FeynRules

Claude Duhr

2015 MadGraph School on Collider Phenomenology,
Shanghai, 23-27/11/2015

Going Beyond SM

- A BSM model can be defined via
  - ➡ The particles appearing in the model.
  - ➡ The values of the parameters (‘Benchmark point’).
  - ➡ The interactions among the particles, usually dictated by some symmetry group, and quantified in the Lagrangian of the model.
- All this information needs to be implemented into the MC codes, usually in the form of text files that contain the definitions of the particles, the parameters and the vertices.

Going Beyond SM

- This can be a very tedious exercise.
- Most of these codes have only a very limited amount of models implemented by default (~ SM and MSSM).
- However, still these codes do not work at the level of Lagrangians, but need explicit vertices.
- The process of implementing Feynman rules can be particularly tedious and painstaking:
  - ➡ Each code has its own conventions (signs, factors of i, ...).
  - ➡ Vertices need to be implemented one at the time.
- Most codes can only handle a limited amount of color and / or Lorentz structures (~ SM and MSSM)

Going Beyond SM

- The aim of these lectures is to present a code that automatizes all these steps, and allows to implement the model into Matrix element generators starting directly from the Lagrangian.
- Workflow:
  - ➡ Define your particles and parameters.
  - ➡ Enter your Lagrangian.
  - ➡ Let the code compute the Feynman rules.
  - ➡ Output all the information in the format required by your favorite MC code.

Plan of the Lecture

- What is FeynRules?
- Getting started:
  - ➡ ϕ^4 theory.
  - ➡ Adding gauge interactions (scalar QCD).
  - ➡ Adding mixings.
- Extending existing implementations.
- Towards LHC phenomenology: The FeynRules interfaces.
- Going NLO.

N.B.: Tutorials in the afternoon!

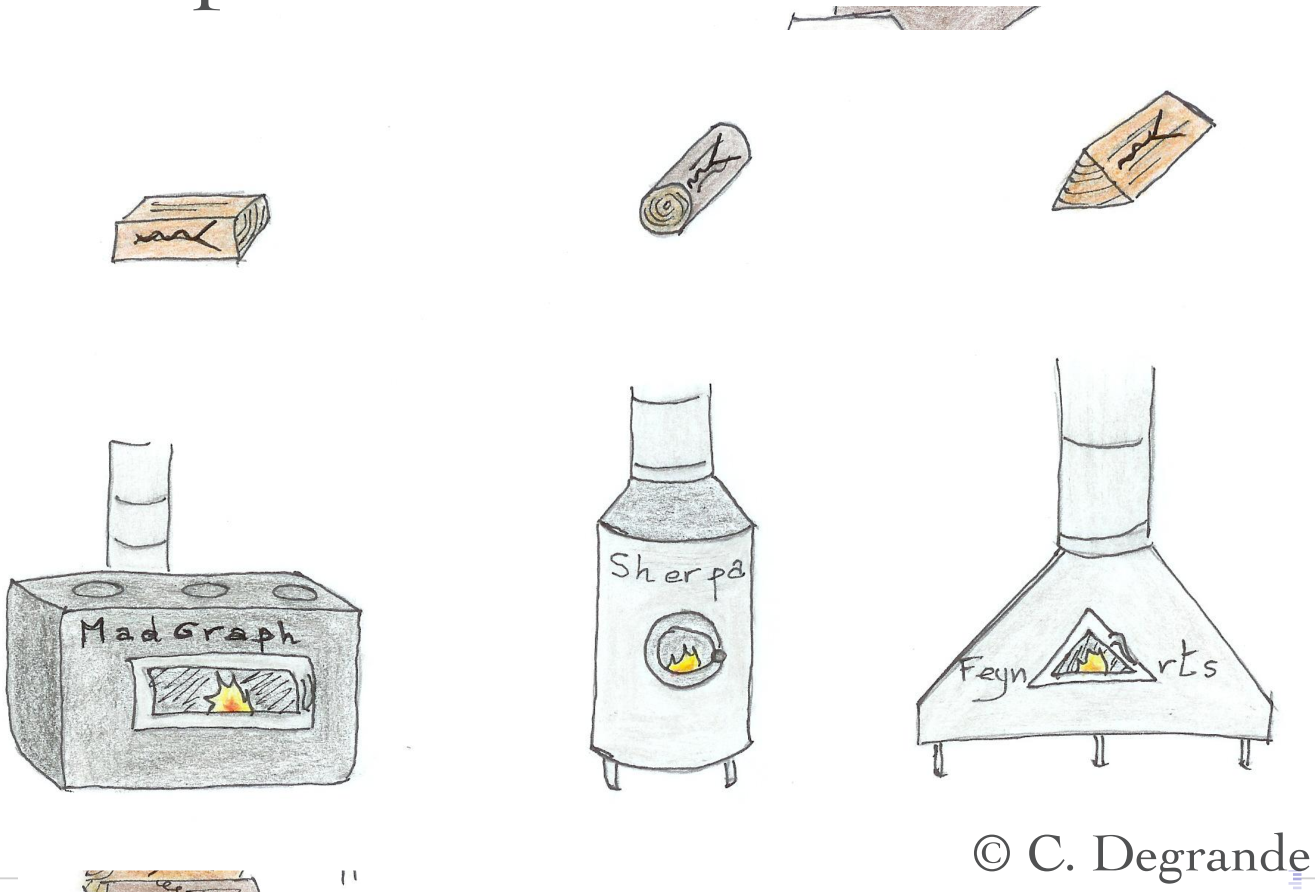
What is FeynRules?

