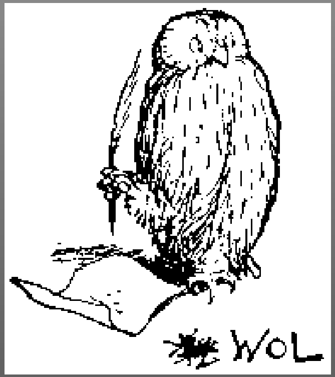
FormCalc and FeynArts

Thomas Hahn

Max-Planck-Institut für Physik
München

Contradiction

An Introduction is to introduce people, but FeynArts and Friends have already been introduced to you. So this is the opposite. When we asked Pooh what the opposite of an Introduction was, he said "The what of a what?" which didn't help us as much as we had hoped, but luckily Owl kept his head and told us that the Opposite of an Introduction, my dear Pooh, was a Contradiction.



. . . actually, a Distinction:

FeynArts/FormCalc	FeynHiggs
Generic answer to an arbitrary question	Concrete answer to a specific question
Paint & brush	Finished painting
Generator generator	Generator

One-loop since mid-1990s

Automated NLO computations is an industry today, with many packages becoming available in the last decade:

- GoSam, MadGraph5_aMC@NLO, OpenLoops, HELAC-NLO, BlackHat, Rocket, ...

Here: FeynArts (1991) + FormCalc (1995)

FormCalc was doing largely the same as FeynCalc (1992) but used FORM for the time-consuming tasks, hence the name FormCalc.

- Feynman-diagrammatic method,
- Analytic calculation as far as possible ('any' model),
- Generation of code for the numerical evaluation of the squared matrix element.

FeynArts + FormCalc also used as 'engine' in SARAH, SloopS.

Package Types

‘Production’	‘Exploration’	‘Specific’
MG5_aMC@NLO	FormCalc	FeynHiggs
GoSam	FeynCalc	DarkSUSY
OpenLoops	Package-X	Prospino

Diagram Evaluation with FeynArts and FormCalc

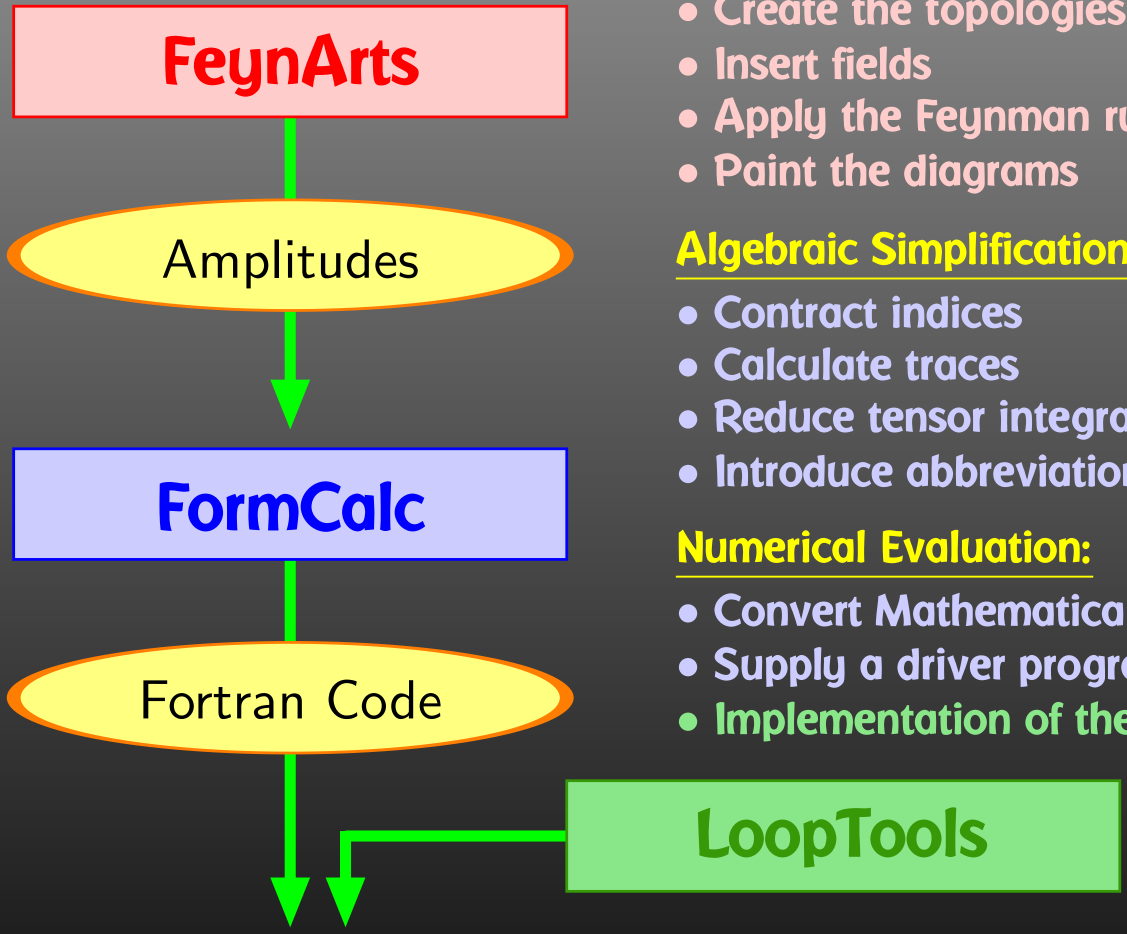


Diagram Generation:

- Create the topologies
- Insert fields
- Apply the Feynman rules
- Paint the diagrams

Algebraic Simplification:

- Contract indices
- Calculate traces
- Reduce tensor integrals
- Introduce abbreviations

Numerical Evaluation:

