FeynRules Tutorial

The aim of this tutorial is to implement a simple extension[1] of the SM into FeynRules In the following we give detailed instructions of how to implement the model into FeynRules and how to obtain the corresponding model files for MadGraph 5.

1 The model

We add two real scalar fields, ϕ_1 and ϕ_2. They are singlets under all SM gauge groups. Their mass terms are[2]:

$$\mathcal{L}_{kin,scalar} = 1/2\partial_\mu\phi_1\partial^\mu\phi_1 + 1/2\partial_\mu\phi_2\partial^\mu\phi_2 - \frac{m_1^2}{2}\phi_1^2 - \frac{m_2^2}{2}\phi_2^2 - m_{12}^2\phi_1\phi_2\,. \tag{1}$$

We will call mass eigenstates Φ_1 and Φ_2, and their masses M_1 and M_2, respectively, and we will assume $M_1 < M_2$.

We add two Dirac fermion fields, U and E. Their SM quantum numbers are those of the SM u_R and e_R, respectively. These fields have mass terms

$$\mathcal{L}_{dirac,mass} = M_U\overline{U}U + M_E\overline{E}E \tag{2}$$

They interact with scalars via

$$\mathcal{L}_{FFS} = \lambda_1\,\phi_1\,\overline{U}P_R t + \lambda_2\,\phi_2\,\overline{U}P_R t + \lambda_1'\,\phi_1\,\overline{E}P_R e + \lambda_2'\,\phi_2\,\overline{E}P_R e\; + H.c., \tag{3}$$

where t and e are the SM top-quark and electron fields. Note that there is a $\mathbb{Z}_2$ symmetry under which all fields we added $(\phi_{1,2}, U, E)$ flip sign, while all SM fields do not, so the new particles must be pair-produced and the lightest new particle (LNP) is stable. This same $\mathbb{Z}_2$ also forbids $U-u$ and $E-e$ mixing via Yukawas with the SM Higgs.

We will assume the following ordering of masses:

$$M_U > M_2 > M_E > M_1\,, \tag{4}$$

so that Φ_1 is the LNP. The goal of the tutorial is to simulate with MadGraph 5 the process

$$p\,p \to \overline{U}\,U\,, \tag{5}$$

at a 8 TeV LHC, and the subsequent U decays

$$\begin{aligned} &U \to t\,\Phi_1\,, \\ &U \to t\,\Phi_2\,, \quad \Phi_2 \to e\,E\,, \quad E \to e\,\Phi_1\,. \end{aligned} \tag{6}$$

[1] The model used for this tutorial is based on the model presented in the tutorial of arXiv:1209.0297.

[2] All Lagrangian parameters, here and below, are assumed to be real

2 Preparation of the model file

First, FeynRules can be download from https://feynrules.irmp.ucl.ac.be and requires Mathematica 7 or above. As the model we are going to implement is a simple extension of the SM, it is not necessary to start from scratch, but we can use the implementation of the SM included in the folder `/Models/SM`. We therefore start by making a copy of this folder, in order to keep a clean version of the SM. To do so, change directory to the `Models` subdirectory and make a copy of the SM folder, before going into the new directory

```
cd Models
cp -r SM Tutorial
cd Tutorial
```

Even though the implementation is based on the model file `SM.fr` for the Standard Model, the SM sector of the model will be implemented into a separate file that will simply be loaded on top of `SM.fr`. We therefore start by opening a blank text file called `Tutorial.fr`. You can start by personalizing the model file by including a name for you model, the name of the author, *etc.*,

```
M$ModelName = "Tutorial";

M$Information = {Authors      -> {"C. Duhr"},
                 Version      -> "1.0",
                 Date         -> "27. 02. 2012",
                 Institutions -> {"ETH Zurich"},
                 Emails       -> {"duhrc@itp.phys.ethz.ch"}
             };
```

Note that his information is purely optional and could be omitted.

3 Implementation of the new parameters

We start by implementing the new parameters. The model we are considering depends on 9 new parameters, which we assume to be real in the following. FeynRules distinguishes between two types of parameters, the so-called *external* parameters given as numerical inputs and the *internal* parameters, related to other external and/or internal parameters via algebraic expressions. All the parameters of the model, both external and internal, are given in the FeynRules model file as a list named `M$Parameters`. Note that if an internal parameter x depends on some other parameter y, then y should appear in `M$Parameters` *before* the internal parameter x.

The new external parameters of the model are

- 5 mass parameters: m_1, m_2, m_{12}, M_U, M_E.
- 4 coupling constants: λ_1, λ_2, λ_1', λ_2'.

