O'Mega: An Optimizing Matrix Element Generator. I: Basic Algorithms

Mauro Moretti*

Dipartimento di Fisica, Università di Ferrara
and INFN, Sezione di Ferrara, Ferrara, Italy

Thorsten Ohl†

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

Jürgen Reuter‡

Deutsches Elektronen-Synchrotron DESY
Notkestraße 85, 22607 Hamburg, Germany

LC-TOOL-2001-040	DESY 11-130	hep-ph/0102195

June 15, 2012

*moretti@fe.infn.it
†ohl@physik.uni-wuerzburg.de, http://physik.uni-wuerzburg.de/ohl
‡juergen.reuter@desy.de

Abstract

We sketch the architecture of *O'Mega*, an optimizing compiler for tree amplitudes in quantum field theory, and briefly describe its usage. O'Mega generates optimally efficient code for scattering amplitudes for many polarized particles in the Standard Model and its extensions.

1 Introduction

Current and planned experiments in high energy physics can probe physics in processes with polarized beams and many tagged particles in the final state. The combinatorial explosion of the number of Feynman diagrams contributing to scattering amplitudes for many external particles calls for the development of more compact representations that translate well to efficient and reliable numerical code. In gauge theories, the contributions from individual Feynman diagrams are gauge dependent. Strong numerical cancellations in a redundant representation built from individual Feynman diagrams lead to a loss of numerical precision, stressing further the need for eliminating redundancies.

Due to the large number of processes that have to be studied in order to unleash the potential of modern experiments, the construction of nearly optimal representations must be possible algorithmically on a computer and should not require human ingenuity for each new application.

O'Mega [1] is a compiler for tree-level scattering amplitudes that satisfies these requirements. O'Mega is independent of the target language and can therefore create code in any programming language for which a simple output module has been written. To support a physics model, O'Mega requires as input only the Feynman rules and the relations among coupling constants.

Similar to the numerical approaches [2] and [3], O'Mega reduces the growth in calculational effort from a factorial of the number of external particles to an exponential. The symbolic nature of O'Mega, however, increases its flexibility. Indeed, O'Mega can emulate both [2] and [3] and produces code that is empirically at least twice as fast. The detailed description of all algorithms is contained in the extensively commented source code of O'Mega.

In this note, we sketch the architecture of O'Mega and describe the usage of the first version. The building blocks of the representation of scattering amplitudes generated by O'Mega are described in section 2 and directed acyclical graphs are introduced in section 3. The algorithm for constructing the directed acyclical graph is presented in section 4 and its implementation is described in section 5. We conclude with a few results and examples in section 6. Practical information is presented in the appendices: installation

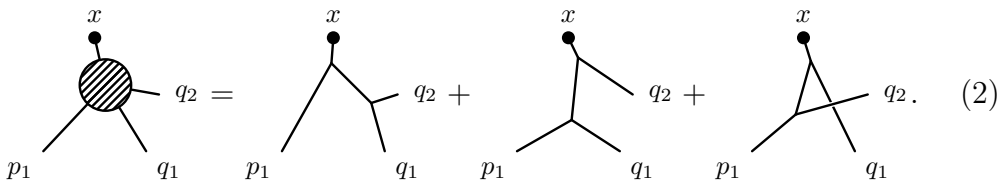
of the O'Mega software in appendix A, running of the O'Mega compiler in appendix B and using O'Mega's output in appendix C. Finally, appendix D briefly discusses mechanisms for extending O'Mega.

2 One-Particle Off-Shell Wave Functions

One-Particle Off-Shell Wave Functions (1POWs) are obtained from connected Green's functions by applying the LSZ reduction formula to all but one external line while the remaining line is kept off the mass shell

$$W(x; p_1, \ldots, p_n; q_1, \ldots, q_m) = \\ \langle \phi(q_1), \ldots, \phi(q_m); \mathrm{out} | \Phi(x) | \phi(p_1), \ldots, \phi(p_n); \mathrm{in} \rangle \ . \quad (1)$$

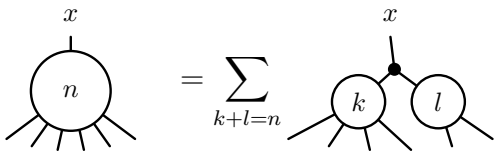
Depending on the context, the off-shell line will either be understood as amputated or not. For example, $\langle \phi(q_1), \phi(q_2); \mathrm{out} | \Phi(x) | \phi(p_1); \mathrm{in} \rangle$ in unflavored scalar ϕ^3-theory is given at tree level by



(2)

The number of distinct momenta that can be formed from n external momenta is $P(n) = 2^{n-1} - 1$. Therefore, the number of tree 1POWs grows exponentially with the number of external particles and not with a factorial, as the number of Feynman diagrams, e. g. $F(n) = (2n-5)!! = (2n-5) \cdot \ldots 5 \cdot 3 \cdot 1$ in unflavored ϕ^3-theory.

At tree-level, the set of all 1POWs for a given set of external momenta can be constructed recursively



$$= \sum_{k+l=n} \quad , \quad (3)$$

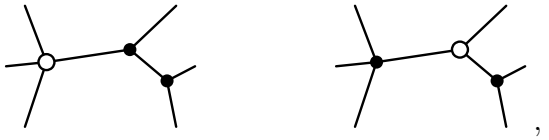
where the sum extends over all partitions of the set of n momenta. This recursion will terminate at the external wave functions.

