– Modern ARtificial Theoretical phYsicist A C++ framework automating theoretical calculations Beyond the Standard Model

Grégoire Uhlrich, Farvah Mahmoudi, Alexandre Arbey

Version of Record: https://www.sciencedirect.com/science/article/pii/S001046552100062X
Manuscript_6c0ca6b2f03c54be2e943abaad09cf1e

MARTY - *Modern ARtificial Theoretical phYsicist*

A C++ framework automating theoretical calculations Beyond the Standard Model

Grégoire Uhlrich[a,*], Farvah Mahmoudi[a,b,c], Alexandre Arbey[a,b,c]

[a] *Université de Lyon, Université Claude Bernard Lyon 1, CNRS/IN2P3, Institut de Physique des 2 Infinis de Lyon, UMR 5822, F-69622, Villeurbanne, France*

[b] *Theoretical Physics Department, CERN, CH-1211 Geneva 23, Switzerland*

[c] *Institut Universitaire de France, 103 boulevard Saint-Michel, 75005 Paris, France*

Abstract

Studies Beyond the Standard Model (BSM) will become more and more important in the near future with the rapidly increasing amount of data from different experiments around the world. The full study of BSM models is in general an extremely time-consuming task involving long and difficult calculations. It is in practice not possible to do exhaustive predictions in these models by hand. Here we present MARTY, a new C++ framework that fully automates calculations from the Lagrangian to physical quantities such as amplitudes or cross-sections. It can fully simplify, automatically and symbolically, physical quantities in a very large variety of models and compute Wilson coefficients in effective theories. This will considerably facilitate BSM studies in flavour physics. Contrary to the existing public codes in this field MARTY aims at providing a unique, free, open-source, powerful and user-friendly tool for high-energy physicists studying predictive BSM models, in effective or full theories up to the one-loop level, which does not rely on any external package. With a few lines of code one can gather final expressions that may be evaluated numerically for statistical analysis.

Keywords: BSM, 1-loop calculation, Automated, Wilson coefficient

*Corresponding author.
E-mail address: g.uhlrich@ipnl.in2p3.fr

PROGRAM SUMMARY

Program Title: MARTY
Website: `https://marty.in2p3.fr`
https://fr.overleaf.com/project/5fb7c30c3175f910d2392059 *CPC Library link to program files:* (to be added by Technical Editor)
Code Ocean capsule: (to be added by Technical Editor)
Licensing provisions: GPLv3
Programming language: C++
Supplementary material: GNU-GSL [1], LoopTools [2]
Nature of problem: From a given Lagrangian, Beyond the Standard Model phenomenology often requires one-loop theoretical calculations. Those calculations must be done analytically because a numerical evaluation of such quantities is not possible, the number of terms being too large for any computer to handle. Simplifications must be done analytically, and are very long and error prone justifying to automate them. This is why we need a computer algebra system dedicated to this task. Some packages written with Mathematica [3] - a private and closed software for symbolic manipulations - implement those calculations. There are multiple packages, the user needs to pay for Mathematica and the calculation of Wilson coefficients for BSM and at the one-loop level is actually hard to automate.
Solution method: MARTY is a code written for BSM phenomenology that comes with its own computer algebra system, *CSL*, and automates the calculation of theoretical quantities at the one-loop level, in a very large variety of BSM model. Physicists now can have a unique C++ code, free and open-source, that can be used to do model building BSM and calculate any kind of amplitude, squared amplitude or Wilson coefficient BSM from the Lagrangian. From the developers point of view, having such a code, independent of any other framework using only the C++ standard library (C++17 standard), is a great opportunity to develop it even further, implementing new simplifications procedures, new calculation procedures and more.

References

[1] GSL - GNU Scientific Library. https://www.gnu.org/software/gsl/.

[2] T. Hahn, M. Perez-Victoria, Automatized one loop calculations in four-dimensions and D-dimensions, Comput. Phys. Commun. 118 (1999) 153–165. arXiv:hep-ph/9807565, doi:10.1016/S0010-4655(98)00173-8.

[3] Wolfram Research Inc., Mathematica. https://www.wolfram.com/mathematic.

1. Introduction

Calculations Beyond the Standard Model (BSM) have been a challenge for a long time, especially at the one-loop level. Transition amplitudes, differential cross-sections, and Wilson coefficients need to be evaluated in order to obtain the values for physical observables. Comparing them to the SM predictions and experimental results provides means to discriminate BSM models.

The one-loop order is often needed, as some relevant processes only appear at this order. This is the case of Flavour Changing Neutral Currents (FCNC) in flavour physics [1], that require one-loop values for Wilson coefficients. At the one-loop level, calculations cannot be done numerically as the very large number of terms together with renormalization processes require symbolic computations similar to what can be done by hand.

Currently, symbolic calculations at the one-loop level are done mainly using Mathematica [2], a commercial and closed computer algebra system. Several packages based on Mathematica implement high-energy physics calculations. FeynRules [3] computes Feynman rules from a BSM Lagrangian, from which FormCalc [4] can derive tree-level and one-loop quantities such as transition amplitudes or differential cross-sections. Packages such as FormFlavor [5] or FlavorKit [6, 7] calculate Wilson coefficients using the FormCalc machinery.

Many efforts have been employed to design independent computer algebra systems for high-energy physics. GiNaC [8] is a C++ library for symbolic computations written for that purpose. LanHEP [9], CompHEP [10] and CalcHEP [11] automate together calculations from the Lagrangian using their own symbolic computation framework. However, Mathematica-based packages are still the only ones able to provide one-loop calculations and Wilson coefficients for BSM.

We introduce *MARTY*, a modern and user-friendly solution to this issue. It provides for the first time a unique, free and open-source code fully written in modern C++ (2017 standard), implementing all the theoretical BSM machinery up to the one-loop order. Amplitudes, cross-sections and Wilson coefficients up to dimension 5 operators can be computed automatically. Support for dimension 6 operators will come in a future version. *MARTY* does not depend on any other framework, as it includes its own C++ Symbolic computation Library called *CSL* to manipulate mathematical expressions. It also comes with *GRAFED*, a desktop application displaying Feynman dia-

grams on screen.

The reader can find the code, manuals, the documentation and more information on `https://marty.in2p3.fr`.

2. Code overview

MARTY is organised in several modules. They have been logically separated to ensure the code to be modular and general. The results are three modules: the physics core of *MARTY*, and two other modules which are both fully independent from each other and can be used as standalone, namely the computer algebra system of *MARTY*(*CSL*) and the application drawing Feynman diagrams (*GRAFED*). A schematic view of *MARTY*'s design is presented in figure 1.

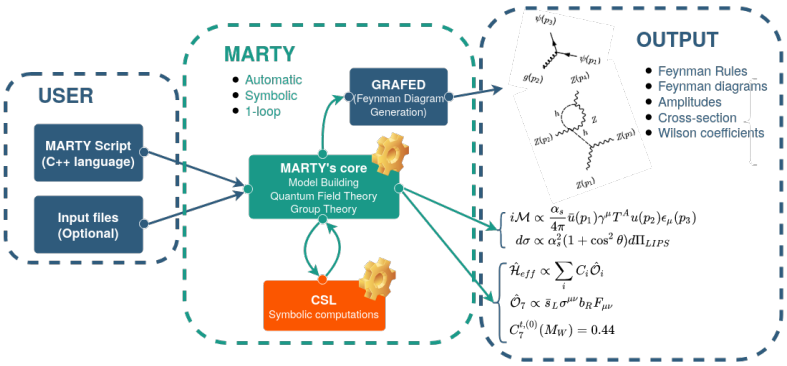

Figure 1: Schematic design of *MARTY*, including the three sub-modules: *MARTY* (physics part), *CSL* (symbolic manipulation) and *GRAFED* (Feynman diagram generation).

