GoSam 2.0 Manual

The GoSam Collaboration

Version May 22, 2015

Contents

1 Introduction

GoSam is a program package for the automated generation and evaluation of one-loop amplitudes, within and beyond the Standard Model. Version 1.0 has been published in Ref. [1], version 2.0 in Ref. [2]. GoSam-2.0 can also produce spin- and colour correlated tree amplitudes. The program produces `Fortran 95` code for a given process by generating Feynman diagrams and translating the corresponding one-loop expressions into a form where the integrand can be reduced and evaluated numerically with either Ninja [3, 4, 5] or `golem95C` [6, 7, 8] or Samurai [9, 10]. A file containing the pictorial represenation of the diagrams along with other information about the process is also produced.

In this manual, shell commands (for the `bash` shell) are indicated by lines starting with a dollar sign `$`. 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 '`>>>`' prompts, with '`...`' for continuation lines.

2 Download and installation

2.1 Prerequisites

The distribution of GoSam-2.0, together with the install script, will provide all external and auxiliary programs which are necessary to successfully run GoSam. Therefore, the user does not have to install any external programs manually.

The program GoSam is designed to run in any modern Unix-like environment (Linux, Mac).
The system requirements are `Python` ($\geq$ 2.6), a `fortran` compiler (gfortran or ifort), a C/C++ compiler (gcc/icc), and (GNU) `make`. By default, GoSam uses the gfortran/gcc compilers from the GNU Compiler Suite. To use an Intel compiler (ifort/icc), the `--intel` option can be used. Specific paths to the compilers can be provided using the `--fc, --cc, --cxx` options.

2.2 Download

The GoSam-2.0 source code will be downloaded automatically by the install script `gosam_installer.py`. The install script can be downloaded by
$ `wget http://gosam.hepforge.org/gosam-installer/gosam_installer.py`
or by going to the webpage
`http://gosam.hepforge.org/` and clicking on `installation script`.

The GoSam-2.0 package can also be downloaded manually either via `subversion` or via download from the HepForge webpage.

HTTP Download

At the URL
`http://www.hepforge.org/downloads/gosam/`
one can download the package GoSam-2.0 as a tar-ball. one can unpack it using the command

```
$ tar xzvf gosam-2.0.tar.gz
```

Subversion

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

```
$ svn co http://svn.hepforge.org/gosam/branches/
↪gosam-2.0
```

This will create a folder `gosam-2.0` in your current directory. Authenticated users can use

```
$ svn co svn+ssh://svn.hepforge.org/hepforge/svn/
↪gosam/branches/gosam-2.0
```

to gain read and write access to the project files.

2.3 Installation

The installation of GoSam-2.0 is very simple when using the installation script. The latter can be downloaded by
`$ wget http://gosam.hepforge.org/gosam_installer.py`
By default GoSam and the external programs it uses will be installed into a subfolder `./local` of the current directory. A different path can be specified using the `--prefix=PATH_where_to_install` option.

To run the script the user should execute the following commands

```
$ chmod +x gosam_installer.py
$ ./gosam_installer.py [--prefix=...]
```

or

```
$ python gosam_installer.py [--prefix=...]
```

Upon installation, the installer will ask some questions, which are described in detail below. To use the default installation all the questions can be "answered" by hitting the `ENTER` key.

As soon as all questions are answered, the main installation process will start. The components will be downloaded, built and installed. The whole procedure takes about 10-30 minutes.

At the end of the installation process, a script `gosam_setup_env.sh` will be created in the `bin/` subdirectory of the install location, which will (temporarily) set all environment variables, as soon as the script is sourced into a shell, by
`$ source [path]/gosam_setup_env.sh`. The installer also gives a recommendation how these environment variables can be set permanently. The script can be used in all `tcsh`- and `bash`-compatible shells.

All files which have been installed are tracked in the file `installer-log.ini`. Please keep this file and the install script. They are needed to update and uninstall GoSam.

For the default installation, internet access is required.

2.3.1 Additional information about the installation process

Before the installation starts, the installer first checks if a new version of the installation script is available on the GoSam webpage, and if this is the case, asks if it is allowed to update the script.

Then the installer searches for existing components (`QGraf`, `FORM`).

If they are not found, one can either press `ENTER` to have them installed by the script, or provide a path to the binary (tab-completion can be used).

If they are found, their version is checked, and if needed the installation of a version which has been tested to run with GoSam is suggested.

All files are downloaded to the `GoSam` subfolder and extracted. The build files remain on the system; they can be removed manually.

By default, GoSam uses the gfortran/gcc from the GNU Compiler Suite compiler. If one wants to use the compiler from Intel (ifort/icc), the –intel option can be used. Specific paths to the compilers can be provided using the `--fc,` `--cc` and `--cxx` option.

All installation options can be listed with the –help flag:

```
$ ./gosam_installer.py --help
```

2.3.2 Updating an existing installation

To update an existing installation, the installer can be started again without any option. It determines the install location from the file `installer-log.ini` which is searched for in the current directory. The script will look online for updates and ask if they should be installed.

Before overwriting or deleting existing files which have been modified by the user, the installer stops. One can use the `-f` flag to force the action.

2.3.3 Uninstalling

To uninstall GoSam, including the installed auxiliary libraries, the installer needs to be started with the `-u` or `--uninstall` flag.

As in the case of updates, the install script searches for the file `installer-log.ini` in the current directory.

Modified files are not deleted. Use the `-f` flag to force the action of deleting them.

2.4 Description of the components

The generation of matrix element code using GoSam can be understood as a three step process.

1. **diagram generation**: `Python` and `QGraf` are used. This phase is initiated by running `gosam.py process.in`, where `process.in` contains the user input for the process to be calculated.

2. **code generation**: only FORM and haggies are run. This phase is initiated by `$ make source`.

3. **compilation and running**: a fortran compiler and the chosen reduction libraries are used. This phase is initiated by `$ make compile`. Please note that running `make compile` will invoke `make source` if the latter has not been run before, or if some of the source files are missing.

If you use the GoSam package, you should be aware that the following programs are used. (The numbers indicate during which phase of the code generation the tools will be required).

QGraf (1) QGraf [11] is required in version 3.1 or higher. The install script coming with GoSam will download and install QGraf automatically. It can be also downloaded manually from `http://cfif.ist.utl.pt/~paulo/qgraf.html`.

Python (1) The program has been tested with Python versions 2.6 and 2.7.

FORM (2) FORM [12, 13] version 4.0 or higher is required to profit from all optimisation features. FORM is distributed with the GoSam-2.0 package for the user's convenience. For manual download, FORM is available from `http://www.nikhef.nl/~form/`.

haggies (2) The code generator haggies [14] is also 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.

Ninja/golem95C/Samurai (3) For one-loop calculations, at least one of these three libraries is required. The libraries are already distributed with the code and compiled by the install script. Alternatively,

- golem95C can be downloaded from `https://golem.hepforge.org`.
- Samurai can be downloaded from `http://samurai.hepforge.org`.
- Ninja can be downloaded from `http://ninja.hepforge.org`.

refrep.cls (3) The generation of documentation (optional) is based on the LaTeX-class refrep, which may not be present in 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 like drawing the diagrams, listing the colour structures, etc.

! → Please note that some of these programs may 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.

3 Setup of a process

GoSam can be used either as a standalone code producing one-loop (and tree level) amplitudes, or it can be used as a *One Loop Provider* (OLP) in combination with a Monte Carlo program. The usage in the latter case is described in detail in Section 7. Below we will first describe the setup for the standalone version.

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

Before we give a commented list of all possible entries in a *process card*, we should emphasize that almost all specifications in the *process card* are options, which will take default values if they are not filled in by the user. The paths to the libraries will be inserted automatically by the install script. The only mandatory fields are the `in` and `out` particles, the perturbative order and the path where to store the process files. Therefore, a *minimal process card* can look like this:

Listing 3.1: `eett.in`

```
process_path=eett
in=     e+, e-
out=    t, t~
order= gs, 0, 2
```

3.1 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 generates the file `eett.in` with some documentation for all defined options. Some options, e.g. the paths to the reduction libraries, have been set by the installation script. The other options are filled with some default values, which also can be changed either in the individual process cards, or in a global configuration file[1].

In the following example it is assumed that the process $e^+e^- \to t\bar{t}$ should be calculated to order $\mathcal{O}(\alpha\alpha_s)$ (virtual QCD corrections).

[1] For the latter, the script will search (in this order) in the GoSam directory, in the user's home directory and in the current working directory for a file named '`.gosam`' or '`gosam.in`'. If one needs to adapt paths by hand, such a file can be generated with the command `$ gosam-config.py > gosam.in`

We neglect the exchange of a Z or a Higgs boson and treat the electron as massless. The output directory is assumed to be in the relative path `eett`.

Listing 3.2: `eett.in`

```
process_name=eett
process_path=eett
in=    e+, e-
out=   t, t~
model= smdiag
model.options=ewchoose
order= gs, 0, 2
zero=me
one=gs,e
regularisation_scheme=dred
helicities=
qgraf.options=onshell,notadpole,nosnail
qgraf.verbatim= true=iprop[Z, 0, 0];\n\
                true=iprop[H, 0, 0];
qgraf.verbatim.lo=
qgraf.verbatim.nlo=
polvec=numerical
diagsum=True
reduction_programs=ninja,golem95,samurai
extensions=shared
debug=nlo
select.lo=
select.nlo=
filter.lo=
filter.nlo=
filter.module=
renorm_beta=True
renorm_mqwf=True
renorm_decoupling=True
renorm_mqse=True
renorm_logs=True
renorm_gamma5=True
reduction_interoperation=-1
reduction_interoperation_rescue=-1
samurai_scalar=2
nlo_prefactors=0
PSP_check=True
PSP_rescue=True
PSP_verbosity=False
PSP_chk_th1=8
PSP_chk_th2=3
PSP_chk_th3=5
PSP_chk_kfactor=10000
PSP_chk_li1=16
```

```
45 PSP_chk_li2=7
46 PSP_chk_li3=6
47 PSP_chk_li4=19
48 PSP_chk_method=Automatic
49 reference-vectors=
50 abbrev.limit=0
51 templates=
52 qgraf.bin=qgraf
53 form.bin=form
54 form.threads=2
55 form.tempdir=/tmp
56 haggies.bin=
57 fc.bin=/usr/bin/gfortran
58 python.bin=python
59 ninja.fcflags=
60 ninja.ldflags=
61 samurai.fcflags=
62 samurai.ldflags=
63 golem95.fcflags=
64 golem95.ldflags=
65 r2=explicit
66 symmetries=family,generation
67 crossings=
```

The comments to the file `eett.in` are as follows.

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

2 The item `process_path` specifies the directory to which all generated files and directories are written. If the subdirectory with the name specified under `process_path` does not exist, GoSam will create it. Specification of a process path is mandatory.

3–4 The items `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 corresponding model file.
Specifying `in` and `out` particles is mandatory.

