FormCalc 6

Thomas Hahn[*]

Max-Planck-Institut für Physik
Föhringer Ring 6
D–80805 München
E-mail: `hahn@feynarts.de`

The new features and improvements in FormCalc Version 6 as well as some recent additions in FeynArts for easier diagram selection are reported.

---

[*]Speaker.

1. Introduction

FeynArts [1] and FormCalc [2] generate and calculate Feynman diagrams to the extent that one can easily obtain the analytic expression of an amplitude at up to one-loop order and create a Fortran code for the numerical evaluation of the squared matrix element almost fully automatically.

This note presents some recent improvements and additions to both programs. Firstly, it elaborates on diagram filters in FeynArts, for which several ancillary functions have been added. Secondly, the following new features in FormCalc 6 are described:

- Alternate-channel communication between FORM and Mathematica.

- Code-generation for CutTools.

- Improvements in Dirac chains (4D) for analytical purposes.

- Functions to re-use previously introduced abbreviations and subexpressions in a new session.

- The Keep function to resume interrupted calculations.

- New Abbreviate mode.

- Options to fine-tune code generation.

2. Diagram filters in FeynArts

The DiagramSelect function in FeynArts performs on Feynman diagrams (i.e. the output of InsertFields, denoted by 'ins' in the following examples) what Select does to ordinary Mathematica expressions: it retains those for which a given test function returns True.

In the case of DiagramSelect, the test function receives three arguments:

1. the insertions, of the form Graph[h][Field[1] → f_1, Field[2] → f_2, ...],

2. the topology belonging to these insertions,

3. the header of the topology list to which the topology belongs.

The DiagramSelect function is part of FeynArts already for quite a while and the information it provides to the test function allows the skilled programmer a very detailed selection. Recently, a number of functions have been added with which the construction of sophisticated test functions is straightforward. Unless otherwise stated, the functions introduced in the following subsections take the three arguments detailed above.

2.1 **LoopFields**

LoopFields returns a list of fields running in the loop(s) of the given diagram. It might be used as in

```
DiagramSelect[ins, FreeQ[LoopFields[##], V[1]]&]
```

to omit all diagrams where a photon runs in the loop. Or, to select only diagrams of $\mathcal{O}(\alpha_s)$ or higher with respect to tree-level, one could use:

```
CountQCDFields[f_] :=
  Count[f, _. (F[3|4, ___] | V[5, ___] | U[5, ___])];
DiagramSelect[ins, CountQCDFields[LoopFields[##]] > 2 &]
```

2.2 `WFCorrectionFields`, `WFCorrectionCTFields`

`WFCorrection(CT)Fields` returns the fields external to wave-function corrections, i.e. self-energies on external legs (or their counter-terms in the case of `WFCorrectionCTFields`). For example, if the diagram contains a self-energy insertion `S[1]` → `S[3]` on an external leg, the result of `WFCorrectionFields` is {`S[1]`, `S[3]`}. If the diagram contains no wave-function correction, the list is empty.

This filter is typically used to eliminate wave-function corrections with identical external legs, i.e. remove corrections of the type $a \to a$ but keep $a \to b$ (for, the former contain a denominator $1/(p^2 - m_a^2)$ which, if naively inserted, evaluates to $1/(m_a^2 - m_a^2)$ for an on-shell p). This can be done with a construction like

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

2.3 `Vertices`, `FieldPoints`

`Vertices` returns the vertices contained in a topology, not counting the endpoints of the external legs (even though they are internally represented as `Vertex[1][n]`). Note that this is a purely topological list of vertices, with no field information.

`FieldPoints` returns the field content for each vertex of a topology, i.e. a list of objects of the form `FieldPoint[cto][fields]`, where *cto* is the counter-term order.

These functions are most efficiently used together with the following.

2.4 `FieldMatchQ`, `FieldMemberQ`, `FieldPointMatchQ`, `FieldPointMemberQ`

`FieldMatchQ[f, patt]` returns `True` if the field *f* matches the pattern *patt* and `False` otherwise. It works like `MatchQ` but takes into account field levels, e.g. `F[1]` matches `F`.

`FieldMemberQ[flist, patt]` returns `True` if an element of *flist* matches *patt* in the sense that `FieldMatchQ` returns `True`.

`FieldPointMatchQ[fp, patt]` returns `True` if the field point *fp* matches the pattern *patt* and `False` otherwise. The matching takes into account field levels, e.g. `F[1]` matches `F`.

`FieldPointMemberQ[fplist, patt]` returns `True` if an element of *fplist* matches *patt* in the sense that `FieldPointMatchQ` returns `True`.

With these functions, it is quite simple to select diagrams with a particular field point, e.g.

```
DiagramSelect[ins, FieldPointMemberQ[FieldPoints[##],
  FieldPoint[V[1], F[2], -F[2]]]&]
```

selects only diagrams with a photon–lepton–lepton coupling.

3. Alternate Link between FORM and Mathematica

FORM is renowned for being able to handle very large expressions. To yield (pre-)simplified expressions, however, terms have to be wrapped in functions, to avoid immediate expansion. The number of terms in a function is rather severely limited in FORM: on 32-bit systems to 32568.