FeynRules

- FeynRules is a Mathematica package that allows to derive Feynman rules from a Lagrangian.
- FeynRules is publicly available from http://feynrules.irmp.ucl.ac.be/
- The only requirements on the Lagrangian are:
  - ➡ All indices need to be contracted (Lorentz and gauge invariance).
  - ➡ CPT invariance (~ 'normal' particle/anti-particle relation).
  - ➡ Locality.
  - ➡ Supported field types: spin 0, 1/2, 1, 3/2, 2 & ghosts.

FeynRules

- FeynRules comes with a set of interfaces, that allow to export the Feynman rules to various matrix element generators.
- Interfaces coming with current public version
  - ➡ CalcHep / CompHep
  - ➡ FeynArts / FormCalc
  - ➡ UFO
  - ➡ Whizard / Omega



FeynRules

- The input requested form the user is twofold.

- **The Model File:**
  Definitions of particles and parameters (e.g., a quark)

```
F[1]  ==
  {ClassName      ->   q,
   SelfConjugate ->  False,
   Indices           ->  {Index[Colour]},
   Mass              ->  {MQ,  200},
   Width             ->  {WQ, 5}   }
```

- **The Lagrangian:**

$$\mathcal{L} = -\frac{1}{4} G^a_{\mu\nu}\, G^{\mu\nu}_a + i\bar{q}\,\gamma^\mu\, D_\mu q - M_q\,\bar{q}\,q$$

```
L =
-1/4 FS[G,mu,nu,a] FS[G,mu,nu,a]
+ I qbar.Ga[mu].DC[q,mu]
- MQ qbar.q
```

FeynRules

- Once this information has been provided, FeynRules can be used to compute the Feynman rules for the model:

```
FeynmanRules[ L ]
```

- Equivalently, we can export the Feynman rules to a matrix element generator, e.g., for MadGraph 5,

```
WriteUFO[ L ]
```

- This produces a set of files that can be directly used in the matrix element generator.

Getting Started:
phi4 theory

Phi 4 theory

- Let us consider a model consisting of two complex scalar fields, interacting with each other:

$$\mathcal{L} = \partial_\mu \phi_i^\dagger \partial^\mu \phi_i - m^2 \phi_i^\dagger \phi_i + \lambda (\phi_i^\dagger \phi_i)^2$$

- We need to implement into a FeynRules model file
  - ➡ The two fields ϕ_1 and ϕ_2 , or rather one field carrying an index.
  - ➡ The two new parameters m and λ .

- In a second step, we need to implement the Lagrangian into Mathematica.

How to write a model file

- A model file is simply a text file (with extension *.fr*).
- The syntax is Mathematica.
- General structure:

Preamble

(Author info, model info, index definitions, ...)

Particle Declarations

(Particle class definitions, spins, quantum numbers, ...)

Parameter Declarations

(Numerical Values, ...)

Preamble of the model file

- The preamble allows to 'personalise' the model file, and define all the indices that are carried by the fields
  - ➡ In our case we have one index, taking the values 1 or 2.

```
M$ModelName =  "Phi_4_Theory";



M$Information = {Authors -> {"C. Duhr"},
                 Version -> "1.0",
                 Date -> "23. 11. 2015"};


IndexRange[ Index[Scalar] ] = Range[2];
IndexStyle[ Scalar, i];
```

Preamble of the model file

- Sometimes it is useful to introduce auxiliary indices to obtain compact Lagrangians, but these indices should always be expanded.
  - ➡ Example: Weak isospin indices.
- There is a way to instruct FeynRules at run time to expand certain indices (see later).
- In addition, one can specify in the model file if a certain type of indices should **always** be expanded:

```
IndexRange[ Index[Scalar] ] = Unfold[ Range[2] ];
IndexStyle[ Scalar, i];
```

Particle Declaration

- Particles are defined as ‘classes’, grouping together particles with similar quantum numbers, but different masses (~multiplet).

```
M$ClassesDescription = {
  S[1] == {
        ClassName -> phi,
        ClassMembers -> {phi1,phi2},
        SelfConjugate -> False,
        Indices -> {Index[Scalar]},
        FlavorIndex -> Scalar,
        Mass -> {MS, 100}
     }
};
```

Spin (S, F, V, U, T)

Symbol used for the particle in the Lagrangian. Antiparticle called phibar.

