Mass matrices generation and diagonalization within FeynRules

Adam Alloul, J. D'Hondt, K. De Causmaecker, B. Fuks, M. Rausch de Traubenberg

IPHC / Université de Strasbourg

November 6^{th}, 2012

Outline

FeynRules in a nutshell (1)

A framework for LHC analyzes based on FeynRules to:

- Develop **new models**.
- implement (and validate) new models in Monte Carlo tools.
- **Facilitate** phenomenological investigations of the models.
- Test the models **against data**.

Main features

- FeynRules is a **Mathematica** package.
- FeynRules derives **Feynman rules** from a lagrangian.
- Requirements: locality. Lorentz and gauge invariance.
- Supported fields: **scalar, fermion, vector, tensor, ghost, superfield**.
- Interfaces: export the Feynman rules to Monte Carlo generators.

FeynRules in a nuthsell (2)

Interfacing FeynRules with MC tools

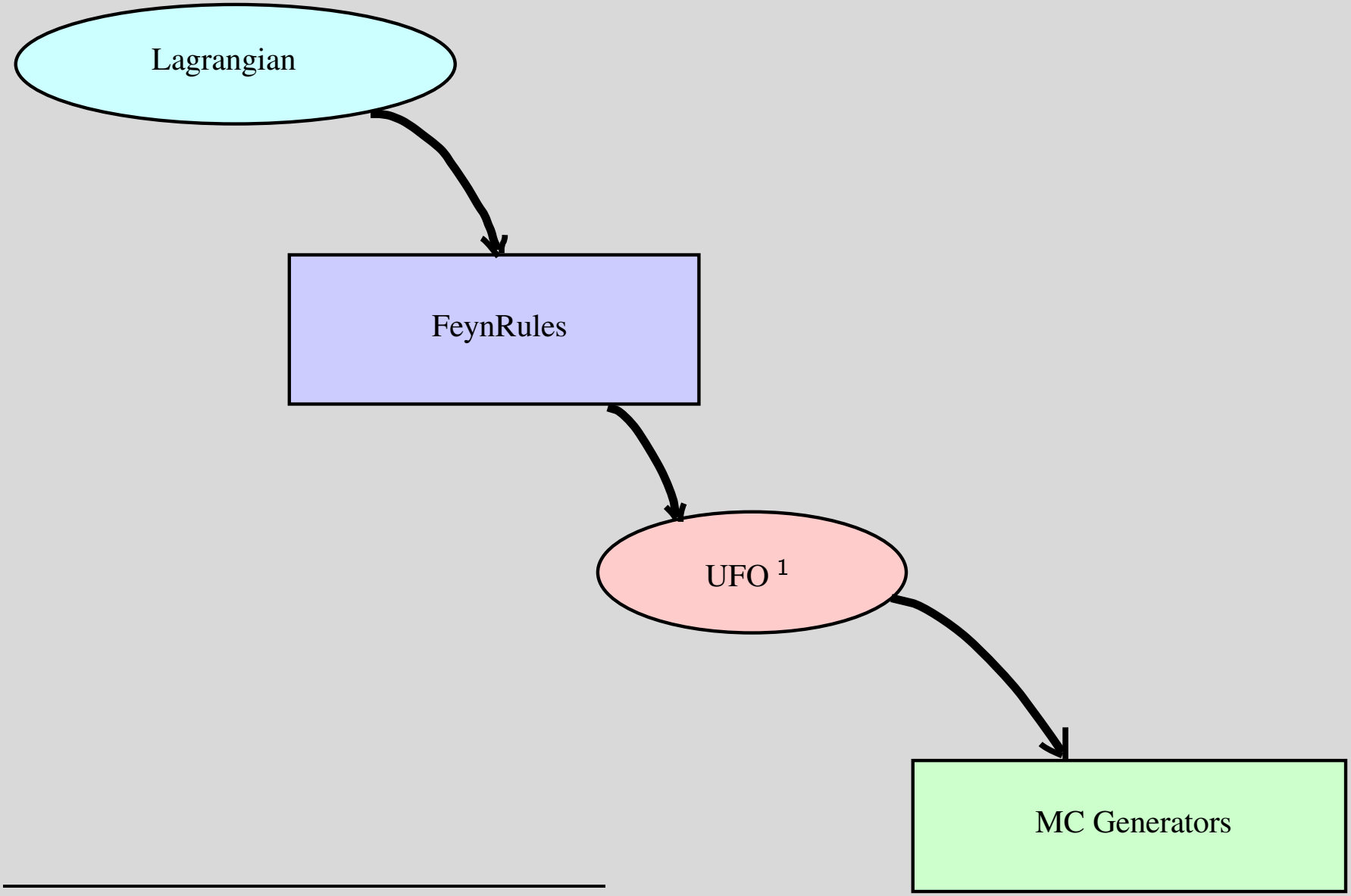
CalcHep
FeynArts
MadGraph and GoSam
Sherpa
Whizard

