KA–TP–23–2000
hep-ph/0012260

Generating Feynman Diagrams and Amplitudes with *FeynArts* 3

Thomas Hahn
Institut für Theoretische Physik
Universität Karlsruhe
D–76128 Karlsruhe, Germany

December 20, 2000

Abstract

This paper describes the *Mathematica* package *FeynArts* used for the generation and visualization of Feynman diagrams and amplitudes. The main features of version 3 are: generation of diagrams in three levels, user-definable model files, support for supersymmetric models, and publication-quality Feynman diagrams in PostScript or LaTeX.

PACS numbers: 02.70.–c, 07.05.Bx, 89.80.+h.

Keywords: Feynman diagrams, Perturbation theory, Quantum field theory, Green's functions, S-matrix elements.

PROGRAM SUMMARY

Title of program: FeynArts

Catalogue identifier:

Program obtainable from: CPC Program Library, Queen's University of Belfast, N. Ireland, and `http://www.feynarts.de`

Computer for which the program is designed and others on which is has been tested:
Designed for: platforms on which *Mathematica* and Java are available
Tested on: Intel-based PCs, DEC Alpha workstations

Operating systems or monitors under which the program has been tested: Linux, Tru64 Unix

Programming language used: Mathematica, Java

Memory required to execute with typical data: 8M words

No. of bits in a word: 8

No. of processors used: 1

Has the code been vectorized or parallelized? No

No. of bytes in distributed program, including test data, etc.: $\sim$ 400 K bytes

Distribution format: gzipped tar archive

Keywords: Feynman diagrams, Perturbation theory, Quantum field theory, Green's functions, S-matrix elements.

Nature of the physical problem: Feynman-diagrammatic computations in field theory.

Method of solution: *FeynArts* works in three steps: 1) creation of the topologies, 2) insertion of fields into the topologies, 3) application of the Feynman rules to produce Feynman amplitudes. Information about the physical model, such as the Feynman rules, is provided in a so-called model file.

Typical running time: About a minute to generate all amplitudes for a one-loop, $2 \to 2$ process in the electroweak Standard Model.

Unusual features of the program: *FeynArts* can produce high-quality images of the Feynman diagrams e.g. in PostScript or LaTeX format for inclusion in publications.

Restrictions on the complexity: Currently diagrams up to three loops can be generated. Model files other than the Standard Model and QCD (and soon also the Minimal Supersymmetric Standard Model) are not contained in *FeynArts* and must be set up by the user.

LONG WRITE-UP

1 Introduction

Much as field theorists would love to abandon them in favour of a less laborious technique, Feynman diagrams [1] are unlikely to become extinct in the forseeable future. Adding to the amount of work, increasingly precise experimental data nowadays mandate calculations involving substantial numbers of Feynman diagrams. Beginning in the 1960s, people started following up on the rather obvious idea of letting the computer do all those involved calculations, and indeed one of the first computer-algebra systems, SCHOONSCHIP, was invented for precisely this purpose by Nobel-laureate Martinus Veltman.

This paper describes another part of this quest: the *Mathematica* package *FeynArts*, used for the generation and visualization of Feynman diagrams and amplitudes. It performs the first step of a field-theoretic perturbative calculation, leaving simplification and numerical evaluation of the amplitudes to other programs, e.g. [2, 3]. Programs with a similar functionality are QGRAF [4], the grc part of GRACE [5], CompHEP [6], and to a lesser extent MadGraph [7].

A bit of history: *FeynArts* started out in 1990 as a Macsyma code written by Eck and Küblbeck which could produce diagrams in the Standard Model [8], but it soon was ported to the *Mathematica* platform. In 1995, Eck and Küblbeck designed the second version to be a fully general diagram generator. To achieve this, they implemented some decisive new ideas [9], the most important one being the generation of diagrams in three levels. The program was taken up again in 1998 by Hahn who developed version 2.2. The well-designed conceptual framework was kept, but the actual code was reprogrammed almost entirely to make it more efficient and a user-friendly topology editor was added. The current third version features in particular significantly improved graphics. For example, it is now very easy to include Feynman diagrams produced by *FeynArts* in a LaTeX document.