2.1. MARTY *(physics core)*

This is the main part of the code, which contains all physics calculations, conventions, models, simplifications, etc. It uses its own *CSL* module as a mathematical backend to perform calculations specific to high-energy physics and the *GRAFED* module to draw Feynman diagrams. It also contains all group theory implementations, and model building features. Amplitudes, differential partonic cross-sections and Wilson coefficient may be calculated at tree-level or at the one-loop order. All calculations are automatic, symbolic, and can be performed in a very large variety of models as detailed in section 5.

2.2. CSL *(computer algebra system)*

This module does not know anything about physics, is logically separated from the physics part and can be used independently. It is a C++ Symbolic computation Library allowing us to handle mathematical expressions, tensors and simplifications needed to perform high-energy physics calculations. It is not as comprehensive as a standard computer algebra system like Mathematica because many features were not required for particle physics. It may however be extended in that direction if needed.

2.3. GRAFED *(Feynman diagram generation and edition)*

When doing calculations in particle physics, it is often convenient to visualize what the code is doing, and possibly include diagrams in publications. *GRAFED* was developed for this purpose and is also fully independent of the other modules of *MARTY*. It has three major features:

- An algorithm that finds an optimal way to place nodes in a 2D space to display Feynman diagrams. This algorithm is fully general (with no limit in the diagram size or number of loops) and automated. This allows one to quickly draw all diagrams for a particular process, without asking anything from the user and independently of the diagram topologies.

- A Graphical User Interface (GUI) that displays the generated diagrams. When asked, *MARTY* will run *GRAFED* with all the diagrams of a particular process. These diagrams appear then in the GUI, and may be exported (as png files or LATEX codes for the tikz-feynman package) directly to be included in a publication for example.

- The possibility to edit or create diagrams from scratch. Diagrams generated automatically by *GRAFED* are rather neat, but there is the possibility to edit, graphically, any aspects of the diagram (nodes, edges, labels, layout, etc) very easily. One can also create diagrams from scratch using *GRAFED* independently of *MARTY*.

3. *MARTY* design philosophy

The design of *MARTY* is guided by strong principles ensuring a final result corresponding to programming standards. First, the general principles unrelated to physics are:

- **Independence.** *MARTY* is written from scratch and is thus fully independent of any other framework. As such, there is no limit in what can be implemented in the code. Developers have a full control on any aspects of *MARTY*, to modify or extend its capabilities.

- **User-friendliness.** The code must be easy to use. The fact that it is written in C++ is a supplementary challenge in that purpose, but a modern knowledge of this language provides freedom for the user-interface. We think that this objective is fulfilled, since the normal usage of *MARTY* does not require any particular C++ knowledge and would be similar in many languages, including Mathematica.

- **Modularity.** *MARTY* is built as modular as possible. This means that unnecessary logical connections between different parts of the code are avoided. This is an important advantage for maintainability, since replacing or correcting a part of the code will become simpler.

- **Readability.** It is important for a code to be easily understandable by everyone, and in particular by a user willing to further develop the code. Strict coding conventions, clear naming for files / functions / variables and clever separation of different logical units make *MARTY* easy to understand considering its large size.

- **Performance.** C++ and python were the two main languages possible for *MARTY* as they are well-known in the high energy physics community. The choice of the language, C++, is related to performance reasons. A C++ code will run much faster in average than python for this type of code.

Concerning physics aspects, *MARTY* has been written with the following aims in mind:

- **Generality.** *MARTY* is designed to be as general as possible in the models it can handle, algebraic simplifications it can do, and calculations automated with it. A high level of generality has already been reached, and further developments will continue to focus on this aspect. In particular there is no hard-coding because *MARTY* is expected to be extended even further in its future developments.

- **Model independent calculations.** In order to have an easy-to-use code for high energy physics, the computations have to be done in a

model-independent way. The same code computing a given quantity should work for all models. Studying new models would hence imply only to write the Lagrangian or Feynman rules associated to it, and then using the same scripts to calculate the same quantities in the given model.

- **One-loop level automated calculations.** Calculations in BSM phenomenology often require at least one-loop level quantities. Many processes are trivial at tree-level but higher order corrections can be important from a phenomenological point of view when studying BSM models, as it is the case for instance for FCNC decays in flavour physics. The one-loop level being significantly more difficult to calculate by hand than the tree-level, it is important to automate its calculation.

The efforts made to respect this philosophy will be useful in *MARTY*'s future developments as well. A code as general and independent as *MARTY* could benefit from a community effort to be maintained and developed and we think that the way it is written would allow for such a collaborative work.

4. Installation and usage

MARTY is available for download from its website: `https://marty.in2p3.fr` where one can find the manuals, documentation, publications and more information.

4.1. Installation

MARTY is open-source, GPL3 licenced and written in C++. By default, the installation script will use `gcc-7, g++-7, gfortran-7` to compile *MARTY* and its dependencies. In case another version is more suitable, one can write

```
$ export MARTY_COMPILER_VERSION=n
```

with `n` the version number (if `0` is given the default compilers will be used instead but this is not recommended).

On Ubuntu[1] / Debian since MARTY-1.0 (other Linux distributions may also be compatible but have not been tested) and on Mac since MARTY-1.1, the code can be built and installed using the following instructions:

```
$ source setup.sh <installation-path>
$ make
$ make install
```

The installation path is `/usr/local/` by default. Paths for header files, library files and binaries (the *GRAFED* application) must appear in environment variables in order to compile *MARTY* programs easily. This will be done by the `setup.sh` script for the current bash session, if *MARTY* is installed with the corresponding `make` target. This script must be executed using `source` to allow one to compile C++ programs using *MARTY* as presented in the following. Once the installation is performed, a `marty_env.sh` script is created in the root directory that contains the required bash instructions to set all environment variables. It can be executed at any time, again using `source`

```
$ source marty_env.sh
```

One can also copy its content in a `.bashrc` configuration file for example, to automate its execution for all bash sessions.

A procedure exists to use a *MARTY* or *CSL* program without installing it. A program.cpp file in `marty/scripts` can be compiled and executed typing from the main directory:

```
$ cd marty
$ make program.x
$ bin/program.x
```

[1] On Ubuntu 16.04 or prior versions, MARTY can be built properly only since version 1.2.

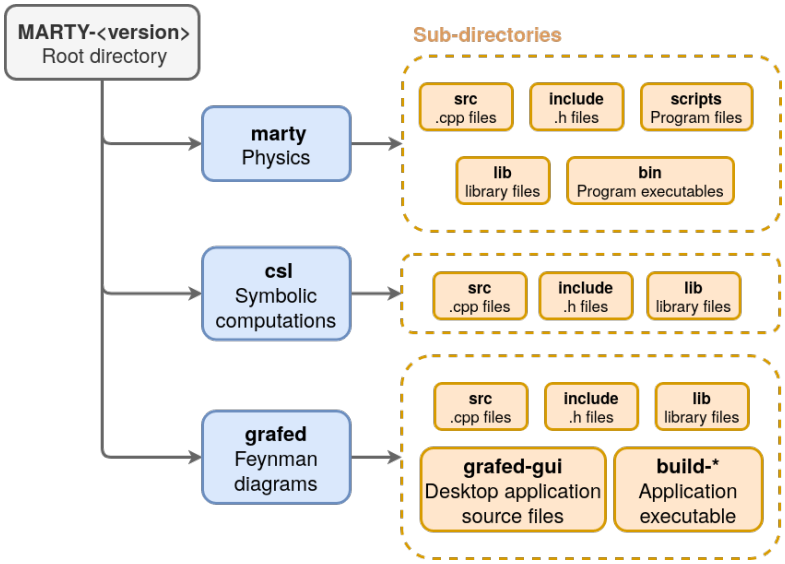

Figure 2: Layout of the MARTY project. The main sub-directories are shown with their content.

The project layout specific to the `marty/` directory together with the other modules (*CSL* and *GRAFED*) is presented in figure 2.