For all quantum field theories, there are—well defined, but not unique—sets of *Keystones* K such that the sum of tree Feynman diagrams for a given

process can be expressed as a sparse sum of products of 1POWs without double counting. In a theory with only cubic couplings this is expressed as

$$T = \sum_{i=1}^{F(n)} D_i = \sum_{k,l,m=1}^{P(n)} K^3_{f_k f_l f_m}(p_k, p_l, p_m) W_{f_k}(p_k) W_{f_l}(p_l) W_{f_m}(p_m)\,, \qquad (4)$$

with obvious generalizations. The non-trivial problem is to avoide the double counting of diagrams like



where the circle denotes the keystone. The problem has been solved explicitly for general theories with vertices of arbitrary degrees. The solution is inspired by arguments [2] based on the equations of motion (EOM) of the theory in the presence of sources. The iterative solution of the EOM leads to the construction of the 1POWs and the constraints imposed on the 1POWs by the EOM suggest the correct set [2] of partitions $\{(p_k, p_l, p_m)\}$ in equation (4).

The maximally symmetric solution selects among equivalent diagrams the keystone closest to the center of a diagram. This corresponds to the numerical expressions of [2]. The absence of double counting can be demonstrated by counting the number $F(d_{\max}, n)$ of unflavored Feynman tree diagrams with n external legs and vertices of maximum degree $d_{\max}$ in two different ways: once directly and then as a sum over keystones. The number $\tilde{F}(d_{\max}, N_{d,n})$ of unflavored Feynman tree diagrams for one keystone $N_{d,n} = \{n_1, n_2, \ldots, n_d\}$, with $n = n_1 + n_2 + \cdots + n_d$, is given by the product of the number of subtrees and symmetry factors

$$\tilde{F}(d_{\max}, N_{d,n}) = \frac{n!}{|\mathcal{S}(N_{d,n})|\sigma(n_d, n)} \prod_{i=1}^{d} \frac{F(d_{\max}, n_i + 1)}{n_i!} \qquad (5a)$$

where $|\mathcal{S}(N)|$ is the size of the symmetric group of N, $\sigma(n, 2n) = 2$ and $\sigma(n, m) = 1$ otherwise. Indeed, it can be verified that the sum over all keystones reproduces the number

$$F(d_{\max}, n) = \sum_{d=3}^{d_{\max}} \sum_{\substack{N=\{n_1,n_2,\ldots,n_d\} \\ n_1+n_2+\cdots+n_d=n \\ 1\le n_1\le n_2\le\cdots\le n_d\le\lfloor n/2\rfloor}} \tilde{F}(d_{\max}, N) \qquad (5b)$$

of *all* unflavored Feynman tree diagrams.

A second consistent prescription for the construction of keystones is maximally asymmetric and selects the keystone adjacent to a chosen external line. This prescription reproduces the approach in [3] where the tree-level Schwinger-Dyson equations are used as a special case of the EOM.

Interfering color amplitudes are implemented by using O'Mega's basic algorithm for the Feynman rules in the color flow basis [4, 5]. In the case of Dirac fermions, Fermi statistics can be implemented straightforwardly in the 1POWs by maintaining a list of the ordered pairs of external fermions connected by directed fermion lines. This approach can be generalized [4] for models with Majorana fermions and fermion number violating interactions, such as supersymmetric models using the Feynman rules of [6].

Recursive algorithms for gauge theory amplitudes have been pioneered in [7]. The use of 1POWs as basic building blocks for the calculation of scattering amplitudes in tree approximation has been advocated in [8] and a heuristic procedure, without reference to keystones, for minimizing the number of arithmetical operations has been suggested. This approach is used by MADGRAPH [9] for fully automated calculations. The heuristic optimizations are quite efficient for $2 \to 4$ processes, but the number of operations remains bounded from below by the number of Feynman diagrams. In the time since O'Mega was first made available publically, more programs using a recursive construction of amplitudes have been published, e. g. Comix [10].

2.1 Ward Identities

A particularly convenient property of the 1POWs in gauge theories is that, even for vector particles, the 1POWs are 'almost' physical objects and satisfy simple Ward Identities

$$\frac{\partial}{\partial x_\mu} \langle \mathrm{out} | A_\mu(x) | \mathrm{in} \rangle_{\mathrm{amp.}} = 0 \tag{6a}$$

for unbroken gauge theories and

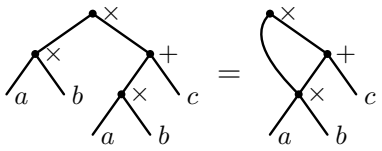
$$\frac{\partial}{\partial x_\mu} \langle \mathrm{out} | W_\mu(x) | \mathrm{in} \rangle_{\mathrm{amp.}} = -m_W \langle \mathrm{out} | \phi_W(x) | \mathrm{in} \rangle_{\mathrm{amp.}} \tag{6b}$$

for spontaneously broken gauge theories in R_ξ-gauge for all physical external states $|in\rangle$ and $|out\rangle$. Thus the identities (6) can serve as powerful numerical checks both for the consistency of a set of Feynman rules and for the numerical stability of the generated code. The code for matrix elements can optionally be instrumented by O'Mega with numerical checks of these Ward identities for intermediate lines.