The main features of *FeynArts* are:

- The generation of diagrams is possible in three levels: generic fields, classes of fields, or specific particles.
- The model information is contained in two special files: The *generic model file* defines the representation of the kinematical quantities like spinors or vector fields. The *classes model file* sets up the particle content and specifies the actual couplings. Since the user can create own model files, the applicability of *FeynArts* is virtually unlimited within perturbative quantum field theory.
- In addition to ordinary diagrams, *FeynArts* can generate counter-term diagrams and diagrams with placeholders for one-particle irreducible vertex functions (skeleton diagrams).
- *FeynArts* employs the so-called "flipping-rule" algorithm [10] to concatenate fermion chains. This algorithm is unique in that it works also for Majorana fermions and for the fermion-number-violating couplings they entail and hence allows supersymmetric models to be implemented.
- Restrictions of the type "field X is not allowed in loops" can be applied. This is necessary e.g. for the background-field formulation of a field theory.
- Vertices of arbitrary adjacency[1], required for effective theories, are allowed.
- Mixing propagators, such as appear in non-R_ξ-gauges, are supported.
- *FeynArts* produces publication-quality Feynman diagrams in PostScript or LaTeX in a format that allows easy customization.

[1]The adjacency of a vertex is the number of lines that join at the vertex.

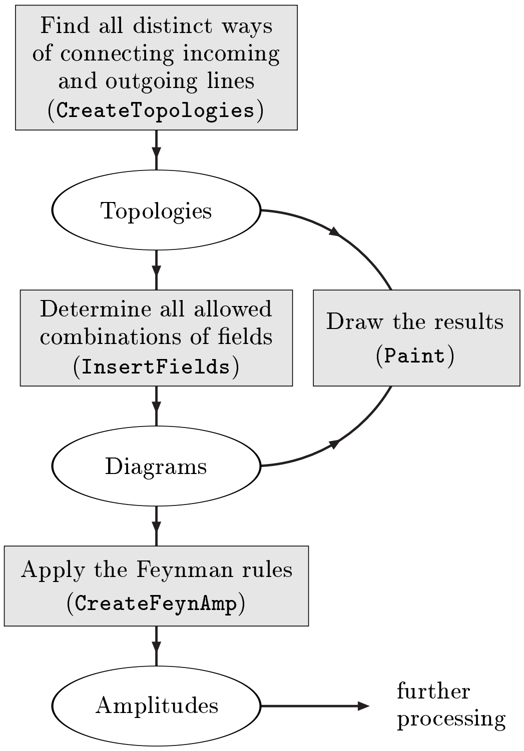


Figure 1: Flowchart for the generation of Feynman amplitudes with *FeynArts*.

These features have been introduced in version 2 and some of them have received considerable improvements in version 3. The user interface, on the other hand, has through the versions suffered only minor and mostly backward-compatible changes, and the major functions can still be used in essentially the same way as in version 1.

This paper is divided into two parts: Sect. 2 gives a brief survey of the main functions from a user's perspective. The concepts of the computer-algebraic generation of Feynman diagrams and amplitudes and their implementation in *FeynArts* are discussed in Sect. 3.

2 Using *FeynArts*

FeynArts works in the three basic steps sketched in Fig. 1.

