# `GoSam` Manual

The GoSam Collaboration

Version March 27, 2012

# Contents

# 1 Introduction

## 1.1 Synopsis

GoSam is a general one-loop evaluator for matrix elements. The program produces Fortran 95 code from a given process description by evaluating Feynman diagrams and translating the associated one-loop diagrams into a numerical representation of the numerator such that it can be evaluated and reduced numerically with either the golem95 library [BGH$^+$09, CGH$^+$11] or SAMURAI [MORT10] or PJFry [Yun11, FR11].

## 1.2 Conventions

In this manual, shell commands are indicated by lines starting with a dollar sign ($) and are given for the bash shell only. Lines that are broken for type setting reasons and should continue the previous line(s) start with a $\hookrightarrow$.

Python program fragments are denoted by the '>>>' and '...' (for continuation lines) prompts.

# 2 Setup

## 2.1 Prerequisites

The generation of matrix element code using GoSam can be understood as a three step process, although the three steps are not necessarily obvious to the user. In principle, each step could be run on a different machine and the programs listed below only need to be available during the respective step.

1. During the **setup of the process directory** Python and QGraf need to be installed. This phase is initiated by running `gosam.py` or any user written Python script of similar functionality.
2. During the **code generation** only Form and haggies are run. This phase is initiated by running `make source`.
3. During the **compilation and running** of the matrix element a Fortran 95 compiler and the chosen reduction libraries need to be installed. At the level of the matrix element, this phase is initiated by running `make compile`. Please note that running `make compile` will invoke `make source` if the latter has not been run successfully before that.

Before running the GoSam package, Please ensure that the following programs are available on your system. The numbers indicate during which phase of the code generation the tools will be required.

| | |
|---|---|
| QGraf (1) | QGraf [Nog93] is required in version 3.1 or higher and can be downloaded from `http://cfif.ist.utl.pt/~paulo/qgraf.html`. |
| Python (1) | This program has been tested with Python versions 2.6, 2.7 and 3.1. Please see also `http://python.org`. |
| Form (2) | You will need Form [Ver00] version 3.3 (build 11–aug–2010 or later). The most recent version is available from `http://www.nikhef.nl/~form/`. |
| haggies (2) | The code generator haggies [Rei10] is included in the GoSam distribution already. Alternatively, it can also be obtained separately from the URL `http://sourceforge.net/projects/haggies/`. haggies requires Java in version 1.5 or higher. The current version of GoSam requires haggies in version 1.1 or higher. |
| golem95/SAMURAI/PJFry (3) | For one-loop calculations, at least one of these three libraries is required. If the program is used for the extraction of the $R_2$ term only, the libraries are not required. |

- golem95 can be downloaded from http://projects.hepforge.org/golem/.
- Samurai can be downloaded from http://projects.hepforge.org/samurai/.
- PJFry can be obtained via git from https://github.com/Vayu/PJFry/.

refrep.cls (3) The documentation is based on the LaTeX-class refrep, which appears not to be in the default installation of all LaTeX distributions. It can be downloaded from http://www.ctan.org/ as part of the refman package. This file is only needed if one intends to run make doc, which generates some documentation for the matrix element.

! → Please note that these programs might have license policies which are different from the license applying to GoSam. The authors of GoSam do *not* take any responsibility for any problems related to the above mentioned software packages.

## 2.2 Download

The GoSam source code can be downloaded either via subversion or via HTTP download.

### 2.2.1 Subversion

You can check-out a working copy of the repository with the command

```
$ svn co http://svn.hepforge.org/gosam/trunk/ gosam-1.0
```

This will create a folder gosam-1.0 in your current directory. Authenticated users can use the URL

```
$ svn co svn+ssh://svn.hepforge.org/hepforge/svn/
↪gosam/trunk/ gosam-1.0
```

to gain read and write access to the project files.

### 2.2.2 HTTP Download

Under http://www.hepforge.org/downloads/gosam/ you can download the sources of GoSam using a web browser or a HTTP client like wget or curl. If you received GoSam as a tar-ball you can unpack it using the command

```
$ tar xzvf gosam-1.0.tar.gz
```

## 2.3 Installation

GoSam is distributed as a Python package. The installation of the source package is done by running the setup script. One of the following scenarios will be encountered most probably:

- If the Python installation resides in /usr or /usr/local and the user has super-user privileges:

```
$ sudo python setup.py install
```

- If the user wants (or has to) maintain an alternative installation path for Python modules. < XXXX > here denotes the name of the alternative installation tree:

```
$ python setup.py install --prefix=<XXXX>
```

The prefix option can also be permanently set in the user's pydistutils config file[1] by adding the following lines.

```
[install]
prefix=<XXXX>
```

After successful installation the user should also update the environment variable:

```
PATH=$PATH:<XXXX>/bin
```

For Bourne shell compatible shells (bash, zsh, ...), the PATH environment variable can be permanently changed by adding the following lines to ~/.profile:

```
PATH="$PATH:<XXXX>/bin"
export PATH
```

For csh compatible shells (tcsh, ...), the following line need to be added to ~/.cshrc:

```
setenv PATH "$PATH:<XXXX>/bin"
```

To enable this environment variables change, you need to run `source ~/.profile` or `source ~/.cshrc` and `rehash`, or re-login.

## 2.4 Directory Structure

The GoSam source directory has the structure as described below:

`doc/` This directory contains the documentation and example setup files. You can run `make` in this directory to generate the document `refman.pdf`; this is the document you are currently reading.

`models/` For each implemented model this directory contains the QGraf model file (no extension), a Form interface (`*.hh`) and a Python module (`*.py`). Currently, only the Standard Model (`sm`) is distributed with GoSam. A second version of the Standard Model (`smdiag`) implements diagonal flavour structure ($V_{\mathrm{CKM}} = \mathsf{diag}\{1, 1, 1\}$)

[1] On Unix systems and MacOS this file is called `$HOME/.pydistutils.cfg`, on Windows it is `%HOME%\pydistutils.cfg`.

The structure of the model files is discussed in more detail in Chapter C.1. Model files for the MSSM based on LanHEP [Sem10] and FeynRules/UFO [DDF$^+$11] can be found in the directory `examples/model/`.

`templates/` Contains templates for the creation of the files in the process directory. The contents are transformed by the class `golem.util.parser.Template` and its subclasses in `golem.templates.*`. The translation of the templates is controled by the file `templates.xml` of the same directory.

`src/python/` All model independent `Python` modules can be found in this directory tree.

`src/form/` Here one finds all `Form` files which are not part of the template.

`build` This directory is created during building and installation of this package by running `setup.py`. The files in this directory are of temporary nature and can be safely removed.

`dist` This directory is created by running `setup.py` with the `sdist` or `bdist` command and contains the distributable package files. To create a tar-ball from the working copy, Please run

```
$ python setup.py sdist --formats=gztar
```

For more information please run

```
$ python setup.py --help-commands
```

`examples` This directory contains some simple example processes for which `GoSam` has been compared to the literature.

`olp` Files in this directory are used by `gosam.py --olp`, which is `GoSam`'s implementation of the Les Houches interface for one-loop programs [BBD$^+$10].

# 3 Setup of a Process

## 3.1 Introduction

This chapter provides a step by step guide how to set up a new process.

In order to generate the matrix element for a given process one has to create a process specific setup file, which we call *process card.*

The syntax of this file is closely related to that of Java `.properties` files. The detailed syntax and a full list of options are given in Appendix D. Here we first give a commented example, which should be sufficient to explain the most important features of a *process card.*

## 3.2 Example: $e^+e^- \to t\bar{t}$ at NLO in QCD

It is recommended to generate and modify a template file for the process card instead of starting from scratch. This can be done by invoking the shell command

```
$ gosam.py --template eett.in
```

This would generate the file `eett.in` with some documentation for all accepted options. The options are filled with some default values, which can be set in a global configuration file. The script will search[1] in the `GoSam` directory, in the user's home directory and in the current working directory for a file named '`.golem`' or '`golem.in`'. Such a file can be generated with the following command:

```
$ golem-config.py > golem.in
```

In the following brief tutorial it is assumed that the process $e^+e^- \to t\bar{t}$ should be calculated to order $\mathcal{O}(\alpha)\mathcal{O}(\alpha_s)$ (virtual corrections); the tree-level process is of order $\mathcal{O}(\alpha)$. We neglect the exchange of a $Z$ or a Higgs boson and treat the electron massless. The output directory is assumed to be in the relative path `eett`.

Listing 3.1: `eett.in`

```
process_path=eett
process_name=eett
in=    e+, e-
out=   t, t~
model= sm
```

[1] in this order

```
 6 order= gs, 0, 2
 7
 8 qgraf.options=nosnails,notadpoles,onshell
 9 qgraf.verbatim=\
10    true=iprop[Z, 0, 0];\n\
11    true=iprop[H, 0, 0];
12 zero=me
13 one=gs,e
14
15 extensions=samurai
16 samurai.fcflags=‘pkg-config --cflags samurai‘
17 samurai.ldflags=‘pkg-config --libs samurai‘
```

The above lines are discussed one by one. The line numbers on the left are only included for better readability and *must not* be included in the setup file.

1 The option `process_path` specifies the directory to which all generated files and directories are written. The directory which is specified here must already exist.
Specification of a process path is mandatory.

2 Setting a process name is optional but recommended. All module names will be prefixed with the process name (e.g. `precision` → `eett_precision`). This will avoid name conflicts if at a later stage more than one matrix elements are linked into one executable.

3–4 The options `in` and `out` specify the particles of the initial and final state. The particle names must be defined in the selected model file. As the model files usually define mnemonics for the particle names there might be several ways of specifying the same process. Instead of ‘`e+`’ one could have written ‘`ep`’ or ‘`positron`’. For a complete list of alternative particle names please refer to the documentation of the according model file.
Specifying `in` and `out` particles is mandatory.

→ Appendix C

5 The option `model` specifies which model files should be used in order to generate and evaluate the diagrams.
This option is mandatory.

6 The option `order` is a comma separated list with three entries. The first entry specifies a symbol that denotes a coupling constant. In the Standard Model file `sm` the only two possibilities are ‘`gs`’ for the strong coupling constant $g_s$ and ‘`e`’ for the electro-weak coupling. The second number is the power of the chosen coupling constant for the tree-level diagrams and the third parameter specifies the power of that coupling constant for the one-loop diagrams. Note that the numbers refer to the powers in the diagrams of the amplitude

**!** →

rather than the squared amplitude. In the above example the string '`gs, 0, 2`' specifies that the tree-level diagrams should be of order $g_s^0$ and the one-loop diagrams should be of order $g_s^2$ and an unspecified power of $e$ in both cases. If there is no tree level, i.e. the process is loop induced, the keyword `NONE` should be put as second item in the list, instead of the tree level power of the coupling.

The value of this option is translated into a `vsum` constraint in the file `qgraf.dat`.
This option is mandatory.

8–11 The option `qgraf.options` creates the line '`options=`. . . ;' in the file `qgraf.dat`. The value of the option `qgraf.verbatim` is passed verbatim to the file `qgraf.dat`. In our example we specify that loops of size one and self-energy insertions at external lines should be omitted in the graph generation.
! $\rightarrow$ Lines 9–11 suppress the generation of diagrams containing Higgs and $Z$ bosons. As these commands are passed verbatim to `QGraf` no mnemonic names are allowed here, e.g. the Higgs particle has to be denoted by '`H`' and cannot be replaced by '`h`'. For a complete list of available options, Please consult the `QGraf` manual. For a complete list of particle names see Appendix C.2 resp. the documentation of the model file.
These options can be omitted.

12–13 The keywords `zero` and `one` specify a set of symbols that should be treated as zero (resp. one). These simplifications are applied at the symbolical level. Only symbols that appear in the `Form` interface of the model file should be specified here (masses, couplings, CKM-matrix elements, etc). In the example we specify the electron mass '`me`' to be zero and we do not keep the coupling constants in the calculation explicitly ($g_s = e = 1$).
These options can be omitted.