5 The option `model` specifies which model files should be used in order to generate and evaluate the diagrams. How to import models in `UFO` or `LanHep` format is described in section

3.5. The default for this field is smdiag, i.e. the built-in Standard Model file with a diagonal CKM matrix.

6 The option model.options can be used to pass options which are specific to a certain model. The default is ewchoose, which means that the electro-weak scheme is selected automatically according to the given input parameters. Note that this option also allows to set values of masses, widths etc. For example, model.options = mB=4.19,w*=0 defines massive b-quarks and sets all widths to zero.

7 The item order is a comma separated list with three entries. The first entry specifies a symbol that denotes a coupling constant. With the Standard Model files sm*, the possibilities are 'gs' or (equivalently) 'QCD' for the strong coupling constant g_s and 'e' or 'gw' or 'QED' for the electroweak coupling (the latter three are equivalent in what concerns the counting of orders in the electroweak coupling). The keywords 'QCD' and 'QED' also can be used to calculate QCD resp. electroweak corrections to BSM processes.
The second number is the power of the chosen coupling constant for the tree-level diagrams and the third number 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 values of order are translated into a vsum constraint in the file qgraf.dat.
This field is mandatory.

8–9 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.

10 The option regularisation_scheme allows to choose the dimensional regularisation scheme, in our example dred for dimensional reduction, which is the default. cdr for "conventional dimensional regularisation" is also possible.

11 `helicites`: a comma separated list of helicities to be calculated. An empty list means that all possible helicities should be generated. The characters correspond to particles 1, 2, ... from left to right.
Example: $e^+e^- \to \gamma\gamma$:
Only three helicities are required; the other ones are either zero or can be obtained by symmetry transformations. This corresponds to
`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` .

12 `qgraf.options=onshell,notadpole,nosnail`: a list of options which is passed to `QGraf` via the 'options' line. Possible values (as of QGraf 3.1.4) are the following keywords: `onepi, onshell, nosigma, nosnail, notadpole, floop, topol`. In our example, it means that external lines are on-shell, i.e. do not contain selfenergy corrections, and that tadpole and snail diagrams are discarded. This is also the default if `qgraf.options` is not specified. We refer to the `QGraf` documentation for more details.

13-16 The value of the option `qgraf.verbatim` is passed verbatim to the file `qgraf.dat`. In our example we 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 we refer to the documentation of the corresponding model file.
These options can be omitted.

17 `polvec`: by default (`polvec=numerical`), numerical polarisation vectors are used for the massless gauge bosons, rather than producing separate code for each helicity (see section 3.3.3). To switch off the use of numerical polarisation vectors, use `polvec=explicit`.

18 `diagsum`: if `True`, one-loop diagrams sharing some propagators are combined before the algebraic reduction. The default is `diagsum = True`.

19 The option `reduction_programs` allows to choose the amplitude reduction method. If several choices are given, the code is produced such that the reduction methods can be switched at runtime. The default is `ninja,golem95`.

20 `extensions`: this option contains a list of useful extensions to the core of the program, which operate at the code generation stage. The currently available extensions are

- `autotools`: use autotools to generate Makefiles
- `shared`: create shared libraries (i.e. dynamically linkable code rather than static libraries). This extension is enabled by default when using the `autotools` extension.
- `f77`: in combination with the BLHA interface it generates a file `olp_module.f90` linkable with Fortran77.
- `noformopt`: disables diagram optimization using `FORM`
- `gaugecheck`: modifies the massless gauge boson wave functions to allow for a check of gauge invariance for processes involving gluons or photons.
- `customspin2prop` allows to replace the propagator of spin-2 particles with a custom function (we refer to subsection 3.5.3 for details).

In our example `shared` tells the program to build dynamic rather than static libraries.

21 `debug`: can take the values `lo`, `nlo`, `all`. It sets the level of information printed to the file `matrix/debug.xml` when running the test program.

22 `select.lo`: can be used to select/discard diagrams by their diagram numbers. It can contain a list of integer numbers, indicating leading order diagrams to be selected. If no list is given, all diagrams are selected. Otherwise, all diagrams whose numbers are not in the list will be discarded. The list may also contain ranges, with increments different from one, e.g. `select.lo=1,2,5:10:3, 50:53` is equivalent to `select.lo=1,2,5,8,50,51,52,53`, i.e. the 3 in `5:10:3` is the increment.

23 `select.nlo`: analogous to `select.lo`, for the one-loop diagrams.

24 `filter.lo`: a python function which provides a filter for tree diagrams. Example:
`filter.lo=lambda d:  d.iprop(Z) == 1 and d.vertices(Z, U, Ubar) == 0` filters out diagrams containing exactly one Z propagator and no $Zu\bar{u}$ couplings.

25 `filter.nlo`: analogous to `filter.lo`, for the one-loop diagrams. For details we refer to subsection 6.3.

26 `filter.module`: a python file of predefined functions which can be used as filters.

27 `renorm_beta`: activates or disables beta function renormalisation. The default is `True`.

28 `renorm_mqwf`: activates or disables UV countertems coming from external massive quarks. The default is `True`.

29 `renorm_decoupling`: activates or disables UV counterterms coming from massive quark loops. The default is `True`.

30 `renorm_mqse`: activates or disables the UV counterterm coming from the massive quark propagators. The default is `True`.

31 `renorm_logs`: activates or disables the logarithmic finite terms associated with the UV counterterms. The default is `True`.

32 `renorm_gamma5`: activates finite renormalisation for axial couplings in the 't Hooft Veltman scheme (CDR). Implemented for QCD only, works only with the built-in model files. The default is `True`.

33 `reduction_interoperation`: denotes the reductuion libraries to be used. Possible values are: ninja, samurai, golem95 (listing all of them simultaneously is possible). A value of -1 lets GoSam decide. See `common/config.f90` for details.

34 `reduction_interoperation_rescue`: specifies the reduction library to be used to rescue 'unstable points'. A value of -1 lets GoSam decide.

35 `samurai_scalar`: integer which specifies the library Samurai chooses for the basis integrals. 1: QCDLoop, 2: OneLOop, 3: `golem95C`. The default is 2.

36 `nlo_prefactors`: can take the integer values 0,1,2, which 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 result includes all prefactors (see also section A).

Note, however, that the factor of $1/\Gamma(1-\epsilon)$ is not included in any of the cases. Please note also that `nlo_prefactors=0` is enforced in `test.f90` in order to recognize rational numbers for the pole coefficients. In the OLP interface mode (BLHA/BLHA2), the default is `nlo_prefactors=2`.

37 `PSP_check`: allows to switch the stability test of the full amplitude for each phase space point on or off. If `PSP_check` is set to `False`, the following flags concerning `PSP_rescue` and the various thresholds for the rescue system have no effect. Details about the stability tests are given in the GoSam-2.0 paper. Please note that this test only works for QCD with the built-in model files. The default is `PSP_check= True`.

38 `PSP_rescue`: activates the phase space point rescue system based on the estimated accuracy. The accuracy is estimated using information on the single pole accuracy and information from the rotation test. The default is `PSP_rescue= True`.

39 `PSP_verbosity`: sets the verbosity of the `PSP_check`. `verbosity = False` means no output, `verbosity = True` means that bad points are written to a file
`gs_badpts.log`. The default is `verbosity = False`.

40 `PSP_chk_th1`: an integer indicating the number of desired accurate digits of the single pole coefficient. For poles coefficients more precise than this threshold the finite part is not checked separately. Note that this works only for QCD, with the built-in model files. The default is 8.

41 `PSP_chk_th2`: threshold (number of accurate digits) to declare a phase space point as *bad point*, based on the precision of the pole coefficient. Points with precision less than this threshold are directly reprocessed with the rescue system (if available), or declared as unstable. According to the verbosity level set, such points are written to a file and not used when the code is interfaced to an external Monte Carlo program using the new BLHA2 standards. The default is 3.

42 `PSP_chk_th3`: threshold (number of accurate digits) to declare a phase space point as *bad point*, based on the precision of the finite part estimated with a *rotation*. According to the verbosity level set, such points are written to a file and not used when the code is interfaced to an external Monte Carlo program using the new BLHA2 standards. The default is 5.

43 `PSP_chk_kfactor`: threshold on the K-factor to declare a phase space point as *bad point*. According to the verbosity level set, such points are written to a file and not used when the code is interfaced to an external Monte Carlo program using the new BLHA2 standards. The default is 1000.

44 `PSP_chk_li1`: sets the same variable in config.f90. For loop-

induced processes, it is used instead of `PSP_chk_th1`. It is the precision of the pole part (which should be zero) in comparison to the finite part. If the pole part is at least `PSP_chk_li1` orders smaller than the finite part, the point is accepted without any further check. The default is 16. If Samurai is used as default reduction program, it should be reduced to 8. It works normally only for QCD and with built-in model files.

45 `PSP_chk_li2`: sets the same variable in config.f90. For loop-induced processes, it is used instead of `PSP_chk_th2`. It is the threshold to declare a phase space point as bad point, based on the precision of the pole in comparison to the finite part. Points with precision less than this threshold are directly reprocessed with the rescue system (if available), or declared as unstable. According to the verbosity level set, such points are written to a file and not used when the code is interfaced to an external Monte Carlo using the new BLHA standards. If Samurai is used as default reduction program, it should be be reduced to 6. The default is 7.

46 `PSP_chk_li3`: sets the same variable in config.f90. For loop-induced processes, it is used instead of `PSP_chk_th3`. It is the threshold to declare a PSP as bad point, based on the precision of the finite part estimated with a rotation test. Points with precision less than this threshold are reprocessed with the rescue system (if available) or declared as unstable. According to the verbosity level set, such points are written to a file and not used when the code is interfaced to an external Monte Carlo using the new BLHA standards. The default value is 6.

47 `PSP_chk_li4` Sets the same variable in config.f90. Similar to `PSP_chk_li2,` but for the rescue system (by default Golem95). It is the threshold to declare a PSP as bad point in the rescue system, based on the precision of the pole part in comparison to the finite part. According to the verbosity level set, such points are written to a file and not used when the code is interfaced to an external Monte Carlo using the new BLHA standards. The default value is 19.

48 `PSP_chk_method`: this option can be used to overwrite the automatic precision test method enabled with `PSP_check=True`. Except for some BSM scenarios, the user does not need to change this. Possible options are

- `Automatic` – chooses automatically a suitable phase-space point test
- `PoleRotation` – checks first the pole and then rotates if necessary.
- `Rotation` – forces a rotation check on every phase space

point (slow).

- `LoopInduced` – checks that the pole part is zero and rotates if necessary. Needed e.g. for interference between BSM Born and SM loop-induced virtual.

49 `reference-vectors`: comma separated list of reference vectors for massive fermions and vector bosons. If no reference vectors are assigned here, the program picks the reference vectors 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 k_2 as reference momentum for the polarisation vector. The massive top quark (particle 3) uses the light-cone projection l_4 of the W-boson as reference direction for its own momentum splitting. Similarly, the momentum of the W-boson is split into a direction l_4 and one along l_3.

50 `abbrev.limit`: maximum number of instructions per subroutine when calculating abbreviations. The default is 0, which means that no maximum is set.

