

# Feynman Diagram Calculations with FeynArts, FormCalc, and LoopTools

---

**Thomas Hahn\***

*Max-Planck-Institut für Physik  
Föhringer Ring 6  
D-80805 Munich, Germany  
E-mail: hahn@feynarts.de*

This article describes the latest versions of the Mathematica packages FeynArts, FormCalc, and LoopTools for the generation and evaluation of one-loop diagrams.

*13th International Workshop on Advanced Computing and Analysis Techniques in Physics Research  
February 22-27, 2010  
Jaipur, India*

---

\*Speaker.

## 1. Introduction

FeynArts [1], FormCalc, and LoopTools [2] are programs for the generation and calculation of Feynman diagrams. The ‘canonical’ problem they solve is the computation of the cross-section up to one-loop order for a given scattering reaction, in a highly automated way. Mathematica as the basic programming language makes it straightforward to use in particular intermediate results in a variety of ways, for example checks of Ward identities or the extraction of Wilson coefficients. This note gives an overview of the system with emphasis on recently added features.

## 2. FeynArts

FeynArts is a Mathematica package for the generation and visualization of Feynman diagrams and amplitudes. Launched in 1991 [3], the current version 3.5 still uses almost the same syntax, though with many extensions. The generation of amplitudes is a three-step process. In the first step, the distinct topologies for a given number of loops and external legs are produced, e.g.

```
top = CreateTopologies[1, 1 → 2]
```

This is a purely geometrical/topological task and requires no physics input. The internal algorithm starts from given zero-leg topologies of the requested loop order and successively adds legs.

In the second step, the model’s particle content is read from a Model File and the fields are distributed over the topologies in all admissible ways, e.g.

```
ins = InsertFields[top, F[4, {3}] → {F[4, {2}], V[1]}]
```

Finally, the Feynman rules are applied with

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

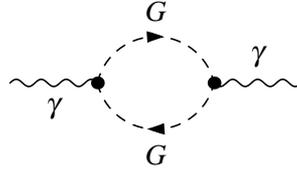
The field labelling above is the one of the default model, SM.mod, and corresponds to the decay  $b \rightarrow s\gamma$ , where  $b$  and  $s$  are the third and second members of the down-type quark class  $F[4]$ , and  $V[1]$  is the photon. This notation is part of the more general concept of field levels:

- The Generic Level determines the space–time properties of a field, e.g. a fermion  $F$ . It also fixes the kinematic properties of the couplings. For example, the FFS coupling is of the form  $G_+\omega_+ + G_-\omega_-$ , where  $\omega_\pm = (\mathbb{1} \pm \gamma_5)/2$ , with coefficients  $G_\pm$  that depend on model parameters only.
- The Classes Level specifies the particle up to ‘simple’ index substitutions, e.g. the down-type quark class  $F[4]$  (where the generation index is not yet given).
- The Particles Level spells out any indices left unspecified, e.g. the bottom quark  $F[4, \{3\}]$ .

The reason for this splitting is mainly economy: kinematic simplifications can be performed at Generic Level, where there are typically much fewer diagrams than at lower levels. Likewise, ‘trivial’ sums e.g. over fermion generations need not be written out explicitly in terms of Particles-Level Feynman diagrams.

The diagrams returned by `CreateTopologies` and `InsertFields` can be drawn with `Paint`, with output as Mathematica Graphics object, PostScript, or  $\text{\LaTeX}$ .  $\text{\LaTeX}$  code produced by `Paint` can be post-processed (e.g. ‘touched up’ for publication) with the FeynEdit editor [4].

A diagram in the output is encoded as `FeynAmp[id, loopmom, genamp, ins]`. For illustration, consider the diagram



