Computer Physics Communications 306 (2025) 109372

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

Computer Programs in Physics

MaRTIn – Massive Recursive Tensor Integration ☆

Joachim Brod [a], Lorenz Hüdepohl [b], Emmanuel Stamou [c], Tom Steudtner [a,c,*]

[a] Department of Physics, University of Cincinnati, Cincinnati, OH 45221, USA
[b] Max Planck Computing and Data Facility, D-85748 Garching, Germany
[c] Department of Physics, TU Dortmund, D-44221 Dortmund, Germany

ARTICLE INFO

Keywords:
Loop calculation
Perturbation theory
Momentum expansion
Infrared rearrangement

ABSTRACT

We present MaRTIn, an extendable all-in-one package for calculating amplitudes up to two loops in an expansion in external momenta or using the method of infrared rearrangement. Renormalisable and non-renormalisable models can be supplied by the user; an implementation of the Standard Model is included in the package. In this manual, we discuss the scope and functionality of the software, and give instructions of its use.

Contents

☆ The review of this paper was arranged by Prof. Z. Was.

* Corresponding author at: Department of Physics, TU Dortmund, D-44221 Dortmund, Germany.
E-mail address: tom2.steudtner@tu-dortmund.de (T. Steudtner).

https://doi.org/10.1016/j.cpc.2024.109372
Received 29 April 2024; Received in revised form 31 July 2024; Accepted 5 September 2024
Available online 10 September 2024
0010-4655/© 2024 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

PROGRAM SUMMARY

Program Title: MaRTIn
CPC Library link to program files: https://doi.org/10.17632/bf96rjxnky.1
Developer's repository link: https://gitlab.com/manstam/martin
Licensing provisions: GPLv3
Programming language: FORM, python
Nature of problem: The computation of tree-level processes and quantum corrections in perturbation theory requires several complicated steps. This potentially includes diagram generation, resolving gauge and flavour algebras, spinor contractions, tensor reduction, application of integration-by-parts relations and scalar integration.
Solution method: Arbitrary relativistic QFTs can be defined via field contents and Feynman rules. Additional information regarding the desired amplitude and compuation are supplied by the user. Using these inputs, all steps of the calculation are conducted automatically.
Additional comments including restrictions and unusual features: MaRTIn is designed as an extensible all-in-one solution, no additional glue code is required. Currently, computations may be conducted up to two loops with either an expansion in external momenta or infrared rearrangement via a mass parameter. Several options for the treatment of γ_5 are implemented.

1. Overview

Perturbation theory is one of the most reliable tools to obtain quality predictions from quantum field theories. In order to further enhance the precision, higher loop orders are required, increasing the computational complexity of the calculation and with it the need for more sophisticated software pipelines to automatise them. Traditionally, these tool chains follow a modular structure, though not all pieces or build scripts are public. For instance, generic Feynman diagrams can be generated via QGRAF [1], Feynarts [2,3] or future versions of FORM [4,5]. These expressions require further manipulation such as inserting Feynman rules, problem-specific diagram filtering and expansions, partial fractioning, tensor reduction and topology identification. At one-loop, these tasks can be performed using tools such as FeynCalc [6]. At higher loop orders, subsets of these tasks can be delegated to tools like DIANA [7], q2e and exp [8,9], LIMIT [10], tapir [11], TopoID [12], Alibrary [13] and Feynson [14]. Moreover, the algebra of spinor contractions and symmetry groups has to be simplified. Note that there is a plethora of software packages automatising computations at tree-level and one-loop order, employing a Passarino–Veltman decomposition of integrals [15]. However, already at two loops a more sophisticated methodology is required. Integration-by-parts (IBP) relations are recursively applied to loop integrals until they have reduced to master integrals. This step may be performed by LiteRed [16,17], FIRE [18–21], Reduze [22,23] or Kira [24,25]. Examples of packages combining the IBP reductions and insertion of master integrals for specific families of problems are MINCER [26], MATAD [27], FORCER [28] and FMFT [29].