$\Rightarrow$ Parton showering, hadronization, detector simulation, **Analysis!!**

Code status

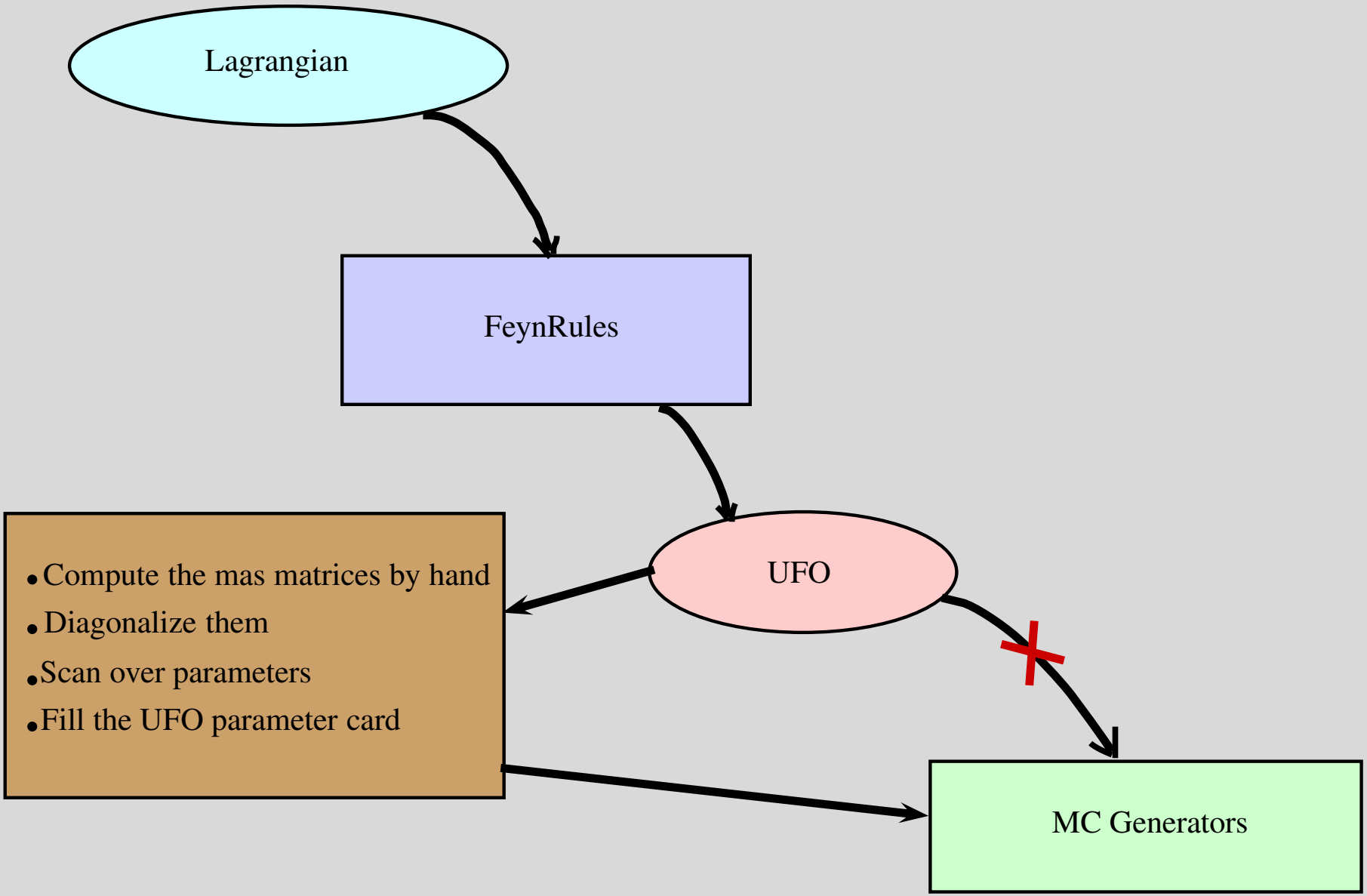
- Current version: 1.6.0
- Downloadable from https://feynrules.irmp.ucl.ac.be/
- Current online model database:
  - Standard Model and simple extensions (12)
  - Supersymmetric models (4)
  - Extra-dimensional models (4)
  - Strongly coupled and effective field theories (4)

From Lagrangian to events: standard steps



[1] or any model file dedicated to a given MC tool

When masses are UNKNOWN



Code overview (1)

The idea

- Adapt the FeynRules model file for field mixings declaration.
- Create a routine that generates the mass matrices **analytically**.
- Generate a C++ code to diagonalize the mass matrices **numerically**.
- Output a paramcard in the **Les Houches format**.

