CERN-PH-TH/2013-239, MCNET-13-14, IPPP/13/71, DCPT/13/142, PITT-PACC-1308

FeynRules 2.0- A complete toolbox for tree-level phenomenology

Adam Alloul [a], Neil D. Christensen [b], Céline Degrande [c,d], Claude Duhr [d], Benjamin Fuks [e,f]

[a] *Groupe de Recherche de Physique des Hautes Énergies (GRPHE), Université de Haute-Alsace, IUT Colmar, 34 rue du Grillenbreit BP 50568, 68008 Colmar Cedex, France, E-mail: adam.alloul@iphc.cnrs.fr*

[b] *PITTsburgh Particle physics, Astrophysics and Cosmology Center (PITT PACC), University of Pittsburgh, Pittsburgh, PA 15260 USA, E-mail : neilc@pitt.edu*

[c] *Department of Physics, University of Illinois at Urbana-Champaign, 1110 West green Street, Urbana, IL 61801, United States, E-mail : cdegrand@illinois.edu*

[d] *Institute for Particle Physics Phenomenology, University of Durham, Durham, DH1 3LE, United Kingdom, E-mail: duhrc@itp.phys.ethz.ch*

[e] *Theory Division, Physics Department, CERN, CH-1211 Geneva 23, Switzerland*

[f] *Institut Pluridisciplinaire Hubert Curien/Département Recherches Subatomiques, Université de Strasbourg/CNRS-IN2P3, 23 Rue du Loess, F-67037 Strasbourg, France, E-mail : benjamin.fuks@iphc.cnrs.fr*

Abstract

FeynRules is a Mathematica-based package which addresses the implementation of particle physics models, which are given in the form of a list of fields, parameters and a Lagrangian, into high-energy physics tools. It calculates the underlying Feynman rules and outputs them to a form appropriate for various programs such as CalcHep, FeynArts, MadGraph, Sherpa and Whizard. Since the original version, many new features have been added: support for two-component fermions, spin-3/2 and spin-2 fields, superspace notation and calculations, automatic mass diagonalization, completely general FeynArts output, a new universal FeynRules output interface, a new Whizard interface, automatic $1 \to 2$ decay width calculation, improved speed and efficiency, new guidelines for validation and a new web-based validation package. With this feature set, FeynRules enables models to go from theory to simulation and comparison with experiment quickly, efficiently and accurately.

Key words: Model building, Feynman rules, Monte Carlo programs.

PROGRAM SUMMARY

Manuscript Title: FeynRules 2.0- A complete toolbox for tree-level phenomenology

Authors: Adam Alloul, Neil. D. Christensen, Céline Degrande, Claude Duhr, Benjamin Fuks.

Program Title: FeynRules 2.0

Journal Reference:

Catalogue identifier:

Licensing provisions: None.

Programming language: Mathematica.

Computer: Platforms on which Mathematica is available.

Operating system: Operating systems on which Mathematica is available.

Keywords: Model building, Feynman rules, Monte Carlo simulations.

Classification: 11.1 General, High Energy Physics and Computing.
11.6 Phenomenological and Empirical Models and Theories.

External routines/libraries: None.

Nature of problem: The program computes the Feynman rules of any quantum field theory, expressed in four-dimensional spacetime, directly from the Lagrangian of the model. Various interfaces to Feynman diagram calculators are included that allow to export the interaction vertices in a format readable by different Monte Carlo event generators or symbolic calculation tools.

Solution method: FeynRules works in three steps:

(1) If necessary, the model Lagrangian is written in terms of four-component fermions and the usual fields of particle physics, instead of Weyl fermions or superfields.

(2) Derivation of the Feynman rules directly form the Lagrangian using canonical commutation relations among fields and creation operators.

(3) Implementation of the new physics model into FeynArts as well as into various Monte Carlo programs via dedicated interfaces.

Restrictions: Mathematica version 7.0 or higher. The Lagrangian must fulfill basic quantum field theory requirements, such as locality and Lorentz and gauge invariance. Fields with spin 0, 1/2, 1, 3/2 and 2 are supported.

Unusual features: Translation interfaces to various Feynman diagram generators exist. Superfields are also supported and can be expanded in terms of their component fields, which allows to perform various sets of superspace computations.

Running time: The computation of the Feynman rules from a Lagrangian varies with the complexity of the model, and runs from a few seconds to several minutes. See Section 7 of the present manuscript for more information.

Contents

1 Introduction

The current era of theoretical particle physics is on the cusp of making several important discoveries. Although the Standard Model (SM) has been amazingly successful at describing and predicting the outcome of most experiments, there are some significant experiments and observations that can not be explained by the SM. As a result, new as-yet unknown physics beyond the SM is required and is expected to be discovered in the coming decades at current and future experiments. Among these puzzles is the instability of the Higgs boson generated vacuum to quantum corrections, the nature of dark matter, neutrino mass, the baryon/antibaryon asymmetry and the hierarchy of fermion masses.

The common feature among these puzzles is that each requires new theoretical models to be built and implemented into simulation software to be compared with experiment and observation. There are several powerful packages that simulate the effects of new models at particle colliders. Each of these has its strengths and may be appropriate for particular kinds of calculations. Among the multi-purpose matrix element generators are CalcHep [1–4], FeynArts/FormCalc [5–9], Helac [10,11], MadGraph [12–16], Sherpa [17,18] and Whizard [19,20]. The parton-level collision is often followed by the important step of radiation and hadronization by packages such as Herwig [21–24], Pythia [25–27] and Sherpa. However, the step of implementing a new theoretical model into the simulation software has been difficult in the past. The chief problems have been that:

1. Each event generator uses its own syntax for a new model. Learning one syntax did not transfer to another generator;
2. Implementing a new model often required the modification of the event generator code itself. These implementations did not transfer well between theorists or to experimentalists;
3. These implementations required the hand coding of each vertex one-by-one. Doing this for the hundreds or thousands of vertices in a model was tedious and very error prone.

LanHep [28–32], FeynRules [33] and Sarah [34,35] were each created to overcome these challenges. In this paper we present version 2.0 of the FeynRules package. Even at the stage of the first version, FeynRules had a significant feature set, allowing theorists to quickly implement their new models into simulation packages. It allowed the theorist to implement their model once in a single unified syntax, which was Mathematica based and closely related to the syntax of FeynArts. It also allowed the user to enter the Lagrangian for their model rather than the individual vertices. It then did the work of calculating the vertices from the Lagrangian, thus saving theorists time and errors. It came with dedicated export interfaces which wrote

the model to file in a form appropriate for the event generators CalcHep, FeynArts, MadGraph and Sherpa. The user was not expected to know the syntax of any of these event generators. Furthermore, for any supported vertex, the final product could be used in the standard version of these event generators and did not require modifications of their code. Thus, these implementations could be passed between theorists and experimentalists with great success. This early version was thoroughly tested both between different simulation tools, between different gauges and with independent implementations of a standard set of models [36]. A complete chain was established that went from theoretical model to parton-level simulation package to hadronization, at which point comparison with experiment could be done [37]. Since then, many new important features have been added. We will summarize the major improvements now.

The core was updated to support two-component Weyl fermion notation [38]. The Lagrangian can now be written in terms of two-component fermions, both left- and right-handed, which may greatly facilitate the implementation of complicated models. Matrix element generators, however, in general work at the level of four-component fermions, and FeynRules has the capability to convert the Lagrangian to four-component fermions in an automated way. See Section 2.4 and 3.2 for further details.

Support for spin-3/2 was added to the core as well as to the CalcHep and UFO interfaces [39, 40]. The resulting output was tested in the context of three models. The first contained a spin-3/2 Majorana gravitino, the second contained a spin-3/2 Dirac top-quark excitation, and the third contained a general effective operator approach for a spin-3/2 Dirac quark excitation. The total cross sections, high energy growth cancellations and angular distributions were all tested with theory and agreement was found. See Section 2.4 for further details.

A superspace module was added [41] allowing the user to implement a supersymmetric (SUSY) model using the superfields directly. The user has the possibility to implement the superspace action, which can then be automatically expanded into component fields. In addition, the equations of motion for the auxiliary fields can be solved by FeynRules. The superspace module was tested on various non-trivial supersymmetric models, including the Minimal Supersymmetric Standard Model (MSSM) and the left-right symmetric supersymmetric model [42]. See Sections 2.5, 3.2, 4.5 and 4.6 for further details.

An automatic mass diagonalization module was created [40,43] allowing FeynRules to extract automatically from any Lagrangian the tree-level mass matrices and to export them through the ASperGe interface for a numerical extraction of the spectrum. In practice, one re-organizes slightly the FeynRules model file and then uses the new built-in functions to extract the mass

matrices and generate the ASperGe source code. The latter is an ensemble of C++ routines able to diagonalize the mass matrices and store the results in a SUSY Les-Houches Accord (SLHA)-like file. See Sections 2.8, 4.7 and 6.2 for further details.

The FeynArts interface was greatly extended to allow the implementation of operators that give rise to Lorentz structures that are not SM or MSSM-like, as is generally the case for higher-dimensional operators. In particular, the new version of the interface creates, on the fly, the generic model file required by FeynArts to be able to use vertices with non-standard Lorentz structures. The new interface was tested on dimension-six operators for top-quark production and decay and for anomalous electroweak gauge boson couplings. See Section 6.4 for further details.

The universal FeynRules output (UFO) interface was created [44]. The idea of this interface is to create a model format that is independent of any individual Feynman diagram calculator and which contains the full information contained in the FeynRules model. It is currently supported by MadGraph 5, MadAnalysis [45] and GoSam [46], and will be used in the future by Herwig++ [23]. See Section 6.7 for further details.

A new export interface to Whizard was written [47] that allows the output of models in the native format of Whizard. One highlight of this interface is that it not only supports unitary and Feynman gauge, but will automatically generate R_ξ gauge model files from a model implemented in Feynman gauge. The resulting output was tested between gauges and with CalcHep and MadGraph for the SM, MSSM and 3-Site model. In conjunction with these validations, the first phenomenological study with this interface was performed [48]. See Section 6.8 for further details.

A module for automatically computing the $1 \to 2$ widths was added [40, 49]. FeynRules computes the squared matrix elements for all possible $1 \to 2$ decays present in the model and multiplies by the appropriate phase-space factor. In addition, the analytic formulas can be included into the UFO output and used by the matrix element generators to obtain the tree-level two-body widths and branching ratios for all the particles defined in the model. See Section 4.8 for further details.

The speed and efficiency of FeynRules was greatly improved. Parts of the core were rewritten to make better use of more efficient Mathematica functions. Parallelization was also added to some of the most labor-intensive parts of the code. See Section 7 for further details.

New guidelines were established to improve the quality of FeynRules based model implementations [38]. Additionally, a new web-based validation platform was created [50] to help accomplish this task. This validation platform

allows any user to upload their model implementation. It then generates a list of $2 \to 2$ processes which it compares between matrix element generators and between gauges. It also has the ability to compare with existing, independently implemented versions of the model, if available. The user is not required to know the details of the different matrix element generators. Processes with large discrepancies between gauges or between matrix element generators are flagged for the user to look into. See Section 8 for further details.

The purpose of this manual is to give an up-to-date, integrated and complete instruction set for using FeynRules and all its new features. The organization is as follows. Sections 2 and 3 describe the implementation of the model and the Lagrangian. Section 4 describes how to run FeynRules in a Mathematica session. Section 5 gives a simple example of a model implementation. Section 6 describes the use of the export interfaces to generate code that can be run by one of the Feynman diagram calculators. Section 7 discusses the speed and efficiency improvements. Section 8 discusses the web based validation package and its use to improve the accuracy of the models built with FeynRules. In Section 9, we conclude.

2 The Model Description

To use FeynRules, the user must start by entering the details of the new model in a form that can be parsed by FeynRules. First of all, this means that, since FeynRules is a Mathematica package, the model must be written in a valid Mathematica syntax. Secondly, FeynRules specifies a set of special variables and macros for the user to enter each aspect of a new model. In this section, we will describe each of these in turn. These definitions can be placed in a pure text file or in a Mathematica notebook. The latter can be helpful during the development of the model details, however, we suggest storing the final version of any model in a pure text file.

2.1 Model Information

The user has the possibility to include general information about the model implementation into the model file via the variables `M$ModelName` and `M$Information`. While the first of these variables is a string giving the name of the model, the variable `M$Information` acts as an electronic signature of the model file that allows to store, *e.g.*, the names and addresses of the authors of the model file, references to the papers used for the implementation, the version number of the model file, *etc.* This information is stored as a Mathematica replacement list as shown in the following example

```
M$ModelName = "my_new_model";

M$Information = {
          Authors      -> {"Mr. X", "Ms. Y"},
          Institutions -> {"UC Louvain"},
          Emails       -> {"X@uclouvain.be", "Y@uclouvain.be},
          Date         -> "01.03.2013",
          References   -> {"reference 1", "reference 2"},
          URLs         -> {"http://feynrules.irmp.ucl.ac.be"},
          Version      -> "1.0"
                };
```

A summary and complete set of options available for `M$Information` can be found in Table 1.

The model information will be printed on the screen whenever the model is loaded into MATHEMATICA. In addition, the contents of `M$Information` can be retrieved by issuing the command `ModelInformation[]` in a MATHEMATICA session, after the model has been loaded.

2.2 Index Definitions

In general the Lagrangian describing a model is a polynomial in the fields (and their derivatives) as well as in the parameters of the model. Very often, these quantities carry indices specifying their members and/or how the different quantities transform under symmetry operations. For example, the gauge field G^a_μ of an unbroken gauge group $SU(N)$ carries two different types of indices:

- a Lorentz index μ ranging from 0 to 3;
- an adjoint gauge index a ranging from 1 to $N^2 - 1$.

It is therefore crucial to define at the beginning of each model file the types of indices that appear in the model, together with the range of values each type of index may take.

A field $\psi_{i_1 i_2 \ldots}(x)$ carrying indices i_1, i_2, ... is represented inside FEYNRULES by an expression of the form `psi[`$index_1$, $index_2$, ...`]`. Each $index_i$ denotes an object of the form `Index[`name, i`]`, and represents an index of type name taking the value i. In this expression name is a symbol and value can be both a symbol or an integer. In general the name can be chosen freely by the user, but we emphasize that there are predefined names for the index types describing four-vectors (`Lorentz`), four-component spinors (`Spin`) and two-component left and right-handed Weyl spinors (`Spin1` and `Spin2`).

Table 1: Model Information

`M$ModelName`	A string, specifying the name of the model. The default value is the name of the model file.
`M$Information`	A replacement list, acting as an electronic signature of the model implementation.
`ModelInformation[]`	Prints the contents of `M$Information`.

Options for `M$Information`

`Authors`	A list of strings, specifying the authors of the model implementation.
`Institutions`	A list of strings, specifying the institutions of the authors.
`Emails`	A list of strings, specifying the email addresses of the authors. The order of the email addresses is the same as the order in which the names of the authors have been specified.
`Date`	A string specifying the date.
`References`	A list of strings, containing the references that the authors would like to be cited whenever the model implementation is used.
`URLs`	A list of strings, containing Internet addresses where more information about the model can be found.
`Version`	A version number for the model. Each time the model is improved, this version number should be increased and a note written in the model file describing the change.

For FeynRules to run properly, the different types of indices that appear in the model have to be declared at the beginning of the model file, together with the range of values they can take. This is achieved like in the following examples

```
IndexRange[ Index[Colour] ] = Range[3];
IndexRange[ Index[SU2W] ]  = Unfold[ Range[3] ];
IndexRange[ Index[Gluon] ]  = NoUnfold[ Range[8] ];
```

These commands declare three types of indices named `Colour`, `SU2W` and `Gluon` ranging form 1 to 3 and 1 to 8 respectively. The function `Range` is an internal Mathematica command taking an integer n as input and returning the range $\{1, \ldots, n\}$. Moreover, the indices of type `Lorentz`, `Spin`, `Spin1` and `Spin2` are defined internally and do not need to be defined by the user.

At this stage we have to comment on the functions `Unfold` and `NoUnfold` used in the declaration of the indices of type `SU2W` and `Gluon`:

(1) The `Unfold` command instructs FeynRules that if an index of this type appears contracted inside a monomial, then it should be expanded, *i.e.*, the monomial with the contracted pair of indices should be replaced by the explicit sum over the indices. Any index that expands in terms of non-physical states must be wrapped in `Unfold`. For instance, the $SU(2)_L$ indices in the Standard Model or in the Minimal Supersymmetric Standard Model must always be expanded in order to get the Feynman rules in terms of the physical states of the theory. Otherwise, wrong results could be obtained when employing matrix element generators. We refer to Section 4 for more details.

(2) The `NoUnfold` is ignored by FeynRules. It however plays a role in FeynArts, and we refer to Section 6.4 or to the FeynArts manual [6] for more details.

While indices are represented internally inside FeynRules by expressions of the form `Index[`name, `i]`, the user does not need to enter indices in this form. Since it is always possible to reconstruct the type of an index from its position inside the expression `psi[`$index_1$, $index_2$, ...`]`. For example, the gluon field `G[mu, a]` has been declared as carrying two indices, the first one being of type `Lorentz` and the second one of type `Gluon` (see Section 2.4). FeynRules can then employ particle class properties to restore the correct notation internally, as in

`G[mu, a]` $\longrightarrow$ `G[Index[Lorentz, mu], Index[Gluon, a]]` .

In addition, it is possible to specify how the different types of indices should be printed on the screen. This is done via the `IndexStyle` command, *e.g.*,

```
IndexStyle[ Colour, i ];
IndexStyle[ Gluon, a ];
```

Issuing these commands at the beginning of a model file instructs FeynRules to print indices of type `Colour` and `Gluon` with symbols starting with the letters `i` and `a`, respectively, followed by an integer number.

A summary of information for the index can be found in Table 2.

2.3 The model parameters

All the model parameters (coupling constants, mixing angles and matrices, masses, *etc.*) are implemented as elements of the list `M$Parameters`,

```
M$Parameters = {
  param1 == { options1 },
  param2 == { options2 },
          ...
};
```

Each component of this list consists of an equality whose left-hand side is a label and the right-hand side is a list of Mathematica replacement rules. The labels (`param1` and `param2` in the example) are user-defined names to be used when building the Lagrangian. The sets of replacement rules (`options1` and `options2` in the example) contain optional information allowing to define each parameter together with its properties. The model parameters are split into two categories according to whether they carry indices or not. We start by reviewing in Section 2.3.1 the implementation of scalar parameters, *i.e.*, parameters that do not carry any index. Tensorial parameters, *i.e.*, parameters carrying one or several indices, are then discussed in Section 2.3.2.

2.3.1 Scalar parameters

To illustrate the implementation of scalar parameters, we focus on the example of the strong coupling constant. The declaration of any other parameter is similar. Although the strong coupling constant g_s usually appears in the Lagrangian, it is in general more convenient to use the quantity $\alpha_s = g_s^2/4\pi$ as an input parameter, since its numerical value (*e.g.*, at the electroweak scale) has been precisely determined from experiments. It is therefore desirable to have both parameters in the FeynRules model file. This motivates us to choose, in our example, α_s as a free parameter of the model, *i.e.*, as an *external* parameter (in FeynRules parlance) or equivalently as an independent parameter. In contrast, g_s is an *internal* parameter, or in other words, a parameter de-

Table 2: Index Information

`Index[name, value]`	Represents an index of type name with a value value.
`IndexRange[Index[name]]`	Declares an index of type name along with its range.
`Range[n]`	Internal MATHEMATICA command, returning the range of integers $\{1, \ldots, n\}$. Allows to declare the range of an index to be from 1 to n.
`IndexStyle[name, label]`	Fixes the `StandardForm` and `TraditionalForm` of an index of type name to be printed as a symbol starting with label.
`NoUnfold`	FEYNARTS command. This is only used by the FEYNARTS interface.
`Unfold`	Indices of this type will always be expanded out in the interfaces. See Section 4.
Predefined types of indices	
`Lorentz`	Name for Lorentz indices, ranging from 1 to 4, and printed as μ.
`Spin`	Name for Dirac indices, ranging from 1 to 4, and printed as s.
`Spin1`	Name for left-handed Weyl indices, ranging from 1 to 2, and printed as α.
`Spin2`	Name for right-handed Weyl indices, ranging from 1 to 2, and printed as $\dot{\alpha}$.

pending on one or several of the other internal and/or external parameters of the model. As a result, α_s (`aS`) and g_s (`gs`) could be implemented as in the example

```
aS == {
  TeX               -> Subscript[\[Alpha],s],
```

```
  ParameterType    -> External,
  InteractionOrder -> {QCD, 2},
  Value            -> 0.1184,
  BlockName        -> SMINPUTS,
  OrderBlock       -> 3,
  Description      -> "Strong coupling constant at the Z pole"
};

gs == {
  TeX              -> Subscript[g,s],
  ParameterType    -> Internal,
  ComplexParameter -> False,
  InteractionOrder -> {QCD, 1},
  Value            -> Sqrt[4 Pi aS],
  ParameterName    -> G,
  Description      -> "Strong coupling constant at the Z pole"
 }
```

The external or internal nature of a parameter can be specified by setting the attribute `ParameterType` to the value `External` or `Internal`. Another difference between external and internal parameters lies in the value taken by the attribute `Value` of the parameter class. For external parameters, it refers to a *real* number [1] while for internal parameters, it contains a formula. This formula is given in standard MATHEMATICA syntax and provides the way an internal parameter is connected to the other model parameters. It is important to note that only previously declared parameters can be employed when implementing the formula. Internal parameters can be either real or complex variables, which is specified by setting the attribute `ComplexParameter` to the value `True` or `False` (default).

Instead of providing the value of a parameter (a real number for an external parameter or a formula for an internal parameter) through the attribute `Value`, the user has the option to use the attribute `Definitions`. This option of the parameter class takes as argument a MATHEMATICA replacement rule. Returning to our example above, we could replace the `Value` attribute of `gs` by

```
Definitions -> { gs -> Sqrt[4 Pi aS] }
```

The difference between these two choices appears at a later stage and concerns the derivation of the interaction vertices by FEYNRULES. A parameter provided with a definition is removed from the vertices and replaced by its definition, whilst otherwise, the associated symbol is kept.

[1] Complex external parameters have to be split into their real and imaginary parts and declared individually.

Table 3: Attributes of the parameter class common for scalar and tensorial parameters.

`ParameterType`	Specifies the nature of a parameter. The allowed choices are `External` or `Internal`. By default, scalar parameters are considered as external and tensorial parameters as internal.
`Value`	Refers to a real number (for external parameters) or to an analytical formula defining the parameter that can be expressed in terms of other parameters (for internal parameters). By default, the `Value` attribute takes the value 1.
`Definitions`	Refers to a list of MATHEMATICA replacement rules that are applied by FEYNRULES before computing the interaction vertices of a model.
`ComplexParameter`	Defines whether a parameter is a real (`False`) or complex (`True`) quantity. By default, scalar parameters are real while tensorial parameters as complex. External parameters must be real.
`BlockName`	This provides information about the name of the Les Houches block containing an external parameter (see Section 6.1.4). By default, the block name is taken as `FRBlock`.
`OrderBlock`	Provides information about the position of an external parameter within a given Les Houches block. By default, this number starts at one as is incremented after the declaration of each parameter. It must be left unspecified for tensorial parameters (see Section 6.1.4).
`TeX`	Teaches FEYNRULES how to write the TEX form of a parameter. By default, it refers to (the string of) the associated MATHEMATICA symbol.
`Description`	Refers to a string containing a description of the physical meaning of the parameter.

Table 4: Additional attributes of the parameter class related to Feynman diagram calculators

`ParameterName`	Specifies what to replace the symbol by before writing out the Feynman diagram calculator model files. By default, it is taken equal to the symbol representing the parameter. See Section 6.1.1 for more details.
`InteractionOrder`	Specifies the order of the parameter according to a specific interaction. It refers to a pair with the interaction name, followed by the order, or a list of such pairs. This option has no default value. See Section 6.1.7 for more details.

The list of all the attributes of the parameter class for scalar quantities is given in Table 3 and Table 4. With the exception of the `TeX` attribute allowing for the TEX-form of a parameter and the `Description` attribute which gives, as a string, the physical meaning of a parameter, the options non-described so far are related to the interfaces to Feynman diagram generators and discussed in greater detail in Section 6.

2.3.2 Tensorial parameters

The second category of parameters are tensorial parameters, *i.e.*, parameters carrying indices. The index structure can be specified through the attribute `Indices` of the parameter class as in the following example,

```
Indices -> {Index[Scalar], Index[Generation]}
```

The parameter under consideration carries two indices, one index of type `Scalar` and one index of type `Generation`. In many cases tensorial parameters correspond to unitary, Hermitian or orthogonal matrices. These properties can be implemented in the FeynRules model description by turning the values of the attributes `Unitary`, `Hermitian` or `Orthogonal` to `True`, the default value being `False`.

In contrast to scalar parameters, the attribute `Value` (`Definitions`) is now a list of values (replacement rules) for all possible numerical value of the indices. Moreover, tensorial parameters are by default complex quantities, *i.e.*, the

default value for the attribute `ComplexParameter` is `True`.

For example, the Cabibbo-Kobayashi-Maskawa (CKM) matrix could be defined as follows,

```
CKM == {
  ParameterType    -> Internal,
  Indices          -> {Index[Generation], Index[Generation]},
  Unitary          -> True,
  ComplexParameter -> True,
  Definitions      -> {
    CKM[i_,3] :> 0 /; i!=3,
    CKM[3,i_] :> 0 /; i!=3,
    CKM[3,3]  -> 1 },
  Value            -> {
    CKM[1,1] ->  Cos[cabi],
    CKM[1,2] ->  Sin[cabi],
    CKM[2,1] -> -Sin[cabi],
    CKM[2,2] -> Cos[cabi] },
  Description      -> "CKM-Matrix"
}
```

In these declarations, `cabi` stands for the Cabibbo angle, assumed to be declared previously in the model file. We have also simultaneously employed the attributes `Value` and `Definitions` so that vanishing elements of the CKM matrix are removed at the time of the extraction of the interaction vertices by FeynRules. In this way, zero vertices are removed from the output.

FeynRules assumes that all indices appearing inside a Lagrangian are contracted, *i.e.*, all indices must come in pairs. However, it may sometimes be convenient to be able to break this rule. For example, in the case of a diagonal Yukawa matrix y, the associated Lagrangian term

$$\mathcal{L}_{\text{yuk}} = H\ \bar{\psi}_f\ y_{ff'}\ \psi_{f'}\ , \tag{2.1}$$

where ψ denotes a generic fermionic field and H a generic scalar field, can be written in a more compact form as

$$\mathcal{L}_{\text{yuk}} = \tilde{y}_f\ H\ \bar{\psi}_f\ \psi_f\ . \tag{2.2}$$

In the second expression, the Yukawa matrix has been replaced by its diagonal form $y_{ff'} = \tilde{y}_f \delta_{ff'}$, rendering the summation convention explicitly violated since the index f appears three times. In order to allow for such violations in FeynRules, the attribute `AllowSummation` must be set to the value `True` when implementing the Yukawa coupling $\tilde{y}$. Consequently, this allows FeynRules to sum the single index carried by the vectorial parameter y along with

Table 5: Attributes of the parameter class associated to tensorial parameters.

`Indices`	Mandatory. Provides, as a list, the indices carried by a parameter.
`Unitary`	Defines a matrix as unitary (`True`) or not (`False`). The default value is `False`.
`Hermitian`	Defines a matrix as Hermitian (`True`) or not (`False`). The default value is `False`.
`Orthogonal`	Defines a matrix as orthogonal (`True`) or not (`False`). The default value is `False`.
`AllowSummation`	See the description in the text. By default this option is set to `False`. Moreover, it is only available for parameters with one single index.

the two other indices fulfilling the summation conventions (in our example, the indices of the fermions). We stress that this option is only available for parameters carrying one single index.

In the case of contracted Weyl spin indices, the upper and lower position of the indices also plays an important role, since

$$\chi^\alpha \chi'_\alpha = -\chi_\alpha \chi'^\alpha \ , \tag{2.3}$$

where χ and χ' are two generic left-handed Weyl fermions. This issue is described in details in Section 4.5.

The complete list of specific attributes related to tensorial parameters can be found in Table 5, while all the attributes of Table 3 and Table 4 can also be employed.

2.4 Particle Classes

Particle classes can be instantiated in a similar way to parameter classes. The main difference is that, following the original FeynArts syntax [6], the particle classes are labeled according to the spins of the particles rather than

to their symbol,

```
M$ClassesDescription = {
   spin1[1] == { options1 },
   spin1[2] == { options2 },
   spin2[1] == { options3 },
   ...}
```

The symbols `spin1`, `spin2`, *etc.*, refer each to one of the field type supported by FeynRules [2]:

- `S`: scalar fields;
- `F`: Dirac and Majorana spinor fields;
- `W`: Weyl fermions (both left- and right-handed);
- `V`: vector fields;
- `R`: four-component Rarita-Schwinger fields (spin-3/2 fields);
- `RW`: two-component Rarita-Schwinger fields (both left- and right-handed spin-3/2 fields);
- `T`: spin-2 fields;
- `U`: ghost fields (only complex ghosts are supported).