The field is complex, i.e., there is an antiparticle.

Symbol for the mass used in the Lagrangian, + numerical value in GeV.

Particle Declaration

- There are many more (optional) properties for particle classes:
  - ➡ `Width`: Total width of the particle. 0 if stable.
  - ➡ `QuantumNumbers`: U(1) charges carried by the field.
  - ➡ `PDG`: PDG code of the particle (if existent).
  - ➡ `ParticleName/AntiParticleName`: A string, by which the particle will be referred to in the MC code.
  - ➡ `Unphysical`: If `True`, then the particle is tagged as not a mass eigenstate, and will not be output to the MC code.
  - ➡ Many more. See the FeynRules manual.

Parameter Declaration

- Parameter classes are defined in a similar way to the particle classes.
  - ➡ In our case, we have two parameters, the mass m and the coupling λ .
  - ➡ The mass was already defined with the particle, no need to define it a second time.

```
M$Parameters = {
   lam == {
          Value -> 0.1
     }
};
```

Parameter Declaration

- Parameters belong to two different classes, specified by the option `ParameterType`:
  - ➡ `External`: Numerical input parameters of the model. The `Value` must be a **real** floating point number. Example:
  $$\alpha_s = 0.118$$
  - ➡ `Internal`: Dependent on other external and/or internal parameters. The `Value` can be a floating point number or an algebraic expression (in Mathematica synthax). Example:
  $$g_s = \sqrt{4\pi\alpha_s}$$
- By default every new parameter is `External`.

Parameter Declaration

- By default, all parameters are defined as real. It can be made complex by setting the ComplexParameter option to True.

- Just like particles, parameters can carry Indices, i.e., they can be matrices

- It is possible to specifiy that a matrix is hermitian, etc.
  - ➡ Hermitian: True/False.
  - ➡ Orthogonal: True/False.
  - ➡ Unitary: True/False.

The Mathematica session

- We now run FeynRules to obtain the Feynman rules of the model
  - ➡ This is done in a Mathematica notebook.

- Step 1: Load FeynRules into Mathematica

```
In[1]:= $FeynRulesPath = SetDirectory["~/FeynRules-SVN/feynrules-current"];

In[2]:= << FeynRules`

 – FeynRules –

Authors: C. Duhr, N. Christensen, B. Fuks


Please cite: Comput.Phys.Commun.180:1614–1641,2009 (arXiv:0806.4194).

http://feynrules.phys.ucl.ac.be
```

The Mathematica session

- Step 2: Load the model file

```
In[3]:= SetDirectory["~/FeynRules-SVN/trunk/models/Phi_4_Theory"];

In[4]:= LoadModel["Phi_4_Theory.fr"]
```

This model implementation was created by

C. Duhr

Model Version: 1.0

For more information, type ModelInformation[].

The Mathematica session

- Step 3: Enter the Lagrangian

$$\mathcal{L} = \partial_\mu \phi_i^\dagger \partial^\mu \phi_i - m^2 \phi_i^\dagger \phi_i + \lambda (\phi_i^\dagger \phi_i)^2$$

```
In[5]:= L = del[phibar[i], mu] del[phi[i], mu] - MS^2 phibar[i] phi[i] +
          lam (phibar[i] phi[i]) (phibar[j] phi[j])
```

Out[5]= $\text{lam}\,\text{phi}_i\,\text{phi}_j\,\text{phi}_i^\dagger\,\text{phi}_j^\dagger + \text{MS}^2\,(-\text{phi}_i)\,\text{phi}_i^\dagger + \partial_{\text{mu}}(\text{phi}_i)\,\partial_{\text{mu}}(\text{phi}_i^\dagger)$

The Mathematica session

- Step 4: Computing the Feynman rules

```
In[6]:= FeynmanRules[L]
```

(* *)

Vertex 1

Particle 1 : Scalar , phi

Particle 2 : Scalar , phi

Particle 3 : Scalar , $\text{phi}^\dagger$

Particle 4 : Scalar , $\text{phi}^\dagger$

Vertex:

$$2\, i\, \text{lam}\, \delta_{i_1,i_4}\, \delta_{i_2,i_3} + 2\, i\, \text{lam}\, \delta_{i_1,i_3}\, \delta_{i_2,i_4}$$

(* *)

Feynman rule for the particle class!