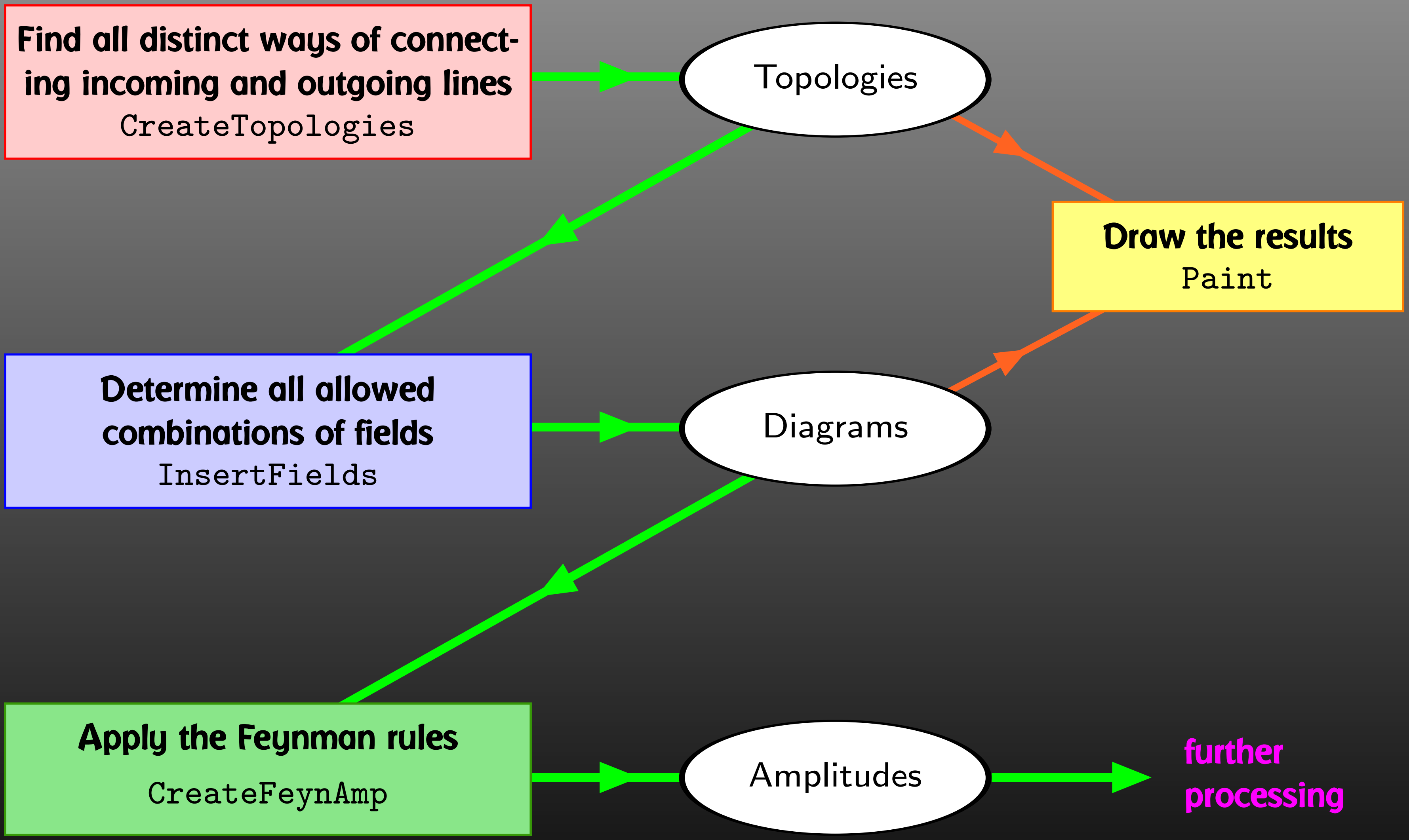
- Convert Mathematica output to Fortran code
- Supply a driver program
- Implementation of the integrals

Symbolic manipulation (Computer Algebra) for the structural and algebraic operations.

Compiled high-level language (Fortran) for the numerical evaluation.

$|\mathcal{M}|^2 \longrightarrow$ **Cross-sections, Decay rates, . . .**

FeynArts


Find all distinct ways of connecting incoming and outgoing lines
CreateTopologies
Topologies
Draw the results
Paint
Determine all allowed combinations of fields
InsertFields
Diagrams
Apply the Feynman rules
CreateFeynAmp
Amplitudes
further processing

Three Levels of Fields

Generic level, e.g. `F, F, S`

$$\boxed{C(F_1, F_2, S) = G_L \mathbb{P}_L + G_R \mathbb{P}_R} \qquad \mathbb{P}_{R,L} = (\mathbb{1} \pm \gamma_5)/2$$

Kinematical structure completely fixed, most algebraic simplifications (e.g. tensor reduction) can be carried out.

Classes level, e.g. `-F[2], F[1], S[3]`

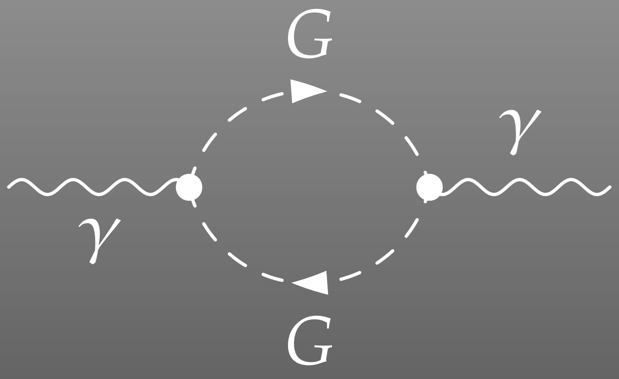
$$\boxed{\bar\ell_i \nu_j G: \quad G_L = -\frac{\mathrm{i}\, e\, m_{\ell,i}}{\sqrt{2} \sin\theta_w M_W} \delta_{ij}\,, \quad G_R = 0}$$

Coupling fixed except for i, j (can be summed in do-loop).