4.2. Usage (scripting)

As a C++ framework, *MARTY* can be used in a C++ program after writing a few lines to include it, provided it is installed on the computer[2]:

```
#include <marty.h>

using namespace mty;
using namespace csl;
using namespace std;
```

[2]For this to work properly, one needs either to install *MARTY* in the standard location (by default, /usr) or make sure that the paths to access include files in C++ (CPATH environment variable on Ubuntu), libraries (LIBRARY_PATH and LD_LIBRARY_PATH environment variable on Ubuntu) and binaries (PATH) contain the installation path of *MARTY*.

The three lines of `using namespace` are not necessary, but will allow us to omit prefixes `mty::` (*MARTY*) `csl::` (*CSL*) and `std::` (C++ standard library) in front of objects and functions. The main function containing the program can now be written:

```
int main() {
    // The MARTY program
    return 0;
}
```

4.3. Usage (compilation and execution)

In C++, source files need to be compiled before being run. *MARTY* uses the C++17 standard that appears in compiler options. To compile a source file `main.cpp` into an executable `main.x` the following commands are necessary[3]:

```
$ g++ -std=c++17 -c main.cpp -o main.o
$ g++ -std=c++17 -o main.x main.o -lmarty
$ ./main.x
```

4.4. Dependencies

MARTY has been written from scratch. Thus it has no dependency before the numerical evaluation of symbolic results. It contains in particular its own computer algebra system, *CSL* (C++ Symbolic computation Library), as a separate module. Therefore for the physics calculations from the Lagrangian to a final one-loop result simplified as much as possible, *MARTY* uses nothing but its own code and the C++ standard library.

For the numerical evaluation of results, there are two dependencies. The first, LoopTools [4], provides numerical values for scalar integrals arising at the one-loop level. Momentum integrals have always the same form, like the following 3-point function integral

$$I_3 \equiv \int \frac{d^4q}{i\pi^2} \frac{p_1^\mu p_2^\nu}{(q^2-m_0^2)((q+p_1)^2-m_1^2)((q+p_2)^2-m_2)^2}. \qquad (1)$$

[3] `g++` may be replaced by another C compiler, `clang` for example.

A way to treat this kind of integrals (including regularization) is to decompose the result in different possible Lorentz structures, each having a scalar factor in front, that is calculable with standard prescriptions [12]. The decomposition for the 3-point function I_3 is the following

$$I_3 \equiv C_{00} g^{\mu\nu} + C_{11} p_1^\mu p_1^\nu + C_{12}(p_1^\mu p_2^\nu + p_2^\mu p_1^\nu) + C_{22} p_2^\mu p_2^\nu, \tag{2}$$

with C_{ij} scalar form factors depending on masses and squared momenta in the loop. The decomposition is done analytically by *MARTY*, and the evaluation uses LoopTools functions to determine the values of C_{ij}.

The second dependency for the numerical evaluation is GSL [13], a well-known numerical library for C and C++. For complicated models with non trivial mixings (such as supersymmetric models), one has to diagonalize mass matrices to obtain the mass spectrum and mixings of the theory. For example from a non-diagonal squared-mass matrix M^2 of Φ, one calculates the eigenvector Φ' that diagonalizes the matrix to M_D^2

$$\Phi = U\Phi', \tag{3}$$

$$M_D^2 = U^\dagger M^2 U. \tag{4}$$

The symbolic result of *MARTY* is fully general (unless specified otherwise by the user) and uses generic symbols for all masses and mixings (matrices M_D^2 and U in the example). For the numerical evaluation, input parameters must be given by the user. The diagonalization is then performed to get the spectrum and mixings, and to calculate the final results. The numerical diagonalization is performed using GSL.

Finally, *GRAFED* has a Graphical User Interface (GUI) that uses Qt [14]. Qt is a C++ framework allowing to build desktop applications fairly easily and is free and open-source with a GPL licence.

5. *MARTY*'s capabilities

This section presents in detail the calculations that can be performed with *MARTY*, the possible models, and the outputs that the code returns to the user.

5.1. Model building

First it is important to have a clear view of the BSM models that *MARTY* can handle. A model lies in a 4-dimensional Minkowski space-time and is defined by:

- **A gauge group.** The gauge group may be any combination of Semi-simple Lie Groups[4] ($U(1)$, $SU(N)$, $SO(N)$, $Sp(N)$, E_6, E_7, E_8, F_4, G_2). The unbroken Standard Model gauge $SU(3)_c \times SU(2)_L \times U(1)_Y$ is an example of such a combination.

- **A particle content.** Each particle is an irreducible representation of the gauge group, i.e. an irreducible representation of each group composing the gauge. A particle may have a spin 0, 1/2 (Weyl, Dirac or Majorana), or 1. All gauge couplings are introduced automatically by *MARTY* without any help from the user.

- **Additional couplings.** The user can add any interaction term in the Lagrangian. *MARTY* simply checks that combining the unbroken gauge representations of the interacting particles gives indeed a trivial representation[5].

There are two ways to build a model in *MARTY*. The first one is the most straightforward way but also the most complicated one. It consists in giving explicitly the full Lagrangian to *MARTY*. Few terms in general are provided by unbroken gauge couplings, in particular when one studies a phenomenological model extending the SM. In the SM, there are about 100 terms to write by hand coming from the symmetry breaking. In the Minimal Supersymmetric extension of the Standard Model (MSSM), several thousands. It is possible to do it but one has to be very careful on every convention, sign and factor in front of each term. A small error can lead to wrong results due to interference between different diagrams for a given process.

The second option is to define a high energy Lagrangian with all symmetries preserved, and give *MARTY* prescriptions to break it. The initial Lagrangian has much less interaction terms and is simpler to write. Based on correct prescriptions (gauge, flavour symmetry breaking, replacements, renaming, etc) *MARTY* will basically re-derive the final Lagrangian for the user. This solution will not necessarily be the easiest one depending on the model but is certainly a practical option. It is in particular the way chosen to build the MSSM in *MARTY*.

[4]Strictly speaking, groups that have a semi-simple Lie algebra.

[5]If it does not, this is the sign of an obvious gauge violating term. If it does, the term may still violate gauge symmetry but it is more difficult to test automatically.

In the following we present a sample code building a $SU(2)_L$ gauge with one quark in the doublet representation

$$Q_L = \begin{pmatrix} u_L & d_L \end{pmatrix}, \tag{5}$$

broken by *MARTY* with a single instruction. We also ask the code to rename the broken fermions Q_1 and Q_2 to u and d which corresponds to standard conventions. With 3 lines of breaking prescriptions *MARTY* derives the 17 interaction terms (including vector-ghost interactions[6]) between the final 8 particles in the model.

```
Model model;
model.addGaugedGroup(group::Type::SU, "L", 2); // Adding a SU(2)
model.init();

Particle Q = weylfermion_s("Q", model, Chirality::Left);
Q->setGroupRep("L", {1}); // Doublet rep of SU(2), Dinkin label 1
model.addParticle(Q);

cout << "Before␣symmetry␣breaking␣:␣" << endl;
cout << model << endl;

model.breakGaugeSymmetry("L");
model.renameParticle("Q_1", "u"); // Broken fields are named with
model.renameParticle("Q_2", "d"); // _1, _2 etc. by convention

cout << "After␣symmetry␣breaking␣:␣" << endl;
cout << model << endl;
```

For a more evolved model one needs more instructions to specify every conventions, but this method is still very practical and more efficient than giving the full Lagrangian by hand. For a gauge symmetry breaking, explicit expressions of gauge generators (T^A_{ij}, f^{ABC}) must be known and *MARTY* will not necessarily know them. For $SU(2)$ and $SU(3)$ SM gauge terms, the whole procedure is automated, but for other groups the user may have to define expressions for generators.

[6]Gauge fixing terms for ghosts are not taken into account here.