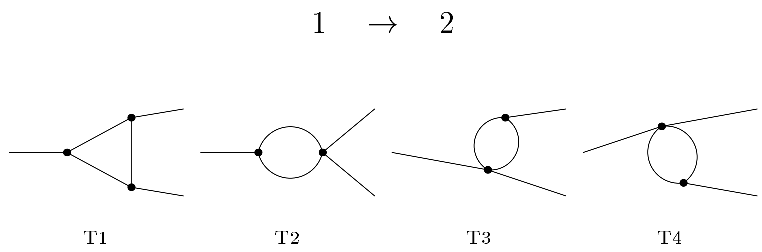
The first step is to create all different topologies for a given number of loops and external legs. The following call to `CreateTopologies` creates for example all one-particle-irreducible (1PI) one-loop topologies for a $1 \to 2$ process. This is done by generating all one-loop $1 \to 2$ topologies and then excluding the reducible ones:

```
top = CreateTopologies[1, 1 -> 2, ExcludeTopologies -> Internal]
```

The output of `CreateTopologies` is an internal data structure called a `TopologyList`. As an example, the first topology in the `TopologyList` just created is shown here:

```
Topology[1][ Propagator[Incoming][Vertex[1][1], Vertex[3][4]],
             Propagator[Outgoing][Vertex[1][2], Vertex[3][5]],
             Propagator[Outgoing][Vertex[1][3], Vertex[3][6]],
             Propagator[Loop[1]][Vertex[3][4], Vertex[3][5]],
             Propagator[Loop[1]][Vertex[3][4], Vertex[3][6]],
             Propagator[Loop[1]][Vertex[3][5], Vertex[3][6]] ]
```

A much nicer way to visualize the `TopologyList` in `top` is to paint it with `Paint[top]`.

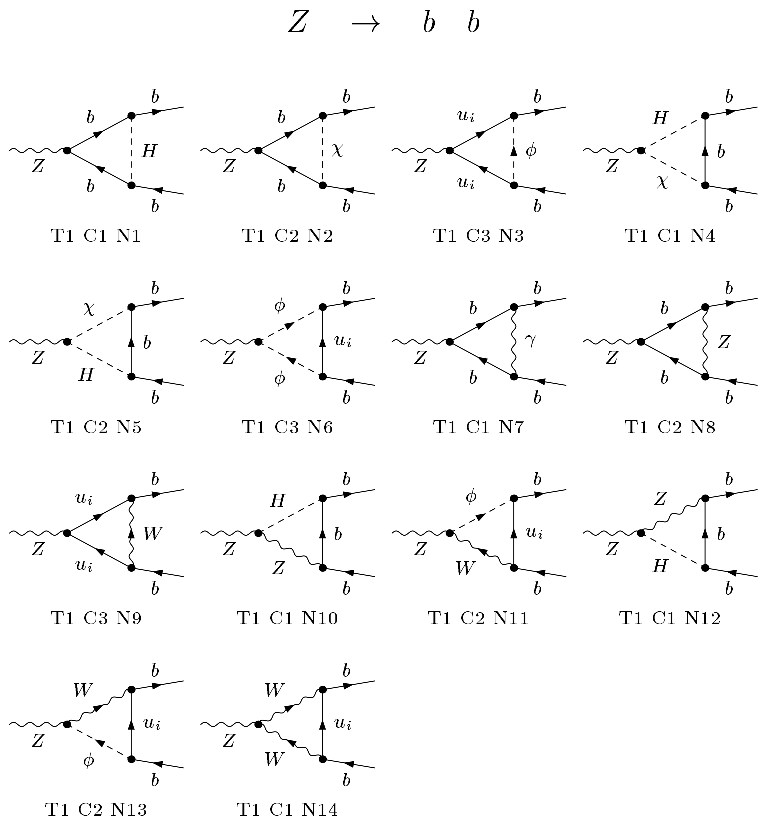


In the second step, the actual particles in the model have to be distributed over the topologies in all allowed ways. For example, the diagrams for $Z \to b\bar{b}$ are produced with

```
ins = InsertFields[top, V[2] -> {F[4,{3}], -F[4,{3}]},
        Model -> SM, InsertionLevel -> {Classes}]
```

where `F[4,{3}]` is the b-quark, `-F[4,{3}]` its antiparticle, and `V[2]` the Z boson. The model information is taken from the file `SM.mod`. The insertion level tells `InsertFields` how detailed each inserted field should be specified: `{Classes}` means that classes of fields should not be expanded. In the Standard Model, for example, the fermions are arranged in classes, so there will be only one diagram for e.g. an up-type quark u_i instead of three for its members $u_1 = u$, $u_2 = c$, $u_3 = t$ (see Sect. 3.2 for more information on field levels).