51 `templates`: Path pointing to the directory containing the template files for the process. If not set, the program uses the directory ⟨ gosam_path⟩/templates. The directory must contain a file called `template.xml`.

52 `qgraf.bin`: path to the `QGraf` executable. The default path will be set by the installation script.

53 `form.bin`: path to the `FORM` executable. The default path will be set by the installation script.

54 `form.threads`: the number of `FORM` threads when using `tform`, the parallel version of `FORM`. The default is 2.

55 `form.tempdir`: the temporary directory where `FORM` can store (large) intermediate files. the default is `/tmp`.

56 `haggies.bin`: path to the `haggies` executable. The default path will be set by the installation script.

57 `fc.bin`: path to the `Fortran` compiler. The default path will be set by the installation script.

58 `python.bin`: path to the `python` executable. The default path will be set by the installation script.

59 `ninja.fcflags`: compiler flags to compile with Ninja. The default will be set by the installation script.

60 `ninja.ldflags`: LDFLAGS required to link the Ninja library. The default will be set by the installation script.

61 `samurai.fcflags`: compiler flags to compile with Samurai. The default will be set by the installation script.

62 `samurai.ldflags`: LDFLAGS required to link the Samurai library. The default will be set by the installation script.

63 `golem95.fcflags`: compiler flags to compile with `golem95C`. The default will be set by the installation script.

64 `golem95.ldflags`: LDFLAGS required to link the `golem95C` library. The default will be set by the installation script.

65 `r2`: treatment of the rational part R_2. The possibilities are

- `implicit`: μ^2 terms are kept in the numerator and reduced at runtime
- `explicit`: μ^2 terms are reduced analytically
- `off`: all μ^2 terms are set to zero.

The default is `r2=explicit`.

66 `symmetries`: this information is used when the list of helicity configurations is generated. An empty list means that all helicity configurations will be generated, even if some of them could be mapped onto each other. Possible values are:

- `flavour`: assumes that no flavour changing interactions are present. When calculating the list of helicities, fermions with PDG codess 1-6 are assumed not to mix.
- `family`: flavour changing only within families. When calculating the list of helicities, fermion lines with PDG codes 1-6 are assumed to mix only within families, i.e. a quark line connecting an up with a down quark would be considered, while up-bottom would be discarded.
- `lepton`: means for leptons what 'flavour' means for quarks
- `generation`: means for leptons what 'family' means for quarks
- $\langle n\rangle = \langle h\rangle$: restriction of particle helicities, e.g. `1=-, 2=+` specifies helicities of particles 1 and 2
- %$\langle n\rangle$ = $\langle h\rangle$ restriction by PDG code, e.g. %23=+- specifies the helicity of all Z-bosons to be '+' and '-' only (no '0' polarisation).
  %$\langle n\rangle$ refers to both $+n$ and $-n$
  %+$\langle n\rangle$ refers to $+n$ only, %-$\langle n\rangle$ refers to $-n$ only

67 `crossings`: 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
```

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

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

3.2 Process directory structure

After running `gosam.py` 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/` Contains `Fortran` files which are common to all helicity amplitudes and to the constructed matrix element code. The file `config.f90` contains some global settings, the file `model.f90` contains the definitions and settings for the model parameters. This directory is always compiled first.

`doc/` Contains all files (apart from `pyxotree.tex` and `pyxovirt.tex`) which are necessary for creating `doc/process.pdf`, which lists all Feynman diagrams of this process, together with colour and helicity information.

`helicity[i]` These directories contain all files for a specific helicity amplitude (labelled by `i`). The labeling of the helicities can be found in `doc/process.pdf`. Note that the code will automatically map equivalent helicity configurations onto one single helicity and perform the corresponding book-keeping.

Before invoking `make source`, this directory only contains the makefiles. After the full code generation, for each diagram three classes of files are created: The basic algebraic expressions for the individual one-loop diagrams are contained in the files `d*h*l1.txt` in an optimized format. The files `d*h*l1.prc` contain the expressions of the numerators as polynomials in the loop momentum. The corresponding `Fortran` files are `d*h*l1.f90` and `abbrevd*h*.f90`, where the latter contains the abbreviations.

Files generated with the `derive` option are named `d*h*l1d.*`, while the input files for NINJA are named `d*h*l1*.*`. In more detail, the three categories of files are named as follows:

```
# Diagrams:
```

```
d*h*l1.prc
d*h*l1.txt
d*h*l1.f90
abbrevd*h*.f90

# Derive:
d*h*l1d.hh
d*h*l1d.txt
d*h*l1d.f90

# Ninja:
d*h*l1.hh
d*h*l12.txt
d*h*l13.txt
d*h*l14.txt
d*h*l1mu2.txt
d*h*l121.f90
d*h*l131.f90
d*h*l132.f90
```

`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.hh`	Contains model specific definitions needed by the `FORM` code which is generating the symbolic expressions for the amplitude. The original files from the `model/` directory of GoSam. are renamed, e.g. `sm` → `model`, `sm.hh` → `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`.
`Makefile.conf`	This files contains the settings which are global for all helicity configurations, like e.g. the paths to the reduction libraries, compiler options, etc.
`Makefile` `Makefile.source`	`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 available 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.
```

3.3 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. By using the make command line option `-j` *JOBS* one can parallelize this.

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 individual helicity amplitudes and diagrams for the set of phase space points used in `test.f90`.

3.3.1 Producing optimised code with FORM version 4

The constant, i.e. q- and μ^2-independent parts of the numerators of the one-loop diagrams are factored out from the numerators and computed as abbreviations.

While in version 1.0 of GoSam the `Fortran` code for the amplitudes was written using `haggies` [14], we now largely use the features provided by `FORM` version 4.x [13] to produce optimized code. This leads to more compact code and a speed-up in amplitude evaluation of about a factor of ten. The option to use `haggies` is still available by setting the extension `noformopt`.