Similar to the declaration of the parameter classes, the quantities `options1`, `options2`, `options3`, *etc.*, are sets of replacement rules defining field properties. Following the spirit of the original FeynArts model file format, each particle class should be thought of as a 'multiplet' consisting of particles that carry the same quantum numbers but might differ in mass. This implies that all fields belonging to the same class necessarily carry the same indices. The main advantage of collecting particles with the same indices into classes is that it allows the user to write compact expressions for Lagrangians. This is illustrated in the example Lagrangian

$$\mathcal{L} = \bar{q}_f i \partial\!\!\!/ q_f + g_s \bar{q}_f \gamma^\mu T_a q_f G^a_\mu \,, \tag{2.4}$$

where q_f denotes the "quark class", g_s the strong coupling constant, T_a the fundamental representation matrices of $SU(3)$ and G_μ stands for the gluon field. The notation of Eq. (2.4) avoids having to write out explicitly a Lagrangian term for each quark flavor.

Just like for the parameter classes, each particle class can be given a number of options which specify the properties of the field. In particular, there are two mandatory options that have to be defined for every field:

(1) Each particle class must be given a name, specified by the `ClassName`

[2] The classes `R`, `W` and `RW` are specific to FeynRules and not supported by FeynArts.

option, which is at the same time the symbol by which the particle class is denoted in the Lagrangian.

(2) The option `SelfConjugate` specifies whether the particle has an antiparticle or not. The possible values for this option are `True` or `False`.

In addition, the particle class comes with various other options, as shown in Table 6. The set of all options can be divided into two groups:

- options which are directly used by FeynRules in the Mathematica session when computing the Feynman rules. Examples include the indices carried by a field, its mass, *etc.*;
- options which are mostly ignored by FeynRules, but describe information required by some Feynman diagram calculators. These options will be passed on to the Feynman diagram calculators via the corresponding FeynRules interfaces (see Section 6).

In the following we only describe the options directly used by FeynRules itself. The interface specific options are discussed in Section 6.

Besides the name of the particle class, the user may also provide names for the individual members of the class. For example, if `uq` denotes the class of all up-type quarks with members {`u,c,t`}, then this class and its members can be defined in the model file as

```
ClassName    -> uq,
ClassMembers -> {u, c, t}
```

If a field is not self-conjugate, an instance of the particle class associated with the antiparticle is created automatically by appending '`bar`' to the name of the particle. For example, after the model file has been loaded into FeynRules, the symbols `uqbar`, as well as {`ubar, cbar, tbar`}, are available and can be used to construct a Lagrangian. For bosonic fields, the field representing the antiparticle corresponds to the Hermitian conjugate of the field, while for fermionic fields, the symbol `psibar` (`psi` representing any fermionic field ψ) corresponds to the quantity $\bar{\psi} = \psi^\dagger \gamma^0$.

Fields transform in general as tensors under some symmetry groups, and they usually carry indices indicating the transformation laws. Just like for parameters, the indices carried by a field can be specified via the `Indices` option, *e.g.*,

```
Indices -> {Index[ Colour ]}
Indices -> {Index[ Colour ], Index[ SU2D ]}
```

Indices labeling the transformation laws of the fields under the Poincaré symmetry (spin and/or Lorentz indices) are automatically inferred by FeynRules

and do not need to be specified. In addition to indices labeling how a field transforms under symmetries, each field may have additional indices such as flavor indices. One of these can be distinguished as the *flavor index* of the class and labels its members. It is declared in the model file via the `FlavorIndex` option. For example, the up-type quark class `uq` previously introduced is usually defined carrying two indices supplementing the spin index (automatically handled by FeynRules), one of type `Colour` ranging from 1 to 3 and specifying the color of the quark, and another index of type `Flavour` ranging from 1 to 3. The latter is specified as the flavor index of the class (via the `FlavourIndex` option) so that it labels the members of the class,

```
Indices -> { Index[ Colour ], Index[ Flavour ] },
FlavorIndex -> Flavour
```

Quantum fields are not always only characterized by the tensor indices they carry, but also by their charges under the discrete and / or abelian groups of the model. FeynRules allows the user to define an arbitrary number of $U(1)$ charges carried by a field, as, *e.g.*, in

```
QuantumNumbers -> {Q -> -1, LeptonNumber -> 1}
QuantumNumbers -> {Q -> 2/3}
```

Next, the user can specify the symbol and the numerical value for the masses and the decay widths of the different members of a particle class using the `Mass` and `Width` options [3] . The argument of `Mass` is a list with masses for each of the class members, as in

```
Mass -> {MW}
Mass -> {MU, MC, MT}
Mass -> {Mu, MU, MC, MT}
```

where in the last example, the symbol `Mu` is given for the entire class, while the symbols `MU`, `MC` and `MT` are given to the members. The symbol for the generic mass (`Mu` in this case) is by default a tensorial parameter carrying a single index corresponding to the `FlavorIndex` of the class. In addition, the `AllowSummation` property is internally set to `True`. The user can not only specify the symbols used for the masses but also their numerical value as in

```
Mass -> {MW, Internal}
Mass -> {MZ, 91.188}
Mass -> {{MU,0}, {MC,0}, {MT, 174.3}}
Mass -> {Mu, {MU, 0}, {MC, 0}, {MT, 174.3}}
```

[3] In the following we only discuss the masses of the particles. Widths however work in exactly the same way.

If no numerical value is given, the default value 1 is assumed. In the first example, `MW` is given the value `Internal`, which instructs FeynRules that the mass is defined as an internal parameter in `M$Parameters`. This is the only case in which a user needs to define a mass in the parameter list. All other masses given in the definition of the particles should not be included in this list.

While Lagrangians can often be written in a compact form in the gauge eigenbasis, the user usually wants to obtain Feynman rules in the mass eigenbasis. More generally, if the mass matrix appearing in the Lagrangian is not diagonal, one has to perform a field rotation that diagonalizes the mass matrix, *i.e.*, the user has to replace certain fields by a linear combination of new fields such that in this new basis the mass matrix is diagonal. For fields in the gauge eigenbasis, the `Mass` and `Width` options do not have to be set, but these options are replaced by the option

```
Unphysical -> True
```

The diagonalization of the gauge eigenbasis can be performed automatically by means of the ASperGe package which is interfaced to FeynRules. We refer to Section 4.7 and Section 6.2 for more details. The values of these rotation matrices can also be expressed analytically (if available) or as numerical values, relying on other tools to perform the diagonalization. The rotation to the mass eigenbasis in FeynRules are, in this case, implemented by using the `Definitions` option of the particle classes, which consists of a set of replacement rules that are applied to the Lagrangian before the computation of the Feynman rules. As an example, the relation between the hypercharge gauge boson (`B`) and the photon (`A`) and Z-boson (`Z`) could be included in a model description as

```
Definitions -> {B[mu_] -> -sw Z[mu] + cw A[mu]}
```

where the symbols `sw` and `cw` stand for the sine and cosine of the weak mixing angle and are declared alongside the other model parameters. The replacement is purely symbolic and can be performed at the Mathematica level, even if the numerical values of the mixing parameters are unknown.

We conclude this section by giving some options specific to certain kinds of fields. For ghost particles, there is an option `Ghost` which tells FeynRules the name of the gauge boson the ghost field is connected to. There is a similar option `Goldstone` in the case of scalar fields.

Majorana fermions are by definition eigenstates of the charge conjugation operator, and the associated eigenvalue must be a phase, since charge conjugation

is unitary. If λ denotes a Majorana fermion, then

$$\lambda^c = C\,\bar{\lambda}^T = e^{i\phi}\,\lambda\,. \tag{2.5}$$

The information on the phase ϕ can be provided by the option `MajoranaPhase` of the particle class,

```
MajoranaPhase -> Phi
```

where `Phi` stands for the phase of the Majorana fermion λ. This phase can be further retrieved in the MATHEMATICA session by typing

```
MajoranaPhase[lambda]
```

where the symbol `lambda` represents the Majorana fermion λ. The default value for the Majorana phase is 0.

Weyl fermion classes have an additional attribute that allows to specify their chirality. In the following we only discuss the case of spin-1/2 two-component fermions (`W`), keeping in mind that the same attributes are also available for spin-3/2 two-component fermions (`RW`). For example, the sets of rules

```
W[1] == {
   ClassName     -> chi,
   Chirality     -> Left,
   SelfConjugate -> False
   },
W[2] == {
   ClassName     -> xibar,
   Chirality     -> Right,
   SelfConjugate -> False
   }
```

define two Weyl fermions χ (`chi`) and $\bar{\xi}$ (`xibar`), the first one being left-handed and the second one right-handed. While Weyl fermions are in general not supported by any Feynman diagram calculator, it is often simpler to write down the Lagrangian in terms of two-component fermions and to transform them into four-component spinors in a second step. For this reason, it is possible to add the option `WeylComponents` to an instance of the particle class `F` associated with four-component Dirac and Majorana fermions. This option instructs FEYNRULES how to perform the mapping of the the two-component to the four-component spinors. For example, the Dirac fermion $\psi = (\chi, \bar{\xi})^T$ could be defined in FEYNRULES as

```
F[1] == {
   ClassName      -> psi,
```

```
  SelfConjugate  -> False,
  WeylComponents -> {chi, xibar}
  }
```

where the Weyl fermions `chi` and `xibar` are defined above. In the case of a four-component Majorana fermion only the left handed component needs to be specified. In Section 4, we will see how we can instruct FeynRules to transform a Lagrangian written in terms of two-component spinors to its counterpart expressed in terms of four-component fermions.

2.5 Implementing superfields in FeynRules

Supersymmetric theories can usually be written in a very compact form when using superfields, which are the natural objects to describe fields in supersymmetry. Feynman diagram calculators usually require a model to be implemented in terms of the component fields, *i.e.*, the standard fields of particle physics. FeynRules allows the user to implement the model in terms of superfields, and the code then takes care of deriving internally the component field Lagrangian [41].

Most of the phenomenologically relevant supersymmetric theories can be entirely built in terms of chiral and vector superfields [4]. Superfields are declared in FeynRules in a way similar to ordinary fields,

```
M$Superfields = {
  superfield1[1] == { options1 },
  superfield2[2] == { options2 },
             ...
}
```

The elements in the left hand side specify the type of superfield (similar to the way ordinary particle classes are defined by their spin). Currently, FeynRules supports two types of superfields:

- `CSF`: chiral superfields;
- `VSF`: vector superfields in the Wess-Zumino gauge.

To illustrate the main features of the superfield declaration, we discuss the implementation of a left-handed chiral superfield Φ, a right-handed chiral superfield Ξ and a non-abelian vector superfield $V_{\text{w.z.}}$ in the Wess-Zumino gauge,

```
CSF[1] == {
```

[4] Vector superfields must be constructed in the Wess-Zumino gauge.

Table 6: Particle Class Options

`S`, `F`, `W`, `V`, `R`, `RW`, `T`, `U`	Particle classes.
`ClassName`	Mandatory. This option gives the symbol by which the class is represented.
`SelfConjugate`	Mandatory. Takes the values `True` or `False`.
`Indices`	The list of indices carried by the field. Note that Lorentz indices (`Lorentz`, `Spin`, `Spin1`, `Spin2`) are implicit and not included in this list.
`FlavorIndex`	The name of the index making the link between the generic class symbol and the class members.
`QuantumNumbers`	A replacement rule list, containing the $U(1)$ quantum numbers carried by the class.
`ClassMembers`	A list with all the members of a class. If the class contains only one member, this is by default the `ClassName`.
`Mass`	A list with the masses for the class members. A mass can be entered either as the symbol which is representing it, or by a two-component list, the first element being the associated symbol and the second one its numerical value. A generic symbol with a default numerical value equal to 1 is generated if omitted.
`Width`	A list with the decay rates for the class members. Similar to `Mass`.
`Ghost`	Option for ghost fields specifying the `ClassName` of the gauge boson the ghost is associated to.
`Goldstone`	For a scalar field, tagging it as the Goldstone boson related to the gauge boson which this option is referring to.
`MajoranaPhase`	The Majorana phase of a Majorana fermion. The default is 0.
`Chirality`	Option for Weyl fermions, specifying their chirality, `Left` or `Right`. The default is `Left`.

Particle Class Options (continued)

`WeylComponents`	Option for the Dirac and Majorana field classes, mapping them to their left- and right-handed Weyl components. For Majorana particles, only the left-handed piece needs to be specified.
`Definitions`	A list of replacement rules that should be applied by FeynRules before calculating the interaction vertices.
`ParticleName`	A list of strings, corresponding to the particle names as they should appear in the output files for the Feynman diagram calculation programs. By default, the value is the same as the one of `ClassMembers`. This name must satisfy the constraints of whatever Feynman diagram calculation package the user wishes to use it with. See Section 6.1.1.
`AntiParticleName`	Similar to `ParticleName`.
`TeXParticleName`	A list of strings with the TeX-form of each class member name. The default is the same as `ParticleName`. See Section 6.1.1.
`TeXAntiParticleName`	Similar to `TeXParticleName`.
`Unphysical`	If `True`, declares that the field does not have to be included in the particle list written for another code by a FeynRules interface but replaced instead by the value of the `Definitions` attribute. The default is `False`.
`PDG`	A list of the PDG codes of the particles. An automatically generated PDG code is assigned if this option is omitted. See Section 6.1.2.
`PropagatorLabel`	A list of strings propagators should be labeled with when drawing Feynman diagrams. The default value is the same as the `ParticleName`. See Section 6.1.8.

Particle Class Options (continued)

`PropagatorArrow`	Whether to put an arrow on the propagator (`True`) or not (`False`). `False` by default. See Section 6.1.8.
`PropagatorType`	This specifies how to draw the propagator line for this field. The default value is inferred from the class. The allowed choices are `ScalarDash` (straight dashed line), `Sine` (sinusoidal line), `Straight` (straight solid line), `GhostDash` (dashed line), and `Curly` (curly *gluonic* line). See Section 6.1.8.
`FullName`	A string, specifying the full name of the particle, or a list containing the names for each class member. By default `FullName` is the same as `ParticleName`.
`MixingPartners`	FeynArts option. See Ref. [6].
`InsertOnly`	FeynArts option. See Ref. [6].
`MatrixTraceFactor`	FeynArts option. See Ref. [6].

```
  ClassName -> PHI,
  Chirality -> Left,
  Weyl      -> psi,
  Scalar    -> z,
  Auxiliary -> FF
}
CSF[2] == {
  ClassName -> XI,
  Chirality -> Right,
  Weyl      -> psibar,
  Scalar    -> zbar
}
VSF[1] == {
  ClassName  -> VWZ,
  GaugeBoson -> V,
  Gaugino    -> lambda,
  Indices    -> {Index[SU2W]}
  }
```

This declares two chiral superfields labeled by the tags `CSF[1]` and `CSF[2]` and one vector superfield labeled by the tag `VSF[1]`. For all three superfields, the symbols used when constructing a Lagrangian or performing computations in superspace are specified through the attribute `ClassName`. Moreover, as for Weyl fermions, the chirality of the fermionic component of a chiral superfield can be assigned by setting the attribute `Chirality` to `Left` or `Right`.

It is mandatory to match each superfield to its component fields. The fermionic and scalar components of a chiral superfield are hence referred to as the values of the attributes `Weyl` and `Scalar`, respectively, whilst the vector and gaugino components of a vector superfield are stored in the attributes `GaugeBoson` and `Gaugino`. Each of these options must point to the symbol associated to the relevant fields. In contrast, the declaration of the auxiliary (F- and D-) fields is optional. If not specified by the user, FeynRules takes care of it internally (as for the `XI` and `VWZ` superfields above). If the user however desires to match the auxiliary component of a superfield to an existing unphysical scalar field, he/she has to provide a value for the attribute `Auxiliary` (as for the `PHI` superfield above). All the components (`psi`, `z`, `FF`, `V` and `lambda` in the examples above) are assumed to be properly declared at the time of the field declaration as described in Section 2.4.

In addition, the attributes `Indices` and `QuantumNumbers` are also available for the superfield class. We refer to Section 2.4 for more details. The complete list of the attributes of the superfield class is indicated in Table 9.

2.6 Gauge Groups

2.6.1 Gauge group declaration

The structure of the interactions of a model is in general dictated by gauge symmetries. Fields are chosen to transform in specific representations of the gauge group, and the invariance of the action under gauge transformation then governs the form of the interactions. In this section, we focus on the declaration of the gauge groups in FeynRules. Doing so allows the user to use several functions dedicated to an efficient implementation of the Lagrangian, such as, *e.g.*, covariant derivatives or (super)field strength tensors.

The gauge group of a model is either simple or semi-simple, and the different simple factors are defined independently in the list `M$GaugeGroups`,

```
M$GaugeGroups = {
  Group1 == { options1 },
  Group2 == { options2 },
        ...
```

Table 9: Superfield class options

`ClassName`	Refers to the symbol associated to a superfield.
`Chirality`	Refers to the chirality of a chiral superfield. It has to be set to the value `Left` or `Right`.
`GaugeBoson`	Refers to the symbol associated to the vector component of a vector superfield.
`Gaugino`	Refers to the symbol associated to the gaugino component of a vector superfield.
`Weyl`	Refers to the symbol associated to the fermionic component of a chiral superfield.
`Scalar`	Refers to the symbol associated to the scalar component of a chiral superfield.
Optional attributes	
`Auxiliary`	Refers to the symbol associated to the auxiliary component of a superfield. If not specified, FeynRules handles it internally.
`Indices`	Refers to the list of indices carried by a superfield.
`QuantumNumbers`	A list with the $U(1)$ quantum numbers carried by a superfield.

```
};
```

Each element of this list consists of an equality defining one specific direct factor of the full symmetry group. The left-hand sides of these equalities are labels associated to the different subgroups (`Group1`, `Group2`, *etc.*) while the right-hand sides contain sets of replacement rules defining the properties of the subgroups (`options1`, `options2`, *etc.*). For example, let us consider the implementation of the SM gauge group in FeynRules,

```
M$GaugeGroups = {
  U1Y  == {
    Abelian          -> True,
```

```
    CouplingConstant -> gp,
    GaugeBoson       -> B,
    Charge           -> Y
  },
  SU2L == {
    Abelian           -> False,
    CouplingConstant  -> gw,
    GaugeBoson        -> Wi,
    StructureConstant -> ep,
    Representations   -> {{Ta,SU2D}},
    Definitions       -> {Ta[a__]->PauliSigma[a]/2, ep->Eps}
  },
  SU3C ==  {
    Abelian           -> False,
    CouplingConstant  -> gs,
    GaugeBoson        -> G,
    StructureConstant -> f,
    Representations   -> {{T,Colour}},
    SymmetricTensor   -> dSUN
  }
}
```

Abelian and non-abelian groups are distinguished by the option `Abelian`, which may take the values `True` or `False`.

In the case of abelian groups, the user has the possibility to define the associated $U(1)$ charge by setting the attribute `Charge` of the gauge group class to a symbol related to this operator. This allows FeynRules to check for charge conservation at the Lagrangian or at the Feynman rules level, assuming that the charges of the different (super)fields have been properly declared via the option `QuantumNumbers` of the particle class.

Non-abelian groups have specific attributes to define the generators and structure constants. For example, the symbols of the symmetric and antisymmetric structure constants (whose indices are those of the adjoint representation) of the group are defined as the values of the attributes `SymmetricTensor` and `StructureConstant`. Furthermore, the representations of the group necessary to define all the fields of the model are listed through the attribute `Representations`, which takes as a value a list of pairs. The first component of each pair is a symbol defining the symbol for the generator, while the second one consists of the type of indices it acts on. For example, in the declaration of the $SU(3)$ gauge group above, FeynRules internally creates a matrix of $SU(3)$ denoted by `T[Gluon,Colour,Colour]`, where we have explicitly indicated the index types. The first index, represented here by the symbol `Gluon`, stands for the adjoint gauge index. Even if this information

is not explicitly provided at the time of the gauge group declaration, FeynRules infers it from the value of the `GaugeBoson` (or `Superfield`) attribute (see below). Finally, the attribute `Definitions` allows the user to provide analytical formulas expressing representation matrices and structure constants in terms of the model parameters and/or standard Mathematica variables (see the $SU(2)_L$ declaration in the example above).

At this stage we have to make a comment about how FeynRules deals with group representations. As should be apparent from the previous definition, FeynRules does not make any distinction between a representation and its complex conjugate. Indeed, if T_R denote the generators of some irreducible representation and $T_{\overline{R}}$ are the generator of the complex conjugate representation, then it is always possible to eliminate all the generators $T_{\overline{R}}$ from the Lagrangian using the formula

$$\left(T_{\overline{R}}\right)_{ij} = -\left(T_R\right)^*_{ij} = -\left(T_R\right)_{ji} . \tag{2.6}$$

Furthermore, if a field transforms in the representation $\overline{R}$, then the antifield transforms in the representation R. We can thus always choose the particles as the ones transforming in the representation R. If the user wants to introduce an $SU(3)$ antitriplet, for example, he/she can add a field `phi` transforming as a triplet. Consequently, `phibar` transforms as an antitriplet and can be used as such when writing the Lagrangian.

Information on the gauge boson responsible for the mediation of the gauge interactions is provided through the `GaugeBoson` attribute of the gauge group class. It refers to the symbol of the vector field associated to the group, defined separately in `M$ClassesDescriptions`. For non-abelian symmetries, the (non-Lorentz) index carried by this field consists of the adjoint index. As briefly mentioned above, this is the way used by FeynRules to define adjoint gauge indices. For supersymmetric models, it is also possible to link a vector superfield instead of a gauge boson through the attribute `Superfield`. If both the `Superfield` and `GaugeBoson` attributes are specified by the user, they must be consistent, *i.e.*, the gauge boson must be the vector component of the vector superfield.

The last attribute appearing in the examples above consists of the model parameter to be used as the gauge coupling constant, which is specified through the option `CouplingConstant`.

The list of all the options described above is summarized in Table 10.

Table 10: Gauge group options

`Abelian`	Mandatory, specifying whether the gauge group is abelian (`True`) or not (`False`).
`GaugeBoson`	Refers to the name of the gauge boson associated with the gauge group. This attribute must be specified, except if a superfield is related to the gauge group through the attribute `Superfield`. In addition, the gauge boson must be properly declared in the particle list (see Section 2.4).
`Superfield`	Refers to the name of the vector superfield associated with the gauge group. This attribute must be specified, except if a vector boson is related to the gauge group through the attribute `GaugeBoson`. The superfield must be declared in the superfield list (see Section 2.5).
`CouplingConstant`	Refers to the parameter standing for the coupling constant associated with the gauge group.
`Charge`	Mandatory for abelian groups, referring to the symbol associated with the $U(1)$ charge connected with this gauge group.
`StructureConstant`	Mandatory for non-abelian groups, referring to the symbol associated with the structure constants of the gauge group.
`SymmetricTensor`	Refers to the symbol associated with the totally symmetric tensor of a non-abelian group.

2.6.2 FeynRules *functions related to gauge groups*

The declaration of a gauge group enables FeynRules to automatically construct field strength tensors, superfield strength tensors and covariant deriva-

Table 10: Gauge group options (continued)

`Representations`	Refers to a list of two-component lists containing all the representations defined for this gauge group. The first component of these lists consists of the symbol by which the generators of the representation are denoted, while the second component is the name of the index it acts on.
`Definitions`	Contains a list of replacement rules that should be applied by FeynRules before calculating vertices, expressing representation matrices and/or structure constants in terms of the model parameters and Mathematica standard objects.

tives associated with this group, so that they can be further used when building Lagrangians. In the case of abelian gauge groups, the field strength tensor is invoked by issuing

```
FS[A, mu, nu]
```

where `A` is the corresponding gauge boson and `mu` and `nu` denote Lorentz indices. Its supersymmetric counterparts can be called by the command

```
SuperfieldStrengthL[ V, sp    ]
SuperfieldStrengthR[ V, spdot ]
```

respectively. In these commands, `V` stands for the vector superfield associated with the gauge group and `sp` and `spdot` are left-handed and right-handed spin indices. These three functions can be easily generalized to the non-abelian case,

```
FS[ A, mu, nu, a ]
SuperfieldStrengthL[ V, sp   , a ]
SuperfieldStrengthR[ V, spdot, a ]
```

where a stands for an adjoint gauge index. Following the FeynRules conventions, these quantities are defined as

$$\begin{aligned} F^a_{\mu\nu} &= \partial_\mu A^a_\nu - \partial_\nu A^a_\mu + g f^a{}_{bc} A^b_\mu A^c_\nu \ , \\ W_\alpha &= -\frac{1}{4}\bar{D}\cdot\bar{D} e^{2gV} D_\alpha e^{-2gV} \ , \\ \overline{W}_{\dot{\alpha}} &= -\frac{1}{4} D\cdot D e^{-2gV} \bar{D}_{\dot{\alpha}} e^{2gV} \ , \end{aligned} \tag{2.7}$$

where g and f denote the coupling constant and the structure constants of the gauge group and D and $\bar{D}$ are the superderivatives defined below, in Section 4.5. The abelian limit is trivially derived from these expressions. We emphasize that the spinorial superfields W_α and $\overline{W}_{\dot{\alpha}}$ are not hard-coded in FeynRules and are recalculated each time. However, they are evaluated only when an expansion in terms of the component fields of the vector superfield `V` is performed.

From the information provided at the time of the declaration of the gauge group, FeynRules can also define, in an automated way, gauge covariant derivatives. These can be accessed through the symbol `DC[phi, mu]`, where `phi` is the field that it acts on and `mu` the Lorentz index. In our conventions, the covariant derivative reads

$$D_\mu \phi = \partial_\mu \phi - i g A^a_\mu T_a \phi \tag{2.8}$$

where T_a stands for the representation matrices associated to the representation of the gauge group in which the field ϕ lies. The sign convention in Eq. (2.8) is consistent with the sign convention in Eq. (2.7).

All the functions presented in this section are summarized in Table 12.

2.7 Model restrictions

In phenomenological studies, it can sometimes be useful to consider restricted models which are obtained from a parent model by putting some of the external parameters to zero. As an example, one might be interested in the Standard Model with a diagonal CKM matrix. While it is of course always possible to make the CKM matrix numerically diagonal, it is desirable to remove the interaction terms proportional to the off-diagonal terms altogether in order to avoid a proliferation of vanishing diagrams in Feynman diagram calculations. This can be achieved by the use of the so-called restriction files in FeynRules. Restriction files are text files (with the extension `.rst`) that contain a list, named `M$Restrictions`, of replacement rules to be applied to the Lagrangian before the evaluation of the Feynman rules. As an example, a possible restriction file to restrict the CKM matrix to a diagonal matrix reads

Table 12: Field strength tensor and covariant derivative

`FS[A, mu, nu]`	Constructs the field strength tensor $F_{\mu\nu}$ connected with the $U(1)$ gauge boson `A`.
`FS[A, mu, nu, a]`	Constructs the field strength tensor $F^a_{\mu\nu}$ connected with the non abelian gauge boson `A`.
`SuperfieldStrengthL[V, sp]`	Calculates the left-handed superfield strength tensor associated with the abelian vector superfield `V`.
`SuperfieldStrengthL[V, sp, a]`	Calculates the left-handed superfield strength tensor associated with the non-abelian vector superfield `V`.
`SuperfieldStrengthR[V, spdot]`	Calculates the right-handed superfield strength tensor associated with the abelian vector superfield `V`.
`SuperfieldStrengthR[V, spdot, a]`	Calculates the right-handed superfield strength tensor associated with the non-abelian vector superfield `V`.
`DC[phi, mu]`	The covariant derivative acting on the field `phi` with a Lorentz index `mu`.

```
M$Restrictions = {
            CKM[i_,i_] -> 1,
            CKM[i_?NumericQ, j_?NumericQ] :> 0 /; (i =!= j)
}
```

Applying these rules to the Lagrangian (or to the vertices) obviously removes all the flavor-changing interactions among quark fields.

Restriction files can be loaded at run time after the model file has been loaded, *i.e.*, after the `LoadModel[ ]` command has been issued. The corresponding command is

```
LoadRestriction[ file1.rst, file2.rst, ... ]
```

After this function has been called, the parameter definitions inside FeynRules are updated and the parameters appearing in the left-hand side of the replacement rules are removed from the model. In addition, the rules are kept in memory and are applied automatically to the Lagrangian when computing the Feynman rules. Note that this process is irreversible, and the restrictions cannot be undone after the `LoadRestriction[]` command has been issued (unless the kernel is restarted).

For complicated models, one often chooses the benchmark point in a special way such that many of the new parameters in the model are zero, and only a few new interactions are considered. FeynRules provides a command that allows to identify the parameters that are numerically zero and to create a restriction file that removes all the vanishing parameters from the model. More precisely, the command

```
WriteRestrictionFile[]
```

creates a restriction file called `ZeroValues.rst` which can be loaded just like any other restriction file. We emphasize that for complicated model the use of this restriction file may considerably speed up calculation performed with Feynman diagram calculators.

