FormCalc 9 and Extensions

T. Hahn[1], **S. Paßehr**[2], **C. Schappacher**[3]

[1] Max-Planck-Institut für Physik, Föhringer Ring 6, D–80805 Munich, Germany,
[2] DESY, Notkestr. 85, D–22607 Hamburg, Germany,
[3] Institut für Theoretische Physik, KIT, D–76128 Karlsruhe, Germany (former address).

E-mail: hahn@mpp.mpg.de, sebastian.passehr@desy.de, schappacher@kabelbw.de

Abstract. We present Version 9 of the Feynman-diagram calculator FormCalc and a flexible new suite of shell scripts and Mathematica packages based on FormCalc, which can be adapted and used as a template for calculations. Report DESY 2016-060, MPP-2016-63

1. Introduction

The Mathematica package FormCalc [1] simplifies Feynman diagrams generated by FeynArts [2] up to one-loop order. It provides the analytical results and can generate Fortran or C code for the numerical evaluation of the squared matrix element.

This note presents new features in FormCalc 9:

- Combination of processes with (almost) arbitrary kinematics.
- A wider range of cuts.
- Improved code generation and driver programs.
- Optional output of non-vectorized code.
- Various QCD functions (running, $\overline{\text{MS}} \to \overline{\text{DR}}$, etc.).
- Improvements in the makefiles and the build system.

Most of these improvements are the 'collected wisdom' from several nontrivial SUSY calculations [3, 4, 5, 6] and provide, as it were, 'extended support' for the MSSMCT model file [7].

Furthermore a new suite of shell scripts and Mathematica packages based on FormCalc is introduced. Presently used for the computation of the two-loop $\mathcal{O}(\alpha_t^2)$ Higgs-mass corrections in the MSSM, this code can easily be adapted and employed as a template for further calculations, and in this sense complements and generalizes what FormCalc does at one-loop.

2. New Features and Improvements in FormCalc 9

2.1. Combining Processes

FormCalc 9 allows to combine processes with (almost) arbitrary kinematics – only the phase-space dimension must match, i.e. the number of external legs must be the same. This is mostly used in hadronic computations, but one could do e.g. $e^+e^- \to \{e^+e^-, \mu^+\mu^-, \tau^+\tau^-\}$ in one go this way.

In a first step, the user generates code for all partonic processes needed with `WriteSquaredME` as usual, though with output to separate folders via the `Folder` option. The symbol `ProcName` may be used in the latter, as in:

```
WriteSquaredME[..., Folder -> {"squaredme", ProcName}]
```

and is substituted by the process name, abridged in such a way that it can safely be used as symbol or file name.

The file `partonic.h` determines which processes are combined and how. For each subprocess an entry of the following form must be added:

```
#define PID 1
#define PARTON1 i
#define PARTON2 j
#include "folder/specs.h"
#include "parton.h"
```

- `PID` is an identifier enumerating the partonic processes and runs from 1 to the number of processes (though not necessarily in ascending order),
- `PARTON1` and `PARTON2` give the PDG codes for the incoming partons, if the `lumi_hadron.F` module is used (they are ignored otherwise),
- *folder* is the folder into which code for the subprocess has been written,
- `parton.h` comes with FormCalc (same for all subprocesses) and does the processing of the partonic information given.

Optionally, the user can choose in `process.h` whether the results shall be combined at the differential or integrated level:

- `JOIN_PARTONIC` = 0, add differential partonic cross-sections, then integrate sum (default),
- `JOIN_PARTONIC` = 1, integrate each differential partonic cross-section, then add up.

Once `partonic.h` is complete it takes just the usual `./configure` and `make` to build the code.

2.2. Veto Cuts

Before FormCalc 9, only few one-particle cuts were available. The selection was restricted since the cuts were applied by actually restricting the integration bounds, hence solvability of the cut equations was a limiting factor.