In the case where `haggies` instead of `FORM` version 4 is used to produce abbreviations, i.e. if `extensions=noformopt` is used, please note the following: 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
↪ ${GOSAMPATH}/haggies/haggies.jar
```

which assigns 3 GB of memory to Java.

In some cases the list of abbreviations is too big to be compiled into one subroutine. One can restrict the number of instructions for haggies that go into a single subroutine by setting abbrev.limit to a positive number in the process card.

3.3.2 Grouping/summing of diagrams which share common subdiagrams

Already in the first release of GoSam, the diagrams were analyzed according to their kinematic matrix S_{ij} and grouped together before reduction. These lead to an important gain in efficiency, both when using integrand reduction methods, as well as classical tensor reduction techniques. Details about the way diagrams are grouped can be found in [1]. This feature is still present when Samurai or golem95C are used to reduce the amplitudes.

In release 2.0 an option called diagsum combines diagrams which differ only by a subdiagram into one “meta-diagram” to be processed as an entity. This allows to further reduce the number of calls to the reduction program and therefore to increase the computational speed.

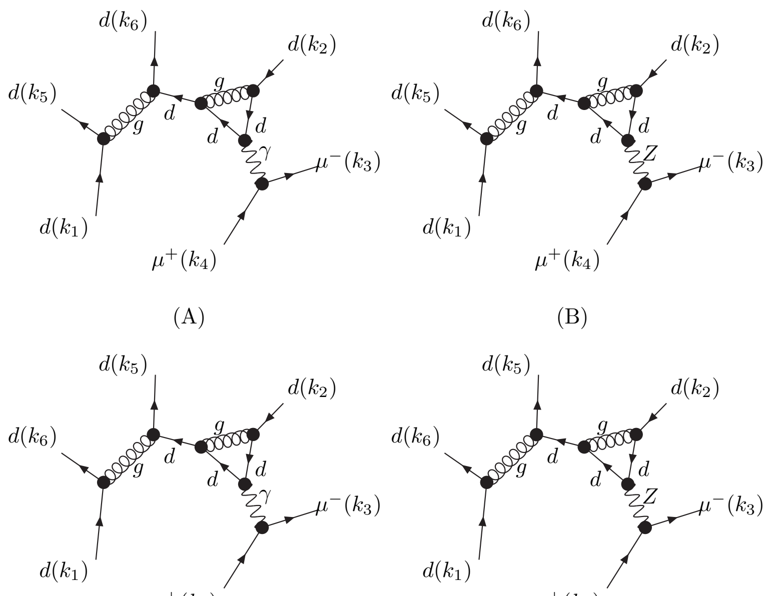

Figure 3.1: Example of diagrams sharing a common tree part, which are summed when the diagsum option is set to diagsum=true.

When the option diagsum is active, diagrams which differ only by a propagator external to the loop, as is the case e.g. for the $Z/\gamma^\star$ propagator in QCD corrections to the production of Z+jets, are summed together before being processed by FORM. Similarly, diagrams which differ only by an external tree part, but which share exactly the same set of loop propagators, are summed together prior the algebraic manipulation. An example is shown Figure 3.1.

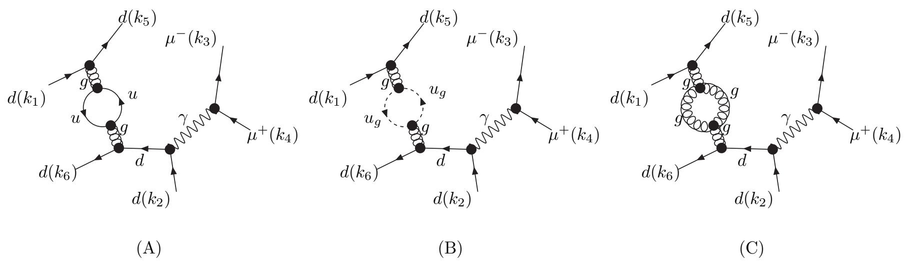

Figure 3.2: Example of diagrams sharing a common loop propagator, but with different particle content in the loop, which are summed when the `diagsum` option is set to `diagsum=true`.

Finally, diagrams which share the same set of propagators, but have different particles circulating in the loop, as shown in Figure 3.2, are also summed into one "meta-diagram". The default setting for this option is `diagsum=true`.

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`. However, the latter can only be used in combination with the extension `noformopt`.

3.3.3 Numerical polarisation vectors

The use of numerical polarisation vectors for massless gauge bosons (gluons, photons) is activated by default. This means that the various helicity configurations for the massless bosons will be evaluated numerically, based on the same code, rather than producing separate code for each helicity configuration. In order to switch off this default setting, for example if the user would like to optimize the choice of reference vectors for each helicity configuration, the option `polvec=explicit` should be given in the process card `process.in`. In this case, GoSam will choose explicit reference vectors automatically. If the user wants to specify his/her preferred reference vectors, this can be done using the option `reference-vectors=...` in the process card.

3.3.4 The extension `derive`

The `derive` feature generates code to access the tensor coefficients of each diagram or group of diagrams individually. While it has been among the possible keywords for the `extensions` option in GoSam-1.0 already, it now has been promoted to be used by default in the context of tensorial reconstruction [15]. It improves both the speed and the precision of tensorial reconstruction and makes connection to other reduction methods.

The idea behind it is to compute the numerator $\mathcal{N}(\hat{q})$ from a Taylor series

$$\mathcal{N}(\hat{q}) = \mathcal{N}(0) + \hat{q}^\mu \frac{\partial}{\partial \hat{q}_\mu}\mathcal{N}(\hat{q})|_{q=0} + \frac{1}{2!}\hat{q}^\mu\hat{q}^\nu \frac{\partial}{\partial \hat{q}_\mu}\frac{\partial}{\partial \hat{q}_\nu}\mathcal{N}(\hat{q})|_{q=0} + \ldots \tag{3.1}$$

In this form one can read off a one-to-one correspondence between derivatives at $\hat{q} = 0$ and the coefficients of the tensor integrals.

At a technical level, the files `helicity*/d*h*l1d.f90` contain the routines `derivative(`μ^2`, [`i_1`], [`i_2`],...)` and `reconstruct_d*(coeffs)`, where the latter is only generated in conjunction with the extension `golem95`, and `coeffs` is a type which comprises all coefficients of a diagram of a certain rank. The number of optional indices i_1, i_2, ... determine which derivative should be returned. The subroutine `reconstruct_d*` also takes into account the proper symmetrisation.

3.3.5 Customization

Runtime parameters. Many settings can be changed without recompiling the code, by creating and modifying a 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 defined parameter.
- 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`. With the built-in Standard Model file (`sm`) one can re-set, for example, the value for the Higgs mass by `mH = 125.5`. All model constants that have not been specified as zero or one can be set in this way. In addition there are some model independent parameters which can be found in the file `common/config.f90`.

Compile time parameters. Other configuration options can be found in the file `common/config.f90`. Examples of options contained in `config.f90` are

`ki`	the floating point kind used throughout the calculation, the default is double precision.
`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 ε multiplying poles are taken into account.
`include_eps2_terms`	controls if terms of order ε^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 [16], which reduces the error accumulated in presence of large cancellations.
`tens_rec_by_derivatives`	controls whether the tensorial reconstruction method is used.

3.4 Drawing the Feynman diagrams

In order to print out the diagrams the makefile contains the target `doc` which produces the file `process.pdf`. We use LaTeX plus the package axodraw [17] to create the graphical representation.

The layout of the diagrams is determined by the algorithm used in feynMF [18], modelling the propagators by springs. The implemented algorithm works in two steps: first, the topology is constructed 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. 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

$$L(v_1, \ldots, v_n; e_1, \ldots, e_N) = \frac{1}{4}\sum_{i,j=1}^{n} t_{ij}\left(v_i - v_j\right)^2 + \frac{1}{2}\sum_{i=1}^{n}\sum_{k=1}^{N}\lambda_{ik}\left(v_i - e_k\right)^2 \quad (3.2)$$

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 equations, which can easily be solved.

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

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

$$M_{rc} = \begin{cases} \left(\sum_{i\neq r} t_{ri}\right) + \left(\sum_{k=1}^{N}\lambda_{rk}\right), & r = c \\ -t_{rc}, & \text{otherwise} \end{cases} \tag{3.3}$$

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

3.5 Import of model files

The GoSam-2.0 package comes with the built-in model files `sm`, `smdiag`, `smdiag_mad`, `smehc`, `smdiagehc`, `sm_complex`, `smdiag_complex`, where the latter two are needed in the case of complex masses and couplings, see section 5.2. The model files `smdiag_mad` contain some `MadGraph5` specific settings, while the model files `smehc` and `smdiagehc` contain the effective Higgs-gluon couplings.

Other models can be imported most easily in the `UFO` (Universal FeynRules Output) [19] format. The model import in the `UFO` format can be used in the standalone as well as the OLP mode of GoSam, where both the BLHA1 and BLHA2 standards are supported for the syntax of the model import.

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

3.5.1 Import from FeynRules

A model description in the UFO [19] 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. For example, if the `Python` model files for the MSSM are in the directory `$HOME/models/MSSM_UFO`, the process card must contain the line

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

3.5.2 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[2]. 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 ',
   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`

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

3.5.3 Propagators for spin-2 particles

The propagator for massive spin-2 particles can be split into two parts

$$i\Delta_{\mu\nu,\rho\sigma}(k,m_{\vec{n}}) = \underbrace{\frac{i}{k^2 - m_{\vec{n}}^2 + i\epsilon}}_{D(k^2,m_{\vec{n}})} B_{\mu\nu,\rho\sigma}(k,m_{\vec{n}}) \; , \tag{3.4}$$

where $B_{\mu\nu,\rho\sigma}$ carries the Lorentz structure

$$\begin{aligned} B_{\mu\nu,\rho\sigma}(k,m) = & \left(\eta_{\mu\rho} - \frac{k_\mu k_\rho}{m^2}\right)\left(\eta_{\nu\sigma} - \frac{k_\nu k_\sigma}{m^2}\right) \\ & + \left(\eta_{\mu\sigma} - \frac{k_\mu k_\sigma}{m^2}\right)\left(\eta_{\nu\rho} - \frac{k_\nu k_\rho}{m^2}\right) \\ & - \frac{2}{3}\left(\eta_{\mu\nu} - \frac{k_\mu k_\nu}{m^2}\right)\left(\eta_{\rho\sigma} - \frac{k_\rho k_\sigma}{m^2}\right) \; . \end{aligned} \tag{3.5}$$

If all particles attached to the propagator are on-shell, the mass dependent terms in $B_{\mu\nu,\rho\sigma}(k,m)$ drop out. Further, if the on-shell condition is not always fulfilled, it turned out in phenomenological applications that the impact of the mass dependent terms is numerically negligible [20, 21] and therefore we did not include them in our implementation in order to avoid an enormous proliferation of terms. In this case the summation over the graviton states in $D(s,m_{\vec{n}})$, leading to

$$D(s) = \sum_{\vec{n}} \frac{i}{s - m_{\vec{n}}^2 + i\epsilon} \; , \tag{3.6}$$

can be performed independently from the $B_{\mu\nu,\rho\sigma}$ part carrying the Lorentz structure. Further, in models with large extra dimensions (LED), we can use the assumption that the widths of the KK modes are negligible, as the dominant effects come from the almost on-shell production of KK modes, and that the discrete spectrum of the KK modes can be approximated by an integral over a mass density, as the KK modes are very contiguous. The density as a function of the mass $m_{\vec{n}}$ is given by

$$\rho(m_{\vec{n}}) = \frac{R^\delta m_{\vec{n}}^{\delta-2}}{(4\pi)^{\delta/2}\Gamma(\delta/2)} \; , \tag{3.7}$$

where δ is the number of extra dimensions, leading to [22]

$$D(s) \to \int_0^{M_S} d\,m_{\vec{n}}^2 \, \frac{i\,\rho(m_{\vec{n}})}{s - m_{\vec{n}}^2 + i\epsilon} = \begin{cases} \frac{s^{\delta/2-1}}{2M_s^{\delta+2}G_N}\left(\pi + 2i\,I(\frac{M_S}{\sqrt{s}})\right) & \text{for } s > 0 \\ \frac{(-s)^{\delta/2-1}}{2M_s^{\delta+2}G_N}(-2i)\,I_E(\frac{M_S}{\sqrt{-s}}) & \text{for } s < 0 \end{cases} \tag{3.8}$$

with

$$I(x) = \begin{cases} -\sum_{k=1}^{\delta/2-1}\frac{1}{2k}x^{2k} - \frac{1}{2}\log(x^2-1) & \text{if } \delta \text{ even} \\ -\sum_{k=1}^{(\delta-1)/2}\frac{1}{2k-1}x^{2k-1} + \frac{1}{2}\log\left(\frac{x+1}{x-1}\right) & \text{if } \delta \text{ odd} \end{cases} \tag{3.9}$$

and

$$I_E(x) = \begin{cases} (-1)^{\delta/2+1}\left(\sum_{k=1}^{\delta/2-1}\frac{(-1)^k}{2k}x^{2k} + \frac{1}{2}\log(x^2+1)\right) & \text{if } \delta \text{ even} \\ (-1)^{(\delta-1)/2}\left(\sum_{k=1}^{(\delta-1)/2}\frac{(-1)^k}{2k-1}x^{2k-1} + \frac{1}{2}\tan^{-1}(x)\right) & \text{if } \delta \text{ odd}\,. \end{cases} \tag{3.10}$$

The UV cutoff M_S is introduced as the effective theory approach loses its validity beyond the scale M_S.

GoSam supports spin-2 propagators with the `customspin2propagator` extension which needs to be enabled in the process card by `extensions=customspin2propagator`.

The extension works only if the model is imported from an `UFO` file. The latter can be adjusted to the needs of the particular model the user would like to consider by editing the file `custompropagator.f90` in the subdirectory `common`. In order to generate the latter file, the spin-2 particle which should get a customized propagator needs to have a separate attribute 'CustomSpin2Prop' in the `UFO` file with a non-vanishing value:
Excerpt of `LED_UFO/particle.py` with a customized propagator:

```
Gr = Particle(pdg_code = 9000006,
              name = 'Gr',
              # ...
              CustomSpin2Prop = 1
             )
```

Then GoSam generates a the file `common/custompropagator.f90` where the user needs to adapt the `customSpin2Prop` subroutine. Beside the squared momentum, the `customSpin2Prop` subroutine also receives the mass of the corresponding spin-2 particle as an argument, which can be used to distinguish between multiple spin-2 particles if necessary. The tensorial part of the spin-2 propagator, $B_{\mu\nu,\rho\sigma}(k, m_{\vec{n}})$, is treated separately and should not be modified. If the user would like to modify it, we refer to the documentation in section 6.3 of `src/form/lorentz.pdf` in the GoSam tarball.

4 Stability tests and rescue system

GoSam contains various options to assess in real time, for each phase space point, the level of precision of the corresponding one-loop matrix element. Whenever a phase space point is found in which the quality of the result falls below a certain threshold, either the point is discarded or the evaluation of the amplitude is repeated by means of a safer, albeit less efficient procedure. This procedure is traditionally called “rescue system”.

Apart from improvements in the stability of the reduction itself, which are provided by the new versions of Samurai and `golem95C`, and in particular by the new reduction algorithm Ninja, the new version of GoSam also has a more refined rescue system as compared to version 1.0.

A first commonly used approach relies on the comparison between the numerical values of the infrared pole coefficients computed by the one-loop program with their known analytic results dictated by the universal behaviour of the infrared singularities [23]. We refer to this as the *pole test*.

The main advantages of this method are its broad applicability and the fact that it requires a negligible additional computation time. However, since not all integrals which appear in the reconstruction of the amplitude give a contribution to the double and single poles, this method often provides an overestimate of the precision, which might result in keeping phase space points whose finite part is less precise than what is predicted by the pole test.

To target directly the precision of the finite part, various possibilities exist. Using the symmetry properties of scattering amplitudes under scaling of all physical scales, or alternatively the invariance under rotation of the momenta, we can build pairs of points that should provide identical results, both for the finite parts and for the poles, and use the difference between them as an estimator of the precision.

The *scaling test* [24], is based on the properties of scaling of scattering amplitudes when all physical scales (momenta, renormalization scale, masses) are rescaled by a common multiplicative factor x. As shown in [24], this method provides a very good correlation between the estimated precision and the actual precision of the finite parts.

The *rotation test* [4] exploits the invariance of the scattering amplitudes under an azimuthal rotation about the beam axis, namely the direction of the initial colliding particles. Whenever the initial particles are not directed along the beam axis, one can perform a rotation of all particles by an arbitrary angle in the space of momenta. A validation of this technique, and the corresponding correlation plots, has been presented in [4].

While the *scaling* and the *rotation test* provide a more reliable estimate of the precision of the finite parts that enter in the phase space integration, their downside is that they require two evaluations of the same matrix element, therefore leading to a doubling in the computational time.

For the precision analysis contained in GoSam, and to set the trigger for the rescue system, we decided to employ a hybrid method, that takes advantage of the computational speed of the *pole test*, combined with the higher reliability of the *rotation test*. This hybrid method requires setting three different thresholds. After computing the matrix elements, GoSam checks the precision δ_{pole} of the single pole with the *pole test*. Comparing the single pole $\mathcal{S}_{IR}$ that can be obtained from the general structure of infrared singularities and the one provided by GoSam, which we label $\mathcal{S}$, we define

$$\delta_{pole} = \left| \frac{\mathcal{S}_{IR} - \mathcal{S}}{\mathcal{S}_{IR}} \right| . \tag{4.1}$$

The corresponding estimate of the number of correct digits in the result is provided by $P_{pole} = -\log_{10}(\delta_{pole})$. This step does not require any increase in computational time. The value of P_{pole} is then compared with two thresholds P_{high} and P_{low}.

If $P_{pole} > P_{high}$ the point is automatically accepted. Given the high quality of the computed pole, the finite part is very unlikely to be so poor that the point should be discarded.

If $P_{pole} < P_{low}$ the point is automatically discarded, or sent to the rescue system. If already the pole has a low precision, we can expect the finite part to be of the same level or worse.

In the intermediate region where $P_{high} > P_{pole} > P_{low}$, it is more difficult to determine the quality of the result solely based on the pole coefficients. Only in this case the point is recalculated using the *rotation test*, which requires additional computational time.

If we call the finite part of the amplitudes evaluated before and after the rotation $\mathcal{A}$ and $\mathcal{A}_{rot}$ respectively, we can define the error δ_{rot} estimated with the rotation as

$$\delta_{rot} = 2 \left| \frac{A_{rot} - A}{A_{rot} + A} \right| . \tag{4.2}$$

and the corresponding estimate on the number of correct digits as $P_{rot} = -\log_{10}(\delta_{rot})$. P_{rot} provides a reliable estimate of the precision of the finite part [4], and can be compared with a threshold P_{set} to decide whether the point should be accepted or discarded.

The values of the three thresholds P_{high}, P_{low} and P_{set} can be chosen by the user, to adjust the selection mechanism to the fluctuations in precision which occur between different processes. In the input card, P_{high}, P_{low} and P_{set} correspond to `PSP_chk_th1`, `PSP_chk_th2` and `PSP_chk_th3`, respectively, see section 3. It is worth to notice that the *rotation test* can be bypassed simply by setting the initial thresholds $P_{high} = P_{low}$. In this case the selection is performed solely on the basis of the *pole test*.

5 Electroweak corrections

5.1 Electroweak scheme choice

When computing amplitudes within the Standard Model, there are different possibilities how to choose which electroweak parameters are considered as input parameters, and which are instead derived ones. Within GoSam different schemes can be chosen in several different ways, depending on whether the scheme might be changed after the generation of the code or not, by setting appropriately the flag `model.options`.

By default, when the flag is not set in the input card, GoSam generates a code which uses $\mathrm{m_W}$, $\mathrm{m_Z}$ and α as input parameters, allowing however to change this in the generated code, by setting the variable `ewchoice` in the configuration file `config.f90` to the desired value. The user can choose among 8 different possibilities, which are listed in Table 5.1. When the electric charge e is set algebraically to one, the schemes $6-8$ cannot be used.

The flag `model.options` in the input card allows also to directly set the values of the different parameters appearing in the model. If the values of exactly three electroweak parameters are specified, GoSam automatically takes them as input parameters. In that case, in order to be able to switch among different schemes after code generation, the variable `ewchoose` also must be added to the `model.options` flag.

ewchoice	input parameters	derived parameters
1	$\mathrm{G_F}$, $\mathrm{m_W}$, $\mathrm{m_Z}$	e, sw
2	α, $\mathrm{m_W}$, $\mathrm{m_Z}$	e, sw
3	α, sw, $\mathrm{m_Z}$	e, $\mathrm{m_W}$
4	α, sw, $\mathrm{G_F}$	e, $\mathrm{m_W}$
5	α, $\mathrm{G_F}$, $\mathrm{m_Z}$	e, $\mathrm{m_W}$, sw
6	e, $\mathrm{m_W}$, $\mathrm{m_Z}$	sw
7	e, sw, $\mathrm{m_Z}$	$\mathrm{m_W}$
8	e, sw, $\mathrm{G_F}$	$\mathrm{m_W}$, $\mathrm{m_Z}$

Table 5.1: Possible choices to select the electroweak scheme. To simplify the notation we write the sine of the Weinberg angle as sw. The lists of derived parameters contain only the parameters which are computed and used in the expressions for the amplitudes.

5.2 Support of complex masses

The integral libraries contained in the GoSam package as well as the GoSam code itself fully support complex masses. This refers to the introduction of finite widths for fermions as well as for W- and Z-bosons. A fully consistent treatment of complex W- and Z-boson masses requires the use of the complex mass scheme [25]. The boson masses are promoted to complex masses by

$$m_V^2 \to \mu_V^2 = m_V^2 - i m_V \Gamma_V , \quad V = W, Z \; . \tag{5.1}$$

In order to maintain gauge invariance this affects the definition of the Weinberg angle:

$$\cos^2 \theta_w = \frac{\mu_W^2}{\mu_Z^2} \; . \tag{5.2}$$

To make use of the complex mass scheme, we introduce two new model files, `sm_complex` and `smdiag_complex`, which contain the Standard Model with complex mass scheme, the first with a full CKM matrix, the latter with a diagonal CKM matrix. An example dealing with a complex top quark mass is given in the `examples/singletop` subdirectory of the GoSam distribution.

6 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`.

6.1 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 referred to the examples and to the QGraf documentation.

6.2 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`.

6.3 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× 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.

7 The Binoth Les Houches Accord Interface

The interface of GoSam with a Monte Carlo event generator program is based on the Binoth-Les Houches Accord (BLHA) standard interface. GoSam-2.0 supports both BLHA1 [26] and BLHA2 [27]. Certainly, a dedicated interface without using the BLHA is also possible.

7.1 Preparation of the order file

This step should be done by the Monte Carlo (MC) program. We give a generic example of an order file for the process $pp \to (Z \to e^+e^-)+$jet in both BLHA1 and BLHA2 standards in Figs. 7.1 and 7.2.

```
# OLP_order.lh
# created by MC Sherpowig-1.0
# Process:  p p -> e+ e- jet
Model SMdiag
CorrectionType QCD
IRregularisation DRED
AlphasPower 2
AlphaPower 1
MatrixElementSquareType CHsummed
OperationMode CouplingsStrippedOff
SubdivideSubprocess no
# Subprocesses
1 -1 -> 11 -11 21
1 21 -> 11 -11 1
2 -2 -> 11 -11 21
...
21 -2 -> 11 -11 -2

# Process specific GoSam settings
#@ symmetries family,generation
```

```
# vim:  syntax=olp
#@OLP GOSAM 2.0
#@IgnoreUnknown False
#@IgnoreCase False
#@SyntaxExtensions
CorrectionType QCD | OK
IRregularisation DRED | OK
AlphasPower 2 | OK
AlphaPower 1 | OK
1
MatrixElementSquareType CHsummed | OK
OperationMode CouplingsStrippedOff | OK
SubdivideSubprocess no | OK
1 -1 -> 11 -11 21 | 1 1
1 21 -> 11 -11 1 | 1 2
2 -2 -> 11 -11 21 | 1 3
...
21 -2 -> 11 -11 -2 | 1 13
```

Figure 7.1: Examples of order and contract files for Z+jet, with BLHA1 standards.

Remarks

- The order file can have any name and any extension. We use the extension `.lh` for order files and `.olc` for contract files.
- The options `WidthScheme, EWScheme` in the BLHA2 example are optional.
- The option `SubdivdeSubprocess` has the following effect on the code generation with GoSam: if set to `no`, GoSam generates one label per subprocess, if set to `yes` it generates one

```
# OLP_order.lh
# created by MC Sherpowig-2.0
InterfaceVersion BLHA2
CorrectionType QCD
IRregularisation DRED
WidthScheme ComplexMass
EWScheme alphaGF
AccuracyTarget 0.0001
DebugUnstable True

AlphasPower 1
AmplitudeType ccTree
1 -1 - > 11 -11 21
...
21 -2 - > 11 -11 -2
AmplitudeType scTree
1 -1 - > 11 -11 21
...
21 -2 - > 11 -11 -2
AmplitudeType Loop
1 -1 - > 11 -11 21
...
21 -2 - > 11 -11 -2

AlphasPower 2
AmplitudeType Tree
1 1 - > 11 -11 1 1
...
21 21 - > 11 -11 2 -2
```

```
# vim:  syntax=olp
#@OLP GOSAM 2.0
#@IgnoreUnknown False
#@IgnoreCase False
#@SyntaxExtensions
InterfaceVersion BLHA2 | OK
CorrectionType QCD | OK
IRregularisation DRED | OK
WidthScheme ComplexMass | OK
EWScheme alphaGF | OK
AccuracyTarget 0.0001 | OK
DebugUnstable True | OK

AlphasPower 1 | OK
AmplitudeType ccTree | OK
1 -1 - > 11 -11 21 | 1 131
...
21 2 - > 11 -11 2 | 1 70
AmplitudeType scTree | OK
1 -1 - > 11 -11 21 | 1 145
...
21 2 - > 11 -11 2 | 1 71
AmplitudeType Loop | OK
1 -1 - > 11 -11 21 | 1 137
...
21 2 - > 11 -11 2 | 1 63

AlphasPower 2 | OK
AmplitudeType Tree | OK
1 1 - > 11 -11 1 1 | 1 42
...
21 21 - > 11 -11 -2 2 | 1 106
```

Figure 7.2: Order and contract files for Z+jet with BLHA2 standards.

label per helicity subamplitude and therefore *many* labels per subprocess.

- GoSam specific settings can be put into commentary lines starting with the letter combination '`#@`'. This is not part of the BLHA standard. The line '`#@ symmetries` ...' restricts the helicity subamplitudes being generated to the ones relevant for this particular process, using the information that flavour changings only occur within the same quark families resp. lepton generations.

  Additionally, the `Extra` keyword of BLHA2 for OLP-specific settings is also supported.

7.2 Running GoSam

To run GoSam within the MC/OLP setup one can use the following command:

```
gosam.py --olp --mc=MCname
↪--config=<your-path-to>/gosam.conf order.lh
```

Remarks

- The extension `--olp` is mandatory whenever a BLHA order file is processed.
- The extension `--mc=MCname` is optional. By specifying the name of the Monte Carlo which is the intended partner program, GoSam can choose some settings simplifying the communication and linking. One can either specify `--mc=MCname` or `--mc=name/version`. Alternatively (and also optional), one can put this information into the order file:
  ```
  #@olp.mc.name mypreferredmc
  #@olp.mc.version 1.0.0
  ```
  The short option for `--mc` is `-M`.
- The extension `--config` is optional and points to a GoSam configuration file. The latter can be used to define GoSam specific settings, such as diagram filters, treatment of the rational parts, etc.

  If this option is left out GoSam searches for a configuration in one of the following locations:

  – GoSam installation directory,

  – user's home directory,

  – current working directory.

  Possible names for default configuration files are `gosam.in`, `gosam.conf` and `.gosam`. If such a file is not found, GoSam takes the default values for all unspecified settings. The short form is `-c`.
- The option `--destination=<dir>` allows to place the generated files into the directory `<dir>`. The short form is `-D<dir>`.
- One can specify the name of the contract file which should be written using the option `--output-file=<contractfile>` or simply `-o<contractfile>`.
- The option `--force` will overwrite an already existing contract file without any warning.

The contract file From the contract file one can see whether the order file has been processed successfully. If everything went smoothly it should look like the one in Fig. 7.1 resp. Fig. 7.2. All settings are either acknowledged by the word `OK` or, in case of a failure, by the word `error` followed by an error message.

The subprocesses receive an assignment to one or more labels per subprocess. In the line
`2 -2` $\rightarrow$ `11 -11 21` $|$ `1 3`
the suffix `| 1 3` states that this subprocess has been assigned to `1` single label which has the value `3`. Had we set `SubdivideSubprocess` (keyword in BLHA1) to `yes` this line might have looked like
`2 -2` $\rightarrow$ `11 11 21` $|$ `4 0 1 2 3`
meaning that the subamplitudes have been assigned to `4` labels (which is the first number after the bar) with the values `0` to `3`, each denoting an individual helicity subamplitude. These labels will enter the first argument of the routine `OLP_EvalSubProcess`. In order to retrieve the full amplitude the calling (MC) program should sum over the contributions from all labels. Alternatively, it is possible to sample the different channels by Monte Carlo techniques.

7.3 Producing the libraries containing the virtual amplitudes

The procedure depends on whether the `autotools` extension is enabled (by default for some Monte Carlo programs). If not, using instead the extension `shared` in the GoSam settings is highly recommended.

With the `autotools` extension, the following sequence of commands will generate and compile the virtual matrix element files:

```
./autogen.sh --prefix=$(pwd)
make install
```

Now one should find the following files[1] in a subdirectory `lib/` or `lib64/`:

- `libgolem_olp.a` for static linking,
- `libgolem_olp.so` for dynamic linking,
- `libgolem_olp.la` for linking with libtool.

Without the `autotools` extension, one only needs to call `make` without any parameters. If the `shared` extension is enabled, one gets a `libgosam_olp.so` file in the top folder.

The Monte-Carlo program can now be linked to these files or can use the dynamical library at runtime using the `dlopen()` and `dlsym()` system calls[2].

[1] Due to backwards compatibility, they are still named `libgolem_olp` instead of `libgosam_olp`.
[2] For more details we refer to the corresponding man pages.

The required compiler and linking flags can be generated by calling the `config.sh` script:

```
sh ./config.sh -cflags  # prints C/C++ flags
sh ./config.sh -fcflags # prints Fortran flags
sh ./config.sh -libs    # prints linking flags
```

Inside a makefile, one can use the following lines to extend existing build flags:

```
CFLAGS+=$(shell ./config.sh -cflags)
FCFLAGS+=$(shell ./config.sh -ldflags)
LDFLAGS+=$(shell ./config.sh -libs)
```

The path to `config.sh` needs of course to be adapted if the makefile is not in the same directory.

7.4 Calling the interface routines

For the default settings the call of the interface routines will be automatic, so the user does not have to care about the details described below.

We should note however that there are slight differences in naming (underscoring) and calling conventions (call by reference versus call by value) depending on the extensions in use. For `--mc=powhegbox` the extension `f77` is automatically included and therefore the underscoring works such that `gfortran` used as a Fortran 77 compiler would not complain. For all other Monte Carlo programs we follow the C/C++ conventions (see the file `olp.h`).

In the following, we will describe BLHA1 and BLHA2 conventions separately, even though large parts are identical for the two BLHA versions.

7.4.1 BLHA1

Initialization

The generated GoSam library is initialized with the call
`call OLP_Start("path/to/contract.olc",ierr)`
The variable `ierr` should be declared as an integer. If the contract file is not found, `ierr` is set to a negative value. A non-negative value indicates success.

Please note that calling `OLP_Start` is mandatory even if the contract file is not present or not read.

Importing external model files

If the contract file contains the option `ModelFile`, which should point to a SLHA file, the matrix element code tries to load the parameters from that file.

Setting options (optional)

Parameters can be passed by calling `OLP_option`.
`call OLP_Option("name=value",ierr)`

Note that the initialization of derived parameters only works correctly if the corresponding input parameters are set with `OLP_Option` *before* `OLP_Start` is called.

Example:

```
      call OLP_Option("mZ=91.234",ierr)
      call OLP_Option("mW=80.123",ierr)
!   at this point sin(theta_w) is not up to date.
       call OLP_Start("path/to/contract.olc",ierr)
!   now sin(theta_w) is set consistently
```

Some options can be changed at any time; it is instructive to look at the file `common/model.f90` which contains the available parameter names and their settings.

Computing the matrix element

In BLHA1, the routine which returns a value for the matrix element is `OLP_EvalSubProcess`:

```
      integer ilabel
      double precision moms(5*nlegs)
      double precision mu,params(1)
      double precision res(4)
      !...
      call OLP_EvalSubProcess(
     &           ilabel,moms,mu,params,res)
```

The first argument, `ilabel` is one of the labels from the contract file. The momenta are passed in the argument `moms`, which has the format

$$(/E_1, p_1^x, p_1^y, p_1^z, m_1, E_2, p_2^x, p_2^y, p_2^z, m_2, \ldots E_N, p_N^x, p_N^y, p_N^z, m_N/)$$

The momenta are expected to be given in physical (in-out) kinematics: $p_1 + p_2 = p_3 + \ldots + p_N$. The components are in units of GeV.

The argument `mu` is the renormalisation scale μ (not μ^2!) in GeV. The argument `params` is an array of which the first argument is $\alpha_s(\mu)$. Any further array entries are ignored within BLHA1[3].

The last argument is an array of length four which is filled by the subroutine, containing the result of the evaluation. The entries

[3] Passing more than one parameter is implemented by the `Parameters` option in the order file, which is not part of the BLHA1 standard.

have as a unit some power of GeV ($\text{GeV}^{(4-N)}$).

$$\begin{aligned}\mathcal{M}_B^\dagger \mathcal{M}_B &= \texttt{res(4)}\\ 2\text{Re}\left(\mathcal{M}_B^\dagger \mathcal{M}_V\right) &= \frac{(4\pi)^\varepsilon}{\Gamma(1-\varepsilon)}\left(\frac{\texttt{res(1)}}{\varepsilon^2}+\frac{\texttt{res(2)}}{\varepsilon}+\texttt{res(3)}\right)\end{aligned} \tag{7.1}$$

This means that the coefficients `amp(1:3)` contain an explicit factor of $\alpha_s(\mu)/(4\pi)$.

Finalize (optional)

There is also a routine `OLP_Finalize` which is only needed if the client code needs to call `OLP_Start` more than once, e.g.