The output of `InsertFields` is again a `TopologyList` which is now supplemented with the field information. Its printed form is lengthy and looks rather unappealing, but it can be drawn with `Paint[ins]` which results in



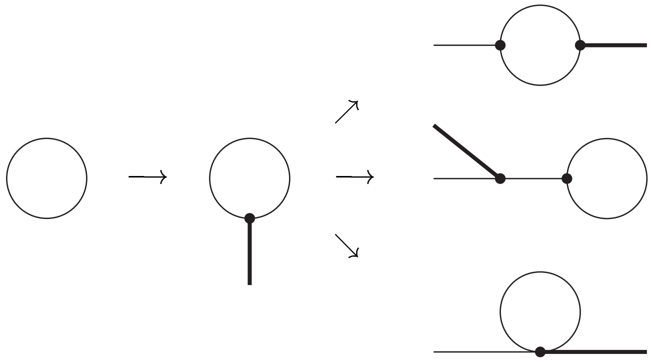


Figure 2: The algorithm of `CreateTopologies`, shown for a one-loop starting topology to which two external legs are added. The leg added in each step is drawn with a thick line.

The fields, their propagators, and their couplings are defined in a special file, the model file, which the user can supply or modify. The following model files are included in *FeynArts*: the electroweak Standard Model (`SM.mod`) [11], the same including QCD (`SMQCD.mod`), and in the background-field formulation (`SMbgf.mod`) [12]. These model files all include the full set of one-loop counter terms. A model file for the Minimal Supersymmetric Standard Model (MSSM) will be available soon.

The graphics output of `Paint` can be saved in a file with the standard *Mathematica* functions `Display` and `Export`, e.g. the following two lines draw the diagrams contained in the variable `ins` and save them in a PostScript file:

```
diags = Paint[ins];
Display["diags.ps", diags]
```

Finally, the analytic expressions for the diagrams are obtained by

```
amp = CreateFeynAmp[ins]
```

The output of `CreateFeynAmp` requires a detailed discussion which is deferred to Sect. 3.5.

Needless to say, many details have been omitted in this brief survey of *FeynArts*. All functions and their options are however fully documented in the *FeynArts* manual which is included in the *FeynArts* package (see Sect. 4 for download information).

3 Concepts of diagram generation

3.1 Generation of topologies

`CreateTopologies` generates all distinct topologies for a given number of loops and external legs. This is a purely topological process with essentially no physics input. (If a model allows vertices of a degree larger than four, this has to be specified explicitly.)

The topologies are created with a recursive algorithm [8]: it takes pre-defined topologies with zero external legs, the so-called starting topologies, and successively adds legs until the desired number of external legs is reached (see Fig. 2 for an example). The starting topologies have to be entered once for every loop order, and by default, *FeynArts* knows the tree-level, one-loop, two-loop, and three-loop starting topologies. Note that this algorithm is not self-contained because it depends on external input in the form of starting topologies. The available numbers of loops are however quite sufficient for most applications in theories like the electroweak Standard Model.

`CreateTopologies` has two companion functions, `CreateCTTopologies` and `CreateVF-Topologies`, for creating counter-term topologies and topologies with placeholders for one-particle irreducible vertex functions (skeleton diagrams), respectively. They use the same algorithm as `CreateTopologies`. In contrast to the ordinary topologies, however, these topologies may contain two-vertices (e.g. for propagator-type counter terms), and the number of such vertices is not topologically restricted by the input parameters (number of loops and external legs). Therefore, one has to keep track of the total order of a diagram, i.e. loop order plus counter-term/vertex-function order. This is solved in *FeynArts* by defining additional counter-term starting topologies. For example, in addition to the two-loop starting topologies there exist starting topologies with one loop and first-order counter terms, and tree-level starting topologies with one second-order or two first-order counter terms.