Particles level, e.g. `-F[2,{1}], F[1,{1}], S[3]`

$\boxed{\textbf{insert fermion generation (1, 2, 3) for } i \textbf{ and } j}$

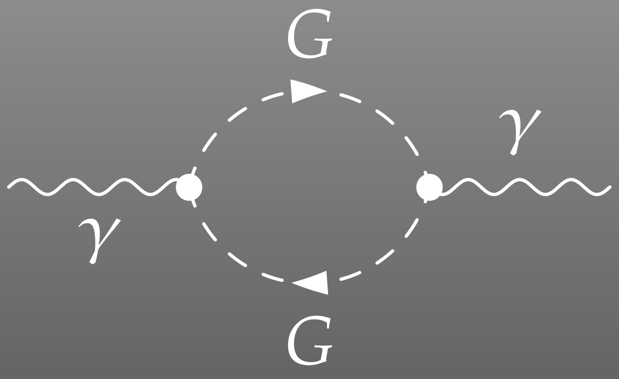
Sample CreateFeynAmp output



= `FeynAmp[` $\boxed{\textit{identifier}}$,
loop momenta,
generic amplitude,
insertions `]`

```
GraphID[Topology == 1, Generic == 1]
```

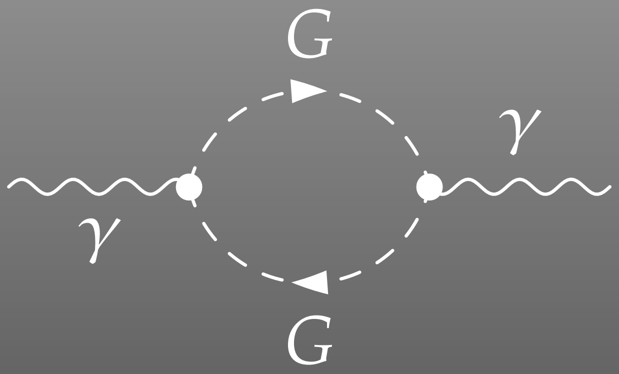
Sample CreateFeynAmp output



= `FeynAmp[` *identifier* ,
$\boxed{\textit{loop momenta}}$,
generic amplitude ,
insertions `]`

```
Integral[q1]
```

Sample CreateFeynAmp output



= FeynAmp[*identifier*,
loop momenta,
$\boxed{\textit{generic amplitude}}$,
insertions]

$\frac{\texttt{I}}{\texttt{32 Pi}^4}$ `RelativeCF` ....*prefactor*

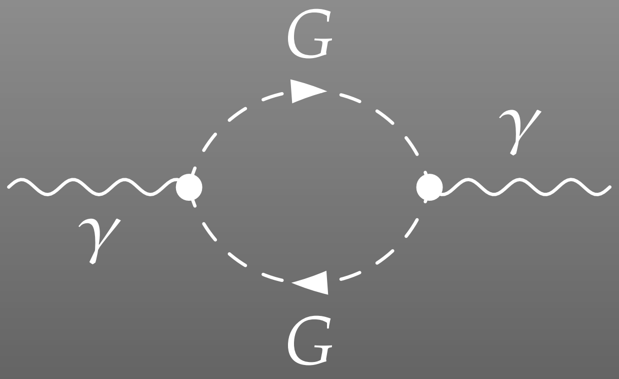
$\texttt{FeynAmpDenominator}\Big[\frac{1}{\texttt{q1}^2 - \texttt{Mass[S[Gen3]]}^2},$
$\frac{1}{\texttt{(-p1 + q1)}^2 - \texttt{Mass[S[Gen4]]}^2}\Big]$ .... *loop denominators*

`(p1 - 2 q1)[Lor1] (-p1 + 2 q1)[Lor2]` .... *kin. coupling structure*

$\texttt{ep[V[1], p1, Lor1] ep}^*\texttt{[V[1], k1, Lor2]}$ ....*polarization vectors*

$\texttt{G}^{(0)}_{\texttt{SSV}}\texttt{[(Mom[1] - Mom[2])[KI1[3]]]}$
$\texttt{G}^{(0)}_{\texttt{SSV}}\texttt{[(Mom[1] - Mom[2])[KI1[3]]]},$ .... *coupling constants*

Sample CreateFeynAmp output



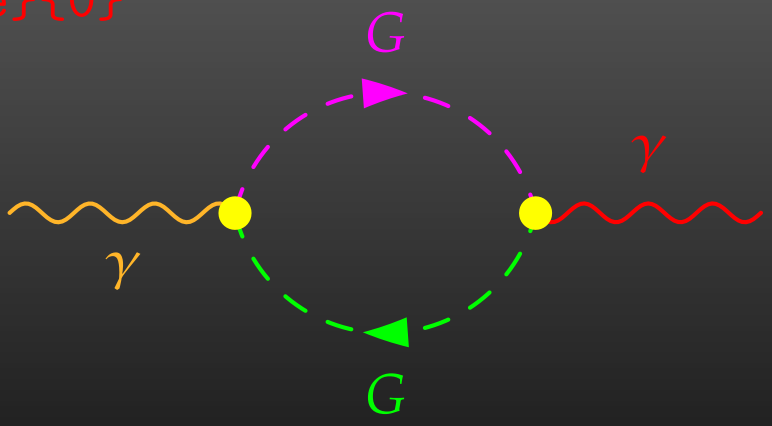
= FeynAmp[*identifier*,
loop momenta,
generic amplitude,
$\boxed{insertions}$]

```
{ Mass[S[Gen3]],
  Mass[S[Gen4]],
```
$G^{(0)}_{SSV}$[(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}]
```

Sample Paint output