The Mathematica session

- Step 4: Computing the Feynman rules

```
In[7]:= FeynmanRules[L, FlavorExpand → True]
```

(∗ ∗)

Vertex 3

Particle 1 : Scalar , phi2

Particle 2 : Scalar , phi2

Particle 3 : Scalar , $\mathrm{phi2}^\dagger$

Particle 4 : Scalar , $\mathrm{phi2}^\dagger$

Vertex:

$4\, i\, \mathrm{lam}$

(∗ ∗)

The Mathematica session

- A selection of options for the FeynmanRules function:
  - ➡ `FlavorExpand`: List of all flavor indices that should be expanded. If `True`, then all flavor indices are expanded.
  - ➡ `ScreenOutput`: If `False`, the vertices are not printed on screen (useful for big models with 100's of vertices).
  - ➡ `SelectParticles`: Allows to only compute certain specific vertices.
  - ➡ `MaxParticles/MinParticles`: an integer, specifying the maximal/minimal number of particles that should appear in a vertex.
  - ➡ `Exclude4Scalars`: If `True`, rejects all four-scalar vertices (useful for big models with a plethora of phenomenologically irrelevant four-scalar interactions).

Getting Started: Gauging our model

Gauging phi4 theory

- Let us gauge our model, say the scalar is in the adjoint of SU(3) (QCD octet).
- The change in the Lagrangian is very minor:
  - ➡ add field strength tensor
  - ➡ replace derivative by covariant derivative.

$$\mathcal{L} = -\frac{1}{4}F^a_{\mu\nu}F^{\mu\nu}_a + D_\mu\phi_i^\dagger D^\mu\phi_i - m^2\phi_i^\dagger\phi_i + \lambda(\phi_i^\dagger\phi_i)^2$$

$$D_\mu = \partial_\mu - ig_s T^a G^a_\mu$$

- Technically speaking, we just added two new objects to our model:
  - ➡ a new particle: the gluon G.
  - ➡ a new parameter: the gauge coupling g_s.

Preamble of the model file

- The fields now carry an index in the adjoint index.
  - ➡ Need to define this new index in the preamble.

```
M$ModelName = "Phi_4_Theory_Octet";


M$Information = {Authors -> {"C. Duhr"},
                 Version -> "1.0",
                 Date -> "09. 09. 2011"};

IndexRange[ Index[Scalar] ] = Range[2];
IndexStyle[ Scalar, i];
IndexRange[ Index[Gluon] ] = Range[8];
IndexStyle[ Gluon, a];
```

Particle Declaration

- The scalar is now an octet.

```
M$ClassesDescription = {
   S[1] == {
         ClassName -> phi,
         ClassMembers -> {phi1,phi2},
         SelfConjugate -> False,
         Indices -> {Index[Scalar], Index[Gluon]},
         FlavorIndex -> Scalar,
         Mass -> {MS, 100}
     }
};
```

Particle Declaration

- We also need to define the gluon field.

```
M$ClassesDescription = {
  S[1] == {...},

  V[1] == {
        ClassName -> G,
        SelfConjugate -> True,
        Indices -> {Index[Gluon]},
        Mass -> 0
    }
};
```

Parameter Declaration

- We also need to define the gauge coupling.

```
M$Parameters = {
   lam == {
            Value -> 0.1
       },

   gs == {
            Value -> 1.22
       }
};
```

Gauge groups

- We have now defined the gauge coupling and the gauge boson.
- To gauge the theory we need however more:
  - ➡ Structure constants.
  - ➡ Representation matrices.
  - ➡ ...
- FeynRules allows to define gauge group classes in a similar way to particle and parameter classes.

Gauge groups

- FeynRules allows to define gauge group classes in a similar way to particle and parameter classes.

```
M$GaugeGroups = {

  SU3C == {
       Abelian -> False,
       GaugeBoson -> G,
       StructureConstant -> f,
       CouplingConstant -> gs
     }
}
```

- Could add other representations via
  `Representation -> {T, Colour}`

The Mathematica session

- Step 1: Load FeynRules into Mathematica
- Step 2: Load the model file
- Step 3: Enter the Lagrangian

$$\mathcal{L} = -\frac{1}{4} F^a_{\mu\nu} F_a^{\mu\nu} + D_\mu \phi_i^\dagger D^\mu \phi_i - m^2 \phi_i^\dagger \phi_i + \lambda (\phi_i^\dagger \phi_i)^2$$

```
In[9]:= L = -1 / 4 FS[G, mu, nu, a] FS[G, mu, nu, a] +
          DC[phibar[i, a], mu] DC[phi[i, a], mu] - MS^2 phibar[i, a] phi[i, a] +
          lam (phibar[i, a] phi[i, a]) (phibar[j, b] phi[j, b])
```