3 Directed Acyclical Graphs

The algebraic expression for the tree-level scattering amplitude in terms of Feynman diagrams is itself a tree. The much slower growth of the set of 1POWs compared to the set of Feynman diagrams shows that this representation is extremely redundant. In this case, *Directed Acyclical Graphs* (DAGs) provide a more efficient representation, as illustrated by a trivial example

$$ab(ab+c) = \quad = \qquad (7)$$



where one multiplication is saved. The replacement of expression trees by equivalent DAGs is part of the repertoire of optimizing compilers, known as *common subexpression elimination*. Unfortunately, this approach fails in practice for all interesting expressions appearing in quantum field theory, because of the combinatorial growth of space and time required to find an almost optimal factorization.

However, the recursive definition in equation (3) allows to construct the DAG of the 1POWs in equation (4) *directly* [1], without having to construct and factorize the Feynman diagrams explicitly.

As mentioned above, there is more than one consistent prescription for constructing the set of keystones. The symbolic expressions constructed by O'Mega contain the symbolic equivalents of the numerical expressions computed by [2] (maximally symmetric keystones) and [3] (maximally asymmetric keystones) as special cases.

4 Algorithm

By virtue of their recursive construction in (3), tree-level 1POWs form a DAG and the problem is to find the *smallest* DAG that corresponds to a given tree, (i. e. a given sum of Feynman diagrams). O'Mega's algorithm proceeds in four steps

Grow: starting from the external particles, build the tower of *all* 1POWs up to a given height (the height is less than the number of external lines for asymmetric keystones and less than half of that for symmetric keystones) and translate it to the equivalent DAG D.

Select: from D, determine *all* possible *flavored keystones* for the process under consideration and the 1POWs appearing in them.

Harvest: construct a sub-DAG $D^* \subseteq D$ consisting *only* of nodes that contribute to the 1POWs appearing in the flavored keystones.

Calculate: multiply the 1POWs as specified by the keystones and sum the keystones.

By construction, the resulting expression contains no more redundancies and can be translated to a numerical expression. In general, asymmetric keystones create an expression that is smaller by a few percent than the result from symmetric keystones, but it is not obvious which approach produces the numerically more robust results.

The details of this algorithm as implemented in O'Mega are described in the source code. The persistent data structures [11] used for the determination of D^* are very efficient. The generation of Fortran code for amplitudes in the Standard Model is always much faster than the subsequent compilation.

5 Implementation

The O'Mega compiler is implemented in `OCaml` [12], a functional programming language of the ML family with a very efficient, portable and freely available implementation, that can be bootstrapped on all modern computers in a few minutes.

The powerful module system of `OCaml` allows an efficient and concise implementation of the DAGs for a specific physics model as a functor application. This functor maps from the category of trees to the category of DAGs and is applied to the set of trees defined by the Feynman rules of any model under consideration.

The module system of `OCaml` has been used to make the combinatorial core of O'Mega demonstrably independent from the specifics of both the physics model and the target language, as shown in Figure 1. A Fortran backend has been realized first. The electroweak Standard Model and the Minimal Supersymmetric Standard Model (MSSM) have been implemented and tested extensively [13]. Many extensions of these models and more exotic models are available in the distribution.

Many extensions of the Standard Model, most prominently the MSSM, contain Majorana fermions. In this case, fermion lines have no canonical