15 The option `extensions` contains a list of extensions to the core of the program.

16–17 For each extension one can add options of the form *extension.name*. Currently the program is scanning for options of the form *extension*.`ldflags` and *extension*.`fcflags`. These options are copied to the contens of the according variables (`FCFLAGS` and `LDFLAGS`) in the makefiles.

In order to populate the specified process directory with files one invokes

```
$ gosam.py eett.in
```

## 3.3 Process Directory Structure

After running golem with an appropriate setup file the process directory contains a number of files which are described below.

| | |
|---|---|
| codegen/ | This directory contains files which are only relevant for code generation. These files will therefore not be included in a tar-ball created with make dist. |
| common/ | Fortran files which are common to all helicity amplitudes and to the constructed matrix element code. This directory is always compiled first. |
| doc/ | Contains all files (apart from pyxotree.tex and pyxovirt.tex) which are necessary for creating doc/process.ps, which lists all Feynman diagrams of this process, together with colour and helicity info. |
| helicity* | This directory contains all files for a specific helicity amplitude. The labeling of the helicities can be found in doc/process.ps. Before invoking make source, this directory only contains the makefiles. |
| matrix | This folder contains the code to combine the helicity amplitudes into a matrix element. Here one also finds the test program test.f90. This folder is always compiled last. |
| model,model.hh,model.py | The files from the model/ directory of GoSam. The original files are renamed, e.g. sm $\rightarrow$ model, sm.py $\rightarrow$ model.py and sm.hh $\rightarrow$ model.hh. |
| diagrams-[01].hh | The diagram files generated by QGraf. |
| config.sh | This script facilitates linking with external programs. For details, run `$ sh ./config.sh -help` |
| process.hh | contains the process dependent definitions for Form. This file is used by golem.frm to generate the expressions for each diagram in every helicity configuration. |
| process.dat | contains the on-shell conditions, the number of incoming particles and an expression for momentum conservation. This file is needed by the program golem-analyzer.py. |
| func.txt | Defines dependencies between parameters of the model files. |
| Makefile.conf | This files contains the settings which might need to be modified by the user. Please check the contents of this file if you have trouble running the makefiles. |
| Makefile Makefile.source | These two files are part of each directory. Makefile.source is used when calling make source. Running make from the process directory will pass through all subdirectories. The following targets of make are recommended for direct use: |

| | |
|---|---|
| `help` : | lists all major targets. |
| `source` : | generate source files, mainly `Fortran 95` files. |
| `compile` : | compile the `Fortran 95` sources. |
| `dist` : | create a tar-ball of the source files. |
| `clean` : | remove object files and intermediate files. |
| `very-clean` : | remove files including targets of `make source`. |
| `doc` : | create various documents related to the process. To obtain a description of the *topologies*, you need to run `source` before `make doc`. |

## 3.4 Code Generation and Compilation

The `Fortran 95` code is generated by the command

```
$ make source
```

and can be compiled using

```
$ make compile
```

Please note that the `compile` target invokes the `source` target if necessary.

A simple test program, which gives the value of the amplitude at a randomly generated phase space point, can be found in the directory `matrix/`, in order to compile and run it, type

```
$ cd matrix $ make test.exe $ ./test.exe
```

The program will generate a file `_debug.xml`, which, depending on the settings contains the values of helicity amplitudes and diagrams for a set of phase space points.

### 3.4.1 Customization

**Runtime Parameters.** Many settings can be changed without recompiling the code, by creating and modifying the file `matrix/param.dat`. This file has a very simple format:

- Lines starting with a comment character ('!', '#', ';') in the first column and blank lines are ignored.
- All other lines have the format

  ```
  name = float
  # or
  name = float, float
  ```

  where the first line defines a real number and the second line defines a complex number, and *name* is a parameter de.

- Whitespace is ignored but must not appear inside names or literals. Physical lines can not be continued nor can multiple entries appear on one line.

The list of recognized names can be found in the file `common/model.f90`. In addition there are some model independent parameters:

| | |
|---|---|
| `samurai_scalar` | selects a library of scalar integrals (see SAMURAI documentation). |
| `samurai_test` | sets a method to detect unstable points (see SAMURAI documentation). |
| `samurai_verbosity` | sets the verbosity level of SAMURAI; it should be set to zero in a production environment (see SAMURAI documentation). |
| `renormalisation` | An integer number indicating if no renormalisation (0) or $\beta$-function renormalisation (1, QCD only) should be applied. Other values are reserved for future extensions. |
| `gauge`*i*`o` | for the external vector particle with index $i$ (e.g. `gauge1o`, `gauge2o`...), if not defined as a constant. |
| `gauge`*i*`z` | as `gauge`*i*`o`. The polarisation vector is transformed into $$\varepsilon^\mu(k_i) \to \mathtt{gauge}i\mathtt{o} \cdot \varepsilon^\mu(k_i) + \mathtt{gauge}i\mathtt{z} \cdot k_i^\mu$$ This allows for a quick check of gauge invariance. |

Furthermore, all model constants that have not been specified as zero or one can be set in this way. One can can re-set, for example, the value for the Higgs mass using the entry

```
mH = 124.5
```

Please note that upper and lower case letters have to be distinguished and that the names need to be spelled exactly as defined in `model.py`.

**Compile Time Parameters.** Other configuration options can be found in the file `common/config.f90` but require the recompilation of the source code (`make clean; make compile`). Examples of options contained in `config.f90` are

| | |
|---|---|
| `ki` | the floating point kind used throughout the calculation. |
| `debug_lo_diagrams` | controls if information about the tree level diagrams is written to the output file. |
| `debug_nlo_diagrams` | controls if information about the loop-diagrams is written to the output file. |
| `include_eps_terms` | controls if terms of order $\varepsilon$ multiplying poles are taken into account. |
| `include_eps2_terms` | controls if terms of order $\varepsilon^2$ multiplying double poles are taken into account. |
| `include_color_avg_factor` | controls if the color averaging factor for inital state partons is multiplied to the final result. |
| `include_helicity_avg_factor` | controls if the helicity averaging factor for inital state particles is multiplied to the final result. |
| `include_symmetry_factor` | controls if the symmetry factor for identical final state particles is multiplied to the final result. |
| `use_sorted_sum` | controls if the diagrams are summed using the algorithm Malcolm [Mal70], which reduces the error accumulated in presence of large cancellations. |

## 3.5 Drawing the Feynman Diagrams

In order to print out the diagrams the makefile contains the target `doc` which produces the file `process.ps`. We use LaTeX plus the package axodraw [Ver94] to create the graphical representation.

The layout of the diagrams is determined by the algorithm used in feynMF [Ohl95], modelling the propagators by springs. The implemented algorithm works in two steps: first, the topology is disentangled by ordering the external legs such that the diagram can be drawn as a planar graph. The coordinates $e_k$ of the external legs are fixed along a contour around the drawing area.[2] In a second step the remaining degrees of freedom, the coordinates of the vertices $v_i = (x_i, y_i)$, are fixed by minimizing the Lagrangian

$$L(v_1, \ldots, v_n; e_1, \ldots, e_N) = \frac{1}{4}\sum_{i,j=1}^{n} t_{ij}\left(v_i - v_j\right)^2 + \frac{1}{2}\sum_{i=1}^{n}\sum_{k=1}^{N}\lambda_{ik}\left(v_i - e_k\right)^2 \quad (3.1)$$

Here, $n$ is the number of vertices and $N$ is the number of external legs. Minimization of the Lagrangian leads to a system of linear

[2] Currently, this contour is chosen as an ellipse but in principle any convex shape could be used.

equations, which can easily be solved.

$$\begin{aligned}
&\frac{\partial L}{\partial v_r} = 0 \\
\Leftrightarrow &\frac{1}{2}\sum_{i,j=1}^{n} t_{ij}\left(v_i - v_j\right)\cdot\left(\delta_{ir}-\delta_{jr}\right) + \sum_{i=1}^{n}\sum_{k=1}^{N}\lambda_{ik}\left(v_i - e_k\right)\cdot\delta_{ir} = 0 \\
\Leftrightarrow &M_{rj}v_j \equiv \sum_{j=1}^{n} t_{rj}\left(v_r - v_j\right) + \left(\sum_{k=1}^{N}\lambda_{rk}\right)v_r = \sum_{k=1}^{N}\lambda_{rk}e_k
\end{aligned}$$

In the last step we used the symmetry of $t_{ij}$. The matrix $M$ can be written as

$$M_{rc} = \begin{cases} \left(\sum_{i\neq r} t_{ri}\right) + \left(\sum_{k=1}^{N}\lambda_{rk}\right), & r = c \\ -t_{rc}, & \text{otherwise} \end{cases} \tag{3.2}$$

The symbol $t_{ij}$ is the sum of the spring constants of all propagators connecting vertices $i$ and $j$; similarly, $\lambda_{ik}$ is the spring constant of the leg $k$ if it is connected to vertex $i$ and zero otherwise.

## 3.6 Import of Model Files

Examples about how to import model files can be found in the subdirectory `examples`.

### 3.6.1 Import from FeynRules

A model description in the UFO [DDF+11] format consists of a `Python` package stored in a directory. In order to import the model into `GoSam` one needs to set the `model` variable specifying the keyword `FeynRules` in front of the directory name, where we assume that the model description is in the directory `$HOME/models/MSSM_UFO`.

```
model= FeynRules,$HOME/models/MSSM_UFO
```

### 3.6.2 Import from LanHEP

In order to use model files generated by LanHEP the following steps have to be taken:

1. When generating the tables using LanHEP, one should include the following option to ensure that the generated tables have the correct headings[3]. The number of spaces in the column headers are irrelevant as long as the columns are wide enough to contain the respective values.

```
prtcformat
fullname:  ' fullname ',
name:  ' name ',
```

[3] `GoSam` relies on the column names rather than some specific order.

```
aname:  '  aname ',
spin2:  '  spin2 ',
mass:  '  mass ',
width:  '  width ',
color:  '  color ',
aux:  '  aux ',
texname:  '  texname ',
atexname:  '  atexname ',
pdg:  '  pdg '.
```

2. If the model file is not already equipped with pdg codes the user might want to use the `prtcprop` command in LanHEP to add the relevant codes.

3. In the setup file, one needs to specify the model as a pair of path and integer number. If the table files are under the directory `lanhep/ued/` in the tables `func7.mdl`, `lgrng7.mdl`, `prtcls7.mdl` and `vars7.mdl`, the correct statement in the setup file would be

   ```
   model=lanhep/ued, 7
   ```

4. The use of user defined functions (`external_func` in LanHEP) requires an adaption of the file `codegen/haggies-10.in`. If one wants to use the function `double foo(double,double)` the following line sould be added.

   ```
   @define mdlfoo :  real, real -> real =
   "foo(%2$s, %3$s)";
   ```

   The function also needs to be declared in `codegen/functions.out` in the subroutine `init_functions`

## 3.7 Handling Big Processes

Although the default settings should work for most cases, very big processes in terms of the number of diagrams and the size of the expressions can cause the compiler to become very slow or even to crash. In this section we discuss solutions which can help to reduce the load for the compiler and to speed-up the code generation. It should be mentioned that some of these measures can have a negative impact on the runtime efficiency of the generated code.

### 3.7.1 Grouping of Tree Level Diagrams

By default the expressions of all tree-level diagrams are grouped into one file. This has the advantage that subexpressions which appear in several tree-level diagrams can be reused across the amplitude. In some cases it can happen that the sum of all terms of the tree-level diagrams is too big to be compiled in one subroutine. In this case it is recommended to set the option `group` to `false`.

### 3.7.2 Computation of Abbreviations

The constant, i.e. $q$- and $\mu^2$ independent parts of the numerators of the one-loop diagrams are factored out from the numerators and computed as abbreviations. In some cases the list of abbreviations is too big to be compiled into one subroutine. One can restrict the number of instructions that go into a single subroutine by setting `abbrev.limit` to a positive number in the setup file. The variable `abbrev.level`, which by default is set to `helicities`, can be set to `groups` or `diagrams` if the list of abbreviations common to a helicity configuration is too large.

