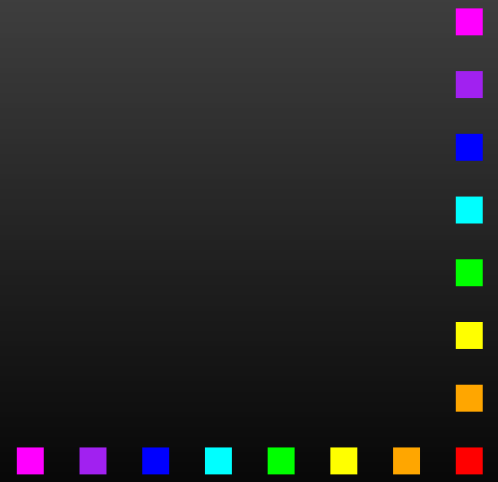
The FeynSystem: FeynArts, FormCalc, LoopTools

Thomas Hahn

Max-Planck-Institut für Physik
München



Automated Diagram Evaluation

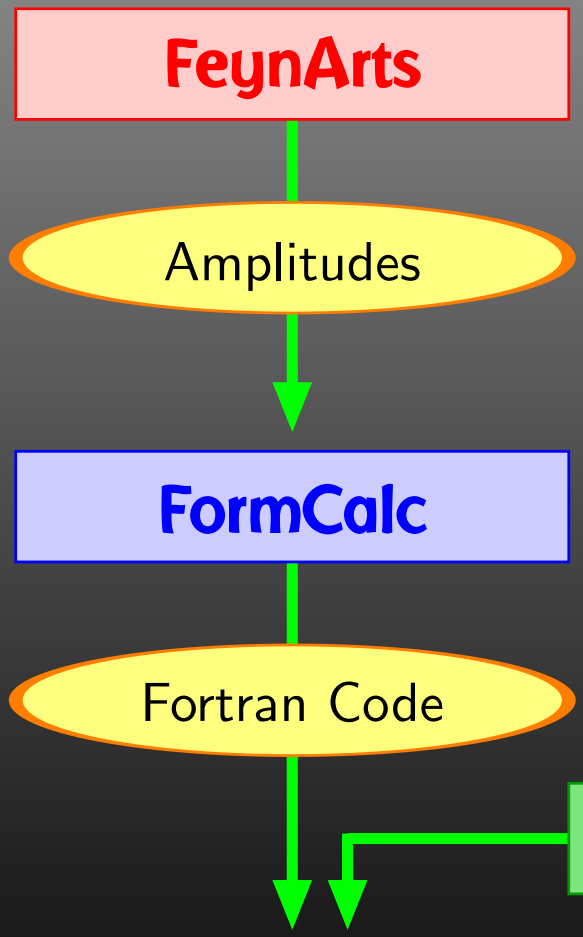


Diagram Generation:

- 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:

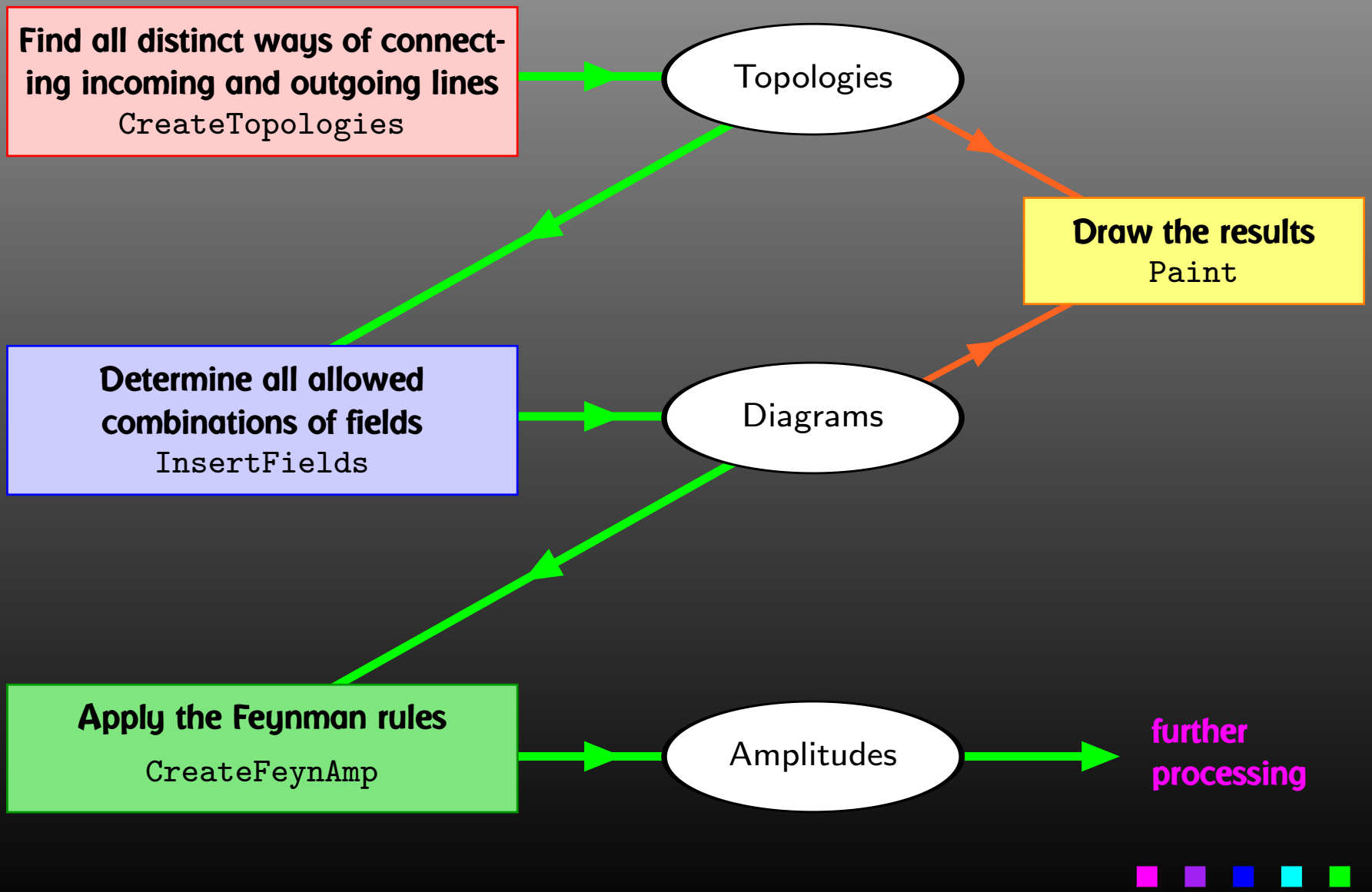
- Convert Mathematica output to Fortran code
- Supply a driver program
- Implementation of the integrals

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

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 an add-on model file which 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
```

- **ModelMaker utility (part of FeynArts) generates Model Files from the Lagrangian.**
- **FeynRules package generates Model Files for FeynArts and other packages.**
- **SARAH package derives SUSY Models.**

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.**

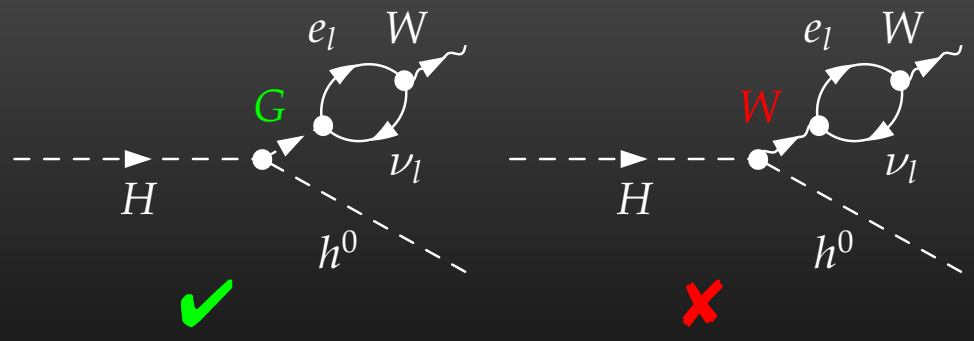
Enhanced Diagram Selection

FeynArts has easier way to pick wave-function corrections:

```
CreateTopologies[ ...
  ExcludeTopologies -> WFCorrections[1|3] ]
```

Select only those where the in- and out-fields of the self-energy are not the same:

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

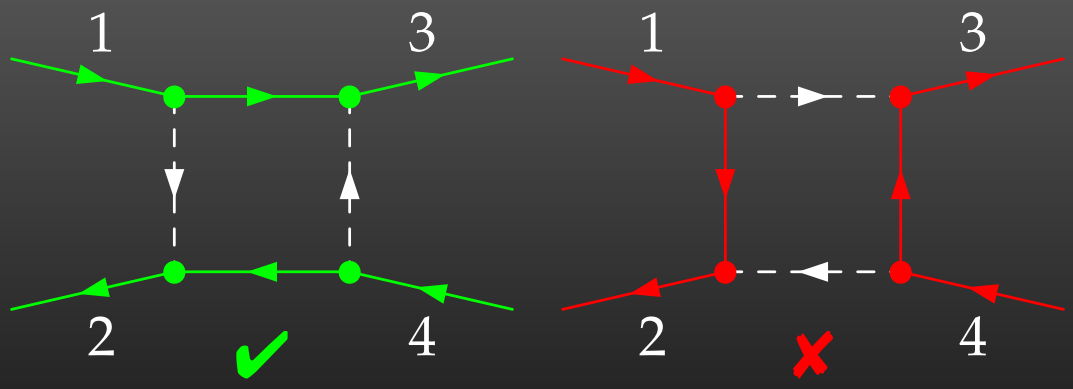


Enhanced Diagram Selection

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

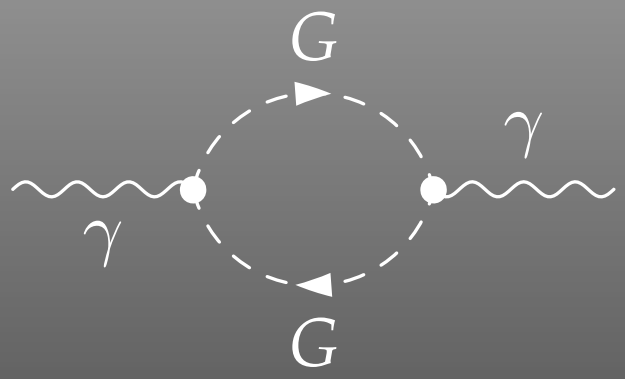
```
DiagramSelect[...,
   FermionRouting[##] === {1,3, 2,4} & ]
```

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



More Functions: `DiagramGrouping`, `DiagramMap`, `DiagramComplement`.
More Filters: `Vertices`, `FieldPoints`, `FeynAmpCases`, `FieldMatchQ`, `FieldMemberQ`, `FieldPointMatchQ`, `FieldPointMemberQ`.

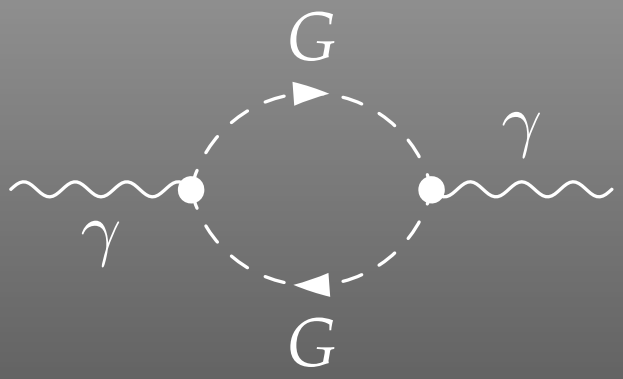
Sample CreateFeynAmp output



= FeynAmp[*identifier* ,
loop momenta ,
generic amplitude ,
insertions]

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

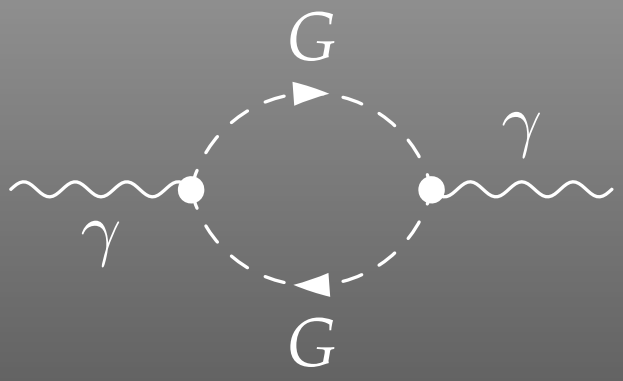
Sample CreateFeynAmp output



= FeynAmp[*identifier*,
loop momenta,
generic amplitude,
insertions]