- `id` is an identifier for bookkeeping, e.g. `GraphID[Topology == 1, Generic == 1]`,
- `loopmom` identifies the loop momenta in the form `Integral[q1]`,
- `genamp` is the generic amplitude,

$$\frac{1}{32 \text{Pi}^4} \text{RelativeCF} \textcircled{1}$$

$$\text{FeynAmpDenominator}\left[\frac{1}{q1^2 - \text{Mass}[S[\text{Gen3}]]^2}, \frac{1}{(-p1 + q1)^2 - \text{Mass}[S[\text{Gen4}]]^2}\right] \textcircled{2}$$

$$(p1 - 2q1)[\text{Lor1}] (-p1 + 2q1)[\text{Lor2}] \textcircled{3}$$

$$\text{ep}[V[1], p1, \text{Lor1}] \text{ep}^*[V[1], k1, \text{Lor2}] \textcircled{4}$$

$$G_{\text{SSV}}^{(0)}[(\text{Mom}[1] - \text{Mom}[2])[\text{KI1}[3]]] G_{\text{SSV}}^{(0)}[(\text{Mom}[1] - \text{Mom}[2])[\text{KI1}[3]]] \textcircled{5}$$

where individual items can easily be identified: prefactor  $\textcircled{1}$ , loop denominators  $\textcircled{2}$ , coupling structure  $\textcircled{3}$ , polarization vectors  $\textcircled{4}$ , coupling constants  $\textcircled{5}$ .

- `ins` is a list of rules substituting the unspecified items in the generic amplitude,

$$\{ \text{Mass}[S[\text{Gen3}]], \text{Mass}[S[\text{Gen4}]],$$

$$G_{\text{SSV}}^{(0)}[(\text{Mom}[1] - \text{Mom}[2])[\text{KI1}[3]]],$$

$$G_{\text{SSV}}^{(0)}[(\text{Mom}[1] - \text{Mom}[2])[\text{KI1}[3]]], \text{RelativeCF} \} \rightarrow$$

$$\text{Insertions}[\text{Classes}] [\{\text{MW}, \text{MW}, \text{I EL}, -\text{I EL}, 2\}]$$

## 2.1 Model Files

The Model Files are ordinary Mathematica text files loaded by FeynArts during model initialization. They supply certain objects, e.g. `M$ClassesDescription`, the list of particles, and `M$CouplingMatrices`, the list of couplings. Generic (`.gen`) and Classes (`.mod`) Model Files store the kinematic and constant part of the coupling, respectively.

FeynArts further distinguishes Basic and Partial (Add-On) Model Files. Basic Model Files, such as `SM.mod`, `MSSM.mod`, can be modified by Add-On Model Files, as in

```
InsertFields[... , Model -> {"MSSMQCD", "FV"}]
```

This loads the Basic Model File `MSSMQCD.mod` and modifies it through the Add-On Model File `FV.mod` (non-minimal flavour violation). The brace notation works similarly for Generic Model files. The Add-On Model File typically modifies (rather than overwrites) its objects.

Model Files for FeynArts can currently be generated by FeynRules [5] and LanHEP [6]. The SARAH package [7] is useful for the high-level derivation of SUSY models. FeynArts itself includes the ModelMaker tool which turns a suitably defined Lagrangian into a Model File. For further details of model construction the reader is referred to the respective manuals.

## 2.2 Linear Combination of Fields

Starting from Version 3.5, FeynArts can automatically linear-combine fields, i.e. one can specify the couplings in terms of gauge rather than mass eigenstates. For example:

```
M$ClassesDescription = { ...,
  F[11] == {...,
    Indices → {Index[Neutralino]},
    Mixture → ZNeu[Index[Neutralino],1] F[111] +
              ZNeu[Index[Neutralino],2] F[112] +
              ZNeu[Index[Neutralino],3] F[113] +
              ZNeu[Index[Neutralino],4] F[114]} }
```

M\$CouplingMatrices may now be given in terms of the gauge eigenstates F[111] ... F[114], which makes the expressions much shorter. F[111] ... F[114] drop out completely after model initialization, however, as they are not themselves listed in M\$ClassesDescription.

Higher-order mixings can be added, too:

```
M$ClassesDescription = { ...,
  S[1] == {...},
  S[2] == {...},
  S[10] == {...,
    Indices → {Index[Higgs]},
    Mixture → UHiggs[Index[Higgs],1] S[1] +
              UHiggs[Index[Higgs],2] S[2],
    InsertOnly → {External, Internal}} }
```

This time, S[10] and S[1], S[2] appear in M\$ClassesDescription and hence the coupling list contains both mixed and unmixed states, in all possible combinations. Due to the InsertOnly, S[10] is inserted only on tree-level parts of the diagram, not in loops.