MaRTIn (Massive Recursive Tensor Integration) aims to be an all-in-one tool for the computation of multi-leg amplitudes in an expansion of small external momenta at tree and loop level in arbitrary quantum field theories. A typical computation starts from Feynman rules specified by the user, which may comprise both renormalisable and non-renormalisable operators. MaRTIn leverages QGRAF [1] to generate Feynman diagrams automatically, and proceeds with the main computation using routines implemented in FORM [4,5]. This includes the resolution of spinor, gauge, and global symmetry index contractions, as well as the tensor and IBP reduction and, eventually, scalar loop integration. Several schemes for the treatment γ_5 are available.

The major mode of operation employs either an expansion in all external momenta, or the infrared rearrangement outlined in Refs. [30,31]. In either case, integrals are reduced to vacuum topologies with arbitrary, possibly vanishing masses. Tensor integrals are reduced to scalar integrals using the algorithms in Refs. [31,32]. The integration is then performed symbolically up to two-loop order in $4-2\epsilon$ space-time dimensions using the algorithms in Refs. [33,34], to a user-specified order in powers of ϵ and external momenta. The functionality to perform calculations in $3-2\epsilon$ space-time dimensions is contained in the package, but is not as well-tested.

Another mode of operation retains the full momentum dependence but does not perform any integration. This is implemented up to one-loop order. The functionality of MaRTIn may be extended in future releases.

MaRTIn produces outputs in both FORM and Mathematica [35] compatible formats, and optionally a graphical representation of all involved Feynman diagrams.

In the following sections, we specify in detail how MaRTIn is obtained, installed, operated, and extended.

2. Setup

MaRTIn is distributed under the *GNU Public License, Version 3* [36] and available to download at https://gitlab.com/manstam/martin. The code itself is mostly written in and tested against FORM 4 [5], which has to be installed separately. Similarly, MaRTIn requires QGRAF [1] to generate diagrams. Note that both packages have several subdependencies and need to be compiled from their source codes. Thus, a POSIX compatible system with a working GNU toolchain is required. MaRTIn is operated using GNU make as well as a python3 script, which in turn requires the common modules os, sys, shutil, re, argparse, subprocess and configparser. The underlying Makefile also utilises bash, (GNU) awk and several other standard shell programs. In order to output Feynman diagrams in PDF format, graphviz with the neato engine are required. Optionally, the package richard_draw may be utilised, which produces higher-quality output using lualatex. It can be downloaded at https://gitlab.com/manstam/richard_draw.

Once all prerequisites are met, MaRTIn needs to be set up in three places:

- The source code directory martin. It requires no further installation, but a system soft link to the main executable martin/martin.py is recommended. The source directory includes the subdirectories code with the source codes, user_template with some prototypes of the files, and further directories mentioned below, as well as this manual.
- A config file .martin.conf, which should be manually placed in the user's home directory. A template version of the file is contained in martin/user_template/template_martin.conf. This file specifies the locations to the FORM and QGRAF executables, the number of default parallel processes, FORM options and default paths. The options are detailed in the template file itself.

Table 1
Overview of MaRTIn input and output files. See text for details.

Input files
QGRAF files (diagrams)
models/qgraf/*model*.prop.lag
models/qgraf/*model*.vrtx.lag
FORM model file (Feynman rules)
models/form/model_*MODEL*
Problem file (process and options)
problems/*process*/loop.*problem*.dat
Optional: richard_draw file (drawing)
models/rdraw/*model*.json
Output files
FORM output
results/*process*/form.*problem*/dia*.sav
Mathematica output
results/*process*/math.*problem*/dia*.m
Feynman graphs
results/*process*/graphs.*problem*.pdf
richard_draw Feynman graphs
results/*process*/rgraphs.*problem*.pdf

- The user's working directory, e.g., named martin_user, which should be created by the user and populated with the following subdirectories (examples are found in martin/user_template):
  - models: This directory contains all user-defined model files. Further details are deferred to Sec. 4.
  - problems: contains (possibly nested) subdirectories, which can have arbitrary names. Each of these subdirectories may contain several loop.*.dat files. Every such file specifies all options required for a single run of MaRTIn. The contents of these files will be explained in Sec. 3.
  - prc: is the canonical location for user-defined FORM routines that will be automatically detected by MaRTIn. The routines can then be included in the FORM folds of the model or loop.*.dat files (see Sec. 3.1).
  - results: This is the location were MaRTIn automatically writes intermediate and final output, creating the same folder structure as in the user-defined problems directory. The user should not manually modify or add to the contents of this directory since MaRTIn's functionality can overwrite and delete these directories.