MARTY also contains built-in models that can be used directly for calculations: Scalar ϕ^3 theory, Scalar QED, QED, QCD, Electroweak model, Standard Model, 2 Higgs Doublet Models and Minimal Supersymmetric Standard Models (unconstrained and phenomenological).

5.2. Amplitudes

Transition amplitudes from an initial state to a final state noted $i\mathcal{M}(i \to f)$ are the basic quantities that *MARTY* is able to calculate. It uses the Lagrangian exponentiation as well as the Wick's theorem [15] to find all possible diagrams and derive their corresponding expressions. This step is fully general and has no limit in the diagram complexity or in the number of external legs. Amplitudes are then used to calculate differential cross-sections or to derive Wilson coefficients.

Once an analytical expression for a given diagram has been found, it needs to be simplified in several ways in order to obtain a numerical evaluation of the result. Simplification steps done by *MARTY* are the following:

Dirac algebra simplification. This includes calculation of traces in the Dirac space and simplifications in $\gamma-$matrix products, for particles of spin $1/2$ [16].

Group algebra simplification. Similarly to $\gamma-$matrices, algebra generators have to be simplified into amplitudes. Projection operators are used [17] and traces are calculated in all semi-simple groups [18]. The remaining colour structures that cannot be simplified are stored and factored from the rest of the amplitude, in dedicated abbreviations. For standard gauge groups and in particular for fundamental representations all possible terms will be simplified automatically[7].

Minkowski Index contraction. Minkowski indices are expanded and contracted as much as possible in D-dimensions to perform Dimensional Regularization (DREG) at the one-loop order. As in D-dimension $g^\mu_\mu = D$, one has to expand the whole diagram to gather all factors of D.

[7]Group simplifications are easy to implement in *MARTY* in case there are missing ones. The complicated part is to determine theoretically the simplification rules to apply.

Reduction of one-loop momentum integrals. A momentum integral at one-loop can be decomposed on the basis of scalar form factors [12]. These form factors depend on masses and momenta and can be provided by LoopTools [4] up to the rank 4 5-point functions, i.e. loops with five external legs and four momenta in the numerator. This is the actual limit for fully-simplified one-loop quantities in *MARTY*.

Dimensional Regularization. The form factors coming from one-loop integrals can have a divergent part that is regularized by taking the dimension $D = 4 - 2\epsilon$. In this case, integrals take the form

$$I \approx \frac{a}{\epsilon} + b + \mathcal{O}(\epsilon). \tag{6}$$

Factors of D coming from Minkowski index contractions must then be kept to determine the local terms they generate when they are multiplied by a divergent integral [19, 20]. For the scalar one-point function for example, we get the finite part

$$\text{Finite}(DA_0(m^2)) = \text{Finite}((4-2\epsilon)A_0(m^2)) = -2m^2 + 4 \cdot \text{Finite}(A_0(m^2)). \tag{7}$$

Equations of motion. For spin 1 particles, the equation of motion is simply

$$\epsilon_\mu(p) p^\mu = 0, \tag{8}$$

where $\epsilon(p)$ is the polarization 4-vector of the boson. For spin $1/2$ particles, the Dirac equation is applied. It reads

$$\not{p} u(p) = m u(p) \quad \text{for particles,} \tag{9}$$

$$\bar{v}(p) \not{p} = -m \bar{v}(p) \quad \text{for anti-particles.} \tag{10}$$

Factorization. Results are partially factored to compactify at most the final expressions. In particular, factorization by masses and momenta are performed.

Abbreviation. Abbreviations are introduced automatically by *MARTY* to lighten expressions and gain in execution time. All abbreviations used can be displayed by typing `DisplayAbbreviations()`.

Using the toy model presented in section 5.1, the calculation of an amplitude in *MARTY* is very simple. One has to give the order (`mty::Order::TreeLevel`

here), the model, and the field insertions. Let us consider the process $u\bar{u} \to d\bar{d}$ at tree-level, the following instruction is needed to run the computation:

```
auto res = ComputeAmplitude(
    Order::TreeLevel,
    model,
    {Incoming("u"), Incoming(AntiPart("u")),
     Outgoing("d"), Outgoing(AntiPart("d"))}
    );
```

The code is in C++, therefore variable types must be declared. However since C++11, the compiler does not need the user to specify the type returned by a function anymore. By typing `auto` the compiler is told to find the exact type itself. Then, `res.expressions` contains the different terms of the amplitude and `res.diagrams` the Feynman diagrams. To display expressions in standard output, show the diagrams in *GRAFED* and display the abbreviations introduced by *MARTY* the following three lines are necessary respectively:

```
Display(res);
Show(res);
DisplayAbbreviations();
```

MARTY's output is presented in the following, displaying diagrams in *GRAFED* as the screenshot in figure 3 shows. Only the index IDs have been changed to lighten the output:

```
0  :  1/8*Ab_0001*EXT_{k,\%eps_1,i,\%del_1,l,\%del_2,j,\%eps_2}
  *gamma_{+\%sigma,\%del_2,\%eps_1}*gamma_{\%sigma,\%del_1,\%eps_2}
1  :  1/8*Ab_0001*EXT_{k,\%eps_1,i,\%del_1,l,\%del_2,j,\%eps_2}
  *gamma_{+\%sigma,\%del_2,\%eps_1}*gamma_{\%sigma,\%del_1,\%eps_2}
2  :  -1/8*Ab_0002*EXT_{k,\%eps_1,i,\%del_1,l,\%del_2,5,\%eps_2}
  *gamma_{+\%sigma,\%del_1,\%eps_1}*gamma_{\%sigma,\%del_2,\%eps_2}

Total : 3 particle amplitudes.
```

The abbreviations read:

```
4 abbreviations:
Ab = g_L^2
EXT = d_{k,del}(p_4)*d_{i,gam}(p_3)^(*)
    *u_{l,beta}(p_2)^(*)*u_{j,alpha}(p_1)
Ab_0001 = i*g_L^2/s_13
Ab_0002 = i*g_L^2/s_12
```

The resulting amplitude can be read off form *MARTY*'s output, keeping a separate term for each diagram:

$$\begin{aligned} i\mathcal{M} = \frac{ig_L^2}{8}\Bigg(& \frac{1}{s_{13}}\bar{d}(p_4)\gamma^\mu u(p_1)\bar{u}(p_2)\gamma_\mu d(p_3) \\ & + \frac{1}{s_{13}}\bar{d}(p_4)\gamma^\mu u(p_1)\bar{u}(p_2)\gamma_\mu d(p_3) \\ & - \frac{1}{s_{12}}\bar{u}(p_2)\gamma^\mu u(p_1)\bar{d}(p_4)\gamma_\mu d(p_3)\Bigg), \end{aligned} \tag{11}$$

with $s_{ij} \equiv p_i \cdot p_j$.

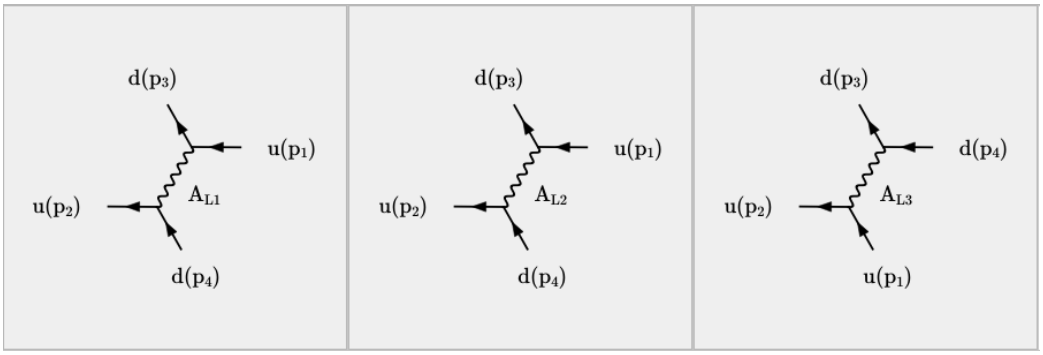

Figure 3: Screenshot of *GRAFED* displaying the three Feynman diagrams for the process $u\bar{u} \to d\bar{d}$ at the tree-level in the toy model defined in section 5.1. The tree diagrams correspond to the exchange of each broken vector from the $SU(2)_L$ initial symmetry, noted A_{Li} by default in *MARTY*.