While these direct cuts are still available (and likely more efficient, in particular if large parts of phase-space are removed), a much wider range of so-called 'veto cuts' has now been added. They operate through an indicator ('veto') function which sets the cross-section to zero whenever a phase-space point removed by the cuts is sampled. The following one- and two-particle cuts are currently available and it is straightforward to add more in FormCalc's `cuts.F`:

`CUT_E(`i`)`	`CUT_R(`i`,`j`)`
`CUT_k(`i`)`	`CUT_rho(`i`,`j`)`
`CUT_ET(`i`)`	`CUT_deltay(`i`,`j`)`
`CUT_kT(`i`)`	`CUT_deltaeta(`i`,`j`)`
`CUT_y(`i`)`	`CUT_yprod(`i`,`j`)`
`CUT_eta(`i`)`	`CUT_etaprod(`i`,`j`)`
`CUT_deltatheta(`i`)`	`CUT_deltaalpha(`i`,`j`)`
`CUT_cosdeltatheta(`i`)`	`CUT_cosdeltaalpha(`i`,`j`)`
	`CUT_invmass(`i`,`j`)`

The overall cut condition is assembled from the one- and two-particle cuts in `run.F`, in the definitions for `CUT1...20`, for example

```
#define CUT1 CUT_kT(3).gt.10
```

requires the transverse momentum of particle #3 to be greater than 10 GeV. The `CUT1...20` definitions together make up one single logical expression and may include logical operators, e.g.

```
#define CUT1 CUT_kT(3) .gt. 10
#define CUT2 .and. CUT_kT(4) .gt. 10
```

2.3. Non-vectorized Code

FormCalc's `WriteSquaredME` function generates code which is by default vectorized for the helicities of the external particles [8].

While vectorization is meanwhile almost universally available on all modern hardware, having this feature turned on for all code is not ideal, either. For example, if the user wishes to include the generated subroutines in own code, the vectorization constitutes an extra layer of complexity.

For this reason FormCalc 9 allows to remove all vector instructions and special macros by choosing the output language with the `"novec"` qualifier, viz.

```
SetLanguage["Fortran","novec"]
SetLanguage["C","novec"]
```

2.4. Debugging and Checking Instructions

FormCalc has for long allowed to add debugging statements to the generated code, as in:

```
var = expr
DEB("var", var)
```

with (e.g.)

```
#define DEB(tag,var) print *, tag,"=", var
```

This has been extended in Version 9 to include *checking statements* of the form

```
CHK_PRE(var)
var = expr
CHK_POST("var", var)
```

which might be implemented as

```
#define CHK_PRE(var) tmp = var
#define CHK_POST(tag,var) if(abs(var-tmp).gt.1D7) stop tag
```

The idea here is to catch enormous variations from one parameter point to the next, which often point to some problem of the calculation, e.g. an untreated resonance. Finding the error locus is much more difficult using regular debug statements due to their copious output, which is moreover difficult to compare using standard 'diff' tools because even results that are close (but not identical) numerically are usually highlighted in the diff output.

The user can individually choose the actual debugging statement (`$DebugCmd[`i`]`) as well as commands issued before (`$DebugPre[`i`]`) and after (`$DebugPost[`i`]`) it, for $i = 1$ (debug), 2 (pre-check), and -2 (post-check).

2.5. New/improved QCD Functions

Various QCD utility functions have been adapted, mostly from RunDec [9], and added to the 'util' library of FormCalc. They are used e.g. for translating between various quantities in the MSSMCT model file [7].

- subroutine `AlphaS`: $\alpha_s^{\overline{\rm MS}}(Q)$ up to 4-loop order [9],
- subroutine `MqRun`: $m_q^{\overline{\rm MS}}(Q_1) \to m_q^{\overline{\rm MS}}(Q_2)$ up to 3-loop order [9],

- subroutine `MqMSbar2OS`: $m_q^{\overline{\rm MS}}(Q) \to m_q^{\rm OS}$ up to 3-loop order [9],
- subroutine `AsMSbar2DRbar`: $\alpha_s^{\overline{\rm MS}} \to \alpha_s^{\overline{\rm DR}}$ up to 2-loop order [10].
- subroutine `SetQCDPara`: set inputs ($\alpha_s(M_Z)$, thresholds) for the above routines.

