, 14, 16, 18**,
8, used: **117**
of3, **13, 13, 14, 15, 16, 17, 9, 9**,
9, used: **14, 16, 28, 28**
Off, **197**, used: **197**
OFFSHELL, ??, ??, used: ??, **69**
OFFSHELL (camlyacc token),
70, used: **70**
Offspring (module), **46**, used: **47**,
47, 47, 47, 47, 47, 47, 48,
48, 48, 48, 48, 49
Off-shell, **68, 73, 67**, used: **68**,
73
Off-shell_not, **68, 73, 67**, used:
68, 73
of_float, **355, 357, 354**, used:
of_float2, **355, 357, 354**, used:
of_int, **29, 30, 355, 357, 21**,
354, used: **30, 34, 35**
of_int2, **355, 357, 354**, used:
of_ints, **54, 55, 61, 51**, used: **59**,
66, 260, 261, 261, 73, 111
of_list, **16, 17, 80, 81, 9, 77**,
 used: **303, 349, 350, 351**,
18, 28, 35, 157, 166
of_momenta, **256, 261, 255**,
 used:
of_momentum, **65, 65, 66, 54**,
 used: **66, 264, 265, 266**
of_string, **355, 358, 73, 354**,
 used: **74**

- of_string_list*, **72, 74, 71**, used:
275
of_subarray, **299, 297**, used:
of_vertices, **??, ??, ??**, used:
181, 154, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
280, ??, ??
Omega (module), **271, 271**, used:
278, 282, 282, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??
Omega_GravTest (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_NMSSM (module), **??**,
used:
Omega_NMSSM_CKM (module),
??, used:
Omega_PSSM (module), **??**,
used:
Omega_QCD (module), **278**,
used:
Omega_QED (module), **278**,
used:
Omega_Simplest (module), **??**,
used:
Omega_Simplest_univ (module),
??, used:
Omega_SM (module), **282**, used:

Omega_SM_ac (module), **??**,
used:
Omega_SM_ac_CKM (module),
??, used:
Omega_SM_CKM (module), **??**,
used:

Omega_SM_km (module), **??**,
used:
Omega_SM_top (module), **??**,
used:
Omega_Template (module), **??**,
used:
Omega_Threshl (module), **??**,
used:
Omega_Threshl_nohf (module),
??, used:
Omega_UED (module), **??**, used:

Omega_Xdim (module), **??**, used:

Omega_Zprime (module), **??**,
used:
one, **365, 29, 30, 355, 355, 358,**
21, 354, used: **365, 366,**
31, 32, 34, 34, 356
one_compatible, **73**, used: **73, 75**
Only_Insertion, **??**, used: **??, ??,**
??, ??, ??, ??, ??, ??, ??,
??, ??, ??
ONSHELL, **??, ??**, used: **??, 69**
ONSHELL (camlyacc token), **70**,
used: **70**
on_shell, **72, 75, 103, 119, 72,**
100, used: **121, 212**
on_shell (field), **74, 117**, used:
75, 75, 76, 119, 120, 121
On_shell, **68, 73, 67**, used: **68,**
73
On_shell_not, **68, 73, 67**, used:
68, 73
open_file, **292, 291**, used:
Open_File, **291**, used: **292**
open_in_bin_first, **296**, used:
296
open_out_bin_last, **296**, used:
296
options, **??, 183, 134, 157, 160,**
103, 107, 127, 128, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, 184, 190,
279, ??, ??, 98, 101, ??,
??, used: **134, 157, 160,**
128, 275, ??, ??
Options (module), **289, 289**,
used: **??, 183, 103, 107,**

- 127, 128, 275, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 184, 190, 279, ??, ??, 98,
 101, ??, ??
- Or*, 68, 67, used: 68
- OR*, ??, ??, used: ??, 69
- OR* (camlyacc token), 70, used:
 70, 70
- Ord* (sig), 42, 37, used: 42, 43,
 43, 44, 44, 44, 44, 38, 38,
 41, 41
- ordered_multi_choose*, 335, 329,
 used:
- ordered_multi_split*, 333, 328,
 used: 335
- ordered_partitions*, 333, 328,
 used:
- ordered_partitions'*, 333, used:
 333, 333
- ordered_split*, 332, 328, used:
 332, 333
- ordered_split_unsafe*, 331, used:
 332, 332
- order_flavor*, 112, used: 113
- order_spin_table*, 129, used: 129
- order_spin_table1*, 128, used:
 128, 129
- order_wf*, 113, used: 113, 114,
 114, 115, 123, 123
- other*, 158, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 used: ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??
- other* (type), 158, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 used: 158, 158, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??
- Other*, 158, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 used: ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??
- out* (field), 263, used: 266
- outer*, 326, 324, used:
- outer_self*, 326, 324, used:
- outgoing*, 54, 59, 63, 103, 119,
 53, 100, used: 130, 130,
 198, 217, 217, 217
- outgoing* (field), 117, used: 119,
 120, 121
- out_ghost_flags*, 80, 81, 77,
 used:
- out_to_lists*, 80, 81, 77, used:
 219
- p* (field), 55, used: 56, 56, 57, 57,
 57, 58, 58, 58
- p* (type), 72, 72, 103, 108, 128,
 71, 98, used: 72, 72, 74,
 103, 104, 106, 108, 128,
 71, 72, 72, 72, 72, 98, 101,
 103
- P*, ??, used: ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??
- P* (module), 271, used: 272
- P1*, ??, ??, used: ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??
- P2*, ??, ??, ??, used: ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??
- p2s*, 123, 272, 190, used: 124,
 272, 191, 191
- P3*, 28, ??, used: 28, ??, ??, ??
- P4*, 28, ??, used: 28, ??, ??, ??
- P5*, ??, used: ??, ??, ??
- P6*, ??, used: ??, ??, ??
- P7*, ??, used: ??, ??, ??
- pack_map*, 264, used: 265
- pack_tree*, 264, used: 264
- pair*, 129, used: 129
- pairs*, 337, ??, ??, 337, used: 12,
 16, 23, 36, ??, ??, ??, ??,
 ??, ??, ??, ??
- pairs'*, 337, used: 337, 337
- pair_or_triple* (type), 14, 9,
 used: 14
- panic*, 197, used: 197
- Panic*, 197, used: 198
- parameters*, ??, 177, 137, 157,
 161, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 280, ??, ??, ??, used:
 177, 137, 157, 161, ??, ??,
 228

- parameters* (label), ??, ??, used: 181
- parameters* (type), 193, ??, used: ??, ??
- parameters_to_channel*, 184, 228, ??, used: 275
- parameters_to_fortran*, 196, used: 228
- parameter_module*, 189, used: 190, 196, 226
- Parents* (module), 46, used: 47, 47, 47, 47, 47, 47, 47, 47, 47, 47, 48, 48, 48, 48, 48, 48, 48, 49, 49
- parse*, 288, 289, 286, 289, used: 337, 10, 22, 42, 54, ??, 262, 174, 103, 275, ??, ??, ??, ??, ??, 244, 184, 278, ??, ??
- parse1*, 287, used: 288
- parse_color*, 180, used: 180
- parse_decay*, 274, used: 275
- parse_diagnostic*, 198, used: 198, 198
- parse_diagnostics*, 198, used: 227
- parse_error*, ??, 70, used: ??, ??
- parse_expr*, 179, used: 179, 179, 180
- parse_function_row*, 179, used: 180
- parse_lagragian_row*, 179, used: 180
- parse_particle_row*, 180, used: 180
- parse_process*, 273, used: 274, 274
- parse_scattering*, 274, used: 275
- parse_source*, 287, used: 288
- parse_spin*, 180, used: 180
- parse_symbol*, 180, used: 180
- parse_table*, 180, used: 180
- parse_variable_row*, 180, used: 180
- partial*, 364, 367, 361, used:
- particle* (type), 178, used:
- particles_file*, 182, used: 182, 183
- particle_flavor* (type), 178, used:
- partition*, 26, 35, used: 26, 35, 130, 130
- partition* (type), 22, 23, 26, 28, 28, 35, 36, 20, used: 22, 20, 21
- Partition* (module), 337, 337, used: 12, 14, 16, 18, 23, 26, 35, 36
- partitions*, 333, 22, 24, 26, 28, 28, 35, 36, 328, 20, used: 334, 26, 28, 28, 28, 35, 35, 36
- partitions'*, 333, 23, 26, 35, used: 333, 333, 334, 23, 24, 26, 26, 35, 35
- PBP*, ??, used: 137, ??, ??, ??, ??, ??, ??, ??, ??
- pdg*, ??, 177, 134, 157, 160, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, 282, ??, ??, ??, used: 177, 134, 157, 160, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, 220, 282, 282, ??, ??, ??, ??
- pdg* (label), ??, ??, used: 181
- pdg_heuristic*, 176, used: 179, 179
- pdg_mw*, ??, ??, ??, ??, used: ??, ??, ??, ??, ??, ??, ??, ??
- permutation*, 105, 126, used: 106, 126
- permutation_symmetry*, 34, used: 34
- permute*, 335, 329, used: 336, 136, 136, 156, 165
- permute3*, ??, used: ??
- permute4*, ??, used: ??
- permute_even*, 335, 329, used:
- permute_odd*, 335, 329, used:
- permute_quadruple*, 136, used: 152
- permute_signed*, 335, 329, used: 335, 335, 336
- permute_signed'*, 335, used: 335, 335
- permute_tensor*, 336, 329, used:
- permute_tensor_even*, 336, 329, used:

used:

permute_tensor_signed, **336**, **329**, used: **336**, **336**

permute_triple, **136**, used: **141**,
147, **148**, **149**, **149**, **150**,
150, **152**

Phasespace (module), **256**, **255**,
used:

Phi, ??, ??, used: ??, ??, ??, ??,
??, ??, ??

[illegible]

Phi3 (module), ??, ??, used:

Phi4 (module), ??, ??, used:

phiggs (type), ??, ??, used: ??,
??, ??, ??

PHiggs, ??, ??, used: ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??

phiggs_neutr, ??, ??, used: ??,
??

[illegible]

Phino, ??, used: ??, ??, ??, ??

Phip, ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, used:
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??.

??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??

Photon, ??, used: ??, ??, ??, ??

Pi , ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, used:
 ??, ??

pivot_column, **371**, used: **371**

plist, ??, used: ??

PLUS, ??, ??, ??, ??, used: ??,
??, 69, 172

PLUS (camlyacc token), **70, 173**,
used: **70, 71, 173, 173**

PM (module), **362**, used: **364**,
365, 365, 367, 367, 368

Pmap (module), **306, 305**, used:
303, 321, 362, 349, 318

PMap (module), **303**, used: 303
poles, **103, 123, 100**, used: 264.

123

Poles (module), **263**, used: **263**,
263, **264**, **265**, **266**

poles', **123**, used: **123**

poles_beneath, **122**, used: **123**

poles_to_whizard, **263**, used: **264**

Pole_Map (module), **264**, used:
264, 264, 264

Poly (sig), **321, 11, 317, 9**, used:
321, 11, 14, 16, 44, 106,
106, 107, 107, 318, 9, 9, 9,
38, 103

Pos, ??, used: ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??

Positron, ??, used: ??, ??, ??, ??

POT, ??, used: ??

Pow, ??, used: ??

power, **363, 364, 365, 325, 10,**
12, 13, 16, 18, 171, 360,
324, 8, 170, used: 365,
366, 366, 366, **??, 173**

POWER, ??, ??, used: ??, 172

POWER (camlyacc token), 173,
used: 173, 173

power_fold, 10, 12, 13, 16, 18,
8, used: 16, 109

- pred*, **29, 30, 21**, used: 298, 298,
298, 299, 301, 301, 303,
369, 370, 371, 371, 372,
365, 330, 330, 330, 331,
332, 332, 334, 337, 338,
350, 352, 12, 14, 18, 18,
18, 26, 30, 32, 33, 33, 35,
35, 35, 36, 61, 61, 64, 65,
174, 175, 175, ??, ??
- prepend_tofst_unsorted*, **332**,
used: 333
- print*, **245, 244**, used: 227
- print_add_dscalar2_vector2*,
202, used: 209
- print_add_dscalar4*, **201**, used:
209
- print_add_vector4*, **200**, used:
207
- print_add_vector4_km*, **201**,
used: 207
- print_amplitudes*, **218**, used: 227
- print_amplitude_table*, **222**, used:
227
- print_argument_diagnostics*,
216, used: 218
- print_braket*, **214**, used: 215
- print_brackets*, **215**, used: 218
- print_color_flows*, **223**, used: 227
- print_color_flows_table*, **220**,
used: 221
- print_color_tables*, **221**, used:
227
- print_current*, **185, 188, 202**,
231, used: 202, 212, 212,
214
- print_current_b*, **185, 188, 232**,
used: 202
- print_current_g*, **185, 189, 234**,
used: 202
- print_current_g4*, **185, 189**,
241, used: 207
- print_current_p*, **185, 188, 231**,
used: 202
- print_declarations2*, **199**, used:
218
- print_description*, **225**, used: 227
- print_dispatch_functions*, **224**,
used: 227
- print_dscalar2_vector2*, **201**,
used: 202, 209
- print_dscalar4*, **201**, used: 201,
209
- print_echo*, **195**, used: 196
- print_echo_array*, **195**, used: 196
- print externals*, **217**, used: 218
- print_external_momenta*, **217**,
used: 218
- print_fermion_2_g4_current*,
238, 238, used: 241
- print_fermion_2_g4_current_rev*,
240, used: 241
- print_fermion_2_g4_vector_current*,
239, used: 241
- print_fermion_2_g4_vector_current_rev*,
240, used: 241
- print_fermion_current*, **187**,
230, used: 188, 231, 231,
232
- print_fermion_current2*, **188**,
230, used: 188, 231, 231,
232
- print_fermion_current2_chiral*,
231, used: 231
- print_fermion_current2_vector*,
230, used: 231
- print_fermion_current_chiral*,
231, used: 231
- print_fermion_current_mom*,
232, used: 234
- print_fermion_current_mom_chiral*,
233, used: 234
- print_fermion_current_mom_sign*,
233, used: 234
- print_fermion_current_mom_sign_1*,
233, used: 234
- print_fermion_current_vector*,
230, used: 231
- print_fermion_g4_brs_vector_current*,
235, used: 241
- print_fermion_g4_current*, **238**,
used: 241
- print_fermion_g4_current_rev*,
239, used: 241
- print_fermion_g4_svlr_current*,
236, used: 241
- print_fermion_g4_vector_current*,
239, used: 241
- print_fermion_g4_vector_current_rev*,
240, used: 241