```
Integral[q1]
```

Sample CreateFeynAmp output



= FeynAmp[*identifier*, *loop momenta*, *generic amplitude*, *insertions*]

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

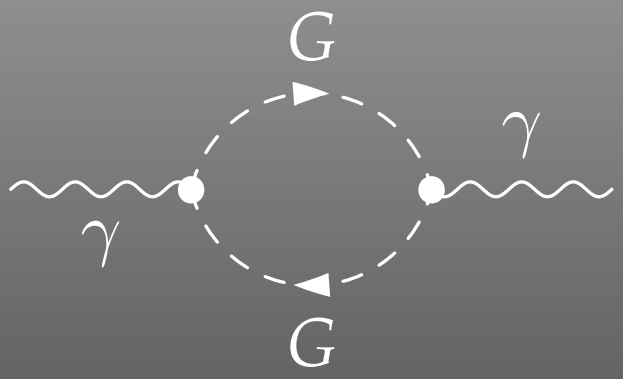
FeynAmpDenominator[$\frac{1}{\text{q1}^2 - \text{Mass[S[Gen3]]}^2}$, $\frac{1}{(-\text{p1} + \text{q1})^2 - \text{Mass[S[Gen4]]}^2}$] ..... *loop denominators*

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

ep[V[1], p1, Lor1] ep*[V[1], k1, Lor2] ..... *polarization vectors*

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

Sample CreateFeynAmp output



= FeynAmp[*identifier*,
loop momenta,
generic amplitude,
insertions]