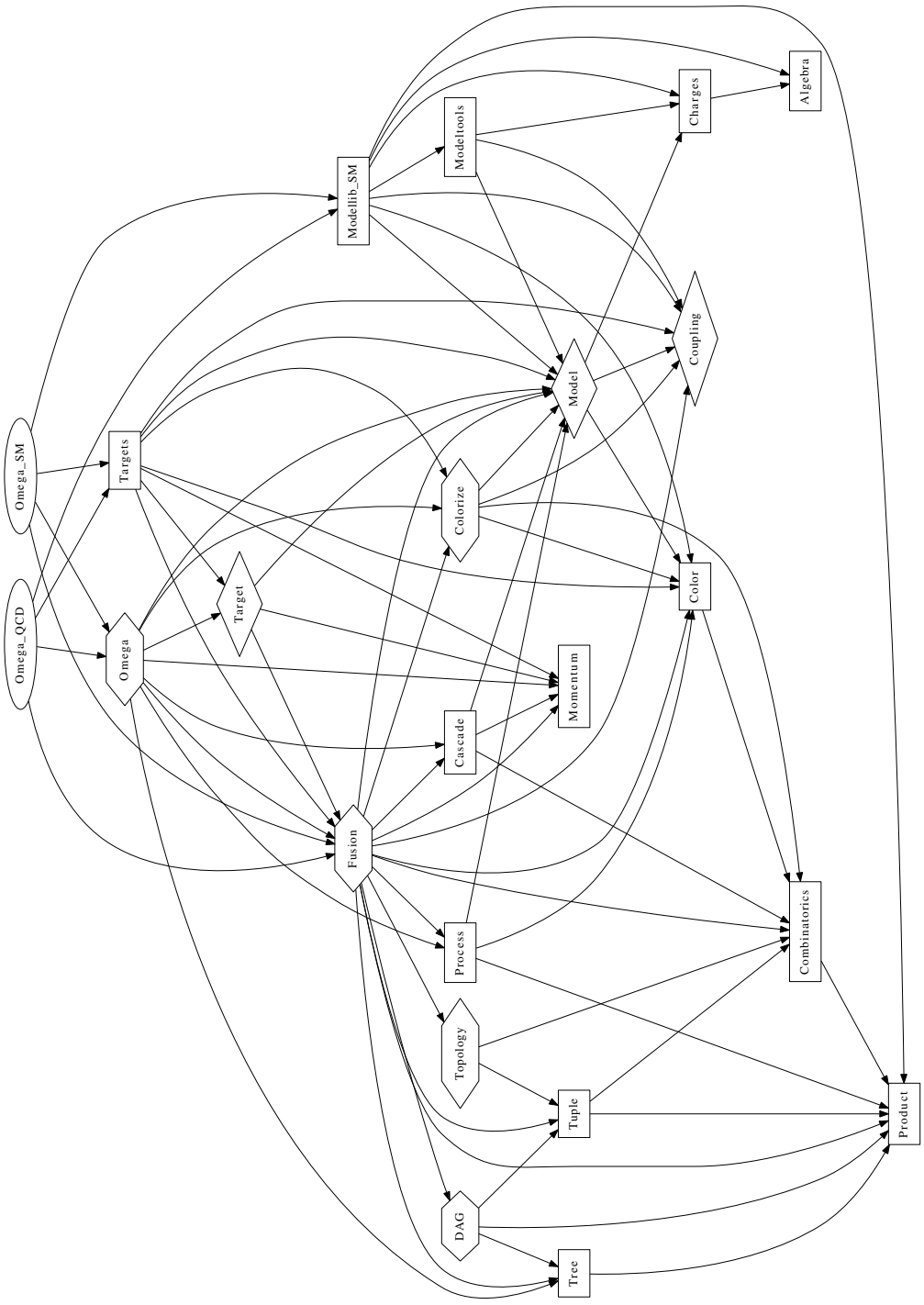


Figure 1: Major module dependencies in O'Mega. The diamond shaped nodes denote abstract signatures defining functor domains and co-domains. The hexagons denote functors and rectangular boxes denote modules, while oval boxes stand for example applications.

process	Diagrams		O'Mega	
	#	vertices	#prop.	vertices
$e^+e^- \to e^+\bar{\nu}_e d\bar{u}$	20	80	14	45
$e^+e^- \to e^+\bar{\nu}_e d\bar{u}\gamma$	146	730	36	157
$e^+e^- \to e^+\bar{\nu}_e d\bar{u}\gamma\gamma$	1256	7536	80	462
$e^+e^- \to e^+\bar{\nu}_e d\bar{u}\gamma\gamma\gamma$	12420	86940	168	1343
$e^+e^- \to e^+\bar{\nu}_e d\bar{u}\gamma\gamma\gamma\gamma$	138816	1110528	344	3933

Table 1: Radiative corrections to four fermion production: comparison of the computational complexity of scattering amplitudes obtained from Feynman diagrams and from O'Mega. (The counts correspond to the full Standard Model—sans light fermion Yukawa couplings—in unitarity gauge with quartic couplings emulated by cubic couplings of non-propagating auxiliary fields.)

orientation and the determination of the relative signs of interfering amplitudes is not trivial. However, the Feynman rules for Majorana fermions and fermion number violating interactions proposed in [6] have been implemented in O'Mega [4] in analogy to the Feynman rules for Dirac fermions and both methods are available. Numerical comparisons of amplitudes for Dirac fermions calculated both ways show agreement at a small multiple of the machine precision.

As mentioned above, the compilers for the target programming language are the slowest step in the generation of executable code. On the other hand, the execution speed of the code is limited by non-trivial vertex evaluations for vectors and spinors, which need $O(10)$ complex multiplications. Therefore, an *O'Mega Virtual Machine* can challenge native code and avoid compilations.

6 Results

6.1 Examples

Tables 1 and 2 show the reduction in computational complexity for some important processes at a e^+e^--linear collider including radiative corrections. Using the asymmetric keystones can reduce the number of vertices by some 10 to 20 percent relative to the quoted numbers for symmetric keystones.

process	Diagrams		O'Mega	
	#	vertices	#prop.	vertices
$e^+e^- \to e^+\bar{\nu}_e d\bar{u}b\bar{b}$	472	2832	49	232
$e^+e^- \to e^+\bar{\nu}_e d\bar{u}b\bar{b}\gamma$	4956	34692	108	722
$e^+e^- \to e^+\bar{\nu}_e d\bar{u}b\bar{b}\gamma\gamma$	58340	466720	226	2212

Table 2: Radiative corrections to six fermion production: comparison of the computational complexity of scattering amplitudes obtained from Feynman diagrams and from O'Mega. (The counts correspond to the full Standard Model—sans light fermion Yukawa couplings—in unitarity gauge with quartic couplings emulated by cubic couplings of non-propagating auxiliary fields.)

6.2 Comparisons

HELAC's [3] diagnostics report more vertices than O'Mega for identical amplitudes. This ranges from comparable numbers for Standard Model processes with many different flavors to an increase by 50 percent for processes with many identical flavors. Empirically, O'Mega's straight line code is twice as fast as HELAC's DO-loops for identical optimizing Fortran compilers (not counting HELAC's initialization phase). Together this results in an improved performance by a factor of two to three.

The numerical efficiency of O'Mega's Fortran runtime library is empirically identical to HELAS [8]. Therefore, O'Mega's performance can directly be compared to MADGRAPH's [9] by comparing the number of vertices. For $2 \to 5$-processes in the Standard Model, O'Mega's advantage in performance is about a factor of two and grows from there.

The results have been compared with MADGRAPH [9] for many Standard Model processes and numerical agreement at the level of 10^{-11} has been found with double precision floating point arithmetic.

6.3 Applications

O'Mega generated amplitudes are used in the multi-purpose event generator generator WHIZARD [14]. The first complete experimental study of vector boson scattering in six fermion production for linear collider physics [15] was facilitated by O'Mega and WHIZARD.

