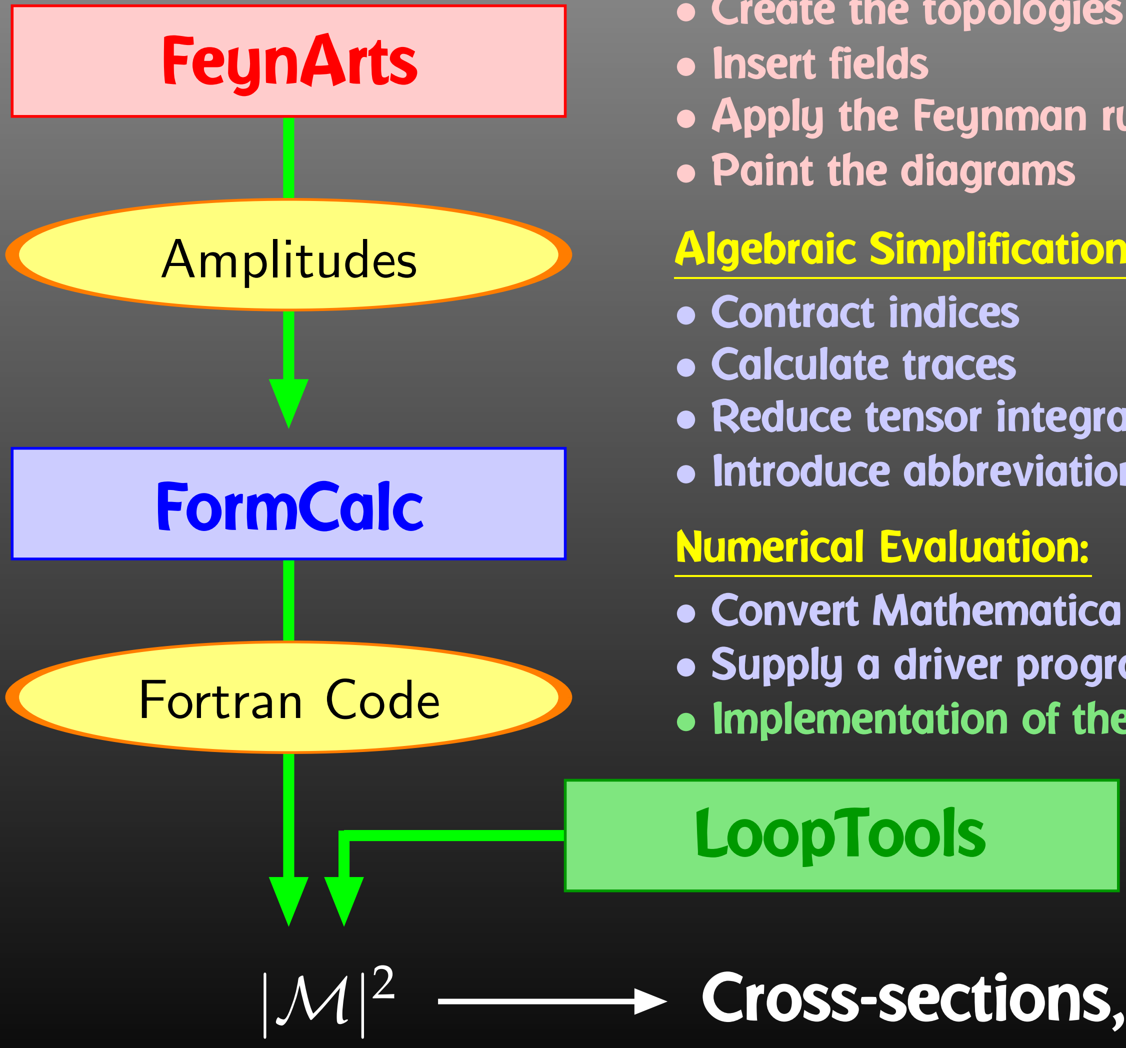
FeynArts and FormCalc

Thomas Hahn

Max-Planck-Institut für Physik
München

Overview



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

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

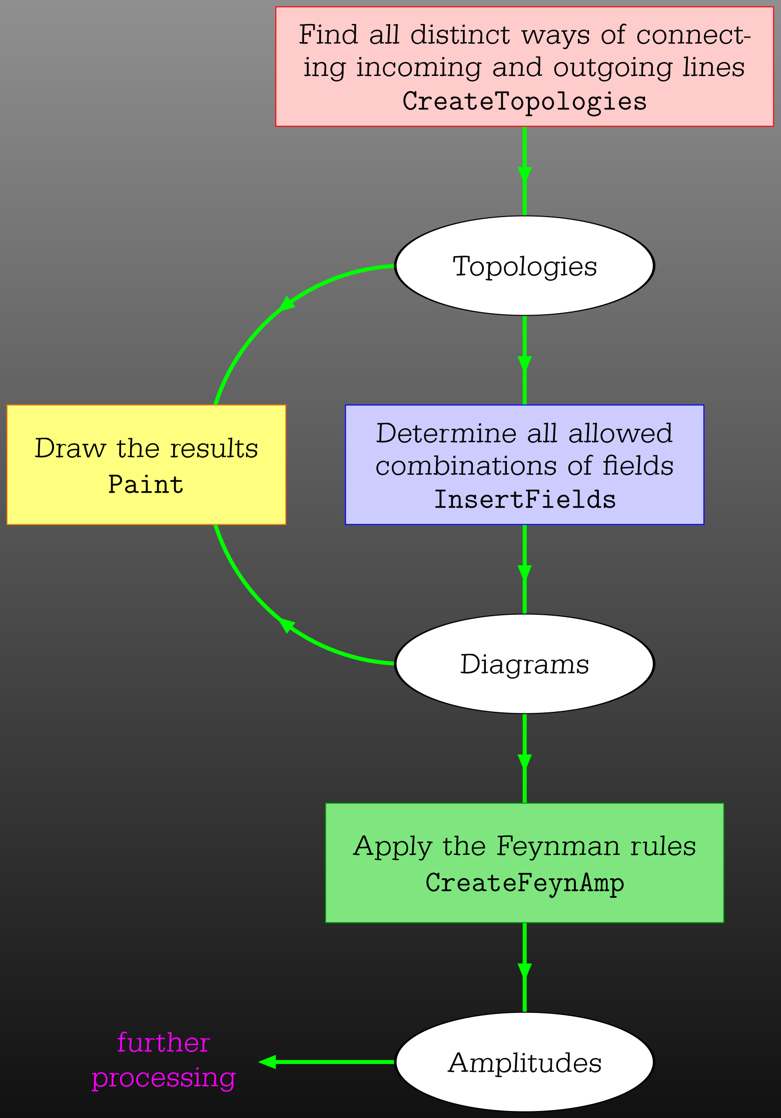
Numerical Evaluation:

- 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.

FeynArts



EXAMPLE: generating the photon self-energy

$$\texttt{top = CreateTopologies[}\ \underbrace{\texttt{1}}_{\text{one loop}},\ \underbrace{\texttt{1}}_{\text{one incoming particle}}\ \texttt{->}\ \underbrace{\texttt{1}}_{\text{one outgoing particle}}\ \texttt{]}$$

```
Paint[top]
```

$$\texttt{ins = InsertFields[top, }\underbrace{\texttt{V[1]}\ \texttt{->}\ \texttt{V[1]}}_{\text{the name of the photon in the “SM” model file}}\texttt{,}\ \underbrace{\texttt{Model -> SM}}_{\text{use the Standard Model}}\ \texttt{]}$$

```
Paint[ins]
```

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

```
amp >> PhotonSelfEnergy.amp
```