5.3. Cross-sections

Cross-sections are the main observables used in collider physics. They are directly proportional to the number of events observed in the various detectors. *MARTY* does not compute directly the cross-sections but calculates the complicated theoretical part namely the squared amplitudes. For incoming particles $\{I\}$ of spins $\{j_I\}$ and outgoing particles $\{O\}$ of spins $\{j_O\}$ the squared amplitude is (as a function of the amplitude $i\mathcal{M}$ that depends on the particle spins)

$$\frac{1}{\prod_I d_I} \sum_{\{j_I\},\{j_O\}} |\mathcal{M}|^2, \tag{12}$$

with d_I the spin dimension of the incoming particle I taking into account massless effects for spin 1 particles. This quantity is averaged (summed) over the spin dimensions of incoming (outgoing) particles. Calculating the squared amplitudes implies the calculation of traces in Dirac and colour spaces (group algebra) that *MARTY* computes automatically. The result is a scalar depending on momenta and masses of particles in the process. The differential cross-section has always the same form for a given process of amplitude $i\mathcal{M}$

$$d\sigma \equiv K(p_i, m_i) \cdot \frac{1}{\prod_I d_I} \sum_{\{j_I\},\{j_O\}} |\mathcal{M}|^2 d\Pi_{LIPS}, \tag{13}$$

with $K(p_i, m_i)$ a factor coming from kinematics, and $d\Pi_{LIPS}$ the Lorentz Invariant Phase Space. Once the amplitude squared has been calculated and simplified, no more computer algebra system is needed to pursue the calculation. This is the quantity that *MARTY* can compute automatically.

Considering the toy model of section 5.1, calculating the squared amplitude is very simple. The user must first calculate the amplitude, and simply square it with *MARTY*. The average over incoming spins is done by *MARTY*, i.e. the returned quantity corresponds to equation 12. After calculating this quantity with *MARTY*, the user has again one single line to write:

```
Expr squared_ampl = model.computeSquaredAmplitude(res);
cout << "<|M|^2>␣=␣" << squared_ampl << endl;
DisplayAbbreviations();
```

`Expr` is the main variable type of *CSL*, internal representation of a symbolic mathematical expression. The output in terminal is presented in the following.

```
<|M|^2> = 1/4*s_14*s_23*(
           1/2*Ab_0001^(*)*Ab_0002 + Ab_0001*Ab_0001^(*)
         + 1/2*Ab_0001*Ab_0002^(*) + 1/4*Ab_0002*Ab_0002^(*))
Ab_0001 = i*g_L^2/s_13
Ab_0002 = i*g_L^2/s_12
```

One can see that abbreviations have been introduced by *MARTY*. They can be expanded and the result can be further factored by *CSL* typing

```
// Evaluate abbreviations
Evaluate(squared_ampl, eval::abbreviation);
// Factor the whole expression
DeepFactor(squared_ampl);
cout << "<|M|^2>␣=␣" << squared_ampl << endl;
```

In this way, one can obtain a compact result:

```
<|M|^2> = 1/16*g_L^4*(s_12^(-2) + 4*s_13^(-2) + 4/(s_12*s_13))
       *s_14*s_23
```

As one can see above, *MARTY*'s outputs contain scalar products of external momenta, namely $s_{ij} \equiv p_i \cdot p_j$. *MARTY* does not perform any kinematics for now, i.e. stops the simplification as in the output shown. This could be easily implemented in the future, for example introducing Mandelstam variables. However as it does not represent an important analytical challenge by hand and that it has no impact on the following numerical evaluation (see section 5.5), this part is for now left to the user.

5.4. Wilson coefficients

Wilson coefficients are the coefficients in front of particular operator structures in an amplitude [1]. For the $b \to s\gamma$ process that will be detailed in

section 6, the amplitude may be decomposed on a two operator basis, each one with a scalar coefficient in front. Naming q the photon momentum and ϵ its polarization vector, one obtains

$$i\mathcal{M}(b \to s\gamma) = \frac{-4G_F}{\sqrt{2}} \frac{e}{16\pi^2} V_{tb} V_{ts}^* m_b \left(C_7 \langle \hat{\mathcal{O}}_7 \rangle + C_7' \langle \hat{\mathcal{O}}_7' \rangle \right), \tag{14}$$

with

$$\langle \hat{\mathcal{O}}_7^{(\prime)} \rangle \equiv \langle s\gamma | \hat{\mathcal{O}}_7^{(\prime)} | b \rangle = \bar{s} \sigma^{\mu\nu} P_{R(L)} b F_{\mu\nu}, \tag{15}$$

$$\sigma_{\mu\nu} = \frac{i}{2} \left[\gamma_\mu, \gamma_\nu \right], \tag{16}$$

$$F_{\mu\nu} = i q_{[\mu} \epsilon_{\nu]} = \frac{i}{2} \left(q_\mu \epsilon_\nu - q_\nu \epsilon_\mu \right). \tag{17}$$

The global factor $\frac{-4G_F}{\sqrt{2}} \frac{e}{16\pi^2} V_{tb} V_{ts}^* m_b$ is defined by convention. This procedure to decompose amplitudes in Wilson coefficients and operator matrix elements is used in particular in flavour physics. As quarks appear only in bound states, the partonic amplitude is not the full story. One has to take into account long-distance effects that cannot be calculated pertubatively. The $b \to s\gamma$ transition may correspond for example to a hadronic process $\bar{B}^0 \to \bar{K}^0 \gamma$. These long-distance effects are model-independent and arise only in the operator matrix element between final and initial states $\langle F | \hat{\mathcal{O}} | I \rangle$. The BSM dependence is then contained in the Wilson coefficient that can be calculated perturbatively by *MARTY*.

For now, operators of dimension 6 with 4 fermions cannot be directly given by *MARTY* at one-loop as some simplifications are needed that are not yet implemented. The step missing is a double application of Fiertz identities, to simplify all possible momenta in fermion currents. A Wilson coefficient for such an operator could still be read off in the results but would ask the user to do some algebra by hand, to determine which part of the amplitude contributes to the coefficient.

A concrete example of Wilson coefficient calculation is presented in section 6 which is devoted to the calculation of C_7 in the MSSM.

5.5. Library generation

Results of *MARTY* can in general not be used directly. Depending on the process, it may be a very big and complicated analytical expression. What the user may need are numbers, i.e. numerical evaluations of the analytical

results, for a given set of values of the model parameters. Let us consider the cross-section of section 5.3. The result is rather simple, but the principle would be exactly the same for a more complicated expression. The way it works in *MARTY* is also rather simple and is contained in a few lines (giving a library name and a path to create it):

```
mty::Library myLib("uubar_to_ddbar", ".");
myLib.addFunction("squared_ampl", squared_ampl);
myLib.build();
```

A `mty::Library` is an abstract object that takes symbolic expressions as functions (here the cross-section), creates and compiles a C++ library allowing to evaluate them numerically. The function generated by *MARTY* is:

```
complex_t squared_ampl(
    const complex_t g_L,
    const complex_t s_12,
    const complex_t s_13,
    const complex_t s_14,
    const complex_t s_23
    )
{
    return 0.0625*std::pow(g_L, 4)*(std::pow(s_12, -2)
        + 4*std::pow(s_13, -2) + 4/(s_12*s_13))*s_14*s_23;
}
```