- print_fermion_g_2_current*, **234**,
used: **234**
- print_fermion_g_2_current_rev*,
234, used: **234**
- print_fermion_g_current*, **233**,
used: **234**
- print_fermion_g_current_rev*,
234, used: **234**
- print_fermion_g_current_vector*,
234, used: **234**
- print_fermion_g_current_vector_rev*,
234, used: **234**
- print_fermion_s2lr_current*, **237**,
used: **241**
- print_fermion_s2p_current*, **237**,
used: **241**
- print_fermion_s2_current*, **236**,
used: **241**
- print_flavor_table*, **220**, used:
220
- print_flavor_tables*, **220**, used:
227
- print_fudge_factor*, **217**, used:
218
- print_fusion*, **212**, used: **214**
- print_fusions*, **214**, used: **218**
- print_fusion_diagnostics*, **212**,
used: **214**
- print_gauss*, **211**, used: **212**
- print_ghost_flags_table*, **221**,
used: **221**
- print_helicity_selection_table*,
222, used: **227**
- print_incoming*, **215**, used: **217**
- print_inquiry_functions*, **223**,
used: **227**
- print_inquiry_function_declarations*,
223, used: **226**
- print_integer_parameter*, **219**,
used: **227**
- print_list*, **190**, used: **191**, **194**,
198, **198**, **216**, **226**
- print_logical_parameter*, **219**,
used: **227**
- print_maintenance_functions*,
222, used: **227**
- print_md5sum_functions*, **222**,
used: **227**
- print_module_footer*, **227**, used:
227
- print_module_header*, **226**, used:
227
- print_momenta*, **213**, used: **218**
- print_numeric_inquiry_functions*,
223, used: **227**
- print_outgoing*, **215**, used: **217**
- print_projector*, **211**, used: **212**
- print_propagator*, **210**, used: **212**
- print_public*, **226**, used: **226**, **226**
- print_public_interface*, **226**,
used:
print_spin_table, **219**, used: **220**
- print_spin_tables*, **220**, used: **227**
- print_vector4*, **200**, used: **200**,
207
- print_vector4_km*, **200**, used:
201, **207**
- print_version*, **226**, used: **227**
- process* (type), **273**, used:
- processes*, **127**, **128**, **102**, used:
218
- processes* (field), **128**, used: **128**
- process_sans_color_to_string*,
217, used: **225**
- process_to_string*, **128**, **217**,
used: **130**
- Prod*, **??**, used: **??**, **??**, **??**, **??**,
??, **??**
- product*, **363**, **363**, **365**, **366**, **10**,
12, **13**, **16**, **17**, **360**, **361**,
7, used: **12**, **13**
- Product*, **170**, **170**, used: **171**,
171, **171**
- Product* (module), **325**, **324**,
used: **336**, **336**, **344**, **12**,
12, **13**, **14**, **16**, **17**, **18**, **18**,
49, **265**, **111**, **112**, **122**,
122, **123**, **129**, **274**, **274**,
??, **??**, **??**, **??**, **??**, **??**, **??**,
??, **??**, **??**, **??**, **??**, **??**, **??**,
??, **??**, **??**, **??**, **??**, **??**, **??**,
??, **??**, **??**, **??**, **??**, **??**, **??**,
??, **??**, **??**, **??**, **??**, **??**, **??**,
??, **??**, **??**, **??**, **??**, **??**, **??**,
??, **??**, **??**, **??**, **??**, **??**, **??**,
??, **??**, **??**, **??**, **??**, **??**
- product_fold*, **10**, **12**, **13**, **16**, **17**,
7, used: **12**, **13**, **16**, **18**

Progress (module), **291, 291**,
used: **130, 130**
progress_mode (type), **127**, used:

progress_option, **128**, used: **128**,
130
project, ??, used: ??, ??, ??, ??,
??, ??, ??
prop (field), **31**, used: **31, 32, 32**,
32, 32
propagator, ??, **177, 135, 157**,
160, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??
??, ??, ??, ??, ??, ??, ??
279, ??, ??, ??, used:
177, 135, 157, 160, 108,
??, ??, **210, 211, 211, 212**
propagator (label), ??, ??, used:
181
propagator (type), ??, used: **178**,
??, ??
propagator_of_lorentz, **178**,
used: **179, 179**
Prop_Col_Feynman, ??, used:
134, 160
Prop_Col_Majorana, ??, used:
134, 160
Prop_Col_Scalar, ??, used: **134**,
160
Prop_Col_Unitarity, ??, used:
134, 160
Prop_ConjSpinor, ??, used: **178**,
??, ??, ??, ??, ??, ??, ??
??, ??, ??, ??, ??, ??, ??
??, ??, ??, **279**, ??, ??
Prop_Feynman, ??, used: **178**,
??, ??, ??, ??, ??, ??, ??
??, ??, ??, ??, ??, ??, ??
??, **279**, ??, ??
Prop_Gauge, ??, used: ??
Prop_Ghost, ??, used:
Prop_Majorana, ??, used: **178**,
??, ??, ??, ??
Prop_Rxi, ??, used: ??
Prop_Scalar, ??, used: **178, 178**,
??, ??, ??, ??, ??, ??, ??
??, ??, ??, ??, ??, ??, ??
??, ??, ??
prop_spinor, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??

[illegible]

[illegible]

psi-propagator, **185, 186, 229**,
 used: **210**
psi-type, **185, 186, 228**, used:
198
PSSSM (module), **??, ??**, used:
??
PSSSM-QCD (module), **??, ??**,
 used:
pullback, **134, 160**, used: **134**,
135, 135, 155, 160, 160,
161, 164
PV, **??**, used:
p-aux (field), **178**, used: **179**,
179, 180
p-color (field), **178**, used: **179**,
179, 180
p-mass (field), **178**, used: **179**,
179, 180
p-name (field), **178**, used: **179**,
179, 180
P_NNA, **??**, used:
P_NNG, **??**, used:
P_NNH1, **??**, used:
P_NNH2, **??**, used:
p-spin (field), **178**, used: **179**,
179, 180
p-symbol (field), **178**, used: **178**,
180, 181
p-to-string, **261**, used: **261, 261**
P_Whizard (module), **271**, used:
272
p-width (field), **178**, used: **179**,
179, 180
Q, **??**, used: **??, ??**
QED (module), **??, ??**, used: **278**
qgc, **??, ??, ??, ??, ??, ??, ??**,
??, used: **??, ??, ??, ??**,
??, ??, ??, ??, ??, ??, ??
QH, **??**, used: **??, ??, ??, ??, ??**,
??, ??, ??, ??
quark, **??**, used: **??, ??**
Quark, **??, ??**, used: **??, ??, ??**,
??, ??, ??, ??, ??, ??, ??,
??
quark-currents, **??**, used: **??**
quartic_anom, **??, ??, ??, ??, ??**,
??, ??, ??, used: **??, ??**
quartic-gauge, **??, ??, ??, ??, ??**,
??, ??, ??, ??, ??, ??, ??,
??, ??, used: **??, ??, ??**,

- ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??
quartic_gravitino, ??, used: ??
quartic_grav_char, ??, used: ??
quartic_grav_gauge, ??, used: ??
quartic_grav_neu, ??, used: ??
quartic_grav_sneutrino, ??, used:
 ??
Quiet, **127**, used: **128**
Quot, ??, used: ??, ??, ??, ??,
 ??, ??, ??, ??
quote, ??, ??, used: **275**
Q_charg, ??, ??, ??, used: ??,
 ??, ??
Q_down, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, used: ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??
Q_lepton, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, used: ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??
Q_top, ??, used: ??
Q_up, ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 used: ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??
Q_Z_up, ??, ??, ??, used: ??, ??
r (field), **55**, used: **55, 56, 56, 56,**
56, 56, 57, 57, 58, 59,
59, 59, 60
range, **299, 297**, used: **18, 18, 60,**
181, 73, 136, 156, 165,
120, 121, ??, 191
Range (exn), **54, 55, 61, 52**,
 used:
rank, **43, 44, 44, 44, 47, 49, 54,**
56, 61, 80, 81, 82, 114,
41, 41, 52, 77, used: **46,**
47, 62, 62, 62, 63, 64, 64,
108, 114, 124, 219
rank (type), **45, 45, 46, 49, 50,**
41, used: **45, 45, 45, 49,**
50, 41, 42
rank0, **61**, used: **61, 62, 62, 62,**
62, 63, 63
ranked, **45, 46, 47, 49, 50, 41**,
 used: **47, 115**
ranks, **45, 46, 47, 49, 50, 41**,
 used: **47**
rank_default, **82**, used: **82**
rank_set, **46**, used: **46**
Rational (sig), **362, 359**, used:
363, 363, 365, 359, 360,
361
raw (field), **289**, used: **289, 289,**
290, 290
raw (type), **287, 170, 286, 170**,
 used: **170, ??, 178, 286,**
170, 170, 170, ??, 173
rcs, **337, 10, 11, 13, 14, 17, 22,**
23, 26, 28, 28, 30, 35,
36, 42, 46, 54, 55, 60,
??, 262, 177, 157, 159,
166, 104, 104, 107, 125,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
186, 228, 278, ??, ??,
337, 8, 21, 40, 54, ??,
 used: **266, 157, 159, 166,**
110, 121, 124, 272, ??,
189, 196, 225, 226
RCS (module), **287, 286**, used:
337, 10, 10, 11, 13, 14, 17,
22, 22, 23, 26, 28, 28, 35,
36, 42, 42, 46, 54, 54, 55,
60, ??, ??, 262, 266, 174,
157, 159, 166, 103, 103,
104, 104, 107, 109, 125,
272, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, 244, 184, 185, 186,
189, 196, 225, 226, 228,
278, ??, ??, 337, 8, 21,
40, 54, 101, ??, ??
rcs_author (field), **287**, used:
287, 288
rcs_date (field), **287**, used: **287,**
288
rcs_file, **10, 22, 42, 54, ??, 174,**
103, ??, ??, ??, ??, ??,
244, 184, 278, ??, ??,