In order to test whether the MaRTIn installation is complete, users can calculate the example described in greater detail in Sec. 5. All files necessary for this run are contained in MaRTIn. Assuming that FORM, QGRAF, and MaRTIn are properly set up, the command

```
martin ./user_template/problems/SM/loop.1_uu.dat
```

invoked from the main MaRTIn directory should generate and compute the diagram in Fig. 1. The command

```
martin ./user_template/problems/SM/loop.1_uu.dat pdf
```

should generate a pdf file of the corresponding diagrams. Using rpdf instead of the target pdf would instead use richard_draw to create a pdf file for the diagrams. A successful run will provide the screen printout discussed in Sec. 5, and save all results in the directory ./user_template/results/SM/.

In the next section, we describe the concrete workflow of MaRTIn in detail.

3. Workflow

In this section we describe how MaRTIn is used. A list of the required input files, as well as the output files, is contained in Table 1. The two key elements for the operation of MaRTIn are the loop.*problem*.dat, which contains all information about the calculation at hand, as well as the main executable martin.py, which serves as a convenience wrapper around the Makefile. Note that *problem* here represents a placeholder for a user-defined identifier. Different steps of the computation are triggered by *targets* arguments passed to the executable. Assuming that the full path to martin.py has been aliased as the command martin, a typical MaRTIn call has the following shape:

```
martin problems/process/loop.problem.dat targets
```

The command

```
martin -h
```

lists and explains all target options. There are many targets that are only important for the inner workings of MaRTIn and are evaluated successively.

By default, the output of all targets (including intermediate ones) is written to the directory `results/process/`. For the user, the most relevant targets are the following:

- `all` is the default target of `martin.py` and will be called if no target argument is specified. It computes all diagrams and saves the results in the `results` folder, as discussed above. For every diagram that has not been computed already, `MaRTIn` will spawn a separate process to do so. The maximal number of parallel processes can be configured in `.martin.config`.
- `diaNUM` computes the single diagram with number `NUM`. The result is written to `results/process/form.problem/diaNUM.sav` using `FORM`s binary format, as well as `results/process/math.problem/diaNUM.m` in Wolfram Language.
- `sum` computes the sum of all Feynman diagrams. The results are stored as `results/process/form.problem/diasum.sav` in `FORM` as well as `results/process/math.problem/diasum.m` in `Mathematica` compatible format. If any individual `diaNUM.sav` files are missing, they will be computed with the `all` target.
- `pdf` will generate a PDF file containing the graphical representation of each Feynman diagram. This is stored in `results/process/-graphs.problem.pdf`.
- `rpdf` utilises `richard_draw` to generate a higher-quality pdf file containing all Feynman graphs. This is stored under `results/process/r-graphs.problem.pdf`.
- `clean` removes all final and intermediate results of the selected problem.

The executable `martin.py` also takes arguments that can be used to override options in the config file `.martin.conf`, call `martin -h` for further details. In the remainder of this section, we specify the content and structure of a valid `loop.problem.dat` file, before giving an overview of `MaRTIn`'s algorithm.

3.1. Problem file

The file `problems/process/loop.problem.dat` contains all necessary information for a single run of `MaRTIn`. It specifies how the model is assembled from the modular pieces that will be introduced in Sec. 4, defines in detail which Feynman diagrams are to be calculated, configures various aspects of the computation itself, and specifies if and how user-defined code is injected. Example files can be found in the `MaRTIn` source directory at `user_template/problems/SM/`.