Constraints

- The routine should work for **ANY** quantum field theory.
- Minimal changes in the FeynRules model file.
- Possibility to use the numerical code as a **standalone** package.
- Fast C++ code.

The result

- FeynRules model files **simplified**.
- Mixings declared in a **new global variable**: `M$MixingsDescription`.
- To generate the mass matrices just type `ComputeTreeLevelMassMatrix[lagrangian]`.
- Numerical code generated with `WriteMassDiagonalizationInterface[lagrangian]`.
- C++ code uses the GSL libraries.

Code overview (2)

Vacuum expectation values (vevs)*

```
M$vevs = { {field1, vev1} , {field2, vev2}, . . .}
```

Mixings declaration: General case

```
M$MixingsDescription={
   Mix["Id"] == {
    GaugeBasis   -> {gauge eigenstates},
    MassBasis    -> {mass eigenstates},
    MixingMatrix -> name of the matrix,
   BlockName    -> SLHAMix   ||     Value   -> value}  }
```

Don't need to declare the mixing parameters anymore

* a factor of $\frac{1}{\sqrt{2}}$ is added automatically during calculations

Code overview (2)

Mixings declaration: Gauge Bosons or charged scalars

```
M$MixingsDescription={
   Mix["Field1"] == {
      GaugeBasis   ->{ gauge eigenstates},
      MassBasis    ->{ mass eigenstates },
      MixingMatrix -> Mass,
      BlockName    -> Mix }}
```

Mixings declaration: Neutral scalars

```
M$MixingsDescription={
   Mix["Scalar1"] == {
      GaugeBasis   ->{ scalar gauge eisgenstates},
      MassBasis    ->{ {scalars} , {pseudo-scalars }},
      MixingMatrix -> { ScalarMass , PseudoScalarMass},
      BlockName    -> { ScalarMix , PseudoScalarMix}}
```

Code overview (2)

Mixings declaration: Dirac fermions

```
M$MixingsDescription={
   Mix["Dirac1"] == {
      GaugeBasis   ->{ {Left handed fermions},{right handed fermions}},
      MassBasis    ->{ Dirac mass eigenstates },
      MixingMatrix -> DiracMass,
      BlockName    -> DiracMix }}
```

Mixings declaration: Weyl Fermions

```
M$MixingsDescription={
   Mix["Weyl1"] == {
      GaugeBasis   ->{ Weyl gauge eigenstates},
      MassBasis    ->{ Weyl mass eigenstates },
      MixingMatrix -> WeylMass,
      BlockName    -> WeylMix}}
```

Example: Higgs sector of the left-right symmetric model

Model description

- Gauge group $SU(3)_c \times SU(2)_L \times SU(2)_R \times U(1)_{B-L}$
- Right-handed Standard Model's fermions promoted to doublets of $SU(2)_R$
- Rich higgs sector:
  - 2 $SU(2)$ triplets δ_L and δ_R with $B - L = +2$
  - 1 $SU(2)$-bidoublet Φ with $B - L = 0$

Higgs sector mixings

Triplets can be written in **matrix form**

$$\frac{1}{\sqrt{2}}\sigma_k \delta^k_{L,R} = \begin{pmatrix} \Delta^+_{L,R} & \Delta^{++}_{L,R} \\ \Delta^0_{L,R} & -\Delta^+_{L,R} \end{pmatrix}$$

Bidoublet expression

$$\Phi = \begin{pmatrix} \phi^0 & \phi^+ \\ \phi^- & \phi'^0 \end{pmatrix}$$

Physical states

- 4 neutral scalars and pseudo-scalars from $\{\Delta^0_L, \Delta^0_R, \phi^0, \phi'^0\}$ mixing.
- 4 simply charged scalars from $\{\Delta^+_L, \Delta^+_R, \phi^+, \phi^-\}$ mixing.
- 2 doubly charged scalars from $\{\Delta^{++}_L, \Delta^{++}_R\}$ mixing.