FormCalc is thus in a dilemma: it has become much more sophisticated in simplifying amplitudes in recent versions and partly because of this, users now compute larger amplitudes. This is the reason that many users have recently reported (unorthodox) 'overflow' messages from FORM.

The solution introduced in Version 6 is to send the pre-simplified generic amplitude to Mathematica intermediately for introducing abbreviations. To enable this out-of-band communication, the ReadForm executable (which manages the traffic between FORM and Mathematica) threads off a pipe handler which communicates with FORM via its external channels [3].

The result is a significant reduction in size of the intermediate expressions, shown here for the u-channel part of the tree-level amplitude of $uu \to gg$. The pre-simplified generic expression passed from FORM to Mathematica is in this case:

```
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]]]] )
```

and the version coming back from Mathematica is:

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

4. CutTools

The CutTools package evaluates one-loop integrals via the cutting-technique-inspired Ossola, Papadopoulos, Pittau (OPP) method [4].

FormCalc 6 can generate code for linking with CutTools. This is governed by the `CutTools` option of `CalcFeynAmp`, which has three values: `False` selects LoopTools functions [2], i.e. traditional Passarino–Veltman tensor reduction, `True` selects CutTools functions, but with the rational terms worked out by FormCalc analytically, and `Rational` selects CutTools functions with the rational terms computed by CutTools.

The CutTools functions emitted by FormCalc form a set $\texttt{Acut}$, $\texttt{Bcut}$, etc., analogous to the $\texttt{A0i}$, $\texttt{B0i}$, etc. LoopTools functions, i.e. the Passarino–Veltman tensor coefficient functions. From the technical point of view, the CutTools functions require the numerator as a function of the integration momentum q which it can sample to solve the cutting equations. The correspondence is, for example,

$$
\int \frac{(i\pi^2)^{-1}\,\mathrm{d}^4 q \, q_\mu q_\nu}{(q^2 - m_1^2)((q-p)^2 - m_2^2)} = g_{\mu\nu}\,\texttt{B0i}(\texttt{bb00}, p^2, m_1^2, m_2^2) + p_\mu p_\nu\,\texttt{B0i}(\texttt{bb11}, p^2, m_1^2, m_2^2)
$$
$$
= \texttt{Bcut}(2, \texttt{num1}, \texttt{num2}, p, m_1^2, m_2^2)\,.
$$

The first line is the classical Passarino–Veltman decomposition, formulated in terms of the Loop-Tools function $\texttt{B0i}$. The second line is the two-point CutTools function $\texttt{Bcut}$.

The main novelty here are the two numerator functions $\texttt{num1} = q_\mu q_\nu$ and $\texttt{num2} = 0$, the latter of which is the coefficient of $D - 4$ and is sampled only if CutTools is used to compute also the rational terms. Although not evident from this example, the numerators are in general scalar functions, as the q_μ are contracted with other vectors, are part of a fermion chain, or similar. The 2 describes the maximum tensor rank ($q_\mu q_\nu = 2$ instances of q). Note further that the momentum p is passed directly (not squared), as the routine may have to compute dot products etc.

The final link with CutTools, including proper detection of the package in $\texttt{configure}$, is not yet complete but will be available shortly. For this reason, no performance reports can yet be given. Regardless of any performance questions, however, the CutTools option in itself is definitely important as an independent way of checking LoopTools results.

5. Dirac Chains in 4D

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

Already in Version 5, with improvements in Version 6, Fierz methods have been implemented for Dirac chains, thus allowing the user to force the Dirac chains into almost any desired order, via the $\texttt{FermionOrder}$ option of $\texttt{CalcFeynAmp}$.

Version 6 adds the $\texttt{Colour}$ method to the $\texttt{FermionOrder}$ option, which brings the spinors into the same order as the external colour indices. This is not entirely trivial internally because colour is treated as an insertion, i.e. is not present during the simplification of the generic amplitude.

Also new in Version 6 is the $\texttt{Antisymmetrize}$ option of $\texttt{CalcFeynAmp}$ which anti-symmetrizes Dirac chains. Such chains can be recognized by a negative chirality identifier, e.g. $\texttt{DiracChain[-6, ...]}$ instead of $\texttt{DiracChain[6, ...]}$. More precisely,

$$
\texttt{DiracChain}[-\lambda, \mu_1, \ldots, \mu_n] = \frac{1}{n!}\sum_\pi \sigma_\pi \omega_\lambda \gamma_{\pi(\mu_1)} \cdots \gamma_{\pi(\mu_n)}
$$

where the sum extends over all permutations π of $\mu_1, \ldots, \mu_n$, σ_π is the sign of the permutation, and $\omega_1 = \mathbb{1}$, $\omega_5 = \gamma_5$, $\omega_{6,7} = \frac{1}{2}(\mathbb{1} \pm \gamma_5)$. This notation includes as a special case $\sigma_{\mu\nu} = \frac{1}{2}(\gamma_\mu \gamma_\nu - \gamma_\nu \gamma_\mu) = \texttt{DiracChain[-1}, \mu, \nu\texttt{]}$.