2.6. More Renormalization Constants

The calculation of renormalization constants (RCs) has been extended to functions of arbitrary head, e.g. mass shifts. Output is written to a separate file for each head, for example:

```
RenConst[.] := ...   →  RenConst.F, .h      (so far)
MassShift[.] := ... →  MassShift.F, .h     (new)
```

This effectively creates subsets of RC-like objects, i.e. results of a loop calculation but constant (external kinematics fixed). This is useful, for example, when computing the shifts used in the iterative determination of the Yukawa-resummation parameter Δb since not all RCs, only the mass shifts, have to be recomputed in each iteration.

Results for head h can either be requested explicitly by adding h to the argument list, e.g.

```
CalcRenConst[expr, MassShift]
```

or computed together with the others in the usual way (no extra arguments) by adding h to the `$RenConst` list of RC-heads, e.g.

```
$RenConst = {RenConst, MassShift}
```

A model file could for example request that `MassShift` be computed by adding it to `$RenConst`.

3. Extensions

FeynArts/FormCalc have been designed, as many other software packages – and not just in high-energy physics, to do a 'complete' job. That is, all the steps from the generation of Feynman diagrams to the numerical computation of a cross-section are executed from a single control program (e.g. a single Mathematica session) – that at least is how the demo programs insinuate usage. There is nothing inherently wrong with this 'monolithic' approach, it is maybe just not obvious how to extend it beyond the limitations of the package(s) used.

Sometimes one needs to use other packages for specific tasks, or there are special requirements or adaptations necessary, for example

- Resummations (e.g. the *hbb* Yukawa coupling in the MSSM),
- Approximations (e.g. the gaugeless limit),
- K-factors,
- Nontrivial renormalization.

In the following we shall introduce a suite of scripts and small Mathematica programs [11] made for the computation of the two-loop $\mathcal{O}(\alpha_t^2)$ corrections to the Higgs masses in the MSSM [12], with optimized output suitable for inclusion in FeynHiggs [13]. While each step is specific to the corrections computed here the code can be used as a template due to its modular structure and adapted with little effort for similar calculations.

3.1. Components of the Calculation

The MSSM Higgs masses receive leading two-loop (2L) corrections of $\mathcal{O}(\alpha_s\alpha_t)$ and $\mathcal{O}(\alpha_t^2)$. We are performing a diagrammatic calculation of the latter in the full complex MSSM in the gaugeless limit at $p^2 = 0$. The following ingredients are needed:

① The unrenormalized 2L self-energies $\Sigma^{(2)}_{hh}$, $\Sigma^{(2)}_{hH}$, $\Sigma^{(2)}_{hA}$, $\Sigma^{(2)}_{HH}$, $\Sigma^{(2)}_{HA}$, $\Sigma^{(2)}_{AA}$, $\Sigma^{(2)}_{H^+H^-}$ in gaugeless approximation at $p^2 = 0$ at leading order in $\mathcal{O}(\alpha_t^2)$.

② The 1L diagrams with insertions of 1L counterterms.

③ The 2L counterterms for ①.

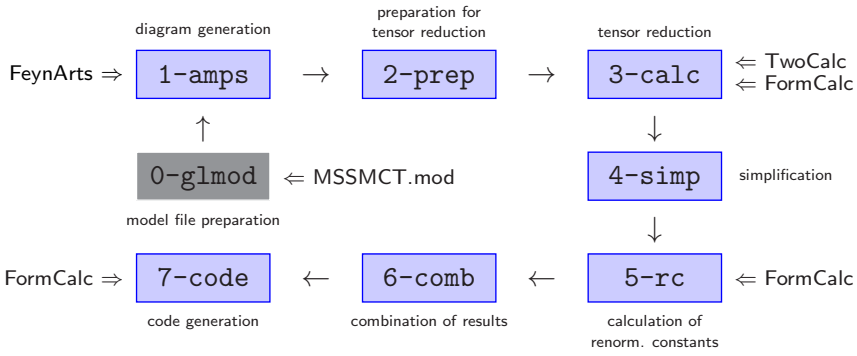
④ The 2L tadpoles $T^{(2)}_h$, $T^{(2)}_H$, $T^{(2)}_A$ at $\mathcal{O}(\alpha_t^2)$ appearing in ③.