```
\begin{feynartspicture}(150,150)(1,1)
\FADiagram{}
\FAProp(6.,10.)(14.,10.)(0.8,){ScalarDash}{-1}
\FALabel(10.,5.73)[t]{$G$}
\FAProp(6.,10.)(14.,10.)(-0.8,){ScalarDash}{1}
\FALabel(10.,14.27)[b]{$G$}
\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]{$\gamma$}
\FAVert(6.,10.){0}
\FAVert(14.,10.){0}
\end{feynartspicture}
```



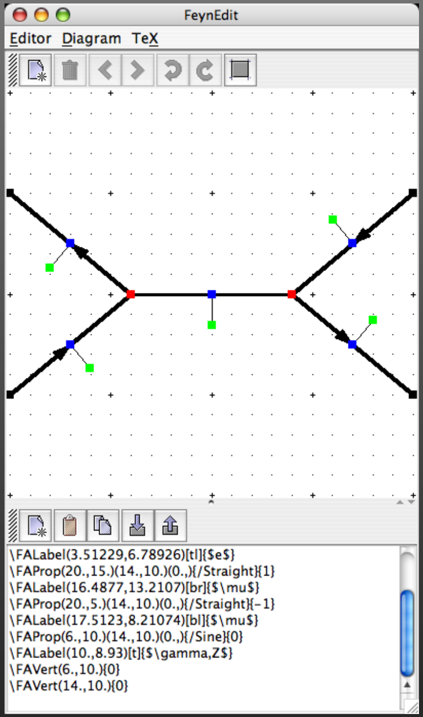
Technically: uses its own PostScript prologue.

Editing Feynman Diagrams

The elements of the diagram are easy to recognize and it is straightforward to **make changes e.g. to the label text** using any text editor.
It is less straightforward, however, to alter the geometry of the diagram, i.e. to **move vertices and propagators.**

The **FeynEdit** tool lets the user:

- copy-and-paste the LaTeX code into the lower panel of the editor,
- visualize the diagram,
- modify it using the mouse, and finally
- copy-and-paste it back into the text.



```
\FALabel(3.51229,6.78926)[tl]{$e$}
\FAProp(20.,15.)(14.,10.)(0.,){/Straight}{1}
\FALabel(16.4877,13.2107)[br]{$\mu$}
\FAProp(20.,5.)(14.,10.)(0.,){/Straight}{-1}
\FALabel(17.5123,8.21074)[bl]{$\mu$}
\FAProp(6.,10.)(14.,10.)(0.,){/Sine}{0}
\FALabel(10.,8.93)[t]{$\gamma,Z$}
\FAVert(6.,10.){0}
\FAVert(14.,10.){0}
```

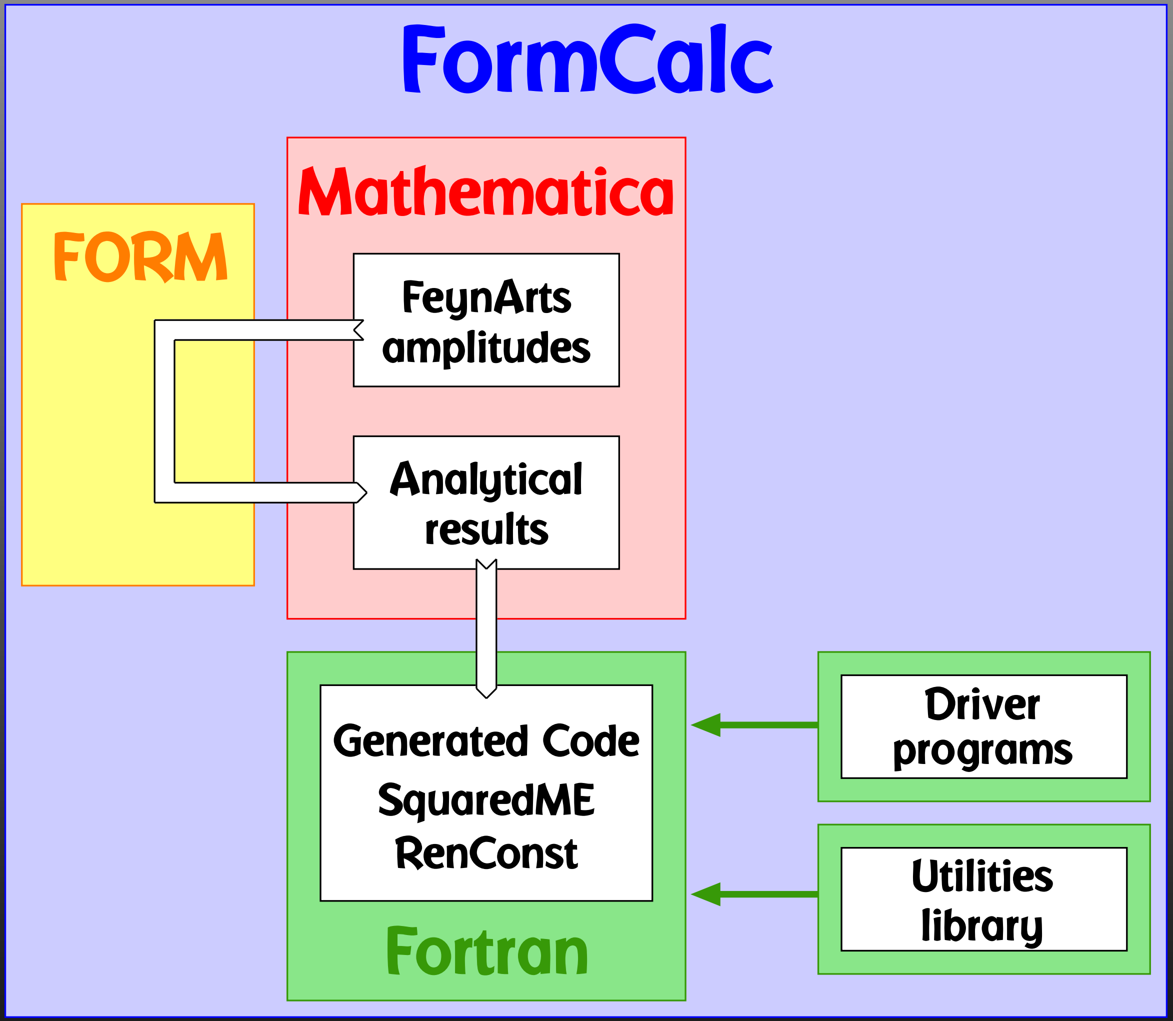
Algebraic Simplification

The amplitudes of `CreateFeynAmp` are in no good shape for direct numerical evaluation.

A number of steps have to be done analytically:

- contract indices as far as possible,
- evaluate fermion traces,
- perform the tensor reduction / separate numerators,
- add local terms arising from D·(divergent integral),
- simplify open fermion chains,
- simplify and compute the square of SU(N) structures,
- "compactify" the results as much as possible.

FormCalc Internals



FormCalc Output

A typical term in the output looks like