Three Levels of Fields

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

$$\boxed{C(F_1, F_2, S) = G_-\omega_- + G_+\omega_+}$$

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]`

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

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

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

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

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

$$C(F, F, S) = G_{-}\omega_{-} + G_{+}\omega_{+} = \vec{G} \cdot \begin{pmatrix} \omega_{-} \\ \omega_{+} \end{pmatrix}$$

```
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

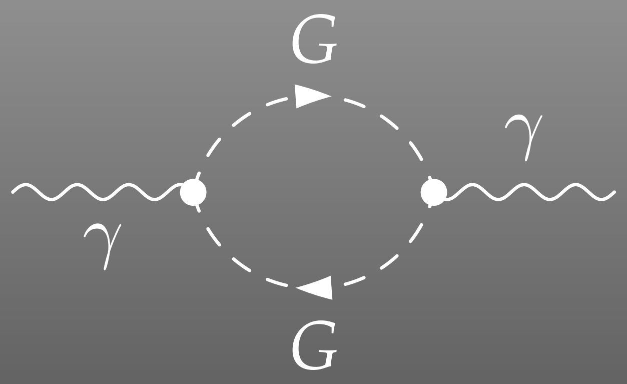
- Classes Model File (for each model) declaring the particles and the allowed couplings

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} }
```

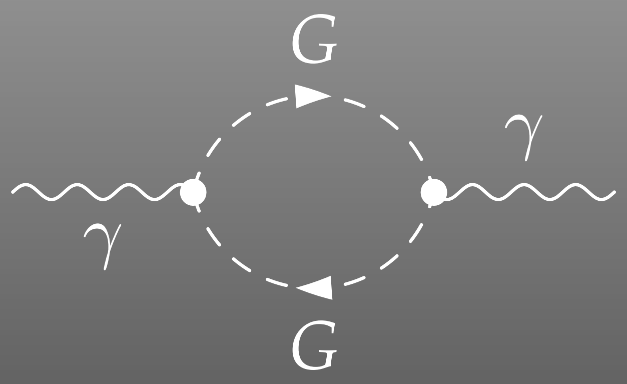
Sample CreateFeynAmp output



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

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

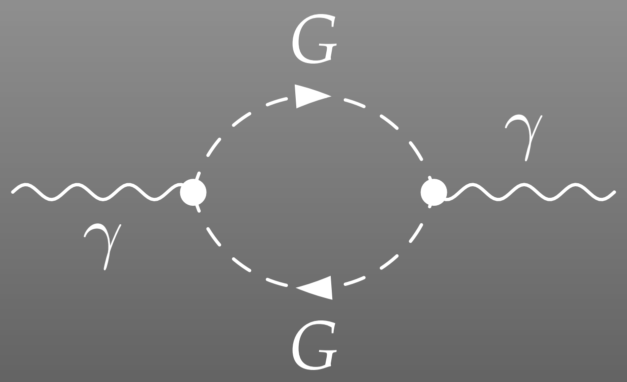
Sample CreateFeynAmp output



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

```
Integral[q1]
```

Sample CreateFeynAmp output



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

$$\frac{\texttt{I}}{\texttt{32 Pi}^4}\ \texttt{RelativeCF} \quad \ldots\ldots \textit{prefactor}$$

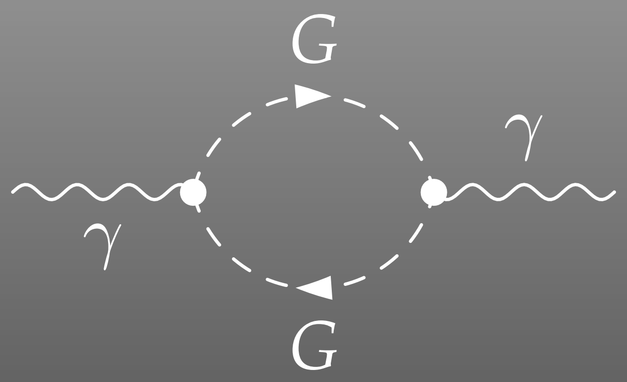
$$\texttt{FeynAmpDenominator[}\frac{1}{\texttt{q1}^2 - \texttt{Mass[S[Gen3]]}^2},\ \frac{1}{(\texttt{-p1 + q1})^2 - \texttt{Mass[S[Gen4]]}^2}\texttt{]} \quad \ldots\ldots \textit{loop denominators}$$

$$\texttt{(p1 - 2 q1)[Lor1] (-p1 + 2 q1)[Lor2]} \quad \ldots\ldots \textit{kin. coupling structure}$$