Acknowledgments

We thank Wolfgang Kilian for providing the WHIZARD Monte Carlo generator environment that turns our numbers into real events with unit weight. Thanks to the ECFA/DESY Linear Collider workshops and their participants for providing a showcase. Parts of this research were supported by Bundesministerium für Bildung und Forschung (Germany), Deutsche Forschungsgemeinschaft and the Helmholtz Alliance *Physics at the Terascale*.

Finally, thanks to the `OCaml` team at INRIA for the lean and mean implementation of a programming language that does not insult the programmer's intelligence.

References

[1] T. Ohl, *O'Mega: An Optimizing Matrix Element Generator*, Proceedings of the *Workshop on Advanced Computing and Analysis Technics in Physics Research,* Fermilab, October 2000, IKDA 2000/30, [arXiv:hep-ph/0011243]; T. Ohl, *O'Mega & WHIZARD: Monte Carlo Event Generator Generation For Future Colliders*, Proceedings of the *Workshop on Physics and Experimentation with Future Linear e^+e^--Colliders (LCWS2000),* Fermilab, October 2000, IKDA 2000/31, [arXiv:hep-ph/0011287].

[2] F. Caravaglios, M. Moretti, Phys. Lett. **B358** (1995) 332, [arXiv:hep-ph/9507237]; Z. Phys. **C74** (1997) 291, [arXiv:hep-ph/9604316].

[3] A. Kanaki, C. G. Papadopoulos, Comput. Phys. Commun. **132** (2000) 306, [arXiv:hep-ph/0002082].

[4] W. Kilian, T. Ohl, J. Reuter, *O'Mega: An Optimizing Matrix Element Generator. II: Amplitudes With Color Flow and Without Fermion Number Conservation*, to be published.

[5] W. Kilian, T. Ohl, J. Reuter, C. Speckner, *QCD in the Color-Flow Representation*, to be published.

[6] A. Denner, H. Eck, O. Hahn, J. Küblbeck, Phys. Lett. **B291** (1992) 278; Nucl. Phys. **B387** (1992) 467.

[7] F. A. Berends, W. T. Giele, Nucl. Phys. **B306** (1988) 759.

[8] H. Murayama, I. Watanabe, K. Hagiwara, *HELAS: HELicity amplitude subroutines for Feynman diagram evaluations*, KEK Report 91-11.

[9] T. Stelzer, W. F. Long, Comput. Phys. Commun. **81** (1994) 357, [arXiv:hep-ph/9401258].

[10] T. Gleisberg, S. Hoeche, JHEP **0812** (2008) 039, [arXiv:0808.3674].

[11] Chris Okasaki, *Purely Functional Data Structures*, Cambridge University Press, 1998.

[12] Xavier Leroy et al., *The `OCaml` System, Release 3.12, Documentation and User's Manual*, Technical Report, INRIA, 2011, `http://pauillac.inria.fr/ocaml/`.

[13] K. Hagiwara *et al.*, Phys. Rev. **D73**, 055005 (2006), [arXiv:hep-ph/0512260].

[14] W. Kilian, *WHIZARD 1.0: A generic Monte-Carlo integration and event generation package for multi-particle processes*, LC-TOOL-2001-039; W. Kilian, T. Ohl, J. Reuter, *WHIZARD: Simulating Multi-Particle Processes at LHC and ILC*, [arXiv:0708.4233].

[15] R. Chierici, S. Rosati, M. Kobel, *Strong Electroweak Symmetry Breaking Signals in* WW *Scattering at TESLA*, LC-PHSM-2001-038.

[16] U. Baur, D. Zeppenfeld, Phys. Rev. Lett. **75** (1995) 1002, [arXiv:hep-ph/9503344].

[17] N. D. Christensen, C. Duhr, Comput. Phys. Commun. **180** (2009) 1614, [arXiv:0806.4194]; N. D. Christensen, C. Duhr, B. Fuks, J. Reuter, C. Speckner, *Exploring the golden channel for HEIDI models using an interface between WHIZARD and FeynRules*, [arXiv:1010.3251].

[18] F. Staub, *Sarah*, [arXiv:0806.0538].

A Installing O'Mega

A.1 Sources

O'Mega is Free Software. The sources can be obtained from `http://www.hepforge.org/downloads/whizard`. Standalone sources are distributed as tarballs `omega-2.`n`.`m`.tar.gz`. They are also contained in the subdirectory `whizard-2.`n`.`m`/src/omega` of the corresponding WHIZARD tarballs `whizard-2.`n`.`m`.tar.gz` with identical version number.

The command

```
zcat omega-2.n.m.tar.gz | tar xf -
```

will unpack the source code, build environment, test suites and documentation to the directory `omega-2.n.m`. Interesting subdirectories are

`src` contains the unabridged and uncensored sources of O'Mega, including comments,

`share/doc` contains LaTeX sources of user documentation,

`tests` contains regression tests, which can be run by `make check` after building O'Mega.

A.2 Prerequisites

A.2.1 `OCaml`

O'Mega needs version 3.10 or higher, which is part of most Linux distributions. You can also get it from `http://caml.inria.fr/`. If necessary, building `OCaml` from source is straightforward (just follow the instructions in the file `INSTALL` in the toplevel directory, the defaults are almost always sufficient) and takes a few minutes minutes on a modern desktop system. The native code compiler is available for most modern systems (cf. the file `README` in the toplevel directory) and should be used instead of the byte code compiler.

A.2.2 Fortran Compiler

Not required for compiling or running O'Mega, but Fortran is currently the only fully supported target language.