- used: 11, 13, 14, 17, 23,
26, 28, 28, 30, 35, 36, 46,
55, 60, ??, 177, 104, 107,
125, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??,
186, 189, 228, 278, ??, ??
- rcs_list*, 103, 124, 184, 189,
101, ??, used: 266, 272,
196, 226
- rcs_revision* (field), 287, used:
287, 288
- rcs_source* (field), 287, used:
287, 288
- re* (field), 355, used: 355, 355,
356, 356, 356, 356, 357,
357, 357, 357, 357, 358
- read*, 295, 296, 294, used: 296
- read_generation*, ??, used: ??
- read_lines*, 275, used:
read_lines_rev, 274, used: 275,
275
- real*, 355, 355, 358, 354, used:
358
- Real*, 358, ??, used: 358, 181, ??,
??
- realspinors* (field), 191, used:
192, 192, 198
- Real_Array*, ??, used: ??, ??
- real_arrays* (field), 193, used:
193, 193, 196
- real_singles* (field), 193, used:
193, 193, 196
- real_variable*, 178, used: 178,
179, 179, 181
- Rec*, ??, used:
- remove*, 306, 309, 314, 318,
319, 321, 322, 32, 305,
316, 317, used: 309, 314,
319, 319, 322, 322, 364,
365, 366, 367, 32, 32, 33
- rename*, 288, 286, used: 11, 13,
14, 17, 23, 26, 28, 28, 35,
36, 46, 55, 60, ??, 157,
159, 166, 104, 107, 125,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, 186, 189, 228
- require_library*, 185, 189, 189,
241, used: 189, 226
- reset*, 293, 291, used:
- rev* (field), 345, 341, used: 346,
346, 346, 348, 348
- reverse_braket*, 185, 189, 241,
used: 214
- revision*, 287, 286, used: 288
- revision* (field), 287, 286, used:
288, 337, 10, 22, 42, 54,
??, 262, 174, 103, ??, ??,
??, ??, ??, 244, 184, 278,
??, ??
- revmap*, ??, used: ??, ??
- revmap2*, ??, used: ??
- rev_multiply*, 301, used: 301, 301
- rhs*, 103, 117, 99, used: 212,
212, 213
- rhs* (type), 103, 108, 98, used:
103, 113, 117, 117, 98, 99,
99, 99, 99
- Ring* (sig), 363, 360, used: 364,
365, 367, 361, 361
- RMOM*, ??, used:
- RPAREN*, ??, ??, ??, ??, used:
??, ??, 69, 172
- RPAREN* (camlyacc token), 70,
173, used: 70, 173
- Running*, ??, used:
- R_CN*, ??, used:
- R_CNG*, ??, used:
- R_NC*, ??, used:
- R_NCH*, ??, used:
- S*, 279, ??, used: ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, 279,
279, 280, 280, ??, ??
- S* (module), ??, 107, 173, used:
??, 107, 107, 173
- S1*, ??, ??, used: ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??
- S1* (module), 46, used: 46
- S2*, ??, ??, ??, used: ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??
- S2* (module), 46, used: 46
- S2L*, ??, used:
- S2LR*, ??, used: ??, ??
- S2P*, ??, used:

scale, **363, 364, 366, 367, 171,**
360, 361, used: **366, 367**
171, 171

scattering (type), **273**, used: 273
Scattering, **273**, used: 273, 273,
 274

Sccs, ??, ??, ??, used:

schisma, **193**, used: **193**, 196

schisma_num, **193**, used: 193,
196

SD , ??, ??, ??, used: ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??

[illegible]

sdown4_1gen, ??, used: ??

sdown4_1gen', ??, used: ??

sdown4_2gen, ??, used: ??

sdown4_2gen', ??, used: ??

search_path, 294, used: 296, 296, 296

selectors (type), **72, 74, 103, 119, 127, 128, 71, 99, 101**, used: **72, 72, 72, 103, 104, 119, 127, 127, 128, 71, 71, 71, 72, 72, 72, 72, 99, 101, 101, 102**

select_p, **72, 75, 72**, used: **120**

select-p (field), **74**, used: **75**, **75**,
76

```
select_wf, 72, 75, 71, used: 120
select_wf (field), 74, used: 75,
75, 76
```

Selfconjugate, **178**, used: **180**

seq (type), **329, 327**, used: **328,**
328, 328, 328, 328, 328,
328, 329, 329, 329, 329,
329

set (type), **326, 324**, used:

[illegible]

- ??, ??, ??, ??, ??, ??, ??,
- ??, ??, ??, ??, ??, ??, ??,
- ??, ??, ??, ??, ??, ??, ??,
- ??, ??, ??, ??, ??, ??, ??
- slepton2_squark2*, ??, used: ??
- slepton2_squark2'*, ??, used: ??
- slepton2_squark2''*, ??, used: ??
- slepton4_1gen*, ??, used: ??
- slepton4_1gen'*, ??, used: ??
- slepton4_2gen*, ??, used: ??
- slepton4_2gen'*, ??, used: ??
- slep_sneu_squark2*, ??, used: ??
- slep_sneu_squark2'*, ??, used: ??
- slep_sneu_squark2''*, ??, used: ??
- slist*, ??, used: ??
- slow_binomial*, **330**, used: **330**
- SLR*, ??, used: ??, ??, ??, ??,
- ??, ??, ??, ??, ??, ??, ??,
- ??, ??, ??, ??, ??, ??, ??,
- ??, ??, ??, ??, ??, ??, ??,
- ??, ??, ??, ??, ??, ??, ??,
- ??, ??, ??, ??, ??, ??, ??,
- ??, ??, ??, ??, ??, ??, ??,
- ??, ??, ??, ??, ??, ??, ??,
- ??, ??, ??, ??
- SLRV*, ??, used: ??, ??, ??
- SLV*, ??, used: ??, ??
- SM* (module), ??, ??, ??, used:
- ??, ??, ??, ??, ??, ??,
- 282**, ??, ??, ??, ??
- SM3* (module), ??, ??, used: ??
- SM3_clones* (module), ??, ??,
- used:
- Small_Rational* (module), **363**,
- 359**, used:
- SM_anomalous* (module), ??, ??,
- used: ??
- SM_anomalous_ckm* (module),
- ??, ??, used: ??
- SM_clones* (module), ??, ??,
- used:
- SM_flags* (sig), ??, ??, ??, ??,
- used: ??, ??, ??, ??, ??,
- ??, ??, ??, ??, ??, ??, ??,
- ??, ??, ??, ??
- SM_gluons* (module), ??, ??,
- used:
- SM_Hgg* (module), ??, ??, used:
- SM_k_matrix* (module), ??, ??,
- used: ??
- SM_no_anomalous* (module), ??,
- ??, ??, ??, used: ??, ??,
- 282**, ??, ??
- SM_no_anomalous_ckm* (module),
- ??, ??, used: ??
- SM_Rxi* (module), ??, ??, used:
- SN*, ??, ??, ??, used: ??, ??, ??,
- ??, ??, ??, ??, ??, ??, ??,
- ??, ??, ??, ??, ??, ??, ??
- sneu2_slep2_1*, ??, used: ??
- sneu2_slep2_1'*, ??, used: ??
- sneu2_slep2_2*, ??, used: ??
- sneu2_slep2_2'*, ??, used: ??
- sneu2_squark2*, ??, used: ??
- sneu2_squark2'*, ??, used: ??
- Sneutrino*, ??, ??, ??, used: ??,
- ??, ??, ??, ??, ??, ??, ??,
- ??, ??, ??, ??, ??, ??, ??,
- ??, ??, ??, ??, ??, ??, ??,
- ??, ??, ??, ??, ??, ??, ??,
- ??, ??, ??, ??, ??, ??, ??,
- ??, ??, ??, ??, ??, ??, ??,
- ??, ??, ??, ??, ??
- sneutrino4*, ??, used: ??
- sneutrino4'*, ??, used: ??
- solve*, **373**, **369**, used:
- solve_destructive*, **372**, used: **373**
- solve_many*, **373**, **369**, used: **351**
- solve_many_destructive*, **373**,
- used: **373**
- sort*, **344**, **256**, **259**, **341**, **255**,
- used: ??, ??, **367**, **368**,
- 333**, **334**, **55**, **258**, **260**,
- 264**, **265**, **115**, **129**, **275**
- sort'*, **344**, used: **344**, **344**
- sort3*, **260**, used: **260**
- sorted_keys*, **46**, used:
- sort_2i*, **345**, used: **347**, **348**
- sort_2i'*, **345**, used: **345**, **345**
- sort_children*, ??, used: ??
- sort_decay*, **256**, **257**, **255**, used:
- 257**, **259**
- sort_momenta*, **258**, used: **259**,
- 260**
- sort_signed*, **336**, **329**, used: **105**,
- 126**
- sort_spin_table*, **129**, used: **129**
- source*, **287**, **286**, used: **288**
- source* (field), **287**, **286**, used:
- 288**, **337**, **10**, **22**, **42**, **54**,
- ??, **262**, **174**, **103**, ??, ??,