2.8 Mixing declaration

FeynRules comes with a module allowing for the derivation of the mass spectrum included in any Lagrangian (see Section 4.7). The mass matrices calculated in this way can be further passed to the ASperGe program [43] for a numerical evaluation of the necessary rotations yielding a diagonal mass basis (see Section 6.2). In order to employ these functionalities, the user has to implement particle mixings in a specific way, using instances of the mixing class collected into a list dubbed `M$MixingsDescription`,

```
M$MixingsDescription = {
   Mix["l1"] == { options1 },
   Mix["l2"] == { options2 },
          ...
}
```

This maps a label given as a string (`"l1"`, `"l2"`, *etc.*) to a set of Mathematica replacement rules defining mixing properties (`options1`, `options2`, *etc.*). To illustrate the available options, we take the example of the W_1 and W_2 gauge

fields of $SU(2)_L$ that mix, in the Standard Model, to the $W^{\pm}$-bosons

$$W_\mu^+ = \frac{W_\mu^1 - iW_\mu^2}{\sqrt{2}} \quad \text{and} \quad W_\mu^- = \frac{W_\mu^1 + iW_\mu^2}{\sqrt{2}} \,. \tag{2.9}$$

As this mixing relation does not depend on any model parameter and is fully numerical, it can be declared in a very compact form,

```
Mix["l1"] == {
  MassBasis  -> {W, Wbar},
  GaugeBasis -> {Wi[1], Wi[2]},
  Value      -> {{1/Sqrt[2],-I/Sqrt[2]},{1/Sqrt[2],I/Sqrt[2]}}
}
```

The previous mixing declaration should be understood as the matrix product

```
MassBasis = Value . GaugeBasis
```

Information on the gauge and mass bases is provided through the attributes `GaugeBasis` and `MassBasis` and the mixing matrix is given in a numerical form as the argument of the attribute `Value`. In the case the user already knows analytical formulas for the element of the mixing matrix, the syntax above cannot be employed. This matrix has instead to be declared as a standard model parameter (see Section 2.3) and referred to via the `MixingMatrix` attribute of the mixing class (see below). As the gauge basis is unphysical, it has to only contain fields declared as such, while the mass basis can in contrast contain either physical fields, unphysical fields or both. When unphysical fields are present, particle mixings are effectively implemented in several steps. We refer to Ref. [43] for examples. In order to implement the bases in a more compact form, spin and Lorentz indices can be omitted. Moreover, if some indices are irrelevant, *i.e.*, if they are identical for all the involved fields, underscores can be employed, as in the example

```
Mix["l2"] == {
  MassBasis  -> {sdL[1,_], sdL[2,_], sdL[3,_]},
  GaugeBasis -> {QLs[2,1,_], QLs[2,2,_], QLs[2,3,_]},
                ...
}
```

Underscores reflect that the last index of each field has the same type and is not affected by the mixing (such as color indices for instance).

Most of the time, the mixing matrix is not known numerically before implementing the model, which leads to a slightly different syntax at the level of the implementation. The declaration

```
Mix["l3"] == {
```

```
  MassBasis    -> {A, Z},
  GaugeBasis   -> {B, Wi[3]},
  MixingMatrix -> UW,
  BlockName    -> WEAKMIX
}
```

describes the rotation of the third weak boson W_3 and the hypercharge gauge boson B to the photon and Z-boson states,

$$\begin{pmatrix} A_\mu \\ Z_\mu \end{pmatrix} = U_w \begin{pmatrix} B_\mu \\ W_\mu^3 \end{pmatrix} \ , \tag{2.10}$$

where we have introduced the mixing matrix U_w, the associated symbol being `UW`. The latter is referred to by means of the attribute `MixingMatrix` of the mixing class. The matrix `UW` does not have to be declared separately as this task is internally handled by FeynRules. Note that it is assumed that `UW` is a complex matrix with two indices, and according to our conventions we declare separately its real and imaginary parts. In other words, FeynRules creates two real external tensorial parameters, the real and imaginary parts of the matrix, and one internal complex tensorial parameter for the matrix `UW` itself. The user must specify the name of a Les Houches block [51, 52] which will contain the numerical values associated with the elements of the matrix (see Section 6.1.4). In the example above, we impose the real part of the elements of U_w to be stored in a block named `WEAKMIX` and their imaginary part in a block `IMWEAKMIX`.

It must be noted that mixing matrices declared through a mixing declaration cannot be employed explicitly in Lagrangians (such as for the CKM matrix in the Minimal Supersymmetric Standard Model). If the user wants to use the mixing matrices explicitly in the Lagrangian, he/she must declare the matrices following the standard syntax (see Section 2.3), numerical values being provided from the beginning.

Finally, it is sometimes more practical to implement a mixing relation under the form

```
GaugeBasis = MixingMatrix . MassBasis
```

as for the CKM rotation in the Standard Model,

$$d_L^0 = V_{\rm CKM} \cdot d_L \ , \tag{2.11}$$

where d_L^0 and d_L are respectively gauge- and mass-eigenstates. In order to avoid a cumbersome use of inverse matrices in mixing declarations, the user can make use of the optional attribute `Inverse` of the mixing class, setting it to `True`.

Table 13: Attributes of the mixing class.

`MassBasis`	Specifies the field basis in which the mass matrix is diagonal. Takes a list of bases as argument when multiple bases are involved.
`GaugeBasis`	Specifies the field basis that must be diagonalized. Takes a list of bases as argument when several bases are involved.
`Value`	Refers to the numerical value of a mixing matrix, when it is known at the time of the model implementation. Relates several mixing matrices to their values (given as a list) when mass diagonalization requires several rotations.
`MixingMatrix`	Refers to the symbol associated with a mixing matrix. Relates several mixing matrices to their symbols (given as a list) when mass diagonalization requires several rotations.
`BlockName`	Provides information about the name of the Les Houches block containing the mixing matrices (see Section 6.1.4). Relates several mixing matrices to their Les Houches blocks (given as a list) when mass diagonalization requires several rotations.
`Inverse`	When set to `True`, allows to use an inverse matrix for diagonalizing a basis. Takes a list of Boolean numbers as argument when mass diagonalization requires several rotations.

All the attributes of the mixing class are summarized in Table 13 while more involved cases are now detailed below.

When complex scalar fields are split into their real and imaginary degrees of freedom, it is necessary to declare one scalar and one pseudoscalar mass basis. In this case, the `MassBasis` attribute is given a list of the two bases. The first basis gives a list of the real scalar fields in the mass basis, while the second gives a list of the pseudoscalar fields. Consistently, the arguments of the attributes

`Value`, `BlockName`, `MixingMatrix` and `Inverse` are now lists as well, the first element of these lists always referring to scalar fields and the second one to pseudoscalar fields. When some of the elements of these lists are irrelevant, as for instance when the scalar mixing matrix is unknown (`MixingMatrix` and `BlockName` are used) and the pseudoscalar mixing matrix is known (`Value` is used), underscores have to be used to indicate the unnecessary entries of the lists, such as in

```
Mix["scalar"] == {
  MassBasis    -> { {h1, h2}, {a1, a2} },
  GaugeBasis   -> { phi1, phi2 },
  BlockName    -> { SMIX, _ },
  MixingMatrix -> { US, _ },
  Value        -> { _, ... }
}
```

In the case of four-component fermion mixing, two choices are possible. We recall that the mass terms (for Dirac-type fermions) can be generically written as

$$m\bar{\psi}\psi = m(\bar{\psi}_R\psi_L + \bar{\psi}_L\psi_R) \ , \tag{2.12}$$

all indices being suppressed for clarity. As a first option, the user can use a single gauge basis and FeynRules internally takes care of the chirality projectors appearing in the mass terms. In this case, the attribute `GaugeBasis` contains a list with the ψ fields above. As a second option, two bases with different particle classes standing for the left-handed and right-handed pieces of the fields can be employed. The user here sets the value of the `GaugeBasis` attribute to a two-component list. The first (second) element of this list is another list containing the names of the left-handed (right-handed) components of the fields, *i.e.*, the instance of the particle class describing the ψ_L (ψ_R) fields above. Additionally, the arguments of the `Value`, `BlockName`, `MixingMatrix` and `Inverse` attributes are consistently upgraded to lists too, the first component being associated with left-handed fermions and the second one with right-handed fermions. As for neutral scalar mixing, underscores can be used for irrelevant list elements. This second option is more commonly employed in the existing models although it may seem more complicated. We therefore give an explicit example:

```
Mix["dq"] == {
  MassBasis    -> {dq[1, _], dq[2, _], dq[3, _]},
  GaugeBasis   -> {
                    {QL[2, 1, _], QL[2, 2, _], QL[2, 3, _]},
                    {dR[1, _], dR[2, _], dR[3, _]}
                  },
  MixingMatrix -> {CKM, _},
  Value        -> {_, {{1,0,0}, {0,1,0}, {0,0,1}} },
```

```
  Inverse      -> {True, _}
}
```

In this example, we describe the implementation of the mixing relations associated with the down-type quarks in the Standard Model. The mass eigenstates are denoted by the three fields `dq` standing for the three generations of down-type quarks. The color indices being irrelevant, they are replaced by underscores everywhere. The left-handed gauge eigenstates are the second piece of the $SU(2)_L$ doublets of quarks (the `QL` fields) whereas the right-handed gauge eigenstates are the $SU(2)_R$ singlets (the `dR` quarks). Finally, the last three arguments of the instance of the mixing class above indicate that the left-handed fields mix trough Eq. (2.11) (the first elements of the attributes `MixingMatrix` and `Inverse` are set accordingly) and that the right-handed fields do not have to be rotated (the argument `Value` contains an identity matrix as its second element).

Lagrangian mass terms for charged Weyl fermions are generically written as

$$\left(\psi_1^-, \ldots, \psi_n^-\right) M \begin{pmatrix} \chi_1^+ \\ \vdots \\ \chi_n^+ \end{pmatrix} , \tag{2.13}$$

where M stands for the mass matrix and ψ_i^- and χ_i^+ are Weyl fermions with opposite electric charges. The diagonalization of the matrix M proceeds through two unitary rotations U and V,

$$\begin{pmatrix} \tilde{\psi}_1^- \\ \vdots \\ \tilde{\psi}_n^- \end{pmatrix} = U \begin{pmatrix} \psi_1^- \\ \vdots \\ \psi_n^- \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} \tilde{\chi}_1^+ \\ \vdots \\ \tilde{\chi}_n^+ \end{pmatrix} = V \begin{pmatrix} \chi_1^+ \\ \vdots \\ \chi_n^+ \end{pmatrix} , \tag{2.14}$$

which introduces two mass bases. Therefore, all the attributes `MassBasis`, `GaugeBasis`, `Value`, `MixingMatrix`, and `BlockName` now take lists as arguments (with underscores included where relevant). It is mandatory to consistently order these lists.

Finally, in realistic models, the ground state of the theory is non-trivial and fields must be shifted by their vacuum expectation value (vev). The case of electrically neutral scalar fields can be treated in a specific way and included easily in particle mixing handling. Information on the vevs is included in a list of two-component elements denoted by `M$vevs`. The first of these elements refers to an unphysical field while the second one to the symbol associated with its vev, as in

```
M$vevs = { { phi1, vev1 }, { phi2, vev2 } }
```

the vacuum expectation values `vev1` and `vev2` being declared as any other model parameter (see Section 2.3). In this way, the information on the vevs of the different fields is consistently taken into account when FeynRules internally rewrites the neutral scalar fields in terms of their real degrees of freedom. For the sake of the example, we associate with the vev declaration above the following definition of the mixing of the ϕ_1 and ϕ_2 gauge eigenstates to the real scalar (pseudoscalar) mass eigenstates h_1 and h_2 (a_1 and a_2),

```
Mix["phi"] == {
  MassBasis    -> { {h1, h2}, {a1, a2} },
  GaugeBasis   -> { phi1, phi2 },
  MixingMatrix -> { US, UP }
}
```

This teaches FeynRules to internally understand the mixing pattern of the ϕ_i fields as

$$\begin{pmatrix}\phi_1\\ \phi_2\end{pmatrix} = \frac{1}{\sqrt{2}}\left[\begin{pmatrix}v_1\\ v_2\end{pmatrix} + U_s^\dagger \begin{pmatrix}h_1\\ h_2\end{pmatrix} + iU_p^\dagger \begin{pmatrix}a_1\\ a_2\end{pmatrix}\right], \tag{2.15}$$

the scalar and pseudoscalar mixing matrices U_s and U_p being represented by the symbols `US` and `UP`, respectively.

3 The Lagrangian

An essential ingredient of a model implementation is the Lagrangian of the model. The Lagrangian can be built out of the fields of the models, augmented by some internal FeynRules and Mathematica functions. All the fields can be accessed by their `ClassName`, for example `psi[a,b,...]` where `a,b,...` are the indices of the field starting with its Lorentz indices for vector and tensor bosons, or its spin index for fermions followed by a Lorentz index for a spin-3/2 fermion [5]. The remaining indices are the ones defined in the particle definition through the `Indices` option, given in the same order. The different flavors can also be accessed using the names given in `ClassMembers`. They have the same indices as the full flavor multiplet, with the flavor index omitted. We recall that, if a field is not self-conjugate, FeynRules automatically creates the symbol for the conjugate field by adding '`bar`' at the end of the particle name, *i.e.*, the antiparticle associated to `psi` is denoted by `psibar`. For a fermion ψ, the conjugate field is $\bar{\psi} \equiv \psi^\dagger\gamma^0$. Alternatively, the conjugate field can be obtained by issuing `anti[psi]`.

[5] Spin-3/2 fields always have their spin index before their Lorentz index.

Fields (and their derivatives) can be combined into polynomials. By convention, all the indices appearing inside a monomial in FeynRules must be contracted, *i.e.*, all indices must appear pairwise[6]. Furthermore, all indices must be spelled out explicitly. For anticommuting fields (fermions and ghosts), the Mathematica `Dot` function has to be used, in order to keep the relative order among them fixed. For example, the interaction between the gluon and all the down quarks can be written as

```
gs Ga[mu, s, r] T[a, i, j] dqbar[s, f, i].dq[r, f, j] G[mu, a]
```

There is however one case where indices do not need to be spelled out completely but can be omitted. If in a fermion bilinear, all the indices of the rightmost fermion are connected to all the indices of the leftmost fermion (perhaps with intermediate matrices), then these indices can be suppressed and FeynRules takes care of restoring them internally, such as in

```
dqbar.Ga[mu].T[a].dq
            → Ga[mu,s,r] T[a,i,j] dqbar[s,f,i].dq[r,f,j] .
```

In case of doubt, the user should always spell out all indices explicitly.

The `Dot` product is mandatory for anticommuting fields or parameters. It should be noted that Mathematica does not keep the `Dot` product between the components of vectors or matrices after computing their product explicitly

```
{ubar, dbar}.{u, d}  =  u ubar + d dbar
```

The appropriate treatment requires, therefore, use of the `Inner` function for each `Dot`, *e*.g.

```
Inner[Dot, {ubar, dbar}, {u, d}]  =  ubar.u + dbar.d
```

or for more than one multiplication,

```
Inner[Dot, Inner[Dot, {ubar, dbar} ,
    {{a11, a12} , {a21, a22}}] , {u, d}] .
```

As already discussed in Section 2.6, gauge invariant derivatives can be conveniently defined via the functions `DC[phi,mu]` and `FS[G, mu, nu, a]`. The first argument of both functions is the relevant field, `mu` and `nu` are Lorentz indices and `a` represents an index of the adjoint representation of the associated gauge group. The gauge fields and generators that appear in covariant derivatives of a particular field are fixed by its indices and by the definition of

[6] With the exception of single-index parameters for which the `AllowSummation` option is set to `True` (see Section 2.3).

the gauge group. For example, the QCD Lagrangian for massless down quarks,

$$\mathcal{L}^{\mathcal{QCD}} \equiv -\frac{1}{4} G_a^{\mu\nu} G^a_{\mu\nu} + i\bar{d}\not{D}d, \tag{3.16}$$

is written as

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

All the predefined FeynRules functions useful for the building of the Lagrangian are given in Table 14.

Finally, it is often convenient to write a Lagrangian in terms of two-component fermions and to let FeynRules perform the transformations to four-component fermions. We note that this operation is mandatory for most Feynman diagram calculators, which in general only work with four-component spinors. More precisely, if χ and $\bar{\xi}$ are left and right-handed Weyl spinors, and $\psi = (\chi, \bar{\xi})^T$ is a Dirac fermion, we can easily switch to four-component fermions by using the replacements

$$\begin{aligned} \chi &\to \frac{1-\gamma^5}{2}\psi\,, \qquad & \xi &\to \frac{1-\gamma^5}{2}\psi^c\,, \\ \bar{\chi} &\to \frac{1+\gamma^5}{2}\psi^c\,, \qquad & \bar{\xi} &\to \frac{1+\gamma^5}{2}\psi\,. \end{aligned} \tag{3.17}$$

These transformation rules are implemented in FeynRules via the `WeylToDirac` function, which takes as an argument a Lagrangian written in terms of two-component fermions, and returns the same Lagrangian in terms of four-component fermions.

3.1 Tools for Lagrangians

FeynRules provides functions, collected in Table 15, that can be used while constructing Lagrangians. For example, the function `ExpandIndices[]` returns the Lagrangian with all the indices written explicitly. Each of the other functions return a different part of the Lagrangian as described in the table.

Once the Lagrangian is implemented, several sanity checks can be performed by means of the functions presented in Table 16. First, the function

```
CheckHermiticity[ L ];
```

checks if the Lagrangian $\mathcal{L}$ is Hermitian. Next, three functions are available to check if the kinetic terms and the mass terms are diagonal, `CheckDiagonalKineticTerms`, `CheckDiagonalMassTerms` and `CheckDiagonalQuadraticTerms`.

Table 14: Special symbols for Lagrangians

`del[`ϕ, μ`]`	Partial derivative of ϕ with respect to the space-time coordinate x^μ.
`DC[`ϕ, μ`]`	Gauge covariant derivative of ϕ with respect to the space-time coordinate x^μ.
`ME[`μ, ν`]`	Minkowski metric tensor $\eta_{\mu\nu}$.
`IndexDelta[`i,j`]`	Kronecker delta δ_{ij}.
`Eps[`a,. . . , b`]`	Totally antisymmetric Levi-Civita tensor with respect to the indices `a,...,b`. In the case of four indices, the conventions on the ϵ-tensors are important when employing interfaces to matrix element generators. They are based on $\epsilon_{0123} = +1$.
`HC[`expr `]`	Hermitian conjugate of expr.
`CC[`ψ`]`	The charge conjugate ψ^c of a Dirac field ψ.
`FS`	Field strength tensor.
`Ga[`μ`]`, `Ga[`μ, i, j`]`	Dirac Matrix γ^μ, γ^μ_{ij}.
`ProjP`, `ProjP[`i, j`]`	Chiral projection operator $\frac{1+\gamma^5}{2}$, $\left(\frac{1+\gamma^5}{2}\right)_{ij}$.
`ProjM`, `ProjM[`i, j`]`	Chiral projection operator $\frac{1-\gamma^5}{2}$, $\left(\frac{1-\gamma^5}{2}\right)_{ij}$.
`ProjP[`μ`]`, `ProjP[`μ, i, j`]`	$\gamma^\mu \frac{1+\gamma^5}{2}$, $\left(\gamma^\mu \frac{1+\gamma^5}{2}\right)_{ij}$.
`ProjM[`μ`]`, `ProjM[`μ, i, j`]`	$\gamma^\mu \frac{1-\gamma^5}{2}$, $\left(\gamma^\mu \frac{1-\gamma^5}{2}\right)_{ij}$.
`right[`ψ`]`	The right-handed part of the four-component fermion field ψ, $\frac{1+\gamma^5}{2}\psi$.
`left[`ψ`]`	The left-handed part of the four-component fermion field ψ, $\frac{1-\gamma^5}{2}\psi$.
`si[`μ`]`, `si[`μ, α, $\dot{\alpha}$`]`	Pauli matrix σ^μ, $\sigma^\mu{}_{\alpha\dot{\alpha}}$.
`sibar[`μ`]`, `sibar[`μ, $\dot{\alpha}$, α`]`	Pauli matrix $\bar{\sigma}^\mu$, $\bar{\sigma}^{\mu\dot{\alpha}\alpha}$.

Table 15: Manipulating a Lagrangian

All the functions below share the same options as `FeynmanRules`, which can be found in Table 19.

`ExpandIndices[`$\mathcal{L}$, options`]`	Restores all the suppressed indices in the Lagrangian $\mathcal{L}$.
`GetKineticTerms[`$\mathcal{L}$, options`]`	Returns the kinetic terms in the Lagrangian $\mathcal{L}$.
`GetMassTerms[`$\mathcal{L}$, options`]`	Returns the mass terms in the Lagrangian $\mathcal{L}$.
`GetQuadraticTerms[`$\mathcal{L}$, options`]`	Returns the quadratic terms in the Lagrangian $\mathcal{L}$.
`GetInteractionTerms[`$\mathcal{L}$, options`]`	Returns the interaction terms in the Lagrangian $\mathcal{L}$.
`SelectFieldContent[`$\mathcal{L}$, list`]`	Returns the part of the Lagrangian $\mathcal{L}$ corresponding to the field content specified in list.

Finally, two functions, `CheckKineticTermNormalisation` and `CheckMassSpectrum`, allow to check the normalization of the kinetic terms and compare the masses computed from the Lagrangian to those of the model description. The FeynRules conventions on the normalization of the kinetic and mass terms for the scalar, spin 1/2 and vector fields are

(1) Scalar fields:
- Real:
$$\frac{1}{2}\partial_\mu\phi\partial^\mu\phi - \frac{1}{2}m^2\phi^2,$$
- Complex (including ghost fields):
$$\partial_\mu\phi^\dagger\partial^\mu\phi - m^2\phi^\dagger\phi,$$

(2) Spin-1/2 fermions:
- Majorana:
$$\frac{1}{2}\bar{\lambda}i\partial\!\!\!/\lambda - \frac{1}{2}m\bar{\lambda}\lambda,$$
- Dirac:
$$\bar{\psi}i\partial\!\!\!/\psi - m\bar{\psi}\psi,$$

(3) Vectors:

Table 16: Checking the consistency of a Lagrangian

All the functions below share the same options as `FeynmanRules`, which can be found in Table 19.

`CheckHermiticity[` $\mathcal{L}$, options `]`

Checks if the Lagrangian $\mathcal{L}$ is Hermitian.

`CheckDiagonalKineticTerms[` $\mathcal{L}$, options `]`

Checks if all the kinetic terms in the Lagrangian $\mathcal{L}$ are diagonal.

`CheckDiagonalMassTerms[` $\mathcal{L}$, options `]`

Checks if all the mass terms in the Lagrangian $\mathcal{L}$ are diagonal.

`CheckDiagonalQuadraticTerms[` $\mathcal{L}$, options `]`

Checks if all the quadratic terms in the Lagrangian $\mathcal{L}$ are diagonal.

`CheckKineticTermNormalisation[` $\mathcal{L}$, options `]`

Checks if all the kinetic terms in the Lagrangian $\mathcal{L}$ are correctly diagonalized and normalized.

`CheckMassSpectrum[` $\mathcal{L}$, options `]`

Checks if all the mass terms in the Lagrangian $\mathcal{L}$ are correctly diagonalized and if their value corresponds to the numerical value given in the model file.

- Real:

$$-\frac{1}{4}F_{\mu\nu}F^{\mu\nu}-\frac{1}{2}m^2A_\mu A^\mu,$$

- Complex:

$$-\frac{1}{2}F^\dagger_{\mu\nu}F^{\mu\nu}-m^2A^\dagger_\mu A^\mu.$$

FeynRules does not use the quadratic pieces of a Lagrangian. However, the propagators hard-coded either in FeynRules or in the event generators assume that the quadratic piece of the Lagrangian follow the above-mentioned conventions. Furthermore, since the kinetic and mass terms for spin-3/2 and spin-2 fields are model dependent, they are therefore not implemented. Finally, checks on Weyl fermion kinetic and mass terms are also not supported since there exist several ways to write them down.

The implementation of supersymmetric Lagrangians in FeynRules can be highly facilitated by means of a series of dedicated built-in functions. The Lagrangian describing the kinetic terms and the gauge interactions of the chiral content of any supersymmetric theory is given by

$$\begin{aligned}\mathcal{L}_{\text{chiral}} &= \left[\Phi_i^\dagger\, e^{-2g_j V^j}\, \Phi^i\right]_{\theta\cdot\theta\bar{\theta}\cdot\bar{\theta}} \\ &= D_\mu\phi_i^\dagger D^\mu\phi^i + F_i^\dagger F^i - \frac{i}{2}\Big(D_\mu\bar{\psi}_i\bar{\sigma}^\mu\psi^i - \bar{\psi}_i\bar{\sigma}^\mu D_\mu\psi^i\Big) \\ &+ i\sqrt{2}g_j\bar{\lambda}^{ja}\cdot\bar{\psi}_i T_a\phi^i - i\sqrt{2}g_j\phi_i^\dagger T_a\psi^i\cdot\lambda^{ja} - g_j D^{ja}\phi_i^\dagger T_a\phi^i\ ,\end{aligned} \tag{3.18}$$

where the superfield content of the theory is represented by a set of chiral superfields $\{\Phi^i = (\phi^i, \psi^i, F^i)\}$ and vector superfields $\{V^j = (v^j, \lambda^j, D^j)\}$. In the first line of the equation above, the $[\,.\,]_{\theta\cdot\theta\bar{\theta}\cdot\bar{\theta}}$ indicates that one has to extract the highest-order coefficient in θ and $\bar{\theta}$ from the expansion of the superfield expression included in the brackets. We recall that the covariant derivatives are defined in Eq. (2.8) and that the matrices T_a stand for representation matrices of the gauge group relevant to the fields they act on. This Lagrangian can be computed in FeynRules by issuing

```
Theta2Thetabar2Component[ CSFKineticTerms[ ] ]
```

The function `CSFKineticTerms` returns the kinetic and gauge interaction Lagrangian terms for all the chiral supermultiplets of the model. As a result, the Lagrangian is computed in terms of superfields. The method `Theta2Thetabar2Component` is then needed to ensure that only the terms in $\theta\cdot\theta\bar{\theta}\cdot\bar{\theta}$ are kept, after the superfields have been expanded in terms of their component fields. If the user is interested in one specific superfield, represented by, *e.g.*, the symbol `Phi`, the corresponding Lagrangian can be obtained by specifying this superfield as the argument of the function `CSFKineticTerms`,

```
Theta2Thetabar2Component[ CSFKineticTerms[ Phi ] ]
```

Non-gauge interactions among the chiral superfields are driven by the superpotential, a holomorphic function $W(\Phi)$ of the chiral superfields Φ of the theory. The associated interaction Lagrangian is given by

$$\mathcal{L}_W = \Big[W(\Phi)\Big]_{\theta\cdot\theta} + \text{h.c.} = F^i\frac{\partial W(\phi)}{\partial\phi^i} + \frac{1}{2}\frac{\partial^2 W(\phi)}{\partial\phi^i\partial\phi^j}\psi^i\cdot\psi^j + \text{h.c.}\ , \tag{3.19}$$

where in our notations, $[\,.\,]_{\theta\cdot\theta}$ indicates that one has to select the component in $\theta\cdot\theta$ from the expansion of the superpotential in terms of the Grassmann variables θ and $\bar{\theta}$. Assuming that $W(\Phi)$ is represented in the FeynRules model file by a variable `SuperW`, the Lagrangian can be calculated by issuing

```
Theta2Component[SuperW] + Thetabar2Component[HC[SuperW]]
```

where the two functions `Theta2Component` and `Thetabar2Component` allow to extract the correct coefficients of the expansion of the superpotential in θ and $\bar{\theta}$.

We now turn to the Lagrangian associated with the gauge sector. Kinetic and gauge interaction terms are built from squaring the superfield strength tensors and read, in the non-abelian case[7],

$$\begin{aligned}\mathcal{L}_V &= \frac{1}{16g^2}\Big[W^{\alpha a}W_{\alpha a}\Big]_{\theta\cdot\theta} + \text{h.c.} \\ &= -\frac{1}{4}F_a^{\mu\nu}F^a_{\mu\nu} + \frac{i}{2}(\lambda_a\sigma^\mu D_\mu\bar{\lambda}^a - D_\mu\lambda_a\sigma^\mu\bar{\lambda}^a) + \frac{1}{2}D^aD_a\ .\end{aligned} \tag{3.20}$$

Implementing this Lagrangian into FeynRules can be achieved by issuing

```
LV = Theta2Component[ VSFKineticTerms[ ] ] +
    Thetabar2Component[ VSFKineticTerms[ ] ]
```

where an implicit sum over the whole vector superfield content of the theory is performed. The function `VSFKineticTerms` allows us to construct the Lagrangian in terms of superfields and the functions `Theta2Component` and `Thetabar2Component` to select the $\theta\cdot\theta$ and $\bar{\theta}\cdot\bar{\theta}$ components of the expansion of the superfield Lagrangian in terms of the Grassmann variables, respectively. Specifying the arguments of the function `VSFKineticTerms`,

```
LV = Theta2Component[ VSFKineticTerms[ VSF ] ] +
    Thetabar2Component[ VSFKineticTerms[ VSF ] ]
```

allows us to compute the kinetic and gauge interaction Lagrangian terms associated with the vector superfield represented by the symbol `VSF`.