If the list of abbreviations causes `haggies` to crash, one needs to increase the amount of memory reserved for Java. This can be done by adding the `-Xmx` option to the call of Java. A typical setting of the variable `haggies.bin` would be

```
haggies.bin=java -Xmx3g -jar
↪ ${GOLEMPATH}/haggies/haggies.jar
```

which assigns 3 GB of memory to Java.

### 3.7.3 Splitting the Process

If a process becomes too big in order to be linked[4] there are some possibilities to split the process into independent programs:

- the generation of a subset of the helicity configurations, e.g. one helicity configuration per process directory.
- the generation of a subset of diagrams. If the diagrams are not split according to gauge invariant subsets the user should ensure that all subsets are called with the same set of phase space points. An easy way of splitting the diagrams into subsets is by using the option `select.nlo=`⟨*first*⟩`:`⟨*last*⟩, where `first` and `last` refer to the diagram numbers in *process.ps*.

## 3.8 Advanced Usage

The call to the executable `gosam.py` can be simulated inside more complex `Python` programs. It is an easy exercise to run the file generation in user defined `Python` scripts as long as one includes the module files in the environment variable `PYTHON_PATH`. The following script emulates the program `gosam.py`:

```
>>> from golem.util.config import Properties
>>> from golem.util.main_misc import *
>>> props = Properties()
```

[4] Currently, most systems support programs to a size up to 4 GB. Although 64 bit systems can handle a much bigger address space, the current limitation comes from some legacy code in the GNU linker.

```
>>> props.setProperty("in", ["e+", "e-"])
>>> props.setProperty("out", ["t", "t~"])
>>> # ... populate props with further values ...
>>> workflow(props)
>>> generate_process_files(props)
```

## 3.9 Advanced Diagram Selection

GoSam implements several ways of selecting subsets of diagrams:

- by restricting QGraf,
- by selecting specific diagrams by their number,
- by defining filters using Python.

### 3.9.1 Restricting the Generation with QGraf

The options for restricting the set of diagrams at the level of the diagram generation is the most efficient way since this happens already at the earliest possible stage. However, QGraf's built-in filters are sometimes too limited in order to express more advanced criteria.

GoSam allows one to pass information to QGraf through the option `qgraf.options` and through `qgraf.verbatim`, `qgraf.verbatim.lo` and `qgraf.verbatim.nlo`. For the exact syntax the user is refered to the QGraf documentation.

### 3.9.2 Selecting Diagrams by their Number

An a posteriori selection 'by eye' can be achieved after all (also unwanted) diagrams of a process have been generated and inspected in `doc/process.ps`. The user can then modify the options `select.lo` and `select.nlo` and rerun `gosam.py`.

### 3.9.3 Filtering Diagrams in Python

The user can write short Python functions in order to decide whether a specific diagram is to be taken or not. This function should return `True` for all diagrams which are kept, and `False` for all diagrams which should be discarded. These functions are passed by the options `filter.lo` and `filter.nlo`.

Longer functions should be defined in an external file, which can be passed using `filter.module`.

When writing a filter the one can use the predefined particle lists `QUARKS`, `LEPTONS`, `FERMIONS` and `BOSONS`. The underscore (_) matches any field.

A diagram object `d` has the following methods which are inteded to be used in filters. Alternative predefined functions and functors are also given.

`d.rank()` : returns the tensor rank of a diagram.
`RANK` $\equiv \lambda$`d`.(`d.rank()`)

`d.loopsize()` : returns the number of propagators in the loop of a diagram.
`LOOPSIZE` $\equiv \lambda$`d`.(`d.loopsize()`)

`d.sign()` : computes the sign coming from closed fermion loops.
`SIGN` $\equiv \lambda$`d`.(`d.sign()`)

`d.isNf()` : reports if a diagram contains a closed quark loop of size two where all loop propagators are massless.
`NF` $\equiv \lambda$`d`.(`d.isNf()`)

`d.isMassiveQuarkSE()` : returns True if the diagram contains a QCD self energy insertion at a massive quark line.
`MQSE` $\equiv \lambda$`d`.(`d.isMassiveQuarkSE()`)

`d.isScaleless()` : returns True if the loop integral associate with this diagram carries no scale.
`SCALELESS` $\equiv \lambda$`d`.(`d.isScaleless()`)

`d.vertices(f1,f2,...)` : returns the number of vertices in the diagram with the specified fields. The arguments `f1`, `f2`, ... are lists of field names.
`VERTICES(f1,f2,...)` $\equiv \lambda$`d`.(`d.vertices(f1,f2,...)`)

`d.loopvertices(f1,f2,...)` : same as `vertices`, but only counts vertices which have loop propagators attached.
`LOOPVERTICES(f1,f2,...)` $\equiv \lambda$`d`.(`d.loopvertices(f1,f2,...)`)

`d.iprop(f,**opts)` : returns the number of propagators of the given fields. Optional arguments are `momentum` to specify the momentum of the propagator, `twospin` to filter by the $2\times$ the spin, `massive` to specify whether massive or massless propagators should be considered and `color` to filter for certain color representations.
`IPROP(...)` $\equiv \lambda$`d`.(`d.iprop(...)`)

`d.chord(f,**opts)` : same as `iprop` but only counts loop propagators.
`CHORD(...)` $\equiv \lambda$`d`.(`d.chord(...)`)

`d.bridge(f,**opts)` : same as `iprop` but only counts propagators which are not in a loop.
`BRIDGE(...)` $\equiv \lambda$`d`.(`d.bridge(...)`)

`d.QuarkBubbleMasses()` : returns a list of all different masses in a closed quark loop of size two or an empty list if the diagram is not a quark bubble.
`QBMASSES` $\equiv \lambda$`d`.(`d.QuarkBubbleMasses()`)

Furthermore, the following predefined filters exist:

`NFGEN(f1,f2,...)` : for closed quark loops of size two this filter returns true only if all loop propagators belong to one of the fields in the argument list. For all diagrams which are not quark bubbles it returns True.

| | |
|---|---|
| `AND(filter1,filter2,...)` : | returns True if all filters in the argument list return True. |
| `OR(filter1,filter2,...)` : | returns True if at least one filter in the argument list returns True. |
| `NOT(filter)` : | returns True if the argument evaluates to False. |
| `TRUE` : | always returns True. |
| `FALSE` : | always returns False. |

# 4 The Binoth Les Houches Accord Interface

## 4.1 Initialisation Phase

The script `gosam.py --olp` which comes with `GoSam` can be used to generate matrix elements compatible with the specifications of the Binoth Les Houches Accord [BBD+10]. This script expects at least the name of an order file. This order file is usually but not necessarily created by a Monte Carlo program. An example file for the partonic $2 \to 3$ processes of $pp \to t\bar{t} +$ jets is given below:

```
 1  MatrixElementSquareType  CHsummed
 2  IRregularisation         tHV
 3  OperationMode            CouplingsStrippedOff
 4  SubdivideSubprocess      yes
 5  AlphasPower              3
 6  CorrectionType           QCD
 7
 8  # Here comes the list of subprocesses
 9  # specified through PDG codes
10  # g      g -> t  t-bar  g
11    21     21 -> 6  -6    21
12  # u  u-bar -> t  t-bar  g
13     2     -2 -> 6  -6    21
14  # u      g -> t  t-bar  u
15     2     21 -> 6  -6     2
```

The line numbers are not part of the file. The arrow '→' is generated by the two characters '`->`'. The following options are part of the Standard and accepted by `GoSam`:

`MatrixElementSquareType` : accepts the values **Hsummed**, **Csummed**, **Haveraged**, **Caveraged**, **CHsummed**, **CHaveraged**.

The value **NOTsummed** is not supported. Sensible combinations are also allowed, as in

**MatrixElementSquareType Hsummed Caveraged**

In `GoSam` this statement is optional. Any quantity which is not explicitely averaged is assumed to be summed

`CorrectionType` : accepts the values **QCD**, **QED** and **EW**, whereas `GoSam` does not distinguish between the latter two (this behaviour might change in the future when appropriate model files are available).

This statement is mandatory and must not be omitted.

`IRregularisation` : accepts the values **tHV** ('t Hooft-Veltman scheme) and **DRED** (dimensional reduction). The value CDR (conventional dimensional regularisation) is not supported and therefore rejected.

This statement is mandatory and must not be omitted.

`MassiveParticleScheme` : accepts the value **OnShell** only. At the moment this option has no effect on the generation of the matrix element. This statement is optional; if it appears in the order file a warning is issued, reminding the user that no UV-counterterms for massive particles are implemented yet.

`IRsubtractionMethod` : accepts the value **None** only. `GoSam` does not provide any subtracted output.

This statement is optional.

`ModelFile` : accepts the name of parameter file in the Les Houches Accord format. The script reads the parameter file setting all masses to zero which are not specified explicitly to be non-zero.

This statement is mandatory.

It is recommended to use absolute paths here as the file will later be read in the function `OLP_Start` in the matrix element code, which might be located elsewhere.

`OperationMode` : accepts the value **CouplingsStrippedOff** only.

This statement is optional. If it is given, the coupling constants are stripped off from the amplitude.

`SubDivideSubProcess` : accepts logical values (**yes** or **no**).

If the value is **yes** a separate channel for each helicity is assigned. Otherwise there will be one channel per subprocess.

This statement is optional. Its default value is **no**.

`AlphasPower` : the power of $\alpha_s$ of the Born cross-section. At least one of the options **AlphaPower** and **AlphasPower** has to be specified.

`AlphaPower` : the power of $\alpha$ of the Born cross-section. At least one of the options **AlphaPower** and **AlphasPower** has to be specified.

The options which have been proposed for electro-weak corrections are currently not supported.

### 4.1.1 Command Line Arguments of `gosam.py --olp`

The syntax for the invocation of `gosam.py` is as follows:

```
$ gosam.py --olp {⟨option⟩}
↪⟨order file⟩ {⟨order file⟩}
```

```
↪{⟨key⟩=⟨value⟩}
```

The allowed options are given below. The list of ⟨*key*⟩=⟨*value*⟩-pairs supplements the options given in the configuration files.

| | |
|---|---|
| `-h`, `--help` : | Prints a help screen with all available command line options and exits. |
| `-d`, `--debug` : | With this options the script will print lots of extra information to the screen, which is usually not useful for non-experts. |
| `-v`, `--verbose` : | The script will print information e.g. about creating directories and reading files. |
| `-w`, `--warn` : | Warnings and errors are printed. This is the default setting. |
| `-q`, `--quiet` : | Only errors are printed, no warnings are issued. |
| `-l`*file*, `--log-file=`*file* : | All messages are written to a log file. When one or more log files are specified the information is still written to the screen with the latest specified level of detail. The following example will read the order file `test.olo`; messages at the debug level will be written to `detailed.log`, warnings and errors are written to `short.log` and only errors are printed to the screen. |

```
$ gosam.py --olp -d -ldetailed.log -w
↪ -lshort.log -q test.olo
```

| | |
|---|---|
| `-c`*file*, `--config=`*file* : | Overlay default config files by the specified file. Usually, the script first searches in the default locations for configuration files. Afterwards, all files specified by `-c` options are read in the order in which they are encountered. Values which are already set by earlier files will be overwritten. See also option '`-C`'. |
| `-C`, `--no-defaults` : | The script will not search for configuration files (`.golem` and `golem.in`) in the standard locations (`GoSam` installation directory, user's home directory and current working directory). |
| `-f`, `--force` : | Overwrite contract files without asking. The default behaviour is that contract files are not overwritten. If a contract file already exists the program gives an error message. |
| `-e`, `--use-single-quotes` : | Activates syntax extensions that allow the use of single quotes in order and contract files (See Section 4.1.2). |
| `-E`, `--use-double-quotes` : | Activates syntax extensions that allow the use of double quotes in order and contract files (See Section 4.1.2). |
| `-b`, `--use-backslash` : | Activates syntax extensions that allow the use of backslash escape sequences in order and contract files (See Section 4.1.2). |
| `-i`, `--ignore-case` : | Activates syntax estensions which make the parsing of order and contract files case-insensitive (See Section 4.1.2). |

`-x`, `--ignore-unknown` : Unknown statements or values in order and contract files will be ignored. The default behaviour is that unknown statements and/or values will lead to an error message.