- ??, ??, ??, **244**, **184**, **278**,
 ??, ??
SP, ??, used: ??, ??, ??
spacelike, **54**, **59**, **64**, **53**, used:
 260, 263
Sparse (module), **358**, used:
species, **345**, used: 346, 346, 346
spin, **191**, used: 215, 215
spinor, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, used: ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??
Spinor, ??, used: 176, 180, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, 279, ??, ??
spinors (field), **191**, used: 192,
 192, 198
split, **308**, **332**, **10**, **12**, **13**, **16**,
17, **54**, **60**, **64**, **328**, **7**,
53, used: 308, 310, 311,
 368, 333, 336, 351, 17, 36,
 264, 105, 113
split', **331**, used: 331, 331, 332
splitn, **298**, **297**, used: 157, 166,
 193, 193
splitn', **298**, used: 298, 298
split_color_string, **154**, **163**,
 used: 154, 163
sqrt, **355**, **357**, **354**, used: 356,
 357
Sqrt, ??, used: ??, ??
squ, ??, used:
SR, ??, used: ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??
SRV, ??, used:
standard_gauge_higgs, ??, ??,
 ??, ??, ??, ??, ??, ??,
 used: ??, ??, ??, ??, ??,
 ??, ??, ??
standard_gauge_higgs', ??, ??,
 ??, used: ??, ??, ??
standard_gauge_higgs4, ??, ??,
 ??, ??, ??, ??, ??, used:
 ??, ??, ??, ??, ??, ??, ??
standard_higgs, ??, ??, ??, ??,
 ??, ??, ??, ??, used: ??,
 ??, ??, ??, ??, ??, ??, ??
standard_higgs4, ??, ??, ??, ??,
 ??, ??, ??, used: ??, ??,
 ??, ??, ??, ??, ??, ??, ??
standard_quartic_gauge, ??, ??,
 ??, ??, ??, ??, ??, ??,
 used: ??, ??, ??, ??, ??, ??,
 ??, ??, ??
standard_triple_gauge, ??, ??,
 ??, ??, ??, ??, ??, ??,
 used: ??, ??, ??, ??, ??, ??,
 ??, ??, ??
stat, **104**, **105**, **107**, **125**, used:
 107, 119
stat (type), **104**, **105**, **107**, **125**,
 used: 104, 107, 113, 115,
 116, 116, 116
Stat (module), **115**, used:
Stat (sig), **104**, used: 104, 104,
 125
state (type), **291**, used: 291
Stat_Dirac (module), **104**, used:
 124, 126, 127, 127
stat_fuse, **104**, **105**, **107**, **125**,
 used: 107, 113, 117
stat_keystone, **117**, used: 117
Stat_Majorana (module), **125**,
 used: 126, 126, 127, 127
Stat_Maker (sig), **104**, used: 107
stat_sign, **104**, **106**, **107**, **126**,
 used: 107, 113, 117
step (field), **291**, used: 292, 293,
 293, 293
steps (field), **291**, used: 293, 293
string_of_angle, ??, used: ??
string_of_char, ??, ??, ??, used:
 ??, ??, ??, ??, ??, ??
string_of_fermion_gen, ??, ??,
 used: ??, ??
string_of_fermion_type, ??, ??,
 used: ??, ??
string_of_gen, ??, ??, ??, used:
 ??, ??, ??
string_of_higgs, ??, used: ??, ??
string_of_neu, ??, ??, ??, used:
 ??, ??, ??, ??, ??, ??, ??
string_of_phiggs, ??, ??, used:
 ??, ??, ??, ??
string_of_sff, ??, ??, ??, used:
 ??, ??, ??, ??

- string-of_sfm*, ??, ??, ??, used:
 ??, ??, ??
string-of_shiggs, ??, ??, used:
 ??, ??, ??, ??
strip_array_tag, **195**, used: **195**
strip-before_a_keyword, **287**,
 used: **288**
strip-before_keyword, **287**, used:
strip_dollars, **287**, used: **287**, **287**
strip-from_first, ??, ??, used:
287
strip-from_last, ??, ??, used:
287
strip-keyword, **287**, used: **287**,
287
strip-prefix, ??, ??, used: **287**,
287
strip-prefix_star, ??, ??, used:
287
strip-required_prefix, ??, ??,
 used: **287**
strip-single_tag, **195**, used: **195**
strip_svn_repos, **288**, used: **288**
sty, **348**, **341**, used:
style, **345**, used: **345**
style (field), **345**, **341**, used: **345**,
348, **348**
SU, ??, ??, ??, used: ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??
su0, **134**, **159**, used: **136**, **155**,
156, **164**, **165**
sub, **362**, **363**, **363**, **364**, **366**,
367, **54**, **57**, **62**, **355**,
356, **359**, **360**, **361**, **52**,
354, used: ??, ??, ??, ??,
 ??, ??, **287**, **287**, **366**, **366**,
367, **350**, **57**, **62**, **258**, **261**,
174, ??, **154**, **163**, **273**, ??,
187, **230**, **69**
sub', **57**, used: **57**, **57**
substitute_char, **176**, used: **176**
substring, **174**, used: **175**, **175**,
175
subtract, **171**, **170**, used: ??, **173**
succ, **29**, **30**, **21**, used: **299**, **299**,
300, **303**, ??, ??, ??, ??,
291, **293**, **320**, **323**, **371**,
371, **371**, **372**, **337**, **338**,
346, **349**, **349**, **17**, **23**, **26**,
30, **32**, **34**, **34**, **35**, **48**, **56**,
59, **61**, **65**, **263**, **174**, **175**,
175, **73**, **154**, **155**, **163**,
164, **115**, **273**, **273**, **273**,
 ??, ??, ??, **187**, **195**, **213**,
219, **220**, **220**, **221**, **230**
suffix, **294**, **197**, used: **295**, **212**,
216
sum, **363**, **364**, **366**, **367**, **351**,
361, **361**, used: **351**
Sum, **170**, **170**, ??, used: **171**,
171, **171**
summary, **288**, **293**, **286**, **291**,
 used: **266**, **130**, **272**, **196**,
226
SUM_1, ??, used:
SUN, **80**, **77**, used: **80**, **180**, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
279, ??, ??
sup (field), **344**, **345**, used: **344**,
345, **345**
Sup, **345**, ??, ??, ??, used: **345**,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??
sup2_sdown2, ??, used: ??
sup2_sdown2_1, ??, used: ??
sup2_sdown2_2, ??, used: ??
sup2_sdown2_3, ??, used: ??
sup4_1gen, ??, used: ??
sup4_1gen', ??, used: ??
sup4_2gen, ??, used: ??
sup4_2gen', ??, used: ??
Supp, ??, ??, ??, used:
Supp2, ??, ??, ??, used:
supremum_or_infinity (type),
345, used: **345**
SV, ??, used:
swap, **370**, used: **371**, **371**
symbol, **170**, **170**, used: ??, **173**