To summarize, implementing a (renormalizable) Lagrangian density for a supersymmetric theory in FeynRules only requires the definition of the superpotential `SuperW`, since the rest of the Lagrangian computation is automatic. The full (supersymmetric) Lagrangian is thus calculated by issuing

```
LC=Theta2Thetabar2Component[CSFKineticTerms[]];
LV=Theta2Component[VSFKineticTerms[]] +
      Thetabar2Component[VSFKineticTerms[]];
LW=Theta2Component[SuperW]+Thetabar2Component[HC[SuperW]];
Lag = LC + LV + LW;
```

[7] The abelian limit can be easily derived.

The Lagrangian density obtained in this way still depends on the auxiliary F and D-fields that can be eliminated by inserting the solutions of their equations of motion back into the Lagrangian. This operation is automated in FeynRules by means of the `SolveEqMotionD` (this eliminates the D-fields), `SolveEqMotionF` (this eliminates the F-fields) and `SolveEqMotionFD` (this eliminates all auxiliary fields) functions. More precisely, all the auxiliary fields that could appear in the Lagrangian represented by the symbol `Lag` are eliminated by typing one of the two commands

```
SolveEqMotionFD[Lag]
SolveEqMotionF[SolveEqMotionD[Lag]]
```

Finally, the component fields of a supermultiplet are Weyl fermions and need to rewritten in terms of four-component spinors. This can be achieved using the `WeylToDirac[]` function described in the beginning of this section.

For more information on the superspace module of FeynRules we refer to refs. [41, 53] and to Table 17 which collects all the functions presented in this section.

4 Running FeynRules

After the model description is created and the Lagrangian constructed, it can be loaded into FeynRules and the Feynman rules obtained.

4.1 Loading FeynRules

The first thing that must be done when using FeynRules is to load the package into a Mathematica session. This should be done before any of the model description is loaded in the kernel and so should be the first line of the Mathematica notebook[8]. In order to load FeynRules, the user must first specify the directory where it is stored and then load it by issuing

```
$FeynRulesPath = SetDirectory[ <the address of the package> ];
<< FeynRules`
```

[8] In other words, if the model description is done in a Mathematica notebook, it should come after FeynRules is loaded.

Table 17: Constructing supersymmetric Lagrangians.

`CSFKineticTerms[csf]`	Derives kinetic and gauge interaction terms associated with the chiral superfield `csf`. If called without any argument, a sum over the whole chiral content of the theory is performed.
`VSFKineticTerms[vsf]`	Derives kinetic and gauge interaction terms associated with the vector superfield `vsf`. If called without any argument, a sum over the whole gauge content of the theory is performed.
`SolveEqMotionFD[` $\mathcal{L}$ `]`	Computes and solves the equations of motion for all auxiliary fields. The solutions are then inserted in the Lagrangian $\mathcal{L}$.
`SolveEqMotionD[` $\mathcal{L}$ `]`	Computes and solves the equations of motion for the auxiliary D-fields. The solutions are then inserted in the Lagrangian $\mathcal{L}$.
`SolveEqMotionF[` $\mathcal{L}$ `]`	Computes and solves the equations of motion for the auxiliary F-fields. The solutions are then inserted in the Lagrangian $\mathcal{L}$.
`WeylToDirac[` $\mathcal{L}$ `]`	Reexpresses a Lagrangian $\mathcal{L}$, containing two-component Weyl fermions, in terms of four-component fermions.

4.2 Loading the model file

After the FeynRules package has been loaded[9], the model can be loaded using the command `LoadModel`,

```
LoadModel[ < file.fr >, < file2.fr >, ... ]
```

The model may be contained in one model file or split among several files. For FeynRules to run properly, the extension of each model file should be `.fr`.

[9] The user may want to change the current directory of Mathematica at this point. Otherwise, all the files and directories generated by FeynRules may end up in the main FeynRules directory.

If the model description is entered directly in the MATHEMATICA notebook, the list of files is then empty. In this case, `LoadModel[]` has to be executed after all the lines of the model description are loaded into the kernel.

Any time the model description changes, the model must be reloaded. Currently, this means that the MATHEMATICA kernel must be quit and the FEYNRULES package and model must be reloaded from the beginning. An exception to this is the Lagrangian. It can be changed and extended without having to reload the model information.

In the rest of this section, we describe the main utilities included in FEYNRULES which are summarized in Table 18.

4.3 Extracting the Feynman rules

After the model is loaded and the Lagrangian is defined, the Feynman rules can be extracted using the command `FeynmanRules`. For the rest of this section, we use the QCD Lagrangian defined in Eq. (3.16) as an example. The Feynman rules can be generated by means of the command [10]:

```
vertsQCD = FeynmanRules[ LQCD ];
```

The vertices derived by FEYNRULES are then written out on the screen and stored internally in the variable `vertsQCD`. The function `FeynmanRules` has several options, that are described below.

The user can instruct MATHEMATICA to not write the Feynman rules to the screen with the option `ScreenOutput` set to `False`,

```
vertsQCD = FeynmanRules[ LQCD, ScreenOutput -> False];
```

In this case, the Feynman rules are generated, stored in `vertsQCD`, but not displayed on screen.

In the two previous examples, the flavors were not expanded. For example, the preceding commands will only generate a single quark-gluon vertex (`dq dqbar G`). It is often desirable to perform a flavor expansion, *i.e.*, to obtain separately the vertices `d dbar G`, `s sbar G`, and `b bbar G`. To achieve this, the user can employ the `FlavorExpand` option. This option can be used to specify individual flavor indices to be expanded over, as in `FlavorExpand->Flavor` where only the `Flavor` index is expanded over (and not any other flavor indices if defined). It may also refer to a list of flavor indices to be expanded, as in

[10] Since the vertices list may be very long, it is usually desirable to end this statement with a semicolon.

`FlavorExpand->{Flavor,SU2W}`. Similarly, it may be used to expand over all the flavor indices as in `FlavorExpand->True`. Note that it is always possible to expand over the flavors at a later stage using the `FlavorExpansion[ ]` function, i.e.,

```
vertices = FeynmanRules[ L ];
vertices = FlavorExpansion[ vertices ];
```

which is equivalent to calling `FeynmanRules[ L, FlavorExpand -> True ]`. Note that calling the `FlavorExpansion[ ]` function in general runs faster than the `FlavorExpand` option. The difference is that when using the `FlavorExpand` option with `FeynmanRules`, the flavors are expanded before the vertices are computed, so FeynRules computes the vertices for the individual flavors separately.

At this stage we have to make a comment about the `Unfold[ ]` function introduced in Section 2.2. If an index type is wrapped in the `Unfold[ ]` command, then FeynRules will always expand over these indices. Moreover, any index which expands field in terms of non-physical states must be wrapped in `Unfold[ ]`. As an example, the adjoint index carried by the weak gauge bosons must always be expanded over, because the physical states are the photon and the $W^{\pm}$ and Z bosons, while for quarks it can be interesting to keep the vertices in a compact form, keeping explicit flavor indices in the vertices. Note that, in order to speed up the computation of the vertices, (some of) the interfaces perform the flavor expansion using the `FlavorExpansion[ ]` command.

The list of Feynman rules can be quite long and it may sometimes be desirable to extract just one or a few vertices. There are several options available that limit the number of vertices computed by FeynRules:

- `MaxParticles -> n` instructs `FeynmanRules` to only derive vertices whose number of external legs does not exceed `n`. The option `MinParticles` works in a similar way.
- `MaxCanonicalDimension -> n` instructs `FeynmanRules` to only derive vertices whose canonical dimension does not exceed `n`. The option `MinCanonicalDimension` works in a similar way.
- `SelectParticles ->{{...}, {...},...}` instructs `FeynmanRules` to only derive the vertices specified in the list. For example, the command:
  `FeynmanRules[ LQCD, SelectParticles ->{{G,G,G}, {G,G,G,G}}]`
  only leads to the derivation of the three and four-point gluon vertices.
- `Contains -> { ... }` instructs `FeynmanRules` to only derive the vertices which involve all the particles indicated in the list.
- `Free -> { ... }` instructs `FeynmanRules` to only derive the vertices which do not involve any of the particles indicated in the list.

FeynmanRules, by default, checks whether the quantum numbers that have been defined in the model file are conserved for each vertex. This check can be turned off by setting the option ConservedQuantumNumbers to False. Alternatively, the argument of this option can be a list containing all the quantum numbers FeynRules should check for conservation.

The Feynman rules constructed are stored internally as a list of vertices where each vertex is a two-component list. The first element enumerates all the particles that enter the vertex while the second one gives the analytical expression for the vertex. We illustrate this for the quark-gluon vertex,

{{{G, 1}, {qbar, 2}, {q, 3}}, $ig_s\,\delta_{f_2 f_3}\,\gamma^{\mu_1}_{s_2 s_3}\,T^{a_1}_{i_2 i_3}$ }

Moreover, each particle is also given by a two-component list, the first element being the name of the particle while the second one the label given to the indices referring to this particle in the analytical expression.

As the list of vertices computed by FeynRules can be quite long for complicated models, the SelectVertices routine can be employed. The options shared with the function FeynmanRules are MaxParticles, MinParticles, SelectParticles, Contains and Free and can be invoked as

```
vertsGluon = SelectVertices[vertsQCD,
                    SelectParticles->{{G,G,G},{G,G,G,G}}];
```

It is sometimes convenient to construct the Feynman rules by parts, for instance, when one splits the QCD Lagrangian into three pieces as in

```
LQCD = LGauge + LQuarks + LGhosts;
```

The Feynman rules can be constructed all at once as in the previous examples, or they can be constructed separately as in:

```
vertsGluon = FeynmanRules[ LGauge ];
vertsQuark = FeynmanRules[ LQuarks ];
vertsQuark = FeynmanRules[ LGhosts ];
```

They can later be merged using the function MergeVertices

```
vertsQCD = MergeVertices[ vertsGluon, vertsQuark, vertsGhosts ];
```

This merges the results obtained for vertsGluon, vertsQuark and vertsGhosts into a single list of vertices. If there are two contributions to the same vertex, they will be combined into one vertex.

The parameters are also an important part of the model and several functions allow to manipulate them. The numerical values of any parameter or function of one or several parameters can be obtained with the use of the `NumericalValue` function, as for example in

```
NumericalValue[ Sin[ cabi ]]
```

where `cabi` is the Cabibbo angle. This returns the numerical value of the sine of the Cabibbo angle.

In the case the user wants to change the value of a subset of external parameters, the function `UpdateParameters` can be used such as in

```
UpdateParameters[ gs -> 0.118 , ee -> 0.33 ]
```

where `gs` and `ee` are the strong and electromagnetic coupling constants, respectively.

In order to write and read the numerical values of the parameters to and from a file, FeynRules comes with the two functions `WriteParameters` and `ReadParameters`. By default, they write and read the parameters to and from the file named `M$ModelName` with a ".pars" appended, but this can be changed with the options `Output` and `Input` as in:

```
WriteParameters[Output -> "parameters.pars"]
ReadParameters[Input -> "parameters.pars"]
```

The routine `WriteParameters` writes out the external and internal parameters (including masses and widths) to the specified file, while the function `ReadParameters` only reads in the external parameters (including masses and widths). This gives the user another way to change the values of the external parameters. The user can modify the values of the parameters included in the parameter file created by `WriteParameters`, and then read them back in using `ReadParameters`. Any changes made to the internal parameters are ignored.

In addition to the native FeynRules parameter files (`.pars`), there is a second way to update numerical values of external parameters. Indeed, FeynRules is equipped with its own reading and writing routine for Les Houches Accord (LHA)-like parameter files [51,52]. For example, issuing the command

```
ReadLHAFile[ Input -> "LHA-file" ]
```

reads the LHA-like parameter file `"LHA-file"` and updates the numerical values of all the external parameters of the model in the current Mathematica

session. Similarly, the command

```
WriteLHAFile[ Output -> "LHA-file" ]
```

writes all the numerical values to file in a LHA-like format.

4.5 Manipulating superspace expressions

The $N = 1$ superspace is defined by supplementing to the ordinary space-time coordinates x^μ a Majorana spinor $(\theta_\alpha, \bar{\theta}^{\dot{\alpha}})$. These extra coordinates are represented in FeynRules by the symbols `theta` and `thetabar`

$$\texttt{theta[alpha]} \leftrightarrow \theta_\alpha \quad \text{and} \quad \texttt{thetabar[alphadot]} \leftrightarrow \bar{\theta}_{\dot{\alpha}}$$

By convention, both spin indices are assumed to be lower indices. The position of the spin indices can be modified by employing the rank-two antisymmetric tensors represented by the objects `Deps` and `Ueps`, which equivalently take as arguments left-handed or right-handed spin indices. For instance, one could implement the following expressions in a Mathematica session as

$$\bar{\theta}^{\dot{\alpha}} = \epsilon^{\dot{\alpha}\dot{\beta}}\bar{\theta}_{\dot{\beta}} \leftrightarrow \texttt{Ueps[alphadot,betadot] thetabar[betadot]}$$
$$\theta_\alpha = \epsilon_{\alpha\beta}\epsilon^{\beta\gamma}\theta_\gamma \leftrightarrow \texttt{Deps[alpha,beta] Ueps[beta,gamma] theta[gamma]}$$

Proper computations in superspace require to keep track, on the one hand, of the position of the spin indices and, on the other hand, of the fermion ordering. To this aim, FeynRules always assumes that an explicit spin index is a lower index. Moreover, fermion ordering information is provided by using the syntax `nc[chain]`, where `chain` stands for any ordered sequence of fermions (with lower spin indices). As a simple example, we show a possible implementation for the scalar product $\bar{\lambda}\cdot\bar{\lambda}'$, where $\bar{\lambda}$ and $\bar{\lambda}'$ are right-handed Weyl fermions,

$$\bar{\lambda}\cdot\bar{\lambda}' \leftrightarrow \texttt{nc[lambdabar[bd], lambdabarprime[ad]] Ueps[bd,ad]}$$

This way of inputting superspace expressions is however highly unpractical for longer expressions. Therefore, FeynRules contains an environment `ncc` similar to the `nc` environment, the difference lying in the fact that all spin indices and ϵ-tensors can be omitted,

$$\bar{\lambda}\cdot\bar{\lambda}' \leftrightarrow \texttt{ncc[lambar, lambarprime]}$$

The `ncc` environment is automatically handled by FeynRules. The result of the command above consists of the same expression, but given in canonical

Table 18: FeynRules commands

`LoadModel[`f1.fr, f2.fr, ...`]`

Loads and initializes the model defined by the model files f1.fr, f2.fr,... .

`FeynmanRules[`$\mathcal{L}$, options`]`

Calculates the Feynman rules associated with the Lagrangian $\mathcal{L}$. The list of available options are given in Table 19.

`SelectVertices[`verts, options`]`

Selects a subset of the vertices contained in verts. The list of available options are given in Table 19.

`MergeVertices[`v1, v2, ...`]`

Merges the vertex lists v1, v2,... into one single list.

`NumericalValue[` f[pars] `]`

Gives the numerical value of f[pars] where f is some function and pars is some set of parameters of the model.

`UpdateParameters[`$p1 \to v1$, $p2 \to v2$,...`]`

Changes the values of the parameters p1,p2,... to v1,v2,....

`WriteParameters[`options`]`

Writes the numerical values of the internal and external parameters (including masses and widths) to a text file whose the filename consists of the value of `M$ModelName` with the extension `.pars` appended, unless otherwise specified by means of the `Output` option.

`ReadParameters[`options`]`

Reads the external parameters from a text file. By default, the text file is named as in `WriteParameters` and must have the same format as the one created by `WriteParameters`. Another input filename can be specified by means of the `Input` option.

`WriteLHAFile[ Output ->` file `]`

Writes the numerical values of all external parameters to the text file file in an LHA-like format.

`ReadLHAFile[ Input ->` file `]`

Reads the numerical values of the external parameters from the LHA-like text file file.

Table 19: `FeynmanRules` and `SelectVertices` options

`ScreenOutput`	Option of `FeynmanRules`. If turned to `False`, the derived Feynman rules do not appear on the screen. The default is `True`.
`FlavorExpand`	Option of `FeynmanRules`. Expands over the flavor indices specified by the list which this option is referring to. If turned to `True`, all flavor indices are expanded.
`ConservedQuantumNumbers`	Option of `FeynmanRules`. Checks whether the quantum numbers specified by the list which this option is referring to are conserved. If `True` (`False`), the conservation of all (no) quantum numbers is checked. The default is `True`.
`MinParticles`	Only vertices involving at least the specified number of external fields are derived.
`MaxParticles`	Only vertices involving at most the specified number of external fields are derived.
`MinCanonicalDimension`	Option of `FeynmanRules`. Only vertices of at least the specified canonical dimension are derived.
`MaxCanonicalDimension`	Option of `FeynmanRules`. Only vertices of at most the specified canonical dimension are derived.
`SelectParticles`	Calculates only the vertices specified in the list which this option is referring to.
`Contains`	Only the vertices which contain at least the particles contained in the list which this option is referring to are derived.
`Free`	Only the vertices which do not contain any of the particles contained in the list which this option is referring to are derived.

form, employing rank-two antisymmetric tensors and two-component spinors with lowered indices, ordered within a `nc` environment.

Another consequence of the mandatory usage of this canonical form is that the MATHEMATICA output associated to a superspace expression is in general difficult to read. To improve the readability, it is possible to force FEYNRULES to form invariant products of spinors and to simplify products of Grassmann variables according to the relations

$$\theta^\alpha\theta^\beta = -\frac{1}{2}\theta\cdot\theta\epsilon^{\alpha\beta} \ , \quad \bar{\theta}^{\dot{\alpha}}\bar{\theta}^{\dot{\beta}} = \frac{1}{2}\bar{\theta}\cdot\bar{\theta}\epsilon^{\dot{\alpha}\dot{\beta}} \quad \text{and} \quad \theta^\alpha\bar{\theta}^{\dot{\alpha}} = \frac{1}{2}\theta\sigma^\mu\bar{\theta}\bar{\sigma}_\mu{}^{\dot{\alpha}\alpha} \ , \tag{4.21}$$

deduced from the Grassmann algebra fulfilled by the variables θ and $\bar{\theta}$. This operation is achieved by means of the function `ToGrassmannBasis` which takes any expression `exp`, depending on the superspace coordinates, as an argument,

```
ToGrassmannBasis[ exp ]
```

First of all, this function rewrites the expression `exp` in terms of a restricted set of scalar products involving Grassmann variables and Pauli matrices, and, second, optimizes the index naming scheme used. This allows to collect and simplify MATHEMATICA expressions that are equal up to the names of contracted indices, such as, *e.g.*,

```
Dot[lambda[al], lambda[al]] -  Dot[lambda[be], lambda[be]]
```

The output of the `ToGrassmannBasis` function consists therefore of expressions very close to their original forms. This method also works on tensorial quantities containing non-contracted spin indices. As a result, the `ToGrassmannBasis` method matches the free indices either to single fermions or to chains containing one fermion and a given number of Pauli matrices. For instance, applying the `ToGrassmannBasis` function on $(\bar{\sigma}^\nu\lambda)_\alpha$, λ being a left-handed Weyl fermion, leads to

```
ToGrassmannBasis[ nc[lambda[b]] * sibar[nu,ad,b] ]
 ⇒ nc[ TensDot2[sibar[nu,ad,b], lambda[b]][up, Right,ad] ]
```

One observes that a chain containing one Pauli matrix and the fermion λ has been formed and stored in a `TensDot2` environment. The structure related to this environment is defined by

```
TensDot2[ chain ][pos, chir, name]
```

where `chain` is a sequence of one Weyl fermion and possibly one or several Pauli matrices, `pos` is the `up` or `down` position of the free spin index, `chir`

Table 20: Superspace conventions in FeynRules

`theta[al]`	The Grassmann variable θ_α.
`thetabar[aldot]`	The Grassmann variable $\bar{\theta}_{\dot{\alpha}}$.
`Ueps[al, be]`	The rank-two antisymmetric tensor with upper indices $\epsilon^{\alpha\beta}$.
`Deps[al, be]`	The rank-two antisymmetric tensor with lower indices $\epsilon_{\alpha\beta}$.
`nc[seq]`	Ordered sequence of fermionic field(s), labeled by `seq`. The indices of the fields are explicitly written.
`ToGrassmannBasis[exp]`	This function rewrites any function `exp` of the superspace coordinates in a human-readable form.
`OptimizeIndex[exp]`	This function optimizes the index naming scheme used in the expression `exp`.
`Tonc[exp]`	This function transforms expressions to their canonical form, with lower spin indices and $\epsilon-$tensors.
`TensDot2[chain][pos,chir,name]`	This environment contains a sequence, labeled by `chain`, of one Weyl fermion and possibly one or several Pauli matrices. The symbols `pos`, `chir` and `name` are the upper (`up`) or lower (`down`) position, the chirality (`Left` or `Right`) and the name of the free index.

indicates if it is a left-handed (`Left`) or right-handed (`Right`) index and `name` denotes the symbol associated to the index.

The optimization of the index naming scheme performed by the `ToGrassmannBasis` function can also be applied directly on any expression, even if not concerned with superspace computations. In this case, it is enough to type

```
OptimizeIndex[ expression ]
```

The basis associated with the output of the `ToGrassmannBasis` function can also be used to input superspace expressions. There are only two rules to follow. First, products of spinors, connected or not by one or several Pauli

matrices, are always written as

```
ferm1[sp1].ferm2[sp2] chain[sp1,sp2]
```

In this expression, we have introduced two Weyl fermions `ferm1` and `ferm2` and the symbol `chain` stands for a series of Pauli matrices linking the spin indices `sp1` and `sp2`. Next, the implementation of expressions involving fermions carrying a free spin index must always employ the `nc` and `TensDot2` environments as described above. The conversion to the canonical form can subsequently be performed by means of the `Tonc` function,

```
Tonc[ expression ]
```

The list of functions and environments useful for manipulating and inputting superspace expressions is collected in Table 20.

4.6 Functions dedicated to superspace computations

The FeynRules package comes with a set of predefined functions dedicated to superspace computations. First, FeynRules offers the possibility to employ the generators Q_α and $\bar{Q}_{\dot{\alpha}}$ of the supersymmetric algebra as well as the superderivatives D_α and $\bar{D}_{\dot{\alpha}}$ to perform superspace computations. In our conventions [54], these operators read

$$\begin{aligned} Q_\alpha &= -i\Big(\frac{\partial}{\partial\theta^\alpha} + i\sigma^\mu{}_{\alpha\dot{\alpha}}\bar{\theta}^{\dot{\alpha}}\partial_\mu\Big) \;, \qquad \bar{Q}_{\dot{\alpha}} = i\Big(\frac{\partial}{\partial\bar{\theta}^{\dot{\alpha}}} + i\theta^\alpha\sigma^\mu{}_{\alpha\dot{\alpha}}\partial_\mu\Big) \;, \\ D_\alpha &= \frac{\partial}{\partial\theta^\alpha} - i\sigma^\mu{}_{\alpha\dot{\alpha}}\bar{\theta}^{\dot{\alpha}}\partial_\mu \;, \qquad \bar{D}_{\dot{\alpha}} = \frac{\partial}{\partial\bar{\theta}^{\dot{\alpha}}} - i\theta^\alpha\sigma^\mu{}_{\alpha\dot{\alpha}}\partial_\mu \;. \end{aligned} \tag{4.22}$$

and can be called in FeynRules by typing

Q_α(expression) $\leftrightarrow$ `QSUSY[expression,alpha]` ,
$\bar{Q}_{\dot{\alpha}}$(expression) $\leftrightarrow$ `QSUSYBar[expression,alphadot]` ,
D_α(expression) $\leftrightarrow$ `DSUSY[expression,alpha]` ,
$\bar{D}_{\dot{\alpha}}$(expression) $\leftrightarrow$ `DSUSYBar[expression,alphadot]` .

The expression on which these operators act must be provided in its canonical form, employing the `nc` environment. Next, the computation of the variation of a quantity Φ under a supersymmetric transformation of parameters $(\varepsilon, \bar{\varepsilon})$ can be achieved by issuing

$$\delta_\varepsilon\Phi = i\Big(\varepsilon\cdot Q + \bar{Q}\cdot\bar{\varepsilon}\Big)\Phi \;. \tag{4.23}$$

However, we recommend the user to employ the shortcut function `DeltaSUSY` rather than a more complicated function of `QSUSY` and `QSUSYBar`,

Table 21: Superspace functionalities

`QSUSY[exp,al]`	Computes the action of the supercharge Q_α on the expression `exp`. This expression must be given under its canonical form and the symbol `al` refers to the spin index of the supercharge.
`QSUSYBar[exp,ad]`	Same as `QSUSY`, but for the supercharge $\bar{Q}_{\dot{\alpha}}$.
`DSUSY[exp,al]`	Same as `QSUSY`, but for the superderivative D_α.
`DSUSYBar[exp,ad]`	Same as `QSUSYBar`, but for the superderivative $\bar{D}_{\dot{\alpha}}$.
`SF2Components[exp]`	Expands the expression `exp` in terms of the Grassmann variables and simplifies the result to a human-readable form. The output consists of a two-component list, the complete series and a new list with all the individual coefficients.

Shortcuts to the individual component fields

The series.	`GrassmannExpand[exp]`
The scalar term.	`ScalarComponent[exp]`
The θ_α term.	`ThetaComponent[exp, a]`
The $\bar{\theta}_{\dot{\alpha}}$ term.	`ThetabarComponent[exp, ad]`
The $\theta\sigma^\mu\bar{\theta}$ term.	`ThetaThetabarComponent[exp, mu]`
The θ^2 term.	`Theta2Component[exp]`
The $\bar{\theta}^2$ term.	`Thetabar2Component[exp]`
The $\theta^2\bar{\theta}_{\dot{\alpha}}$ term.	`Theta2ThetabarComponent[exp, ad]`
The $\bar{\theta}^2\theta_\alpha$ term.	`Thetabar2ThetaComponent[exp, a]`
The $\theta^2\bar{\theta}^2$ term.	`Theta2Thetabar2Component[exp]`

```
DeltaSUSY [ expression , epsilon ]
```

The symbol `expression` stands for any function of superfields and/or component fields while `epsilon` refers to the left-handed piece of the supersymmetric transformation parameters, given without any spin index. There are ten predefined quantities that can play the role of the transformation parameters and that can be employed by typing the symbol `eps`x, x being an integer between zero and nine.

Superfield expressions can always be expanded as a series in terms of the Grassmann variables θ and $\bar{\theta}$ via the `SF2Components` routine,

```
SF2Components [ expression ]
```

This expands in a first step all the chiral and vector superfields appearing in the quantity `expression` in terms of their component fields and the usual space-time coordinates. Next, the function `ToGrassmannBasis` is internally called so that scalar products of Grassmann variables are formed and the expression is further simplified and rendered human-readable. The output of the `SF2Components` function consists of a two-component list of the form

```
{ Full series , List of the nine coefficients }
```

The first element of this list (`Full series`) is the complete series expansion in terms of the Grassmann variables, which could also directly be obtained by typing in a MATHEMATICA session

```
GrassmannExpand [ expression ]
```

The second element of the list above contains a list with the nine coefficients of the series, *i.e.*, the scalar piece independent of the Grassmann variables, followed by the coefficients of the θ_α, $\bar{\theta}_{\dot{\alpha}}$, $\theta\sigma^\mu\bar{\theta}$, $\theta\cdot\theta$, $\bar{\theta}\cdot\bar{\theta}$, $\theta\cdot\theta\bar{\theta}_{\dot{\alpha}}$, $\bar{\theta}\cdot\bar{\theta}\theta_\alpha$ and $\theta\cdot\theta\bar{\theta}\cdot\bar{\theta}$ terms. Each of these could also be obtained using the shortcut functions

```
ScalarComponent [ expression ] ,
ThetaComponent [ expression ] ,
ThetabarComponent [ expression ] ,
ThetaThetabarComponent [ expression ] ,
Theta2Component [ expression ] ,
Thetabar2Component [ expression ] ,
Theta2ThetabarComponent [ expression ] ,
Thetabar2ThetaComponent [ expression ] ,
Theta2Thetabar2Component [ expression ] .
```

(4.24)

where `expression` stands for any superspace expression written in terms of superfields and/or component fields. In order to specify the free spin or Lorentz index related to some of these coefficient, the user has the option to append to the argument of the functions related to fermionic (vectorial) coefficients the name of a spin (Lorentz) index.