## 2.3 Enhanced Diagram Selection

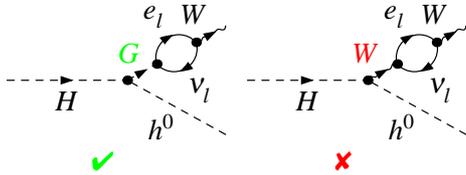
In recent FeynArts versions, many functions have been added or extended to ease diagram selection: DiagramSelect, DiagramGrouping, DiagramMap, DiagramComplement. Also new or extended are many ‘filter functions’ which simplify the construction of sophisticated filters for the selection functions above: Vertices, FieldPoints, FermionRouting, FeynAmpCases, FieldMatchQ, FieldMemberQ, FieldPointMatchQ, FieldPointMemberQ.

To pick just two examples: the selection of wave-function corrections (WFC) has become more flexible. The exclusion of WFC can be specified individually for every external leg,

```
CreateTopologies[..., ExcludeTopologies → WFCorrections[1|3]]
```

The filter function `WFCorrectionFields` returns the in- and out-fields of the self-energy constituting the WFc. It solves the problem of treating WFc with same outer particles (usually omitted) and different particles (kept unless some on-shell scheme is employed) differently, e.g.

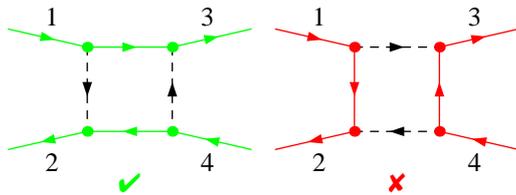
```
DiagramSelect[... , UnsameQ@@ WFCorrectionFields[##] &]
```



The new filter function `FermionRouting` can be used to select diagrams according to their fermion structure, e.g.

```
DiagramSelect[... , FermionRouting[##] === {1,3, 2,4} &]
```

selects only diagrams where external legs 1–3 and 2–4 are connected through fermion lines.



### 3. FormCalc

The output of FeynArts is not directly in a good shape for numerical evaluation. It contains uncontracted indices, unregularized loop integrals, fermion traces,  $SU(N)$  generators, etc. The symbolic expressions for the diagrams are thus first simplified algebraically with FormCalc, which performs the following steps: indices are contracted, fermion traces evaluated, open fermion chains simplified, colour structures standardized, tensor integrals reduced, abbreviations introduced.

Most of these steps are internally executed in FORM [8], a computer-algebra system whose instruction set has many adaptations especially useful in high-energy physics. The interfacing with FORM is transparent to the user, i.e. the user does not have to work with the FORM code. FormCalc thus combines the speed of FORM with the powerful instruction set of Mathematica and the latter greatly facilitates further processing of the results.

The main function is `CalcFeynAmp` which is applied to a FeynArts amplitude (the output of `CreateFeynAmp`) and combines the steps outlined above. Its output is in general a linear combination of loop integrals with prefactors that contain model parameters, kinematic variables, and abbreviations introduced by FormCalc, e.g.

```
C0i[cc0, MW2, MW2, S, MW2, MZ2, MW2] *
(-4 Alfa2 CW2 MW2/SW2 S AbbSum16 +
 32 Alfa2 CW2/SW2 S^2 AbbSum28 +
```

```

4 Alfa2 CW2/SW2 S2 AbbSum30 -
8 Alfa2 CW2/SW2 S2 AbbSum7 +
Alfa2 CW2/SW2 S (T-U) Abb1 +
8 Alfa2 CW2/SW2 S (T-U) AbbSum29 )

```

The first line represents the one-loop integral  $C_0(M_W^2, M_W^2, s, M_W^2, M_Z^2, M_W^2)$ , multiplied with a linear combination of abbreviations like Abb1 or AbbSum29 with coefficients containing kinematical invariants like the Mandelstam variables S, T, and U and model parameters such as  $\text{Alfa2} = \alpha^2$ .

