# An Introduction to FeynRules

Liam Moore[1]

[1] CP3 Louvain-la-Neuve

MC4BSM 2017

# Contents

# Contents

# Overview of FeynRules

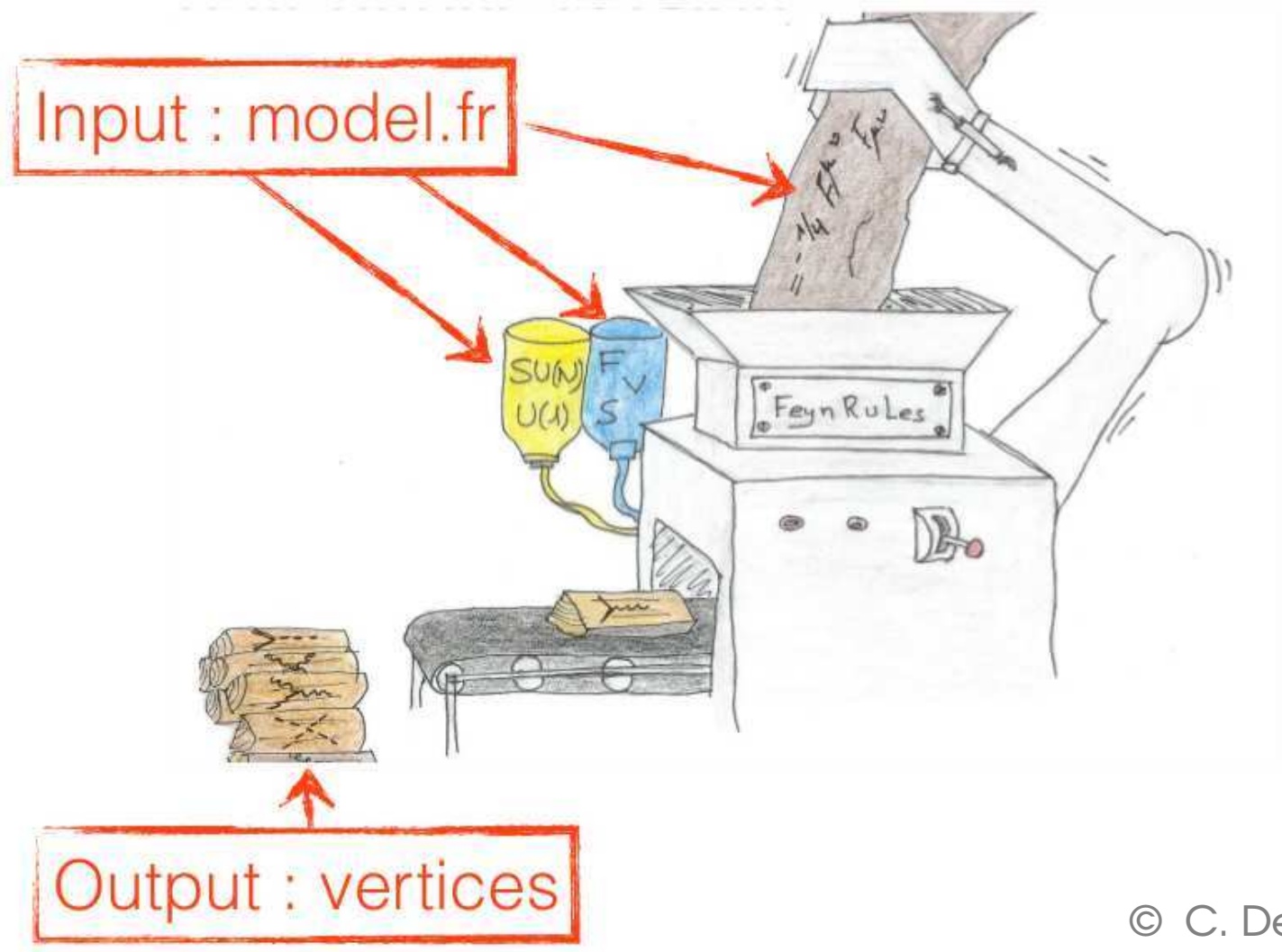
FeynRules is a Mathematica package which allows the implementation of general, user-input QFT models, requiring only:

- Lorentz and Gauge Invariance, Locality
- Massive/massless Spin $0, \frac{1}{2}, 1, \frac{3}{2}$ or $2$ fields (incl. Superfields)

and from a given model file, will:

- Derive the Feynman Rules
- Generate (N)LO input files for both matrix element generators and analytic tools
- Automate e.g. diagonalization of mass matrices, calculation of $1 \to 2$ decay widths, mixings. . .

Available at: `http://feynrules.irmp.ucl.ac.be/`.