Finally, after generating all possible topologies, the topologically equivalent ones have to be weeded out and the symmetry factors of the remaining ones adjusted accordingly. To this end, the topologies are sorted into some canonical order and then compared. However, this simple procedure may fail to detect equivalent graphs if there are groups of vertices whose permutation does not change the graph. In that case, the indices of these symmetrical vertices (which are especially tagged in the starting topologies for this purpose) have to be permuted to give all topologically equivalent versions. It is this "power set" of each topology that is actually compared. While this sounds like an uncomfortably slow procedure, the actual performance is not so bad. For example, a modern PC takes about 10 seconds to generate all 2214 two-loop $2 \to 2$ topologies.

The symmetry factor of a topology is determined in the following way: the starting topology carries an integer s, the inverse of its symmetry factor. The starting topology always has the highest s because adding legs can only diminish its symmetry. After adding each leg, the topologically equivalent versions are gathered in the result and replaced by one representative whose s is divided by the number of versions found.

3.2 Field levels

An important feature of *FeynArts* is that it distinguishes three levels of fields:

– *Generic level*, e.g. the fermion `F`,
– *Classes level*, e.g. the down-type quark `F[4]`,
– *Particles level*, e.g. the b-quark `F[4, {3}]`.

This is a quite natural concept in field theory (compare e.g. the Feynman rules in [11]) and has three enormous benefits in practical calculations.

The kinematical structure of a coupling is fixed once the generic fields are specified. For example, all fermion–fermion–scalar couplings are of the form

$$C(F, F, S) = G_- \omega_- + G_+ \omega_+ \equiv \vec{G} \cdot \begin{pmatrix} \omega_- \\ \omega_+ \end{pmatrix} \tag{1}$$

where $\omega_\pm = (1 \pm \gamma_5)/2$ are the chirality projectors. This means that most algebraic simplifications, like the tensor reduction, need to be carried out on the generic-level amplitude only.

The classes level saves further CPU time because the sum over a particle index (e.g. a fermion-generation index) can be performed much more efficiently, say in a Fortran program, than the full computation of the proportionate number of diagrams at particles level.

Thirdly, since the kinematical structure of a coupling is dictated by the choice of representation of the Poincaré group (which is not often changed), it is very profitable to store the kinematical structure apart from the actual coupling constants so that it can be used with more than one model. *FeynArts* stores the kinematical structure of the couplings in a file called the *generic model file*. For example, the entry corresponding to Eq. (1) is

```
AnalyticalCoupling[ s1 F[i,mom1], s2 F[j,mom2], s3 S[k,mom3] ] ==
  G[1][s1 F[i], s2 F[j], s3 S[k]] .
    { NonCommutative[ChiralityProjector[-1]],
      NonCommutative[ChiralityProjector[+1]] }
```

Like Eq. (1), the right-hand side of this equation is the dot product of an as yet unspecified vector of coupling constants $\vec{G}$ with the kinematical vector (ω_-, ω_+), i.e. the components of $\vec{G}$ in this case provide the prefactors of ω_- and ω_+, respectively.

The actual coupling vector $\vec{G}$ is then specified in the *classes model file*, e.g. the $\overline{\ell}_{j_1} \nu_{j_2} \phi^-$-coupling in the electroweak Standard Model is defined as

```
C[ -F[2,{j1}], F[1,{j2}], S[3] ] ==
  -I EL/(Sqrt[2] SW) Mass[F[2,{j1}]]/MW IndexDelta[j1, j2] *
  { {1, dZe1 - dSW1/SW + dMWsq1/(2 MW^2) + dMf1[2,{j1}]/Mass[F[2,{j1}]] -
       1/2 (Conjugate[dZfR1[2,{j1}]] + dZfL1[1,{j1}])},
    {0, 0} }
```