`-o`*file*, `--output-file=`*file* : Specifies the name of the contract file(s). The following set of wildcard sequences can be used to derive the name of the contract file from the name of the order file. A value of '`-`' writes to the standard output.

- `%f` : The full file name (e.g. '`dir/process.olo`')
- `%F` : The file name without any leading path ('`process.olo`')
- `%p` : Path name only ('`dir/`')
- `%s` : The stem of the file name ('`process`')
- `%e` : The extension of the file name ('`.olo`')

By default this option is set to '`%p%s.olc`'.

`-D`*dir*, `--destination=`*dir* : Chooses the output directory, to which each process is written. The same wildcards as above can be used. By default, all output is written to the current working directory. It is therefore not recommended to set this option using wildcards when more than one order file is specified.

`-t`*path*, `--templates=`*path* : Sets an alternative templates directory or template XML-file.

`-z`, `--scratch` : Overwrites all process files, including those which otherwise would be preserved (`Makefile.conf`, `config.f90` etc).

### 4.1.2 `GoSam` Extensions to the Original Standard

Modern file systems allow for path names which cannot be expressed in the original formulation of the Les Houches accord. Therefore `GoSam` implements syntax extensions for order and contract files for including special characters in statements, especially in file names (as in **ModelFile**).

double qoutes : This syntax extension proposes that inside a pair of double quotes (ASCII character #34) special characters lose their special meaning. The backslash acts as escape character, with the following set of escape sequences being allowed:

- `\t` expands to a horizontal tabulator character (ASCII character #9),
- `\n` expands to a new line character (ASCII character #10),
- `\f` expands to a form feed character (ASCII character #12),
- `\r` expands to a carriage return character (ASCII character #13),

`\x`*hh* , where *hh* are two hexadecimal digits expands to the character of which the ASCII code is the hexadecimal number represented by the digits *hh*.

- any other character following a backslash expands to itself, in particular `\"` and `\\`.

single quotes : This syntax extension proposes that inside a pair of single quotes (ASCII character #39) all characters lose their special meaning. There is no escape character. A literal single quote is generated by a sequence of two single quotes (Pascal like).

backslash escapes : This syntax extension proposes that any character following a backslash loses its special meaning.

Different extensions might prove useful on different operating systems. On a Windows system, the file name `F:\Golem Files\mssm.slha` can only be expressed with the proposed syntax extensions and would have the following three equivalent representations:

- `F:\\Golem\ Files\\mssm.slha`
- `'F:\Golem Files\mssm.slha'`
- `"F:\\Golem Files\\mssm.slha"`

The three extensions can be switched on by the command line options of `gosam.py --olp`, '`-E`', '`-e`' and '`-b`' respectively.

### 4.1.3 Advanced Usage

The core functionality of the script `gosam.py --olp` is implemented by the function `golem.util.olp.process_order_file`, which has the the following signature:

```
process_order_file(order_file_name, out_file, out_dir,
  conf, templates=None, ignore_case=False,
  ignore_unknown=False, single_quotes=False,
  double_quotes=False, backslash_escape=False)
```

| | |
|---|---|
| `order_file_name` : | (character string) name of the order file. |
| `out_file` : | (file object, open for writing) contract file. |
| `out_dir` : | (character string) name of an existing directory to which all matrix-element files will be written. |
| `conf` : | (`golem.util.config.Properties`) configuration shared by all subprocesses. |
| `templates` : | (character string) template directory or name of an XML-file. |
| . . . : | all other arguments activate the corresponding syntax extensions. |

The return value is zero in case of a success and one if an error occurred.

A list of options read from default config files can be obtained by the function `golem.util.main_misc.find_config_files()`. The following example suggests the usage of the interface from a `Python`-based Monte Carlo program

```
import os
import golem

# Monte Carlo program prepars the process
# and writes order file proc.olo ...
# (not shown in example)

conf = golem.util.main_misc.find_config_files()
f = open("proc.olc", 'w')
os.mkdir("proc/")

# Add own options
conf[golem.properties.model] = \
   "FeynRules, ${HOME}/models/mssm_ufo"
conf[golem.properties.fc_bin] = "gfortran"
err_flag = golem.util.olp.process_order_file(\
   "proc.olo", f, "proc/", conf)

if err_flag > 0:
        print "Problems generating OLP"
        print "Please consult the file proc.olc"
```

## 4.2 Runtime Phase

After the script `gosam.py --olp` or any equivalent program has been run successfully, the files in the newly created process directories are compiled by invoking `make` in the respective top-level directory. This generates the object file `olp_module.o` which contains all API functions. The library for a given process can be linked using the script `config.sh` in the same directory. The makefile of a client program would typically contain code similar to the following:

```
PROCESS_PATH=path/to/your/process-files
LDFLAGS+=$(shell sh $(PROCESS_PATH)/config.sh -libs)
```

**!** $\rightarrow$ The module `olp_module.f90` uses `Fortran` 2003 extensions (`ISO_C_BINDING`) for establishing a well defined interface for the linker. Older `Fortran` 95 compilers might therefore not be able to compile this module. Please refer to the compiler documentation for details.

### 4.2.1 API Functions

The file `olp.h` contains the following prototypes.

```
void OLP_Start(char*,int*);
void OLP_EvalSubProcess(int,double*,
   double,double*,double*);
void OLP_Finalize();
void OLP_Option(char*,int*);
```

The first two functions are defined exactly as proposed in [BBD+10]. The other two functions extend the original standard. It should, however, be noted that the generated matrix element code can be run without any calls to either OLP_Finalize or OLP_Option.

`OLP_Start`

```
void OLP_Start(char* contract_file, int* success);
```

This function should be called before the first evaluation of the matrix element. It ensures that all global variables in the matrix element code are initialized properly. The argument contract_file should receive the (full) name of the contract file which was generated together with the matrix element. The integer success is initialized by OLP_Start to either the value one, indicating success, or zero, indicating that an error occurred during initialization.

Matrix elements generated with `GoSam` will try to read the SLHA model file specified by the option **ModelFile** in the contract file. It is not required that the contract file used in the runtime phase points to the same model file as used during the initialisation phase. However, values which were set to zero during initialisation will remain zero during the runtime phase.

`OLP_EvalSubProcess`

```
void OLP_EvalSubProcess(int label, double* momenta,
   double scale, double* parameter, double* amp);
```

This function retrieves the values for a channel of the OLP for a given phase space point. A channel might be a subprocess or a gauge invariant partial amplitude, depending on the settings in the contract file. The channel is labeled by the argument label. The second argument is a one-dimensional array holding the $5 \times N$ components of the momenta for an $N$-particle process. They are in the order

$$(E^{(1)}, p_x^{(1)}, p_y^{(1)}, p_z^{(1)}, m^{(1)}, E^{(2)}, p_x^{(2)}, p_y^{(2)}, p_z^{(2)}, m^{(2)}, \ldots, m^{(N)})$$

The third argument is the renormalization scale (not its square). A list of scale dependent parameters is passed in the fourth argument. Its first entry is expected to be $\alpha_s(\mu)$. Any further entries are user-defined; the user is expected to adapt the subroutine `init_event_parameters` in `olp_module.f90` if he wishes to make use of any additional parameters.

The last argument is an array of length four. Its entries are, in this order,

1. the coefficient of the $1/\varepsilon^2$ pole in the Laurent series of the interference term between virtual and Born amplitude,
2. the coefficient of the $1/\varepsilon$ pole in the Laurent series of the interference term between virtual and Born amplitude,
3. the $\mathcal{O}(1)$ term in the Laurent series of the interference term between virtual and Born amplitude,
4. the square of the Born amplitude.

Matrix elements generated by `GoSam` use the convention that in case of an error during the evaluation of the matrix element, the fourth entry is set to $(-1)$. It is therefore recommended that client programs check for the positiveness of the Born matrix element.

`OLP_Finalize`

```
void OLP_Finalize();
```

This function should be called after the last evaluation of the matrix element. It allows the OLP to close any open file handles, to release allocated memory and to exit gracefully. Although on most modern operating systems this is done automatically, it is good practice and therefore recommended to always call this function before exiting the program.

`OLP_Option`

```
void OLP_Option(char* assignment, int* success);
```

This function can be used to update internal parameters of the OLP which are not part of the standard. The first argument is a character string containing a textual representation of the requested assignment. The second argument will be set by the function according to the success of the request.

Matrix elements generated with `GoSam` accept any string which would also be valid as a (non-comment) line in a parameter file (see `model.f90`). Typical calls would be

```
OLP_Option("samurai_test=3", &flag);
/* The previous call requires
 * reinitialization of the OLP */
OLP_Start(contract_file, &flag);
OLP_Option("Nf=5", &flag);
/* Setting the Higgs mass: */
OLP_Option("mH=124.5", &flag);
```

### 4.2.2 The OLP Socket Protocol

The necessity to link the client program each time another OLP is used might become cumbersome, especially when one likes to work with more than one OLP at the same time. We have therefore developed a socket protocol which enables any client program to access the same functionality as defined in the Les Houches accord [BBD+10] through a TCP/IP connection with a server hosting the OLP. In this way it is possible to access multiple OLPs simultaneously and to load OLPs at runtime.

The OLP Socket Server

OLPs generated with `GoSam` contain additional files in their top-level directory implementing a server for the OLP Socket protocol. These files are

- `olp_daemon.c` : ANSI-C file with service routines especially network related routines,
- `olp_daemon.h` : ANSI-C file, header for `olp_daemon.c`,
- `olp_protocol.l` : Lex/Flex file, part of the grammar definition of the protocol and
- `olp_protocol.y` : Yacc/Bison file, part of the grammar definition of the protocol, contains the main program.

These files are compiled with the command

```
$ make olp_daemon EXTRA_LDFLAGS=...
```

It is often necessary to specify the variable `EXTRA_LDFLAGS` to provide the necessary run-time libraries of the `Fortran 95` compiler.

The compiled program can be run with the following options

```
$ olp_daemon [-p port] [-s|-S] [-f] file-name
```

- `-f`*file-name* : name of a contract file (required).
- `-p`*port* : port at which the program accepts connections, default: 7711.
- `-s/-S` : forbid resp. allow the `SHUTDOWN` command, default: allow.
- `-r/-R` : forbid resp. allow the `RESTART` command, default: allow.
- `-d` : detach from terminal (run as daemon).

OLP Socket Clients

Sample client implementations for `C++`, `Java` and `Python` are provided in the directory `olp/contrib/`. Below, a brief example for the `C++` case is given:

```
olp::OLPClient OLP_EvalSubProcess("localhost", 7711);
OLP_EvalSubProcess(0, num_legs, mom, scale,
                         num_param, param, amp);
OLP_EvalSubprocess.close();
```

The class `OLPClient` overwrites the operater `()` emulating the original protocol as closely as possible. For technical reasons, two additional arguments (`num_legs` and `num_param`) are required, specifying the number of external legs and the length of the array `param` respectively.

Definition of the Protocol

The protocol consists of statements sent by the client to the server. Each statement is terminated by a newline character. The server responds with one line starting with a three digit number followed by a space and an optional message. The three digit number contains the response code. A response code of 200 signals success, all other values denote an error.