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

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

Sample CreateFeynAmp output

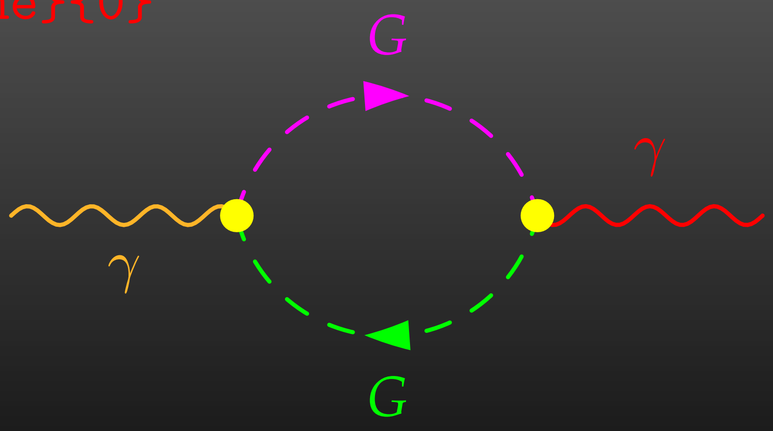


= FeynAmp[*identifier*,
loop momenta,
generic amplitude,
$\boxed{\textit{insertions}}$]

```
{ 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}]
```

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}
```



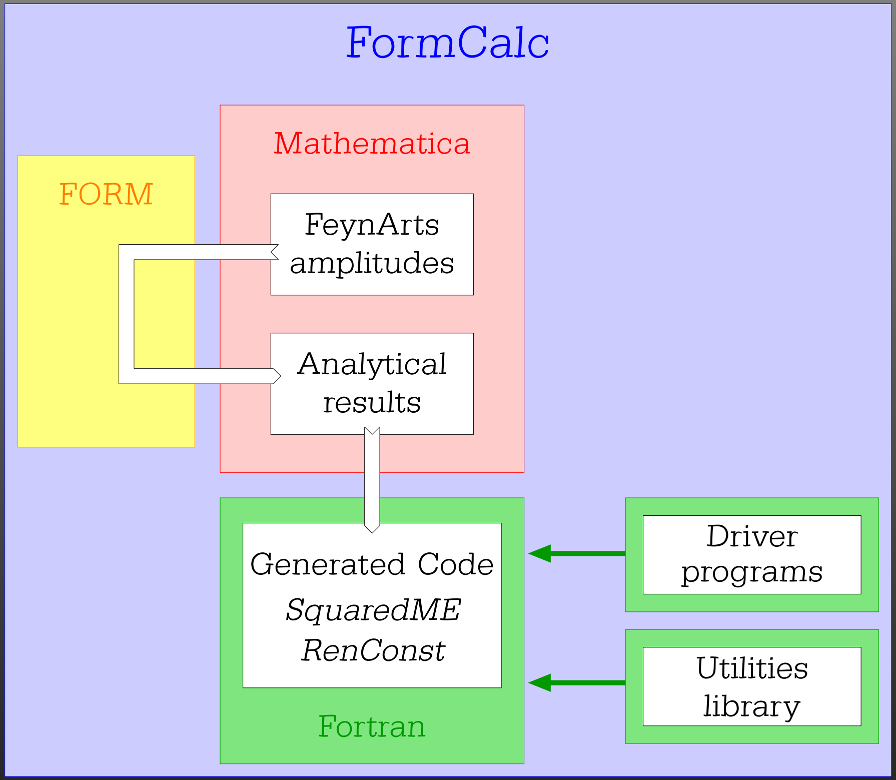
Algebraic Simplification

The amplitudes so far 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 $D\cdot$(divergent integral),
- ▷ simplify open fermion chains,
- simplify and compute the square of SU(N) structures,
- ▷ "compactify" the results as much as possible.

FormCalc



EXAMPLE: Calculating the photon self-energy

```
In[1]:= << FormCalc`

FormCalc 5
by Thomas Hahn
last revised 10 Mar 06

In[2]:= CalcFeynAmp[<< PhotonSelfEnergy.amp]

preparing FORM code in /tmp/m1.frm
> 2 amplitudes with insertions
> 5 amplitudes without insertions
running FORM...  ok

                           -3 Alfa Pair1 A0[MW2]
Out[2]= Amp[{0} -> {0}][--------------------- +
                                 2 Pi