Code generated by O'Mega is known to be compiled correctly with recent versions of the GNU Fortran compiler `gfortran` (preferably version 4.5.0 or later, which is required by WHIZARD 2.x) and the NAG Fortran compiler.

A.3 Building O'Mega

O'Mega uses the GNU autotools. Configuration is performed automatically by testing some system features with the command[1]

```
$ ./configure
```

[1] NB: `configure` keeps its state in `config.cache`. If you want to reconfigure after adding new libraries to your system, you should remove `config.cache` before rerunning `configure`.

See

```
$ ./configure --help
```

for additional options.

B Running O'Mega

O'Mega is a simple application that takes parameters from the command-line and writes results to the standard output device (diagnostics go to the standard error device). E. g., the UNIX commandline

```
$ omega_SM.opt -scatter "e+ e- -> e+ nue ubar d" > cc20_amplitude.f95
```

will cause O'Mega to write a Fortran module containing the Standard Model tree level scattering amplitude for $e^+e^- \to e^+\nu_e\bar{u}d$ to the file `cc20_amplitude.f95`. Particles can be combined with colons. E. g.,

```
$ omega_SM.opt -scatter "ubar:u:dbar:d ubar:u:dbar:d -> e+:mu+ e-:mu-"
```

will cause O'Mega to write a Fortran module containing the Standard Model tree level parton scattering amplitudes for all Drell-Yan processes to the standard output.

A synopsis of the available options, in particular the particle names, can be requested with the option `--help`. A partial list will look like

```
$ omega_SM.opt -help
usage: ./bin/omega_QCD.opt [options]\
 [e-|nue|u|d|e+|nuebar|ubar|dbar|mu-|numu|c|s|mu+|numubar|\
  cbar|sbar|tau-|nutau|t|b|tau+|nutaubar|tbar|bbar|\
  A|Z|W+|W-|gl|H|phi+|phi-|phi0]
  -scatter in1 in2 -> out1 out2 ...
  -scatter_file in1 in2 -> out1 out2 ...
  -decay in -> out1 out2 ...
  -decay_file in -> out1 out2 ...
  -cascade select diagrams
  -help  Display this list of options
```

B.1 Processes

More than one process can be computed in the same module, as long as the number of incoming and outgoing particles match. In particular, scatterings can not be mixed with decays. Also, the number of helicity states of the particles must match, i. e. massive vector bosons must not be combined with massless vector bosons, but quarks can be combined with gluons.

`-scatter` *in_1 in_2* `->` *out_1 out_2 ... out_n*

Construct the amplitude for a $2 \to n$ particle scattering.

`-scatter_file` *file*

Read scattering process descriptions from a file. This is equivalent to giving multiple `-scatter` arguments, one for each line.

`-decay` *in* `->` *out_1 out_2 ... out_n*

Construct the amplitude for a $1 \to n$ particle decay.

`-decay_file` *file*

Read decay descriptions from a file. This is equivalent to giving multiple `-decay` arguments, one for each line.

In the process descriptions above, each particle can be specified as a set of flavors, with elements separated by `:`, i. e. $f_1{:}f_2{:}\ldots{:}f_n$. O'Mega will generate code for the multiple cartesian product of these flavor sets, ignoring combinations prohibited by charge conservation, etc. For example, `-decay "Z -> e+:mu+ e-:mu-"` expands to the combination of $Z \to e^+e^-$, $Z \to \mu^+e^-$, $Z \to e^+\mu^-$ and $Z \to \mu^+\mu^-$. However only code for $Z \to e^+e^-$ and $Z \to \mu^+\mu^-$ will be generated, since the others are prohibited by flavor conservation assumed in the SM. If two amplitudes differ only by the permutation of final state particles, only one representative is generated in order to preserve code size.

It is possible to select a subset of Feynman diagrams by requiring one or more propagators carrying sums or differences of external momenta, optionally with particular flavors

`-cascade` *restriction*

Select diagrams, e. g. for the process $e^+e^- \to e^+e^-$, the restriction '`-cascade "3 + 4 ~ Z"`' will generate code for $e^+e^- \to Z^{(*)} \to e^+e^-$.

In the general case, the propagators are specified as an unsigned sum of momenta, with '`+`' as binary combination operator. This allows to specify propagators with timelike and spacelike moments.

For each propagator, a set of flavors can be requested with the '`~`' operator, where the elements of the flavor set are combined by '`:`'. If the momentum is spacelike, both particle and antiparticle are accepted, since the distinction would depend on the reference frame. The complement of the flavor set can be selected with '`!`', as in '`-cascade "1 + 2 ~ !A:gl"`', which demands an arbitrary s-channel resonance that is neither a photon nor a gluon. If '`~`' is replaced by '`=`' or '`#`', the propagator in question will be replaced by an on-shell projector or a gaussian smearing, respectively.

Arbitrary restrictions can be combined with logical-and '`&&`' and can be grouped with parentheses '`(`' and '`)`'.

Note that, if an outgoing momentum is part of a restriction, O'Mega will never change the position of this particle, even if this forces the same amplitude to appear twice.

B.2 Diagnostics

`-warning:`

Include code that checks the supplied arguments at runtime and prints a warning in case of an error.

`-warning:a`

Check the number of input arguments (momenta and spins).

`-warning:m`

Check the values of the input momenta.

`-warning:g`

Check Ward identities for internal currents numerically.

`-error:`

`-error:a`

`-error:m`

`-error:g`

Like `-warning:` but terminates on error.