5

6. Re-using Abbreviations

Abbreviations are a fundamental concept in FormCalc to reduce the size of expressions. Abbreviations were so far restricted to one FormCalc session, however, e.g. one could not save intermediate results involving abbreviations and resume computation in a new session. FormCalc 6 adds two functions to 'register' abbreviations and subexpressions from an earlier session.

RegisterAbbr registers a list of abbreviations, e.g. the output of Abbr[] in a previous session, such that future invocations of CalcFeynAmp will make use of them. Note that abbreviations introduced for different processes are in general not compatible.

RegisterSubexpr registers a list of subexpressions, e.g. the output of Subexpr[] in a previous session, such that future invocations of Abbreviate will make use of them.

7. `Keep`

The Keep function is an application of RegisterAbbr and RegisterSubexpr and is helpful in long-running calculations to store intermediate expressions, such that the calculation can be resumed after a crash. As a side effect, the intermediate results can be inspected easily, even while a batch job is in progress.

Keep has two basic arguments: a file (path and name) and an expression. If the file exists, it is loaded. If not, the expression is evaluated and the results stored in the file, thus creating a checkpoint. If the calculation crashes, it suffices to restart the very same program, which will then load all parts of the calculation that have been completed and resume at the point it left off.

Keep[*expr*, *name*, *path*] loads "*path*/*name*.m" if it exists, otherwise evaluates *expr* and stores the result (together with the output of Abbr[] and Subexpr[]) in that file. *path* is optional and defaults to $KeepDir (= "keep" by default).

Keep[*lhs* = *rhs*] is the same as *lhs* = Keep[*rhs*, "*lhs*"]. This second syntax makes adding Keep functionality to existing programs very simple. For example, a statement like

```
amps = CalcFeynAmp[...]
```

simply becomes

```
Keep[amps = CalcFeynAmp[...]]
```

Since the file name is derived from the assigned-to identifier, this logic fails to work if symbols are being re-assigned, i.e. appear more than once on the left-hand side, as in

```
Keep[amps = CalcFeynAmp[virt]]
Keep[amps = Join[amps, CalcFeynAmp[counter]]
```

Due to the first Keep statement, the second will always find the file keep/amps.m and never execute the computation of the counter terms.

There are other ways to confuse the system, which usually require some deliberation, however: mixing intermediate results from different calculations, changing flags out of sync with the intermediate results, etc. In case of doubt, i.e. if results seem suspicious, remove all intermediate files and re-do the calculation from scratch.

8. New `Abbreviate` mode

The Abbreviate function, first introduced in Version 5, lets the user introduce abbreviations for arbitrary expressions. The main deal here is the optimization performed during Fortran-code generation, where the abbreviations are grouped into categories, such that each one is evaluated as few times as possible – e.g. for MSSM calculations this routinely leads to speed-ups of ≈ 3.

The only invocation so far was Abbreviate[*expr*, *lev*], where the integer *lev* was the level inside *expr* below which abbreviations were introduced and hence controlled how much of *expr* was 'abbreviated away.' This has been complemented by the Abbreviate[*expr*, *patt*] mode, which introduces abbreviations for all things free of the pattern *patt*. Aside from performance issues, this is useful to get a picture of the structure of an expression. For example,

```
Abbreviate[a + b + c + (d + e) x, x, MinLeafCount -> 0]
```

gives Sub2 + Sub1 x, thus indicating that the original expression is linear in x. (Note that the MinLeafCount option here is necessary because by default no abbreviations would be introduced for such simple subexpressions.) As before, Subexpr[] recovers the subexpressions.

9. Varia

WriteSquaredME and WriteRenConst have new options to fine-tune Fortran-code generation. This is useful if one needs the generated code for purposes other than compilation with FormCalc's own driver programs. The FileIncludes and SubroutineIncludes options respectively specify per-file and per-subroutine #include statements or similar declaration code, and FileHeader gives the header written to each generated Fortran file.

The ReadForm MathLink executable has been made much more stable with respect to user aborts (Ctrl-C), i.e. ReadForm zombie processes eating up CPU time will hopefully be a relict of the past now.

10. Summary

The latest FeynArts release (www.feynarts.de) features a number of functions which aid the user in constructing selection functions for DiagramSelect.

The new FormCalc Version 6 (www.feynarts.de/formcalc) has many new and improved features. The most important change is one the user will scarcely notice: the exchange of expressions with Mathematica halfway through the evaluation in FORM, which removes a bottleneck for largish amplitudes. The support for CutTools is complete as far as the algebraic generation of amplitudes; the remaining few (mostly technical) details regarding linking etc. will be finalized soon.

References

[1] T. Hahn, *Comp. Phys. Commun.* **140** (2001) 418 [hep-ph/0012260].

[2] T. Hahn, M. Pérez-Victoria, *Comp. Phys. Commun.* **118** (1999) 153 [hep-ph/9807565].

[3] M. Tentyukov, J. Vermaseren, *Comp. Phys. Commun.* **176** (2007) 385 [cs/0604052].

[4] G. Ossola, C. Papadopoulos, R. Pittau *JHEP* **0803** (2008) 042 [arXiv:0711.3596].