The file itself consists of `FORM` folds, i.e. named objects of the syntax

```
*--#[ FOLDNAME :
  some code here
*--#] FOLDNAME :
```

which are being inserted at the right place by `FORM`'s preprocessor. These objects must never be deleted from the file, even if they are empty. The content of each fold is `FORM` code. The only exception is the very first fold we encounter, which actually contains `QGRAF`'s own scripting language. In fact, this fold contains the contents of the `QGRAF` input file:

```
*--#[ QGRAF :
*   This is a comment.
*   Do not remove or reorder statements below.

*   Define which QGRAF model files to use.
*   We typically split them into propagators and vertices, but that is not required.
*   Both are should be valid files located in models/qgraf/
*   The functionality of including multiple models in this
*   way is a MaRTIn feature; it is not possible in solo QGRAF runs
  model = "qmodel.prop.lag";
  model = "qmodel.vrtx.lag";

*   Define external legs using field names.
*   Use q's for external momenta (note that martin substitutes q0 = 0.
  in  = field1[q1], field2[q2];
  out = field3[q3], field4[q4];

*   Loop order, e.g. 2. Loop momenta have to be p1, p2.
  loops = 2;
  loop_momentum = p;

*   Some options to select topologies, e.g.:
  options = onepi;

*   More selectors like vsums and psums here

*--#] QGRAF :
```

Note that `MaRTIn` allows for the presence of several separate `QGRAF` model files; they are combined automatically before being passed to `QGRAF` by concatenating (`cat`) them in the order they appear in the fold. It is the user's responsibility to ensure that the model files are consistent with each other. We refer to the `QGRAF` manual for a more detailed description of the valid syntax.

The next fold sets up important options for the `FORM` part of `MaRTIn`. These are saved as preprocessor variables, which are collected and described in Table 2. An example and more explanations are given below.

```
*--#[ MAIN :
*   This is a FORM comment.

*   Each model may consist of several form model files found in models/form/
*   The total number of model files must be given as NM.
*   For instance, for two model files with names "model_fmodelA" and "model_fmodelB":
```

Table 2
Overview of preprocessor options to configure MaRTIn. These are found in the fold MAIN located in each problem file.

Option	Description
NM	Represents the number of FORM model files that are being combined for the current problem.
MODEL*i*	Series of variables with integer *i* going from 1 to NM. Each refers to a filename model_MODELx of a FORM model file found in models/form/.
EXPDENO	Before integration, expand in external momenta to a power given by this variable. This option is exclusive with IRA and GENERICLOOPFUNCTIONS.
IRA	Perform infrared rearrangement before integration. Expand external momenta up to a power specified by this variable. Exclusive option with EXPDENO and GENERICLOOPFUNCTIONS.
GENERICLOOP -FUNCTIONS	Skip integration, collect integrand in generic function. Exclusive option with EXPDENO and IRA.
FINALEPLIM	Power in ϵ up to which the final result is expanded.
DSCHEME	Scheme for handling γ_5. Options are "NDR" (default), "sNDR", "HV" and "LARIN". See main text for details.
CLLABEL	If defined, mark each closed fermion loop with a function cl tracking the fields in the loop.
PRINT	If defined, intermediate steps in the computation of each diagram are printed to the terminal.
FINALPRINT	If defined, the final result of each diagram is printed to the terminal.

```
  #define NM "2"
  #define MODEL1 "fmodelA"
  #define MODEL2 "fmodelB"

*** Modes of operation. Only one of these three should be defined.
*   Expand in external momenta to this power
  #define EXPDENO "2"
*   Define if infrared rearrangement should be used.
  #define IRA "1"
*   No integration, keep a generic expression of unsolved integrals
  #define GENERICLOOPFUNCTIONS
```

Defining `EXPDENO "n"` will perform a Taylor expansion of all propagators $\frac{1}{(p+q)^2-m^2}$ up to powers n in all external momenta q. Alternatively, setting the option `IRA "n"` the code performs an infrared rearrangement [30,31], i.e., it introduces the regularising mass parameter m_{IRA} via the repeated application of the identity

$$\frac{1}{(p+q)^2-M^2} = \frac{1}{p^2-m_{\text{IRA}}^2} + \frac{M^2-m_{\text{IRA}}^2-2\,p\cdot q-q^2}{p^2-m_{\text{IRA}}^2}\frac{1}{(p+q)^2-M^2}\,. \tag{1}$$