3.2. Template for Calculations

Our present code is based on several Mathematica Notebooks used in the former calculation [12]. From the aspect of software development these Notebooks had various shortcomings: duplicate code (e.g. gaugeless limit implemented multiply), parallel instructions poured all over, even the requirement to run with a particular Mathematica version – all signs that the use of different packages together with various special requirements was nontrivial and that the reorganization was helpful.

We broke the calculation into several steps and implemented each step as an independent (stand-alone) shell script. These scripts are run from the command line during development, while in the production run a makefile organizes the entire sequence. In lieu of *in vivo* debugging, e.g. setting breakpoints, which is not easily possible with a shell script we have set up detailed log files. The outcome is a template for 2L calculations with optimized output in a nontrivial model with nontrivial renormalization.

The scripts use several external packages: FeynArts for the diagram generation [2] using the MSSM model file with 1L counterterms MSSMCT.mod [7], FormCalc for the 1L tensor reduction and code generation [1], and TwoCalc for the 2L tensor reduction [14].



3.3. Script Structure

The shell scripts (`/bin/sh`) have several traits in common:

- Each script is run from the command line with up to two arguments, e.g.

  `scripts/1-amps` arg_1 arg_2

  where arg_1 = `h0h0`, `h0HH`, `h0A0`, `HHHH`, `HHA0`, `A0A0`, `HmHp` (self-energies),
  `h0`, `HH`, `A0` (tadpoles),
  arg_2 = `0` for virtual 2L diagrams,
  `1` for 1L diagrams with 1L counterterms.

  That is, each of the components listed in Sect. 3.1 is computed and stored individually. Arguments are checked upon entry and meaningless arguments are omitted (e.g. no arg_2 after combination of virtual and counterterm diagrams).
- Inputs and outputs are defined in the first few lines, e.g.

```
in=m/$1/2-prep.$2
out=m/$1/3-calc.$2
```

- Symbolic output including log files are written to the '`m`' subdirectory. The log file names are the output file names with `.log.gz` appended.
- Fortran code is written to the '`f`' subdirectory.
- The scripts use the shell's 'here documents' to run the Mathematica Kernel [15].

3.4. Script Tasks

In the following we shall briefly describe the main tasks of each script. The total running time for all scripts for all self-energies/tadpoles (i.e. the complete calculation) is about 15–20 min.

Step 0: Gaugeless Limit

We have to work in the gaugeless limit and also set $m_b = 0$ in order that the $\mathcal{O}(\alpha_t^2)$ corrections form a supersymmetric and gauge-invariant subset. The gaugeless limit consists of ① setting the gauge couplings $g, g' = 0$ and hence $M_W, M_Z = 0$ while keeping ② the weak mixing angle and ③ the quantities $\delta M_W^2/M_W^2$ and $\delta M_Z^2/M_Z^2$ finite.

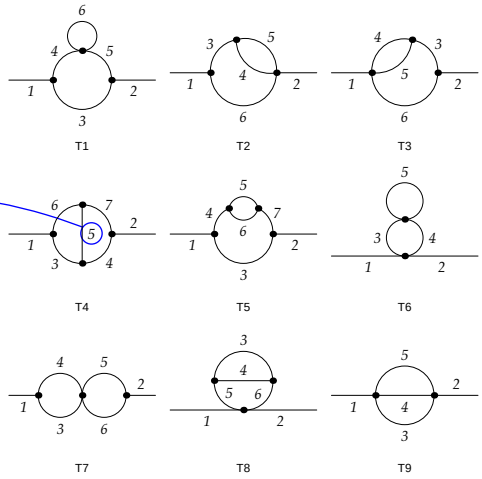
No. ① and ② are most efficiently accomplished by modifying the Feynman rules, carried out by the `0-glmod` script:

- Load the `MSSMCT.mod` model file.
- Modify couplings and remove those that become zero.
- Write out an `MSSMCTgl.mod` model file.