### 3.1 Abbreviations

The automated introduction of abbreviations is a key concept in FormCalc. It is crucial in rendering an amplitude as compact as possible. The main effect comes from three layers of recursively defined abbreviations, introduced when the amplitude is read back from FORM, i.e. during CalcFeynAmp. For example:

$$\begin{aligned}
 \text{AbbSum29} &= \text{Abb2} + \boxed{\text{Abb22}} + \text{Abb23} + \text{Abb3} \\
 \boxed{\text{Abb22}} &= \text{Pair1} \boxed{\text{Pair3}} \text{Pair6} \\
 \boxed{\text{Pair3}} &= \text{Pair}[\text{e}[3], \text{k}[1]]
 \end{aligned}$$

Written out, this abbreviation is equivalent to

$$\begin{aligned}
 &\text{Pair}[\text{e}[1], \text{e}[2]] \text{Pair}[\text{e}[3], \text{k}[1]] \text{Pair}[\text{e}[4], \text{k}[1]] + \\
 &\text{Pair}[\text{e}[1], \text{e}[2]] \text{Pair}[\text{e}[3], \text{k}[2]] \text{Pair}[\text{e}[4], \text{k}[1]] + \\
 &\text{Pair}[\text{e}[1], \text{e}[2]] \text{Pair}[\text{e}[3], \text{k}[1]] \text{Pair}[\text{e}[4], \text{k}[2]] + \\
 &\text{Pair}[\text{e}[1], \text{e}[2]] \text{Pair}[\text{e}[3], \text{k}[2]] \text{Pair}[\text{e}[4], \text{k}[2]]
 \end{aligned}$$

In addition to these abbreviations assigned by CalcFeynAmp, FormCalc introduces another set of abbreviations for the loop integrals when generating Fortran code.

### 3.2 Categories

Both of the aforementioned types of abbreviations, the latter in particular, are costly in CPU time. It is thus key to performance that the abbreviations are grouped into different categories:

1. Abbreviations that depend on the helicities.
2. Abbreviations that depend on angular variables.
3. Abbreviations that depend only on  $\sqrt{s}$ .

Correct execution of the different categories guarantees that almost no redundant evaluations are made, e.g. in a  $2 \rightarrow 2$  process with external unpolarized fermions, statements in the innermost loop over the helicities are executed  $2^4$  times as often as those in the loop over the angle. This technique of moving invariant expressions out of the loop is known as ‘hoisting’ in computer science.

The Abbreviate function extends the advantages of the abbreviation system to arbitrary expressions. Its usage is for example:

```
abbrexp = Abbreviate[expr, 5]
```

The second argument, 5, specifies the level below which abbreviations are introduced, i.e. how much of expression is ‘abbreviated away.’ In the extreme, for a level of 1, the result is just a single symbol. Abbreviating also has the nice side effect that duplicate expressions are replaced by the same symbol. This new type of abbreviations for subexpressions has to be retrieved separately from the other ones with `Subexpr[]`.

Abbreviations and subexpressions from an earlier Mathematica session must be ‘registered’ first using `RegisterAbbr[abbr]` and `RegisterSubexpr[subexpr]`.

### 3.3 Weyl chains and Dirac chains

Amplitudes with external fermions have the form  $\mathcal{M} = \sum_{i=1}^n c_i F_i$ , where the  $F_i$  are (products of) fermion chains. The textbook recipe is to compute probabilities, e.g.  $|\mathcal{M}|^2 = \sum_{i,j=1}^n c_i^* c_j F_i^* F_j$ , and evaluate the  $F_i^* F_j$  by trace techniques:  $|\langle u | \Gamma | v \rangle|^2 = \langle u | \Gamma | v \rangle \langle v | \bar{\Gamma} | u \rangle = \text{Tr}(\Gamma | v \rangle \langle v | \bar{\Gamma} | u \rangle \langle u |)$ .

The problem with this approach is that instead of  $n$  of the  $F_i$  one needs to compute  $n^2$  of the  $F_i^* F_j$ . Since essentially  $n \sim (\text{number of vectors})!$ , this quickly becomes a limiting factor in problems involving many vectors, e.g. in multi-particle final states or polarization effects.

The solution is of course to compute the amplitude  $\mathcal{M}$  directly and this is done most conveniently in the Weyl–van der Waerden formalism [9]. The implementation of this technique in an automated program has been outlined in [10].

The `FermionChains` option of `CalcFeynAmp` determines how fermion chains are returned: `Weyl`, the default, selects Weyl chains. `Chiral` and `VA` select Dirac chains in the chiral ( $\omega_+/\omega_-$ ) and vector/axial-vector ( $\mathbb{1}/\gamma_5$ ) decomposition, respectively. The Weyl chains need not be further evaluated with `HelicityME`, which applies the trace technique.