```
{ Mass[S[Gen3]],
  Mass[S[Gen4]],
  G_SSV^(0)[(Mom[1] - Mom[2])[KI1[3]]],
  G_SSV^(0)[(Mom[1] - Mom[2])[KI1[3]]],
  RelativeCF } ->
Insertions[Classes][{MW, MW, I EL, -I EL, 2}]
```

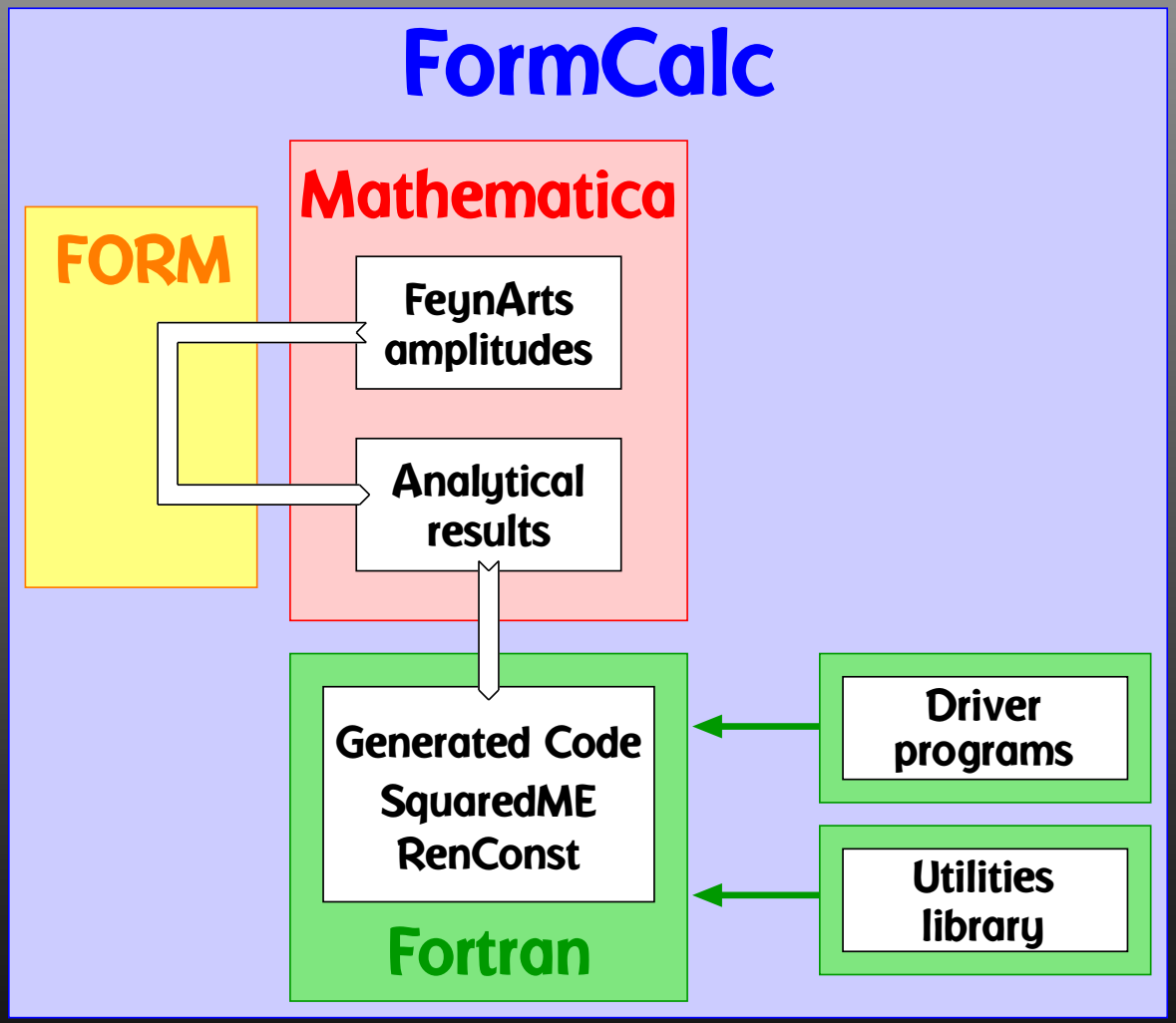
Algebraic Simplification

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,
- add local terms arising from D·(divergent integral) (dim reg + dim red),
- 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.

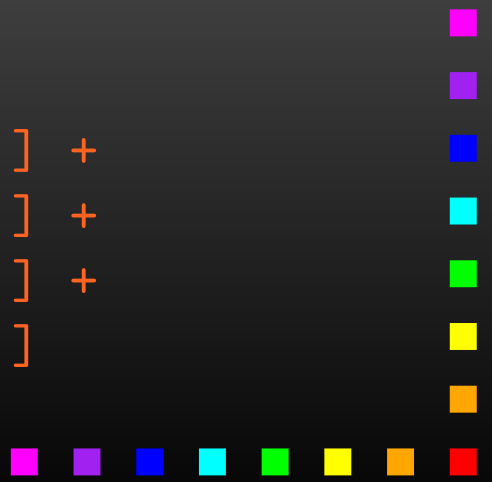
```
AbbSum29 = Abb2 + Abb22 + Abb23 + Abb3

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 Fortran 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.

More on Abbreviations

The `Abbreviate` Function allows to introduce abbreviations for arbitrary (sub-)expressions and extends the advantage of categorized evaluation.

The subexpressions are retrieved with `Subexpr[]`.

Abbreviations were so far restricted to one FormCalc session, 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[abbr]
RegisterSubexpr[subexpr]
```

Dirac Chains in 4D

As numerical calculations are done mostly 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, Fierz methods have been implemented for Dirac chains, thus allowing the user to force the fermion chains into almost any desired order.
- Version 6 further adds the `Colour` method to the `FermionOrder` option of `CalcFeynAmp`, which brings the spinors into the same order as the external colour indices.
- Also new in Version 6: completely antisymmetrized Dirac chains, i.e. `DiracChain[`$-1, \mu, \nu$`]` $= \sigma_{\mu\nu}$.

Alternate Link between FORM and Mathematica

FORM is able to handle very large expressions. To produce (pre-)simplified expressions, however, terms have to be wrapped in functions, to avoid immediate expansion:

```
a*(b + c)    →  a*b + a*c
a*f(b + c)   →  a*f(b + c)
```

The number of terms in a function is rather limited in FORM: on 32-bit systems to 32767.

Dilemma: FormCalc gets more sophisticated in pre-simplifying amplitudes while users want to compute larger amplitudes. Thus, recently many 'overflow' messages from FORM.

Solution: Send pre-simplified generic amplitude via external channel to Mathematica for introducing abbreviations. Significant reduction in size of intermediate expressions.

Tentukov, Vermaseren 2006

Effect on Intermediate Amplitudes

FORM $\longrightarrow$ Mathematica:

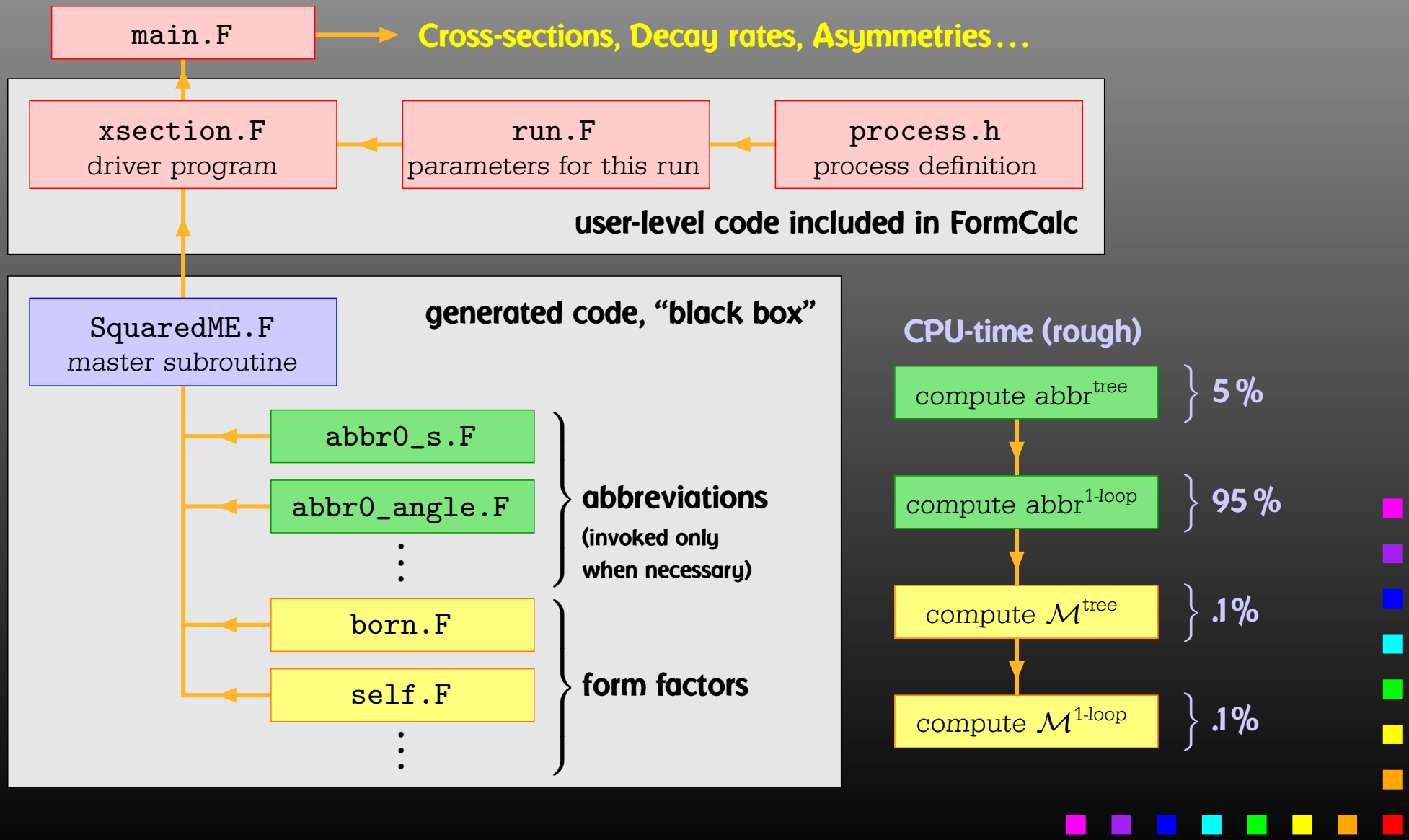
part of $uu \longrightarrow gg$ @ tree level