Step 1: Diagram Generation The `1-amps` script generates the 2L virtual ($arg_2 = 0$) and 1L-plus-counterterm ($arg_2 = 1$) diagrams using wrappers around FeynArts functions provided by the `FASettings.m` package. The wrappers mainly simplify the diagram selection, e.g. the following list of filters, one for each topology, is composed from elementary functions like `htb[`i`]` which restrict the particles on the i-th propagator:

```
sel[0][S[_] -> S[_]] = {
  t[3] && htb[6],
  t[3] && tb[6],
  t[3] && tb[6],
  t[3] && t[4] && htb[5],
  t[3] && htb[5|6],
  t[3] && htb[5],
  t[3] && t[5],
  t[5] && ht[3|4],
  t[3|4|5] && ht[3|4|5] }
```

"one of $h_i, \tilde{\chi}$, $t, \tilde{t},\ b, \tilde{b}$"



Step 2: Preparation for Tensor Reduction The tensor reduction is traditionally the step which increases the number of terms in the amplitude most, therefore we inserted Step 2 before and Step 4 after the tensor reduction to keep the number of terms as small as possible at all times.

Step 2, 'before', takes the $p^2 \to 0$ limit and simplifies the ubiquitous sfermion mixing matrices U_{ij}, mostly by exploiting unitarity ($\sim 50\%$ size reduction).

The unitarity relations of 2×2 matrices are easily written down, e.g. $U_{11}U_{11}^* + U_{12}U_{12}^* = 1$, but Mathematica rarely arranges expressions in just the way that they apply directly. This can be improved by adding definitions for single summands, e.g. $|U_{22}|^2 = |U_{11}|^2$. Because they apply *while* Mathematica ponders the simplification strategy they increase the incentive for `Simplify` to choose the unitarity-simplified version. The product of two matrix elements is too 'deep' for Mathematica to consider as an l.h.s., however, hence we must introduce intermediate symbols: we substitute `USf[1,`j`] USfC[1,`j`]` $\to$ `UCSf[1,`j`]` etc. and can then formulate unitarity as e.g. `UCSf[2,2] = UCSf[1,1]`. The `USfSimplify.m` package contains the complete set of such rules.

Step 3: Tensor Reduction The actual tensor reduction is a relatively straightforward invocation of TwoCalc [14] for the 2L and FormCalc [1] for the 1L amplitudes. Noteworthy here is perhaps that TwoCalc and FormCalc cannot be loaded into one Mathematica session because of symbol conflicts; this is easily accomodated in the shell-script setup, however.

Step 4: Simplification The amplitudes are simplified again after tensor reduction using a largely empirical recipe. The following trick, lifted from [16], enhances simplification efficiency greatly: already during diagram generation a function of the form `DiagMark[`m_i`]` has been inserted by the FeynArts wrapper functions, where m_i are the masses running in the loop. Few simplifications can be made between parts with different `DiagMark`, hence simplification can be restricted to the prefactors of the `DiagMark`, rather than the entire expression, which is of course much faster.

Step 5: Calculation of Renormalization Constants The 1L RCs are computed with FormCalc. As mentioned in Step 0, $\delta M_V^2/M_V^2$ ($V = W, Z$) must remain finite in the gaugeless limit in which M_V vanishes. By substituting the explicit mass dependence: `dMVsq1` $\to$ `MV2 dMVsq1MV2` the masses cancel and we can take the limit safely.

Furthermore we insert expansions of the loop integrals in $\varepsilon = \frac{4-D}{2}$ (package `ExpandDel.m`) and collect the RCs w.r.t. powers of ε. For example, the 1L top-mass counterterm becomes

```
{ dMf1[3,3] → RC[-1, dMf1[-1,3,3]] + RC[0, dMf1[0,3,3]],
  {dMf1[-1,3,3] → ..., dMf1[0,3,3] → ...} }
```

where the first line denotes the expansion in ε (the interim function `RC` is used for power counting later) while the second gives the actual expressions for the coefficients. This enables us to write down just the ε^0-coefficient of the final result in Step 6 without substituting back all RCs.

Step 6: Combination of Results The virtual 2L and 1L-plus-counterterm amplitudes, computed separately so far, are combined with the genuine 2L counterterms (hand-coded, in `MSSMCT.rc2`) into one expression for each self-energy/tadpole. We insert the ε-expansions of the RCs from Step 5 and once more those of the loop integrals, and extract the coefficient of ε^0 (or of ε^{-1} or ε^{-2} if a debug flag is set). This expression is simplified again and forms then the final analytical result.