As numerical calculations are done mostly using Weyl chains therefore, there has been a paradigm shift for Dirac chains to make them better suited for analytical purposes, e.g. the extraction of Wilson coefficients.

The `FermionOrder` option of `CalcFeynAmp` implements Fierz reordering, allowing the user to force Dirac chains into almost any desired order. `FermionOrder` does not only allow for explicit orderings, but can take the `Colour` option, too, in which case the spinor indices are brought into the same order as the colour indices, a convention commonly found in the literature.

Antisymmetrized Dirac chains are chosen with the `Antisymmetrize` option of `CalcFeynAmp`. They are indicated by a negative identifier, e.g. `DiracChain[-1,  $\mu$ ,  $\nu$ ] =  $\sigma_{\mu\nu}$` .

### 3.4 Alternate Link between FORM and Mathematica

FORM is renowned for being able to handle very large expressions. To produce (pre)simplified expressions, however, terms have to be wrapped in functions, to avoid immediate expansion. The number of terms in a function is unfortunately rather limited in FORM: on 32-bit systems to 32767. While `FormCalc` gets more sophisticated in pre-simplifying amplitudes, users want to compute larger amplitudes and have thus seen many ‘overflow’ messages from FORM recently.

In `FormCalc` Versions 6 and up, the pre-simplified generic amplitude is intermediately sent to Mathematica for introducing abbreviations through FORM’s external channels [11]. This results in a significant reduction in size of intermediate expressions.

The following example is taken from the tree-level  $uu \rightarrow gg$  amplitude. The expression passed from FORM to Mathematica is

```
+Den[U,MU2]*(
-8*SUNSum[Col5,3]*SUNT[Glu3,Col5,Col2]*SUNT[Glu4,Col1,Col5]*mul[Alfas*Pi]*
abb[fme[WeylChain[DottedSpinor[k1,MU,-1],6,Spinor[k2,MU,1]]]*ec3.ec4
-1/2*fme[WeylChain[DottedSpinor[k1,MU,-1],6,ec3,ec4,Spinor[k2,MU,1]]]
+fme[WeylChain[DottedSpinor[k1,MU,-1],7,Spinor[k2,MU,1]]]*ec3.ec4
-1/2*fme[WeylChain[DottedSpinor[k1,MU,-1],7,ec3,ec4,Spinor[k2,MU,1]]]*MU
-4*SUNSum[Col5,3]*SUNT[Glu3,Col5,Col2]*SUNT[Glu4,Col1,Col5]*mul[Alfas*Pi]*
abb[fme[WeylChain[DottedSpinor[k1,MU,-1],6,ec3,ec4,k3,Spinor[k2,MU,1]]]
-2*fme[WeylChain[DottedSpinor[k1,MU,-1],6,ec4,Spinor[k2,MU,1]]]*ec3.k2
-2*fme[WeylChain[DottedSpinor[k1,MU,-1],6,k3,Spinor[k2,MU,1]]]*ec3.ec4
+fme[WeylChain[DottedSpinor[k1,MU,-1],7,ec3,ec4,k3,Spinor[k2,MU,1]]]
-2*fme[WeylChain[DottedSpinor[k1,MU,-1],7,ec4,Spinor[k2,MU,1]]]*ec3.k2
-2*fme[WeylChain[DottedSpinor[k1,MU,-1],7,k3,Spinor[k2,MU,1]]]*ec3.ec4]
+8*SUNSum[Col5,3]*SUNT[Glu3,Col5,Col2]*SUNT[Glu4,Col1,Col5]*mul[Alfas*MU*Pi]*
abb[fme[WeylChain[DottedSpinor[k1,MU,-1],6,Spinor[k2,MU,1]]]*ec3.ec4
-1/2*fme[WeylChain[DottedSpinor[k1,MU,-1],6,ec3,ec4,Spinor[k2,MU,1]]]
+fme[WeylChain[DottedSpinor[k1,MU,-1],7,Spinor[k2,MU,1]]]*ec3.ec4
-1/2*fme[WeylChain[DottedSpinor[k1,MU,-1],7,ec3,ec4,Spinor[k2,MU,1]]] )
```

while the abbreviated expression returning from Mathematica is just

```
-4*Den(U,MU2)*SUNSum(Col5,3)*SUNT(Glu3,Col5,Col2)*SUNT(Glu4,Col1,Col5)*
AbbSum5*Alfas*Pi
```

### 3.5 Translation to Fortran code

Numerical evaluation of the FormCalc results is typically done in Fortran, firstly for speed, and secondly for ease of inclusion into other programs. The choice of Fortran by no means precludes usage in C/C++ as it is straightforward to invoke and link the Fortran code. There is also a way to turn FormCalc-generated code into a Mathematica function [13].