# Appendix A Conventions of the Amplitude

## A.1 Convention of `golem95`

The integral library `golem95` computes integrals of the form

$$\int \frac{\mu^{2\varepsilon}\mathrm{d^n k}}{i\pi^{n/2}} \frac{k^{\mu_1}\cdots k^{\mu_r}}{((k+r_1)^2-m_1^2)\cdots(k+r_N)^2-m_N^2)} = \\ r_\Gamma \cdot \left[\frac{c_{-2}}{\varepsilon^2}+\frac{c_{-1}}{\varepsilon}+c_0+\mathcal{O}(\varepsilon)\right] \quad \text{(A.1)}$$

where $n=(4-2\varepsilon)$ and

$$r_\Gamma = \frac{\Gamma(1+\varepsilon)\Gamma^2(1-\varepsilon)}{\Gamma(1-2\varepsilon)}. \quad \text{(A.2)}$$

The integration measure for the internal momentum $k$ is

$$\frac{\mu^{2\varepsilon}\mathrm{d}^n k}{(2\pi)^n} = \mu^{2\varepsilon}\frac{i}{2^n\pi^{n/2}} \cdot \frac{\mathrm{d}^n k}{i\pi^{n/2}} = \frac{(4\pi)^\varepsilon \cdot i}{(4\pi)^2} \cdot \frac{\mu^{2\varepsilon}\mathrm{d}^n k}{i\pi^{n/2}}. \quad \text{(A.3)}$$

## A.2 Convention of `GoSam`

The factor from above which does not go into the integral definition of `golem95` can be written as

$$\frac{(4\pi)^\varepsilon \cdot i}{(4\pi)^2} = \frac{(4\pi)^\varepsilon}{(2\pi)(4\pi)}\frac{i}{2} \quad \text{(A.4)}$$

The factor of $i/2$ is included in the amplitude definition of `GoSam`. The factors $(2\pi)$ and $(4\pi)$ are later used to build up a factor of $\alpha_x/2\pi$, where $\alpha_x$ is either $\alpha$ or $\alpha_s$.

In the following we assume that the coupling constants[1] have been set to one in the setup of `GoSam`. This ensures that the one-loop matrix element in QCD is calculated in the $\overline{\mathrm{MS}}$ scheme as

$$|\mathcal{M}|^2_{\text{1-loop}} = \frac{\alpha_s}{2\pi}\frac{(4\pi)^\varepsilon}{\Gamma(1-\varepsilon)} \cdot \left[\frac{c_{-2}}{\varepsilon^2}+\frac{c_{-1}}{\varepsilon}+c_0+\mathcal{O}(\varepsilon)\right](g_1^{n_1}\cdots g_q^{n_q}) \quad \text{(A.5)}$$

The factor $(g_1^{n_1}\cdots g_q^{n_q})$ are the coupling constants appearing in the squared tree-level matrix element. `GoSam` will return the coefficients $c_{-2}$, $c_{-1}$ and $c_0$.

The conversion between different conventions for the $\Gamma$-functions is straightforward:

$$\frac{1}{\Gamma(1-\varepsilon)} = r_\Gamma + \mathcal{O}(\varepsilon^3) = \left(1-\frac{\pi^2}{6}\varepsilon^2\right)\Gamma(1+\varepsilon)+\mathcal{O}(\varepsilon^3) \quad \text{(A.6)}$$

[1] $e$ and $g_s$ in the standard model

The relevant terms in the expansion of $r_\Gamma$ are

$$r_\Gamma = e^{-\gamma_E \varepsilon} \left( 1 - \frac{\pi^2}{12} \varepsilon^2 \right) + \mathcal{O}(\varepsilon^3) \tag{A.7}$$

If one prefers to pull out a factor of $e^{-\gamma_E \varepsilon}(4\pi)^\varepsilon$ the appropriate definition of the matrix element up to terms of $\mathcal{O}(\epsilon)$ is

$$\frac{|\mathcal{M}|^2_{\text{1-loop}}}{e^{-\gamma_E \varepsilon}(4\pi)^\epsilon} = \frac{\alpha_s}{2\pi} \cdot \left[ \frac{c_{-2}}{\varepsilon^2} + \frac{c_{-1}}{\varepsilon} + \left( c_0 - \frac{\pi^2}{12}\, c_{-2} \right) \right] (g_1^{n_1} \cdots g_q^{n_q}) \tag{A.8}$$

# Appendix B Explicit Reduction of the $R_2$ Terms

The $R_2$ term [OPP08] consists of all terms of the numerator containing an explicit $\varepsilon$ or $\mu^2$ coming from the Lorentz algebra. For an explicit reduction of these terms, a list of all integrals of the form

$$\int \frac{\mu^{2\varepsilon}\mathrm{d}^n k}{i\pi^{n/2}} \frac{N(\hat{q})\cdot \mu^{2\alpha}\cdot \varepsilon^{\beta}}{D_0 \cdots D_N} \tag{B.1}$$

where either $\alpha$ or $\beta$ is a positive integer number and the denominators are $D_i = (q+r_i)^2 - m_i^2 + i\delta$. Note that integrals where both $\alpha$ and $\beta$ are non-zero, will not contribute to the final result. We expand the above tensor integral and only consider the term of rank $r$, similarly to Eq. (208) in Ref. [Rei09]:

$$\begin{aligned} I_N^{n,\alpha,\beta;\mu_1\ldots\mu_r} &= (-1)^r \frac{\Gamma(\alpha-\varepsilon)}{\Gamma(-\varepsilon)}\varepsilon^{\beta} \sum_{l=0}^{\lfloor r/2\rfloor}\left(-\frac{1}{2}\right)^l \sum_{j_1,\ldots,j_{r-2l}=1}^{N} \times \\ & \left[\hat{g}^{\bullet\bullet}\ldots\hat{g}^{\bullet\bullet} r_{j_1}^{\bullet}\cdots r_{j_{r-2l}}^{\bullet}\right]^{\mu_1\ldots\mu_r} I_N^{n+2\alpha+2l}(j_1,\ldots,j_{r-2l}). \end{aligned} \tag{B.2}$$

Here, the integral $I_N^d(j_1, j_2, \ldots)$ denotes a Feynman parameter integral with the parameters $z_{j_1}, z_{j_2}, \ldots$ in the numerator,

$$\begin{aligned} I_N^d(j_1,\ldots,j_p) &= \\ & (-1)^N \Gamma\left(N-\frac{d}{2}\right)\int \mathrm{d}_{\square}^N z\, \delta_z \frac{\prod_{\nu=1}^{p} z_{j_\nu}}{\left[-\frac{1}{2}z^{\mathsf{T}}Sz - i\delta\right]^{N-d/2}}, \end{aligned} \tag{B.3}$$

where $\mathrm{d}_{\square}^N z = \prod_{j=1}^N \mathrm{d}z_j \Theta(z_j)\Theta(1-z_j)$ and $\delta_z = \delta(1-\sum_i z_i)$. The square brackets $[\ldots]^{\mu_1\cdots\mu_p}$ expand to the sum of all possible assignments of indices to the $\hat{g}^{\bullet\bullet}$-tensors where a (one) arbitrary assignment of indices to the momenta $r_j^{\bullet}$ is chosen.

We only need to consider integrals containing an UV pole, which leads to a rational term when multiplied with $\varepsilon$ stemming either from $\varepsilon^\beta$ or from

$$\frac{\Gamma(\alpha-\varepsilon)}{\Gamma(-\varepsilon)} = (\alpha-1)!\left[-\varepsilon + \mathcal{O}(\varepsilon^2)\right], \quad \text{for}\, \alpha > 0. \tag{B.4}$$

The UV divergence comes from the Gamma function

$$\Gamma\left(N - \frac{n+2\alpha+2l}{2}\right) = \Gamma(\varepsilon - (2+\alpha+l-N)) \equiv \Gamma(\varepsilon-\eta) \tag{B.5}$$

in the Feynman parameter integral $I_N^{n+2\alpha+2l}$. Hence, we examine further the expression

$$\varepsilon \cdot I_N^{n+2l+2\alpha}(l_1, \ldots, l_{r-2l}) = \begin{cases} \mathcal{O}(\varepsilon), & \eta < 0 \\ (-1)^N \frac{1}{2^\eta \eta!} \int \mathrm{d}_\square^N z \delta_z \left[z^\mathsf{T} S z\right]^\eta \prod_{i=1}^{r-2l} z_{l_i}, & \eta \geq 0 \end{cases} \tag{B.6}$$

The remaining integration can be understood as a special case of the Feynman parameter identity

$$\frac{1}{\prod_{j=1}^N A_j^{\alpha_j}} = \frac{\Gamma(\alpha)}{\prod_{j=1}^N \Gamma(\alpha_j)} \int \mathrm{d}_\square^N z \, \delta_z \frac{\prod_{j=1}^N z_j^{\alpha_j - 1}}{\left(\sum_{j=1}^N z_j A_j\right)^\alpha} \tag{B.7}$$

for $A_j = 1$, in which case one finds

$$\int \mathrm{d}_\square^N z \, \delta_z \prod_{j=1}^N z_j^{\alpha_j - 1} = \frac{\prod_{j=1}^N \Gamma(\alpha_j)}{\Gamma(\alpha)} \tag{B.8}$$

All phenomenologically relevant, non-zero cases for renormalizable gauge theories (working in Feynman gauge) are listed below:

$$I_1^{n,0,1} = -\frac{1}{2} S_{11} \tag{B.9}$$

$$I_1^{n,0,1;\mu_1} = \frac{1}{2} S_{11} \cdot r_1^{\mu_1} \tag{B.10}$$

$$I_2^{n,1,0} = -\frac{1}{6}\left(S_{11} + S_{12} + S_{22}\right) \tag{B.11}$$

$$I_2^{n,0,1} = 1 \tag{B.12}$$

$$I_2^{n,0,1;\mu_1} = -\frac{1}{2}\left(r_1^{\mu_1} + r_2^{\mu_1}\right) \tag{B.13}$$

$$\begin{aligned} I_2^{n,0,1;\mu_1\mu_2} &= \frac{1}{6}\left(2 r_1^{\mu_1} r_1^{\mu_2} + r_1^{\mu_1} r_2^{\mu_2} + r_2^{\mu_1} r_1^{\mu_2} + 2 r_2^{\mu_1} r_2^{\mu_2}\right) \\ &\quad - \frac{1}{12} \hat{g}^{\mu_1\mu_2}\left(S_{11} + S_{12} + S_{22}\right) \end{aligned} \tag{B.14}$$

$$I_3^{n,1,0} = \frac{1}{2} \tag{B.15}$$

$$I_3^{n,1,0;\mu_1} = -\frac{1}{6}\left(r_1^{\mu_1} + r_2^{\mu_1} + r_3^{\mu_1}\right) \tag{B.16}$$

$$I_3^{n,0,1;\mu_1\mu_2} = \frac{1}{4} \hat{g}^{\mu_1\mu_2} \tag{B.17}$$

$$I_3^{n,0,1;\mu_1\mu_2\mu_3} = -\frac{1}{12} \sum_{l=1}^{3} [\hat{g}^{\bullet\bullet} r^\bullet]^{\mu_1\mu_2\mu_3} \tag{B.18}$$

$$I_4^{n,1,0;\mu_1\mu_2} = \frac{1}{12} \hat{g}^{\mu_1\mu_2} \tag{B.19}$$

$$I_4^{n,2,0} = -\frac{1}{6} \tag{B.20}$$

$$I_4^{n,0,1;\mu_1\mu_2\mu_3\mu_4} = \frac{1}{4!} [\hat{g}^{\bullet\bullet} \hat{g}^{\bullet\bullet}]^{\mu_1\mu_2\mu_3\mu_4} \tag{B.21}$$

All other integrals of that type are identically zero.

# Appendix C The included Model Files

## C.1 Format of the Model Files

GoSam expects three files for a proper model definition:

| | |
|---|---|
| ⟨*model*⟩`.hh` : | Form file containing the Feynman rules |
| ⟨*model*⟩`.py` : | Python file |
| ⟨*model*⟩ : | (no extension) QGraf model file |

### C.1.1 The Python File

Thy Python file contains the following definitions

`model_name` : a variable of string type containing a human-readable name for this model, such as "Standard Model (Feyn. Gauge) w/o Higgs" etc.

`particles` : a Python `dict` that contains all particles *and* anti-particles of the model. The keys are the QGraf names of the fields; the values are objects of the class `Particle`. The constructor has the arguments

```
Particle(name, two_spin, mass, color_rep, partner, width='0')
```

`mnemonics` : a Python `dict` of human-readable particle names. The values are objects of the class `Particle`. It is save to refer to the dictionary `particles`.

`parameters` : a Python `dict` of model parameters with their default values. Both key and value are strings.

`functions` : a Python `dict` of variable names and initialization expressions. Both key and value are strings.

`types` : the types of all parameters and functions indicated by `'R'` for real numbers and `'C'` for complex numbers.

`latex_names` : a Python `dict` assigning LaTeX code to the field names. Math mode is assumed.