Out[9]=

$$\left(\partial_{\text{mu}}(\text{phi}_{i,a}) - i\,\text{gs}\,G_{\text{mu},a\$979}\,\text{phi}_{i,i\$979}\,\text{FSU3C}^{a\$979}_{a,i\$979}\right)\left(\partial_{\text{mu}}(\text{phi}^\dagger_{i,a}) + i\,\text{gs}\,G_{\text{mu},a\$978}\,\text{FSU3C}^{a\$978}_{i\$978,a}\,\text{phi}^\dagger_{i,i\$978}\right) + \text{lam}\,\text{phi}_{i,a}\,\text{phi}_{j,b}\,\text{phi}^\dagger_{i,a}\,\text{phi}^\dagger_{j,b} - \frac{1}{4}\left(\text{gs}\,G_{\text{mu},\text{bb}\$976}\,G_{\text{nu},\text{cc}\$976}\,f_{a,\text{bb}\$976,\text{cc}\$976} - \partial_{\text{nu}}(G_{\text{mu},a}) + \partial_{\text{mu}}(G_{\text{nu},a})\right)\left(\text{gs}\,G_{\text{mu},\text{bb}\$977}\,G_{\text{nu},\text{cc}\$977}\,f_{a,\text{bb}\$977,\text{cc}\$977} - \partial_{\text{nu}}(G_{\text{mu},a}) + \partial_{\text{mu}}(G_{\text{nu},a})\right) + \text{MS}^2\left(-\text{phi}_{i,a}\right)\text{phi}^\dagger_{i,a}$$

The Mathematica session

- **Step 4:** Computing the Feynman rules

```
In[6]:= FeynmanRules[L]
```

(* *)

(* *)

Vertex 5

Particle 1 : Scalar , phi

Particle 2 : Scalar , phi

Particle 3 : Scalar , $\text{phi}^{\dagger}$

Particle 4 : Scalar , $\text{phi}^{\dagger}$

Vertex:

$$2\, i\, \text{lam}\, \delta_{a_1,a_4}\, \delta_{a_2,a_3}\, \delta_{i_1,i_4}\, \delta_{i_2,i_3} + 2\, i\, \text{lam}\, \delta_{a_1,a_3}\, \delta_{a_2,a_4}\, \delta_{i_1,i_3}\, \delta_{i_2,i_4}$$

(* *)

(* *)

Getting Started: Mixings

Mixings

- So far our model has the following form:

$$\mathcal{L} = D_\mu \phi_i^\dagger D^\mu \phi_i - m^2 \phi_i^\dagger \phi_i + \lambda(\phi_i^\dagger \phi_i)^2$$

- In many BSM models the new fields are not mass eigenstates, but they mix, e.g.

$$\mathcal{L} = D_\mu \phi_i^\dagger D^\mu \phi_i - m^2 \phi_i^\dagger \phi_i - m_{12}^2(\phi_1^\dagger \phi_2 + \phi_2^\dagger \phi_1) + \lambda(\phi_i^\dagger \phi_i)^2$$

- The gauge and mass eigenstates are then related via some unitary rotation,

$$\begin{pmatrix} \phi_1 \\ \phi_2 \end{pmatrix} = U \begin{pmatrix} \Phi_1 \\ \Phi_2 \end{pmatrix}$$

Mixings

- FeynRules offers the possibility to write the Lagrangian in terms of the gauge eigenstates, and let Mathematica perform the rotation.
- N.B.: There is a way to let FeynRules diagonalize the mass matrices.
- For small mixing matrices, this can simply be done in Mathematica.
- For larger matrices, need to use some external numerical code.

Mixings

- The mixing matrix is declared as a parameter:

```
M$Parameter = {
  ...

   UU == {
         ComplexParameter -> True,
         Unitary -> True
         Indices  -> {Index[Scalar], Index[Scalar]},
         Value    -> { UU[1,1]  ->  ...,
                       UU[1,2]  ->  ...,
                        ...}
     }
   ...
};
```

Mixings

- The mass eigenstates are declared as normal particles

```
M$ClassesDescription = {
   ....
   S[11] == {
         ClassName       -> PP,
         ClassMembers -> {PP1,PP2},
         SelfConjugate   -> False,
         Indices            -> {Index[Scalar], Index[Gluon]},
         FlavorIndex      -> Scalar,
         Mass               -> {{MP1, ...},  {MP2, ...}}
     }
    ...
};
```

Mixings

- The gauge eigenstates are declared in a similar way

```
M$ClassesDescription = {
   S[1] == {
         ClassName        -> phi,
         ClassMembers     -> {phi1,phi2},
         SelfConjugate    -> False,
         Indices          -> {Index[Scalar], Index[Gluon]},
         FlavorIndex      -> Scalar,
         Mass             -> {MS, 100}
         Unphysical       -> True,
         Definitions -> {phi[i_, a_] :> Module[{j}, UU[i,j] PP[j,a]]}
      }
};
```

Extending existing implementations

Extending the SM

- So far we have only considered our model standalone.
- For LHC phenomenology, one usually wants a BSM model that is an extension of the SM.
- FeynRules offers the possibility to start form the SM model, and to add/change/remove particles and operators.
- For this, it is enough to load our new model together with the SM implementation:

```
LoadModel[ “SM.fr“, “Phi_4_Gauged“ ];
```

- Note that the ‘parent model’ should always be loaded first in order to ensure that everything is set up correctly.