```
+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]]]] )
```

Mathematica $\longrightarrow$ FORM:

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

Numerical Evaluation in Fortran 77



Features of the Generated Code

- **Modular:** largely autonomous pieces of code provide
  - kinematics,
  - model initialization,
  - convolution with PDFs.
- **Extensible:** default code serves (only) as an example. Other 'Frontends' can be supplied, e.g. HadCalc, sofox.
- **Re-usable:** external program need only call `ProcessIni` (to set up the process) and `ParameterScan` (to set off the calculation).
- **Interactive:** Mathematica interface provides Mathematica function for cross-section/decay rate.
- **Parallel:** built-in distribution of parameter scans.

Code-generation Functions

FormCalc's code-generation functions are now public and disentangled from the rest of the code. They can be used to **write out an arbitrary Mathematica expression as optimized Fortran code:**

- *handle* = `OpenFortran["`*file.F*`"]`
  opens *file.F* as a Fortran file for writing,
- `WriteExpr[`*handle*`, {`*var* `->` *expr*`, ...}]`
  writes out Fortran code which calculates *expr* and stores the result in *var*,
- `Close[`*handle*`]`
  closes the file again.

Code generation

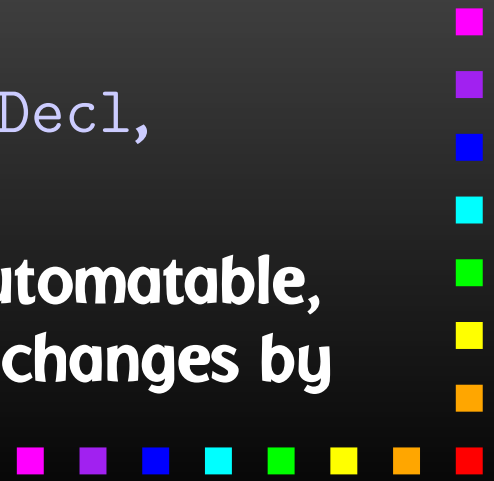
- **Expressions too large** for Fortran are split into parts, as in

```
var = part1
var = var + part2
...
```

- **High level of optimization,** e.g. common subexpressions are pulled out and computed in temporary variables.
- **Many ancillary functions,** e.g.

  `PrepareExpr`, `OnePassOrder`, `SplitSums`, `$SymbolPrefix`, `CommonDecl`, `SubroutineDecl`, **etc.**

make code generation versatile and highly automatable, such that the resulting code needs few or no changes by hand.



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:

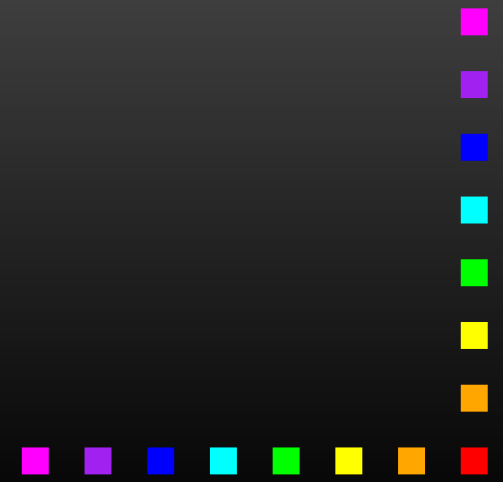
```
file = OpenFortran["bsgamma.F"]