External momenta are kept up to power n. In practice, this identity is applied recursively to every term until its degree of divergence is negative; these terms are then dropped. Note that the m_{IRA}^2 in the numerator of the second term on the right side of the equation is dropped; accordingly, additional local counterterms must be introduced (see Ref. [31] for details).

```
*   Order in ε to which final result is expanded
*   This is not relevant for the GENERICLOOPFUNCTIONS option
  #define FINALEPLIM "-1"

*** Scheme how gamma5 should be treated. The options are:
*   NDR - naive dimensional regularisation, abort on gamma5 encounter inside traces
*   sNDR - semi-naive dimensional regularisation, use naive relations
*          in (4-2ε) dimensions, but set flagsNDR
*   HV - Breitenlohner - Maison - 't Hooft - Veltman treatment
*   LARIN - Larin's scheme: replace gamma5 by Levi-Civita contraction
  #define DSCHEME "NDR"
```

A few comments about the implemented γ_5 schemes are in order. The issues with γ_5 arise due to the choice of dimensional regularisation [37]. The γ_5 matrix is defined as

$$\gamma_5 = \frac{i}{4!}\epsilon_{\mu\nu\rho\sigma}\gamma^\mu\gamma^\nu\gamma^\rho\gamma^\sigma\,, \tag{2}$$

where $\epsilon_{\mu\nu\rho\sigma}$ denotes the completely antisymmetric Levi-Civita tensor in four space-time dimensions, with the convention $\epsilon_{0123}=1$.

It is well known that the anticommutation relation

$$\{\gamma_5, \gamma^\mu\} = 0 \tag{3}$$

is algebraically inconsistent with the trace operation in $d \neq 4$ space-time dimensions [38]. In order enable consistent treatments of γ_5, `MaRTIn` internally can keep track of whether fermion lines are open or closed, how many γ_5 appear, and whether Lorentz indices are $(4-2\epsilon)$-, 4- or (-2ϵ)-dimensional. The following options to handle γ_5 are currently implemented:

- `#define DSCHEME "NDR"`: "Naive dimensional regularisation"
  This is the default choice of the algorithm. If γ_5 is encountered inside a trace, `MaRTIn` exits with an error since NDR can give algebraically inconsistent results in traces. In open fermion lines, γ_5 matrices are moved to the right, assuming anticommutation $\{\gamma_5, \gamma^\mu\} = 0$.
- `#define DSCHEME "sNDR"`: "Semi-Naive dimensional regularisation" [39,40]
  This algorithm always uses the anticommutation relation in Eq. (3). Inside traces, γ_5 is replaced via the relation in Eq. (2), and the Levi-Civita tensors and their contractions are formally treated as $(4-2\epsilon)$-dimensional. The implication of this simplifying assumption is that only $(4-2\epsilon)$-dimensional metric tensors and γ-matrices will appear in the final result. Of course, the relation in Eq. (2) is then actually inconsistent unless additional terms $\propto \epsilon$ are introduced, which this scheme neglects. Instead, whenever this insertion is made the term is marked by the variable `flagsNDR`. Thus, starting from a certain order of ϵ, the `sNDR` algorithm does not yield a consistent result. The highest consistent order is the first one at which `flagsNDR` appears. Tracking the `flagsNDR` dependence is the responsibility of the user.
- `#define DSCHEME "HV"`: "'t Hooft–Veltman scheme" [41–44]
  The γ_5 matrix is assumed to anticommute with γ^μ if $\mu = 0, 1, 2, 3$, and to commute with γ^μ, otherwise; i.e. $\{\gamma_5, \tilde{\gamma}^\mu\} = 0$ and $[\gamma_5, \hat{\gamma}^\mu] = 0$, where $\tilde{\gamma}^\mu$ and $\hat{\gamma}^\mu$ are the projections onto the 4- and $d-4$-dimensional subspaces, respectively. Internally, projected Lorentz indices are contracted via the projected metric tensors `dd`, `ddtilde` and `ddhat` in $(4-2\epsilon)$, 4 and (-2ϵ) dimensions, respectively. Levi-Civita tensors are treated in four integer dimensions.
- `#define DSCHEME "LARIN"`: "Larin scheme" [45]
  In this scheme, any instance of γ_5 on both open and closed fermion lines is replaced via the relation in Eq. (2) before traces are evaluated. The Levi-Civita tensor is treated in four integer dimensions.