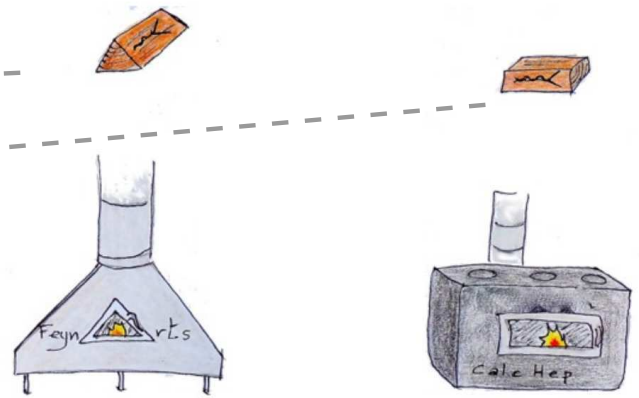



© C. Degrande

# FeynRules Interfaces

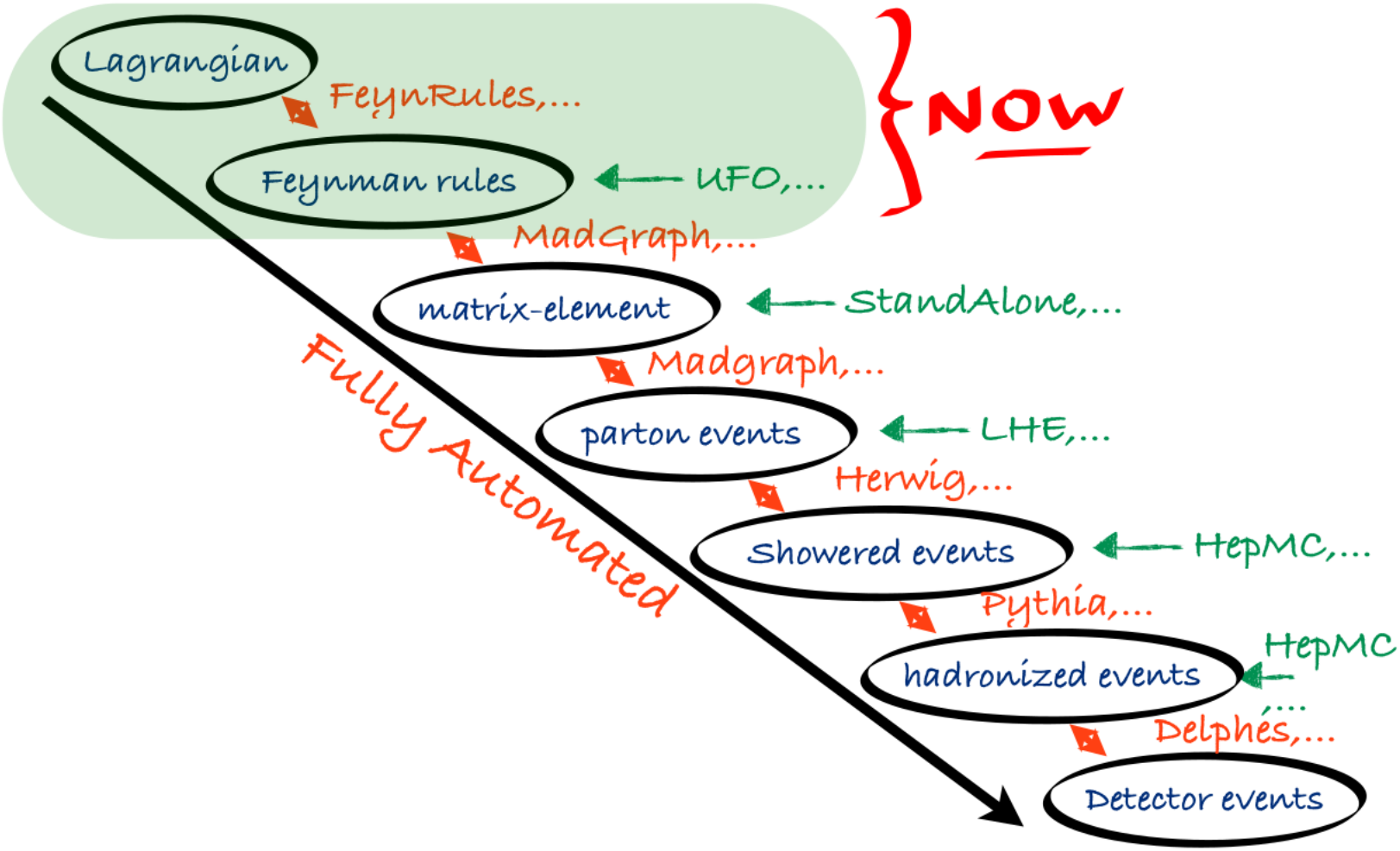
Vertices derived from model can be exported via a set of interfaces, many tools now support common Universal FeynRules Output (UFO) format:

- CalcHep/CompHep
- FeynArts/FormCalc
- GoSam
- Herwig
- MadGraph5_aMC@NLO
- Sherpa
- Whizard/Omega




© C. Degrande

# FeynRules in the BSM Toolchain



# Contents

# The Model File

A FeynRules **model file** is a text file (.fr) written in Mathematica syntax.

Examples can be found on the public model repository:

https://feynrules.irmp.ucl.ac.be/wiki/ModelDatabaseMainPage.

```
(* Model Information *)

M$ModelName = "MC_4_BSM";

M$Information = {

        Authors -> {"Mr X.", "Ms. Y"},
        Institutions -> {"UC Louvain"},
        Emails -> {"X@uclouvain.be", "Y@uclouvain.be},
        Date -> "13.05.2017",
        References -> {"reference 1", "reference 2"},
        URLs -> {"http://feynrules.irmp.ucl.ac.be"},
        Version -> "10.0"

};
```

# Structure of a Model File

A model file defines:

- Gauge groups:
- Indices:
- Field content and quantum numbers:
- Parameters:
- $\mathcal{L}$:

```
M$GaugeGroups
= { myGaugeGroup == {...}, ...}

IndexRange[myGaugeIndex]
= ...
IndexStyle[myGaugeIndex, A]

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

M$Parameters
= { myCoupling == {...},
    myScalarMass == {...},
    myVectorMass == {...}, ...}

LagKinS := ...
LagKinV := ...
LagInt := ...
FullLag := LagKinS + LagKinV + ...
```