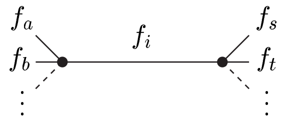
The outer braces on the right-hand side delimit the coupling vector, with components corresponding to ω_- and ω_+, while the inner braces host the orders of the coupling, e.g. `1` for the tree-level coupling and `(dZe1 - ...)` for the first-order counter term. An overall factor `(-I EL...)` is pulled out for clarity, but of course multiplies all components. Incidentally, the neutrino's left-handedness can clearly be seen from the fact that the second component of $\vec{G}$, multiplying ω_+, is zero in all orders. *FeynArts* puts almost no restrictions on what can appear in a coupling. Indeed, most of the symbols appearing in the example have been chosen by the model file's creator and have no specific meaning to *FeynArts*.

Note that as yet the class indices are specified (e.g. `F[1]`), but not the particle indices (`{j1}`). No extra model file is needed for particles level, however—the replacement of the remaining particles indices by integers is trivial enough to be performed without further input.

3.3 Insertion of fields

The computer-algebraic generation of Feynman diagrams corresponds to the distribution of fields over the topologies in such a way that the resulting diagrams possess the external fields the user has chosen and contain only couplings allowed by the model. This process is called "inserting fields into a topology" and is performed by the `InsertFields` function.

As would be expected from the level-concept of fields, the insertion of fields is a three-stage process, with functions for the insertion of generic-, classes-, and particles-level fields nested inside each other. It suffices however to describe the main insertion function which is eventually invoked at all levels. This function works as follows: it is called for each propagator and receives as input the fields coming in at either end of the propagator, $\{f_a, f_b, \dots\}$ and $\{f_s, f_t, \dots\}$, and the field running on the propagator itself, f_i:



All of this field information is specified as precisely as known at that stage of the insertion process. The insertion function then looks up which particles are allowed for f_i, given that it joins $\{f_a, f_b, \dots\}$ at the left end, and similarly for the right end. Taking the intersection of these two "allowed" lists yields the possible choices for f_i.

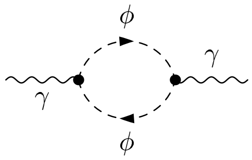
Mixing propagators introduce a slight complication: instead of the field f_i itself, *FeynArts* has to take the left and right partner of f_i for look-up at the left and right end, respectively. For example, the left and right partners of a γ–Z mixing field are γ and Z.

The look-up tables used for finding the allowed fields obviously play a very important role. They are built during the model-initialization phase and account for most of the speed of the `InsertFields` function.

3.4 Drawing Feynman diagrams

Both the bare topologies of `CreateTopologies` and the inserted diagrams of `InsertFields` can be drawn with the `Paint` function. The output of `Paint`, displayed also on screen by default, can be rendered in PostScript, LaTeX, or any other graphics format known to *Mathematica*, e.g. GIF, JPEG, PDF, etc. The most useful output formats are however `"TeX"` (LaTeX), `"PS"` (PostScript), and `"EPS"` (encapsulated PostScript).

The LaTeX format is probably the most convenient one for including the diagrams in publications. For example, the diagram



has the LaTeX representation

```
\begin{feynartspicture}(100,100)(1,1)
\FADiagram{}
\FAProp(0.,10.)(6.,10.)(0.,){/Sine}{0}
\FALabel(3.,8.93)[t]{$\gamma$}
\FAProp(20.,10.)(14.,10.)(0.,){/Sine}{0}
\FALabel(17.,11.07)[b]{$\gamma$}
\FAProp(6.,10.)(14.,10.)(0.8,){/ScalarDash}{-1}
\FALabel(10.,5.73)[t]{$\phi$}
\FAProp(6.,10.)(14.,10.)(-0.8,){/ScalarDash}{1}
\FALabel(10.,14.27)[b]{$\phi$}
\FAVert(6.,10.){0}
\FAVert(14.,10.){0}
\end{feynartspicture}
```

Such fragments can be inserted into a LaTeX document, thus eliminating external files for the figures. It is also fairly easy to change or move around diagrams with any text editor. The only requirement is to include the `feynarts.sty` style in which the LaTeX commands used in the *FeynArts* output are defined.