`line_styles` : a Python `dict` assigning line styles to field names. The line style used when drawing Feynman diagrams. Allowed values are `photon`, `ghost`, `scalar`, `gluon`, `fermion`.

### C.1.2 The QGraf File

The propagators in the QGraf file must contain the following functions:

| | |
|---|---|
| TWOSPIN : | twice the spin of the particle. |
| COLOR : | the color representation of the particle $\in \{1, 3, 8\}$. |
| MASS : | the mass of the particle. |
| WIDTH : | the width of the particle (currently not used). |
| AUX : | must be zero for most fields. Tensor Ghosts, as introduced by CalcHep have the value 1 here. |
| CONJ : | for self-conjugate particles the value is `('+')`, otherwise it is `('+','-')`. |

The vertices must provide all fields that should be accessible in VSUM statements and therefore also the ones that GoSam uses in the `order` option.

### C.1.3 The Form File

There are two possible ways of specifying the Feynman rules in the Form file. If a model contains only Standard Model like interactions one can make use of the file `src/form/vertices.hh` in the GoSam directory and just define the coefficients CL and CR in front of the vertices. This strategy is implemented by the modelfiles `models/sm`. The file Form contains a procedure `VertexConstants` which replaces the the vertex constants by their symbols. A QED example would be

```
#Procedure VertexConstants
   Id CL([field.em], [field.ep], [field.ph]) = e;
   Id CR([field.em], [field.ep], [field.ph]) = e;
#EndProcedure
```

In the header of the Form file all model specific symbols and functions need to be defined. For this simple model we have the fields and the coupling constant as only new symbols.

```
Symbols [field.em], [field.ep], [field.ph], e;
```

Instead of using the file `vertices.hh` one can also use his own vertex definitions. In this case the Form file must contain the definition

```
#Define USEVERTEXPROC "1"
```

and it must define the procedure `ReplaceVertices`. An example for QED is given below.

```
#Procedure ReplaceVertices
Identify Once vertex(iv?,
      [field.ep], idx1?, -1, k1?, idx1L1?, -1, idx1C1?,
      [field.em], idx2?,  1, k2?, idx2L1?,  1, idx2C1?,
      [field.ph], idx3?,  2, k3?, idx3L2?,  1, idx3C1?) =
   PREFACTOR(i_ * e) *
   NCContainer(Sm(idx3L2), idx1L1, idx2L1) *
   node(idx1, idx2, idx3);
#EndProcedure
```

It should be noted that `GoSam` expects the procedure `VertexConstants` to exist in both cases. If all the constants are already substituted inside `ReplaceVertices` the file must still provide a possibly empty empty implementation of `VertexConstants`. `GoSam` ensures that `VertexConstants` is always called after `ReplaceVertices`.

It is recommended to wrap any factors that are global prefactors to the diagram into the argument of the function `PREFACTOR` as `GoSam` scans for these functions and brackets them out. Each vertex definition must contain a factor `node` which contains the indices[1] of the fields at this vertex.

The `QGraf` style file generates vertex functions as follows:

$$\begin{gathered}\texttt{vertex}(\text{vertex index},\\ \text{field}_1, \text{index}_1, \pm 2\text{spin}_1, \text{momentum}_1, \mu_1, \pm\text{color rep}_1, \text{color index}_1,\\ \text{field}_2, \text{index}_2, \pm 2\text{spin}_2, \text{momentum}_2, \mu_2, \pm\text{color rep}_2, \text{color index}_2,\\ \vdots\\ \text{field}_n, \text{index}_n, \pm 2\text{spin}_n, \text{momentum}_n, \mu_n, \pm\text{color rep}_n, \text{color index}_n)\end{gathered}$$

The entries are:

| | |
|---|---|
| vertex index : | The unique index of this vertex. (`iv1`, `iv2`, ...) |
| field$_i$ : | The field name of the $i$-th particle. These names are constructed from the `QGraf` field name as `[field.`⟨*name*⟩`]`. |
| index$_i$ : | A unique name for this "ray" (at index 1 they are `idx1r1`, `idx1r2`, ...) |
| $\pm 2$spin$_i$ : | twice the spin of the $i$-th particle. The sign distinguishes particles (+) from antiparticles (−). |
| momentum$_i$ : | the incoming momentum of the $i$-th particle. |

[1] In `QGraf`'s terminology these indices are a combination of vertex and ray index of the field.

$\mu_i$ : the Lorentz index of the $i$-th particle. Depending on the spin of the particle this is a spinor index (spin 1/2), a Lorentz index (spin 1) or a dummy index (spin 0). For higher spins this index must be split into its components using the function `SplitLorentzIndex`. For its proper definition the reader is referred to the document `src/form/lorentz.pdf`.

$\pm$color rep$_i$ : the color representation of the $i$-th particle. Allowed values currently are $\pm 1, \pm 3, \pm 8$, although the sign only really makes sense for the fundamental representation 3 and its conjugate $\bar{3} \equiv -3$.

color index$_i$ : The color index of the $i$-th particle. Depending on the color representation this is an index in the fundamental, the adjoint or the trivial representation.

! $\rightarrow$ All symbols defined in `src/form/symbols.hh` are also accessible in this `Form` file. Note: until recently the definition of `Sqrt2` and `sqrt2` were part of the model file. Now these symbols are part of `src/form/symbols.hh` and must not be redefined.

! $\rightarrow$ All Dirac matrices and metric tensors must use the notation introduced by `spinney`. The metric tensor is $g^{\mu\nu} = \texttt{d}(\mu, \nu)$ and $\gamma^\mu = \texttt{Sm}(\mu)$, $\gamma_5 = \texttt{Gamma5}$, $\Pi_+ = \texttt{ProjPlus}$, $\Pi_- = \texttt{ProjMinus}$. All non-commuting objects must reside inside the function `NCContainter` (see example).

The color structure must use the objects $t^A_{ij} = \texttt{T}(A, i, j)$ (where the color flow is such that$j$ is the index of an anti-quark), $f^{ABC} = \texttt{f}(A, B, C)$ and $f^{ABE} f^{CDE} = \texttt{f4}(A, B, C, D)$. At vertices coupling colored with colorless particles it might be necessary to use the `d_` tensor to file the color flow through the vertex.

! $\rightarrow$ Note that all propagators and wave functions are defined in a model independent way in the files `src/form/propagators.hh` and `src/form/legs.hh`. Please, refrain from modifying these files directly but make all changes to `src/form/lorentz.nw`.

In theories with Maiorana fermions the model file should include the following line:

**#Define** DISPOSEQGRAFSIGN "1"

## C.2 Standard Model (`sm`)

### C.2.1 Synopsis

The model '`sm`' contains the Feynman rules for the Standard Model in Feynman gauge as described in [BDJ01, Appendix A].

### C.2.2 Particle Content

Leptons

| Name | Alternative Names | Mass | Comment |
|---|---|---|---|
| `ep` | `positron e+` | `me` | $e^+$ |
| `em` | `electron e-` | `me` | $e^-$ |
| `ne` | | 0 | $\nu_e$ |
| `nebar` | `ne~` | 0 | $\bar{\nu}_e$ |
| `mup` | `mu+` | `mmu` | $\mu^+$ |
| `mum` | `mu-` | `mmu` | $\mu^-$ |
| `nmu` | | 0 | $\nu_\mu$ |
| `nmubar` | `nmu~` | 0 | $\bar{\nu}_\mu$ |
| `taup` | `tau+` | `mtau` | $e^+$ |
| `taum` | `tau-` | `mtau` | $e^-$ |
| `ntau` | | 0 | $\nu_\tau$ |
| `ntaubar` | `ntau~` | 0 | $\bar{\nu}_\tau$ |

Quarks

| Name | Alternative Names | Mass | Comment |
|---|---|---|---|
| `U` | `u` | `mU` | $u$ |
| `Ubar` | `u~` | `mU` | $\bar{u}$ |
| `D` | d | `mD` | $d$ |
| `Dbar` | `d~` | mD | $\bar{d}$ |
| `S` | `s` | `mS` | $u$ |
| `Sbar` | `s~` | `mS` | $\bar{u}$ |
| `C` | c | `mC` | $d$ |
| `Cbar` | `c~` | mC | $\bar{d}$ |
| `T` | `t` | `mT` | $t$ |
| `Tbar` | `t~` | `mT` | $\bar{t}$ |
| `B` | b | `mB` | $b$ |
| `Bbar` | `b~` | mB | $\bar{b}$ |

Gauge Bosons

| Name | Alternative Names | Mass | Comment |
|---|---|---|---|
| `g` | `gluon` | 0 | $g$ |
| `A` | `photon gamma` | 0 | $\gamma$ |
| `Z` | | `mZ` | $Z$ |
| `Wp` | `W+` | `mW` | $W^+$ |
| `Wm` | `W-` | `mW` | $W^-$ |

Scalar Bosons

| Name | Alternative Names | Mass | Comment |
|---|---|---|---|
| `H` | `h higgs` | `mH` | $H$ |
| `phim` | `phi-` | `mW` | $\phi^-$ |
| `phip` | `phi+` | `mW` | $\phi^+$ |
| `chi` | | `mZ` | $\chi$ |

Ghost Fields

| Name | Alternative Names | Mass | Comment |
|---|---|---|---|
| `gh` | | 0 | $u^g$ |
| `ghbar` | | 0 | $\bar{u}^g$ |
| `ghA` | | 0 | $u^A$ |
| `ghAbar` | | 0 | $\bar{u}^A$ |
| `ghZ` | | `mZ` | $u^Z$ |
| `ghZbar` | | `mZ` | $\bar{u}^Z$ |
| `ghWp` | | `mW` | $u^+$ |
| `ghWpbar` | | `mW` | $\bar{u}^+$ |
| `ghWm` | | `mW` | $u^-$ |
| `ghWmbar` | | `mW` | $\bar{u}^-$ |

### C.2.3 Parameters

This section lists all model parameters which are not already listed as particle masses.

| Name | Symbol | Description |
|---|---|---|
| `NC` | $N_C$ | Number of colors in QCD |
| `e` | $e$ | electro-weak coupling constant: $\alpha = e^2/(4\pi)$ |
| `gs` | $g_s$ | strong coupling constant: $\alpha_s = g_s^2/(4\pi)$ |
| `sw` | $s_w = \sin\theta_w$ | sine of weak mixing angle |
| `cw` | $c_w = \cos\theta_w$ | cosine of weak mixing angle |
| `VUD` | $V_{ud}$ | CKM mixing matrix element |
| `CVDU` | $V^\dagger_{du}$ | — ” — |
| `VUS` | $V_{us}$ | — ” — |
| `CVSU` | $V^\dagger_{su}$ | — ” — |
| `VUB` | $V_{ub}$ | — ” — |
| `CVBU` | $V^\dagger_{bu}$ | — ” — |
| `VCD` | $V_{cd}$ | — ” — |
| `CVDC` | $V^\dagger_{dc}$ | — ” — |
| `VCS` | $V_{cs}$ | — ” — |
| `CVSC` | $V^\dagger_{sc}$ | — ” — |
| `VCB` | $V_{cb}$ | — ” — |
| `CVBC` | $V^\dagger_{bc}$ | — ” — |
| `VTD` | $V_{td}$ | — ” — |
| `CVTD` | $V^\dagger_{dt}$ | — ” — |
| `VTS` | $V_{ts}$ | — ” — |
| `CVST` | $V^\dagger_{st}$ | — ” — |
| `VTB` | $V_{tb}$ | — ” — |
| `CVTB` | $V^\dagger_{bt}$ | — ” — |

# Appendix D Template for a Process Setup File

→ Chapter 3.1

In order to create a new process setup file one can invoke

```
$ gosam.py --template your_new_file.in
```

This is the recommended way of obtaining the most recent documentation of the available options.