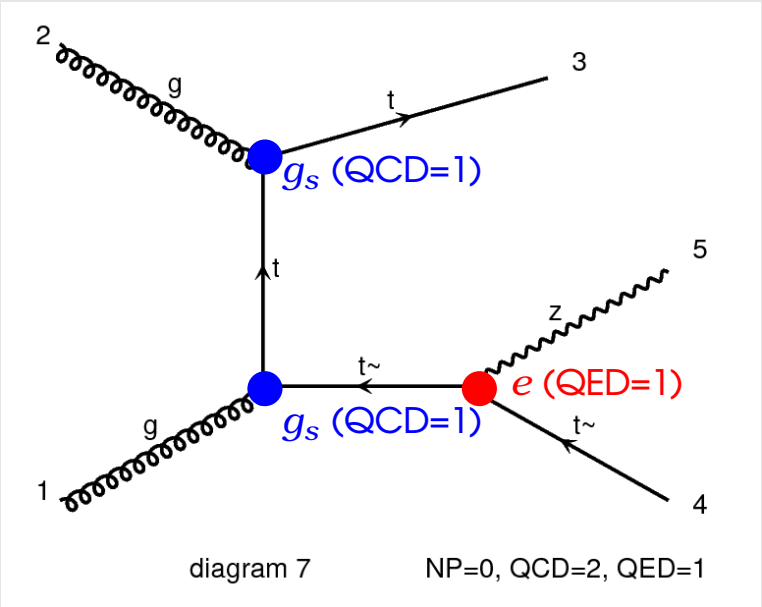
# Implementing Parameters

- `External` parameters are numerically specified by `Value` as inputs of the model
- `Internals` specified by analytic dependence on these
- `InteractionOrder` specifies power dependence on a type of coupling
- LHA card labels
- $\mathbf{g_w} \to \frac{e}{s_w} = \frac{\sqrt{4\pi}}{s_w\sqrt{\alpha_{EM}^{-1}}}$

```
aEWM1 == {
  ParameterType    -> External,
  BlockName        -> SMINPUTS,
  OrderBlock       -> 1,
  Value            -> 127.9,
  InteractionOrder -> {QED,-2},
  Description
  -> "Inverse EW coupling @ Z pole"
},
ee == {
  ParameterType    -> Internal,
  Value            -> Sqrt[4 Pi/aEWM1],
  InteractionOrder -> {QED,1}, ...
},
gw == {
  ParameterType    -> Internal,
  Definitions      -> {gw->ee/sw},
  InteractionOrder -> {QED,1},  ...
}, ...
```

# Implementing Parameters (II)

- Specifying **`InteractionOrder`** for model couplings allows **restriction of diagrams** in matrix element generators




```
gs == {
  ParameterType    -> Internal,
  Value            -> Sqrt[4 Pi aS],
  InteractionOrder -> {QCD,1},
  Description
  -> "Strong coupling @ Z pole",
  ...
},

ee == {
  ParameterType    -> Internal,
  Value            -> Sqrt[4 Pi aEW],
  InteractionOrder -> {QED,1},
  Description
  -> "EM coupling @ Z pole",
  ...
},

...
```

# Implementing Indices

- Fields and (tensor) parameters carry **Index** arguments
- `IndexRange` and `IndexStyle` **set indices' variable names and span**, e.g. Colour has $c_i = \{1, 2, 3\}$
- **(No) Unfold** flags expressions with indices repeated to **(not) explicitly write out sum**, e.g. `Phibar[i]Phi[i]` $\equiv \varphi_i^\dagger \varphi_i = |\varphi_1|^2 + |\varphi_2|^2$

```
(* Def Index Names/Ranges *)

IndexRange[Index[Colour]]
= NoUnfold[Range[3]];

IndexRange[Index[SU2D]]
= Unfold[Range[2]];

IndexRange[Index[Generation]]
= Range[3];

IndexStyle[Colour,     c];
IndexStyle[SU2D,       i];
IndexStyle[Generation, f];
```

# Implementing Tensor Parameters

- Parameters promoted to tensors by defining `Indices`: $V_{ij}^{\text{CKM}} \equiv$ **`CKM[i,j]`**
- `ComplexParameter` & `Unitary` options enforce $V_{ij}^* \neq V_{ij}$ , $V_{ji}^* V_{jk} = \delta_{ik}$
- `TeX` option specifies `TeX` output style: `CKM` $\longrightarrow V_{\text{CKM}}$
- Parameter indices match those carried by fields **`CKM[i,j]`** `dq[sp,j,c]` $\equiv \boldsymbol{V_{ij}^{\text{CKM}}} (d,s,b)_{\boldsymbol{j}}^T$

```
CKM == {
  ParameterType -> Internal,
  Indices       -> {Index[Generation],
                    Index[Generation]},
  ComplexParameter -> True,
  Unitary       -> True,
  Value
  -> {CKM[1,1] ->  Cos[cabi],
      CKM[1,2] ->  Sin[cabi],
      CKM[2,1] -> -Sin[cabi],
      CKM[2,2] ->  Cos[cabi],
      CKM[1,3] -> 0, CKM[2,3] -> 0,
      CKM[3,1] -> 0, CKM[3,2] -> 0,
      CKM[3,3] -> 1},
  TeX          -> Superscript[V,CKM],
  Description -> "CKM-Matrix"}
};
```