N.B.: In the SM implementation, the gluon and the QCD gauge group are already defined, so no need to redefine them.

Other available models

- The same procedure can be used to extend any other models.
- Many models can be downloaded from the FeynRules web page, and can serve as a start to implement new models (http://feynrules.irmp.ucl.ac.be/).

  - ➡ SM (+ extensions: 4th generation, diquarks, See-saw...).
  - ➡ MSSM, NMSSM, RPV-MSSM, MRSSM.
  - ➡ Extra dimensions: UED, LED, Higgsless, HEIDI.
  - ➡ Minimal walking Technicolor.

Model database

We encourage model builders writing
order to make them useful to a comm
FeynRules model database, please se

- claude.duhr@durham.ac.uk
- neil@hep.wisc.edu
- fuks@cern.ch

Available models

Standard Model

Simple extensions of the SM (9)

Supersymmetric Models (4)

Extra-dimensional Models (4)

Strongly coupled and effective field theories (4)

Miscellaneous (0)

Towards LHC phenomenology: The FeynRules interfaces

The Interfaces

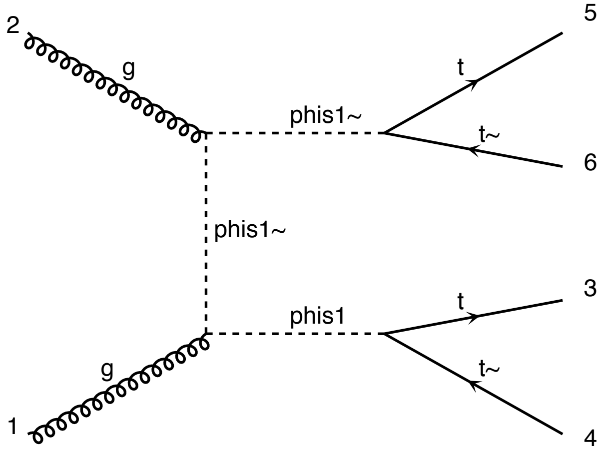
- So far we have only discussed how to implement a model into FeynRules and how to obtained the vertices.
- Next we want to do phenomenology!
- FeynRules contains interfaces to the following codes:
  - ➡ CalcHep / CompHep
  - ➡ FeynArts / FormCalc
  - ➡ UFO (GoSam, Herwig, MadGraph, Sherpa)
  - ➡ Whizard / Omega
- Each interface produces a set of text files that can be read into the existing generators.

Running Interfaces

- The interfaces are called via the Mathematica commands

```
WriteCHOutput[ LSM, L ];              (* CalcHep *)
WriteFeynArtsOutput[ LSM, L ];        (* FeynArts/FormCalc *)
WriteUFO[ LSM, L ];                   (* UFO *)
WriteWOOutput[ LSM, L];               (* Whizard / Omega *)
```

- The files produced by FeynRules can then be processed by the matrix element generators.



The UFO

UFO = Universal FeynRules Output

- Idea: Create Python modules that can be linked to other codes and contain all the information on a given model.
- The UFO is a self-contained Python code, and not tied to a specific matrix element generator.
- The content of the FR model files, together with the vertices, is translated into a library of Python objects, that can be linked to other codes.
- By design, the UFO does not make any assumptions on Lorentz/color structures, or the number of particles.

FeynRules MC conventions

- FeynRules itself does not make any assumption on the model, but its core is completely agnostic of any structure, like QCD, QED, etc.
- In order for the MC generator to function properly, they must be able to identify in each new model some standard information, like for example
  - ➡ Color and electric charges of particles.
  - ➡ Color structures of vertices.
  - ➡ Strong and weak coupling constant.
  - ➡ etc.
- Roughly speaking, each MC code needs the information on the SM parameters to be provided in a specific format.

FeynRules MC conventions

- As a consequence, even though the FeynRules core is completely agnostic, the SM parameters must be entered following specific conventions.
- The SM gauge groups must be defined in the same way as in the SM implementation, e.g., for QCD,
  - ➡ Fundamental representation matrices: T
  - ➡ Structure constants: f
  - ➡ Strong coupling: gs
- The SM input parameters should correspond to the SMINPUTS of the SUSY Les Houches Accord:

$$M_Z, \alpha_s, \alpha_{EW}^{-1}, G_F$$

Interaction orders

- MadGraphs 'tags' coupling constants and counts how many coupling constants of a given type enter a diagram.