Example: Higgs sector of the left-right symmetric model

Higgs sector mixings

Triplets can be written in **matrix form**

$$\frac{1}{\sqrt{2}}\sigma_k \delta^k_{L,R} = \begin{pmatrix} \Delta^+_{L,R} & \Delta^{++}_{L,R} \\ \Delta^0_{L,R} & -\Delta^+_{L,R} \end{pmatrix}$$

Bidoublet expression

$$\Phi = \begin{pmatrix} \phi^0 & \phi^+ \\ \phi^- & \phi'^0 \end{pmatrix}$$

Vacuum expectation values

```
M$vevs = { {DeltaL0,vL}, {DeltaR0,vR}, {h1[1,1],v1}, {h1[2,2],v1p} }
```

Triplets in matrix form

```
M$MixingsDescription={
 Mix["2a"] == { MassBasis -> {DeltaLpp,DeltaL0},
                GaugeBasis -> {hL[1],hL[2]},
                Value -> {{1/Sqrt[2],-I/Sqrt[2]},{1/Sqrt[2],I/Sqrt[2]}}},

 Mix["2b"] == { MassBasis -> {DeltaRpp,DeltaR0},
                GaugeBasis -> {hR[1],hR[2]},
                Value -> {{1/Sqrt[2],-I/Sqrt[2]},{1/Sqrt[2],I/Sqrt[2]}}},
```

Example: Higgs sector of the left-right symmetric model

Physical states

- 4 neutral scalars and pseudo-scalars from $\{\Delta_L^0, \Delta_R^0, \phi^0, \phi'^0\}$ mixing.
- 4 simply charged scalars from $\{\Delta_L^+, \Delta_R^+, \phi^+, \phi^-\}$ mixing.
- 2 doubly charged scalars from $\{\Delta_L^{++}, \Delta_R^{++}\}$ mixing.

Mass eigenstates

```
Mix["2c"] == { MassBasis -> {DH1,DH2},
               GaugeBasis -> {DeltaLpp,DeltaRpp},
               MixingMatrix->UHD, BlockName->HDMix},

Mix["2d"] == { MassBasis -> {GP1,GP2,H1,H2},
               GaugeBasis -> {hL[3],hR[3],h1[1,2], h1bar[2,1]},
               MixingMatrix->UHC, BlockName -> HCMix},

Mix["2e"] == { MassBasis -> { {h01,h02,h03,h04} , {G01,G02,a01,a02} },
               GaugeBasis->{ DeltaL0 , DeltaR0 , h1[1,1] , h1[2,2] },
               MixingMatrix->{ UHN , UAN }, BlockName->{ HMix , AMix }}
```

Code overview (3)

- **To generate** the spectrum generator:
  - Load FeynRules and your model
  - Run `WriteMassDiagonalizationInterface[lag]`
- A directory with name `ModelName_MD` is **automatically** created.
- In a Terminal
  - `cd` into it
  - Type `make` and `./RUN`
- A new parameter card is **automatically generated** with all the mixings and masses computed.

Code overview (3)

- To generate the spectrum generator:
  - Load FeynRules and your model

Available commands in FeynRules

- `ComputeTreeLevelMassMatrices[lag]`
  Computes all the mass matrices that have no `Value`
- `MassMatrix["Id"]`
  Displays the mass matrix for mixing `"Id"`
- `GaugeBasis["Id"]`
  Displays the gauge basis for mixing `"Id"`
- `MassBasis["Id"]`
  Displays the mass basis for mixing `"Id"`
- `WriteMassDiagonalizationInterface[lag]`
  Writes the C++ code for diagonalization

Code overview (3)

- A directory with name `ModelName_MD` is **automatically** created.

Structure of the directory

Directory	Model dependent files	Description
`bc`	`BoundaryConditions.dat`	SLHA-paramcard of the model
`inc`	`Parameters.hpp`	Paremeters and mass matrices declaration
`src`	`Parameters.cpp`	Definition of the parameters
`out`	`paramcard_ModelName.dat`	Generated parameter card of the model
	`main.cpp`	Definitions of the mass matrices, PDG-Ids

`BoundaryConditions.dat` can be **updated** by the user at will.

`paramcard_ModelName.dat` can be used as parameter card for MC tools.

Tested models and Benchmarks

Tested on

- MATHEMATICA v7 and v8
- MacOSX, Linux

Benchmarks for the tested models

Model	Mass Matrices generation	C++ code generation	make ; ./RUN
2HDM	$\sim$ 5s	$\sim$ 1 s	$\sim$ 5s
LRSM	12 for 7 mass matrices	$\sim$ 1 s	$\sim$ 5s
MSSM	200s for 8 mass matrices	$\sim$ 1 s	$\sim$ 5s

Conclusion and perspectives

Status of the code

- Mass matrix generation is now available as a module of FeynRules.
- The generated C++ code is a **standalone package**.
- Requirements: Mathematica, a C++ compiler and GSL libraries.
- Paper in preparation.

Future plans

- Generate 1-loop mass matrices.
- Interface with MadGraph.