The function takes as arguments all symbols (possibly complex here) that did not contain any value at the time the library was generated. The library is compiled automatically and can be used as demonstrated in the following.

```
#include "uubar_to_ddbar.h"
using namespace std;
using namespace uubar_to_ddbar;
```

```
int main() {
    cout << "XSec␣=␣" << squared_ampl(0.1, 100, 60, 40, 40) << endl;
    return 0;
}
```

A library may contain as many functions as wanted. This procedure is fully general and is automated. Note that if the library needs additional include or library paths (in particular if *CSL* and *MARTY* are not installed in standard locations), it is possible to specify them with:

```
myLib.addIPath("/home/.local/include");
myLib.addLPath("/home/.local/lib");
```

5.6. Feynman diagrams

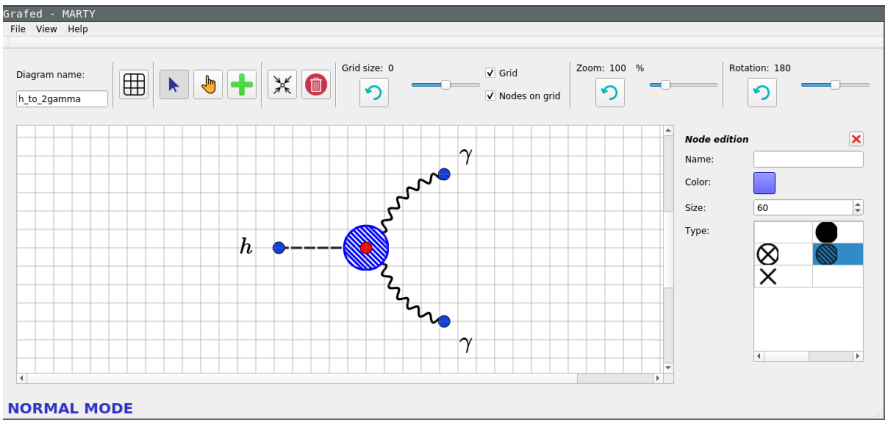

Figure 4: Generic $h \to \gamma\gamma$ diagram creation with *GRAFED*. Node colour / type / name / size, edge colour / type / curve / thickness can be edited easily with the application to create any kind of diagram. Diagrams may then be exported in a .png file directly.

GRAFED is the part of *MARTY* generating and rendering Feynman diagrams. It is used to create automatically diagrams when calculating a process with the `Show(res)` command as we discussed in section 5.2. It can also be used to edit or create diagrams from scratch. Many aspects of the

diagrams can be chosen by the user in an intuitive way. A screenshot of *GRAFED* is shown in figure 4.

GRAFED will in the future be released as standalone. All diagrams in this publication are generated automatically or created with *GRAFED*.

6. Calculation of $\delta^{LO}C_7^{\chi,\tilde{t}}(M_W)$ in the pMSSM

An example of *MARTY*'s capabilities is presented in this section, namely the calculation of the MSSM contribution to the Wilson coefficient C_7. This coefficient describes the $b \to s\gamma$ transition and is non zero only at the one-loop level.

6.1. The pMSSM

We consider here the phenomenological Minimal Supersymmetric Standard Model (pMSSM) which is a generic CP conserving MSSM framework that has 19 parameters more than in the SM as opposed to the full MSSM which has 105 extra parameters. The pMSSM has been chosen for validation because of its complexity and generality. Obtaining a correct result in this model demonstrates *MARTY*'s capabilities for model building and symbolic calculations.

6.2. The Wilson coefficient C_7

We calculate the Leading Order (LO) value of the Wilson coefficient C_7, associated with the operator in equation 15. The process is showed in figure 5. It is a FCNC process with a photon changing the quark flavour $b \to s$ which is forbidden at tree-level in the SM and the pMSSM, and the LO is thus at the one-loop level. Strong experimental constraints exist for FCNCs and their calculations for BSM models represent an important task for phenomenology.

We consider in this example one of the supersymmetric contributions i.e. diagrams with the top squarks and charginos shown in figure 6. We perform the calculation on-shell in the Feynman-'t Hooft gauge[8]. The reversal of the fermion-flow in the diagram is due to fermion-number violating interactions between charginos and SM fermions. This is mostly related to the definition of charginos and may be treated following prescriptions of [21]. At the end of the calculation, the fermion flow is regular but may get a sign due to charge

[8]Other gauges can be used such as the unitary or Lorentz gauges.

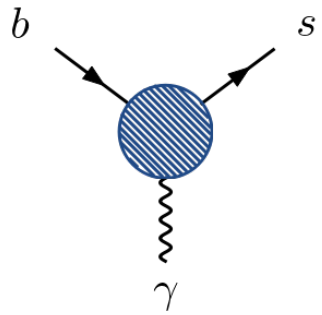

Figure 5: $b \to s\gamma$ process represented in a model-independent way. The transition amplitude is the sum of all diagrams that can fill correctly the hatched disk. This diagram has been built using *GRAFED*.

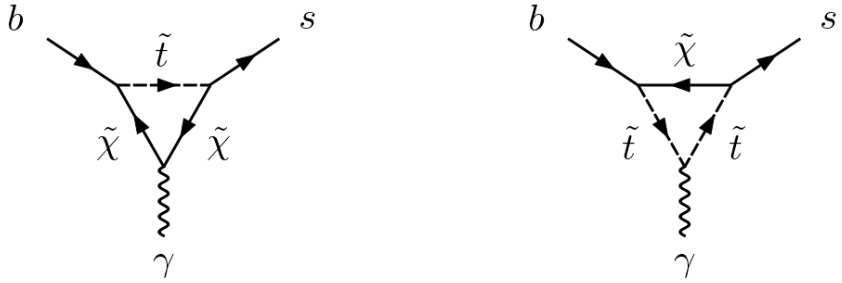

Figure 6: Two types of contribution for C_7 in the pMSSM, with stops $\tilde{t}$ and charginos $\tilde{\chi}$. These diagrams have been built using *GRAFED*.

conjugation matrix C which appears. This sign is important to determine exactly because of interferences between the diagrams.

We vary two pMSSM parameters, μ (the Higgsino parameter) and M_2 (the Wino mass). More details on MSSM parameters are given in [22]. Contributions to C_7 come from chargino and stop loops and depend on μ and M_2 in particular. The chargino mass matrix reads

$$M_\chi = \begin{pmatrix} 0 & X^T \\ X & 0 \end{pmatrix}, \tag{18}$$

with

$$X = \begin{pmatrix} M_2 & \sqrt{2}\sin\beta M_W \\ \sqrt{2}\cos\beta M_W & \mu \end{pmatrix}, \tag{19}$$

$\tan\beta$ being the angle between the two Higgs doublets' Vacuum Expectation Values (VEVs).

The stop squared mass matrix reads

$$M_{\tilde{t}}^2 = \begin{pmatrix} m_{Q_3}^2 + m_t^2 + \Delta_{\tilde{u}_L} & v(A_t^* \sin\beta - \mu y_t \cos\beta) \\ v(A_t \sin\beta - \mu^* y_t \cos\beta) & m_{u_3}^2 + m_t^2 + \Delta_{\tilde{u}_R} \end{pmatrix}. \tag{20}$$

m_{Q_3}, m_{u_3} are soft supersymmetry breaking parameters, A_t is a trilinear coupling, y_t the top Yukawa and finally

$$\Delta_{\tilde{u}_L} = \left(\frac{1}{2} - \frac{2}{3}\sin^2\theta_W\right)\cos(2\beta)M_Z^2, \tag{21}$$

$$\Delta_{\tilde{u}_R} = \frac{2}{3}\sin^2\theta_W \cos(2\beta)M_Z^2. \tag{22}$$

The exact numerical values of pMSSM parameters used to evaluate C_7 are presented in table 1.