All the functions are summarized in Table 21.

4.7 Mass spectrum generation with FEYNRULES

When mixing relations and vacuum expectation values are declared as presented in Section 2.8, the mass matrices included in a Lagrangian denoted by

Lag can be extracted by typing

```
ComputeMassMatrix[ Lag, options ]
```

where options stands for optional arguments. If no option is provided, the function calculates all the mass matrices included in the Lagrangian for which the numerical value of the mixing matrix is unknown. By issuing

```
ComputeMassMatrix[ Lag, Mix ->"l1" ]
```

FeynRules only extracts the mass matrix associated with the mixing relation denoted by the label "l1". Replacing this label by a list of labels leads to the computation of multiple mass matrices simultaneously. Finally, to avoid information to be printed on the screen during the computation of the mass matrices, it is enough to include the optional argument ScreenOutput -> False in the commands above.

The results of the method above can be retrieved through the intuitive functions MassMatrix, GaugeBasis, MassBasis, MixingMatrix, BlockName and MatrixSymbol which all take as argument the label of a mixing relation. A wrapper is also available,

```
MixingSummary [ "l1" ]
```

which sequentially calls all these functions and organizes the output in a human-readable form.

The FeynRules method to extract analytically a mass matrix can also be employed to compute any matrix M defined by

$$\mathcal{L}_{\text{mass}} = \mathcal{B}_2^\dagger \; M \; \mathcal{B}_1 \; , \tag{4.25}$$

where $\mathcal{B}_1$ and $\mathcal{B}_2$ are two field bases. The calculation of the matrix M is achieved by issuing

```
ComputeMassMatrix[ Lag,
    Basis1 -> b1, Basis2 -> b2 ]
```

where the symbols b1 and b2 are associated with the bases $\mathcal{B}_1$ and $\mathcal{B}_2$ given as two lists of fields. In this case, the printing functions introduced above are however not available.

In Section 2 we saw that it is possible to define the width of every particle appearing in the model file. While FeynRules itself does not require the knowledge of the width at any stage during the computation of the Feynman rules, this information is required when outputting a model to a matrix element generator for which a translation interface exists [11] (See Section 6 for more details).

In this section we shortly review the capability of FeynRules to compute all the two-body decays that appear inside a model. For more details we refer to Ref. [49]. At leading-order the two-body partial width of a heavy particle of mass M into two particles of mass m_1 and m_2 is given by

$$\Gamma = \frac{1}{2|M|S} \int \mathrm{d}\Phi_2 \, |\mathcal{M}|^2 \, , \tag{4.26}$$

where S denotes the phase space symmetry factor and $\mathrm{d}\Phi_2$ the usual two-particle phase space measure

$$\mathrm{d}\Phi_2 = (2\pi)^4 \, \delta^{(4)}(p_1 + p_2 - P) \, \frac{\mathrm{d}^4 p_1}{(2\pi)^3} \, \frac{\mathrm{d}^4 p_2}{(2\pi)^3} \, \delta_+(p_1^2 - m_1^2) \, \delta_+(p_2^2 - m_2^2) \, . \tag{4.27}$$

In this expression, $P = (M, \vec{0})$ denotes the four-momentum of the heavy particle in its rest frame and p_1 and p_2 are the momenta of the decay products in the same frame [12]. The matrix element of a two-body decay is a constant, and so the partial width is simply the product of the (constant) matrix element and a phase space factor

$$\Gamma = \frac{\sqrt{\lambda(M^2, m_1^2, m_2^2)} \, |\mathcal{M}|^2}{16 \, \pi \, S \, |M|^3} \, , \tag{4.28}$$

where $\lambda(M^2, m_1^2, m_2^2) = (M^2 - m_1^2 - m_2^2)^2 - 4 m_1^2 m_2^2$. The matrix element in turn is easily obtained by squaring a three-point vertex by means of the polarization tensors of the external particles. While the polarization tensors are model independent and only depend on the spin of the particle, the three-point vertices are computed by FeynRules. We therefore have all the ingredients

[11] An exception to this is that CalcHEP can calculate the widths on the fly. So, the widths are not required for the CalcHEP interface. In addition, one should note that newer versions of MadGraph 5 are also capable of computing widths on the fly.

[12] The absolute value in Eq. (4.26) comes from the fact that in certain BSM models involving Majorana fermions it is possible to choose the phases of the fermion fields such that the mass is made negative.

to evaluate Eq. (4.28). In the rest of this section we describe the functions that allow to compute two-body partial widths from a list of vertices.

Let us assume that we have computed all the vertices associated with a given Lagrangian `L` in the usual way,

```
vertices = FeynmanRules[ L ];
```

We can then immediately compute all the two-body partial widths arising from `vertices` by issuing the command

```
decays = ComputeWidths[ vertices ];
```

The function `ComputeWidths[]` selects all three-point vertices from the list `vertices` that involve at least one massive particle and no ghost fields and/or Goldstone bosons. Next, the vertices are squared by contracting them with the polarization tensors of the external particles and multiplied by the appropriate phase space factors. The output of `ComputeWidths[]` is a list of lists of the form

$$\{\{\phi_1,\, \phi_2,\, \phi_3\},\ \Gamma_{\phi_1 \to \phi_2\, \phi_3}\}\,,$$

where the first element of the list contains the initial state (ϕ_1) and the two decay products (ϕ_2 and ϕ_3) and the second element is the analytic expression for the corresponding partial width.

Some comments are in order about the function `ComputeWidths[]`. First, the output contains the analytic results for all possible cyclic rotations of the external particles $\{\phi_1, \phi_2, \phi_3\}$ with a massive initial state, independently of whether a given decay channel is kinematically allowed, because certain channels might be closed for a certain choice of the external parameters but not for others. Second, we note that the output of `ComputeWidths[]` is also stored internally in the global variable `FR$PartialWidths`. The use of this global variable will become clear below. Every time the function `ComputeWidths[]` is executed, the value of the global variable will be overwritten, unless the option `Save` of `ComputeWidths[]` is set to `False` (the default is `True`).

Besides the main function `ComputeWidth` that allows to compute the two-body decays, FeynRules is equipped with a set of functions that allow to access the output list produced by the main function. For example, the following intuitive commands are available:

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

In the following we only discuss `PartialWidth[]`, the use of the other two functions is similar. FeynRules first checks, based on the numerical values

of the particles defined in the model file, whether the decay $\phi_1 \to \phi_2\,\phi_3$ is kinematically allowed, and if so, it will calculate the corresponding partial width $\Gamma_{\phi_1\to\phi_2\,\phi_3}$ from the list `decay`. The second argument of `PartialWidth[]` is optional and could be omitted. In that case the partial widths stored in the global variable `FR$PartialWidth` will be used by default.

Finally, it can be useful to update the information coming from the original particle declarations by replacing the numerical value of the widths of all particles by the numerical values obtained by the function `TotWidth`, which can be achieved by issuing the command

```
UpdateWidths[ decays ];
```

where, as usual, the argument `decays` is optional. After this command has been issued, the updated numerical results for the widths will be written out by the translation interfaces to matrix element generators.

5 A Simple Example

In this section we present an example of how to implement a model into FeynRules and how to use the code to obtain the Feynman rules. While the model does not have immediate phenomenological relevance, the implementation is complete in the sense that we discuss in detail all the necessary steps. We emphasize that the model does not exploit all the features of FeynRules. For more advanced examples, we recommend to consult the models already implemented [55] or the details given in Refs. [36, 41, 53].

The model under consideration is a variant of the ϕ^4 theory. It displays all the most relevant features the user might encounter when implementing a new model. In particular, it shows how to

- define indices of various types,
- to perform expansions over 'flavor' indices,
- define mixing matrices,
- perform the rotation from the gauge eigenstates to the mass eigenstates (without employing the mass diagonalization package for which we refer to Ref. [43]),
- add gauge interactions to the model.

In the following, we perform the implementation step by step, adding one feature at a time, in order to clearly show how to deal with the various concepts. A user who follows these steps all the way down to the end will achieve a fully functional FeynRules implementation of this model, which may serve as a

starting point for his/her own model implementation.

5.1 The model

The model we consider is a variant of the ϕ^4 theory. More precisely we consider two complex scalar fields $\phi_i(x)$, $i = 1, 2$, interacting through the Lagrangian

$$\mathcal{L}_{scal} = \partial_\mu \phi_i^\dagger \partial^\mu \phi_i - \phi_i^\dagger \mathcal{M}_{ij} \phi_j + (\phi_i^\dagger \lambda_{ij} \phi_j)^2 \,, \tag{5.29}$$

where the mass matrix $\mathcal{M}$ and the coupling matrix λ are assumed to be real and symmetric,

$$\mathcal{M} = \begin{pmatrix} m_1^2 & m_{12}^2/2 \\ m_{12}^2/2 & m_2^2 \end{pmatrix} \quad \text{and} \quad \lambda = \begin{pmatrix} \lambda_{11} & \lambda_{12} \\ \lambda_{12} & \lambda_{22} \end{pmatrix} . \tag{5.30}$$

As the mass matrix is not diagonal, the fields ϕ_i are not mass eigenstates. They are related to the mass eigenstates Φ_i via an orthogonal transformation,

$$\phi_i = U_{ij}\, \Phi_j \ , \tag{5.31}$$

where U denotes the orthogonal matrix that diagonalizes the mass matrix $\mathcal{M}$,

$$U^T \mathcal{M}\, U = \begin{pmatrix} M_1^2 & 0 \\ 0 & M_2^2 \end{pmatrix} . \tag{5.32}$$

In general we cannot diagonalize $\mathcal{M}$ and λ simultaneously, so that after diagonalization, the couplings are explicitly dependent on the mixing. In the expression above, the eigenvalues of the mass matrix are denoted by M_1^2 and M_2^2, and without loss of generality we may assume that $M_1^2 < M_2^2$. The matrix U is then determined by the (normalized) eigenvectors of $\mathcal{M}$. The eigenvalues and eigenvectors can easily be computed in this case using the `Eigenvalues[]` and `Eigenvectors[]` functions of MATHEMATICA [13]. For example, the eigenvalues of $\mathcal{M}$ are given by

$$M_{1,2}^2 = \frac{1}{2}\left(m_1^2 + m_2^2 \mp \sqrt{(m_1^2 - m_2^2)^2 + m_{12}^4} \right) . \tag{5.33}$$

The rotation matrix U can be parametrized by a single angle θ,

$$U = \begin{pmatrix} -\sin\theta & \cos\theta \\ \cos\theta & \sin\theta \end{pmatrix} , \tag{5.34}$$

[13] We stress that, in general, the diagonalization procedure can be very complicated and one needs to resort to external numerical codes to compute the mass eigenvalues and the rotation matrices or use the ASPERGE package (see Section 6.2).

where the angle reads

$$\sin\theta = \frac{m_{12}^2}{\sqrt{m_{12}^4 + \left(m_1^2 - m_2^2 + \sqrt{(m_1^2 - m_2^2)^2 + m_{12}^4}\right)^2}}\,. \tag{5.35}$$

We now describe how this model can be implemented into FeynRules.

5.2 Preparation of the model file – model information

Each FeynRules model file begins by summarizing the model information, which acts as its electronic signature. The user can give his/her model a name (a string) and include additional information on the model, such as contact details of the author(s) of the model file, references to the literature and the version number of the implementation of the model. We emphasize that, although this information is optional, it can be useful to keep track of modifications, and we strongly encourage users to include it for every implementation. In our case, the model information could be entered as follows:

```
M$ModelName = "Example_Model";

M$Information = {
  Authors      -> {"A. Alloul", "N.D. Christensen", "C. Degrande",
                   "C. Duhr", "B. Fuks"},
  Institutions -> {"IPHC / U. Strasbourg", "U. Pittsburgh",
                   "U. Illinois", "ETH Zurich", "CERN"},
  Emails       -> {"adam.alloul@iphc.cnrs.fr", "neilc@pitt.edu",
                   "cdegrand@illinois.edu",
                   "duhrc@itp.phys.ethz.ch", "fuks@cern.ch"},
  Date         -> "April 1st, 2013",
  References   -> {"The FeynRules Manual"}
};
```

5.3 Preparation of the model file – index declarations

Although the model information is optional and may or may not be included in the front matter of a model file, the declaration of all types of indices that appear in the model is mandatory. In our model, there is only one type of index, the index labeling the scalar field ϕ_i, ranging from 1 to 2. It is therefore sufficient to include, at the beginning of the model file,

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

The first line defines an index of type `Scalar` that takes values in the range `{1,2}`. Note that the name `Scalar` for the index can be chosen freely as long as it does not conflict with existing symbols used in the same MATHEMATICA session. The second line instructs FEYNRULES that indices of type scalar should be printed as symbols starting by the letter i.

5.4 Declaration of the objects – parameters

We now describe the declaration of all the parameters that appear inside the model. The Lagrangian of Eq. (5.29) depends on six free real parameters – three for each real symmetric matrix. In addition, in Eqs. (5.33) and (5.35), we have defined the eigenvalues and the mixing angle as functions of the three mass parameters appearing in the Lagrangian. We therefore need to define nine different parameters in the FEYNRULES model file, out of which only six are independent. Note that there is a freedom in how one chooses the independent parameters. Here we follow the convention that the independent parameters are those that appear in the Lagrangian of Eq. (5.29).

We start by implementing the independent, or external, parameters. This can be done as follows,

```
M$Parameters = {

   lam == {
        ParameterType    -> External,
        ComplexParameter -> False,
        Indices          -> {Index[Scalar], Index[Scalar]},
        Value            -> {lam[1,1] -> 0.9,
                             lam[1,2] -> 0.1,
                             lam[2,1] -> 0.1,
                             lam[2,2] -> 0.9},
        Description  -> "Scalar quartic coupling matrix"
        },

    MM == {
        ParameterType    -> External,
        ComplexParameter -> False,
        Indices          -> {Index[Scalar], Index[Scalar]},
        Value            -> {MM[1,1] -> 100^2,
                             MM[1,2] ->  10^2/2,
                             MM[2,1] ->  10^2/2,
                             MM[2,2] ->  200^2},
        Description -> "Mass matrix"
```

```
        }
  };
```

Let us make some comments about the declaration of the external parameters. First, the matrices $\mathcal{M}$ and λ are implemented as parameters `MM` and `lam`, each of them carrying two indices of type `Scalar`. The values of the components of the matrix must be given one by one. Finally, the option `Description`, whose value is a string describing the parameter, is purely optional and could have been omitted.

The implementation of the internal parameters is similar to the case of the external ones, and we append to `M$Parameters` the following elements,

```
M1 == {
   ParameterType    -> Internal,
   ComplexParameter -> False,
   Value            -> Sqrt[1/2 (MM[1, 1] + MM[2, 2] -
                       Sqrt[(MM[1,1]-MM[2,2])^2+4 MM[1,2]^2])],
   Description      -> "Small mass eigenvalue"
   },

M2 == {
   ParameterType    -> Internal,
   ComplexParameter -> False,
   Value            -> Sqrt[1/2 (MM[1, 1] + MM[2, 2] +
                       Sqrt[(MM[1,1]-MM[2,2])^2+4 MM[1,2]^2])],
   Description      -> "Large mass eigenvalue"
   },

sinth == {
   ParameterType    -> Internal,
   ComplexParameter -> False,
   Value            -> 2 MM[1,2]/Sqrt[4 MM[1,2]^2 + (MM[1,1]-
        MM[2,2] + Sqrt[(MM[1,1]-MM[2,2])^2+ 4 MM[1,2]^2])^2],
   Description      -> "Sine of the mixing angle"
   }
```

These declarations are exactly the same as for the external parameters, except that the `Value` options now refer to the algebraic relations of Eqs. (5.33) and (5.35). Finally, we also need to implement the rotation matrix U by appending to `M$Parameters`,

```
UU == {
   ParameterType    -> Internal,
```

```
  ComplexParameter -> False,
  Indices          -> {Index[Scalar], Index[Scalar]},
  Value            -> {UU[1,1] -> -sinth,
                       UU[1,2] -> Sqrt[1-sinth^2],
                       UU[2,1] -> Sqrt[1-sinth^2],
                       UU[2,2] -> sinth},
   Description     -> "Mixing matrix"
   }
```

5.5 Declaration of the objects – fields

Next we turn to the declaration of the fields that appear in the Lagrangian. We need to declare separately the gauge eigenstates ϕ_i and the mass eigenstates Φ_i. Then, we define a replacement rule that instructs FeynRules how to perform the rotation from the gauge to the mass basis.

We start by defining the mass eigenstates by including in `M$ClassesDescription`,

```
S[1] == {
    ClassName     -> Phi,
    ClassMembers  -> {Phi1, Phi2},
    SelfConjugate -> False,
    Indices       -> {Index[Scalar]},
    FlavorIndex   -> Scalar,
    Mass          -> {{M1, Internal}, {M2, Internal}}
    }
```

This defines a scalar field `Phi` carrying an index of type `Scalar`. The symbol `Phibar` for the corresponding conjugate field is automatically defined. The `FlavorIndex` option specifies that indices of the type `Scalar` label the elements of the list `ClassMembers`[14]. Since the masses of the different class member have already been defined previously as internal parameters, the values of the masses of the class members have been set to `Internal`.

Next, the declaration of the gauge eigenstates is performed as follows

```
S[2] == {
    ClassName     -> phi,
    ClassMembers  -> {phi1, phi2},
```

[14] This information seems redundant at this stage. It is however mandatory because a field may carry more than a single index (see Section 5.7). In that case, the `FlavorIndex` option singles out the one type of index in the `Indices` list which labels the `ClassMembers`.

```
    SelfConjugate -> False,
    Indices       -> {Index[Scalar]},
    FlavorIndex   -> Scalar,
    Unphysical    -> True,
    Definitions   -> {phi[i_] :> Module[{j}, UU[i,j] Phi[j]]}
    }
```

Compared to the mass eigenstates, the `Mass` option has been replaced by a pair of options. First, the option `Unphysical -> True` identifies gauge eigenstates. It has no other effect than to instruct FeynRules not to output this field to any Feynman diagram calculators (which in general work at the level of mass eigenstates). Secondly, we use the `Definitions` option to define a replacement rule that rotates the gauge basis to the mass basis. This teaches FeynRules to replace every occurrence of `phi[i_]`, where `i_` is a Mathematica pattern representing some arbitrary index, by the expression `UU[i,j] Phi[j]`. Note the appearance of the environment `Module` in the right-hand side of the replacement rule. A `Module` is an internal Mathematica command that takes two arguments

(1) a list of symbols `{a,b,c,...}`,
(2) an expression `X`.

The `Module` evaluates `X` and replaces every occurrence of the symbols in the list `{a,b,c,...}` by new symbols `a$`n, `b$`n, `c$`n, ..., where n is an integer such that the resulting new symbols are unique. In this way, it is ensured that the contracted index `j` introduced each time the module is called is unique and does not conflict with any other symbol of the same name.

5.6 The Lagrangian and the Feynman rules

After we have defined the parameters and the fields of the model, we can write down its Lagrangian. It is sufficient to translate Eq. (5.29) in terms of the symbols introduced in the previous subsections,

```
Lscal = del[phibar[i], mu] del[phi[i], mu] -
        phibar[i] MM[i,j] phi[j] +
        (phibar[i1] lam[i1, i2] phi[i2]) *
        (phibar[j1] lam[j1, j2] phi[j2])
```

Let us note that we cannot write `(phibar[i1] lam[i1, i2] phi[i2])^2` because this expression is ambiguous,

$$(\phi_i \lambda_{ij} \phi_j)^2 \neq \phi_i^2 \, \lambda_{ij}^2 \phi_j^2 \,. \tag{5.36}$$

The first thing we should do is check that our mass matrix diagonalization worked properly. After loading the FeynRules package and the model, as presented in Sections 4.1 and 4.2, we issue the command

```
CheckMassSpectrum[ Lscal ]
```

If everything was done properly, FeynRules will state that all mass terms are diagonal and will check their numerical values against the numerical values for `M1` and `M2`, which were set as the masses of `Phi1` and `Phi2`, respectively.

We have now all the ingredients to compute the Feynman rules of the model. Loading the FeynRules package and the model, as presented in Sections 4.1 and 4.2, issuing then the command

```
FeynmanRules[ Lscal ]
```

returns the vertex (all particles are considered in-going)

$$\Phi_{i_1}, \Phi_{i_2}, \Phi^\dagger_{i_3}, \Phi^\dagger_{i_4}: \quad i\lambda_{jk}\lambda_{lm}\left(U_{ki_1}\,U_{mi_2}+U_{mi_1}\,U_{ki_2}\right)\left(U_{ji_3}\,U_{li_4}+U_{ji_4}\,U_{li_3}\right)\,.$$

By default the vertex is outputted for particle classes, *i.e.*, all indices labeling the members are symbolic. It is possible to obtain the Feynman rules for the individual class members by applying the function `FlavorExpansion[]` to the output of `FeynmanRules[]`, or by typing

```
FeynmanRules[ Lscal, FlavorExpand -> True ]
```

5.7 Extending the model – Gauge interactions

So far our model does not contain any gauge interactions. In the following, we describe how we can easily extend the model to include such interactions. To be more precise, we assume that the scalar fields ϕ_i transform in the fundamental representation of some $SU(3)$ gauge group. The Lagrangian of this model is then extended from Eq. (5.29) by including the kinetic term for the gauge boson, denoted by G^a_μ, and by replacing the ordinary space-time derivative by the covariant derivative

$$D_\mu = \partial_\mu - i\,g_s\,T^a\,G^a_\mu\,, \tag{5.37}$$

where g_s is the gauge coupling and T^a denotes the generators of the fundamental representation of $SU(3)$. The BRST-invariant Lagrangian then reads, in Feynman gauge,

$$\begin{aligned}\mathcal{L} = &-\frac{1}{4}F^a_{\mu\nu}F^{\mu\nu}_a+\partial_\mu\bar{c}^a D^\mu c^a-\frac{1}{2}(\partial^\mu G^a_\mu)^2\\ &+D_\mu\phi^\dagger_{ik}D^\mu\phi_{ik}-\phi^\dagger_{ik}\mathcal{M}_{ij}\phi_{jk}+(\phi^\dagger_{ik}\lambda_{ij}\phi_{jk})^2\,,\end{aligned} \tag{5.38}$$

where k denotes the $SU(3)$ fundamental index carried by the field, and where c is the ghost field associated with the gauge boson. We now illustrate that gauging a model in FeynRules is not much more complicated than the procedure sketched above.

We start by defining new indices which label the fundamental and adjoint representations of the gauge group, at the beginning of the model file,

```
IndexRange[ Index[Colour] ] = Range[3];
IndexStyle[ Colour, k ];

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

where `Gluon` and `Colour` represent adjoint and fundamental $SU(3)$ indices, respectively. We define a new external parameter `gs` in `M$Parameters` corresponding to the gauge coupling,

```
gs == {
   ParameterType    -> External,
   ComplexParameter -> False,
   Value            -> 1.22
   }
```

Next we turn to the declaration of the fields. First we need to extend the definition of the scalar field to include a gauge index of type `Colour` in the list of indices carried by the field. In other words, we replace the right-hand side of the option `Indices` by `{Index[Scalar], Index[Colour]}`. We also need to extend the `Definitions` option to include the second field index,

```
Definition -> {phi[i_,n_] :> Module[{j}, U[i,j] Phi[j,n]]}
```

Finally, we also need to include the declarations of the gauge boson field G and the ghost field c in `M$ClassesDescription`,

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

 U[1] == {
    ClassName       -> ghG,
    SelfConjugate   -> False,
    Indices         -> {Index[Gluon]},
```

```
    Ghost           -> G,
    QuantumNumbers  -> {GhostNumber -> 1},
    Mass            -> 0
  }
```

At this stage we have defined the indices, the parameters and the particles relating to the gauge group. In addition, we also need to introduce group theory objects such as structure constants and representation matrices. This is achieved by declaring a new instance of the gauge group class in `M$GaugeGroups`,

```
M$GaugeGroups = {

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

};
```

Once the gauge group is declared, we can simply write down the Lagrangian using the functions `FS` and `DC` for the gauge field strength tensors and the covariant derivatives,

```
L = -1/4 FS[G, mu, nu, a] FS[G, mu, nu, a] +
    del[ghGbar[a], mu] DC[ghG[a], mu] -
    1/2 (del[G[mu, a], mu]) (del[G[nu, a], nu])+
    DC[phibar[i,k], mu] DC[phi[i,k], mu] -
    phibar[i,k] MM[i,j] phi[j,k] +
    (phibar[i1,k] lam[i1, i2] phi[i2,k]) *
    (phibar[j1,l] lam[j1, j2] phi[j2,l])
```

Issuing the command `FeynmanRules[ L ]` now returns all the interaction vertices of the model, including the gauge interactions of the gluon field and the scalar fields.

5.8 Implementing the mixing declaration

Another possible way to implement this model into FeynRules is to let it handle automatically the mixings among the gauge eigenstates. In order to do so, one should remove the option `Definitions` from the declaration of the scalar field `S[2]` and, instead, provide the list `M$MixingsDescription` with

the right options. This alternative allows FeynRules to extract automatically the mass matrix $\mathcal{M}$ and further export it to the ASperGe package for a numerical diagonalization. In practice the `M$ClassesDescription` should only comprise the following two declarations

```
S[1] == {
    ClassName     -> Phi,
    ClassMembers  -> {Phi1, Phi2},
    SelfConjugate -> False,
    Indices       -> {Index[Scalar]},
    FlavorIndex   -> Scalar,
    Mass          -> {{M1, External}, {M2, External}},
    PDG           -> {9000001, 9000002}
    };
S[2] == {
    ClassName     -> phi,
    ClassMembers  -> {phi1, phi2},
    SelfConjugate -> False,
    Indices       -> {Index[Scalar]},
    FlavorIndex   -> Scalar,
    Unphysical    -> True
    }
```

while the list `M$MixingsDescription` should be created with the following lines

```
Mix["sca"] == {
  GaugeBasis -> {phi[1], phi[2]},
  MassBasis -> {Phi[1], Phi[2]},
  MixingMatrix -> UU,
  BlockName -> UMIX,
  Inverse -> True}
```

In the above lines, we have set the masses `M1` and `M2` as external as they are to be calculated by the ASperGe code and added a PDG code for each of the mass eigenstates. We gave this mixing the label `"sca"`, the mixing matrix `UU` and the SLHA-block `UMIX`. To be compliant with both FeynRules conventions and equation (5.39), that is

$$\phi_i = U_{ij}\, \Phi_j \ , \tag{5.39}$$

we have added the option `Inverse` with the attribute `True`.

In the parameters declaration, one should remove the entries `M1, M2, sinth` and `UU` as they correspond to quantities that will be calculated by the AsperGe code. Finally, as none of the scalar fields introduced here acquires a

vacuum expectation value, one may skip the declaration of the corresponding variable `M$vevs`.

Now that the declaration of the model fits the requirements of the mass diagonalization package, one can issue the commands

```
WriteASperGe[L];
RunASperGe[];
```

in order to generate the source code for ASperGe, diagonalize the mass matrix `UU` and update the values for `M1`, `M2` and all the components of the mixing matrix `UU`.

6 Interfaces

So far we have only discussed how to implement a model into FeynRules and how to compute the associated interaction vertices. After the Feynman rules have been obtained, the user is typically interested in the phenomenology of the model, which requires the evaluation of Feynman diagrams. There are many tools that allow to evaluate Feynman diagrams automatically. While these tools are in general very flexible and allow the user the generate, at least in principle, Feynman diagrams for any model satisfying basic quantum field theory requirements, most of the tools only have a limited number of new physics models implemented. Implementing a new model into any of these tools generally requires the user to implement the interaction vertices one at the time. In addition, each Feynman diagram generator has its own format, making the task of implementing a new model into a Feynman diagram generator tedious and error-prone.

FeynRules is equipped with interfaces to various Feynman diagram generators which allow to export the Feynman rules to any of these tools (for which an interface exists) in the form of a set of text files specific to each code. Currently the following interfaces exist,

- CalcHep/CompHep,
- FeynArts/FormCalc,
- Sherpa,
- UFO (Universal FeynRules output),
- Whizard/Omega.

The UFO format is a generic model format that stores model information in an abstract way in the form of Python objects [44]. More information is

provided in Section 6.7.

All interfaces are invoked with commands like

`WriteXOutput[`$\mathcal{L}_1$,$\mathcal{L}_2$,..., options`]`

where X is replaced with a particular label referring to the interface name. The main limitations of the interfaces rely in the fact that they can only be used to implement particles and vertices which are natively supported by the corresponding Feynman diagram generator. As an example, each generator makes implicit assumptions on the spins or color representations of the particles present in the model, and/or on the form or the dimension of the interaction vertices. As a consequence, only those models that are compliant with all those constraints are supported by a given tool. The FeynRules interfaces are tailored to the different Feynman diagram generators, and they therefore allow to check on a case by case basis whether or not a given model is compliant with a given tool. For example, if an interface detects that a given interaction vertex is not compliant, the vertex is discarded and a warning is printed on the screen. The user then has the possibility to switch to a different Feynman diagram generator for which the restriction is not present by using a different interface without having to change his/her FeynRules implementation.