- symbol* (type), **178**, used: **178**
symbol (camllex regexp), **172**,
 used: **172**
Symbol, **170, 170**, used: **170**
SYMBOL, **??, ??**, used: **??, 172**
SYMBOL (camlyacc token), **173**,
 used: **173**
symmetry, **331, 34, 103, 119**,
327, 100, used: **34, 121**,
215
symmetry (field), **117**, used: **119**,
120
Syntax_Error (exn), **69, 67**,
 used:
system_cache_dir, **??, ??**, used:
294
s_channel, **54, 59, 64, 103, 123**,
123, 53, 100, used: **263**,
266, 217
s_channel_in, **54, 59, 64, 53**,
 used: **59, 59, 64, 64**
s_channel_out, **54, 59, 64, 53**,
 used: **59, 64**
s_channel_out', **59**, used: **59, 59**
S_NNA, **??**, used:
S_NNG, **??**, used:
S_NNH1, **??**, used:
S_NNH2, **??**, used:
t (type), **306, 306, 312, 287**,
295, 295, 291, 318, 318,
321, 321, ??, 362, 363,
363, 363, 364, 364, 365,
367, 289, 289, 342, 10,
12, 13, 14, 17, 29, 30,
31, 42, 42, 42, 43, 44,
44, 44, 45, 45, 45, 47,
50, 54, 55, 60, 65, 65,
66, 256, 259, 355, 355,
358, 80, 80, 81, 82, ??,
??, ??, ??, ??, 262, 263,
263, 264, 266, 68, 72,
73, 109, 109, 113, 114,
114, 115, 120, 123, 123,
124, 191, 305, 286, 294,
294, 291, 316, 317, ??,
359, 360, 360, 361, 289,
340, 7, 21, 37, 38, 39,
41, 51, 54, 255, 354, 77,
77, ??, 262, 67, 71, ??,
70, 70, 173, 173, used:
306, 306, 295, 318, 318,
321, 321, ??, 362, 362,
363, 363, 364, 364, 365,
367, 289, 342, 10, 11, 13,
14, 16, 22, 24, 26, 28, 28,
29, 30, 31, 35, 36, 42, 42,
42, 43, 43, 43, 43, 44, 45,
45, 45, 45, 45, 45, 46, 47,
49, 50, 50, 54, 56, 61, 65,
65, 66, 256, 257, 355, 355,
355, 355, 355, 355, 355,
355, 355, 358, 80, ??, ??,
262, 263, 263, 263, 264,
178, 178, 68, ??, 72, 72,
72, 72, 73, 103, 104, 104,
106, 106, 106, 107, 107,
108, 108, 108, 109, 109,
114, 115, 116, 116, 116,
117, 127, 128, 128, 271,
278, 185, 191, 305, 286,
286, 286, 286, 286, 294,
291, 316, 316, 316, 316,
316, 316, 317, 317, 317,
317, 317, 317, 317, ??, ??,
359, 359, 360, 360, 360,
360, 361, 361, 361, 361,
361, 361, 361, 361, 361,
361, 361, 289, 337, 340,
340, 341, 341, 341, 341,
341, 341, 341, 342, 342, 7,
7, 7, 7, 7, 7, 7, 8, 8, 8,
8, 8, 9, 9, 9, 9, 21, 21, 21,
22, 22, 22, 37, 38, 38, 38,
38, 38, 39, 39, 39, 39, 39,
39, 39, 39, 40, 40, 40, 40,
40, 40, 40, 40, 40, 40, 41,
41, 41, 41, 41, 42, 51, 52,
52, 52, 52, 52, 52, 52, 52,
52, 53, 53, 53, 53, 53, 53,
53, 53, 53, 53, 54, 54, 54,
54, 255, 255, 255, 255,
255, 354, 354, 354, 354,
354, 354, 354, 354, 354,
77, 77, ??, 262, 262, 67,
67, 67, ??, 71, 71, 72, 98,
98, 99, 100, 101, 101, 101,
102, 102, 103, 103, 271,
??, ??, ??, ??, ??, ??, ??,
??, ??, 70
T, **279**, used: **279, 279, 280, 280**

T (module), **272**, used: 272, 275
T (sig), **306, 295, 318, 22, 42,**
 54, 256, 355, 262, 72,
 103, 271, 305, 294, 316,
 20, 39, 51, 255, 354,
 262, 71, 98, 271, ??, ??,
 used: 318, 321, 49, 257,
 263, 72, 133, 159, 104,
 104, 104, 106, 107, 125,
 127, 127, 272, 184, 189,
 278, 306, 306, 294, 317,
 318, 21, 22, 22, 40, 41, 54,
 255, 355, 262, 174, 72,
 133, 133, 101, 102, 103,
 271, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??
T2, **14, 9**, used: 15, 15, 16, 16,
 16, 16
T3, **14, 9**, used: 15, 15, 16, 16,
 16, 16

??, ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??, ??,
 ??
Targets_Kmatrix (module), **244**,
244, used: **227**
Tau, ??, used: ??, ??, ??, ??
Tbar, **279**, used: **279**, **279**, **280**,
280
Template (module), ??, ??, used
 ??
tension (field), **345**, **348**, **341**,
 used: **348**, **348**
tension_to_equation, **351**, used:
351
tensors_1 (field), **191**, used: **192**,
192, **198**
tensors_2 (field), **191**, used: **192**,
192, **198**
Tensor_1, ??, used: ??, ??
Tensor_2, ??, used: ??, ??
Tensor_2_Vector_Vector, ??,
 used: **137**
Term (module), **364**, **361**, used:
Term (sig), **363**, **360**, used: **364**,
365, **361**, **361**
Ternary (module), **13**, **28**, **9**, **21**,
 used: **16**, **28**, **21**
Ternary (sig), **13**, **9**, used: **9**
test_rhs, **117**, used: **117**
test_sum, **59**, **64**, used: **60**, **64**
tex_lbl, **345**, used: **346**, **346**, **346**
tgc, ??, ??, ??, ??, ??, ??, ??,
 ??, ??, used: ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??
tgc_aux, ??, used: ??, ??
ThoArray (module), **302**, **302**,
 used:
ThoList (module), **298**, **297**,
 used: **367**, **368**, **325**, **331**,
333, **334**, **334**, **334**, **335**,
335, **342**, **343**, **348**, **348**,
349, **349**, **17**, **18**, **18**, **18**,
18, **26**, **28**, **35**, **36**, **60**, ??,
 ??, **265**, **266**, **181**, **73**, **136**,
136, **137**, **137**, **141**, **147**,
148, **149**, **149**, **150**, **150**,
152, **152**, **152**, **152**, **152**,
153, **156**, **157**, **162**, **163**,
163, **165**, **166**, **112**, **120**,

- 121, 122, 123, 123, 129,
130, 130, 130, 130, 272,
274, 274, 275, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
191, 193, 193, 196, 199,
213, 218, 226, 279, ??, ??,
??, ??, ??, ??, ??, ??, ??
- ThoString* (module), ??, ??,
used: 287, 275
- thread*, 325, 324, used:
- thread_unfinished_decays*, 260,
used: 261
- thread_unfinished_decays'*, 260,
used: 260, 260, 261
- three*, 30, used: 31, 31, 32, 33, 33
- Threeshl* (module), ??, ??, used:
??, ??
- Threeshl_ckm* (module), ??, ??,
used:
- Threeshl_ckm_no_hf* (module),
??, ??, used:
- Threeshl_diet* (module), ??, ??,
used:
- Threeshl_diet_no_hf* (module),
??, ??, used:
- Threeshl_no_ckm* (module), ??,
??, used: ??
- Threeshl_no_ckm_no_hf* (module),
??, ??, used: ??
- Threeshl_options* (sig), ??, ??,
used: ??, ??, ??, ??, ??,
??, ??, ??
- three_gluon*, 280, used: 280
- Thw*, ??, used:
- Th_SF*, ??, used:
- timelike*, 54, 59, 64, 53, used:
59, 64, 263, 266, 111, 123,
123
- Timelike*, ??, used: ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
279, ??, ??
- timelike_map*, 263, used: 263,
264
- time_to_string*, 292, used: 293,
293
- tln*, 298, 297, used: 298
- token*, ??, ??, ??, ??, used: ??,
179, ??, ??, 73, 69, 172
- token* (type), ??, ??, ??, ??,
used: ??, ??, ??, ??
- token* (camllex parsing rule), 69,
172, used:
- TopH*, ??, used: ??, ??, ??, ??,
??, ??, ??
- TopHb*, ??, used: ??, ??, ??, ??,
??, ??, ??
- Topology* (module), 22, 20, used:
107, 124, 126, 126, 127,
127
- Topp*, ??, used: ??, ??, ??, ??,
??, ??, ??
- Toppb*, ??, used: ??, ??, ??, ??,
??, ??, ??
- top_quartic*, ??, ??, ??, used: ??,
??
- tower_of_dag*, 115, used:
- tower_to_dot*, 103, 124, 101,
used: 275
- to_feynmf*, 347, 342, used:
- to_feynmf_channel*, 347, used:
347
- to_float*, 362, 363, 355, 357,
359, 354, used:
- to_float2*, 355, 357, 354, used:
- to_gauss*, 76, used: 76
- to_int*, 29, 30, 355, 357, 21,
354, used: 35
- to_int2*, 355, 357, 354, used:
- to_ints*, 54, 56, 62, 52, used: 62,
66, 261, 265, 108
- to_ints'*, 61, used: 61, 62
- to_list*, 10, 12, 14, 16, 18, 81, 8,
used: 350, 351, 81, 81, 81,
109, 109, 111, 111, 112,
113, 122, 123, 275
- to_lists*, 80, 81, 77, used: 220