Code generation for the squared amplitude is a highly automated procedure. FormCalc also offers low-level Fortran output functions with which it is very easy to turn an arbitrary Mathematica expression into Fortran code (see Ref. [12] for some examples of ‘non-standard’ code generation).

#### 3.5.1 Code Generation for the Squared Amplitude

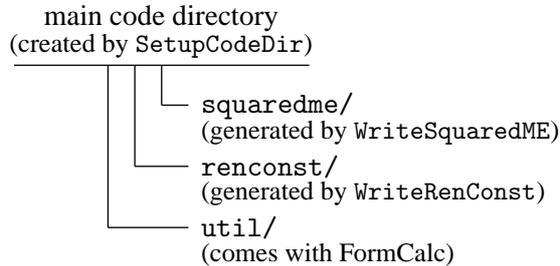
FormCalc has two fairly advanced functions for generating Fortran code, `WriteSquaredME` and `WriteRenConst`. The philosophy is that the user should not have to modify the generated code. This means that the code has to be encapsulated (i.e. no loose ends the user has to bother with), and that all necessary subsidiary files (include files, makefile) have to be produced, too.

First, a directory must be created for the code, and the driver programs copied into this directory with `SetupCodeDir`.

```
dir = SetupCodeDir["fortrandir"]
WriteSquaredME[M_tree, M_loop, abbr, dir]
WriteRenConst[M_loop, dir]
```

`WriteSquaredME` writes out a Fortran subroutine `SquaredME` to numerically evaluate  $|\mathcal{M}_{\text{tree}}|^2$  and  $2\text{Re}\mathcal{M}_{\text{loop}}\mathcal{M}_{\text{tree}}^*$ , where  $\mathcal{M}_{\text{tree}}$  and  $\mathcal{M}_{\text{loop}}$  are outputs of `CalcFeynAmp`. `WriteRenConst` searches  $\mathcal{M}_{\text{loop}}$  for renormalization constants and writes out a subroutine `CalcRenConst` for their computation.

The Fortran code is organized in a main code directory, which contains the main program and all its prerequisite files (e.g. to set up kinematics and model parameters), and subsidiary ‘folders’ (subdirectories to the main code directory). The default setup looks like this:



Each folder is equipped with its own makefile which makes a library of the same name, e.g. the makefile in `util/` makes the library `util.a`. These sub-makefiles are orchestrated by the master makefile.

Occasionally it is useful to have more than one instance of `squaredme` (or `renconst`), e.g. when computing an hadronic cross-section to which several partonic processes contribute. Both `WriteSquaredME` and `WriteRenConst` have the `Folder` option, with which a unique folder name can be chosen, and the `SymbolPrefix` option, with which the symbols visible to the linker can be prefixed with a unique identifier.

### 3.5.2 Low-level code-generation functions

FormCalc’s code-generation functions are also publicly available. They can be used to write out an arbitrary Mathematica expression as optimized Fortran code. The basic procedure is simple:

1. `handle = OpenFortran["file.F"]`  
opens `file.F` as a Fortran file for writing,
2. `WriteExpr[handle, {var → expr, ...}]`  
writes out Fortran code to calculate `expr` and store the result in `var`,
3. `Close[handle]`  
closes the file again.

The code generation is fairly sophisticated and goes well beyond merely applying Mathematica’s `FortranForm`. The generated code is optimized, e.g. common subexpressions are pulled out and computed in temporary variables. Expressions too large for Fortran are split into parts, as in

```

var = part1
var = var + part2
...

```

If the expression is too large even to be reasonably evaluated in one file, e.g. if the compile time becomes too long, the `FileSplit` function can distribute it on several files and optionally write out a master subroutine which invokes the individual parts.