  3 Alfa Pair1 B00[0, MW2, MW2]
  ----------------------------- +
               Pi

   Alfa Pair1 A0[MLE2[Gen1]]
  (------------------------- +
              Pi

   Alfa Pair1 A0[MQD2[Gen1]]
   ------------------------- +
             3 Pi

   4 Alfa Pair1 A0[MQU2[Gen1]]
   --------------------------- -
              3 Pi

   2 Alfa Pair1 B00[0, MLE2[Gen1], MLE2[Gen1]]
   ------------------------------------------- -
                       Pi

   2 Alfa Pair1 B00[0, MQD2[Gen1], MQD2[Gen1]]
   ------------------------------------------- -
                      3 Pi

   8 Alfa Pair1 B00[0, MQU2[Gen1], MQU2[Gen1]]
   -------------------------------------------) *
                      3 Pi

    SumOver[Gen1, 3]]
```

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.

The Abbreviate Function

The `Abbreviate` Function allows to introduce abbreviations for arbitrary expressions and extends the advantage of categorized evaluation. Example:

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

The second argument, 5, determines the Level below which abbreviations are introduced.

The level determines how much of expression is ‘abbreviated away,’ i.e. how much of the structure is preserved. In the extreme, for a level of 1, the result is just a single symbol.

At $\mathcal{O}(30\ \text{sec})$ execution time for `Abbreviate`, the typical speed-up was a factor 3 in MSSM calculations.

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

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.

Sigma Chains

Define Sigma matrices and 2-dim. Spinors as

$$\begin{aligned} \sigma_\mu &= (\mathbb{1}, -\vec\sigma)\,, \\ \overline{\sigma}_\mu &= (\mathbb{1}, +\vec\sigma)\,, \end{aligned} \qquad \begin{aligned} \langle u|_{4\mathrm{d}} &\equiv \big(\langle u_+|_{2\mathrm{d}}\,, \langle u_-|_{2\mathrm{d}}\big)\,, \\ |v\rangle_{4\mathrm{d}} &\equiv \begin{pmatrix} |v_-\rangle_{2\mathrm{d}} \\ |v_+\rangle_{2\mathrm{d}} \end{pmatrix}. \end{aligned}$$

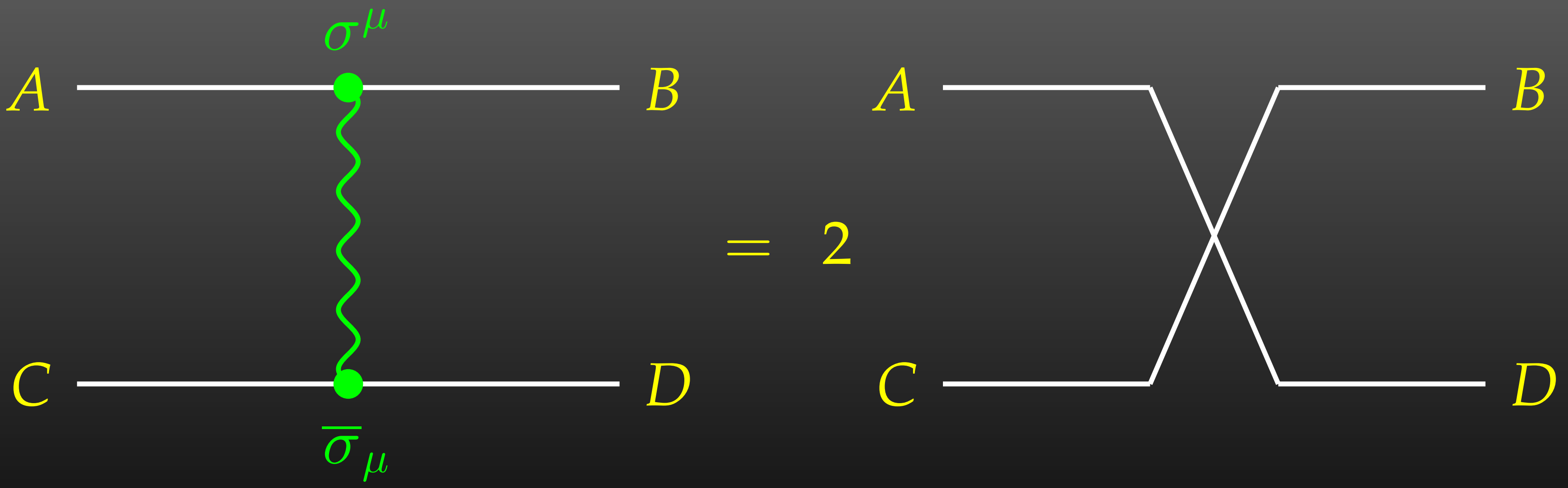
Using the chiral representation it is easy to show that every chiral 4-dim. Dirac chain can be converted to a *single* 2-dim. sigma chain:

$$\begin{aligned} \langle u|\,\omega_-\gamma_\mu\gamma_\nu\cdots|v\rangle &= \langle u_-|\,\overline{\sigma}_\mu\sigma_\nu\cdots|v_\pm\rangle\,, \\ \langle u|\,\omega_+\gamma_\mu\gamma_\nu\cdots|v\rangle &= \langle u_+|\,\sigma_\mu\overline{\sigma}_\nu\cdots|v_\mp\rangle\,. \end{aligned}$$

Fierz Identities

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

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



Implementation