- to_momentum*, **65, 65, 66, 54**,
used: **66**
- to_on_shell*, **76**, used: **76**
- to_ratio*, **362, 363, 359**, used:
- to_selectors*, **72, 76, 71**, used:
275
- to_select_p*, **75**, used: **76**
- to_select_wf*, **75**, used: **76**
- to_string*, **??, 362, 363, 363**,
363, 364, 365, 367, 368,
345, 29, 30, 54, 56, 62,
261, 355, 358, 68, 72,
74, ??, 359, 360, 361,
361, 341, 21, 53, 354,
67, 71, used: **295, ??**,
367, 368, 74, 76, 275
- to_string1*, **261**, used: **261, 261**
- translate_constant*, **181**, used:
181
- translate_tensor3*, **181**, used: **181**
- translate_tensor4*, **181**, used: **181**
- transpose*, **299, 303, 298**, used:
303, 304, 130, 130
- tree* (type), **263**, used: **263**
- Tree* (module), **306, 342, 306**,
340, used: **303, 349, 42**,
49, 103, 122, 271, 275,
278, 40, 100, 271
- Tree2* (module), **??, ??**, used: **42**,
48, 103, 117, 191, 40, 99
- trees*, **262, 264, 262**, used: **275**
- trees* (field), **263**, used: **264, 265**,
266
- Trie*, **318, 321**, used: **318, 318**,
319, 319, 319, 321, 321,
322, 322, 322
- Trie* (module), **318, 316**, used:
- triples*, **338, ??, ??, 337**, used:
14, 16, ??, ??
- triplet_gauge2_higgs*, **??, ??**,
used: **??, ??**
- triplet_gauge_higgs*, **??, ??**, used:
??, ??
- triple_anom*, **??, ??, ??, ??, ??**,
??, ??, ??, used: **??, ??**
- triple_gauge*, **??, ??, ??, ??, ??**,
??, ??, ??, ??, ??, ??, ??,
??, ??, used: **??, ??, ??**,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??
- triple_gluon*, **??**, used: **??**
- triple_gravitino*, **??**, used: **??**
- triple_gravitino'*, **??**, used: **??**
- triple_gravitino''*, **??**, used: **??**
- True*, **68, 73, 67**, used: **68, 68**,
73, 74
- try_add*, **54, 57, 62, 52**, used:
59, 64
- try_add'*, **57**, used: **57, 57**
- try_first*, **296**, used: **296**
- try_fusion*, **54, 60, 64, 53**, used:
260
- try_of_momenta*, **261**, used:
- try_sub*, **54, 58, 63, 52**, used:
59, 63, 64, 261
- try_sub'*, **58**, used: **58, 58**
- try_thread_unfinished_decays*,
261, used: **261**
- TS* (module), **287**, used: **287**,
287, 287, 287, 287
- Tuple* (module), **10, 26, 35, 7**,
used: **24, 25, 26, 26, 28**,
28, 28, 35, 35, 35, 36, 36,
44, 106, 106, 107, 107,
124, 126, 126, 127, 127,
21, 22, 22, 38, 101, 101,
103
- tuples*, **338, 337**, used: **338, 338**,
18, 26, 35
- tuples'*, **338**, used: **338, 338**
- tuple_of_list2*, **25**, used: **26**
- twice_spin*, **216**, used: **216**
- two*, **30**, used: **31, 32, 32, 32, 33**,
33, 33, 34
- two_and_one*, **??, ??**, used: **??**,
??, ??, ??
- two_and_one'*, **??, ??**, used: **??**,
??, ??, ??
- t_channel* (field), **260**, used: **260**,
260, 260, 261
- U*, **??, ??, ??, ??, ??, ??, ??, ??**,
??, ??, ??, ??, 279, ??,
??, used: **??, ??, ??, ??**,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??

[illegible]

- ??, ??, ??, ??, ??, ??, ??, ??,
 ??, **280**, ??, ??, ??, ??
- V2*, ??, used: ??
- V2LR*, ??, used: ??, ??
- v3* (field), ??, used: ??, ??, ??
- V3*, ??, used: ??, ??
- v4* (field), ??, used: ??, ??, ??
- V4*, ??, used: ??, ??
- VA*, ??, used: ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??
- VA2*, ??, used: ??, ??, ??, ??, ??
- value* (field), **295**, used:
- value* (type), **295, 295, 294**,
 used: **295, 295, 294, 294**
- Value* (sig), **295, 294**, used: **295**,
294
- vanilla*, **348, 341**, used: **348**
- Vanishing*, ??, used: ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??
- vanishing_color_flows*, **127, 128**,
102, used: **225**
- vanishing_color_flows* (field),
128, used: **128**
- vanishing_flavors*, **127, 128**,
102, used: **225**
- vanishing_flavors* (field), **128**,
 used: **128**
- variable*, **124, 272, 191**, used:
124, 275, 191, 191, 191,
212, 217
- variable* (type), ??, used: ??
- variable'*, **272**, used:
- variables*, **103, 119, 100**, used:
199
- variables_file*, **182**, used: **182**,
183
- variable_array* (type), ??, used:
 ??
- vc* (type), ??, ??, ??, used: ??,
 ??, ??
- VCache* (module), **109**, used:
110, 110, 121
- vector* (type), **172**, used:
- Vector*, ??, used: **180**, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, **279**,
 ??, ??
- Vector4*, ??, used: **138**, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, **280**, ??, ??
- Vector4_K_Matrix_jr*, ??, used:
138, ??
- Vector4_K_Matrix_tho*, ??,
 used: **138**
- vectors* (field), **191**, used: **192**,
192, 198
- Vectorspinor*, ??, used: ??, ??
- vectorspinors* (field), **191**, used:
192, 192, 198
- vector_keyword*, **172**, used:
- Vector_Scalar_Scalar*, ??, used:
137, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??
- version*, **272**, used: **275**
- vertex3* (type), ??, used: ??, **109**,
 ??, ??, ??, ??
- vertex4* (type), ??, used: ??, **109**,
 ??, ??, ??, ??
- vertexn* (type), ??, used: ??, **109**,
 ??, ??, ??, ??
- vertex_table* (type), **109**, used:
109
- vertices*, ??, **177, 153, 157, 161**,
110, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
280, ??, ??, ??, used:
177, 138, 154, 157, 110,
121, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, **280**, ??, ??
- vertices* (label), ??, ??, used:
- vertices* (type), **109**, used: **109**,
109, 110
- vertices3*, **138**, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, **280**, ??, ??, used:
153, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
280, ??, ??
- vertices3'*, ??, used: ??
- vertices3''*, ??, used: ??
- vertices4*, **138**, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,