`-unphysical` n

Select unphysical polarization state $\epsilon_\mu(k) = k_\mu$ for the nth particle to test Ward identities numerically.

`-quiet`

Don't print a summary

`-summary`

Print only a summary to standard error

`-revision`

Print revision control information to standard error.

`-params`

Produce code to print the model parameters.

B.3 Fusion Options

`-fusion:progress`

Report completion of each process to the standard error stream, including an estimate of the remaining time.

`-fusion:progress_file` name

Write the progress report to the file.

`-fusion:ignore-cache`

Ignore cached lookup tables.

`-fusion:overwrite-cache`

Overwrite cached lookup tables.

B.4 Model Options

B.4.1 Standard Model

The model specific options in the standard model are concerned with the treatment of the width of unstable particles. By default, a "time like width" is used, i. e. the width is suppressed for space like momenta.

`-model:constant_width`

Use a constant width for *all* propagators, including space like.

`-model:fudged_width`

Use the "fudge factor prescription"[16] for $W^{\pm}$, t and $\bar{t}$.

`-model:custom_width` *function*

Compute a custom width as a function of the momentum and a width value.

`-model:cancel_widths`

Set all widths to zero.

These options are implemented by most other models as well.

B.5 Target Options

B.5.1 Fortran

`-target:module` *name*

Name of the generated Fortran module (default: `omega_amplitude`).

`-target:width` *n*

Maximum output line length (default: 80).

`-target:continuation` *n*

Maximum number of continuation lines (default: unlimited).

`-target:kind` *kind*

All real and complex numbers are declared with the given kind (default: `default`).

Depending on the model, the scattering amplitudes need to access parameters (coupling constants, masses, etc.). For this purpose, Fortran modules can be imported.

`-target:use` *name*

Import the specified module. This option can be given more than once.

`-target:parameter_module` *name*

Import the specified parameter module. Only the last of these options takes effect.

The scattering amplitudes generated by O'Mega can become very large and placing all code to compute them in a single function, module or even file can result in a dramatic increase in compilation time for most Fortran compilers. Therefore, O'Mega allows to split the code into smaller pieces:

`-target:single_function`

Compute the scattering amplitudes in a single monolithic function.

`-target:split_function` n

Split the scattering amplitudes into functions with approximately n expressions each (default: 10). This is the default behavior.

`-target:split_module` n

Split the scattering amplitudes into modules with approximately n expressions each (default: 10).

`-target:split_file` n

Split the scattering amplitudes into files with approximately n expressions each (default: 10).

The code generated by O'Mega can be instrumented with OpenMP directives to take advantage of multiple computing cores and symmetric multiprocessing.

`-target:openmp`

Activate OpenMP support.

The implementation of this feature is not yet mature. Depending on the size of the scattering amplitudes and the cache size of the executing processor, it can lead to almost linear speedup with the number of cores or to no speedup due to cache collisions.

B.6 Miscellaneous Options

`-o` *file*

Redirect standard output to the file.

`-initialize` *directory*

Precompute large lookup tables for this model and store them in the directory.

`-template`

Write a template for using handwritten amplitudes with WHIZARD.

C Using O'Mega's Output

The structure of the output file, the calling convention and the required libraries depends on the target language, of course.

Note that the implementation of color in O'Mega is described in detail in [4, 5]. Nevertheless we describe below, for completeness' sake, the application program interface.

C.1 Fortran

C.1.1 Libraries

The imported Fortran modules are

`kinds`

This must define `default`, which can be whatever the Fortran compiler supports. Note that single precision arithmetic is usually not adequate for the computation of complicated amplitudes with gauge cancellations.

`omega95`

Defines the vertices for Dirac spinors in the chiral representation and vectors. An early experiment with inlining all Fortran code was a failure on Linux/Intel PCs. The inlined code was huge, absolutely unreadable and only marginally faster. The bulk of the computational cost is always in the vertex evaluations, and function calls create in comparison negligible costs. This observation is system dependent, of course, and inlining might become beneficial for other architectures, after all.

`omega95_bispinors`

Is an alternative that defines the vertices for Dirac and Majorana spinors in the chiral representation and vectors using the Feynman rules of [6], as described in [4].

`omega_color`

The color factors apply to squared amplitudes and are described by a one-dimensional array of element of

```
type omega_color_factor
   integer :: cf, conjg_cf
   complex(kind=default) :: factor
end type omega_color_factor
```

where the indices signify the pair of color flow and conjugate color flow. For more details see [4].

Therefore the generated module starts with

```
module omega_amplitude
  use kinds
  use omega95
  use omega_color, OCF => omega_color_factor
  implicit none
  private
```

C.1.2 Processes

O'Mega can generate code for the scattering amplitudes of more than one process. It will, however generate only the code for allowed flavor combinations. Therefore, an application must be able to inquire the available processes.