In this section we give a brief account of how to run the FeynRules interfaces and we discuss their features and limitations. While some of the interfaces have been the subject of separate publications [44, 47], we include a summary on how to run all the interfaces for the sake of completeness.

6.1 Conventions

While FeynRules itself is completely agnostic of the underlying model and does not make any *a priori* assumptions on the form of the particles and parameters that are defined in the model file, Feynman diagrams generators usually have some information, in particular about the Standard Model, hard-coded. For example, most Feynman diagram generators have the running of the strong and/or weak coupling constants implemented, which allows one to use the value of α_s specific to a given process. In order for this to work, the value of α_s must be stored in a variable whose name is hard-coded into the code. Similarly, many programs have color implemented in an implicit way and/or perform the color algebra internally. As a consequence, the FeynRules interfaces must communicate information about the Standard Model parameters and gauge groups to the Feynman diagram generators in a specific format. For this to work properly, the user has to follow a set of conventions when writing a model file which are detailed in the rest of this section.

6.1.1 Name restrictions

While MATHEMATICA, and hence FEYNRULES, are case sensitive, many Feynman diagram calculators are not. Consequently, the model builders should avoid to use names that only differ by case. In addition, FEYNRULES allows the use of Greek letters to define the names of parameters and fields, which cannot be exported to the Feynman diagram calculators. However, the user can change the name of a parameter or field to be outputted by the interfaces by using the parameter option `ParameterName`. This allows to output a name matching the requirements of the Feynman diagram calculators. Similar options, called `ParticleName` and `AntiParticleName`, are available for the particle classes. They allow the user to specify the string (or list of strings) that should be used in the external programs. Finally, the TEX-form associated with these names can also be specified via the options `TeXParticleName` and `TeXAntiParticleName` of the particle class and `TeX` of the parameter class. An example follows:

```
ParticleName     -> {"ne", "nm", "nt"},
AntiParticleName -> {"ne~", "nm~", "nt~"}
```

A more detailed name can also be attached, as a string or list of strings, to any particle, by means of the attribute `FullName` of the particle class, such as in

```
FullName -> "Photon"
```

6.1.2 Particle Data Group numbering scheme (PDG) codes

Many programs use the Particle Data Group numbering scheme [56] to refer to the particles inside the code. In addition, many codes have information on the particles and their quantum numbers hard-coded. It is therefore crucial that new models respect the PDG numbering scheme whenever possible and we strongly encourage the users to use the existing PDG codes.

FEYNRULES allows the user to assign PDG codes to all the particles in the model file. For example, the PDG codes for the up-type quarks can be defined by adding the following option to the up-type particle class (which has the u, c and t quarks as members),

```
PDG -> {2,4,6}
```

If a particle class has only one member, the curly brackets on the right-hand side may be omitted. If a user does not define a PDG code for a particle, then FEYNRULES automatically assigns to it a PDG code starting from 9000001.

6.1.3 Decay widths

When computing Feynman diagrams it is in general important, already at tree-level, to include the total width into the propagator of a massive particle in order to avoid singularities in the phase space integration. For this reason most Monte Carlo tools take the total width of a particle as a numerical input for tree-level computations. FeynRules allows the user to include the numerical value for the width into the model file via the option `Width` (For more details see Section 2.4). If done in this way, the numerical values for the widths of all the particles are transmitted to the Monte Carlo codes.

The widths of all the particles are, however, in general unknown at the moment of the implementation in FeynRules, because the evaluation of the (tree-level) width of a particle requires the evaluation of Feynman diagrams. There are four different ways the user can include the widths for the Feynman diagram calculators:

(1) He/she may use the function `TotWidth[particle]` introduced in Section 4.8 to compute the total two-body decay width of particle.
(2) He/she may export the model to a Feynman diagram calculator via one of the interfaces without including the widths of the particles [15] and then use the tool itself to compute all particle partial widths in all possible decay channels. He/she can then update the model file with the corresponding numerical values.
(3) Some Feynman diagram calculators offer the possibility to evaluate the widths of the particles on the fly without the need of specifying their value in the model file. For more information we refer to the documentation of the various tools.
(4) There are dedicated tools that compute the widths and the branching fractions for specific new physics models (*e.g.*, the Minimal Supersymmetric Standard Model). If such a tool exists for the model under consideration, the user may simply want to update his model by reading in the output files of any of these codes.

In all cases we want to stress that the branching fractions and the widths are highly benchmark-dependent and need to be reevaluated every time the numerical value of an external parameter is changed.

6.1.4 Parameter input

In Section 2.3 we have shown how to assign a numerical value to an external parameter using the `Value` option. These values are then transmitted to the

[15] If no numerical value is given in the model, the widths are automatically set to a default value of 1 GeV.

Feynman diagram calculators when running the FeynRules interfaces. In practice, one often wants to scan over the parameter space of a new model. It is however highly inefficient to rerun Mathematica every time the user wants to change the numerical value of some external parameter. Rather, it is more convenient to update the numerical parameters at runtime in the Feynman diagram calculator.

Many Monte Carlo tools rely on a Les Houches (LH)-like format to input the numerical values of the external parameters. This format is inspired by the Supersymmetry-Les Houches-Accord (SLHA) [51,52] which defines a standard format for the numerical parameters in the Minimal Supersymmetric Standard Model and certain extensions thereof. In this format numerical values are grouped into certain blocks, and each parameter is identified inside its own block by one or more integer numbers, called counters.

FeynRules internally stores the numerical values of all external parameters in a form which closely follows the SLHA format. In particular, each external parameter is assigned to a block and counter, which can be specified using the `BlockName` and `OrderBlock` options of the parameter class. For example, the definition of the strong coupling constant can be assigned to the third entry of the block named `SMINPUTS` by adding the following options to the definition of the parameter

```
BlockName        -> SMINPUTS,
OrderBlock       -> 3,
```

The value of `BlockName` can be an arbitrary Mathematica symbol, while the value of `OrderBlock` can be either an integer or a list of integers. The `BlockName` and `OrderBlock` attributes are optional. If omitted, the parameter are assigned automatically to the block `FRBlock`. Moreover, masses and widths are automatically assigned to the blocks `MASS` and `DECAY`, the identifier inside the block being the PDG code of the particle. By convention, the entries of a matrix defines a block on its own, the identifiers inside the block being the position inside the matrix.

By convention, all the parameters defined inside a LH-like format are real. Complex external parameters have then to be implemented by splitting them into their real and imaginary parts. As an illustration, we consider a complex external parameter $a = a_R + ia_I$, where a_R and a_I are real. This parameter is thus implemented into the LH-like format by defining a as internal and a_R and a_I as external.

6.1.5 Definition of Standard Model parameters and gauge groups

The parameters and the gauge groups of the Standard Model have a special significance in most Monte Carlo codes. In particular they must be implemented using certain conventions and giving specific names to certain variables, in order to ensure that, *e.g.*, the strong and/or electroweak couplings are run correctly and the color algebra is performed correctly. As a consequence, despite the fact that FeynRules is generic and does not distinguish between the Standard Model and non-Standard Model parameters when computing the Feynman rules, the interfaces have an explicit dependence on them. The Standard Model input parameters and gauge groups must therefore be implemented into a FeynRules model file following certain conventions.

First, the strong coupling constant g_s and its square over 4π should be declared as in the following example,

```
aS    == {
   ParameterType    -> External,
   BlockName        -> SMINPUTS,
   OrderBlock       -> 3,
   Value            -> 0.1184,
   InteractionOrder -> {QCD,2},
   Description      -> "Strong coupling constant at the Z pole"
  },

gs == {
   ParameterType    -> Internal,
   Value            -> Sqrt[4 Pi aS],
   InteractionOrder -> {QCD,1},
   ParameterName    -> G,
   Description      -> "Strong coupling constant at the Z pole"
  },
```

We note that α_s is implemented as an external parameter while g_s is defined as an internal parameter derived from α_s. Similarly, the electroweak coupling is defined as follows,

```
aEWM1 == {
   ParameterType    -> External,
   BlockName        -> SMINPUTS,
   OrderBlock       -> 1,
   Value            -> 127.9,
   InteractionOrder -> {QED,-2},
   Description      -> "Inverse of the EW coupling constant at the Z pole"
  },
```

```
Gf == {
   ParameterType    -> External,
   BlockName        -> SMINPUTS,
   OrderBlock       -> 2,
   Value            -> 1.16637*^-5,
   InteractionOrder -> {QED,2},
   Description      -> "Fermi constant"
  },

aEW == {
   ParameterType    -> Internal,
   Value            -> 1/aEWM1,
   InteractionOrder -> {QED,2},
   Description      -> "Electroweak coupling constant"
  },

ee == {
   ParameterType    -> Internal,
   Value            -> Sqrt[4 Pi aEW],
   InteractionOrder -> {QED,1},
   Description      -> "Electric coupling constant"
  }
```

The external parameters in the electroweak sector are the inverse of the electromagnetic coupling at the Z scale, $\alpha_{EW}(M_Z)^{-1}$ and the Fermi constant G_F. The reason for choosing $\alpha_{EW}(M_Z)^{-1}$ as the external input parameter, and not $\alpha_{EW}(M_Z)$ itself is only to be compliant with the SLHA. The electromagnetic coupling at the Z scale and the electric charge e are declared as internal parameters. Finally, it is also encouraged to choose the mass of the Z boson as an external parameter, still following the SLHA conventions.

The QCD gauge group has a special status in FeynRules as it defines quantities that have to be dealt with in a special way by the interfaces. This ensures, *e.g.*, that the color algebra is correctly performed by the Feynman diagram calculators. The correct definition of the QCD gauge groups is

```
SU3C == {
    Abelian           -> False,
    CouplingConstant  -> gs,
    GaugeBoson        -> G,
    StructureConstant -> f,
    Representations   -> {{T, Colour},
                          {T6, Sextet}},
    SymmetricTensor   -> dSUN
```

```
}
```

This definition requires the gluon field, defined in `M$ClassesDescription`, to be called `G`. The gluon field carries an index of type `Gluon`, which represents the adjoint representation of QCD. So far, there is, at least to our knowledge, no Monte Carlo code for which an interface exists that can deal with representation other than **1**, **3**, **6** and **8** (and their complex conjugate representations, if applicable). In the example above, we have defined two representations of the QCD gauge group, the fundamental and sextet representations, which act via the generators `T` and `T6` on indices of type `Colour` and `Sextet`, respectively. A full list of the group theoretical object of QCD that can be used to construct a Lagrangian can be found in Table 22. Furthermore, the indices labeling the different representations should be defined at the beginning of the model file as in

```
IndexRange[ Index[ Colour ] ] = Range[3];
IndexRange[ Index[ Sextet ] ] = Range[6];
IndexRange[ Index[ Gluon ] ] = Range[8];
```

The function `NoUnfold[]` might be added in the right-hand side if desired. This is only used by the FeynArts interface.

In addition to the QCD charges of the particles, some Monte Carlo programs also use the information on the electric charge of a particle. The electric charge should be defined in the `QuantumNumbers` option of the particle class as `Q` to ensure that the information is correctly transmitted to the matrix element generators by the FeynRules interfaces.

6.1.6 Switching between unitary and Feynman gauge

Some Monte Carlo programs allow the user to generate QCD processes both in unitary and in Feynman gauge. Having a model that can run in both gauges can be a powerful way to check that the implementation is gauge invariant by generating matrix elements in both gauges. In addition, some codes run faster in one gauge than in the other. It is therefore desirable to implement a model both in unitary and Feynman gauge whenever possible. At this stage, the user has to implement by hand all the terms in the Lagrangian related to the gauge fixing procedure. In practice, every model implemented in Feynman gauge can be transformed to unitary gauge by removing all the vertices that involve ghost fields and/or Goldstone bosons. It can therefore be useful to add a Boolean variable `$FeynmanGauge` into the model, which, if set to `False`, removes all the terms from the Lagrangian which depend on ghost fields and Goldstone bosons. Furthermore, some interfaces use this variable to communicate to the matrix element generator whether or not the model can be run

Table 22: Group-theoretical objects

`T[a,i,j]`	Generators of the fundamental representation, T^a_{ij}.
`T6[a,m,n]`	Generators of the sextet representation, $T^a_{6,ij}$.
`f[a,b,c]`	$SU(3)$ structure constants, f^{abc}.
`dSUN[a,b,c]`	Totally symmetric tensor d^{abc}.
`Eps[i,j,k]`	Totally antisymmetric tensor ϵ_{ijk} connecting three (anti)triplet indices.
`K6[m,i,j]`	Clebsch-Gordan coefficient connecting a sextet and two anti-triplets, considered incoming.
`K6bar[m,i,j]`	Clebsch-Gordan coefficient connecting a anti-sextet and two triplets, considered incoming.

in Feynman gauge. For these reasons, we strongly recommend the use of the variable `$FeynmanGauge` in each model implementation. For an illustration of how to use it, we recommend to look at the implementation of the Standard Model shipped with this package.

6.1.7 Interaction orders

Perturbative expansions allow one to compute an amplitude at a given order in each of the coupling constants. Some programs use this expansion to order contributions to a process according to their relative importance and eventually only keep the largest one[16]. For example, $u\bar{u} \to t\bar{t}$ has three tree-level diagrams, corresponding to the gluon, the Z boson and the photon in the s-channel. However, the contribution of the electroweak bosons is only a small correction to the leading strong contribution due to the hierarchy between the strong and the electroweak coupling constants. MadGraph will only compute the strong contributions by default, *i.e.*, the amplitude proportional to $g_s^2 e^0$ and not the one proportional to $g_s^0 e^2$. The largest allowed power of each coupling can also be specified, which allows to keep the electroweak diagrams as well. As a consequence, the program needs to know how many powers of g_s and e are present in each vertex to compute the dependence of the diagrams. These powers are computed in FeynRules from the analytic expressions of the vertices and the dependence in the coupling constants of each parameter. The latter is given in the parameter description through the `InteractionOrder` option. For example, `QCD` is defined as the power of g_s.

[16] Currently, only the UFO interface uses this feature.

Therefore α_s has

```
InteractionOrder -> {QCD,2}
```

because it is proportional to g_s^2. The second order defined for the Standard Model is `QED` which counts the powers of e. Both electroweak coupling constants g and g' have

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

while α_{EM} has `InteractionOrder -> {QED,2}`. In addition, model builders can define new orders for other couplings similarly.

The default behavior of the Monte Carlo generator comes from the knowledge of the hierarchy of couplings. This is defined in the FeynRules model as an additional list :

```
M$InteractionOrderHierarchy = {
  {QCD,1},
  {QED,2}
}
```

The numbers associated with the couplings give their relative importance. In the above example, one power of e is considered equivalent to two powers of g_s. If a new vector connects the up and top quarks with a new physics coupling g_{NP}, one can possibly set g_{NP} order to `InteractionOrder -> {NP,1}` and the hierarchy can be defined as

```
M$InteractionOrderHierarchy = {
  {QCD,1},
  {NP,1},
  {QED,2}
}
```

as for the strong interactions to avoid that new physics is removed by default. Finally, the user can also define a maximum value for the power of a coupling to be allowed in each diagram,

```
M$InteractionOrderLimit = {
  {NP,2}
}
```

This is used to indicate to the generator that the largest power g_{NP} appearing in an amplitude is two. For example, the interaction of the gluon with the Higgs boson through a top loop can be added as an effective operator in the large top mass limit. The production by gluon fusion of a single Higgs

boson can be simulated in this way by tree-level event generators. However, this vertex cannot be used twice for double Higgs boson production by gluon fusion because it does not include all the contribution at this order in α_s (box-type diagrams with a top loop are missing).

6.1.8 Drawing Feynman diagrams

Some of the programs interfaced to FeynRules allow to draw Feynman diagrams associated with a given process. The way the propagators are drawn can be defined at the FeynRules level by means of the options `PropagatorLabel`, `PropagatorType` and `PropagatorArrow` of the particle class. The first of these attributes allows to modify the label attached to an internal line of a Feynman diagram and takes a string (or a list of strings if one has several class members) as an argument. The second attribute refers to the type of line to employ when drawing the propagator. By default, it is inferred from the spin of the particle but this can be modified by the user, who can set the argument of `PropagatorType` to `ScalarDash` (dashed line), `Sine` (wavy line), `Straight` (straight line), `GhostDash` (dotted line), or `Curly` (gluonic line). Finally, setting the option `PropagatorArrow` to `True` or `False` allows to put an arrow or not on the propagator.

6.2 The ASperGe *interface*

From the information concerning the mixing relations among the different fields of the model (see Section 2.8), FeynRules is capable of generating a package dubbed ASperGe which allows to diagonalize, at tree-level, the associated mass matrices. The interface between FeynRules and ASperGe can be called by typing in the Mathematica session

```
WriteASperGe[ Lag, Output -> dirname ]
```

In this expression, the first argument (*i.e.*, the symbol `Lag`) is mandatory and refers to the model Lagrangian whereas the second argument (*i.e.*, the replacement rule `Output -> dirname`) is optional and indicates the name of the directory where the ASperGe files should be created. If this option is not specified, the directory `ModelName_MD` is used by default, `ModelName` being the name of the FeynRules model.

This interface works in several steps. First, all the mass matrices are extracted from the Lagrangian. This is achieved through the function `ComputeMassMatrix` introduced in Section 4.7. Second, the interface writes a set of model-independent files:

- the three C++ source and header files `MassMatrix.cpp`, `MassMatrix.hpp` and `Matrix.hpp` dedicated to matrices, their properties and their diagonalization;
- the three C++ source files, `Par.cpp`, `CPar.cpp`, `RPar.cpp`, together with the associated header files, that contain the definition of the internal format used by ASperGe with respect to the model parameters;
- the two source files `ParSLHA.cpp`, `SLHABlock.cpp`, together with the associated header files, which contain the mapping of the internal format used by ASperGe and the SLHA structure used by FeynRules;
- the two files `tools.cpp` and `tools.hpp` which are dedicated to printing and string manipulation routines;
- a makefile allowing to compile ASperGe.

Finally, the interface creates four model-dependent files:

- the main C++ program, `main.cpp`, which starts with the declaration of the different mass matrices of the model, then proceeds with their diagonalization and eventually maps the eigenvalues to the PDG codes of the physical fields;
- The two parameter files `Parameters.cpp` and `Parameters.hpp`, which contain the SLHA structure with all the model external parameters as implemented in the FeynRules model, followed by the relations linking the external and internal parameters[17]. The definitions of the mass matrices are also provided in these two files;
- the data file `Externals.dat` (stored in the subdirectory `input`) that contains the numerical values of the external parameters of the model and that the user can modify according to his/her needs.

Technically speaking, matrix diagonalization as performed by ASperGe is based on a symmetric bi-diagonalization of the mass matrices, followed by their QR reduction. This strategy follows one of the matrix diagonalization algorithms implemented in the Gsl library, which is used by ASperGe. There is a single condition that must be satisfied in order to employ such an algorithm: only Hermitian matrices can be diagonalized. Consequently, ASperGe will take care of diagonalizing the matrices $M^\dagger M$ and $MM^\dagger$ when charged fermions are involved, since their mass matrix M is by construction non-hermitian. This allows to obtain left-handed and right-handed fermion mixing matrices separately, as well as the squared mass eigenvalues. Consequently, before running the ASperGe package, the user has to verify that the Gsl libraries are installed on his/her system, together with the `g++` compiler which is employed to compile ASperGe. Otherwise, the makefile generated by the interface must be edited accordingly.

[17] For a correct running of ASperGe, the internal parameters cannot depend on the particle masses and mixing matrices.

Once compiled, ASperGe can be executed by typing in a shell

```
./ASperGe <inputfile> <outputfile>
```

The two arguments of the function refer to the file containing the numerical value of the external parameters (`<inputfile>`, which could be the file `Externals.dat` mentioned above) and the file which will contain the program results (`<outputfile>`). These results consist of the input parameters, followed by the numerical values of the mixing matrices, split in terms of their real and imaginary parts (as imposed by the SLHA conventions). Finally, the output file also contains the masses of the physical eigenstates stored in the SLHA block `MASS`.

It is also possible to indicate to the interface to create a code allowing for the diagonalization of a given subset of the mass matrices of the model, instead of all of them. This is achieved via the `Mix` option, already introduced in the context of the `ComputeMassMatrix` (see Section 4.7) function, and which works in the same way,

```
WriteASperGe[ Lag, Mix -> {"l1", "l2"} ]
```

In order to diagonalize specific mass matrices, the user can type in a shell

```
./ASperGe <infile> <outfile> m1 m2 ...
```

where `m1`, `m2`, *etc.*, are the names of the mixing matrices under consideration. All the information related to the undiagonalized mass matrices is here ignored by the code.

Finally, for practical reasons, both the program compilation and execution can be directly performed from the Mathematica session. The command to use is

```
RunASperGe[ ]
```

which stores the program output in a file named `out.dat`. The content of this file is then directly loaded into the Mathematica kernel in order to update the numerical value of all the model parameters.

6.3 The CalcHep/CompHep *interface*

The CalcHep/CompHep interface can be invoked via the command

`WriteCHOutput[` $\mathcal{L}_1$, $\mathcal{L}_2$, ..., options `]`

Table 23: CalcHEP/CompHEP Interface Options

`CHAutoWidths`	Whether CalcHep should calculate the widths on the fly, if set to `True` (the default choice). Otherwise (`False`), the values given in the model implementation are used. When the `CompHEP` option is set to `True`, the default behavior of `CHAutoWidths` is however `False`.
`CompHEP`	Allows to write a CalcHep model file (`False`, by default) or a CompHep model file (`True`).
`ModelNumber`	The number to name the model with, set to 1 by default. For example, the particle file is in this case denoted by `prtcls1.mdl`.
`Exclude4Scalars`	In some cases, models can have an abundance of vertices with four scalar fields which are phenomenologically irrelevant. These vertices can be discarded by setting this option to `True`, the default choice being `False`.
`LHASupport`	If set to `True`, the CalcHep model reads the external variables from a LHA file. If `False` (the default), the external parameters are stored in `varsN.mdl`.
`Output`	This option is only available for the function `WriteCHExtVars` and specifies an alternate file to write the external variables to. The default choice is `varsN.mdl` in the current directory where `N` refers to the model number (see `ModelNumber`).
`Input`	This option is only available for the function `ReadCHExtVars` and specifies an alternate file to read the external variables from. The default choice is `varsN.mdl` in the current directory, where `N` refers to the model number (see `ModelNumber`).

where $\mathcal{L}_1$, $\mathcal{L}_2$, ... are the various pieces of the model Lagrangian and options are the options to be passed to the interface. They are addressed throughout this section and can be found summarized in Table 23.

When invoked, this interface first creates a directory named `M$ModelName` with `-CH` appended, if it does not already exist. Then, it creates the files `prtclsN.mdl`, `varsN.mdl`, `funcN.mdl` and `lgrngN.mdl` where `N` is the number of the model. It is set to 1 by default but this can be modified through the option `ModelNumber`. The particle definitions are written to the file `prtclsN.mdl`, the external parameters (including external masses and widths) to the file `varsN.mdl` and the internal parameters (including internal masses and widths) to the file `funcN.mdl`. At this point, the interface derives the Feynman rules associated with three-point and four-point interactions and writes them to `lgrngN.mdl`. The vertex list is simplified by renaming the vertex couplings as `x1`, `x2`, `x3`, *etc.*, and the definitions of these new couplings are appended to `funcN.mdl`, along with the other internal parameters. Since CalcHep only computes internal variables once per session, this improves the speed of the phase space integration.

Although CalcHep and CompHep can calculate diagrams in both Feynman and unitary gauge, they are faster when Feynman gauge is adopted. It is therefore highly recommended to implement a new model in Feynman gauge. However, if a user decides to implement the model in unitary gauge, he/she should remember that according to the way CalcHep and CompHep have been implemented, the ghosts associated with massless non-abelian gauge bosons must be implemented. In particular, gluonic ghost fields must always be implemented in either gauge.

One major constraint of the CalcHep/CompHep system is that the color structure is implicit. For many vertices (*e.g.*, quark-quark-gluon), this is not a problem. However, for more complicated vertices, there may be an ambiguity. For this reason, the developers of CalcHep/CompHep have chosen to split them up using auxiliary fields. Although this can be done for very general vertices, it is not yet fully supported in FeynRules. Currently, only the gluon four-point vertex and squark-squark-gluon-gluon vertices are automatically split up in this way. Support for more general vertices is expected in the future.

The model files are ready to be used and can be directly copied to the CalcHep/CompHep `models` directories. Care should be taken not to overwrite model files that are already present. Furthermore, CalcHep/CompHep only load the models whose model numbers are in a consecutive set beginning with 1. Therefore, for example, if the `models` directory contains models 1, 2 and 3, the user should number his/her new model 4. Alternatively, in CalcHep, the user can import the model by specifying the directory where it is stored by using the command `IMPORT OF MODELS` in the CalcHep graphical user interface.

The default format for this interface is the CalcHep format. A user can direct

this interface to write the files in the CompHep format by use of the `CompHEP` option. One subtlety should be mentioned here. If the model is written to the CompHep directory and if the user edits the model inside CompHep and tries to save it, CompHep will complain about `C` math library functions appearing in the model. Nevertheless, it does understand them. If a model correctly works in CalcHep it also works in CompHep and leads to the same physics results.

CalcHep has the ability to calculate the widths of the particles on the fly. By default, the CalcHep interface writes model files configured for automatic width computations, regardless of the numerical values provided in the FeynRules model file. This can be turned off by setting the option `CHAutoWidths` to False. This option is set to False if `CompHEP` is set to True. The user can also fine-tune it afterwards by setting some widths to be automatic and others to be fixed in the CalcHep model files.

In some cases, it is preferable to use the LHA format for the external parameters rather than CalcHep's `varsN.mdl`. For this reason, this interface has the option `LHASupport` which, when set to `True`, causes the CalcHep model files to read the values of the external parameters from an LHA file. If set to `False` (the default), the external variables are read from `varsN.mdl` as usual for CalcHep. More information about the CalcHep support for LHA parameter files can be found in [57].

Some models, such as the most general minimal supersymmetric extension of the Standard Model have so many four-scalar vertices that it takes a very long time for FeynRules to calculate all of the Feynman rules. In many cases, these vertices are not relevant for the physics being studied and can be discarded by setting the option `Exclude4Scalars` of the interface to `True`, the default behavior being `False`.

The CalcHep interface also contains a set of functions that read and write the external parameters from and to the CalcHep variable file `varsN.mdl`. After loading the model into FeynRules, the external parameters can be updated from a CalcHep model by executing the function `ReadCHExtVars[` options `]`. This function accepts all the options of the CalcHep interface, plus the option `Input` which instructs FeynRules where to find the CalcHep variable file. The default is a file named `varsN.mdl`, located in the current working directory. In the case CompHep variable file is read, the option `CompHEP` should be set to `True`.

The current values of the external parameters in FeynRules can also be written to a CalcHep external variable file `varsN.mdl` using `WriteCHExtVars[` options `]`. This allows to bypass writing out the entire model if only the model parameters are changed.

Table 24: Options of the FeynArts interface

All the options of `FeynmanRules` can be employed, in addition to:

`Output`	The name of the FeynArts directory and files generated by the interface. The default is `M$ModelName` with `_FA` appended.
`DiracIndices`	Whether to write spin and Dirac indices in the generic file. If `Automatic`, they are added only if the model Lagrangian contains interactions with more than two fermions. The default value is `Automatic`.
`CouplingRename`	Whether to rename vertices with a new name and store the definitions in `M$FACouplings`. The default value is `True`.

Neither CalcHep nor this interface support `InteractionOrder` defined in Sec. 6.1.7. However, the strong coupling constant g_s must be used explicitly in the Lagrangian. This interface will factor it out of each vertex and put it in the `Factor` column of `lgrngN.mdl`. The electric charge `ee` should also be used explicitly. If its power is the same for each term of a vertex, it will also be factored out and put in the `Factor` column.

Further details of how CalcHep uses these model files can be found in Ref. [4].

6.4 The FeynArts *interface*

The FeynArts model files are generated from a FeynRules model implementation by issuing the command:

`WriteFeynArtsOutput[` $\mathcal{L}_1$, $\mathcal{L}_2$, ..., options `]`

where $\mathcal{L}_1$, $\mathcal{L}_2$, ... are the pieces of the model Lagrangian and options are options among those listed in Table 24. This function creates a directory denoted by `M$ModelName_FA` with three files. Each of these files is named as `M$ModelName`, followed with a different extension among `.gen`, `.mod` and `.pars` appended. Another filename root may be specified with the `Output` option of the interface. This directory can then be moved directly to the FeynArts model directory and used in FeynArts as any other built-in model.