- 280**, ??, ??, used: **153**,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, **280**, ??, ??
vertices4', ??, used: ??
vertices4'', ??, used: ??
vertices4''', ??, used: ??
verticesn, **138**, used: **153**
vertices_aaww, ??, used: ??
vertices_all, ??, used: ??
vertices_aqq, ??, used: ??
vertices_aww, ??, used: ??
vertices_cache, **110**, used: **110**
vertices_ggg, ??, used: ??
vertices_gggg, ??, used: ??
vertices_gqq, ??, used: ??
vertices_nocache, **109**, used: **110**,
121
vertices_wll, ??, used: ??, ??
vertices_wll_diet, ??, used: ??
vertices_wqq, ??, used: ??
vertices_wqq-no-ckm, ??, used:
 ??, ??
vertices_wqq-no-ckm-diet, ??,
 used: ??
vertices_www, ??, used: ??
vertices_wwza, ??, used: ??
vertices_wwzz, ??, used: ??
vertices_zll, ??, used: ??
vertices_zqq, ??, used: ??
vertices_zww, ??, used: ??
Vev, ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, used:
 ??, ??
VHeavy, ??, ??, ??, used:
VL, ??, used: ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??
VLR, ??, used: ??, ??, ??, ??,
 ??, ??
VM (module), **242**, **184**, used:
VMOM, ??, used:
vn (field), ??, used: ??, ??, ??
Vn, ??, used:
VR, ??, used: ??, ??
VSet (module), **109**, used: **109**,
109
V_0, ??, used:
V_CKM, ??, used:
V_P, ??, used:
- w*, ??, used: ??
W, ??, ??, used: ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??
W (module), **272**, used: **275**
ward_vectors (field), **191**, used:
192, **198**
warn, **197**, used: **212**, **216**
Warn, **197**, used: **198**
wavefunctions, **113**, used: **121**
wf (type), **103**, **106**, **106**, **107**,
108, **98**, **102**, used: **103**,
106, **108**, **108**, **113**, **113**,
114, **114**, **115**, **115**, **116**,
116, **116**, **117**, **117**, **117**,
120, **123**, **123**, **191**, **191**,
98, **98**, **98**, **99**, **99**, **99**, **99**,
100, **100**, **100**, **102**
WFFMap (module), **120**, **191**,
 used: **120**, **121**, **121**, **191**,
213
WFFMap2 (module), **191**, used:
191, **213**
WFSet (module), **191**, used: **199**,
217, **218**
WFSet2 (module), **191**, used:
214, **218**
WFTSet (module), **191**, used:
213
wf-of-f, **235**, used: **235**, **236**,
236, **237**, **237**, **238**, **238**,
238, **239**, **239**, **239**, **240**,
240, **240**
wf_tag, **103**, **108**, **98**, used: **191**
wf_tag (field), **108**, used: **108**,
111, **113**, **113**, **119**, **122**
wf_tag_raw, **108**, used: **108**
wf_to_string, **106**, **106**, **107**,
102, used: **108**
white (camlex regexpr), **69**,
 used: **69**
White, **134**, **159**, used: **135**, **135**,
136, **138**, **138**, **139**, **139**,
140, **141**, **141**, **142**, **142**,
142, **142**, **142**, **143**, **143**,
143, **144**, **144**, **144**, **145**,
145, **145**, **146**, **146**, **146**,
147, **147**, **151**, **151**, **151**,
151, **151**, **151**, **152**, **152**,
152, **152**, **153**, **154**, **156**,

[illegible][illegible]

- $??, ??, ??, ??, ??, ??, ??,$
 $??, ??, ??, ??, ??, ??, ??,$
 $??, ??, ??, ??, ??, ??, ??,$
 $??, ??, ??, ??, ??, ??, ??,$
 $??, ??$
Wp1, $??$, used: $??, ??, ??, ??$
Wp2, $??$, used: $??, ??, ??, ??$
wrap_newline, **190**, used: **196**,
227
write, **295, 296, 262, 266, 294**,
262, used: **110**
write_dir, **295, 296, 294**, used:
121
write_interface, **268, 262**, used:
write_interface_function, **267**,
used: **268**
write_interface_subroutine, **267**,
used: **268**
write_makefile, **269, 262**, used:
write_makefile_processes, **269**,
262, used:
write_other_declarations, **268**,
used: **268**
write_other_interface_functions,
268, used: **268**
X0, $??$, used: $??, ??, ??, ??, ??,$
 $??, ??$
XDH2, $??$, used: $??, ??, ??$
XDH-W, $??$, used: $??, ??, ??$
XDH-W', $??$, used: $??, ??, ??$
XDH-Wm, $??$, used: $??, ??, ??$
XDH-Wp, $??$, used: $??, ??, ??$
XDH-Z, $??$, used: $??, ??, ??$
XDH-Z', $??$, used: $??, ??, ??$
XDH-Z'', $??$, used: $??, ??, ??$
Xdim (module), $??, ??$, used: $??$
XGl, $??$, used: $??, ??$
XH, $??$, used: $??, ??, ??$
XH-W, $??$, used: $??, ??, ??$
XH-W', $??$, used: $??, ??, ??$
XH-Z, $??$, used: $??, ??, ??$
XH-Z', $??$, used: $??, ??, ??$
XiA, $??$, used: $??$
XiW, $??$, used: $??$
XiZ, $??$, used: $??$
Xm, $??$, used: $??, ??, ??, ??, ??,$
 $??, ??$
Xp, $??$, used: $??, ??, ??, ??, ??,$
 $??, ??$
XSW3, $??$, used: $??, ??, ??$
XSWm, $??$, used: $??, ??, ??$
XSWmm, $??$, used: $??, ??, ??$
XSWp, $??$, used: $??, ??, ??$
XSWpp, $??$, used: $??, ??, ??$
XSWZ0, $??$, used: $??, ??, ??$
XSZW0, $??$, used: $??, ??, ??$
XSZZ, $??$, used: $??, ??, ??$
XW3, $??$, used: $??, ??, ??$
XWm, $??$, used: $??, ??, ??$
XWp, $??$, used: $??, ??, ??$
xy_currents, $??$, used: $??$
Y0, $??$, used: $??, ??, ??, ??, ??$
YM (module), $??, ??$, used:
yukawa, $??, ??, ??, ??, ??, ??,$
 $??, ??, ??, ??, ??$, used:
 $??, ??, ??, ??, ??, ??, ??,$
 $??, ??, ??, ??$
yukawa_add, $??, ??, ??$, used:
 $??, ??, ??$
yukawa_add', $??, ??$, used: $??,$
 $??$
yukawa-c, $??, ??, ??$, used: $??,$
 $??, ??$
yukawa-cq, $??, ??, ??$, used: $??,$
 $??, ??$
yukawa-cq', $??, ??, ??$, used: $??,$
 $??, ??$
yukawa-cq'', $??$, used: $??$
yukawa-c_1, $??$, used: $??$
yukawa-c_2, $??, ??, ??$, used:
 $??, ??, ??$
yukawa-c_3, $??, ??, ??$, used:
 $??, ??, ??$
yukawa-goldstone, $??$, used: $??$
yukawa-goldstone_2, $??$, used:
 $??$
yukawa-goldstone_2', $??$, used:
 $??$
yukawa-goldstone_2'', $??$, used:
 $??$
yukawa-goldstone-quark, $??$,
used: $??$
yukawa-higgs, $??, ??, ??$, used:
 $??, ??, ??$
yukawa-higgs_2, $??, ??, ??$,
used: $??, ??, ??$
yukawa-higgs_2', $??$, used: $??$
yukawa-higgs_2'', $??$, used: $??$
yukawa-higgs_FFP, $??, ??$, used:
 $??, ??$

[illegible]

zero, **29, 30, 54, 56, 61, 21, 52**,
 used: **31, 31, 31, 32, 32**,
 32, 33, 33, 33, 34, 34, 35,
 73
Zero (exn), **32**, used:
ZH, ??, ??, ??, ??, used: ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??, ??, ??, ??,
 ??, ??, ??, ??
Zprime (module), ??, used: ??
--ocaml_lex_tables, ??, ??, used:
 ??, ??
--ocaml_lex_token_rec, ??, ??,
 used: ??, ??, ??, ??