All processes will have the *same* number of incoming and outgoing particles

```
  pure function number_particles_in () result (n)
    integer :: n
  end function number_particles_in
  pure function number_particles_out () result (n)
    integer :: n
  end function number_particles_out
```

O'Mega will only generate code for flavor combinations with non-vanishing scattering amplitudes and will suppress code for amplitudes that differ only by a permutation of final state particles. The available flavor combinations can be inspected with

```
pure function number_flavor_states () result (n)
  integer :: n
end function number_flavor_states
pure subroutine flavor_states (f)
  integer, dimension(:,:), intent(out) :: f
end subroutine flavor_states
```

where the table `f` is interpreted as

`f(i,j)` contains the PDG code of the `i`th particle in the `j`th flavor combination

and has the shape

$$\begin{aligned} \texttt{size(f,dim=1)} &= N(\text{in}) + N(\text{out}) \\ \texttt{size(f,dim=2)} &= N(\text{flavor states})\,. \end{aligned}$$

Similarly, the possible combinations of helicities can be inspected with

```
pure function number_spin_states () result (n)
  integer :: n
end function number_spin_states
pure subroutine spin_states (h)
  integer, dimension(:,:), intent(out) :: h
end subroutine spin_states
```

where the table `h` is interpreted as

`h(i,j)` contains the helicity of the `i`th particle in the `j`th helicity combination (for particles with half integer spins, the helicities are doubled)

and has the shape

$$\begin{aligned} \texttt{size(h,dim=1)} &= N(\text{in}) + N(\text{out}) \\ \texttt{size(h,dim=2)} &= N(\text{spin states})\,. \end{aligned}$$

C.1.3 Color

Color is described by color flows. In the present version, only particles in the fundamental, anti-fundamental and adjoint representation of SU(N) are supported and at most two color lines can end at an external particle [4]. This will be generalized [5] in future versions and therefore it must be possible to inquire the number of color indices together with the number of possible color flows

```
pure function number_color_indices () result (n)
  integer :: n
end function number_color_indices
pure function number_color_flows () result (n)
  integer :: n
end function number_color_flows
pure subroutine color_flows (c, g)
  integer, dimension(:,:,:), intent(out) :: c
  logical, dimension(:,:), intent(out) :: g
end subroutine color_flows
```

The tables `c` and `g` are interpreted as

`c(:,i,j)` describes the color lines ending at the `i`th particle in the `j`th color flow

`g(i,j)` is `.true.` iff the `i`th particle in the `j`th color flow is an unphysical "U(1) ghost" [4]

and have the shapes

$$\begin{aligned}
\texttt{size(c,dim=1)} &= N(\text{color indices}) \\
\texttt{size(c,dim=2)} &= \texttt{size(g,dim=1)} = N(\text{in}) + N(\text{out}) \\
\texttt{size(c,dim=3)} &= \texttt{size(g,dim=2)} = N(\text{color flows})
\end{aligned}$$

The different color factors can be inquired with

```
pure function number_color_factors () result (n)
  integer :: n
end function number_color_factors
pure subroutine color_factors (cf)
  type(omega_color_factor), dimension(:), intent(out) :: cf
end subroutine color_factors
```

C.1.4 Amplitudes

The computation of the amplitudes is divided in two steps. First, the application program must call

```
subroutine new_event (p)
  real(kind=default), dimension(0:3,*), intent(in) :: p
  logical :: mask_dirty
  integer :: hel
end subroutine new_event
```

which will compute the scattering amplitudes for all combinations of flavors, helicities and color flows for the given set of momenta. This routine is computationally expensive. Subsequently, the application can inquire the individual amplitudes for the *same* momenta with

```
pure function get_amplitude (flv, hel, col) result (amp_result)
  complex(kind=default) :: amp_result
  integer, intent(in) :: flv, hel, col
end function get_amplitude
```

which corresponds to a simple table lookup. If only the color summed amplitude is required, the convenience function

```
pure function color_sum (flv, hel) result (amp2)
  integer, intent(in) :: flv, hel
  real(kind=default) :: amp2
end function color_sum
```

can be used after calling `new_event (p)`.

O'Mega makes no assumptions about the values of the fermion masses in relation to the kinematical invariants. Therefore it can not determine helicity selection rules analytically. Nevertheless, one can call

```
subroutine reset_helicity_selection (threshold, cutoff)
  real(kind=default), intent(in) :: threshold
  integer, intent(in) :: cutoff
end subroutine reset_helicity_selection
```

to activate a useful heuristic: for the following $N =$ `cutoff` calls to `new_event` the scattering amplitudes for each helicity will be compared to the average. Helicities for which this ratio never exceeds `threshold * epsilon()` will be assumed to be forbidden. The function

```
  pure function is_allowed (flv, hel, col) result (yorn)
    logical :: yorn
    integer, intent(in) :: flv, hel, col
  end function is_allowed
```

can be used the test if the given combination is allowed. The combination of flavor and color is determined analytically, while the helicity is treated with the heuristic described above.

C.1.5 Technical

```
  pure function openmp_supported () result (status)
    logical :: status
  end function openmp_supported


end module omega_amplitude
```

C.2 C++

A target for the C++ programming language does not exist yet. We are open for suggestions from C++ experts in HEP on useful calling conventions and support libraries that blend well with the HEP environments based on these languages.

D Extending O'Mega

D.1 Adding A New Physics Model

Adding a new model requires to write some OCaml code. This task can most easily be performed using the interfaces of the Mathematica [2] packages FeynRules [17] and SARAH [18].

However, implementing a new model from scratch is not very difficult either. An inspection of src/modellib_SM.ml shows that all that is required are some tables of Feynman rules that can easily be written by copying and modifying an existing example, after consulting with src/coupling.mli. In src/modellib_BSM.ml there is a model Template which is a verbatim copy of the SM. This could serve as a starting point for modifications. Having the full power of OCaml at one's disposal is very helpful for avoiding needless repetition.

[2] Mathematica is a registered trademark of Wolfram Research Ltd.

D.2 Adding A New Target Language

This requires to write `OCaml` code, which is a straightforward translator of an abstract syntax tree to linear code. In addition, a suitable library for vertex expressions must be implemented.