To further automate the code generation, such that the resulting code needs few or no changes by hand, many ancillary functions are available, e.g. `VarDecl` writes out variable declarations for a given list of variables.

## 4. LoopTools

LoopTools supplies the actual numerical implementations of the one-loop integrals needed for programs made from the FormCalc output. It is based on the FF package [14] and provides in addition to the scalar integrals of FF also the tensor coefficients in the conventions of [15]. LoopTools offers three interfaces: Fortran, C++, and Mathematica.

Using the LoopTools functions in Fortran and C++ is very similar. In Fortran it is necessary to include the file `looptools.h` in every function or subroutine (for the common blocks). In C++, `clooptools.h` must be included once. Before using any LoopTools function, `ffini` must be called and at the end of the calculation `ffexi` may be called to obtain a summary of errors.

Recent additions to LoopTools include:

- The remaining complex-mass case of a  $D_0$  with four complex masses has been added [16].
- Dimensionally regulated IR and collinear divergences, so far implemented publicly only in QCDLoop [17] have been added. Currently only the scalar integrals are available (as in QCDLoop).

Technically, the parameter  $\lambda^2$  (LTLAMBDA), used hitherto to set the IR regulator ('photon') mass and thus implicitly assumed positive, now includes the cases  $\lambda^2 = -2$ ,  $\lambda^2 = -1$ , and  $\lambda^2 = 0$ , in which cases the  $1/\varepsilon^2$ ,  $1/\varepsilon$ , and finite piece are returned.

- The dispatcher for IR and collinear divergences has been replaced by a more efficient code. It constructs a bit pattern: 1 for zero argument, 0 otherwise, and can then jump to the correct case with a single table lookup.

## 5. Requirements and Availability

FeynArts, FormCalc, and LoopTools are available from

<http://feynarts.de>  
<http://feynarts.de/formcalc>  
<http://feynarts.de/looptools>

This website carries a script `FeynInstall` for easy installation. Each package contains a comprehensive manual. All three packages are open source and licensed under the LGPL.

## References

- [1] T. Hahn, *Comput. Phys. Commun.* **140** (2001) 418 [hep-ph/0012260].
- [2] T. Hahn, M. Pérez-Victoria, *Comput. Phys. Commun.* **118** (1999) 153 [hep-ph/9807565].
- [3] J. Küblbeck, M. Böhm, A. Denner, *Comput. Phys. Commun.* **60** (1991) 165.
- [4] T. Hahn, P. Lang, *Comput. Phys. Commun.* **179** (2008) 931 [0711.1345].
- [5] N.D. Christensen, C. Duhr, *Comput. Phys. Commun.* **180** (2009) 1614 [arXiv:0806.4194].
- [6] A. Semenov, *Comput. Phys. Commun.* **180** (2009) 431 [arXiv:0805.0555].

- [7] F. Staub, arXiv:0806.0538.
- [8] J.A.M. Vermaseren, *Nucl. Phys. Proc. Suppl.* **183** (2008) 19 [arXiv:0806.4080].
- [9] S. Dittmaier, *Phys. Rev. D* **59** (1999) 016007 [hep-ph/9805445].
- [10] T. Hahn, *Nucl. Phys. Proc. Suppl.* **116** (2003) 363 [hep-ph/0210220].
- [11] M. Tentyukov, J.A.M. Vermaseren, *Comput. Phys. Commun.* **176** (2007) 385 [cs/0604052].
- [12] T. Hahn, J.I. Illana, *Nucl. Phys. Proc. Suppl.* **160** (2006) 101 [hep-ph/0607049].
- [13] T. Hahn, *Comput. Phys. Commun.* **178** (2008) 217 [hep-ph/0611273].
- [14] G.J. van Oldenborgh and J.A.M. Vermaseren, *Z. Phys. C* **46** (1990) 425.
- [15] A. Denner, *Fortsch. Phys.* **41** (1993) 307 [arXiv:0709.1075].
- [16] D.T. Nhung, L.D. Ninh, *Comput. Phys. Commun.* **180** (2009) 2258 [arXiv:0902.0325].
- [17] R.K. Ellis, G. Zanderighi, *JHEP* **0802** (2008) 002 [arXiv:0712.1851].