```
do i=1,max_i
   write(line,'(A3,F6.3)') "mZ=", mZ(i)
   call OLP_Option(line,ierr)
   ! Need olp_start to update dependent parameters
   call OLP_Start(name,ierr)
   ! ...
   call OLP_Finalize()
enddo
```

7.4.2 BLHA2

! $\rightarrow$ Please note that with BLHA2 all light quark masses (u,d,s,c,b) are set to zero by default. To have massive light quarks, one needs to use the `MassiveParticles` parameter in the order file.

Initialization

The keyword `InterfaceVersion`, which can take the values `BLHA1` or `BLHA2`, should be placed on top of the order file. This way, if the OLP does not support one or the other, it can issue an error message and stop without proceeding further.

To start the run-time phase, the function
`OLP_Start(char* fname, int* ierr)` is the same as in `BLHA1`. A new function
`OLP_Info(char olp_name[15],char olp_version[15],char message[255])`
has been introduced which serves to keep track of the type and version of the OLP which has been used, and to encourage proper citation. The arguments are the name of the OLP, the version, and a string which contains information about the relevant publications, for example the bibtex identifier.

Importing external model files

The BLHA2 offers two alternative ways of model definition, denoted by “keyword model” respectively “`UFO` model” in the following.

Model definitions offer the possibility to define some global settings in the order file, which are intrinsic to the model (e.g. SM, MSSM), which is used. This is done using the required keyword `Model`. For example, `Model: smdiag` sets the CKM matrix to unity globally.

In the "keyword model" setup, the parameters that need to be set within a certain model are passed via PDG codes [28] and keywords with naming conventions as specified in Fig. 7.3 for the Standard Model. The numbers in parenthesis after `mass` and `width` denote the particle's PDG code.

keyword	parameter
`mass(5)`	b quark mass
`mass(6)`	top quark mass
`width(6)`	top quark width
`sw2`	$\sin^2\theta_w$
`vev`	SM vacuum expectation value
`Gf`	G_{Fermi}
`VV12`	V_{ud}
⋮	

Figure 7.3: List of keywords to define parameters to be passed by the function `OLP_SetParameter`.

In the "`UFO` model" setup, the parameters are defined in `UFO` (Universal Feynrules Output) [19] format, which is particularly useful for calculations beyond the Standard Model. The import of the `UFO` model file should be specified in the `order file` by
`Model ufo:/path_to_ufo_model-directory/`.

The `UFO` format also provides human readable name attributes for the model parameters, as well as the SLHA identifiers [29] which are also supported by GoSam. The `UFO` model setup entails the use of a SLHA parameter card to initialize the runtime phase. This requires an additional keyword `ParameterCard`, followed by the path to the SLHA parameter card, to be placed into the order file when using the `UFO` model setup. The parameters which are set by reading in the SLHA parameter card do not need to be set again by `OLP_SetParameter`. However, `OLP_SetParameter` needs to be used at runtime for the dynamic parameters. In this case the SLHA block name should appear as a prefix prepended to the parameter name, in the form `<BlockName>&&<ParamName>`. To avoid confusion, this requires that the characters '`&&`' should never appear in any block or parameter name.

Setting parameters

Parameters are now passed by the subroutine
`OLP_SetParameter(char* para,double* re,double* im,int* ierr)`,

where the first argument is a (pointer to a) string serving as a keyword for the parameter to be set, followed by two double precision numbers so that complex parameters can also be passed (in case of real parameters, the second double is zero). The integer in the fourth argument is set by the OLP to tell the MC whether the setting of the parameter was successful.
`ierr=1` means the parameter has been set successfully,
`ierr=0` means failure: issue an error message,
`ierr=2` means that the parameter is unknown or the setting is ignored (for example because it is irrelevant for the considered case), but the MC program should proceed.

The function `OLP_SetParameter` can be called at runtime, for every phase space point, if used to define a dynamic parameter.

Computing the matrix element

In BLHA2, the routine which returns a value for the matrix element is
`OLP_EvalSubProcess2(int* i, double* pp, double* mu, double* rval, double* acc)`

The arguments are:

- i: pointer to a (one element) array with the label of the subprocess as given in the contract file
- pp: pointer to an array of momenta, conventions $(E_j, k_j^x, k_j^y, k_j^z, M_j)$
- mu: pointer to the renormalisation scale
- rval: pointer to an array of return values
- acc: pointer to a one element array with the outcome of the OLP internal accuracy check

The last argument is an array of length four which is filled by the subroutine, containing the result of the evaluation, as specified in eq. (7.1). The default settings for the prefactor can be changed using the option `nlo_prefactor`, see section 3.

For more details concerning the BLHA2 conventions we refer to [27].

Loop-induced processes

Loop-induced processes are supported by the setting
`AmplitudeType LoopInduced`.

In GoSam, they are not handled like Born processes, but like virtual corrections to non-existing born processes and therefore returned in the virtual field A_0 (`PoleCoeff0`) of `OLP_EvalSubprocess` and `OLP_EvalSubprocess2`. The returned value corresponds to the squared amplitude.

! → Please note that in the order file, `CouplingPower` or `AlphasPower` and `AlphaPower` usually refer to the coupling powers if the corresponding Born amplitude, and the type of the correction is specified as `CorrectionType`. As in the case of loop-induced processes the Born amplitude does not exist, the correct counting of the coupling powers needs to be assured by setting `CouplingPower` (or `AlphasPower` and `AlphaPower`) equal to the order of a corresponding fictitious Born process, i.e. reduce the coupling powers of the loop induced process correspondingly.

BSM-SM-interference processes

GoSam can calculate interference effects between e.g. BSM-Born and SM processes (where the BSM Born for example comes from additional interactions in an effective field theory). These processes are handled as corrections being next-to-leading order in the SM coupling.
In the case where the SM process is loop-induced, the standard pole check would fail due to the non-matching Born. In this case, the user should set `PSP_chk_method=LoopInduced` or `PSP_chk_method=Rotation` in the GoSam input card, or use the GoSam-extension `AmplitudeType LoopInterference` in the BLHA2 order file, which automatically enables `PSP_chk_method=LoopInduced`.

Please note that such a setup may require to define some filters in the input card to select the correct diagrams.

Precision checks

The GoSam input card variable `PSP_chk_method`, which controls the behaviour how GoSam checks the result for each phase-space point, can also be set by `Extra PrecisionCheck`. Possible values are:

- `Extra PrecisionCheck Automatic` *(default)* – chooses automatically between `PoleRotation` and `LoopInduced`
- `Extra PrecisionCheck PoleRotation` – checks the precision of the pole first and rotates if necessary
- `Extra PrecisionCheck Rotation` – estimates the precision of each phase space point by rotating and re-evaluating (slow)
- `Extra PrecisionCheck LoopInduced` – checks that the poles are zero (i.e. very small compared to the finite part) and rotates if necessary
- `Extra PrecisionCheck Disabled` – this sets `PSP_check=False` which switches off all phase space point precision checks.

Subprocess-specific settings in the GoSam input card

Settings in the GoSam input card can be subprocess-specific. This is helpful if various subprocesses, each having different settings, should be calculated at once.

For this purpose, the subprocesses are enumerated as in the BLHA order file, starting at zero (to match to the correspondig `p*` subdirectories created by GoSam).

! → This counting does not necessarily match the labels returned in the BLHA contract file.

The syntax is ***`option[list-of-subprocesses]=value`***. For example, to disable the precision check for the second and third processes in the order file, one can set `PSP_check[1,2]=True` in the input card.

Ranges and exclusion of ranges with `!` (or `^`) are supported. Examples for valid lists:

$$\begin{aligned} \texttt{0-2} &= \{0,1,2\} \\ \texttt{-6,!3-4} &= \{0,1,2,5,6\} \\ \texttt{1-4,!3,9} &= \{1,2,4,9\} \end{aligned}$$

Subprocess-specific settings need to be unambiguous, and they overwrite the corresponding globally set values.

7.4.3 Production of colour-/spin correlated trees

GoSam can also generate tree level amplitudes in a spin- and colour-correlated form. Colour correlated matrix elements are defined as

$$C_{ij} = \langle \mathcal{M} | \mathbf{T}_i \mathbf{T}_j | \mathcal{M} \rangle \; , \tag{7.2}$$

spin-correlated matrix elements can be defined as

$$S_{ij} = \langle \mathcal{M}, - | \mathbf{T}_i \mathbf{T}_j | \mathcal{M}, + \rangle \; . \tag{7.3}$$

The spin-correlated matrix element above (as well as the colour correlated matrix element) contains implicitly the sum over all other helicities, only the helicities with the indices i and j are fixed, i.e.

$$\begin{aligned} &\langle \mathcal{M}_{i,-} | \mathbf{T}_i \cdot \mathbf{T}_j | \mathcal{M}_{i,+} \rangle = \\ &\sum_{\lambda_1,\ldots,\lambda_{i-1},\lambda_{i+1},\ldots,\lambda_n} \langle \mathcal{M}_{\lambda_1,\ldots,\lambda_{i-1},-,\lambda_{i+1},\ldots,\lambda_n} | \mathbf{T}_i \cdot \mathbf{T}_j | \mathcal{M}_{\lambda_1,\ldots,\lambda_{i-1},+,\lambda_{i+1},\ldots,\lambda_n} \rangle \; . \end{aligned} \tag{7.4}$$

These matrix elements are particularly useful in combination with Monte Carlo programs which use these trees to build the dipole subtraction terms for the infrared divergent real radiation part. With these modified tree level matrix elements GoSam is able to generate all necessary building blocks for a complete NLO calculation.
Such a setup has been used successfully in combination with the framework of Herwig++/Matchbox [30, 31, 32].

Appendix A Conventions

A.1 Conventions of `golem95C`

The integral library `golem95C` computes integrals of the form

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

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

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

The commonly used integration measure for the internal momentum k is

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

A.2 Conventions of GoSam

The factor from above which does not go into the integral definition of `golem95C` can be written as

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

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

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

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.

[1] e and g_s in the standard model

The conversion between different conventions for the Γ-functions is straightforward:

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

The relevant terms in the expansion of r_Γ are

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

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

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

Appendix B Explicit reduction of the R_2 terms

The R_2 term [33] consists of all terms of the numerator containing an explicit ε or μ^2 coming from the Lorentz algebra in $D = 4 - 2\varepsilon$ dimensions. For an explicit reduction of these terms, we give a list of all relevant integrals of the form

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

where either α or β is a positive integer number, $\hat{k}$ denotes 4-dimensional loop momenta, $k^2 = \hat{k}^2 - \mu^2$, and the denominators are $D_i = (k + r_i)^2 - m_i^2 + i\delta$. Note that integrals where both α and β are non-zero will not contribute to the final result, as they will be of order ε. An integral of rank r can be written as [34, 35]:

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

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,

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

where $\mathrm{d}^{D_\square} z = \prod_{j=1}^{N} \mathrm{d}z_j \Theta(z_j) \Theta(1 - z_j)$ 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 and momenta $r_j^\bullet$. The kinematic matrix S is given by $S_{ij} = (r_i - r_j)^2 - m_i^2 - m_j^2$.

We only need to consider integrals containing an UV pole, because only the latter lead to a rational term when multiplied with ε stemming either from ε^β or from the integral prefactor

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

The UV divergence comes from the Gamma function

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

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

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

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

$$\frac{1}{\prod_{j=1}^N A_j^{\nu_j}} = \frac{\Gamma(\nu)}{\prod_{j=1}^N \Gamma(\nu_j)} \int \mathrm{d}^D\Box z\, \delta_z \frac{\prod_{j=1}^N z_j^{\nu_j - 1}}{\left(\sum_{j=1}^N z_j A_j\right)^\nu}\ ,\ \nu = \sum_j \nu_j \tag{B.7}$$

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

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

All non-zero cases for integrals where the rank does not exceed the number of propagators are listed below [36, 35]

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

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

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

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

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

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

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

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

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

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

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

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

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

In addition, we list integrals which contribute to the rational part in cases where the rank exceeds the number of propagators, for example in the presence of effective gluon-Higgs couplings, or in models involving gravitons. More details about higher rank integrals can be found in Refs. [8, 3, 10].

$$
\begin{aligned}
I_5^{D,3}(S) &= \int \frac{\mathrm{d}^D k}{i\pi^{D/2}} \frac{\left(\tilde{k}^2\right)^3}{\prod_{j=1}^5 (q_j^2 - m_j^2 + i\delta)} = -\frac{1}{12}\ , \\
I_5^{D,2;\mu_1\mu_2}(a_1, a_2; S) &= \int \frac{\mathrm{d}^D k}{i\pi^{D/2}} \frac{\left(\tilde{k}^2\right)^2 \hat{q}_{a_1}^{\mu_1} \hat{q}_{a_2}^{\mu_2}}{\prod_{j=1}^5 (q_j^2 - m_j^2 + i\delta)} = -\frac{1}{48}\,\hat{g}^{\mu_1\mu_2}\ , \\
I_5^{D,1;\mu_1\cdots\mu_4}(a_1, \ldots, a_4; S) &= \int \frac{\mathrm{d}^D k}{i\pi^{D/2}} \frac{\tilde{k}^2\ \hat{q}_{a_1}^{\mu_1} \ldots \hat{q}_{a_4}^{\mu_4}}{\prod_{j=1}^5 (q_j^2 - m_j^2 + i\delta)} \\
&= -\frac{1}{96}\left[\hat{g}^{\mu_1\mu_2}\hat{g}^{\mu_3\mu_4} + \hat{g}^{\mu_1\mu_3}\hat{g}^{\mu_2\mu_4} + \hat{g}^{\mu_1\mu_4}\hat{g}^{\mu_2\mu_3}\right]\ , \\
\epsilon I_4^{D+6}(S) &= \frac{1}{240}\left(\sum_{i,j=1}^{4} ((r_i - r_j)^2 - m_i^2 - m_j^2) - 2\sum_{i=1}^{4} m_i^2 \right)\ .
\end{aligned}
\tag{B.22}
$$

In the GoSam process card, the default is `r2=explicit`, which means that the rational part R_2 is calculated algebraically using the formulae above, while the integrand reduction can be done in 4 dimensions. Choosing `r2=implicit` means that R_2 will be calculated together with the 4-dimensional part during the reduction.

Appendix C The included model files

C.1 Format of the model files

GoSam expects three files for a proper model definition:

$\langle model \rangle$`.hh` :	`FORM` file containing the Feynman rules
$\langle model \rangle$`.py` :	`Python` file
$\langle model \rangle$:	(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',charge)
```

`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`, ...)
±2spin$_i$:	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.