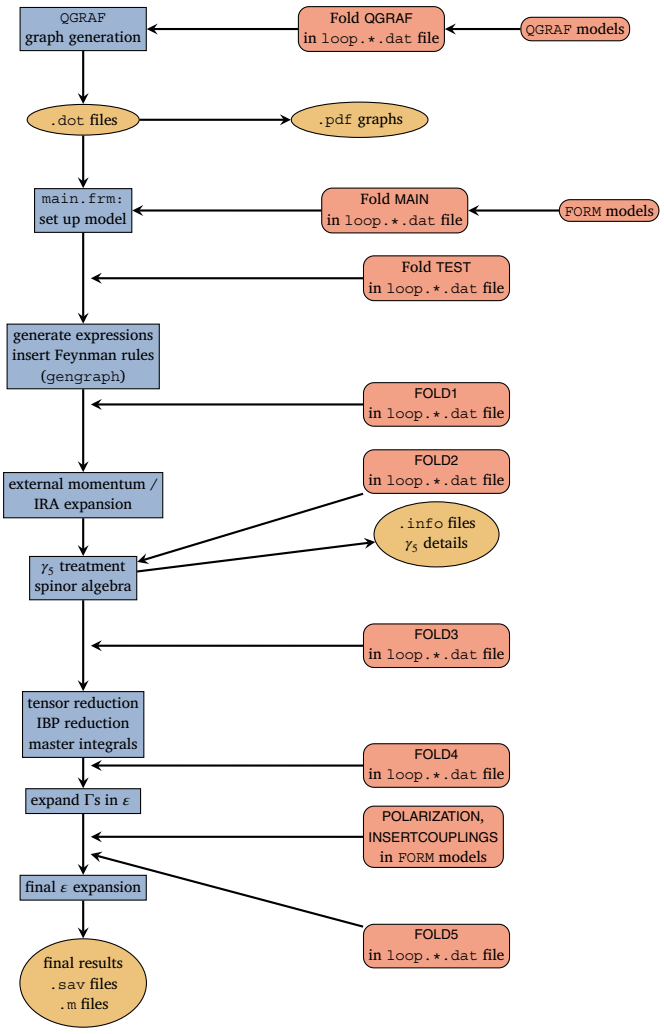
```
*   Define to introduce a label for each closed fermion line.
*   Syntax: cl(line_id, field_1 , ..., field_n )
  #define CLLABEL

*   Define to print the final result to the screen.
  #define FINALPRINT

*   Define to get more information about intermediate steps of the calculation.
  #define PRINT


*--#] MAIN :
```

Finally, there are several folds to inject user-defined `FORM` code, see also Sec. 3.2.

```
*--#[ USERDEF :
*   Fill with your own FORM declarations.
*--#] USERDEF :

*--#[ TEST :
*   Code injection after model definitions, but before diagrams are generated.
*--#] TEST :

*--#[ FOLD1 :
*   Code injection right after expressions for each diagram are generated.
*--#] FOLD1 :

*--#[ FOLD2 :
*   Code injection before Dirac algebra treatment.
*--#] FOLD2 :

*--#[ FOLD3 :
*   Code injection before integration.
*--#] FOLD3 :

*--#[ FOLD4 :
*   Code injection after integration.
*--#] FOLD4 :

*--#[ FOLD5 :
*   Code injection after ε expansion and coupling insertion.
*   Recommendation: replace function internally used for (μ/M)^2ε.
  id eM(M?) = 1
            - 2*ep*Log(M)
            + 2*ep*Log(mu)
            + 2*ep^2*Log(M)^2
            + 2*ep^2*Log(mu)^2
            - 4*ep^2*Log(mu)*Log(M);
*--#] FOLD5 :
```

In the next section, we detail the working algorithm of `MaRTIn` and also show where each fold is included.