# Implementing Fields (I)

- Field arguments specified by `Indices`
- Lorentz/spin first, **set implicitly by class** (`Index[Spin]`)
- **I.D.** uniquely indexes fields in each class
- Flavour variants: common representations/charges
- Numerical masses/widths: **external** parameters
- Vector/Chiral superfields also supported

```
F[3] == {
  ClassName        -> uq,
  ClassMembers     -> {u, c, t},
  FlavorIndex      -> Generation,
  SelfConjugate    -> False,
  PDG              -> {2, 4, 6},
  QuantumNumbers   -> {Q -> 2/3},
  Indices
  -> {Index[Generation], Index[Colour]},
  Width
  -> {0, 0, {WT,1.50833649}},
  Mass
  -> {{MU, 2.5*^-3}, {MC,1.3}, {MT,172}},
},
```

$$\texttt{uq[sp, f, c]} \equiv \{u_{sp,c}, c_{sp,c}, t_{sp,c}\}$$

| **Class** | S | W/**F** | V | RW/R | T | U | C/V SF |
|---|---|---|---|---|---|---|---|
| **Spin** | 0 | 1/2 | 1 | 3/2 | 2 | *Gh.* | – |

# Implementing Fields (II)

- Often $\mathcal{L}$ written in interaction eigenstates/unbroken symmetry...
- Fields with the `Unphysical->True` option are **replaced** by mass eigenstates according to `Definitions`
- $\varphi \to \frac{1}{\sqrt{2}}(-iG_+, v+\mathbf{H}+iG_0)^T$
- `SelfConjugate` automatically generates 'bar' antifields with opposite `QuantumNumbers`

```
S[1] == {
  ClassName        -> H,
  SelfConjugate    -> True,
  Mass             -> {MH,125},
  ...
}, ...

S[11] == {
  ClassName       -> Phi,
  Unphysical      -> True,
  Indices         -> {Index[SU2D]},
  FlavorIndex     -> SU2D,
  SelfConjugate   -> False,
  QuantumNumbers -> {Y -> 1/2},
  Definitions
  -> { Phi[1] -> -I GP,
       Phi[2] -> vev + H + I G0/Sqrt[2]
     }
 } ...
```

# Implementing Gauge Groups

- Predefined $F^A_{\mu\nu}$ and $D_\mu\phi$, form fixed in **M$GaugeGroups** from fields' `QuantumNumbers` & `Indices`
- `DC[uq,mu]` $\equiv D_\mu u = (\delta_{cd}\partial_\mu - ig_s G^a_\mu T^a)u$
- `FS[G,mu,nu,a]` $\equiv G^a_{\mu\nu} = \partial_\mu G^a_\mu - \partial_\nu G^a_\mu + g_s f^a_{bc} G^b_\mu G^c_\nu$
- Naming for e.g. $SU(3)_C$ objects match event generator conventions

```
M$GaugeGroups = {

SU3C == {
Abelian -> False,
CouplingConstant -> gs,
GaugeBoson -> G,
StructureConstant ->  f,
Representations -> {T,Colour},
SymmetricTensor -> dSUN
},

U1Y == { ... },

... }
```

# Implementing Mixings (Optional)

- In lieu of explicit `Definitions` for `MassBasis = Mix x GaugeBasis`, define mixing matrices internally with **`M$MixingsDescription`**
- Run the included ASperGe package **(Alloul et al. 1301.5932)** for a numerical diagonalization:
  `WriteASperGe[`$\mathcal{L}$`];`
  `RunASperGe[];`
- Unspecified mass and $U_{ij}$ values calculated externally in $C++$ and fed into `M$Parameters`

```
M$MixingsDescription = {

Mix["Weak"] == {
MassBasis -> {A, Z},
GaugeBasis -> {B, Wi[3]},
MixingMatrix -> U,
BlockName -> WEAKMIX, ...
}, ...

}
```

$$\begin{pmatrix} A_\mu \\ Z_\mu \end{pmatrix} = \begin{pmatrix} U_{AB} & U_{AW^3} \\ U_{ZB} & U_{ZW^3} \end{pmatrix} \begin{pmatrix} B_\mu \\ W^3_\mu \end{pmatrix}$$

```
Value -> {{gw/Sqrt[gw^2+g1^2],
           -g1/Sqrt[gw^2+g1^2]},
           {g1/Sqrt[gw^2+g1^2],
            gw/Sqrt[gw^2+g1^2]}}
```

# Implementing the Lagrangian

- Products of fields, tensors with repeated indices
- Lorentz / SU2W / Spin / SU2D / Generation / Colour
- Use `Block/Module` to make indices local variables
- Use `Dot` for anticommuting spinor products, `Times` for everything else
- `ExpandIndices`: explicitly write summations over fields with `FlavorIndex` set

```
LagW := Block[{mu,nu,ii},
- 1/4 FS[Wi,mu,nu,ii] FS[Wi,mu,nu,ii]
];
```

$$\mathcal{L}_W = -\frac{1}{4} W^I_{\mu\nu} W_I^{\mu\nu}$$

```
LagQL := Block[{s1,s2,mu,k,f,c},
QLbar[s1,k,f,c].DC[QL[s2,k,f,c],mu]
Ga[mu,s1,s2]
];
```

$$\mathcal{L}_{q_L} = \quad i(\bar{q}_L \not{D} q_L)$$

```
ExpandIndices[
-1/4 FS[Wi,mu,nu,ii] FS[Wi,mu,nu,ii]
,FlavorExpand->{SU2W}]
```

$$-\frac{1}{4} W^I_{\mu\nu} W_I^{\mu\nu} \rightarrow -\frac{1}{2} W^+_{\mu\nu} W^-_{\mu\nu} + \dots$$