```
C0i[cc12, MW2, MW2, S, MW2, MZ2, MW2] *
  ( -4 Alfa2 MW2 CW2/SW2 S AbbSum16 +
    32 Alfa2 CW2/SW2 S^2 AbbSum28 +
    4 Alfa2 CW2/SW2 S^2 AbbSum30 -
    8 Alfa2 CW2/SW2 S^2 AbbSum7 +
    Alfa2 CW2/SW2 S(T-U) Abb1 +
    8 Alfa2 CW2/SW2 S(T-U) AbbSum29 )
```

🟩	= loop integral	🟥	= kinematical variables
🟪	= constants	🟨	= automatically introduced abbreviations

Abbreviations

Outright factorization is usually out of question.
Abbreviations are necessary to reduce size of expressions.

```
AbbSum29 = Abb2 + Abb22 + Abb23 + Abb3

        Abb22 = Pair1 Pair3 Pair6

              Pair3 = Pair[e[3],k[1]]
```

The full expression corresponding to `AbbSum29` is

```
Pair[e[1],e[2]] Pair[e[3],k[1]] Pair[e[4],k[1]] +
Pair[e[1],e[2]] Pair[e[3],k[2]] Pair[e[4],k[1]] +
Pair[e[1],e[2]] Pair[e[3],k[1]] Pair[e[4],k[2]] +
Pair[e[1],e[2]] Pair[e[3],k[2]] Pair[e[4],k[2]]
```

Categories of Abbreviations

- Abbreviations are recursively defined in several levels.
- When generating code, FormCalc introduces another set of abbreviations for the loop integrals.

In general, the abbreviations are thus costly in CPU time.
It is key to a decent performance that the abbreviations are separated into different Categories:

- Abbreviations that depend on the helicities,
- Abbreviations that depend on angular variables,
- Abbreviations that depend only on $\sqrt{s}$.

Correct execution of the categories guarantees that almost no redundant evaluations are made and makes the generated code essentially as fast as hand-tuned code.

External Fermion Lines

An amplitude containing **external fermions** has the form

$$\mathcal{M} = \sum_{i=1}^{n_F} c_i\, F_i \quad \text{where} \quad F_i = \text{(Product of)}\,\langle u|\,\Gamma_i\,|v\rangle\,.$$

n_F = number of fermionic structures.

Textbook procedure: **Trace Technique**

$$|\mathcal{M}|^2 = \sum_{i,j=1}^{n_F} c_i^*\, c_j\, F_i^* F_j$$

where $F_i^* F_j = \langle v|\,\bar\Gamma_i\,|u\rangle\,\langle u|\,\Gamma_j\,|v\rangle = \mathrm{Tr}\big(\bar\Gamma_i\;|u\rangle\langle u|\;\Gamma_j\;|v\rangle\langle v|\big)\,.$

Problems with the Trace Technique

PRO: Trace technique is independent of any representation.

CON: For n_F F_i's there are n_F^2 $F_i^* F_j$'s.