- This allows to select certain types of diagrams, e.g., for
  p p >t t~
  - ➡ purely QCD production,
  - ➡ purely EW production,
  - ➡ both QCD and EW production (including interference).

- This requires that each coupling constant has a 'counter' (= interaction order) defined.

Interaction orders

- This can be done at the FeynRules level using the `InteractionOrder` property.

```
M$Parameters = {
   lam == {
           Value -> 0.1,
           InteractionOrder -> {YUK, 1}
      },

   gs == {
           Value -> 1.22,
           InteractionOrder -> {QCD, 1}
      }
};
```

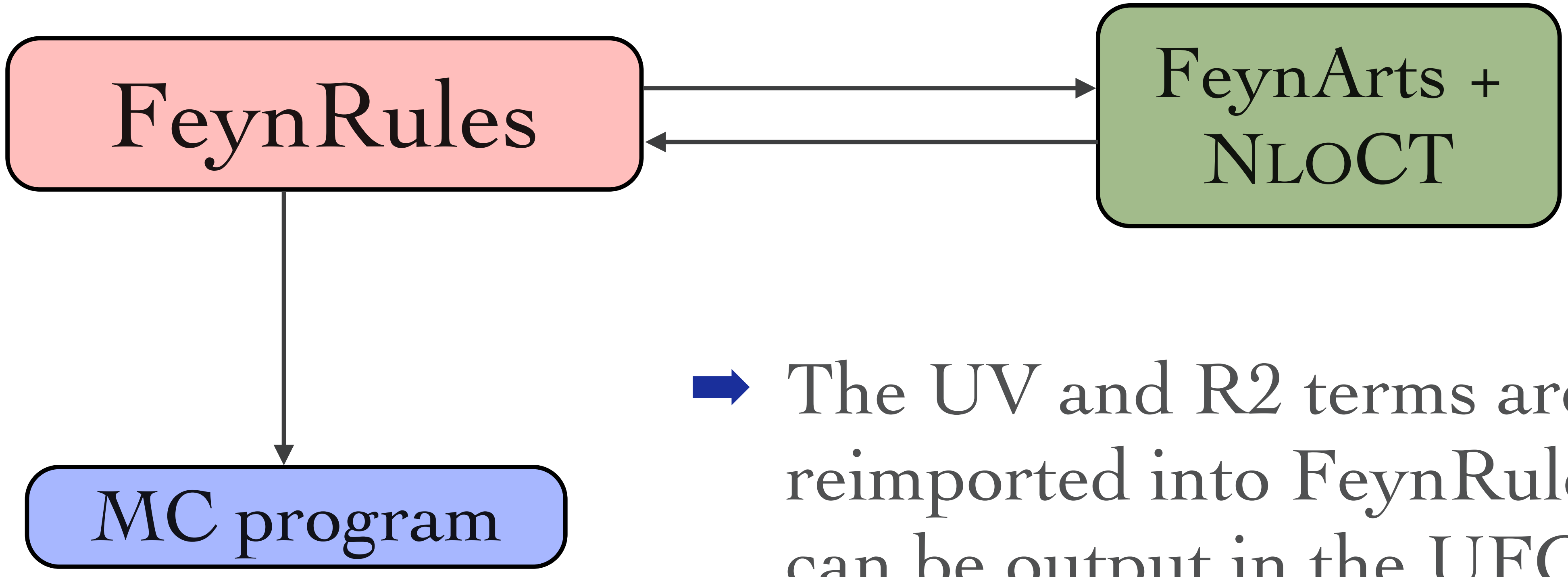
Going NLO

NLOCT - NLO counterterms [Degrande]

- NLO accuracy is/will soon be the new standard for MC simulations.
- Next goal: Bring NLO models to the same level of automation than at LO!
  - ➡ UV counterterms.
  - ➡ 'R2' vertices (effective tree-level vertices arising from (d-4)-dimensional part of numerators).
- The information cannot exclusively be extracted from a tree-level Lagrangian!
  - ➡ Requires the evaluation of loop integrals.
- The NLOCT package allows to solve this problem!

NLoCT - NLO counterterms [Degrande]

- Idea of NLoCT:
  - ➡ Use FeynRules interface to FeynArts to implement the model into FeynArts.
  - ➡ Use FeynArts to write the relevant amplitudes and NLOCT to compute their R2 and UV parts.



➡ The UV and R2 terms are reimported into FeynRules and can be output in the UFO format.