Note however that there is a difference between the mass parameters in the scalar and fermonic sectors: while the masses in fermonic sector are physical masses, the mass matrix in the scalar sector is not diagonal. For this reason, we will not discuss in the following the fermion masses M_U and M_E, as they will be defined together with the particles rather than as parameters of the model.

Let us now turn to the definition of the mass parameters in the scalar sector. The masses m_1, m_2 and m_{12} will be denoted in the FeynRules model file by `MM1`, `MM2` and `MM12`. In the following we only show how to implement `MM1`, all other cases being similar. `MM1` corresponds to the following entry in the list `M$Parameters`,

```
M$Parameters = {
    ...
    MM1 == {
           ParameterType -> External,
           Value         -> 200
         },
    ...
}
```

The first option tags `MM1` as an external parameter, while the second option assign a value of 200GeV to m_1. We stress that this numerical value can be changed later on in the matrix element generators.

The masses in the scalar sector are not the physical masses, because the mass matrix is not diagonal. In order to obtain the physical masses, we need to diagonalize the mass matrix

$$\begin{pmatrix} m_1^2 & m_{12}^2 \\ m_{12}^2 & m_2^2 \end{pmatrix}. \tag{7}$$

In the following, we denote the eigenvalues by `MPe1` and `MPe2`. In addition, we need to introduce a mixing angle θ (`th`) relating the fields ϕ_i to the mass eigenstates Φ_i by,

$$\begin{pmatrix} \phi_1 \\ \phi_2 \end{pmatrix} = \begin{pmatrix} -\sin\theta & \cos\theta \\ \cos\theta & \sin\theta \end{pmatrix} \begin{pmatrix} \Phi_1 \\ \Phi_2 \end{pmatrix}. \tag{8}$$

As in this case the mass matrix is only two-dimensional, we can compute the eigenvalues and the mixing angle analytically, and simply implement the analytical formulas into FeynRules. The implementation follows exactly the same lines as for the masses m_1, m_2, m_{12}, with the only differences that

1. the `ParameterType` is `Internal` (as these parameters are dependent on the external mass parameters,
2. the `Value` is given by an analytical expression (in Mathematica syntax).

Next we turn to the implementation of the new coupling constants, which we will call `lam1`, `la21`, `lam1p`, `lam2p`. They are all external parameters, and thus the implementation follows exactly the same lines as the implementation of the mass parameters, with only one modification: some matrix element generators, like for example MadGraph, keep track of the types of couplings that enter a process. This allows for example to generate a process by only taking into account QCD-type vertices, and to neglect all QED-type vertices. For this reason, it is mandatory to tell the matrix element generator how the new coupling constants should be counted. As in this case we are dealing with new classes of couplings which are *a priori* independent of QCD or QED interactions, we simply assign a new tag, called *interaction order*, to the coupling via the option

```
InteractionOrder ->  {NP, 1}
```

The name of the tag (`NP` for "new physics" in this case) can be chosen freely. The above option instructs the matrix element generator to count one unit of "`NP`" for each new coupling.

4 Implementation of the fields

In this section we discuss the implementation of the new fields. The implementation is similar to the implementation of the parameters, *i.e.*, all the fields are entries of a list called `M$ClassesDescription`. In Tab. 1 we show the names of the fields used in the implementation[3].

U	E	ϕ_1	ϕ_2	Φ_1	Φ_2
`uv`	`ev`	`pi1`	`pi2`	`p1`	`p2`

Table 1: Symbols used for the fields in the FeynRules implementation.

We illustrate the implementation of a new field on the example of the particle U (`uv`). The definition of the particle corresponds to an entry in `M$ClassesDescription` of the following form

```
M$ClassesDescription = {
      ...
      F[100] == {
          ClassName      -> uv,
          SelfConjugate  -> False,
          Indices        -> {Index[Colour]},
          QuantumNumbers -> {Y -> 2/3, Q -> 2/3},
          Mass           -> {Muv, 500},
          Width          -> {Wuv,1}
        },
       ...
 }
```

The meaning of this definition is as follows: each particle class has a name of the form `X[i]`, where `X` is related to the spin of the field (See Tab. 2), and `i` is an integer that labels the classes. Note that `i` can be chosen freely, as long as there is no name clash with an already existing class (in this case, there could be a name clash with the SM particles already defined in `SM.fr`). Each class has a series of options