Things get worse the more vectors are in the game: multi-particle final states, polarization effects . . .

Essentially $n_F \sim (\#$ of vectors$)!$ because all combinations of vectors can appear in the Γ_i.

Solution: Use Weyl-van der Waerden spinor formalism to compute the F_i's directly.

Fermion Chains

FormCalc uses Dirac (4-component) spinors in most of the algebra (extension to D dim more obvious).

Move to 2-comp. Weyl spinors for the numerical evaluation:

$$\langle u|_4 \equiv \big(\langle u_+|_2\,, \langle u_-|_2\big), \qquad |v\rangle_4 \equiv \begin{pmatrix} |v_-\rangle_2 \\ |v_+\rangle_2 \end{pmatrix}.$$

Every chiral Dirac chain maps onto a *<u>single</u>* Weyl chain:

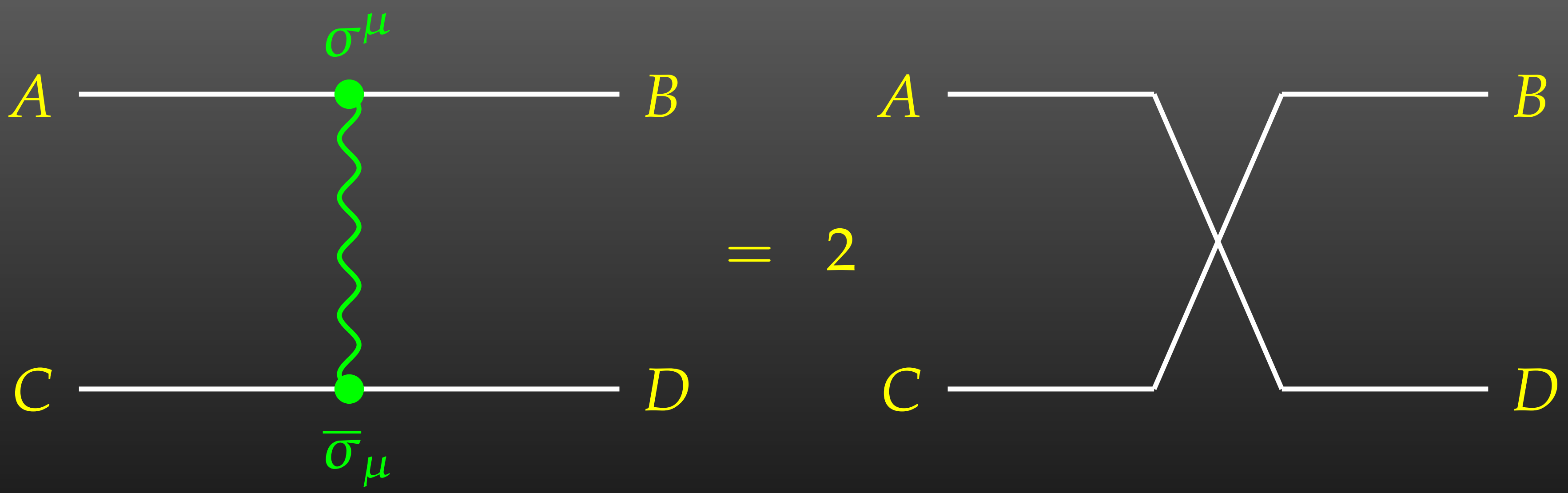
$$\begin{aligned} \langle u|\,\mathbb{P}_L\,\gamma_\mu\gamma_\nu\cdots|v\rangle_4 &= \langle u_-|\,\overline{\sigma}_\mu\sigma_\nu\cdots|v_\pm\rangle_2\,, \\ \langle u|\,\mathbb{P}_R\,\gamma_\mu\gamma_\nu\cdots|v\rangle_4 &= \langle u_+|\,\sigma_\mu\overline{\sigma}_\nu\cdots|v_\mp\rangle_2\,. \end{aligned}$$

FORM-like notation: $\langle u|\,\sigma_\mu\overline{\sigma}_\nu\sigma_\rho\,|v\rangle\,k_1^\mu\varepsilon_2^\nu k_3^\rho \equiv \langle u|\,k_1\overline{\varepsilon}_2k_3\,|v\rangle\,.$

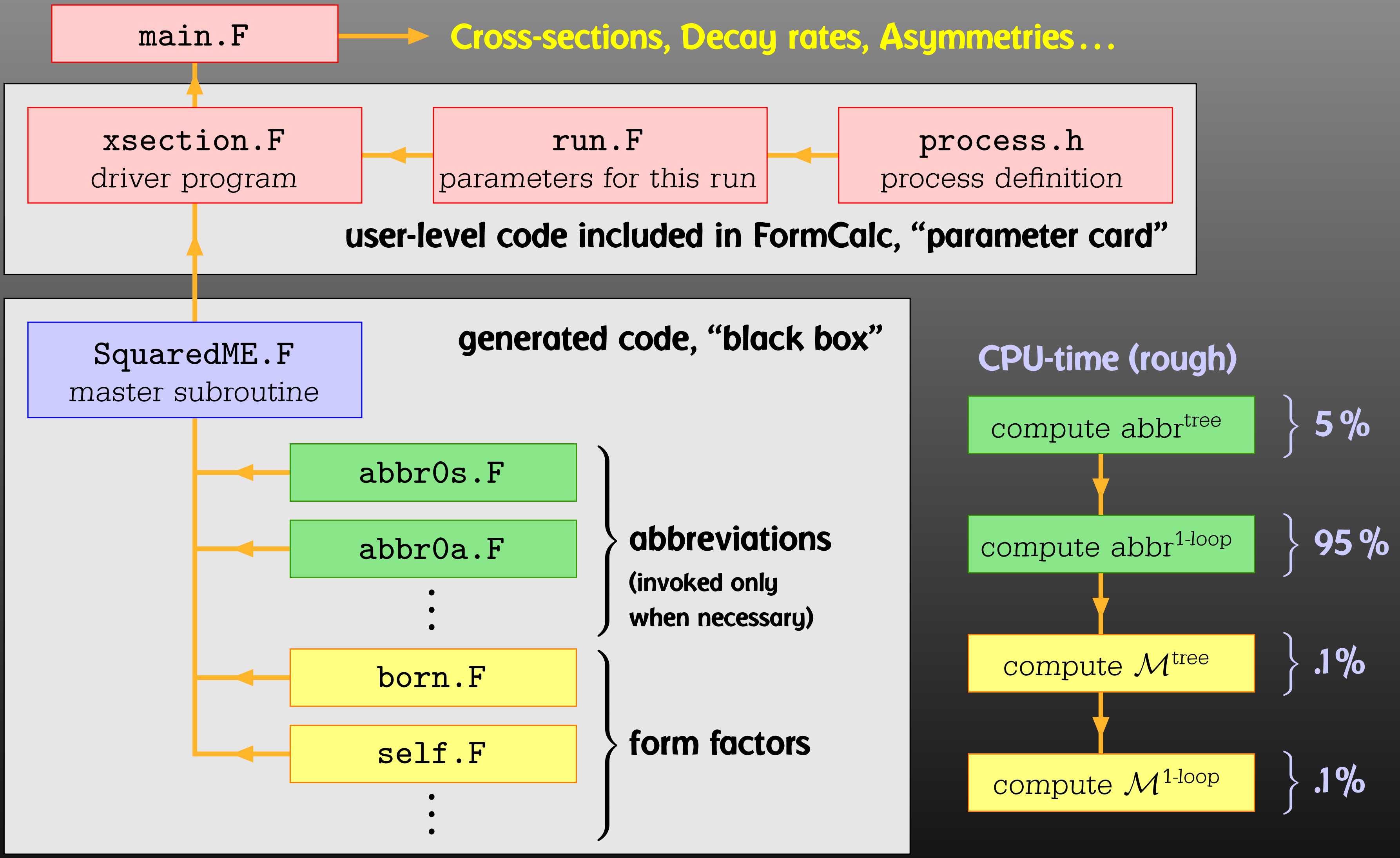
Fierz Identities

With the Fierz identities for sigma matrices it is possible to remove all Lorentz contractions between sigma chains, e.g.

$$\langle A|\,\sigma_\mu\,|B\rangle\ \langle C|\,\overline{\sigma}^\mu\,|D\rangle \;=\; 2\,\langle A|D\rangle\ \langle C|B\rangle$$



Numerical Evaluation


main.F
Cross-sections, Decay rates, Asymmetries…
xsection.F
driver program
run.F
parameters for this run
process.h
process definition
user-level code included in FormCalc, “parameter card”
SquaredME.F
master subroutine
generated code, “black box”
abbr0s.F
abbr0a.F
abbreviations
(invoked only
when necessary)
born.F
self.F
form factors
CPU-time (rough)
compute abbr$^{\text{tree}}$
5 %
compute abbr$^{\text{1-loop}}$
95 %
compute $\mathcal{M}^{\text{tree}}$
.1%
compute $\mathcal{M}^{\text{1-loop}}$
.1%

Mathematica Interface

The Mathematica Interface turns the generated stand-alone Fortran code into a Mathematica function for evaluating the cross-section or decay rate as a function of user-selected model parameters. Think of:

```
ContourPlot[sigma[TB, MA0], {TB, 5}, {MA0, 250}]
```

Changes in code (run.F):

```
TB = 5     →     call MmaGetReal(TB)
```

Compile, load in Mathematica with

```
Install["run"]
```

Compute e.g. a differential cross-section at $\sqrt{s} = \mathtt{sqrtS}$:

```
run[sqrtS, TB, MA0, ...]
```

The Model Files

One has to set up, once and for all, a

- **Generic Model File** (seldomly changed) containing the generic part of the couplings,

Example: the `FFS` coupling

$$C(F,F,S) = G_L \mathbb{P}_L + G_R \mathbb{P}_R = \vec{G} \cdot \begin{pmatrix} \mathbb{P}_L \\ \mathbb{P}_R \end{pmatrix}$$

```
AnalyticalCoupling[s1 F[j1, p1], s2 F[j2, p2], s3 S[j3, p3]]
== G[1][s1 F[j1], s2 F[j2], s3 S[j3]] .
     { NonCommutative[ ChiralityProjector[-1] ],
       NonCommutative[ ChiralityProjector[+1] ] }
```

The Model Files

One has to set up, once and for all, a

- **Classes Model File** (for each model)
  declaring the particles and the allowed couplings

Example: the $\bar\ell_i \nu_j G$ coupling in the Standard Model

$$\vec{G}(\bar\ell_i, \nu_j, G) = \begin{pmatrix} G_- \\ G_+ \end{pmatrix} = \begin{pmatrix} -\dfrac{\mathrm{i}\, e\, m_{\ell,i}}{\sqrt{2}\sin\theta_w M_W}\delta_{ij} \\ 0 \end{pmatrix}$$

```
C[ -F[2,{i}], F[1,{j}], S[3] ]
== { {-I EL Mass[F[2,{i}]]/(Sqrt[2] SW MW) IndexDelta[i, j]},
     {0} }
```

Included Model Files

Model Files presently available for FeynArts:

- SM [w/QCD], normal and background-field version.
  All one-loop counter terms included.
- MSSM [w/QCD].
  All one-loop counter terms included.
- Two-Higgs-Doublet Model.
  Counter terms not included yet.

- ModelMaker utility generates Model Files from the Lagrangian.
- "3rd-party packages" FeynRules and LanHEP generate Model Files for FeynArts and others.
- SARAH package derives SUSY Models.

Partial (Add-On) Model Files

FeynArts distinguishes

- Basic Model Files and
- Partial (Add-On) Model Files.

Basic Model Files, e.g. `SM.mod`, `MSSM.mod`, can be modified by Add-On Model Files. For example,

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

This loads the Basic Model File `MSSMQCD.mod` and modifies it through the Add-On `FV.mod` (non-minimal flavour violation).

Model files can thus be built up from several parts.

Tweaking Model Files

Or, How to efficiently make changes in an existing model file.

Bad: Copy the model file, modify the copy. – Why?

- It is typically not very transparent what has changed.
- If the original model file changes (e.g. bug fixes), these do not automatically propagate into the derivative model file.

Better: Create a new model file which reads the old one and modifies the particles and coupling tables.

- `M$ClassesDescription` = list of particle definitions,
- `M$CouplingMatrices` = list of couplings.

Tweaking Model Files

Example: Introduce enhancement factors for the b-$\bar{b}$-h_0 and b-$\bar{b}$-H_0 Yukawa couplings in the MSSM.

```
EnhCoup[ (lhs:C[F[4,{g_,_}], -F[4,_], S[h:1|2]]) == rhs_ ] :=
  lhs == Hff[h,g] rhs
EnhCoup[other_] := other

M$CouplingMatrices = EnhCoup/@ M$CouplingMatrices
```

To see the effect, make a printout with the `WriteTeXFile` utility of FeynArts.

The `Hff[h,g]` can be defined to include e.g. resummation effects, as in

```
double precision Hff(2,3)
data Hff /6*1/
Hff(1,3) = 1 - CA/(SA*TB)*Delta_b
Hff(2,3) = 1 + SA/(CA*TB)*Delta_b
```

Linear Combinations of Fields

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]} }
```

Since `F[111]...F[114]` are not listed in `M$CouplingMatrices`, they drop out of the model completely.

Linear Combinations of Fields

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 the coupling list (including all mixing couplings) because all three are listed in `M$CouplingMatrices`.

Due to the `InsertOnly`, `S[10]` is inserted only on tree-level parts of the diagram, not in loops.

Not the Cross-Section

Or, How to get things the Standard Setup won't give you.