# Contents

# Getting Started

In a MATHEMATICA notebook:

- Set the directories:
```
$FeynRulesPath = "Path/To/FeynRules";
$ModelPath = "Path/To/Model";
```
- Load the package:
```
SetDirectory[$FeynRulesPath];
<<FeynRules`
```
- Load the model(s):
```
SetDirectory[$ModelPath];
LoadModel[mymodel.fr,...];
```
- **(Optional)** Load Restrictions:
```
LoadRestriction["Massless.rst"]
```

# Checking Models

Once a model is loaded, FeynRules includes functions to manipulate and check the Lagrangian(s):

`CheckMassSpectrum[` $\mathcal{L}$ `]:`

```
Neglecting all terms with more than 2 particles.
All mass terms are diagonal.
Getting mass spectrum.
Checking for less then 0.1% agreement with model file values.
```

`CheckHermiticity[` $\mathcal{L}$ `]:`

```
Checking for hermiticity by calculating the Feynman rules contained in L-HC[L].
If the lagrangian is hermitian, then the number of vertices should be zero.
Starting Feynman rule calculation.
Expanding the Lagrangian...
Collecting the different structures that enter the vertex.
No vertices found.
0 vertices obtained.
The lagrangian is hermitian.
```

Complete list at: https://arxiv.org/abs/1310.1921

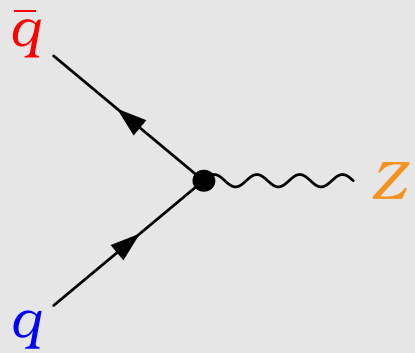
# Feynman Rules

The Feynman Rules are extracted from a Lagrangian as:

```
myVertices = FeynmanRules[ℒ, Options];
```

- Conventions: **momenta incoming**, **mass eigenstates**
- Comes with filtering options, e.g. `MaxParticles`, `Contains`...
- Indices named and numbered according to `IndexStyle` and particle numbers in output list

```
FeynmanRules[LFermions,
SelectParticles -> { uqbar, uq, Z } ]
```



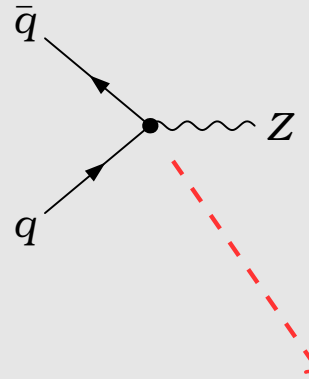

$$\left( \begin{pmatrix} \bar{u}q & 1 \\ uq & 2 \\ Z & 3 \end{pmatrix} \quad \frac{e\, i\, c_w\, \delta_{f_1,f_2}\, \delta_{m,m}\, \gamma^{\mu_3}.\, P_{-s_1,s_2}}{2\, s_w} - \frac{i\, e\, s_w\, \delta_{m,m}\, \delta_{f_1,f_2}\, \gamma^{\mu_3}.\, P_{-s_1,s_2}}{6\, c_w} - \frac{2\, i\, e\, s_w\, \delta_{m,m}\, \delta_{f_1,f_2}\, \gamma^{\mu_3}.\, P_{+s_1,s_2}}{3\, c_w} \right)$$

# $1 \to 2$ Decay Widths

- `ComputeWidths` derives $\Gamma_{1\to 2}$ analytically from vertices & model parameters, stores in list `FR$PartialWidths`
- Expressions **calculated and included automatically** when exporting model via event generator interfaces
- FeynRules includes functions to retrieve decay information

```
myVertices = FeynmanRules[ ℒ ];
ComputeWidths[ myVertices ];
```

```
FR$PartialWidths = {{{ϕ1, ϕ2, ϕ3}, Γ}, ...}
```




$$\Gamma_{1\to 2} = \frac{1}{2mS} \int d\Phi_2 \, |\mathcal{M}_{1\to 2}|^2$$

```
TotWidth[ϕ1];
PartialWidth[{ϕ1, ϕ2, ϕ3}];
BranchingRatio[{ϕ1, ϕ2, ϕ3}];
```

```
PartialWidth[{Z,b,bbar}]
```

$$(e^2 \sqrt{MZ^2\,(MZ^2 - 4\,MB^2)}\;\; (-(42\,MB^2\,c_w^2\,s_w^2) - 9\,MB^2\,c_w^4 + 6\,MZ^2\,c_w^2\,s_w^2 + 9\,MZ^2\,c_w^4 - 17\,MB^2\,s_w^4 + 5\,MZ^2\,s_w^4)) / (288\,\pi\,Abs[MZ]^3\,c_w^2\,s_w^2)$$

# Interfaces

FeynRules has a set of interfaces to export models into formatted input files for a variety of tools. Run:

- UFO: `WriteUFO[`$\mathcal{L}$`,`*`Options`*`]`
- FeynArts/FormCalc: `WriteFeynArtsOutput[`$\mathcal{L}$`,`*`Options`*`]`

. . . to write out coupling and model parameter information in the appropriate format.