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]
```



Choice of Language

Mentioning Fortran 77 as the programming language in many cases draws a "Not that dinosaur again" response.
But consider:

- Fortran was designed more or less exactly for 'number crunching,' i.e. efficient evaluation of large formulas.
- Good and free compilers are available.
- Fortran is still widely used in theoretical physics.
- The code is generated, so largely 'invisible' for the user.
- Linking Fortran 77 to C/C++ is pretty straightforward (particularly inside gcc), so is in some sense a 'smallest common denominator.'

LoopTools

LoopTools is a library for the one-loop integrals. It is based on FF and has a Fortran, C/C++, and Mathematica interface.

- **D0 for complex masses added.**

  Le, Dao 2009

- **Dim.reg. IR/collinear cases (QCDLoop) added.**
  Scalar integrals only so far (as in QCDLoop).

$\lambda^2 > 0$	regularization with photon 'mass' λ,	$\lambda^2 = -2$	coefficient of $1/\varepsilon^2$ in dim.reg.,
		$\lambda^2 = -1$	coefficient of $1/\varepsilon$ in dim.reg.,
		$\lambda^2 = 0$	finite piece in dim.reg.

  Ellis, Zanderighi 2008

- **New dispatcher for IR and collinear divergences.**
  Construct bit pattern: 1 for zero argument, 0 otherwise, then a single table lookup leads to correct case.

LoopTools Environment Variables

Most LoopTools parameters can be set from the outside through environment variables:

`LTCMPBITS`	# of bits compared in cache lookups
`LTVERSION`	bit mask for alternate versions
`LTMAXDEV`	maximum allowed relative deviation in comparing to alternate versions
`LTDEBUG`	bit mask for debugging
`LTRANGE`	range of integrals to print out in debug mode
`LTWARN`	number of digits lost before warning
`LTERR`	number of digits lost before error
`LTDELTA`	'divergence' Δ
`LTMUDIM`	renormalization scale μ^2
`LTLAMBDA`	IR regulator parameter λ^2
`LTMINMASS`	threshold m^2_{min} below which particles are considered 'massless'

E.g. check finiteness without re-compilation by modifying `LTMUDIM`, `LTLAMBDA`.

Alternate Versions

For some functions **Alternate Versions** exist, most of which are based on an implementation by Denner. The user can choose **at run-time** which version to use, and whether checking is performed. This is determined by the **Version Key:**

`0 * key`	compute version 'a' (mostly FF),
`1 * key`	compute version 'b' (mostly Denner),
`2 * key`	compute both, compare, return 'a',
`3 * key`	compute both, compare, return 'b'.

Alternate versions are currently available for the following functions: `A0`, `Bget`, `C0`, `D0`, `D0C`, `Eget`, `EgetC`.

Example: `call setversionkey(2*KeyD0 + 3*KeyBget)`

Command-line Interface

The Command-line Interface is useful in particular for testing and debugging.

It lists the N-point scalar and tensor coefficients corresponding to the number of arguments, i.e. 3 arguments = B, 6 arguments = C, etc.

```
> lt 250000 6464.16 8315.38
 ====================================================
   FF 2.0, a package to evaluate one-loop integrals
 written by G. J. van Oldenborgh, NIKHEF-H, Amsterdam
 ====================================================
 for the algorithms used see preprint NIKHEF-H 89/17,
 'New Algorithms for One-loop Integrals', by G.J. van
 Oldenborgh and J.A.M. Vermaseren, published in
 Zeitschrift fuer Physik C46(1990)425.
 ====================================================
   p      =   250000.000000000
   m1     =   6464.16000000000
   m2     =   8315.38000000000
 bb0      = (-10.1569090105893,2.95011861955466)
 bb1      = (5.07382021909957,-1.46413667259555)
 bb00     = (165409.773493414,-54197.2752510472)
 bb11     = (-3.31323987482202,0.943436559119877)
 bb001    = (-81984.2510317697,26897.9754657431)
 bb111    = (2.43370306005361,-0.683409066031936)
 dbb0     = (-4.868613391538015E-006,7.903840552878544E-007)
 dbb1     = (2.434818091584023E-006,-4.359562268294025E-007)
 dbb00    = (0.880290172138776,-0.260350056737834)
 dbb11    = (-1.624235905363170E-006,4.122913473051599E-007)
 ====================================================

 total number of errors and warnings
 ===================================
 fferr: no errors
```

CutTools

Tensor loop integrals have in FormCalc so far been treated by Passarino-Veltman reduction only, e.g.

$$\frac{q_\mu q_\nu}{D_0 D_1} = g_{\mu\nu}\,\mathtt{B00}(p^2, m_1^2, m_2^2) + p_\mu p_\nu\,\mathtt{B11}(p^2, m_1^2, m_2^2)$$

where `B00` and `B11` are provided by LoopTools.

CutTools implements the cutting-technique-inspired OPP (Ossola, Papadopoulos, Pittau) method. It needs the numerator as a function of q which it can sample:

$$\mathtt{Bcut}(2, \mathtt{num}, p, m_1^2, m_2^2)$$

where $\mathtt{num} = q_\mu q_\nu$.

Independent way of checking LoopTools results.
Performance?

A Final Look

Using FeynArts, FormCalc, and LoopTools is a lot like driving a car:

- You have to decide where to go (this is often the hardest decision).
- You have to turn the ignition key, work gas and brakes, and steer.
- But you don't have to know, say, which valve has to open at which time to keep the motor running.
- On the other hand, you can only go where there are roads. You can't climb a mountain with your car.

feynarts.de