3.2. Algorithm



Simplified overview of MaRTIn's computational pipeline. Red[1] rounded squares represent input data, blue squares parts of the computation, and yellow ellipses indicate output files.

4. Model definition

In the previous section, we saw that each model is defined by a set of related QGRAF and FORM model files. In this section, we will give some details on the vital components of these files, which should enable users to implement their own models.

MaRTIn uses the QGRAF and FORM model files to assemble a symbolic representation of the amplitude from the implemented Feynman rules. To achieve this, it uses the leg numbers generated by QGRAF to (i) bring spinorial terms into the correct order, (ii) generate all necessary (gauge and Lorentz) indices, and (iii) substitute all Feynman rules.

4.1. QGRAF files

The QGRAF-model files are located in the working directory at models/qgraf/. They are simply QGRAF model files, the syntax may include any elements valid for the locally installed QGRAF version. It is recommended to provide two separate files, one for all propagators and one for all vertices, named *model*.prop.lag and *model*.vrtx.lag, respectively. The reason for this is that propagators must be defined before vertices in QGRAF. Splitting each model in a propagator and a vertex part makes it possible to include multiple models. MaRTIn will then automatically combine all QGRAF model files into a single file in the correct order and pass it to QGRAF. The propagator file essentially contains several statements of the type

```
[Boson, BosonBar, +; pfunct='function', m='mass']

[Fermion, FermionBar, -; pfunct='function', m='mass']
```

[1] For interpretation of the colours in the figure(s), the reader is referred to the web version of this article.

where cursive words are placeholders for identifiers to be defined by the user. To be precise, the first two argument define the names of the fields and their adjoints. For real fields, both are the same. The names must be declared symbols in the FORM model file. The third argument determines if the field is fermionic or bosonic. The keyword pfunct after the semicolon is required. Its argument has to be declared as a commuting FORM function in the FORM model file and represents the propagator function. MaRTIn automatically populates its arguments with information about the propagator and its fields. Similarly, the keyword m is mandatory and represents the mass of the propagating field. It has to be a valid FORM symbol, to be declared in the FORM model file. Note that M0 is implicitly assumed to be a vanishing mass. All additional keywords, including external, are admissible. In the same manner, vertex file contains several statements of the form

```
[Field1, Field2, Field3, ...; vfunct='function']
```

where additional keywords are allowed. vfunct is required by MaRTIn and represents the vertex function. It has to be declared in the FORM model file as a commuting function. Note that MaRTIn provides its own QGRAF style file. The output of QGRAF is distributed into one .dot file per diagram, which are read by the FORM routine and handled in a parallelised fashion.

4.2. FORM files

The FORM model files contain the main information about the QFT at hand. While the QGRAF files define fields and specify vertices by their legs, the actual Feynman rules are implemented in the FORM model files, which MaRTIn's gengraph routine uses to generate the amplitude. The FORM model files are located in models/form/ in the user's working directory, and consist entirely of FORM folds. None of these folds may be omitted from the file. We recommend using an existing model file as a template for creating a new one. An empty template is provided in the MaRTIn source directory at user_template/models/form/model_MODEL. A full working example, model_SM, can be found in the same folder.

4.2.1. Internal representation of propagators and vertices

The gengraph routine uses the raw QGRAF output to generate a symbolic amplitude according to the Feynman rules implemented in the FORM model file. The essential steps comprise the generation of all group indices, insertion of the Feynman rules, and ordering of the fermion lines. In this section, we briefly describe how propagators and vertices are represented internally, and introduce MaRTIn's fermionline objects.