- **Objects (arrays):** $|u_\pm\rangle \sim \begin{pmatrix} u_1 \\ u_2 \end{pmatrix}, \quad (\sigma \cdot k) \sim \begin{pmatrix} a & b \\ c & d \end{pmatrix}$
- **Operations (functions):**

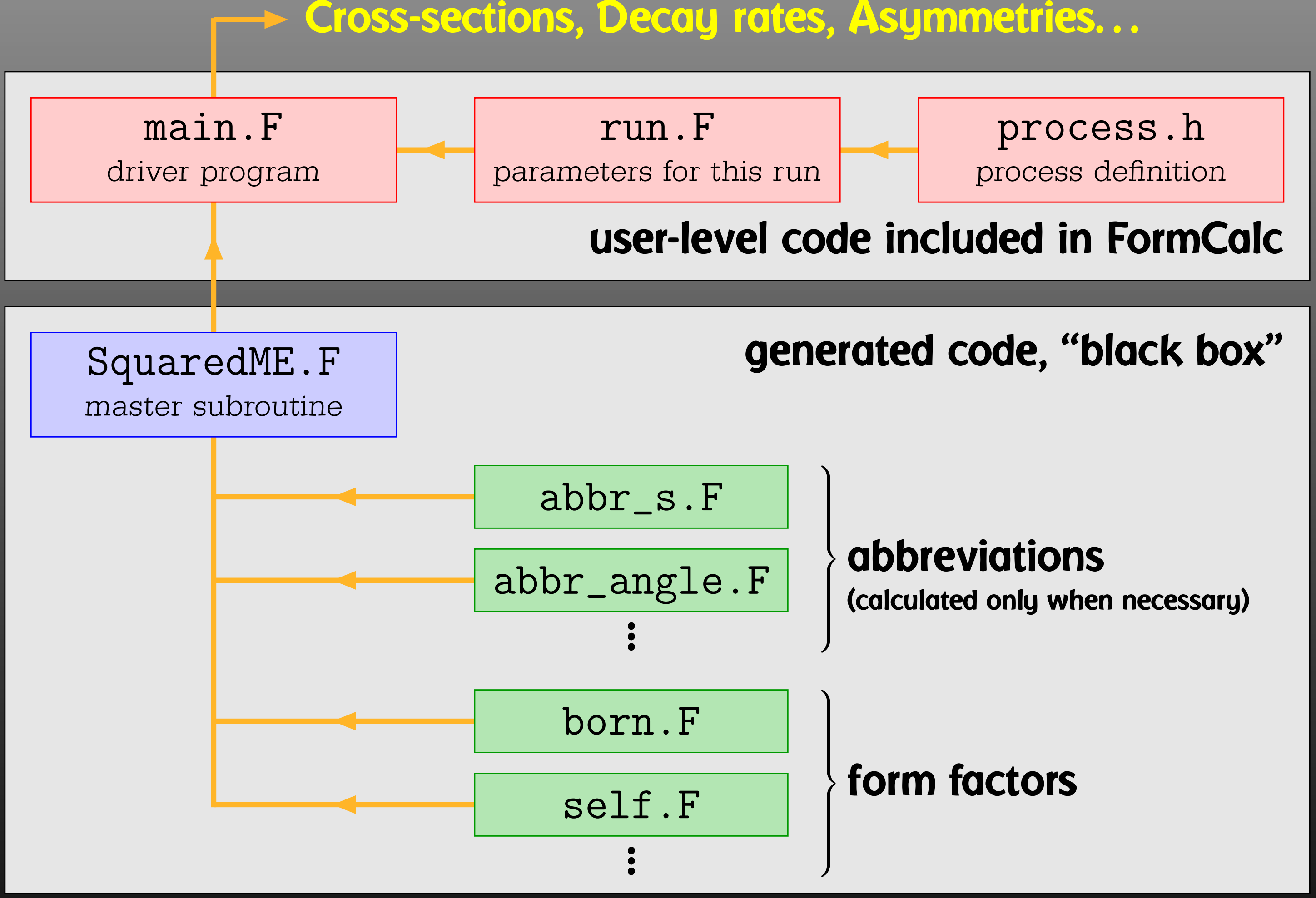
$$\langle u | v \rangle \sim (u_1 \;\; u_2) \cdot \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} \qquad \texttt{SxS}$$

$$(\overset{(-)}{\sigma} \cdot k) \, |v\rangle \sim \begin{pmatrix} a & b \\ c & d \end{pmatrix} \cdot \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} \qquad \texttt{VxS}, \; \texttt{BxS}$$

Sufficient to compute any sigma chain:

$$\langle u | \, \sigma_\mu \overline{\sigma}_\nu \sigma_\rho \, | v \rangle \, k_1^\mu k_2^\nu k_3^\rho = \texttt{SxS}(\, u, \; \texttt{VxS}(\, k_1, \; \texttt{BxS}(\, k_2, \; \texttt{VxS}(\, k_3, \; v \,)\,)\,)\,)$$

Numerical Evaluation in Fortran 77



MSSM Parameters

Input parameters:

`TB, MA0, At, Ab, Atau, MSusy, M_2, MUE`

$$\downarrow$$

$\boxed{\texttt{model\_mssm.F}}$

$$\downarrow$$

All parameters appearing in the Model File:

`Mh0, MHH, MA0, MHp, CB, SB, TB, CA, SA,`
`C2A, S2A, C2B, S2B, CAB, SAB, CBA, SBA,`
`MUE, MGl,` MNeu[n], ZNeu[n,n'],
MCha[c], UCha[c,c'], VCha[c,c'],
MSf[s,t,g], USf[t,g][s,s'], Af[t,g]

Details in TH, C. Schappacher, hep-ph/0105349.

Parameter Scans

With the preprocessor definitions in `run.F` one can either

- **assign a parameter a fixed value,** as in