As of version 3, *FeynArts* possesses a custom PostScript prologue—a piece of PostScript code that explains to the PostScript interpreter how to draw propagators, vertices, and labels. The prologue makes it possible to produce LaTeX output of the form shown above and indeed, `feynarts.sty` consists mostly of the prologue. As a nice side-effect, the PostScript files generated by *FeynArts* 3 are smaller by a factor of 5 or more compared with older versions.

The shapes of the diagrams are not automatically designed by *FeynArts*. That is a matter of human taste and too complicated for a computer program. Each time a diagram is drawn, *FeynArts* looks up its shape in a database, and if no shape is found, calls up the topology editor in which the user can arrange the vertices, propagators, and labels with the mouse. The topology editor is the only part of *FeynArts* not written in *Mathematica* but in Java.

3.5 Applying the Feynman rules

Once the possible combinations of fields have been determined by `InsertFields`, the Feynman rules must be applied to produce the actual amplitudes. This is done by `CreateFeynAmp`. The Feynman rules, more specifically, consist of the expressions for the propagators and vertices defined in the model files, a prefactor which includes symmetry factors and usually depends on the number of loops, and the rules for the concatenation and signs of fermion chains.

The output of `CreateFeynAmp` is intentionally very symbolic to make it easier for other programs to locate certain parts of the amplitude. For example, the amplitude resulting from the photon self-energy diagram painted in the last section is

```
FeynAmp[
  GraphID[Topology == 1, Generic == 1], ................................ ①
  Integral[q1], ........................................................ ②
  (I/(32 Pi^4) RelativeCF ............................................... ③
   FeynAmpDenominator[1/(q1^2 - Mass[S[Gen3]]^2),
     1/((-p1 + q1)^2 - Mass[S[Gen4]]^2)] ................................ ④
   (p1 - 2 q1)[Lor1] (-p1 + 2 q1)[Lor2] ................................. ⑤
   ep[V[1], p1, Lor1] ep*[V[1], k1, Lor2] ............................... ⑥
   G(0)SSV[(Mom[1] - Mom[2])[KI1[3]]] ................................... ⑦
   G(0)SSV[(Mom[1] - Mom[2])[KI1[3]]]),
  {Mass[S[Gen3]], Mass[S[Gen4]], ........................................ ⑧
   G(0)SSV[(Mom[1] - Mom[2])[KI1[3]]],
   G(0)SSV[(Mom[1] - Mom[2])[KI1[3]]],
   RelativeCF} ->
  Insertions[Classes][{MW, MW, I EL, -I EL, 2}]
]
```

The `FeynAmp` function has four arguments: an identifier ①, the integration momenta ②, the generic amplitude ③–⑦, and replacement rules for transforming the generic amplitude into a classes or particles amplitude ⑧.

The generic amplitude has the following elements: a numeric pre-factor ③ (`RelativeCF` stands for RELATIVE Combinatorial Factor and is specified for each diagram by the replacement rules ⑧), the denominators of loop propagators collected in a function `FeynAmpDenominator` ④, the kinematic structure of the two scalar–scalar–vector couplings ⑤, the polarization vectors of the external photons ⑥, and the generic coupling constants of both vertices ⑦.

To turn the generic amplitude into a classes or particles amplitude, all generic objects must be replaced by their concrete values at the particular level. This replacement is specified by the rules ⑧. For example, for the first (and in this simple example only) classes diagram, `Mass[S[Gen3]]` becomes `MW`. In general, one generic amplitude will of course fan out into several derived classes or particles amplitudes, so the `Insertions` function will have several entries.

Even though this method of keeping the generic amplitude apart from the replacement rules has the advantages outlined in Sect. 3.2, it is possible to obtain the more conventional Feynman amplitudes by picking out one level (i.e. applying the replacement rules) with the function `PickLevel`.