Also generator specific legacy interfaces *(no longer maintained)*:

- CalcHep/CompHep: `WriteCHOOutput[`$\mathcal{L}$`,`*`Options`*`]`

Detailed interface options can be found in the manual.

# Contents

# The UFO Format (I)

The UNIVERSAL FEYNRULES OUTPUT (UFO) format is a general Python module containing the **full model information** that can be read by other codes.

- Currently supported by: GOSAM, HERWIG++, MADGRAPH5_AMC@NLO, SHERPA and WHIZARD
- UFOs can also be created by LANHEP and SARAH

To export a model to the UFO format:

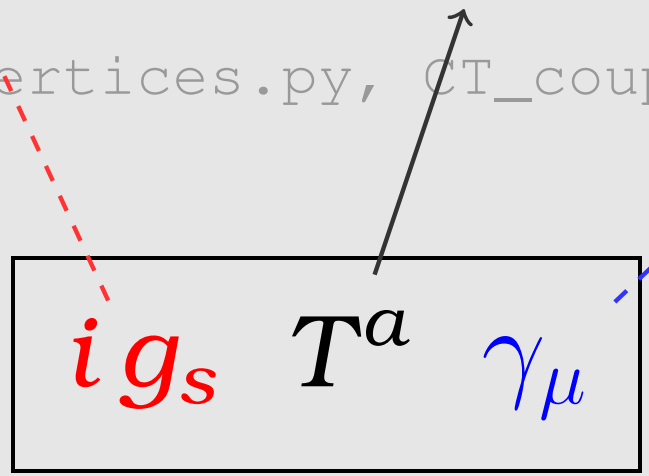
- An `InteractionOrder` must be specified for each coupling
- Fields must belong to the **1**, **3**, **6** or **8** representations of colour

Format detailed at: `https://arxiv.org/abs/1108.2040`

# The UFO Format (II)

`WriteUFO[`$\mathcal{L}$`]` computes the Feynman rules, mass matrices, decay widths and produces a <ModelName>_UFO directory, containing:

- **Particle content and parameters**: `coupling_orders.py`, `parameters.py`, `particles.py`, `decays.py`...
- **Vertices**: `couplings.py`, `vertices.py`, `lorentz.py`
- **Counterterms**: `CT_vertices.py`, `CT_couplings.py` **(NLO)**



# The UFO Format (III)

Vertices written in basis of colour $\otimes$ spin tensors, $C_I$ and $L_J$, as:

$$\mathcal{V}^{a_1 \ldots a_n, \ell_1 \ldots \ell_n}(p_1, \ldots, p_n) = \sum_{I,J} C_I^{a_1 \ldots a_n} \, G_{IJ} \, L_J^{\ell_1 \ldots \ell_n}(p_1, \ldots, p_n) \, ,$$

where $G_{IJ}$ is a matrix of the associated coupling constants.

- vertices.py:

```
V_135 = Vertex(name = 'V_135',
particles = [ P.u__tilde__, P.u, P.g ],
color = [ 'T(3,2,1)' ],
lorentz = [ L.FFV1 ],
couplings = (0,0):C.GC_11)
```

- lorentz.py:

```
FFV1 = Lorentz(name = 'FFV1',
spins = [ 2, 2, 3 ],
structure = 'Gamma(3,2,1)')
```

- couplings.py:

```
GC_11 = Coupling(name = 'GC_11',
value = 'complex(0,1)*G',
order = 'QCD':1)
```

# Contents

# Ingredients for NLO

General one-loop amplitude = linear combination of known scalar integrals with model dependent coefficients.

$$\mathcal{A}^{\text{1-loop}} = \sum_i d_i \,\text{Box}_i + \sum_i c_i \,\text{Triangle}_i + \sum_i b_i \,\text{Bubble}_i + \sum_i a_i \,\text{Tadpole}_i + R$$

- ME generators include virtual corrections by computing coefficients algorithmically
- General one-loop tensor reduction routines implemented in e.g. MadLoop $\Longrightarrow$ automatically obtain $a, b, c, d$ for any model
- UV counterterms + regularization/tensor decomposition dependent (finite) rational terms must be **added by hand**

# Automating One-Loop

Two further model-specific ingredients are required for full automation:

- UV counterterms: $Z \propto \frac{1}{\epsilon} +$ (finite) absorb divergences from loops by redefining the fields and parameters:

$$x_0 \to x + \delta x, \qquad \phi_0 \to (1 + \tfrac{1}{2}\delta Z_{\phi\phi})\phi + \textstyle\sum_\chi \tfrac{1}{2}\delta Z_{\phi\chi}\chi$$

- $R_2$ terms: finite, model-specific integrals due to the $d-4$ component of the numerators $\bar{N}(\bar{q}) = N(q) + \widetilde{N}(\widetilde{q}, q, \epsilon)$:

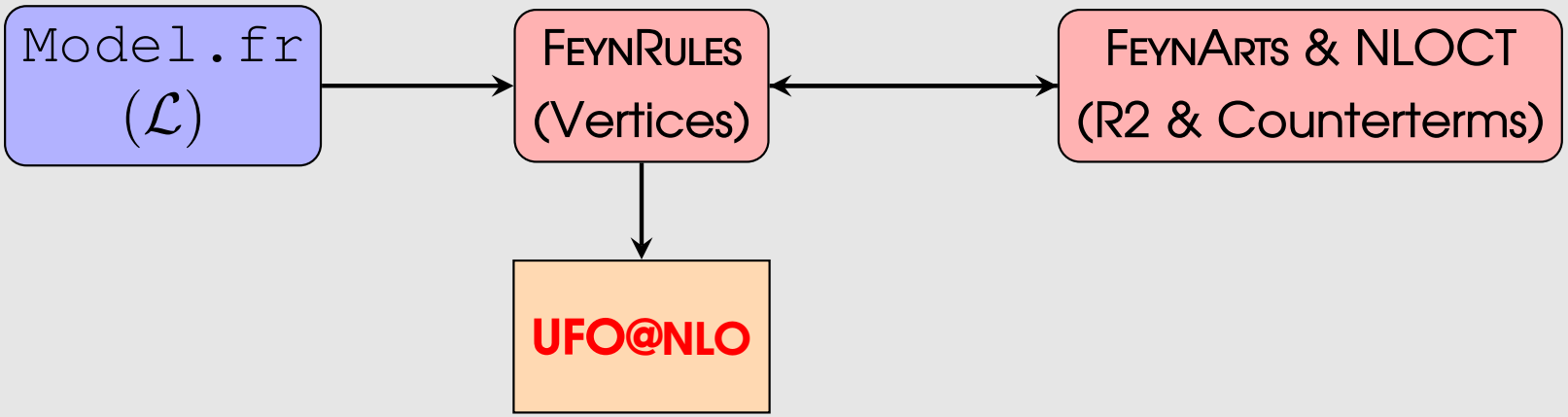
$$R_2 \equiv \lim_{\epsilon\to 0} \frac{1}{(2\pi)^4} \int d^d\bar{q} \frac{\tilde{N}(\tilde{q}, q, \epsilon)}{\bar{D}_0\bar{D}_1\ldots\bar{D}_{m-1}}$$

Both ingredients are implemented as modified tree-level vertices, and can be included in UFO!

# Overview of NLOCT

NLOCT **(Degrande 1406.3030)** creates NLO QCD models in FeynRules:

- FeynRules $\rightarrow$ FeynArts interface implements model into FeynArts
- FeynArts writes the relevant one-loop amplitudes
- NLOCT computes their $R_2$ and UV parts
- The UV and $R_2$ terms are reimported into FeynRules and are output along with tree-level vertices in the UFO format



# Running NLOCT

- Step 1: Introduce renormalization constants for all fields and free parameters, determine those for dependent parameters
- Step 2: Output the model to FeynArts
- Step 3: Run FeynArts and NLOCT to compute the counterterms
- Step 4: Reimport counterterms to FeynRules, export together with tree-level vertices in UFO format

```
(* In a notebook with FeynRules
        + model loaded          *)

Lren = OnShellRenormalization[L];


WriteFeynArtsOutput[ Lren ];


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


Get[ "model.nlo" ]

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

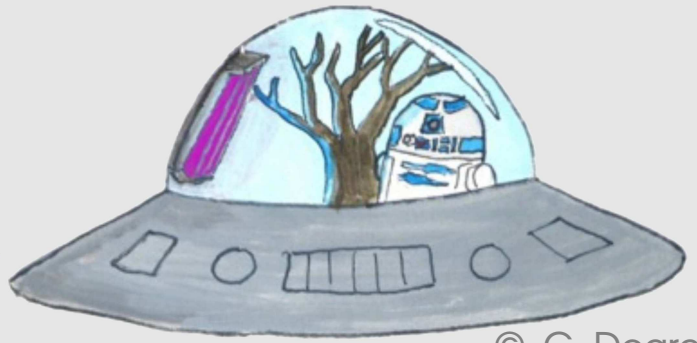
# NLOCT Limitations

Currently, NLOCT requires:

- $\mathcal{L}$ comprised of renormalizable operators: $\dim(\mathcal{O}) \leq 4$
- Feynman Gauge, $\{\gamma_\mu, \gamma_5\} = 0$, 't Hooft-Veltman scheme
- On-shell/complex mass scheme for mass and wavefunction renormalization
- $\overline{\text{MS}}$ otherwise (zero-momentum possible for $\bar{\psi}\psi V$ interactions)

NLO UFO implementations have been verified against analytic and built-in event generator results for the SM, MSSM and 2HDM. . .

- Verified NLO models available on FeynRules model repository
- UFO@NLO supported by MG5_aMC@NLO & GoSam



© C. Degrande

# Contents

# Operator Redundancies in EFT

EFTs generally contain nontrivial operator relationships - $\mathcal{O}_i = k_{ij}\mathcal{O}_j$:

e.g. : $(\bar{u}\gamma^\mu T^A u)(\bar{t}\gamma_\mu T^A t) = \frac{1}{2}(\bar{u}\gamma^\mu t)(\bar{t}\gamma_\mu u) - \frac{1}{6}(\bar{u}\gamma^\mu u)(\bar{t}\gamma_\mu t)$

must eliminate redundancies by **defining an operator basis**.

AllYourBases - implement Lagrangian level operations in FeynRules for:

- EoMs: $\left(D^\rho G_{\rho\mu}\right)^A \to g_s \sum \left(\bar{q}\gamma_\mu T^A q\right) \, , \ldots$
- Fierz identities: $M^I_{ij} M^I_{kj} \to \sum c_J M^J_{il} M^J_{kj} \, , M^J \in \{\Gamma^A, T, \tau, \delta \ldots\}$
- Integration-by-Parts: $\mathcal{A}^\mu (D_\mu \mathcal{B}) \to -(D_\mu \mathcal{A})\mathcal{B}^\mu + \boxed{T}$
- Gamma matrix algebra: $\eta_{\mu\nu}\gamma_\rho \to \gamma_\nu \eta_{\mu\rho} + i\gamma_\mu \sigma_{\nu\rho} + i\epsilon^{\sigma}_{\nu\rho\mu}\gamma_\sigma\gamma_5$
- Bianchi identities: $(D_\mu X_{\nu\rho})^A + (D_\rho X_{\mu\nu})^A + (D_\nu X_{\rho\mu})^A = 0 \ldots$

# Automatic Operator Basis Decomposition

. . . and a reduction algorithm to apply these automatically based on the 'Warsaw' procedure **(Grzadkowski et al.1008.4884)**.

Perform explicit decomposition onto dimension-six basis:

$$\mathcal{L}_{\text{eff}} \supset C_j \mathcal{O}_j^{\text{red}}, \quad \mathcal{O}_j^{\text{red}} = \sum_{i=1}^{59} k_{ij} \mathcal{O}_i^{\text{Warsaw}}$$

For example:

- $\mathcal{O} = (\varphi^\dagger \varphi)(D_\mu \varphi^\dagger D^\mu \varphi)$
- IBP and EoM relations equate this to a linear combination of operators with different field contents
- $\boxed{\varphi^4 D^2} \to \boxed{\varphi^4 D^2} + \boxed{\varphi^3 \psi^2} + \boxed{\varphi^6} + \mu^2 \boxed{\varphi^4} + \boxed{T} + \boxed{E}$

In FeynRules:

```
O = (Phibar[i]Phi[i])(DC[Phibar[j],mu] DC[Phi[j],mu])
```

# Decompose. . .

```
DC[Phi[j], μ] DC[Phi†[l], μ] IndexDelta[j, i] IndexDelta[l, k] Phi[k] Phi†[i] +
 del[del[Phi[l] Phi†[k], mu], mu] IndexDelta[j, i] IndexDelta[l, k] Phi†[i] Phi†[j] +
 lR̄[s1, p].LL[s2, j, r] yl[p, r]† IndexDelta[s1, s2] IndexDelta[j, i] IndexDelta[l, k]
  Phi[l] Phi†[i] Phi†[k] + dR̄[s1, p, a].QL[s2, j, r, b] yd[p, r]† IndexDelta[a, b]
  IndexDelta[s1, s2] IndexDelta[j, i] IndexDelta[l, k] Phi[l] Phi†[i] Phi†[k] -
 μ^2 IndexDelta[j, i] IndexDelta[l, k] Phi[j] Phi[l] Phi†[i] Phi†[k] -
 uR̄[s1, r, a].QL[s2, m, p, b] yu[p, r]† Eps[j, m] IndexDelta[a, b]
  IndexDelta[s1, s2] IndexDelta[i, j] IndexDelta[k, l] Phi[i] Phi[k] Phi†[l] -
 μ^2 IndexDelta[i, j] IndexDelta[k, l] Phi[i] Phi[k] Phi†[j] Phi†[l] + lam IndexDelta[j, i]
  IndexDelta[l, n] IndexDelta[m, k] Phi[j] Phi[l] Phi[m] Phi†[i] Phi†[k] Phi†[n] +
 lam IndexDelta[i, j] IndexDelta[k, m] IndexDelta[l, n] Phi[i] Phi[k] Phi[l]
  Phi†[j] Phi†[m] Phi†[n] + QL̄[s1, j, r, a].dR[s2, p, b] IndexDelta[a, b]
  IndexDelta[s1, s2] IndexDelta[i, j] IndexDelta[k, l] Phi[i] Phi[k] Phi†[l] yd[p, r] +
 LL̄[s1, j, r].lR[s2, p] IndexDelta[s1, s2] IndexDelta[i, j] IndexDelta[k, l] Phi[i]
  Phi[k] Phi†[l] yl[p, r] - QL̄[s1, m, p, a].uR[s2, r, b] Eps[j, m] IndexDelta[a, b]
  IndexDelta[s1, s2] IndexDelta[j, i] IndexDelta[j, k] Phi[l] Phi†[i] Phi†[k] yu[p, r]
```

# AllYourBases (Soon)

AllYourBases automates laborious procedure prevalent in EFT calculations. Currently:

- All operator-level identities implemented in symbolic framework
- Automatic reduction for pure fermionic/bosonic operators implemented
- Fermionic/bosonic mixed operators almost complete . . .

Future applications:

- Quickly translate UV-matching onto $\mathcal{L}_{\text{eff}}$ to common language
- Utilize w/ NLOCT to extract EFT RGEs, verify SMEFT calculation
- Facilitate comparison of limits set using different bases
- Algorithm mostly independent from field content: generalize beyond SM, D=6. . .

# Contents

# Summary and Outlook

Now:

- FeynRules allows implementation of QFT models, quick calculation of properties and Feynman Rules
- Automated LO/NLO phenomenology through generator interfaces, unified UFO standard
- Archive of BSM models available on public repository

Soon:

- Extension of NLOCT to $d > 4$ operators, arbitrary gauges and other renormalization schemes
- Incorporation of RGE running for (B)SM couplings in UFO format
- AllYourBases: Automatic basis decomposition for dimension 6 operators (v1.0)