```
    #define LOOP1 TB = 1.5D0
```

- **declare a loop over a parameter,** as in

```
    #define LOOP1 do 1 TB = 2,30,5
```

which computes the cross-section for `TB` values of 2 to 30 in steps of 5.

Main Program:

```
     LOOP1
     LOOP2
       ⋮
     (calculate
     cross-section)
1    continue
```

Scans are "embarrassingly parallel" - each pass of the loop can be calculated independently.
How to **distribute the iterations automatically** if the loops are
a) user-defined   b) usually nested?
Solution: Introduce a serial number

Unraveling Parameter Scans

```
      subroutine ParameterScan( range )
      integer serial
      serial = 0
      LOOP1
      LOOP2
        ⋮
      serial = serial + 1
      if( serial ∉ range ) goto 1
      (calculate cross-section)
1     continue
      end
```

Distribution on N machines is now simple:

- **Send serial numbers** $1, N+1, 2N+1, \ldots$ **on machine 1,**
- **Send serial numbers** $2, N+2, 2N+2, \ldots$ **on machine 2, etc.**

Shell-script Parallelization

Parameter scans can automatically be distributed on a cluster of computers:

- The machines are declared in a file `.submitrc`, e.g.

```
# Optional: Nice to start jobs with
nice 10
# Pentium 4 3000
pcl301
pcl301a
pcl305
# Dual Xeon 2660
pcl247b   2
pcl321    2
...
```

- The command line for distributing a job is *exactly the same* except that "`submit`" is prepended, e.g.

```
submit run uuuu 0,1000
```

FeynHiggs

FeynHiggs is a program to compute the Higgs masses, mixings, couplings, etc.

It is a major application of the FeynArts/FormCalc system, i.e. much of the code in FeynHiggs has been automatically generated, using among other things the methods in the Excursions.

Incidentally, `model_mssm.F` has not become obsolete. It rather serves as a ‘light’ version and has, of course, a FeynHiggs interface.

Corrections included in FeynHiggs 2.4

$$\begin{pmatrix} q^2 - M_h^2 + \hat{\Sigma}_{hh}^{\bullet\bullet\bullet} & \hat{\Sigma}_{hH}^{\bullet\bullet\bullet} & \hat{\Sigma}_{hA}^{\bullet\bullet} \\ \hat{\Sigma}_{Hh}^{\bullet\bullet\bullet} & q^2 - M_H^2 + \hat{\Sigma}_{HH}^{\bullet\bullet\bullet} & \hat{\Sigma}_{HA}^{\bullet\bullet} \\ \hat{\Sigma}_{Ah}^{\bullet\bullet} & \hat{\Sigma}_{AH}^{\bullet\bullet} & q^2 - M_A^2 + \hat{\Sigma}_{AA}^{\bullet\bullet} \end{pmatrix}$$

- **Most up-to-date leading $\mathcal{O}(\alpha_s\alpha_t, \alpha_t^2)$ + subleading $\mathcal{O}(\alpha_s\alpha_b, \alpha_t\alpha_b, \alpha_b^2)$ two-loop corrections (complex effects only partially included).**
- **Full one-loop evaluation (all phases included).**
- **Complete q^2 dependence.**
- **Full one-loop corrections for the charged Higgs sector.**
- **Mixed $\overline{\text{MS}}$/OS renormalization for one-loop result.**
- **"Δm_b" corrections = leading $\mathcal{O}(\alpha_s\alpha_b)$ terms for Higgs masses, couplings, etc.**
- **[NEW] Full 6×6 non-minimal flavour-violating effects (e.g. $\tilde{c}$–$\tilde{t}$ mixing).**

FeynHiggs Modes

Four operation modes:

- **Library Mode:** Invoke the FeynHiggs routines from a Fortran or C/C++ program linked with `libFH.a`.
- **Command-line Mode:** Process parameter files in FeynHiggs or SLHA format at the shell prompt or in scripts with the standalone executable `FeynHiggs`.
- **WWW Mode:** Interactively choose the parameters at the FeynHiggs User Control Center (FHUCC) and obtain the results on-line.
- **Mathematica Mode:** Access the FeynHiggs routines in Mathematica via MathLink with `MFeynHiggs`.

All programs and subroutines are documented in man pages.

SLHA I/O Library

- The SUSY Les Houches Accord defines a common interface for SUSY tools. The SLHA2 adds various extensions (CPV, RV, NMFV).
- Reading/writing SLHA files not entirely straightforward.
- The SLHA I/O Library fills this gap:
  - ▷ Implemented as native Fortran 77 Library.
  - ▷ All data transferred in one double complex array.
  - ▷ This array is indexed by preprocessor macros, e.g. `MinPar_TB` instead of `slhadata(20)`.
  - ▷ Main functions: `SLHARead`, `SLHAWrite`.
- Latest version: SLHALib 2.0 (for SLHA2), available at http://www.feynarts.de/slha.

Summary and Outlook

- **Serious perturbative calculations these days can generally no longer be done by hand:**
  - ▷ Required accuracy, Models with many particles, . . .
- **Hybrid programming techniques are necessary:**
  - ▷ Computer algebra is an indispensable tool because many manipulations must be done symbolically.
  - ▷ Fast number crunching can only be achieved in a compiled language.
- **Software engineering and further development of the existing packages is a must:**
  - ▷ As we move on to ever more complex computations (more loops, more legs), the computer programs must become more “intelligent,” i.e. must learn all possible tricks to still be able to handle the expressions.