NLOCT - NLO counterterms [Degrande]

- **Step 1:** Introduce renormalization constants for all fields and parameters, and determine renormalization constants for dependent parameters.

```
Lren = OnShellRenormalization[ L ];
```

- **Step 2:** Output the model to FeynArts.

```
WriteFeynArtsOutput[ Lren ];
```

- **Step 3:** Run FeynArts and NLOCT to compute the counterterms:

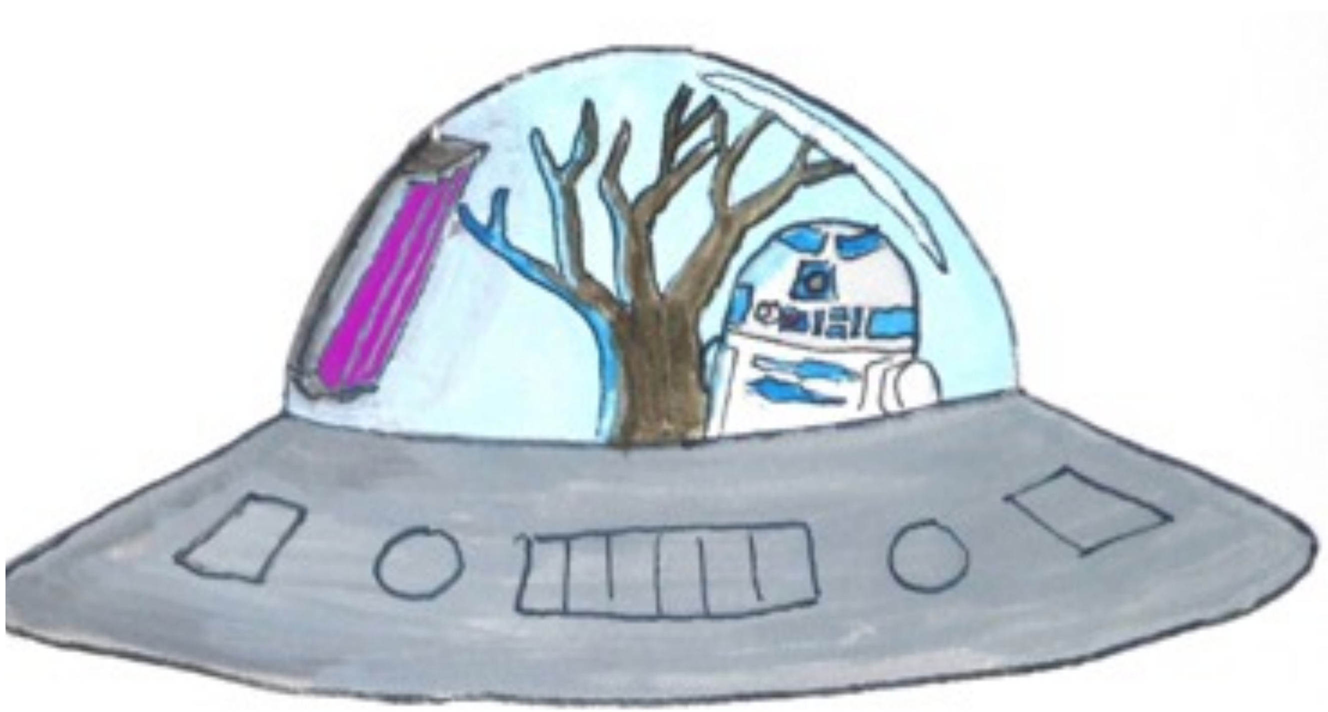
```
WriteCT[ "model", "generic_model", options ];
```

NLoCT - NLO counterterms [Degrande]

- **Step 4:** Reimport the counterterms into FeynRules, and export them together with the tree-level vertices in the UFO format.

```
Get[ "model.nlo" ]

WriteUFO[ L, UVCounterterms -> UV$vertlist,
             R2Vertices -> R2$vertlist ]
```



The NLO Force awakens!

NLOCT - NLO counterterms [Degrande]

- Step 4: Reimport the counterterms into FeynRules, and export them together with the tree-level vertices in the UFO format.

```
Get[ “model.nlo” ]

WriteUFO[ L, UVCounterterms -> UV$vertlist,
             R2Vertices -> R2$vertlist ]
```

- The result is a UFO file that can immediately be used by MadGrapgh 5/aMC@NLO to produce events at NLO accuracy!

Summary

- Implementing a New Physics into a matrix element generator can be a tedious and error-prone task.
- FeynRules tries to remedy this situation by providing a Mathematica framework where a new model can be implemented starting directly from the Lagrangian.
- There are no restrictions on the model, except
  - ➡ Lorentz and gauge invariance
  - ➡ Locality
  - ➡ Spins: 0, 1/2, 1, 3/2, 2, ghosts

- Try it out on your favourite model!
  http://feynrules.irmp.ucl.ac.be/