Step 7: Code Generation The renormalized self-energies are grouped into three subroutines for output, `TLsp_atat_c` for the ones needed for renormalization (`A0A0`, `HmHp`), `TLsp_atat_e` for the CP-even ones, and `TLsp_atat_o` for the CP-odd ones. These are the units in which FeynHiggs needs to compute the self-energies based on its flags.

Abbreviations are introduced for loop integrals and common subexpressions to shorten the overall expression ($\sim 50\%$ size reduction). Optimized Fortran code is then written out using FormCalc's `WriteExpr` and ancillary functions [17]. Finally, static code is added which embeds the generated routines into FeynHiggs. The Fortran output directory '`f`' totals about 350 kBytes in size and can directly be moved into the FeynHiggs source tree.

4. Résumé

FormCalc Version 9, available for download from `http://feynarts.de/formcalc`, introduces various new features, most adapted/generalized from real-life SUSY calculations:

- Combination of processes,
- More cuts,
- Improved code generation,
- New/improved driver and utility programs.

As a showcase of how to flexibly use FeynArts and FormCalc together with other packages, a suite of shell scripts and Mathematica packages was presented, originally made for the computation of the two-loop $\mathcal{O}(\alpha_t^2)$ Higgs-mass corrections, which may serve as template for similar calculations:

- Two-loop,
- Nontrivial model (MSSM) and renormalization,
- Specific approximations (gaugeless, $p^2 = 0$),
- Optimized, compact output.

The code is included in FeynHiggs 2.11.0 and above in the `gen/tlsp/` subdirectory.

[1] Hahn T, Pérez-Victoria M, 1999, *Comput. Phys. Commun.* **118** 153 [hep-ph/9807565].
[2] Hahn T, 2001, *Comput. Phys. Commun.* **140** 418 [hep-ph/0012260].
[3] Fritzsche T, Heinemeyer S, Rzehak H, Schappacher C, 2011, *Phys. Rev.* **D86** 035014 [arXiv:1111.7289].
[4] Heinemeyer S, Schappacher C, 2012, *Eur. Phys. J.* **C72** 1905 [arXiv:1112.2830].
[5] Heinemeyer S, Schappacher C, 2015, *Eur. Phys. J.* **C75** 198 [arXiv:1410.2787].
[6] Heinemeyer S, Schappacher C, arXiv:1511.06002.
[7] Fritzsche T, Hahn T, Heinemeyer S, von der Pahlen F, Rzehak H, Schappacher C, 2014, *Comput. Phys. Commun.* **185** 1529 [arXiv:1309.1692].
[8] Chokoufe Nejad B, Hahn T, Lang JN, Mirabella E, 2010, *J. Phys. Conf. Ser.* **523** 012050 [arXiv:1310.0274].
[9] Chetyrkin KG, Kühn JH, Steinhauser M, 2000, *Comput. Phys. Commun.* **133** 43 [hep-ph/0004189].
[10] Harlander R, Mihaila L, Steinhauser M, 2007, *Phys. Rev.* **D76** 055002 [arXiv:0706.2953].
[11] Hahn T, Paßehr S, arXiv:1508.00562.
[12] Hollik W, Paßehr S, 2014, *Phys. Lett.* **B733** 144 [arXiv:1401.8275].
[13] Frank M, Hahn T, Heinemeyer S, Hollik W, Rzehak H, Weiglein G, 2007, *JHEP* **0702** 047 [hep-ph/0611326].
[14] Weiglein G, Scharf R, Böhm M, 1994, *Nucl. Phys.* **B416** 606 [hep-ph/9310358], Weiglein G, Mertig R, Scharf R, Böhm M, 1992, in: New computing techniques in physics research II, 617–623, La Londe-les-Maures.
[15] Hahn T, Illana J, 2006, *Nucl. Phys. Proc. Suppl.* **160** 101 [hep-ph/0607049].
[16] Heinemeyer S, Stöckinger D, Weiglein G, 2004, *Nucl. Phys.* **B690** 62 [hep-ph/0312264] and 2004, *Nucl. Phys.* **B699** 103 [hep-ph/0405255].
[17] Hahn T, 2010, PoS ACAT **2010** 078 [arXiv:1006.2231].