Parameter	Value
A_t	500
m_{Q_3}	1000 GeV
m_{u_3}	1000 GeV
$\tan\beta$	50
μ	$[-800, 800]$ GeV
M_2	$[-1000, 1000]$ GeV

Table 1: Numerical values of supersymmetric parameters used to evaluate C_7. M_2 and μ are varied in the given ranges. Other pMSSM parameters are irrelevant for the calculation presented here.

The results are shown in figures 7 and 8. *MARTY*'s output is compared with the analytical formula given in [23] and with SuperIso [24, 25, 26]. Numerical evaluations have been done for two different spectra. This first one (figure 7) is a tree-level spectrum computed by *MARTY* using GSL [13], and the result is compared with the analytical formula. The second spectrum (figure 8) is calculated by SOFTSUSY [27, 28] with two-loop order corrections which are known to be important for the charginos [22]. For this spectrum, we compare *MARTY* with the output of SuperIso.

As can be seen for all the results there exists an excellent agreement between the analytical formula, SuperIso, and *MARTY*. The agreement is up to 4 digits. In addition, we also tested the results given by FormCalc [4] for the same process, and there is in this case a perfect agreement with *MARTY*

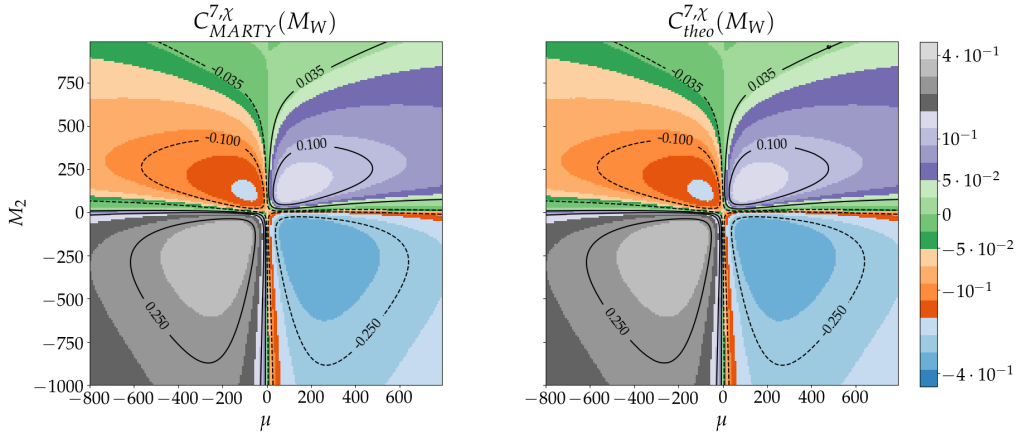

Figure 7: Results for C_7 (chargino and stop contributions) in the pMSSM, from *MARTY* one the left and from the analytical formula [23] on the right, for the spectrum generated by *MARTY* at tree-level. The results match to four digits in average.

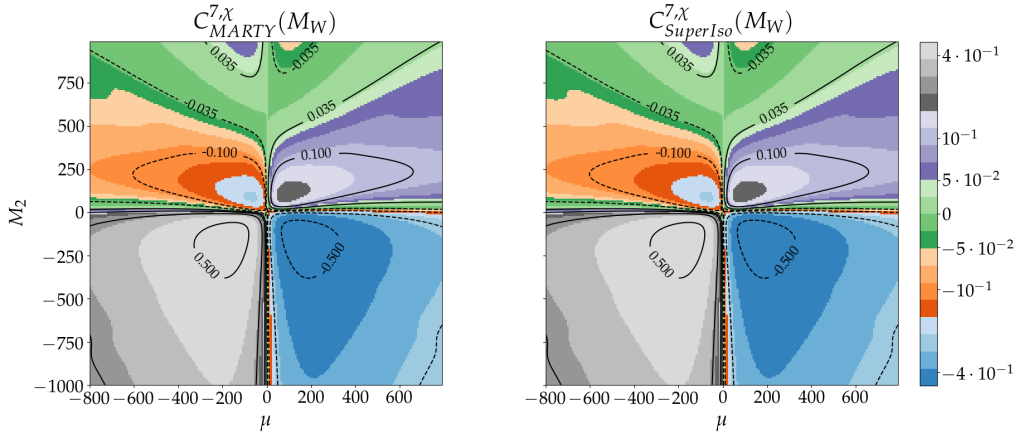

Figure 8: Results for C_7 (chargino and stop contributions) in the pMSSM, from *MARTY* one the left and from the output of SuperIso [25] on the right, for the spectrum generated by SOFTSUSY [27, 28] with two-loop corrections. The results match to four digits in average.

with 10 identical digits in average, for both spectra. One explanation may be that we used quadruple precision (128 bits) floating point variables for FormCalc and *MARTY*'s outputs, and only double precision (64 bits) for SuperIso output and the analytical formula. The 4-digits precision is however completely satisfactory considering the uncertainty coming form higher

orders in perturbation theory.

This example completes the presentation of what *MARTY* can calculate. *MARTY* will generate for any process, in any model, libraries evaluating theoretical quantities and give the user a spectrum generator at the same time. In the case of supersymmetry, spectrum generators already exist with higher-order terms but in a general BSM models one needs this generic tree-level spectrum generator. More information and example can be found on the website `https://marty.in2p3.fr`.

7. Performance

We measure the performance of a computer program with two main indicators. The execution speed and the quantity of memory (RAM) the program needs for running. For BSM symbolic calculations at one-loop, it is not possible to give a standard execution speed nor the quantity of memory as it depends on the model and the process to calculate. The amount of memory taken by *MARTY* is typically very small. It is very rare to reach 1 GB, and is often under 100 MB. Indicative values of execution times are shown in table 2, measured on various processes, always running on a single CPU.

External legs	Tree-level	One-loop
2	$\leq 10^{-1}$	10^{-2}
3	$\leq 10^{-1}$	$10^{0}/10^{1}$
4	$\leq 10^{-1}$	10^{2}

Table 2: Typical execution time (order of magnitude) of *MARTY* in second per 100 Feynman diagrams during the calculation of an amplitude, for different numbers of external legs.

It can be seen in table 2 that the calculation complexity depends strongly on the number of legs at one-loop. The more legs there is for the loop, the more terms appear in the amplitude. This number of terms grows very fast with the number of legs connected to the loop and explains the results shown here.

For squared amplitudes there is no simple rule to determine the execution time but it is in general several orders of magnitude more than the simple amplitude calculation as squaring the amplitude also squares the number of terms to simplify. Improving performance for this calculation is an important development for the next release of *MARTY*.

8. Future developments

MARTY has fulfilled most of the planned requirements, but further developments are ongoing which are listed in the following.

- **Wilson coefficients for 4-fermion operators.** For now, *MARTY* can compute amplitudes for 4-fermions processes, but cannot give automatically the corresponding Wilson coefficients because of a missing simplification step. This step is the double application of Fiertz identities to simplify all momenta in quark currents. Once this simplification will be implemented, 4-quarks operators will become available in *MARTY*.

- **More group theory simplifications.** All simplifications with algebra generators are not implemented in *MARTY*. Some are missing because it is very difficult to automate these identities for all semi-simple groups and all representations. These missing simplifications concern mostly non-fundamental representations, exceptional algebras and squared amplitudes for pure gluonic amplitudes. Further developments will focus on this issue but the user can also define easily the missing properties.

- **Automated NLO corrections**. With *MARTY* one can calculate all one-loop quantities needed to renormalize a BSM model. However this procedure is not automated and will surely be a point of attention in the future.

- **Operator mixing for Wilson coefficients.** Renormalization comes with operator mixing for Wilson coefficients. This task is more challenging but there is currently no code able to fully automate this procedure for general BSM. Therefore having a code able to do this task would be very useful for flavour physics.

- **Interfaces with other codes.** This publication presents the first version of *MARTY*, that is at the moment not interfaced with other codes. This is the most important improvement that is planned for next developments. Universal Feynman Rules Output (UFO [29]) will link *MARTY* directly to event generators, and work is in progress to have direct interface with SuperIso [24, 25, 26] for flavour physics, and SuperIso Relic [30, 31, 32] for dark matter phenomenology.