1. `ClassName`: the symbol by which the particle will be represented in the Lagrangian.
2. `SelfConjugate`: a boolean variable, indicating whether the particle has an antiparticle (`False`) or not (`True`). If the field is not selfconjugate, a symbol for the antiparticle is automatically defined by appending "`bar`" to the name of the particle. In the above example the antiparticle associated to `uv` will be denoted by `uvbar`. Note that in the case of fermions the symbol for the antiparticle refers to the quantity $\bar{U}$ rather than $U^\dagger$.
3. `Indices`: All indices carried by the field. The available types of indices from the SM implementation are
   - `Generation`: fermion flavor index ranging from 1 to 3,

[3]Note that the symbol `u`, `e` and `phi` are already in use in the SM implementation. We also avoid using simply uppercase letters, as some matrix element generators are case insensitive.

Spin	0	1/2	1	2	ghost
Symbol	S	F	V	T	U

Table 2: Available particle classes in FeynRules.

- `Colour`: fundamental color index ranging from 1 to 3,
- `Gluon`: adjoint color index ranging from 1 to 8,
- `SU2W`: adjoint $SU(2)_L$ index ranging from 1 to 3.

4. `QuantumNumbers`: a list of all $U(1)$ charges carried by the field. In the SM implementation the following $U(1)$ charges are already defined
   - `Y`: weak hypercharge,
   - `Q`: electric charge.
5. `Mass`: the mass of the particle. It is a list of two elements, the first being the symbol used to represent the mass, and the second its value (in GeV). If the value of the mass is obtained from some analytic expression defined as an internal parameter with the same symbol (as is the case for example in the scalar sector of the model), the value is set to `Internal`.
6. `Width`: the width of the particle. The definition is similar to `Mass`. Note that as we do not yet know the widths of the new particles, we simply set it for now to 1GeV, and will determine its exact value later using one of the matrix element generators.

The implementation of the other mass eigenstates (`ev`, `p1`, `p2`) is similar, so we do not discuss it here, except noting that for the scalar mass eigenstates, the masses are defined as internal parameters, and so when defining the mass of the particle, we have to put for example

```
Mass   ->   {MPe1, Internal}
```

Let us comment on the implementation of the interaction eigenstates ϕ_i. Indeed, while the matrix element generators work exclusively at the level of the mass eigenstates, the interaction eigenstates are in general useful to write the Lagrangian in a compact form. It is therefore useful to define also the fields for the interaction eigenstates ϕ_i. The definition of these fields is similar to the mass eigenstates, *e.g.*,

```
S[100] == {
      ClassName     -> pi1,
      SelfConjugate -> True,
      Indices       -> {},
      Unphysical    -> True,
      Definitions   -> {pi1  ->  - Sin[th] p1 + Cos[th] p2}
     },
```

First, note that the `Mass` and `Width` options are omitted[4], as these fields are not mass eigenstates. This last fact is made explicit by the option

[4]The `QuantumNumbers` option is also omitted, but for the simple reason that the fields ϕ_i do not carry any $U(1)$ charges.

```
Unphysical    -> True,
```

which instruct FeynRules not to output this field to a matrix element generator. Finally, the relation of the field `pi1` to the mass eigenstates is simply given as a Mathematica replacement rule in the `Definitions` option.

5 Implementation of the Lagrangian

The definitions in the model file being complete, we now turn to the implementation of the Lagrangian. This can be done either in the model file, or directly in a Mathematica notebook. Here we use the latter approach, and we start by opening a new notebook and load the FeynRules package,

```
$FeynRulesPath =  SetDirectory["<your path>/feynrules"];
<< FeynRules`
```

Next we have to load the model files, both for the SM and for the new sector. We first change the default directory in Mathematica to the `Tutorial` directory, and then load the model files via the `LoadModel` command[5],

```
SetDirectory[ $FeynRulesPath <> "/Models/Tutorial"]
FR$Parallel = False;

LoadModel["SM.fr", "Tutorial.fr"]
```

Note that the new model file should be loaded *after* `SM.fr`. Furthermore, we also load two additional files, which restrict the first two fermion generations to be massless and the CKM matrix to be diagonal,

```
LoadRestriction["DiagonalCKM.rst", "Massless.rst"]
```

The new Lagrangian consists of three parts,

$$\mathcal{L} = \mathcal{L}_{scalar,kin} + \mathcal{L}_{fermion,kin} + \mathcal{L}_{Yuk}\,. \tag{9}$$