Example: extract the Wilson coefficients for $b \to s\gamma$.

```
tops = CreateTopologies[1, 1 -> 2]
ins = InsertFields[tops, F[4,{3}] -> {F[4,{2}], V[1]}]
vert = CalcFeynAmp[CreateFeynAmp[ins], FermionChains -> Chiral]

mat[p_Plus] := mat/@ p

mat[r_. DiracChain[s2_Spinor, om_, mu_, s1:Spinor[p1_, m1_, _]]] :=
  I/(2 m1) mat[r DiracChain[sigmunu[om]]] +
  2/m1 r Pair[mu, p1] DiracChain[s2, om, s1]

mat[r_. DiracChain[sigmunu[om_]], SUNT[Col1, Col2]] :=
  r O7[om]/(EL MB/(16 Pi^2))

mat[r_. DiracChain[sigmunu[om_]], SUNT[Glu1, Col2, Col1]] :=
  r O8[om]/(GS MB/(16 Pi^2))

coeff = Plus@@ vert //. abbr /. Mat -> mat

c7 = Coefficient[coeff, O7[6]]
c8 = Coefficient[coeff, O8[6]]
```

Not the Cross-Section

Using FormCalc's output functions it is also pretty straightforward to generate your own Fortran code:

```
file = OpenFortran["bsgamma.F"]

WriteString[file,
  SubroutineDecl["bsgamma(C7,C8)"] <>
  "\tdouble complex C7, C8\n" <>
  "#include \"looptools.h\"\n"]

WriteExpr[file, {C7 -> c7, C8 -> c8}]

WriteString[file, "\tend\n"]

Close[file]
```

More details in hep-ph/0607049.

More Complex Calculations

Often special requirements:

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

Software design so far:

- Mostly 'monolithic' (one package does everything).
- Often controlled by parameter cards, not easy to use beyond intended purpose.
- May want to/must use other packages.

Universality w.r.t. Loop Order

✔ **Model $\rightarrow$ Model file $\rightarrow$ Diagrams** — Diagram Generation

✔ **Fermion algebra (traces, Dirac eq)** — Symbolic Simplification

✔ **Color algebra ($\mathrm{SU}(N)$ traces)**

✘ **Perform tensor reduction, *or***
Isolate integrals for OPP

✔ **Other simplifications (e.g. Fierz, abbreviations)**

✔ **Code generation (Fortran, C/++)** — Numerical Evaluation

✓ **Phase-space integration**

✘ **Evaluation of loop integrals**

Example: $\mathcal{O}(\alpha_t^2)$ MSSM Higgs-mass corrections

Hollik, Paßehr 2014

Template for calculation (2L, nontrivial model + renorm.):

- Break calculation into several steps.
- Implement each step as independent program (invoked from command line).
- In lieu of ‘in vivo’ debugging keep detailed logs.
- Coordinate everything through a makefile.
- No single control program (e.g. single Mathematica session) like in package’s demo programs.

More details in arXiv:1508.00562.

Scripting Mathematica

Efficient batch processing with Mathematica:

Put everything into a script, using sh's Here documents:

```
#! /bin/sh ................. Shell Magic
math << \_EOF_ ........... start Here document (note the \)
  << FeynArts`
  << FormCalc`
  top = CreateTopologies[...];
  ...
_EOF_ ...................... end Here document
```

Everything between "<< \*tag*" and "*tag*" goes to Mathematica as if it were typed from the keyboard.

Note the "\\" before *tag*, it makes the shell pass everything literally to Mathematica, without shell substitutions.

Steps of the Calculation

Calculation split into 7 (8) steps:

$$\begin{tikzcd}
\text{FeynArts} \arrow[r, Rightarrow] & \underset{\text{diagram generation}}{\boxed{\texttt{1-amps}}} \arrow[r] & \underset{\substack{\text{preparation for}\\ \text{tensor reduction}}}{\boxed{\texttt{2-prep}}} \arrow[r] & \underset{\text{tensor reduction}}{\boxed{\texttt{3-calc}}} \arrow[d] & \begin{array}{l}\text{TwoCalc}\\ \text{FormCalc}\end{array} \arrow[l, Rightarrow, shift left] \arrow[l, Rightarrow, shift right] \\
 & \underset{\text{model file preparation}}{\boxed{\texttt{0-glmod}}} \arrow[u] & \text{MSSMCT.mod} \arrow[l, Rightarrow] & \underset{\text{simplification}}{\boxed{\texttt{4-simp}}} \arrow[d] & \\
\text{FormCalc} \arrow[r, Rightarrow] & \underset{\text{code generation}}{\boxed{\texttt{7-code}}} & \underset{\text{combination of results}}{\boxed{\texttt{6-comb}}} \arrow[l] & \underset{\substack{\text{calculation of}\\ \text{renorm. constants}}}{\boxed{\texttt{5-rc}}} \arrow[l] & \text{FormCalc} \arrow[l, Rightarrow]
\end{tikzcd}$$

Example: Step 0 – Take gaugeless limit

Gaugeless approximation:

① Set gauge couplings $g, g' = 0 \Rightarrow M_W, M_Z = 0$.

② Keep finite weak mixing angle.

③ Keep $\frac{\delta M_W^2}{M_W^2}$ and $\frac{\delta M_Z^2}{M_Z^2}$ finite.

Must set $m_b = 0$ so that $\mathcal{O}(\alpha_t^2)$ corrections form supersymmetric and gauge-invariant subset.

Most efficient to modify Feynman rules:

- Load `MSSMCT.mod` model file.
- Modify couplings, remove zero ones.
- Write out `MSSMCTgl.mod` model file.

Finally

- There are many packages for tree-level and increasingly also 1L calculations available.
- For **‘standard tasks’** (e.g. cross-section computation) largely automated “model to events” **toolchains exist.**
- Other tasks requiring evaluation of Feynman diagrams are not so well automated (and may never be).
- Packages like FeynArts, FormCalc, FeynCalc, Package-X provide an **“exploration toolkit” for unusual models, unusual renormalizations, package building, . . .**
- Long-term strategy: maybe best use Unix philosophy **“Do one thing and do it well”** – modular components for individual tasks + stick together by script.