9. Conclusion and Outlook

We presented *MARTY*, a new C++ framework automating theoretical calculations symbolically for BSM physics. The degree of generality reached by *MARTY* has never been achieved before. It has its own computer algebra system (*CSL*) and automates all theoretical calculations directly from the Lagrangian. Feynman rules, Feynman diagrams, amplitudes, cross-sections, and Wilson coefficients can be obtained in a very large variety of BSM models up to the one-loop level. A full NLO treatment will also be implemented in the near future, treating renormalization of fields, masses, couplings, and Wilson coefficients (including operator mixings).

A proof of its capabilities has been demonstrated through a tree-level cross-section calculation, and a one-loop Wilson coefficient in the pMSSM. The results are at first symbolic mathematical expressions, but numerical C++ libraries are built automatically by *MARTY* allowing us to explore in full generality the parameter space of the model for some user-defined quantities. A spectrum generator specific to the user's model is also created automatically by *MARTY* when needed. Most of popular BSM models can be built in *MARTY*. The MSSM, extended gauge models, and vector-like quarks are examples of possible BSM implementations.

MARTY can already be very useful for BSM phenomenology in this current version. The particular advantage of *MARTY* is to be written as a unique code, not depending on any framework. Within *MARTY*, every aspect of model building and high-energy physics calculation are under control, in the same program and in the same language. This is a unique opportunity for future collaborations to take this code even further, extending it to new models, other simplification methods, or even different types of calculations.

References

[1] A. J. Buras, Weak Hamiltonian, CP Violation and Rare Decays (1998). `arXiv:hep-ph/9806471`.

[2] Wolfram Research Inc., Mathematica.
URL `https://www.wolfram.com/mathematica`

[3] A. Alloul, N. D. Christensen, C. Degrande, C. Duhr, B. Fuks, FeynRules 2.0 - A complete toolbox for tree-level phenomenology, Comput. Phys. Commun. 185 (2014) 2250–2300. `arXiv:1310.1921`.

[4] T. Hahn, M. Perez-Victoria, Automatized one loop calculations in four-dimensions and D-dimensions, Comput. Phys. Commun. 118 (1999) 153–165. arXiv:hep-ph/9807565.

[5] J. A. Evans, D. Shih, FormFlavor Manual, (2016). arXiv:1606.00003.

[6] F. Staub, Exploring new models in all detail with SARAH, Adv. High Energy Phys. 2015 (2015) 840780. arXiv:1503.04200.

[7] W. Porod, F. Staub, A. Vicente, A Flavor Kit for BSM models, Eur. Phys. J. C 74 (8) (2014) 2992. arXiv:1405.1434.

[8] R. B. K. Christian Bauer, Alexander Frink, Ginac (2001).
URL https://www.ginac.de/

[9] A. Semenov, LanHEP: A Package for the automatic generation of Feynman rules in field theory. Version 3.0, Comput. Phys. Commun. 180 (2009) 431–454. arXiv:0805.0555.

[10] E. Boos, V. Bunichev, M. Dubinin, L. Dudko, V. Ilyin, A. Kryukov, V. Edneral, V. Savrin, A. Semenov, A. Sherstnev, CompHEP 4.4: Automatic computations from Lagrangians to events, Nucl. Instrum. Meth. A 534 (2004) 250–259. arXiv:hep-ph/0403113.

[11] A. Pukhov, E. Boos, M. Dubinin, V. Edneral, V. Ilyin, D. Kovalenko, A. Kryukov, V. Savrin, S. Shichanin, A. Semenov, CompHEP: A Package for evaluation of Feynman diagrams and integration over multiparticle phase space (1999). arXiv:hep-ph/9908288.

[12] R. K. Ellis, Z. Kunszt, K. Melnikov, G. Zanderighi, One-loop calculations in quantum field theory: From feynman diagrams to unitarity cuts, Physics Reports 518 (4-5) (2012) 141–250.

[13] Project GNU, Gnu scientific library (gsl).
URL https://www.gnu.org/software/gsl/

[14] The Qt Company, Qt open source model (2020).
URL https://www.qt.io/

[15] G. C. Wick, The evaluation of the collision matrix, Phys. Rev. 80 (1950) 268–272.

[16] M. D. Schwartz, Quantum Field Theory and the Standard Model, Cambridge University Press, 2014.

[17] P. Cvitanovic, Group theory for Feynman diagrams in non-Abelian gauge theories, Phys. Rev. D 14 (1976) 1536–1553.

[18] T. van Ritbergen, A. Schellekens, J. Vermaseren, Group theory factors for Feynman diagrams, Int. J. Mod. Phys. A 14 (1999) 41–96. arXiv:hep-ph/9802376.

[19] A. Denner, Techniques for the calculation of electroweak radiative corrections at the one-loop level and results for W-physics at LEP200 (2007). arXiv:0709.1075.

[20] G. Sulyok, A closed expression for the uv-divergent parts of one-loop tensor integrals in dimensional regularization, Physics of Particles and Nuclei Letters 14 (4) (2017) 631–643.

[21] A. Denner, H. Eck, O. Hahn, J. Kublbeck, Compact Feynman rules for Majorana fermions, Phys. Lett. B 291 (1992) 278–280.

[22] S. P. Martin, A Supersymmetry primer, Adv. Ser. Direct. High Energy Phys. 21 (2010) 1–153. arXiv:hep-ph/9709356.

[23] M. Ciuchini, G. Degrassi, P. Gambino, G. Giudice, Next-to-leading QCD corrections to $B \to X_s\gamma$ in supersymmetry, Nuclear Physics B 534 (1-2) (1998) 3–20.

[24] F. Mahmoudi, SuperIso: A Program for calculating the isospin asymmetry of $B \to K^*\gamma$ in the MSSM, Comput. Phys. Commun. 178 (2008) 745–754. arXiv:0710.2067.

[25] F. Mahmoudi, SuperIso v2.3: A Program for calculating flavor physics observables in Supersymmetry, Comput. Phys. Commun. 180 (2009) 1579–1613. arXiv:0808.3144.

[26] F. Mahmoudi, SuperIso v3.0, flavor physics observables calculations: Extension to NMSSM, Comput. Phys. Commun. 180 (2009) 1718–1719.

[27] B. Allanach, SOFTSUSY: a program for calculating supersymmetric spectra, Comput. Phys. Commun. 143 (2002) 305–331. arXiv:hep-ph/0104145.

[28] B. Allanach, S. P. Martin, D. G. Robertson, R. Ruiz de Austri, The Inclusion of Two-Loop SUSYQCD Corrections to Gluino and Squark Pole Masses in the Minimal and Next-to-Minimal Supersymmetric Standard Model: SOFTSUSY3.7, Comput. Phys. Commun. 219 (2017) 339–345. arXiv:1601.06657.

[29] C. Degrande, C. Duhr, B. Fuks, D. Grellscheid, O. Mattelaer, T. Reiter, UFO - The Universal FeynRules Output, Comput. Phys. Commun. 183 (2012) 1201–1214. arXiv:1108.2040.

[30] A. Arbey, F. Mahmoudi, SuperIso Relic: A Program for calculating relic density and flavor physics observables in Supersymmetry, Comput. Phys. Commun. 181 (2010) 1277–1292. arXiv:0906.0369.

[31] A. Arbey, F. Mahmoudi, SuperIso Relic v3.0: A program for calculating relic density and flavour physics observables: Extension to NMSSM, Comput. Phys. Commun. 182 (2011) 1582–1583.

[32] A. Arbey, F. Mahmoudi, G. Robbins, SuperIso Relic v4: A program for calculating dark matter and flavour physics observables in Supersymmetry, Comput. Phys. Commun. 239 (2019) 238–264. arXiv:1806.11489.