The FeynArts generic file (`.gen`) created in this way contains the following

information:

- The kinematical indices as in the standard FeynArts file `lorentz.gen`, except if Dirac [18] indices are necessary. In this case, they are consistently added for the fermionic particles.
- `$FermionLines = True` if there are no spin indices or `$FermionLines = False` otherwise, such that fermion flows are correctly used in FeynArts.
- A copy of the simplification, truncation and last generic rules of `lorentz.gen`.
- Generic propagators as in `lorentz.gen` or with the Dirac indices for the fermion propagators (if necessary).
- The basis of the vertex structures, given as kinematic vectors. The entries of these vectors are all the possible Lorentz structures of an interaction vertex among specific generic particles (scalar fields, spinors, *etc.*). In other words, it includes the parts of the vertices containing Lorentz indices, spin indices, momenta, *etc.*
- The flipping rules, *i.e.*, the rules that are applied to reverse the flow in a vertex.

The FeynArts model file (`.mod`) contains:

- A copy of the index declarations implemented in the FeynRules model file with the `Unfold` removed. In contrast, the `NoUnfold` tag is kept and used in FeynArts. It avoids the creation of one diagram and one amplitude for each possible value of the associated indices.
- A copy of the `M$ClassDeclarations` list, as implemented in the FeynRules model file, but with the attributes not relevant for FeynArts removed.
- Some additional functions used by FeynArts.
- The list `M$CouplingMatrices`. Each element of this list consists of a set of coefficients to be mapped to the generic kinematic vectors given in the `.gen` file.
- The list `M$FACouplings`, containing replacement rules allowing to express the vertex coefficients in terms of the parameters defined in the FeynRules model file.

Finally, the parameter file (`.pars`) contains:

- The list `M$ExtParams` containing replacement rules of the external parameters by their values.
- The list `M$IntParams` containing replacement rules of the internal parameters by their values (in terms of other model parameters).
- The lists `M$Masses` and `M$Widths` containing replacement rules for the masses and widths of the particles.

[18] The spin indices of FeynRules (`Spin`) are called Dirac indices in FeynArts.

The generic file and the model file must be both specified when calling the FeynArts `InsertFields` function via the options `GenericModel` and `Model`, respectively. The parameter file can be loaded in Mathematica if necessary according to the needs of the user.

The FeynArts interface fully expands the vertices before extracting the Lorentz structures. Consequently, each Lorentz structure appearing in the kinematic vectors is never a sum or a difference. Moreover, for fermion vertices, a Lorentz structure always ends with a chirality projector. The interface completes the kinematic vectors such that they close eventually up to a sign under any permutation of the indices of identical particles (at the generic level), as required by FeynArts. The quantity `M$FlippingRules` contains all the flipping rules for zero to two gamma matrices and one chirality projector. Products of more than two gamma matrices are hence always reduced using

$$\gamma^\mu\gamma^\nu\gamma^\rho = \eta^{\mu\nu}\gamma^\rho - \eta^{\mu\rho}\gamma^\nu + \eta^{\rho\nu}\gamma^\mu - i\epsilon^{\mu\nu\rho\sigma}\gamma_\sigma\gamma_5 \ . \tag{6.40}$$

In the case all spin indices are explicit, `M$FlippingRules` is irrelevant and consists thus of an empty list.

If the option of the interface `DiracIndices` is set to `True`, it can be used to force the presence of Dirac indices even if there are at most two fermions in each interaction vertex. An error message is printed to the screen if the `DiracIndices` option is set to `False` and if the model contains vertices with more than two fermions. The default `Automatic` value implies the explicit presence of Dirac indices only if vertices with more than two fermions are found.

Eventually, FormCalc will algebraically simplify and manipulate the amplitudes created by FeynArts. Replacing long analytic expressions by symbols at the level of the vertices makes these manipulations more efficient. Therefore, when the Feynman rules are exported to a FeynArts model file by FeynRules, vertex expressions are replaced by symbols, denoted by `gc`i with i being an integer number, multiplying the various Lorentz structures. The definitions of these `gc`i quantities are stored in the list of replacement rules `M$FACouplings`, an example element of this list being `gc3 -> ee/sw`. The user may then apply (or not) this list to the results computed by FeynArts and FormCalc at any time. To turn this renaming off when calling the FeynArts interface, the option `CouplingRename` has to be set to `False`. The parameters can be further replaced by their numerical values by employing the lists of replacement rules `M$ExtParams`, `M$IntParams`, `M$Masses` and `M$Widths` that contain the values of the different parameters of the model.

Although FeynArts can use vertices expressed in any gauge, FormCalc only supports Feynman gauge. Therefore, FeynRules models can be exported to be used with FormCalc only if they are written in Feynman gauge.

As shown in Section 2.4, it is possible to gather particles with similar properties into classes. This allows FeynArts to generate fewer diagrams and FormCalc to receive fewer terms to simplify when a computation is started. The specific couplings, masses and widths of each particle can then be inserted only after performing the calculation, made in this way more efficient and faster.

The FeynRules model format is inspired from the FeynArts one and also allows to group particles into classes. Obviously, the resulting advantage at the level of FormCalc is only present if the user writes the model and its Lagrangian in a 'classy way'. However, the model can still be exported to FeynArts if the Lagrangian breaks this rule. The interface then defines one new class for each particle as soon as a vertex depending on a specific class member is found. Only the new classes are subsequently transmitted to FeynArts. Their list can be obtained with the function `NewFeynArtsClasses[]`.

6.5 The Sherpa *interface*

The Sherpa interface can be called with the following command,

`WriteSHOutput[`$\mathcal{L}_1$, $\mathcal{L}_2$,..., options`]`

where $\mathcal{L}_1$, $\mathcal{L}_2$, ..., are the different pieces of the model Lagrangian. The interface does not accept any special options besides the usual options available to `FeynmanRules[]`, the option `Exclude4Scalars` described in Section 6.3, and the option `Output` that allows to set the output directory. Issuing the above command produces a directory that contains the following text files:

- `feynrules.dat`: A static file, setting up the model in Sherpa.
- `Particle.dat`: The list of all particles together with their properties.
- `param_card.dat`: LH-like file defining the numerical values of the external parameters.
- `ident_card.dat`: File linking the entries in `param_card.dat` to the variables used in the Sherpa code.
- `param_definition.dat`: File containing analytical expressions for all the internal parameters.
- `Interactions.dat`: File defining all the interaction vertices with their couplings.

The master switch to use a FeynRules generated model within Sherpa is

```
MODEL = FeynRules
```

to be set either in the (model) section of the Sherpa run card or on the command line once the Sherpa executable is called. For further information

on the structure of the SHERPA interface, we refer to Ref. [36].

Although the SHERPA interface has been developed such as to be able to handle interactions that are as general as possible, there are several limitation on the kind of models that can be handled by the interface. The main limitation comes from the kind of structures the matrix element generator of SHERPA can handle. Currently, only Lorentz and color structures which are already included in the Standard Model or in the Minimal Supersymmetric Standard Model are supported by the interface[19]. In addition, the SHERPA interface can only handle fields with spin 0,1/2 or 1 transforming in the trivial, fundamental or adjoint representations of the QCD gauge group. Finally, one should mention that QCD showers are only invoked for colored particles present in the SM or the Minimal Supersymmetric Standard Model (MSSM). New colored states are not hadronized, and they should be decayed before entering the hadronization stage.

6.6 The TEX-Interface

FEYNRULES comes with an extended TEX-interface that can be invoked with the command

`WriteLaTeXOutput[` $\mathcal{L}_1$, $\mathcal{L}_2$, ..., V_1, V_2, ..., options `]`

where $\mathcal{L}_1$, $\mathcal{L}_2$, ... are the different pieces of the model Lagrangian and V_1, V_2, ... are vertex lists (as output by `FeynmanRules`). The options options supported by the interface can be found summarized in Table 25 and are addressed throughout this section. The TEX-interface can be invoked in several ways according to the arguments of the function `WriteLaTeXOutput[]`. It can be either called with no arguments, in which case the interface writes the TEX-files describing the model but does not include any part of the model Lagrangian or vertex list, with only Lagrangians, with only vertex lists, with only options or with any combination of Lagrangians, vertex lists and options.

The Lagrangian pieces and vertex lists can be entered in two ways. Taking the example of the Lagrangian `L1`, the interface can be simply called as in `WriteLaTeXOutput[ L1, ...]`. A TEX-name can also be associated with the Lagrangian. In this case, the interface is called as in `WriteLaTeXOutput[ {Subscript[L,Gauge],L1}, ...]` where `Subscript[L,Gauge]` denotes the Lagrangian `L1`. Vertex lists behave similarly and can be entered directly or with a name as in `WriteLaTeXOutput[ ..., {Subscript[V,Gauge],V1}, ...]`. If entered, these names will be used for the section names. If they are not used,

[19] Currently, the use of fermion number violating interactions with the SHERPA interface is discouraged.

Table 25: TeX Interface Options

`Overwrite`	Determines whether the interface must overwrite (`True`) the generated TEX files, if existing, or not (`False`). If set to `Automatic` (the default value), the user is asked, for each file, whether it has to be overwritten.
`Paper`	This option determines what kind of paper format to use. Choices are `letter` (the default value), `legal` and `a4`.

names will be generated as `L1`, `L2`, *etc.* for the Lagrangians and `V1`, `V2`, *etc.* for the vertex lists.

The files that this interface generates are split among the various sections of the TEX document. The main file is named `M$ModelName` with the extension `.tex` appended. This file instructs LATEX to include the other files,

- title.tex: Contains the title.
- abstract.tex: Contains the abstract.
- introduction.tex: Contains the introduction.
- symmetries.tex: Contains information about the gauge symmetries and indices.
- fields.tex: Contains information about the fields.
- lagrangians.tex: Contains the Lagrangians.
- parameters.tex: Contains information about the parameters.
- vertices.tex: Contains the vertices.
- bibliography.tex: Contains the bibliography.

After being written by the interface, the user can modify these files in any way he/she wishes. The next time the interface runs, it will first check whether the files exist. If they do not, it will generate them in the usual way. If they do exist, the interface will ask the user whether they should be overwritten or not. In this way, the user can make changes to files and not have them overwritten. On the other hand, if the user wishes to have all of the files overwritten, he/she can set the option `Overwrite` to `True`. However, if the user would like none of the files to be overwritten, he/she can set the option `Overwrite` to `False`. In this case, the interface skips any files that already exist. The default is for the interface to ask the user what to do for each file which already exists.

The paper in common use depends on the country a person lives in. For this reason, the option `Paper` was created. Its defaults value is `letter`, but this

option can also be set to `a4` or `legal` as in `Paper -> "a4"`.

6.7 The UFO interface

The UFO interface can be called with the following command,

`WriteUFO[`$\mathcal{L}_1$, $\mathcal{L}_2$,..., options`]`

where $\mathcal{L}_1$, $\mathcal{L}_2$, ..., are the different pieces of the model Lagrangian. The interface accepts, in addition to all the options of `FeynmanRules[]` and the option `Exclude4Scalars` described in Section 6.3, various options which are summarized in Table 26 (see also Ref. [44]). When run, the interface computes all the vertices and stores all the information about the model in the form of a PYTHON module which can be linked to existing Feynman diagram calculators. Currently, the UFO is used by ALOHA [58], MADANALYSIS 5 [45] and MADGRAPH 5 [16] and will be used in the future by GOSAM [46, 59] and HERWIG++ [23]. For details about the structure of the PYTHON module and the syntax used to store the analytical expressions for the Lorentz and color structures of the vertices, we refer to Ref. [44]. Here it suffices to say that any vertex coupling fields of spins no greater than two and transforming in the representations **1**, **3**, **6** or **8** of the QCD gauge group can be implemented into the UFO format. This is done by decomposing each vertex into a color $\otimes$ spin basis using the master formula

$$\mathcal{V}^{a_1\ldots a_n,\ell_1\ldots\ell_n}(p_1,\ldots,p_n) = \sum_{i,j} C_i^{a_1\ldots a_n}\, G_{ij}\, L_j^{\ell_1\ldots\ell_n}(p_1,\ldots,p_n)\ , \tag{6.41}$$

where $C_i^{a_1\ldots a_n}$ and $L_j^{\ell_1\ldots\ell_n}(p_1,\ldots,p_n)$ denote tensors in color and spin space respectively, and G_{ij} is a matrix of coupling constants. The analytic expression for the color and spin tensor are stored inside the UFO, and can be transformed into FORTRAN and/or C++routines using the ALOHA package [58].

Note that for the UFO files to work properly, all coupling constant should be assigned an interaction order (see Section 6.1.7).

6.8 The WHIZARD *interface*

The WHIZARD interface can be called with the following command,

`WriteWOOutput[`$\mathcal{L}_1$, $\mathcal{L}_2$,..., options`]`

where $\mathcal{L}_1$, $\mathcal{L}_2$, ..., are the different pieces of the model Lagrangian. The interface accepts, in addition to all the options of `FeynmanRules[]` and the option

Table 26: Options accepted by the UFO interface

`Output`	A string specifying the output directory. The default is `M$ModelName` with `_UFO` appended.
`Input`	A list of vertices that will be merged with the vertices computed from the Lagrangian before transforming them into the UFO format. The default is an empty list.
`AddDecays`	If `True`, the analytic expressions for all two-body decays are included into the file `decays.py`. The default is `True`.

`Exclude4Scalars` described in Section 6.3, various options which are summarized in Table 27. The full details of this interface can be found in Ref. [47]. We will summarize some of the most important points.

The WHIZARD interface can currently handle spins 0, 1/2, 1 and 2. The color representations supported include the singlet, triplet, antitriplet and octet. The full list of vertices supported by this interface can be found in Table 1 of Ref. [47] and comprises a large number of the possible vertices for these spins and color representations. When an unsupported vertex is identified, a warning message is printed, the vertex is skipped and the interface continues to process the other vertices. The option `WOFast` can be set to `False` to include additional checks to attempt to match model vertices to supported operators. A new version of WHIZARD and this interface is planned that will support all the vertices supported by FEYNRULES.

Since the strong coupling and all parameters depending on it must be evaluated for each scale, the WHIZARD interface flags these parameters to be recalculated for each phase space point while all other parameters are only calculated once for a given simulation. The list of parameters recalculated for each parameter point can be extended by use of the option `WORunParameters` which is given a list of the parameters to be recalculated.

This interface supports the unitary, Feynman and R_ξ gauges. The user can choose the gauge by setting the option `WOGauge`. The choices are `WOFeynman`, `WORxi` and `WOUnitarity` (which is the default). If the R_ξ gauge is chosen, the gauge parameter must be set using the `WOGaugeSymbol` option. If either Feynman or R_ξ gauges are chosen, the option `WOAutoGauge` can be set to `True`. In this case, the gauge symbol will automatically be created and the masses of the Goldstone bosons will automatically be determined. For this to work, the model must be implemented in Feynman gauge. Ghosts are ignored by this interface.

By default this interface writes the WHIZARD model files to the directory based

Table 27: Options of the WHIZARD interface

All the options of `FeynmanRules` can be employed, in addition to:

`Input`	A list of vertices from `FeynmanRules` to use instead of a Lagrangian.
`Output`	The name of the directory where the WHIZARD files should be written by the interface. The default is determined from `M$ModelName`.
`WOModelName`	The name by which the model will be known to WHIZARD. The default is determined from `M$ModelName`.
`WOGauge`	The gauge of the WHIZARD files. Choices are `WOFeynman`, `WORxi` and `WOUnitarity` (the default).
`WOGaugeParameter`	The parameter used to determine the gauge. The default is `Rxi`.
`WOAutoGauge`	Whether to automatically assign Goldstone boson masses in Feynman and R_ξ gauges and automatically append the parameter ξ to the parameter list in R_ξ gauge.
`WORunParameters`	Which parameters are required to be computed for each phase space point. The default is `aS` and `G`.
`WOFast`	Whether to increase the number time consuming checks to determine if a vertex is supported (`False`). The default is `True`.
`WOMaxCouplingsPerFile`	The maximum number of couplings written to a single Fortran file. The default is 500.
`WOVerbose`	Whether to enable verbose output including more extensive information for the vertices that are not supported. The default is `False`.

on `M$ModelName` but can be changed by use of the option `Output`. The name that WHIZARD uses to load the model is also based on `M$ModelName` but can be changed with the option `WOModelName`. Once the interface is done, assuming WHIZARD was installed in the standard way, the user can make the model available to WHIZARD by change into the directory where the model files were written and issuing the commands `./configure` followed by `make install`. (If this directory already had an old set of model files that were overwritten, it may be necessary to call `make clean` before compiling.) This will install the

model files in the `.whizard` subdirectory of the user's home directory. At this point, this model can be used exactly like any built-in WHIZARD model.

If the user only wants to use a new set of values for the external parameters but the model has not changed in any other way, the command `WriteWOExt-Params[filename]` can be used. It will write the current values to a sindarin file called `filename` which can be used in a WHIZARD session.

7 Running time

Compared to the previous versions of the code, the core module of FEYNRULES 2.0 has been greatly improved with respect to speed. In particular, the internal treatment of lists and/or long expressions has been deeply modified to benefit from the various strengths of the MATHEMATICA platform. The validation of the new routines have been performed by comparing results as outputted by version 1.6 of the code to those retrieved by the new version.

In addition, the execution speed of the program has also been improved thanks to parallelization so that FEYNRULES can now use more than one CPU core, if available on the system. By default, the maximum number of available cores is employed and FEYNRULES is loaded on each of them, together with one MATHEMATICA slave kernel per core. This behavior can be modified according to the user's needs by setting one of two dedicated variables accordingly. First, in the case the user wants to run FEYNRULES on a single CPU core, he/she has to issue, in a MATHEMATICA session, the command

```
FR$Parallel = False
```

before loading the FEYNRULES package. Second, in the case the user wants to employ only a subset of the available CPU power for FEYNRULES, he/she can type the command

```
FR$KernelNumber = <n>
```

where `<n>` is an integer number smaller than or equal to the maximum number of available CPU cores. In the case this command is run before FEYNRULES is started, `<n>` MATHEMATICA slave kernels are started and each of these are employed by FEYNRULES. Otherwise, if typed after FEYNRULES has been loaded, this indicates to FEYNRULES to reduce the number of slave MATHEMATICA kernels to `<n>`. Note that in order to increase the number of kernels, FEYNRULES must be restarted.

At this stage, it is still possible to force FEYNRULES to effectively employ one CPU core for the calculations that it will perform by typing

Command	FR 1.6	FR 2.0 - 1	FR 2.0 - 2	FR 2.0 - 4	FR 2.0 - 8
`FeynmanRules`	5.84 s	4.98 s	3.09 s	2.32 s	1.93 s
`WriteCHOutput`	9.33 s	9.51 s	8.05 s	6.26 s	5.53 s
`WriteUFO`	9.05 s	8.82 s	7.89 s	6.51 s	6.05 s

Table 28: Running time associated with the three FeynRules commands `FeynmanRules` (second line), `WriteCHOutput` (third line) and `WriteUFO` (fourth line), where the decay widths are not calculated. For this comparison, we use the Standard Model whose Lagrangian has been calculated before calling these commands. We compare version 1.6 of FeynRules (second column) to version 2.0 when using one (third column), two (fourth column), four (fifth column) and eight (last column) CPU cores.

```
FR$Parallelize = False
```

This commands keeps all the slave Mathematica kernels running but only employs one of them for the computations.

In Table 28, Table 29 and Table 30, we compare the running times of the code in three different contexts. Table 28 is dedicated to the Standard Model. We first compute the model Lagrangian and then execute the `FeynmanRules` command without performing the full expansion over flavor indices,

```
lag = LSM
FeynmanRules [ lag ]
```

The results are presented in the second line of the table and correspond to only the second of the two commands above. Then, we turn to the efficiencies of the interfaces and take the example of the CalcHep (third line of the table) and UFO (last line of the table) interfaces. In the last case, we omit the computation of the decay widths for the sake of the comparison, as this feature is absent in the version 1.6 of FeynRules. The running time are those obtained after typing, in a Mathematica session,

```
WriteCHOutput [ lag ]
WriteUFO [ lag, AddDecays->False ]
```

respectively. They correspond to the sum of the times needed to first extract the Feynman rules, then perform the flavor expansion and eventually translate the rules in terms of the CalcHep and UFO formats, respectively. Whereas this mimics the behavior of most users, an exclusive test of the interfaces can be achieved by computing the Feynman rules separately and providing the vertices as input to the interfaces. This however goes beyond the tests performed in this section where we have chosen to stick to the commands mostly employed by the users. For all the three commands above, we confront

Command	FR 1.6	FR 2.0 - 1	FR 2.0 - 2	FR 2.0 - 4	FR 2.0 - 8
`FeynmanRules`	325.5 s	213.7 s	79.7 s	62.6 s	41.0 s
`WriteCHOutput`	853.4 s	618.9 s	350.8 s	283.9 s	204.4 s
`WriteUFO`	436.0 s	518.5 s	316.1 s	273.8 s	239.7 s

Table 29: Same as Table 28, but for the MSSM. In contrast to the Standard Model case, four scalar interactions have been removed when running the commands `WriteCHOutput` and `WriteUFO`.

results as obtained by means of FeynRules 1.6, the previous stable version, to those obtained with FeynRules 2.0 when using one, two, four and eight cores. For the tests, we employ a machine with a 2.3 GHz Intel Core i7 processor and 16 GB of memory (1600 MHz DDR3). Moreover, the operating system is MacOS X 10.8.4. As can be seen from the table, the speed improvement between the older and new version of FeynRules is significant, especially when more than one core is employed.

Table 29 addresses the MSSM. As for the Standard Model, the Lagrangian is first computed separately and then employed together with the `FeynmanRules` command (second line of the table),

```
lag = Lag
FeynmanRules [ lag ]
```

Concerning the interfaces, we optimize the output by removing the hundreds of vertices describing four-scalar interactions which are, for tree-level computations, in general phenomenologically less relevant. Moreover, the parameters are taken as those of the minimal supergravity scenario SPS 1a [60] [20] . We refer to the Appendix of Ref. [36] for the adopted numerical values of the model's free parameters. For the comparison of the running time, we hence issue, in Mathematica,

```
WriteCHOutput [ lag , Exclude4Scalars -> True]
WriteUFO [ lag, AddDecays->False, Exclude4Scalars -> True ]
```

We find similar conclusions as for the Standard Model case except when comparing FeynRules 1.6 with the newer version run with one single CPU core. It may indeed seem that the UFO interface in FeynRules 2.0 is less efficient than in the previous version 1.6. However, this is explained by the fact that the resulting UFO library is more efficient concerning the handling of the model parameters. The gain in time consequently only appears when performing

[20] Although this benchmark point is now experimentally excluded, it is still employed as a standard choice for the default values of the MSSM model files of most of matrix element generators. This motivates our choice.

Command	FR 1.6	FR 2.0 - 1	FR 2.0 - 2	FR 2.0 - 4	FR 2.0 - 8
`WriteCHOutput`	960.1 s	1134.0 s	356.6 s	305.6 s	254.2 s
`WriteUFO`	871.9 s	984.1 s	316.6 s	283.3 s	250.7 s

Table 30: Same as Table 29, but for after applying restrictions as in the SPS 1a benchmark point (no flavor violation in the fermion and sfermion sectors).

matrix element computations with, *e.g.*, MadGraph 5.

Finally, in Table 30, we apply, before calling the interface, a set of restrictions (see Section 2.7) as motivated by the SPS 1a point. In more detail, flavor violation is the fermion and sfermion sectors is not allowed. This is achieved by issuing, after the computation of the Lagrangian,

```
WriteRestrictionFile[ ]
LoadRestriction["ZeroValues.rst"]
```

8 Model Validation and Debugging

By using FeynRules, the risk of creating faulty model implementations is reduced. However, possible issues can still arise from either faulty FeynRules input or software bugs. We have taken great pains to validate the FeynRules package [36], and we, further, plan to continually extend and improve these tests in order to make the FeynRules package ever more dependable. On the other hand, when a new model is implemented, it has been up to the author of that new model to validate it. In order to ensure a high level of quality for the models implemented using FeynRules, we have proposed the following set of validation guidelines [38]:

- **Documentation:** Full documentation should be included to ensure traceability and reproducibility. Relevant information should be included in the model file in the variable `M$ModelInformation` and include references to the appropriate papers for the model and the model implementation as well as any URLs where further information can be obtained. Furthermore, versions for the operating system, Mathematica and FeynRules where any tests were performed should be included.
- **Basic sanity checks:** The model implementation should satisfy basic sanity tests such as hermiticity and gauge invariance. The Feynman rules obtained from FeynRules should be compared with those in the literature and simple cross sections and/or decay rates should be computed and compared with the literature as well. The results of these tests should be included in the documentation.

- **Testing one generator:** The model should be exported to the format of one of the matrix-element generators and detailed calculations should be performed. Comparisons should be made with the literature, with the built-in SM calculations for processes where the new physics does not have an effect and with any other implementations of the same model. If multiple gauges are available in the matrix-element generator, they should be compared. High energy unitarity cancellation should be tested if applicable (the widths should be set to zero for this test). The results should be included in the documentation.
- **Testing several generators:** A full comparison of a very large set of processes should be done between multiple matrix-element generators supported by FeynRules as well as between supported gauges. Additionally, independently implemented versions of the same model should be compared. For this step it is important that the widths be set to zero since each matrix-element generator has slightly different methods of dealing with them. The results should be included in the documentation.

Although it is virtually impossible to prove the absolute correctness of a model implementation, following these guidelines should give greater confidence. Unfortunately, in practice the level of model validation has been irregular. One of the challenges is that although it is straightforward to test a handful of Feynman rules by hand, validating the full set of vertices, of which there can be thousands, has been difficult. One approach to this task has been to automate the calculation of the cross section by the different matrix element generators (MEG), in different gauges and in different, independent implementations and compare the results [36]. We believe that this procedure nicely complements the sanity tests and the validation of a small set of Feynman rules by hand. However, it can be difficult for a user to set up such an automated test for several reasons. It requires expertise in each code that is to be compared as well as a careful matching of the setups for each code (in particular, the cuts and model input).

It is our goal to attain a very high level of validation for every model implemented in FeynRules. To this end, we have begun developing a new automated web validation which abstracts the details of the matrix element generator setup [50]. Experts in each matrix element generator takes care of the installation and running of the MEG on the server. The model implementation author accesses the MEG through a web interface where he/she uploads his/her FeynRules model files to the server. The server uses the uploaded files to generate model files for the supported MEGs and allows the user to run validations on the model implementation. The server displays the results on the screen and marks processes where inconsistencies are suspected, giving the user an idea of where to look for problems. If no inconsistencies are found, the user gains greater confidence in his/her model.

Although this software is still in a beta stage and we feel there is much more that should be done to improve it, it is already useful for validation. For this reason, it has been made public.

8.1 URL and Account

The web interface can be accessed from any web enabled device by pointing a web browser at the url:

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

The browser will be immediately redirected to the author's personal model page. If not already logged in, the user will be redirected to the login page. To obtain a user account, the author must currently email one of the FeynRules authors to request one. The models page contains a list with all the models uploaded so far.

8.2 Uploading a model

To validate a model, the user must first upload his/her model files. To do this, the user should click the 'New Model' button on his or her model page, which leads to a form where the user specifies the properties of the new model to be uploaded.

The first thing the user must specify is the location of his/her model files. More than one model file can be uploaded as long as all model files can be loaded together using the `LoadModel` function as in

```
LoadModel[file1, file2,...]
```

where `file1, file2,...` are the names of the uploaded files. Although a model file that directly calls code from other files is allowed in FeynRules, it is not supported in the web validation.

The user may also upload restriction files and parameter files if there are any. Multiple restriction and parameter files upload is possible. At a later stage, when the MEG files are generated, the user is hence able to choose which restriction files and which parameter files to employ (see Section 8.3). Multiple sets of MEG files can be generated, each with a unique set of restriction files and parameter files. The restriction files must be loadable by the `LoadRestriction` function (see Section 2.7) as in

```
LoadRestriction[file1, file2, ...]
```

where `file1, file2,` *etc.*, are the restriction files. The parameter files must be in a LH form and be loadable by the `ReadLHAFile` function (see Section 6.1.4) as in

```
ReadLHAFile[Input->file]
```

where `file` is the parameter file.

There is also a text box where the user can specify the Lagrangian of his or her model, in any form allowed by the `FeynmanRules` function (see Section 4.3). For example, taking the Standard Model implementation, the user could equivalently enter any of the three choices

```
LSM
{LGauge,LFermion,LHiggs,LYukawa,LGhost}
LGauge+LFermion+LHiggs+LYukawa+LGhost
```

There is also the option to turn off four-scalar interactions, which is useful for some models where the number of such interactions is impractical (see Section 6.3). Finally, the user can choose between the publicly available version (the default) or the development version of FeynRules.