μ_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 definitions 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 $g^{\mu\nu} = \mathtt{d}(\mu, \nu)$ and $\gamma^\mu = \mathtt{Sm}(\mu)$, $\gamma_5 = \mathtt{Gamma5}$, $\Pi_+ = \mathtt{ProjPlus}$, $\Pi_- = \mathtt{ProjMinus}$. All non-commuting objects must reside inside the function `NCContainter` (see example).

The color structure must use the objects $t^A_{ij} = \mathtt{T}(A, i, j)$ (where the color flow is such that$j$ is the index of an anti-quark), $f^{ABC} = \mathtt{f}(A, B, C)$ and $f^{ABE} f^{CDE} = \mathtt{f4}(A, B, C, D)$. 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 Majorana 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 [37, 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	ν_e
nebar	ne~	0	$\bar{\nu}_e$
mup	mu+	mmu	μ^+
mum	mu-	mmu	μ^-
nmu		0	ν_μ
nmubar	nmu~	0	$\bar{\nu}_\mu$
taup	tau+	mtau	e^+
taum	tau-	mtau	e^-
ntau		0	ν_τ
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	γ
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	ϕ^-
phip	phi+	mW	ϕ^+
chi		mZ	χ

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`	$V^\dagger_{bt}$	— " —

C.3 GoSam 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, the Standard Model (`sm`) is distributed with GoSam, where several variants are available:
`smdiag` implements diagonal flavour structure ($V_{\text{CKM}} = \mathsf{diag}\{1, 1, 1\}$),
`smehc` contains effective gluon-Higgs couplings),
`sm_complex` and `smdiag_complex` support the complex mass scheme.
The structure of the model files is discussed in more detail in Chapter C.1. Model files for the MSSM based on FeynRules/UFO [19] and LanHEP [38] can be found in the directory `examples/model/`, as well as `UFO` files for ADD [39] models with large extra dimensions (LED).

`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 examples of validated processes.

`olp` Files in this directory are used by `gosam.py --olp`, which is GoSam's implementation of the Binoth Les Houches interface for one-loop programs (BLHA). Both the original standards [26] and the new standards (BLHA2) [27] are supported by GoSam 2.0.

Conditions of use

- Hans van Deurzen
- Nicolas Greiner
- Gudrun Heinrich
- Gionata Luisoni
- Pierpaolo Mastrolia
- Giovanni Ossola
- Tiziano Peraro
- Johannes Schlenk
- Johann Felix von Soden-Fraunhofen
- Francesco Tramontano

Former members:

- Gavin Cullen
- Edoardo Mirabella
- Joscha Reichel
- Thomas Reiter

We are indebted to **Thomas Reiter** for his invaluable contributions to the development of GoSam-1.0.

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, H. van Deurzen, N. Greiner, G. Heinrich, G. Luisoni, P. Mastrolia, E. Mirabella, G. Ossola, T. Peraro, J. Schlenk, J.F.von Soden-Fraunhofen, F. Tramontano,
> "GoSam-2.0: a tool for automated one-loop calculations within the Standard Model and Beyond",
> Eur. Phys. J. C **74** (2014) 8, 3001 [arXiv:1404.7096 [hep-ph]].

Bibliography

[1] G. Cullen, N. Greiner, G. Heinrich, G. Luisoni, P. Mastrolia, et al., *Automated One-Loop Calculations with GoSam*, *Eur.Phys.J.* **C72** (2012) 1889, [`arXiv:1111.2034`].

[2] G. Cullen, H. van Deurzen, N. Greiner, G. Heinrich, G. Luisoni, et al., *GOSAM-2.0: a tool for automated one-loop calculations within the Standard Model and beyond*, *Eur.Phys.J.* **C74** (2014), no. 8 3001, [`arXiv:1404.7096`].

[3] P. Mastrolia, E. Mirabella, and T. Peraro, *Integrand reduction of one-loop scattering amplitudes through Laurent series expansion*, *JHEP* **1206** (2012) 095, [`arXiv:1203.0291`].

[4] H. van Deurzen, G. Luisoni, P. Mastrolia, E. Mirabella, G. Ossola, et al., *Multi-leg One-loop Massive Amplitudes from Integrand Reduction via Laurent Expansion*, `arXiv:1312.6678`.

[5] T. Peraro, *Ninja: Automated Integrand Reduction via Laurent Expansion for One-Loop Amplitudes*, `arXiv:1403.1229`.

[6] T. Binoth, J. P. 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, [`arXiv:0810.0992`].

[7] G. Cullen, J. 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, [`arXiv:1101.5595`].

[8] J. P. Guillet, G. Heinrich, and J. von Soden-Fraunhofen, *Tools for NLO automation: extension of the golem95C integral library*, `arXiv:1312.3887`.

[9] P. Mastrolia, G. Ossola, T. Reiter, and F. Tramontano, *Scattering AMplitudes from Unitarity-based Reduction Algorithm at the Integrand-level*, *JHEP* **08** (2010) 080, [`arXiv:1006.0710`].

[10] H. van Deurzen, *Associated Higgs Production at NLO with GoSam*, *Acta Phys.Polon.* **B44** (2013), no. 11 2223–2230.

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

[12] J. A. M. Vermaseren, *New features of FORM*, `math-ph/0010025`.

[13] J. Kuipers, T. Ueda, J. Vermaseren, and J. Vollinga, *FORM version 4.0*, `arXiv:1203.6543`.

[14] T. Reiter, *Optimising Code Generation with haggies*, *Comput.Phys.Commun.* **181** (2010) 1301–1331, [`arXiv:0907.3714`].

[15] G. Heinrich, G. Ossola, T. Reiter, and F. Tramontano, *Tensorial Reconstruction at the Integrand Level*, *JHEP* **1010** (2010) 105, [`arXiv:1008.2441`].

[16] M. A. Malcolm, *An algorithm for floating-point accumulation of sums with small relative error*, tech. rep., Stanford University, Stanford, CA, USA, 1970.

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

[18] T. Ohl, *Drawing Feynman diagrams with Latex and Metafont*, *Comput. Phys. Commun.* **90** (1995) 340–354, [`hep-ph/9505351`].

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

[20] T. Gleisberg, F. Krauss, K. T. Matchev, A. Schalicke, S. Schumann, et al., *Helicity formalism for spin-2 particles*, *JHEP* **0309** (2003) 001, [`hep-ph/0306182`].

[21] N. Greiner, G. Heinrich, J. Reichel, and J. F. von Soden-Fraunhofen, *NLO QCD Corrections to Diphoton Plus Jet Production through Graviton Exchange*, *JHEP* **1311** (2013) 028, [`arXiv:1308.2194`].

[22] T. Han, J. D. Lykken, and R.-J. Zhang, *On Kaluza-Klein states from large extra dimensions*, *Phys.Rev.* **D59** (1999) 105006, [`hep-ph/9811350`].

[23] S. Catani, S. Dittmaier, and Z. Trocsanyi, *One loop singular behavior of QCD and SUSY QCD amplitudes with massive partons*, *Phys.Lett.* **B500** (2001) 149–160, [`hep-ph/0011222`].

[24] S. Badger, B. Biedermann, and P. Uwer, *NGluon: A Package to Calculate One-loop Multi-gluon Amplitudes*, *Comput.Phys.Commun.* **182** (2011) 1674–1692, [`arXiv:1011.2900`].

[25] A. Denner, S. Dittmaier, M. Roth, and L. Wieders, *Electroweak corrections to charged-current $e^+e^- \to 4$ fermion processes: Technical details and further results*, *Nucl.Phys.* **B724** (2005) 247–294, [`hep-ph/0505042`].

[26] 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, [`arXiv:1001.1307`]. Dedicated to the memory of, and in tribute to, Thomas Binoth, who led the effort to develop this proposal for Les Houches 2009.

[27] S. Alioli, S. Badger, J. Bellm, B. Biedermann, F. Boudjema, et al., *Update of the Binoth Les Houches Accord for a standard interface between Monte Carlo tools and one-loop programs*, *Comput.Phys.Commun.* **185** (2014) 560–571, [`arXiv:1308.3462`].

[28] **Particle Data Group** Collaboration, J. Beringer et al., *Review of Particle Physics (RPP)*, *Phys.Rev.* **D86** (2012) 010001.

[29] P. Z. Skands, B. Allanach, H. Baer, C. Balazs, G. Belanger, et al., *SUSY Les Houches accord: Interfacing SUSY spectrum calculators, decay packages, and event generators*, *JHEP* **0407** (2004) 036, [`hep-ph/0311123`].

[30] Proceedings of the Les Houches 2013 workshop on Physics at TeV colliders, 2014.

[31] J. Bellm, S. Gieseke, D. Grellscheid, A. Papaefstathiou, S. Plätzer, et al., *Herwig++ 2.7 Release Note*, `arXiv:1310.6877`.

[32] S. Plätzer and S. Gieseke, *Dipole Showers and Automated NLO Matching in Herwig++*, *Eur.Phys.J.* **C72** (2012) 2187, [`arXiv:1109.6256`].

[33] G. Ossola, C. G. Papadopoulos, and R. Pittau, *On the Rational Terms of the one-loop amplitudes*, *JHEP* **0805** (2008) 004, [`arXiv:0802.1876`].

[34] T. Binoth, J. P. Guillet, G. Heinrich, E. Pilon, and C. Schubert, *An Algebraic/numerical formalism for one-loop multi-leg amplitudes*, *JHEP* **0510** (2005) 015, [`hep-ph/0504267`].

[35] T. Reiter, *Automated Evaluation of One-Loop Six-Point Processes for the LHC*, `arXiv:0903.0947`.

[36] T. Binoth, J. P. Guillet, and G. Heinrich, *Algebraic evaluation of rational polynomials in one-loop amplitudes*, *JHEP* **0702** (2007) 013, [`hep-ph/0609054`].

[37] M. Böhm, A. Denner, and H. Joos, *Gauge Theories of the Strong and Electroweak Interaction.* Teubner, Stuttgart and Leipzig and Wiesbaden, 3rd ed., 2001.

[38] A. Semenov, *LanHEP - a package for automatic generation of Feynman rules from the Lagrangian. Updated version 3.1*, `arXiv:1005.1909`.

[39] N. Arkani-Hamed, S. Dimopoulos, and G. Dvali, *The Hierarchy problem and new dimensions at a millimeter*, *Phys.Lett.* **B429** (1998) 263–272, [`hep-ph/9803315`].