The syntax of a general process setup file should obey the following rules:

- A setup file (*process card*) consists of a sequence of lines representing key-value pairs. A key-value pair can span across several lines if each of the lines except the last line is terminated by a backslash.
- A setup file is allowed to contain any number of blank lines or comment lines, indicated by a '`!`' or a '`#`' as its first non-blank character.
- The key and the value are separated by a blank, a colon '`:`' or an equals sign '`=`'. Notice that the line '`key␣=␣value`' will be interpreted as the key '`key`' followed by the value '`=␣value`' as the terminator of the key is the blank and not the equals sign. In order to produce one of the terminators literally as a part of the key one has to escape it with a backslash, e.g. '`very\␣long\␣name:value`' would translate to the key '`very␣long␣name`' and the value '`value`'.
- The escape characters '`\\`', '`\n`', '`\r`', '`\f`' and '`\t`' work as usual. Backslashes in front of any other character are just dropped.
- Leading and trailing blanks are removed from the key and the value by default and must be escaped to preserve them. Whitespace is also removed in front and after commas if the value is interpreted as a comma separated list.
- If an option expects a logical value, the literals '`1`', '`true`', '`.true.`', '`t`', '`.t.`', '`yes`' and '`.y.`' are recognized as the value *true*. These values are interpreted case-insensitively. If a value is not recognized as *true* it corresponds to *false*.

! → Note, that deviating from the Java standard, unicode escapes, such as '`\u10EF`', have not been implemented; neither are octal and hexadecimal escape sequences recognized.

`process_name` : (*text*)

```
A symbolic name for this process. This name will be used
as a prefix for the Fortran modules.

Golem will insert an underscore after this prefix.
If the process name is left blank no prefix will be used
and no extra underscore will be generated.
```

`process_path` : (*text*)

```
The path to which all Form output is written.
If no absolute path is given, the path is interpreted relative
to the working directory from which golem-main.py is run.

Example:
process_path=/scratch/golem_processes/process1
```

`in` : (*comma separated list*)

```
A comma-separated list of initial state particles.
Which particle names are valid depends on the
model file in use.

Examples (Standard Model):
1) in=u,u~
2) in=e+,e-
3) in=g,g
```

`out` : (*comma separated list*)

```
A comma-separated list of final state particles.
Which particle names are valid depends on the
model file in use.

Examples (Standard Model):
1) out=H,u,u~
2) out=e+,e-,gamma
3) out=b,b~,t,t~
```

`model` : (*comma separated list*)

```
This option allows the selection of a model for the
Feynman rules. It has to conform with one of four possible
formats:

1) model=<name>
2) model=<path>, <name>
3) model=<path>, <number>
4) model=FeynRules, <path>

Format 1) searches for the model files <name>, <name>.hh
and <name>.py in the models/ directory under the installation
```

```
path of Golem.

Format 2) is similar to format 1) but <path> is used instead
of the models/ directory of the Golem installation

Format 3) expects the files func<number>.mdl, lgrng<number>.mdl,
prtcls<number>.mdl and vars<number>.mdl in the directory <path>.
These files need to be in CalcHEP/CompHEP format.

Format 4) expects files according to the new FeynRules Python
interface in the directory specified by <path>.
(Not fully implemented yet)
```

`model.options` : (*comma separated list*)

```
If the model in use supports options they can be passed via this
property.
```

`order` : (*comma separated list*)

```
A 3-tuple <coupling>,<born>,<virt> where <coupling> denotes
a function of the qgraf style file which can be used as
an argument in a 'vsum' statement. For the standard model
file 'sm' there are two such functions, 'gs' which counts
powers of the strong coupling and 'gw' which counts powers
of the weak coupling. <born> is the sum of powers for the
tree level amplitude and <virt> for the virtual amplitude.
The line
   order = gs, 4, 6
would select all diagrams which have (gs)^4 at tree level
and all loop graphs with (gs)^6.

Note: The line
   order = gw, 2, 2
does not imply that no virtual corrections are calculated.
Instead, for the virtual corrections diagrams are chosen
with the same order in gw but higher order in gs.

In other models with more than two different coupling
constants additional 'vsum' statements, which can be passed
via the qgraph.verbatim option, might be needed
to select the correct set of diagrams.

If the last number is omitted no virtual corrections are
calculated.

See also: qgraf.options, qgraf.verbatim
```

`zero` : (*comma separated list*)

```
A list of symbols that should be treated as identically
zero throughout the whole calculation. All of these
```

```
symbols must be defined by the model file.

Examples:
1) # Light masses are set to zero here:
   zero=me,mU,mD,mS
2) # Diagonal CKM matrix:
   zero=VUS, VUB, CVDC, CVDT, \
        VCD, VCB, CVSU, CVST, \
        VTD, VTS, CVBU, CVBC
   one=  VUD,  VCS,  VTB, \
        CVDU, CVSC, CVBT

See also: model, one
```

`one` : (*comma separated list*)

```
A list of symbols that should be treated as identically
one throughout the whole calculation. All of these
symbols must be defined by the model file.

Example:
one=gs, e

See also: model, zero
```

`helicities` : (*comma separated list*)

```
A list of helicities to be calculated. An empty list
means that all possible helicities should be generated.

The helicities are specified as a string of characters
according to the following table:

   spin massive  |  'm'  '-'  '0'   '+'   'k'
     0   YES/NO  | ---- ----    0  ----  ----
   1/2   YES/NO  | ---- -1/2 ----  +1/2  ----
     1     NO    | ----   -1 ----    +1  ----
     1    YES    | ----   -1    0    +1  ----
   3/2     NO    | -3/2 ---- ----  ----  +3/2
   3/2    YES    | -3/2 -1/2 ----  +1/2  +3/2
     2     NO    |   -2 ---- ----  ----    +2
     2    YES    |   -2   -1    0    +1    +2

Please, note that 'k' and 'm' are not in use yet but reserved
for future extensions to higher spins.

The characters correspond to particle 1, 2, ... from left to
right.

Examples:
   # e+, e- --> gamma, gamma:
```

```
   # Only three helicities required; the other ones are
   # either zero or can be obtained by symmetry
   # transformations.
   helicities=+-++,+-+-,+---;

Multiple helicities can be encoded in patterns, which are expanded
at the time of code generation. Patterns can have one of the following
forms:
   [+-], [+-0], [+0] etc. : the bracket expands to one of the symbols
         in the bracket at a time.
   EXAMPLE
         helicities=[+-]+[+-0]
         # expands to 6 different helicities:
         # helicities=+++, ++-, ++0, -++, -+-, -+0
   [a=+-], etc. : as above, but the helicity is also assigned to the
         symbol and can be reused.
   EXAMPLE
         helicities=[i=+-]+i+
         # expands to two helicities
         # helicities=++++, -+-+
   [ab=+-0], etc. : as above, the first symbol is assigned the helicity,
         the second is minus the helicity
   EXAMPLE
         helicities=[qQ=+-][pP=+-]PQ[+-0]
         # expands to 12 helicities
         # helicities=++--+,++---,++--0,+-+-+,+-+--,+-+-0,\
         #             -+-++,-+-+-,-+-+0,--+++,--++-,--++0
```

`qgraf.options` : (*comma separated list*)

```
A list of options which is passed to qgraf via the 'options' line.
Possible values (as of qgraf.3.1.1) are zero, one or more of:
   onepi, onshell, nosigma, nosnail, notadpole, floop
   topol

Please, refer to the QGraf documentation for details.
```

`qgraf.verbatim` : (*text*)

```
This option allows to send verbatim lines to
the file qgraf.dat. This can be useful if the user
wishes to put additional restricitons to the selected diagrams.
This option is mainly inteded for the use of the operators
   rprop, iprop, chord, bridge, psum
Note, that the use of 'vsum' might interfer with the
option qgraf.power.

Example:
qgraf.verbatim=\
   # no top quarks: \n\
   true=iprop[T, 0, 0];\n\
   # at least one Higgs:\n\
   false=iprop[H, 0, 0];\n
```

Please, refer to the QGraf documentation for details.

See also: qgraf.options, order

`qgraf.verbatim.lo` : (*text*)

Same as qgraf.verbatim but only applied to LO diagrams.

See also: qgraf.verbatim, qgraf.verbatim.nlo

`qgraf.verbatim.nlo` : (*text*)

Same as qgraf.verbatim but only applied to LO diagrams.

See also: qgraf.verbatim, qgraf.verbatim.nlo

`qgraf.bin` : (*text*)

Points to the QGraf executable.

```
Example:
qgraf.bin=/home/my_user_name/bin/qgraf
```

Default: `qgraf`

`form.bin` : (*text*)

Points to the Form executable.

```
Examples:
1) # Use TForm:
   form.bin=tform
2) # Use non-standard location:
   form.bin=/home/my_user_name/bin/form
```

Default: `form`

`form.tempdir` : (*text*)

Temporary directory for Form. Should point to a directory on a local disk.

```
Examples:
form.tempdir=/tmp
form.tempdir=/scratch
```

Default: `/tmp`

`haggies.bin` : (*text*)

Points to the Haggies executable.
Haggies is used to transform the expressions of the diagrams into optimized Fortran90 programs. It can be obtained from

```
   http://www.nikhef.nl/~thomasr/download.php

Examples:
   1) haggies.bin=/home/my_user_name/bin/haggies
   2) haggies.bin=/usr/bin/java -Xmx50m -jar ./haggies.jar
```

Default: `java -jar /home/pcl247e/jfsoden/bin/local/share/golem/haggies/haggies.jar`

`fc.bin` : (*text*)

```
Denotes the executable file of the Fortran90 compiler.
```

Default: `gfortran`

`group` : (*true/false*)

```
Flag whether or not the tree-level diagrams should be grouped
into a single file.
```

Default: `True`

`extensions` : (*comma separated list*)

```
A list of extension names which should be activated for the
code generation. These names are not standardised at the moment.

One option which is affected by this is LDFLAGS. In the following
example only ldflags.looptools is added to the LDFLAGS variable
in the makefiles whereas the variable ldflags.qcdloop is ignored.

extensions=golem95,samurai

ldflags.qcdloops=-L/usr/local/lib -lqcdloop

NOTE: Make sure you activate at least one of 'samurai' and 'golem95'.

Currently active extensions:

samurai      --- use Samurai for the reduction
golem95      --- use Golem95 for the reduction
pjfry        --- use PJFry for the reduction (experimental)
dred         --- use four dimensional algebra (dim. reduction)
autotools    --- use Makefiles generated by autotools
qshift       --- apply the shift of Q already at the FORM level
topolynomial --- (with FORM >= 4.0) use the ToPolynomial command
gaugecheck   --- modify gauge boson wave functions to allow for
                 a limited gauge check (introduces gauge*z variables)
olp_daemon   --- (OLP interface only): generates a C-program providing
                 network access to the amplitude
no-fr5       --- do not generate finite gamma5 renormalisation
numpolvec    --- evaluate polarisation vectors numerically
f77          --- in combination with the BLHA interface it generates
                 an olp_module.f90 linkable with Fortran77
```

`templates` : (*text*)

```
Path pointing to the directory containing the template
files for the process. If not set golem uses the directory
<golem_path>/templates.

The directory must contain a file called 'template.xml'
```

`debug` : (*comma separated list*)

```
A list of debug flags.
Currently, the words 'lo', 'nlo' and 'all' are supported.
```

`golem95.fcflags` : (*text*)

```
FCFLAGS required to compile with golem95.

Example:
golem95.fcflags=-I/usr/local/include/golem95
```

Default: `` `pkg-config --cflags golem` ``

`golem95.ldflags` : (*text*)

```
LDFLAGS required to link golem95.

Example:
golem95.ldflags=-L/usr/local/lib/ -lgolem-gfortran-double
```

Default: `` `pkg-config --libs golem` ``

`samurai.fcflags` : (*text*)

```
FCFLAGS required to compile with samurai.

Example:
samurai.fcflags=-I/usr/local/include/samurai
```

`samurai.ldflags` : (*text*)

```
LDFLAGS required to link samurai.

Example:
samurai.ldflags=-L/usr/local/lib/ -lsamurai-gfortran-double
```

`samurai.version` : (*text*)

```
The version of the samurai library in use.

Example:
samurai.version=2.1.0
```

Default: `2.1.1`

`select.lo` : (*comma separated list*)

```
A list of integer numbers, indicating leading order diagrams to be
selected. If no list is given, all diagrams are selected.
```

```
Otherwise, all diagrams not in the list are discarded.

The list may contain ranges:

select.lo=1,2,5:10:3, 50:53

which is equivalent to

select.lo=1,2,5,8,50,51,52,53

See also: select.nlo, filter.lo, filter.nlo
```