Once the user finishes filling in the form, he or she can click on the 'Submit' button. The browser uploads the model files and information. The server organizes the files and runs FeynRules on them for the first time, without generating MEG files. It is just testing whether FeynRules can load the model files, the restriction files and the parameter files without any major error. If there is no error, the user can move on to the validations themselves. If there are errors, the user is blocked from proceeding.

The web validation is not designed to help solving syntax errors. It is the responsibility of the user to test that his or her model files, restriction files and parameter files can all be loaded by FeynRules before uploading them to the web validation. The web validation does not currently support the backup or the versioning of models or validations and so should not be used for long term storage. It is hoped that backup and versioning will be added in a future version.

After uploading the model files, the browser is sent to the model page for the new model. This page is where all the information for the model and access to the validations for the model is presented. At the top of the model page is the user's name. Clicking on his or her name will take the user back to his/her models page where the new model is listed along with any other models the user has uploaded.

8.3 Generating matrix element generator files

Assuming the uploaded model files pass the basic tests described above, the next step is to generate MEG files. To do this, the user must click on the 'Create MEG Files' button. A small dialog opens and allows to choose a name for the MEG files as well as one or more restriction files, if desired, and a parameter file, if desired. The user then clicks the 'Create' button upon which the web server creates jobs for each MEG and submits them to the cluster queue.

Each MEG job loads the model and any restriction and parameter files requested. It then attempts to run the MEG interface on the model. If it is unsuccessful, that MEG will not go any further for this model, restriction file(s) and parameter file combination. If it was successful, the server submits another job to the queue where the MEG attempts to generate a cross section for the process $e^+e^- \to \mu^+\mu^-$. Any non-zero cross section at this stage is taken to indicate that the model, restriction files and parameter files together with the MEG interface are capable of producing compilable code. If the code does not compile or the result returned is zero, the MEG is again blocked from further calculations with this restriction parameter combination. If both of these tests are passed, the MEG is made available for further computations.

The status of each job is communicated to the user by two symbols below the MEG name. The first communicates whether the FeynRules interface succeeded. The second communicates whether the MEG calculation of $e^+e^- \to \mu^+\mu^-$ succeeded. Possible symbols for these are `Q` which means that the job is queued but not yet completed, a green ✓which means the job was successful, a red x which means the job failed, and a `?` which means the result is unknown (and should not normally happen). Any MEG that passes both tests can continue even if others failed.

More than one restriction-parameter file combinations can be created according to the needs of the user. Each can have multiple validations run on them. It is not necessary to generate the same restriction-parameter file combination multiple times since they will be identical.

All restriction-parameter file combinations are presented on the model page in a dedicated section.

8.4 Running $2 \to 2$ validations

The next step is to create a new validation. On the model page, a list of restriction parameter MEG combinations are presented. For each of them,

there is a list of validations that the user has already created as well as a 'Create New Validation' button. Clicking on this button takes the user to the new validation page.

On the new validation page, the user specifies the details of the validation beginning with giving it a name. Next, the user chooses whether he or she would like to compare $2 \to 2$ processes or another topology. The user can also choose whether to compare integrated cross sections or phase space points. At the time of this writing, only $2 \to 2$ integrated cross section comparisons are supported, other validations being planned for the future.

The default is for the server to compare all $2 \to 2$ processes which satisfy all the symmetries specified in the model and cannot be rotated into one another. There is the possibility to reduce the number of processes by use of a restriction menu. The user can specify at least how many particles of a certain type must appear in the external states of each process. The user can also specify at least how many particles of a certain type must not appear in the external states of each process.

The restriction categories include the spin of the particle, the indices of the particle and the charges of the particle. These can be combined to tune the list of processes that results. For example, the user could specify that they want two or more spin 1/2 fermions, two or more particles with triplet color indices, two or more particles that are not spin 1/2 fermions, and four or more particles that are not scalar fields. This would result in processes with two quarks and two vector bosons in the Standard Model.

Once the form has been filled out, the user clicks the 'Create Validation' button. The server submits a job to the cluster to generate the processes. The user is taken to the validation page. When the job is finished, the user is presented with a list of the processes and allowed to start a validation.

At this point the user can choose the MEG's and gauges he or she would like to run. Any MEG that passed the tests during the restriction-parameter combination MEG file generation are available. The user chooses between them and then clicks on either 'Start Fresh Validation' or 'Finish Validation'. If 'Start Fresh Validation' is chosen, the server first removes any previous result for this validation and begins the validation. If 'Finish Validation' is chosen, the server keeps the old results and runs the validation for any missing results.

The 'Finish Validation' option can be useful in many scenarios. If the user would like to remove one or more of the MEG's or a gauge, he or she can simply turn those off and click 'Finish Validation'. The server will remove the results for the MEG's and gauges that the user has removed. It will keep all the others and return the results. In another scenario, the user may wish to add a MEG or a gauge to the results without redoing the results that are

already present. In this case, the user simply checks the boxes for the new MEG's and/or gauges and clicks 'Finish Validations'. The server will keep all the old results and start jobs for the new MEG's and/or gauges.

When a job is started, jobs for all the cross section calculations are submitted to the cluster. They are mixed with any other jobs on the cluster so that everyone who uses the cluster will see progress at roughly the same rate. A user's jobs do not have to wait for someone else's validation to finish. Likewise, the user can run multiple validations at the same time.

The web page for the validation is updated periodically. The user can refresh his or her browser to see the completed results. Any jobs that are finished will appear with their cross section. If all the MEG's and gauges are finished for a particular process, the χ^2 is also presented for that process, calculated as

$$\chi_i^2 = \sum_i \left(\frac{\sigma_i - \sigma_b}{\Delta\sigma_i} \right)^2 \tag{8.42}$$

where σ_i is the cross section for the i^{th} MEG and gauge choice, $\Delta\sigma_i$ is the Monte Carlo uncertainty in that calculation and σ_b is the best value for the cross section which is calculated to minimize the χ^2 associated with the process under consideration

$$\frac{\partial \chi_i^2}{\partial \sigma_b} = 0 \tag{8.43}$$

The best value is shown on the web page along with the other results and the χ^2 for the analysis of the user.

The user can hover his or her mouse over a result to see the cross section for that MEG and gauge as well as the Monte Carlo error and the number of standard deviations it is away from the best value. Additionally, the user can click on the cross section from a MEG to see details from the calculation.

The results are ordered from highest χ^2 to lowest to make it easier for the user to spot potentially problematic processes which may guide the user to problematic areas of his or her model files. Very high values are almost certainly a sign of problems. Smaller values may be the result of statistical fluctuations.

The results of the χ^2 for each process are binned and plotted as a histogram at the top of the validation page. In addition, the theoretical χ^2 distribution (with $n-1$ degrees of freedom where n is the number of MEG's and gauges compared) is plotted alongside the results of the calculations. We find that when a model is correctly implemented, the results are better than or as good as the theoretical χ^2 curve. If there are results from the calculation far outside the theoretical χ^2 curve, it is usually an indication of problems with the model. The user can click on this figure to download an `.eps` version.

We find that the greatest success occurs if multiple MEG's are compared in just one gauge or when gauge invariance is checked separately with just one MEG at a time. As already mentioned, multiple validations can be created for each restriction parameter combination.

A LaTeX/PDF output is planned but not yet implemented.

8.5 Independent Model Implementations

If a model has been implemented independently, it is important to check that the new implementation agrees with the old. For this purpose, the web validation allows to compare to "stock" implementations. The user starts this process on his/her main model page under the heading "Stock Models" by clicking "Add New Stock Model" which brings up a dialog box for the new stock model. The user enters the name that this stock model should be known by, chooses the matrix element generator, the model, the gauge, uploads model files as necessary, uploads a particle translation file, and uploads zero or more parameter files. We will now make some comments about each of these.

The user has the option to choose one of the built-in models from the matrix element generators or to use a version located on their hard drive. To use their own version, they should choose "Upload My Own Model Files". At this point, they can upload the model files as a single gzipped tar-ball. For CalcHep, this should be no directory in this tar-ball. For MadGraph, this should be a tar-ball of the UFO files including the main containing directory. For Whizard, this should be a tar-ball of the directory containing all the code including the makefile.

Where supported, the user can choose the gauge for this stock model. If both gauges are desired, the stock model should be uploaded twice, once for each gauge.

Next, the user must supply a particle translation file which tells the web validation platform what the names of each particle are, if different from the FeynRules names. This is a pure text file with each particle on a separate line. For each particle, the first string should be the FeynRules name followed by a comma followed by the stock name. Lines beginning with hashes are comments and ignored. Here is an example:

```
#Standard Model particles
#FR , CH
a    , A
g    , G
ve   , ne
```

`ve~ , Ne`

This file should not be zipped when uploaded.

Finally, sometimes the same stock model is desired to be run with several different parameter files. This can be done by uploading one or more parameter files in the native format of the matrix element generator. For CalcHep, this is a `varsN.mdl` file. For MadGraph, this is a `param_card.dat` file. For Whizard, this is a `sindarin` file. If the model files uploaded already have the parameter point required for the validations, no parameter files need be uploaded.

Once all these files are uploaded, the server tries to run the process $e^+, e^- \to \mu^+, \mu^-$ with the stock model. If it gets a non-zero result, this model becomes available for validations. To use it in a validation, check the box next to it before running the validation.

We should mention that for this to work, the user has to painstakingly make sure all the parameters and definitions are set exactly the same. This means, for example, that all the widths have to be set to zero and all the masses and parameters have to be fixed to the same value as in the FeynRules model files.

8.6 Model Debugging

When bugs in the model files occur, it can be difficult to track them down and fix them. In this subsection, we would like to describe some of the methods the user can follow to accomplish this. The first problem that users often face is syntax errors. After writing a model file, the user attempts to load it in a FeynRules Mathematica session but gets a list of errors. This is usually caused by simple syntax errors, although sometimes the Mathematica syntax is ok but something has been implemented incorrectly. When this happens, the user should comment out everything in the FeynRules model files that is new since the last time the model files worked. Then, the user should iteratively uncomment small pieces of the model file and retest loading it into FeynRules. This process should be continued until the smallest possible chunk that breaks it is found. Once this is done, the user can find the typo and fix it. There may be several problems like this that need to be solved one-at-a-time.

Once the model loads without error messages, the user should next test the hermiticity of their Lagrangian, the proper diagonalization and normalization of the kinetic and mass parts of the Lagrangian as well as the Feynman rules in the literature. If problems are found, they should be fixed before moving

on. After this, the user should use the export interface of their choice. If problems are found at this stage or at the stage of using the resulting model files in the Feynman diagram calculator, the user should review the appropriate subsection of Section 6 to make sure they have followed all the requirements for that package. Inconsistencies should be corrected before moving on. Once the model appears to be running, the user should do a thorough check between gauges and matrix-element generators on the web validation.

We have found that following this list of steps enables most bugs to be removed. If the user still has issues after following these guidelines, he/she may request help from the FeynRules authors. However, the user should reduce the problem to the minimal case where it is present and show that he/she has put forth a proper effort to uncover the problem themselves. In addition, the version of their operating system, Mathematica, FeynRules, and any matrix-element generator used should be included in any request for help.

9 Conclusions

The FeynRules package allows a model builder to implement his/her new theoretical model in a unified format, independent of the matrix element generator it will be used in. The user inputs a list of fields, parameters, symmetry groups and Lagrangian. FeynRules then computes the Feynman rules and outputs them to the format appropriate for the Feynman diagram calculator of choice. Interfaces for CalcHep/CompHep, FeynArts/FormCalc, MadGraph Sherpa and Whizard/Omega are currently available which allow FeynRules models to be used natively with these Feynman diagram calculational packages without any modification to the packages. This has made these model implementations more robust, dependable and transferable. Since the first version of FeynRules, many new features have been added to FeynRules requiring the release of a new major version with this accompanying manual.

With these new features, FeynRules has created a platform where the tree-level phenomenology of large varieties of new physics models can easily be studied. Recently, however, new automated tools allow the user to generate events at next-to-leading order (NLO) in perturbation theory [46, 61–65]. These tools require not only the input of the tree-level vertices of the model, but in addition the user has to provide all the one-loop ultra-violet counterterms required to renormalize the one-loop amplitude. Furthermore, depending on the underlying algorithm to construct the one-loop matrix element, additional tree-level-like vertices need to be included to reproduce the correct rational terms in the one-loop amplitude [66–69]. While all of these new vertices formally go beyond a simple tree-level computation from a Lagrangian, they

are nevertheless universal (for a given model) and can be computed once and for all. Future versions of FeynRules will allow the user to compute these ingredients required for NLO computations automatically from the tree-level Lagrangian of the model, thus allowing for an automated event generation at NLO accuracy for large classes of new physics models.

Acknowledgments

We are grateful to Johan Alwall, Priscila de Aquino, Karen de Causmaecker, Nicolas Deutschmann, Jorgen D'Hondt, Camilo Garcia-Cely, David Grellscheid, Thomas Hahn, Valentin Hirschi, Fabio Maltoni, Olivier Mattelaer, Kentarou Mawatari, Bettina Oexl, Gizem Öztürk, Michel Rausch de Traubenberg, Thomas Reiter, Jürgen Reuter, Christian Speckner, Tim Stelzer and Yoshitaro Takaesu for discussion and collaborations on the different interfaces. We acknowledge the support of the IT team at UCLouvain (Vincent Boucher, Jerôme de Favereau, Pavel Demin). The FeynRules team is grateful to the IPHC laboratory for support in organizing the 2010 and 2012 FeynRules meetings at Mont Sainte-Odile. This work was supported in part by the Research Executive Agency (REA) of the European Union under the Grant Agreement number PITN-GA-2010-264564 (LHCPhenoNet) and MCnetITN FP7 Marie Curie Initial Training Network PITN-GA-2012-315877. A.A. acknowledges partial support of a PhD fellowship of the French ministry for education and research. A.A. and B.F were supported in part by the French ANR 12 JS05 002 01 BATS@LHC and by the Theory-LHC-France initiative of the CNRS/IN2P3. Ce. D. was supported in part by the U. S. Department of Energy under Contract No. DE-FG02-13ER42001 and by the NSF grant PHY-0757889. N.D.C. was supported in part by the LHC-TI under U.S. National Science Foundation, grant NSF-PHY-0705682, by PITT PACC, and by the U.S. Department of Energy under grant No. DE-FG02-95ER40896 and C.D. was supported by the ERC grant "IterQCD".

References

[1] A. Pukhov, et al., CompHEP: A package for evaluation of Feynman diagrams and integration over multi-particle phase space. User's manual for version 33, arXiv:hep-ph/9908288.

[2] E. Boos, et al., CompHEP 4.4: Automatic computations from Lagrangians to events, Nucl. Instrum. Meth. A534 (2004) 250–259. arXiv:hep-ph/0403113, doi:10.1016/j.nima.2004.07.096.

[3] A. Pukhov, CalcHEP 3.2: MSSM, structure functions, event generation, batchs, and generation of matrix elements for other packages, arXiv:hep-ph/0412191.

[4] A. Belyaev, N. D. Christensen, A. Pukhov, CalcHEP 3.4 for collider physics within and beyond the Standard Model, Comput.Phys.Commun. 184 (2013) 1729–1769. arXiv:1207.6082, doi:10.1016/j.cpc.2013.01.014.

[5] T. Hahn, M. Perez-Victoria, Automatized one-loop calculations in four and D dimensions, Comput. Phys. Commun. 118 (1999) 153–165. arXiv:hep-ph/9807565, doi:10.1016/S0010-4655(98)00173-8.

[6] T. Hahn, Generating Feynman diagrams and amplitudes with FeynArts 3, Comput. Phys. Commun. 140 (2001) 418–431. arXiv:hep-ph/0012260, doi:10.1016/S0010-4655(01)00290-9.

[7] T. Hahn, A Mathematica interface for FormCalc-generated code, Comput. Phys. Commun. 178 (2008) 217–221. arXiv:hep-ph/0611273, doi:10.1016/j.cpc.2007.09.004.

[8] T. Hahn, FormCalc 6, PoS ACAT08 (2008) 121.

[9] S. Agrawal, T. Hahn, E. Mirabella, FormCalc 7arXiv:1112.0124.

[10] A. Kanaki, C. G. Papadopoulos, HELAC: A Package to compute electroweak helicity amplitudes, Comput.Phys.Commun. 132 (2000) 306–315. arXiv:hep-ph/0002082, doi:10.1016/S0010-4655(00)00151-X.

[11] A. Cafarella, C. G. Papadopoulos, M. Worek, Helac-Phegas: a generator for all parton level processes, Comput. Phys. Commun. 180 (2009) 1941–1955. arXiv:0710.2427, doi:10.1016/j.cpc.2009.04.023.

[12] T. Stelzer, W. F. Long, Automatic generation of tree level helicity amplitudes, Comput. Phys. Commun. 81 (1994) 357–371. arXiv:hep-ph/9401258, doi:10.1016/0010-4655(94)90084-1.

[13] F. Maltoni, T. Stelzer, MadEvent: Automatic event generation with MadGraph, JHEP 02 (2003) 027. arXiv:hep-ph/0208156.

[14] J. Alwall, et al., MadGraph/MadEvent v4: The New Web Generation, JHEP 09 (2007) 028. arXiv:0706.2334.

[15] J. Alwall, et al., New Developments in MadGraph/MadEvent, AIP Conf. Proc. 1078 (2009) 84–89. arXiv:0809.2410, doi:10.1063/1.3052056.

[16] J. Alwall, M. Herquet, F. Maltoni, O. Mattelaer, T. Stelzer, MadGraph 5 : Going Beyond, JHEP 1106 (2011) 128. arXiv:1106.0522, doi:10.1007/JHEP06(2011)128.

[17] T. Gleisberg, et al., SHERPA 1.alpha, a proof-of-concept version, JHEP 02 (2004) 056. arXiv:hep-ph/0311263.

[18] T. Gleisberg, et al., Event generation with SHERPA 1.1, JHEP 02 (2009) 007. arXiv:0811.4622, doi:10.1088/1126-6708/2009/02/007.

[19] M. Moretti, T. Ohl, J. Reuter, O'Mega: An Optimizing matrix element generator, arXiv:hep-ph/0102195.

[20] W. Kilian, T. Ohl, J. Reuter, WHIZARD: Simulating Multi-Particle Processes at LHC and ILC, arXiv:0708.4233.

[21] G. Corcella, et al., HERWIG 6: An event generator for hadron emission reactions with interfering gluons (including supersymmetric processes), JHEP 01 (2001) 010. arXiv:hep-ph/0011363.

[22] G. Corcella, et al., HERWIG 6.5 release notearXiv:hep-ph/0210213.

[23] M. Bahr, S. Gieseke, M. Gigg, D. Grellscheid, K. Hamilton, et al., Herwig++ Physics and Manual, Eur.Phys.J. C58 (2008) 639–707. arXiv:0803.0883, doi:10.1140/epjc/s10052-008-0798-9.

[24] K. Arnold, L. d'Errico, S. Gieseke, D. Grellscheid, K. Hamilton, et al., Herwig++ 2.6 Release NotearXiv:1205.4902.

[25] T. Sjostrand, P. Eden, C. Friberg, L. Lonnblad, G. Miu, et al., High-energy physics event generation with PYTHIA 6.1, Comput.Phys.Commun. 135 (2001) 238–259. arXiv:hep-ph/0010017, doi:10.1016/S0010-4655(00)00236-8.

[26] T. Sjostrand, S. Mrenna, P. Skands, PYTHIA 6.4 physics and manual, JHEP 05 (2006) 026. arXiv:hep-ph/0603175.

[27] T. Sjostrand, S. Mrenna, P. Skands, A Brief Introduction to PYTHIA 8.1, Comput. Phys. Commun. 178 (2008) 852–867. arXiv:0710.3820, doi:10.1016/j.cpc.2008.01.036.

[28] A. V. Semenov, LanHEP: A package for automatic generation of Feynman rules in gauge modelsarXiv:hep-ph/9608488.

[29] A. Semenov, LanHEP: A package for automatic generation of Feynman rules from the Lagrangian, Comput. Phys. Commun. 115 (1998) 124–139. doi:10.1016/S0010-4655(98)00143-X.

[30] A. V. Semenov, LanHEP: A package for automatic generation of Feynman rules in field theory. Version 2.0arXiv:hep-ph/0208011.

[31] A. Semenov, LanHEP - a package for the automatic generation of Feynman rules in field theory. Version 3.0arXiv:0805.0555.

[32] A. Semenov, LanHEP - a package for automatic generation of Feynman rules from the Lagrangian. Updated version 3.1arXiv:1005.1909.

[33] N. D. Christensen, C. Duhr, FeynRules - Feynman rules made easy, Comput. Phys. Commun. 180 (2009) 1614–1641. arXiv:0806.4194, doi:10.1016/j.cpc.2009.02.018.

[34] F. Staub, SARAHarXiv:0806.0538.

[35] F. Staub, SARAH 3.2: Dirac Gauginos, UFO output, and more, Computer Physics Communications 184 (2013) pp. 1792–1809. arXiv:1207.0906, doi:10.1016/j.cpc.2013.02.019.

[36] N. D. Christensen, P. de Aquino, C. Degrande, C. Duhr, B. Fuks, et al., A Comprehensive approach to new physics simulations, Eur.Phys.J. C71 (2011) 1541. arXiv:0906.2474, doi:10.1140/epjc/s10052-011-1541-5.

[37] S. Ask, N. D. Christensen, C. Duhr, C. Grojean, S. Hoeche, et al., From Lagrangians to Events: Computer Tutorial at the MC4BSM-2012 WorkshoparXiv:1209.0297.

[38] J. M. Butterworth, et al., The Tools and Monte Carlo working group Summary Report, arXiv:1003.1643.

[39] N. D. Christensen, P. de Aquino, N. Deutschmann, C. Duhr, B. Fuks, et al., Simulating spin-3/2 particles at collidersarXiv:1308.1668.

[40] A. Alloul, N. D. Christensen, C. Degrande, C. Duhr, B. Fuks, New developments in FeynRulesarXiv:1309.7806.

[41] C. Duhr, B. Fuks, A superspace module for the FeynRules package, Comput. Phys. Commun. 182 (2011) 2404–2426. arXiv:1102.4191, doi:10.1016/j.cpc.2011.06.009.

[42] A. Alloul, M. Frank, B. Fuks, M. R. de Traubenberg, Chargino and neutralino production at the Large Hadron Collider in left-right supersymmetric modelsarXiv:1307.5073.

[43] A. Alloul, J. D'Hondt, K. De Causmaecker, B. Fuks, M. R. de Traubenberg, Automated mass spectrum generation for new physics, Eur. Phys. J. C73 (2013) 2325. arXiv:1301.5932.

[44] C. Degrande, C. Duhr, B. Fuks, D. Grellscheid, O. Mattelaer, et al., UFO - The Universal FeynRules Output, Comput.Phys.Commun. 183 (2012) 1201–1214. arXiv:1108.2040, doi:10.1016/j.cpc.2012.01.022.

[45] E. Conte, B. Fuks, G. Serret, MadAnalysis 5, A User-Friendly Framework for Collider Phenomenology, Comput.Phys.Commun. 184 (2013) 222–256. arXiv:1206.1599.

[46] G. Cullen, N. Greiner, G. Heinrich, G. Luisoni, P. Mastrolia, et al., Automated One-Loop Calculations with GoSam, Eur.Phys.J. C72 (2012) 1889. arXiv:1111.2034, doi:10.1140/epjc/s10052-012-1889-1.

[47] N. D. Christensen, C. Duhr, B. Fuks, J. Reuter, C. Speckner, Introducing an interface between WHIZARD and FeynRules, Eur.Phys.J. C72 (2012) 1990. arXiv:1010.3251, doi:10.1140/epjc/s10052-012-1990-5.

[48] N. D. Christensen, B. Fuks, J. Reuter, C. Speckner, Exploring compactified HEIDI models at the LHCarXiv:1204.6264.

[49] J. Alwall, C. Duhr, B. Fuks, O. Mattelaer, D. G. Öeztürk, C.-H. Shen (in preparation).

[50] G. Brooijmans, B. Gripaios, F. Moortgat, J. Santiago, P. Skands, et al., Les Houches 2011: Physics at TeV Colliders New Physics Working Group ReportarXiv:1203.1488.

[51] P. Z. Skands, B. Allanach, H. Baer, C. Balazs, G. Belanger, et al., SUSY Les Houches accord: Interfacing SUSY spectrum calculators, decay packages, and event generators, JHEP 0407 (2004) 036. arXiv:hep-ph/0311123, doi:10.1088/1126-6708/2004/07/036.

[52] B. Allanach, C. Balazs, G. Belanger, M. Bernhardt, F. Boudjema, et al., SUSY Les Houches Accord 2, Comput.Phys.Commun. 180 (2009) 8–25. arXiv:0801.0045, doi:10.1016/j.cpc.2008.08.004.

[53] B. Fuks, Beyond the Minimal Supersymmetric Standard Model: from theory to phenomenology, Int.J.Mod.Phys. A27 (2012) 1230007. arXiv:1202.4769, doi:10.1142/S0217751X12300074.

[54] B. Fuks, M. Rausch de Traubenberg, Supersymétrie - exercices avec solutions, Editions Ellipses Marketing, 2011.

[55] [online][link].

[56] J. Beringer, et al., Review of Particle Physics (RPP), Phys.Rev. D86 (2012) 010001. doi:10.1103/PhysRevD.86.010001.

[57] G. Belanger, N. D. Christensen, A. Pukhov, A. Semenov, SLHAplus: a library for implementing extensions of the standard model, Comput.Phys.Commun. 182 (2011) 763–774. arXiv:1008.0181, doi:10.1016/j.cpc.2010.10.025.

[58] P. de Aquino, W. Link, F. Maltoni, O. Mattelaer, T. Stelzer, ALOHA: Automatic Libraries Of Helicity Amplitudes for Feynman Diagram Computations, Comput.Phys.Commun. 183 (2012) 2254–2263. arXiv:1108.2041, doi:10.1016/j.cpc.2012.05.004.

[59] G. Cullen, N. Greiner, G. Heinrich, G. Luisoni, P. Mastrolia, et al., GoSam: A Program for Automated One-Loop Calculations, J.Phys.Conf.Ser. 368 (2012) 012056. arXiv:1111.6534, doi:10.1088/1742-6596/368/1/012056.

[60] B. Allanach, M. Battaglia, G. Blair, M. S. Carena, A. De Roeck, et al., The Snowmass Points and Slopes: Benchmarks for SUSY Searches, Eur.Phys.J. C25 (2002) 113–123. doi:10.1007/s10052-002-0949-3.

[61] C. Berger, Z. Bern, L. Dixon, F. Febres Cordero, D. Forde, et al., An Automated Implementation of On-Shell Methods for One-Loop Amplitudes, Phys.Rev. D78 (2008) 036003. arXiv:0803.4180, doi:10.1103/PhysRevD.78.036003.

[62] V. Hirschi, R. Frederix, S. Frixione, M. V. Garzelli, F. Maltoni, et al., Automation of one-loop QCD corrections, JHEP 1105 (2011) 044. arXiv:1103.0621, doi:10.1007/JHEP05(2011)044.

[63] F. Cascioli, P. Maierhofer, S. Pozzorini, Scattering Amplitudes with Open Loops, Phys.Rev.Lett. 108 (2012) 111601. arXiv:1111.5206, doi:10.1103/PhysRevLett.108.111601.

[64] G. Bevilacqua, M. Czakon, M. Garzelli, A. van Hameren, A. Kardos, et al., HELAC-NLO, Comput.Phys.Commun. 184 (2013) 986–997. arXiv:1110.1499, doi:10.1016/j.cpc.2012.10.033.

[65] S. Badger, B. Biedermann, P. Uwer, V. Yundin, Numerical evaluation of virtual corrections to multi-jet production in massless QCD, Comput.Phys.Commun. 184 (2013) 1981–1998. arXiv:1209.0100, doi:10.1016/j.cpc.2013.03.018.

[66] P. Draggiotis, M. Garzelli, C. Papadopoulos, R. Pittau, Feynman Rules for the Rational Part of the QCD 1-loop amplitudes, JHEP 0904 (2009) 072. arXiv:0903.0356, doi:10.1088/1126-6708/2009/04/072.

[67] M. Garzelli, I. Malamos, R. Pittau, Feynman rules for the rational part of the Electroweak 1-loop amplitudes, JHEP 1001 (2010) 040. arXiv:0910.3130, doi:10.1007/JHEP01(2010)040,10.1007/JHEP10(2010)097.

[68] M. Garzelli, I. Malamos, R. Pittau, Feynman rules for the rational part of the Electroweak 1-loop amplitudes in the $R_x i$ gauge and in the Unitary gauge, JHEP 1101 (2011) 029. arXiv:1009.4302, doi:10.1007/JHEP01(2011)029.

[69] H.-S. Shao, Y.-J. Zhang, Feynman Rules for the Rational Part of One-loop QCD Corrections in the MSSM, JHEP 1206 (2012) 112. arXiv:1205.1273, doi:10.1007/s13130-012-4240-2.