From the appearance of polarization vectors it is clear that the sample amplitude shown above is part of an S-matrix element. Just as well it is possible to produce amplitudes for Green's functions by selecting `Truncated -> True` as an option in `CreateFeynAmp`.

3.6 Supersymmetric models

Supersymmetric theories in general contain Majorana fermions and hence fermion-number-violating couplings (e.g. quark–squark–gluino). The textbook prescription of ordering the Dirac matrices opposite to their occurrence along the arrows on fermionic lines obviously breaks down in this case since one cannot define a fermion-number flow. Put differently, Majorana-fermion lines have no arrow.

In fact, *FeynArts* has to address this problem for all fermions, not just for Majorana ones, because the amplitude is constructed at generic level and generic fermion fields are defined to be undirected.

The implemented solution is the "flipping-rule" algorithm [10]: instead of traversing the fermion lines along the fermion-number flow imposed from the outside, *FeynArts* chooses a direction for each fermion chain. If it turns out later that, for a Dirac fermion, the chosen direction is opposite to the actual fermion flow, it "flips" the coupling, i.e. it derives the coupling appropriate for the reversed fermion flow from the known coupling. This is in fact nothing but a charge (as opposed to hermitian) conjugation of the coupling and the flipping rules, which act on elements of the Dirac algebra, are actually quite simple, e.g.

$$\gamma_\mu \omega_\pm \xrightarrow{\text{flip}} -\gamma_\mu \omega_\mp \,. \tag{2}$$

4 Availability, Requirements

The *FeynArts* package can be downloaded from `http://www.feynarts.de` and includes a comprehensive manual which explains installation and usage. *FeynArts* requires *Mathematica* 3 or above. For the topology editor, a Java VM and the J/Link package are needed, both of which can be obtained free of charge (see the instructions on the web site). *FeynArts* is an open-source program and stands under the GNU library general public license.

Acknowledgements

The development of *FeynArts* 2.2 has been supported by the Deutsche Forschungsgemeinschaft (Forschergruppe "Quantenfeldtheorie, Computeralgebra und Monte-Carlo Simulation") under contract number Ku 502/8–1. The new graphics functions of *FeynArts* 3 have been developed on a Visiting Scholar grant of Wolfram Research, Inc., and I am grateful in particular to M. Malak and L. D'Andria for discussions on how to get the most out of *Mathematica*. I thank A. Denner and W. Hollik for proofreading the manuscript.

References

[1] see e.g. M. Veltman, Diagrammatica, Cambridge University Press (1994).

[2] R. Mertig, M. Böhm, and A. Denner, *Comp. Phys. Commun.* **64** (1991) 345.

[3] T. Hahn and M. Pérez-Victoria, *Comp. Phys. Commun.* **118** (1999) 153.

[4] P. Nogueira, *J. Comput. Phys.* **105** (1993) 279.

[5] F. Yuasa et al., *Prog. Theor. Phys. Suppl.* **138** (2000) 18;
T. Kaneko, *Comp. Phys. Commun.* **92** (1995) 127.

[6] E. Boos, M. Dubinin, V. Ilyin, A. Pukhov, V. Savrin, INP MSU 94-36/358 (hep-ph/9503280);
A. Pukhov et al., hep-ph/9908288.

[7] T. Stelzer, W.F. Long, *Comp. Phys. Commun.* **81** (1994) 357.

[8] J. Küblbeck, M. Böhm, and A. Denner, *Comp. Phys. Commun.* **60** (1990) 165.

[9] H. Eck, Ph.D. thesis, University of Würzburg (1995), the PostScript file is available from `http://www.feynarts.de`.

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

[11] A. Denner, *Fortschr. Phys.* **41** (1993) 307.

[12] A. Denner, S. Dittmaier, and G. Weiglein, *Nucl. Phys.* **B440** (1995) 95.