Default: ,

`select.nlo` : (*comma separated list*)

```
A list of integer numbers, indicating one-loop diagrams to be selected.
If no list is given, all diagrams are selected.
Otherwise, all diagrams   not in the list are discarded.

The list may contain ranges:

select.nlo=1,2,5:10:3, 50:53

which is equivalent to

select.nlo=1,2,5,8,50,51,52,53

See also: select.lo, filter.lo, filter.nlo
```

Default: ,

`filter.lo` : (*text*)

```
A python function which provides a filter for tree diagrams.

filter.lo=lambda d: d.iprop(Z) == 1 \
   and d.vertices(Z, U, Ubar) == 0

The following methods of the diagram class can be used:

* d.rank() = the maximum rank in Q possible for this diagram
* d.loopsize() = the number of propagators in the loop
* d.vertices(field1, field2, ...) = number of vertices
    with the given fields
* d.loopvertices(field1, field2, ...) = number of vertices
    with the given fields; only those vertices which have
    at least one loop propagator attached to them
* d.iprop(field, momentum="...", twospin=..., massive=True/False,
                                                       color=...) =
    the number of propagators with the given properties:
     - field: a field or list of fields
```

```
      - momentum: a string denoting the momentum through this propagator,
             such as "k1+k2"
      - twospin: two times the spin (integer number)
      - massive: select only propagators with/without a non-zero mass
      - color: one of the numbers 1, 3, -3 or 8, or a list of
               these numbers
  * d.chord(...) = number of loop propagators with the given properties;
      the arguments are the same as in iprop
  * d.bridge(...) = number of non-loop propagators with the given
      properties; the arguments are the same as in iprop

  See also: filter.nlo, select.lo, select.nlo
```

`filter.nlo` : *(text)*

```
A python function which provides a filter for loop diagrams.

See filter.lo for more explanation.
```

`filter.module` : *(text)*

```
A python file of predefined functions which should be available
in filters.

Example:

filter.module=filter.py
filter.nlo=my_nlo_filter("vertices.txt")
filter.lo=my_nlo_filter("vertices.txt")

------ filter.py -----

class my_nlo_filter_class:
   def __init__(self, fname):
      self.fields = []
      f = open(fname, 'r')
      for line in f.readlines():
         fields = map(lambda s: s.strip(),
               line.split(","))
         self.fields.append(fields)
      f.close()

   def __call__(self, diag):
      for lst in self.fields:
         if diag.vertices(*lst) > 0:
            return False
      return True

----------------------

See filter.lo, filter.nlo
```

`renorm_beta` : (*true/false*)

```
Set the name of the same variable in config.f90

Activates or disables beta function renormalisation

QCD only
```

Default: `True`

`renorm_mqwf` : (*true/false*)

```
Set the name of the same variable in config.f90

Activates or disables UV countertems coming from
external massive quarks

QCD only
```

Default: `True`

`renorm_decoupling` : (*true/false*)

```
Set the name of the same variable in config.f90

Activates or disables UV counterterms coming from
massive quark loops

QCD only
```

Default: `True`

`renorm_mqse` : (*true/false*)

```
Set the name of the same variable in config.f90

Activates or disables the UV counterterm coming from the
massive quark propagators

QCD only
```

Default: `True`

`renorm_logs` : (*true/false*)

```
Set the name of the same variable in config.f90

Activates or disables the logarithmic finite terms
of all UV counterterms

QCD only
```

Default: `True`

`renorm_gamma5` : (*true/false*)

```
Set the same variable in config.f90

Activates finite renormalisation for axial couplings in the
't Hooft-Veltman scheme

QCD only, works only with built-in model files.
```

Default: `True`

`reduction_interoperation` : (*integer number*)

```
Set the same variable in config.f90. A value of '-1' lets gosam
decide depending on the specified extensions.

See common/config.f90 for details.
```

Default: `-1`

`nlo_prefactors` : (*integer number*)

```
Set the same variable in config.f90. The values have the following
meaning:

0: A factor of alpha_(s)/2/pi is not included in the NLO result
1: A factor of 1/8/pi^2 is not included in the NLO result
2: The NLO includes all prefactors

Note, however, that the factor of 1/Gamma(1-eps) is not included
in any of the cases.
```

Default: `0`

`reference-vectors` : (*comma separated list*)

```
A list of reference vectors for massive and higher spin particles.
For vectors which are not assigned here, the program picks a reference
vector automatically.

Each entry of the list has to be of the form <index>:<index>

EXAMPLE

in=g,u
out=t,W+
reference-vectors=1:2,3:4,4:3

In this example, the gluon (particle 1) takes the momentum k2
as reference momentum for the polarisation vector. The massive
top quark (particle 3) uses the light-cone projection l4 of the
W-boson as reference direction for its own momentum splitting.
Similarly, the momentum of the W-boson is split into a direction
```

```
l4 and one along l3.

If cycles are generated in the list (l3 has to be known in order
to determine l4 and vice versa in the above example) they must be
at most of length two. For the reference momenta of lightlike
gauge bosons the length of cycles does not matter, e.g.

in=g,g
out=g,g
reference-vectors=1:2,2:3,3:4,4:1
```

`abbrev.limit` : (*text*)

```
Maximum number of instructions per subroutine when calculating
abbreviations, if this number is positive.
```

Default: `0`

`abbrev.level` : (*text*)

```
The level at which abbreviations are generated. The value should be
one of:
   helicity       generates files helicity<X>/abbrevh<X>.f90
   group          generates files helicity<X>/abbrevg<G>h<X>.f90
   diagram        generates files helicity<X>/abbrevd<D>h<X>.f90
```

Default: `helicity`

`r2` : (*text*)

```
The algorithm how to treat the R2 term:

implicit    -- mu^2 terms are kept in the numerator and reduced
               at runtime
explicit    -- mu^2 terms are reduced analytically
only        -- same as 'explicit' but only the R2 term is kept in
               the result
off         -- all mu^2 terms are set to zero
```

Default: `implicit`

`symmetries` : (*comma separated list*)

```
Specifies the symmetries of the amplitude.

This information is used when the list of helicities is generated.

Possible values are:

* flavour    -- no flavour changing interactions
         When calculating the list of helicities, fermion lines
    of PDGs 1-6 are assumed not to mix.

* family     -- flavour changing only within families
```

```
          When calculating the list of helicities, fermion lines
     of PDGs 1-6 are assumed to mix only within families,
     i.e. a quark line connecting a up with a down quark would
     be considered, while up-bottom is not.
* lepton     -- means for leptons what 'flavour' means for quarks
* generation -- means for leptons what 'family' means for quarks
* parity     -- the amplitude is invariant under parity tranformation.
                === Parity is not implemented yet.
* <n>=<h>    -- restriction of particle helicities,
        e.g. 1=-, 2=+ specifies helicities of particles 1 and 2
* %<n>=<h>   -- restriction by PDG code,
        e.g. %23=+- specifies the helicity of all Z-bosons to be
        '+' and '-' only (no '0' polarisation).

        %<n> refers to both +n and -n
        %+<n> refers to +n only
        %-<n> refers to -n only
```

`crossings` : (*comma separated list*)

```
A list of crossed processes derived from this process.

For each process in the list a module similar to matrix.f90 is
generated.

Example:

process_name=ddx_uux
in=1,-1
out=2,-2

crossings=dxd_uux: -1 1 > 2 -2, ud_ud: 2 1 > 2 1
```

`pyxodraw` : (*true/false*)

```
Specifies whether to draw any diagrams or not.
```

Default: `True`

## Conditions of Use

Scientific publications prepared using the present version of GoSam or any modified version of it or any code linking to GoSam or parts of it should make a clear reference to the publication:

> G. Cullen et al.,
> "Automated One-Loop Calculations with GoSam,"
> arXiv:1111.2034 [hep-ph]

### The GNU General Public License Version 3

## Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program–to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

## Terms and Conditions

0. Definitions.

   "This License" refers to version 3 of the GNU General Public License.

   "Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

   "The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

   To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

   A "covered work" means either the unmodified Program or a work based on the Program.

   To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

   To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

   An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

   The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

   A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

   The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

   All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

   You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

   Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

   No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

   When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

   You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

   You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

   You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

   (a) The work must carry prominent notices stating that you modified it, and giving a relevant date.

   (b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".

   (c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

   (d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

   A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

   You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

   (a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

(b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

(c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

(d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

(e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A “User Product” is either (1) a “consumer product”, which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, “normally used” refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

“Installation Information” for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use

(e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

(f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

   You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

   However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

   Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

   Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

   You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License.

Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

    Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

    An “entity transaction” is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party’s predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

    You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

    A “contributor” is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor’s “contributor version”.

    A contributor’s “essential patent claims” are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, “control” includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

    Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor’s essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

    In the following three paragraphs, a “patent license” is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To “grant” such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

    If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to

## End of Terms and Conditions

How to Apply These Terms to Your New Programs

# Bibliography

[BBD$^+$10] T. Binoth, F. Boudjema, G. Dissertori, A. Lazopoulos, A. Denner, et al., *A Proposal for a standard interface between Monte Carlo tools and one-loop programs*, Comput.Phys.Commun. **181** (2010), 1612–1622, Dedicated to the memory of, and in tribute to, Thomas Binoth, who led the effort to develop this proposal for Les Houches 2009.

[BDJ01] Manfred Böhm, Ansgar Denner, and Hans Joos, *Gauge theories of the strong and electroweak interaction*, 3rd ed., Teubner, Stuttgart and Leipzig and Wiesbaden, 2001.

[BGH$^+$09] T. Binoth, J. Ph. Guillet, G. Heinrich, E. Pilon, and T. Reiter, *Golem95: a numerical program to calculate one-loop tensor integrals with up to six external legs*, Comput. Phys. Commun. **180** (2009), 2317–2330.

[CGH$^+$11] G. Cullen, J.Ph. Guillet, G. Heinrich, T. Kleinschmidt, E. Pilon, et al., *Golem95C: A library for one-loop integrals with complex masses*, Comput.Phys.Commun. **182** (2011), 2276–2284.

[DDF$^+$11] Celine Degrande, Claude Duhr, Benjamin Fuks, David Grellscheid, Olivier Mattelaer, et al., *UFO - The Universal FeynRules Output*, * Temporary entry *.

[FR11] J. Fleischer and T. Riemann, *A complete algebraic reduction of one-loop tensor Feynman integrals*, Phys. Rev. **D83** (2011), 073004.

[Mal70] Michael A. Malcolm, *An algorithm for floating-point accumulation of sums with small relative error*, Tech. report, Stanford University, Stanford, CA, USA, 1970.

[MORT10] P. Mastrolia, G. Ossola, T. Reiter, and F. Tramontano, *Scattering AMplitudes from Unitarity-based Reduction Algorithm at the Integrand-level*, JHEP **08** (2010), 080.

[Nog93] Paulo Nogueira, *Automatic Feynman graph generation*, J. Comput. Phys. **105** (1993), 279–289.

[Ohl95] Thorsten Ohl, *Drawing Feynman diagrams with Latex and Metafont*, Comput. Phys. Commun. **90** (1995), 340–354.

[OPP08] Giovanni Ossola, Costas G. Papadopoulos, and Roberto Pittau, *On the Rational Terms of the one-loop amplitudes*, JHEP **0805** (2008), 004.

[Rei09] Thomas Reiter, *Automated Evaluation of One-Loop Six-Point Processes for the LHC.*

[Rei10] ———, *Optimising Code Generation with haggies*, Comput.Phys.Commun. **181** (2010), 1301–1331.

[Sem10] A. Semenov, *LanHEP - a package for automatic generation of Feynman rules from the Lagrangian. Updated version 3.1.*

[Ver94] J. A. M. Vermaseren, *Axodraw*, Comput. Phys. Commun. **83** (1994), 45–58.

[Ver00] ———, *New features of FORM.*

[Yun11] Valery Yundin, ***** working title *****, Ph.D. thesis, DESY Zeuthen, 2011, PhD thesis.