The kinetic terms for the new scalars can easily be implemented by using the symbols for the gauge eigenstates and the mass parameters defined in the model file, as well as the symbol for the space-time derivative ∂_μ in FeynRules, `del[ ..., mu]`. As an example, we have

$$\frac{1}{2}\partial_\mu\phi_1\partial^\mu\phi_1 - \frac{1}{2}m_1^2\phi_1^2$$

```
1/2 del[pi1, mu] del[pi1, mu] - 1/2 MM1^2 pi1^2
```

The kinetic terms for the fermions can be implemented in a similar way. However, as the fermions are charged under the SM gauge group, we have to use the covariant derivative `DC` rather than the space-time derivative `del`. Furthermore, we have to use a "." instead of an ordinary multiplication in order to take the non-commuting nature of the fermions into account. As an example, we have

$$i\,\bar{U}\gamma^\mu D_\mu U - M_U\bar{U}U$$

```
I uvbar.Ga[mu].DC[uv, mu] - Muv uvbar.uv
```

[5]The "`<>`" operator in Mathematica is just string concatenation, *i.e.*, in our case it simply appends `/Models/Tutorial` to the directory of FeynRules.

where `Ga[mu]` is the FeynRules symbol for the Dirac matrix γ^μ. Finally, the Yukawa interactions can be implemented in the same way as the kinetic terms for the fermions, *e.g.*,

$$\lambda_1\,\phi_1\,\bar{U}P_+t$$
```
lam1 pi1 uvbar.ProjP.t
```

where `u` denotes the u quark field defined in `SM.fr` and `ProjP` denotes the right chiral projector (the left projector is denoted by `ProjM`). Note that FeynRules contains a function `HC[ ]` which allows to obtain the hermitian conjugate of an expression in an automated way.

6 Computing the Feynman rules and running the interfaces

Our model implementation is now complete, and so we can compute the Feynman rules. The Feynman rules of the new sector can be obtained by issuing the command

```
vertices = FeynmanRules[ LNew ];
```

where `LNew` is the name of the variable that contains the new Lagrangian.

You can use the vertices to compute analytically the partial widths and the branching ratios of the new particles. The width are computed by issuing the command

```
ComputeWidths[vertices];
```

The partial widths of all two-body decays are computed and stored in some internal format. They can be accessed via, e.g.,

```
PartialWidth[ {uv, t, p1} ]
```

which returns the partial width for U decaying into u, Φ_1. The functions `TotWidth[ ]` and `BranchingRatio[ ]` work in a similar fashion. As an application, compute the partial widths $\Gamma_{U\to t\,\Phi_1}$, $\Gamma_{U\to t\,\Phi_2}$, $\Gamma_{\Phi_2\to e\,\bar{E}}$, $\Gamma_{E\to e\Phi_1}$. The command `NumericalValue[ ]` returns the numerical value an expression and can be used to get the value of the branching ratios and so on.

The Feynman rules can be written to file in a format suitable to various matrix element generators by using the FeynRules interfaces. In this tutorial, we will use the interfaces to the UFO, and thus to MadGraph 5, which can be called via

```
WriteUFO[ LSM + LNew ];
```

where `LSM` is the SM Lagrangian implemented in `SM.fr`. Note that the SM implementation is available in both Feynman gauge and unitary gauge. A boolean variable `FeynmanGauge` allows to switch between both gauges.

7 Importing the model into MadGraph 5

After successfully running the UFO interface, a directory `Tutorial_UFO` has been created in the `/Models/Tutorial/` directory. This directory contains all the UFO files needed to run the model in MadGraph 5. To import the model into MadGraph it is enough to copy the UFO directory into the `/models/` subdirectory of MadGraph 5,

```
cp -r  Tutorial_UFO <your MadGraph directory>/models/
```

The MadGraph 5 shell the new model can now be called in the same way as any other built in model,

```
mg5> import model Tutorial_UFO
```

8 Checking the model

The next step is to check that the implementation of the model is correct and safe to use. A first simple check is to use FeynRules itself to test if the Lagrangian is actually hermitian via the command

```
CheckHermiticity[ LSM + LNew ];
```

The next check is to use the built-in test suite of MadGraph 5 to check that processes with external gluons are Lorentz and/or gauge-invariant, *e.g.*, you can try to issue in the MadGraph shell the following command (after importing the model)

```
mg5> check gauge   g g > uv uv~
mg5> check lorentz_invariance g g > uv uv~
```

Both checks as well as a few more can be done at once using

```
mg5> check g g > uv uv~
```

Finally, compute and check the values of the partial width computed in Section 6.

9 Do phenomenology!

You should be all set now to study the phenomenology of your new model! As an example, try to generate some events for $p\,p \to U\,\overline{U}$, including decays.