All propagators and vertices are initially (i.e. before the substitution of the Feynman rules) represented by commuting functions. Their names are assigned by the pfunct and vfunct statements in the QGRAF model files. These commuting functions must be declared in the DEF fold of the FORM model file (see Sec. 4.2.8). In addition, commuting functions representing external leg factors are automatically generated. Therefore, all field (and anti-field) variables used in the QGRAF model files must also be declared as symbols in the DEF fold.

Using MaRTIn's style file, QGRAF provides a unique description of the topology of any given Feynman diagram by providing either unique pairs of positive numbers, with one number being associated with one end of a propagator and the same number being associated with the corresponding (internal) leg of a vertex; or by providing a unique pair of negative numbers, with one number being associated with an external leg of a vertex, and the same number labelling a corresponding "polarization function". The initial symbolic representation of any propagator function PROP is, thus, of the form

$$\texttt{PROP}(i_1,i_2,\texttt{mom},p,\texttt{mass},m,\texttt{field},f)$$

Here, i_1 and i_2 are two unique positive integers provided by QGRAF. They are followed by the momentum p of the propagator, the mass m of the propagator, and a symbol f denoting the name of the field to which the propagator belongs. These variables are indicated by the "separator" variables mom, mass, and field, respectively. These arguments help the user identifying each propagator and inserting the correct Feynman rule.

The situation is similar for all N-leg vertices. Here the order of fields is the same as defined in the QGRAF model file. Any vertex function VERTEX is of the form

$$\texttt{VERTEX}(i_1,\ldots,i_N,\texttt{mom},p_1,\ldots,p_N)$$

where $i_1,\ldots,i_N$ are unique integers provided by QGRAF, positive of negative depending on whether the corresponding leg is internal or external. mom is a separator, followed by the incoming momenta for each leg.

Finally, each external leg receives a polarization function POL of the form

$$\texttt{POL}(f,i,p)$$

with f again the symbol denoting the name of the corresponding field, i a unique negative integer provided by QGRAF, and p the momentum of the external line. Whether the momentum is incoming or outgoing depends on whether the corresponding leg/state has been chosen as incoming or outgoing in the QGRAF fold.

These representations are the starting point of all further manipulations, described in the following subsections.

4.2.2. Generation of group indices

While the gauge-group and global symmetry structure of the Feynman rules is model-dependent, the Lorentz structure of the Feynman rules is more generic. Thus, MaRTIn will always generate Lorentz indices automatically. Specifically, for each pfunct propagator function PROP, MaRTIn will generate a unique pair of Lorentz indices as follows:

$$\texttt{PROP}(i_1,i_2,\texttt{lorentz},\nu_{i_1},\nu_{i_2},\texttt{mom},p,\texttt{mass},m,\texttt{field},f)$$

Similarly, for each vfunct vertex function VERTEX corresponding to a vertex with N legs, MaRTIn will generate Lorentz indices for each leg as follows:

```
VERTEX(i_1,...,i_N,lorentz,v_{i_1},...,v_{i_N},mom,p_1,...,p_N)
```

The polarization functions are treated in the same way. In each case, a new separator `lorentz` is generated, followed by N unique *internal* or *external* Lorentz indices of the form `nu1` (internal) or `mu1` (external). The actual integers appearing in these indices are in one-to-one correspondence with the positive or negative integers generated by `QGRAF`, respectively. The Lorentz indices are used to generate the explicit expression for the Feynman rules in the remainder of the code. (In particular, they are just dropped for any scalar or fermionic legs.) *All other (gauge or global) indices need to be generated explicitly by the user in the model file.*

As an aid for the user, the module `docolour` is included in the distribution of `MaRTIn`; it uses equations from Ref. [46] and may be used to handle fields transforming under the fundamental or adjoint representations of an $SU(N)$ algebra. (In particular, this comprises the case of QCD.) The module handles both index generation and algebra, and provides the following functions:

`Dcol(i,j)`	δ^{ij}	Kronecker delta for fundamental indices
`Dadcol(a,b)`	δ^{ab}	Kronecker delta for adjoint indices
`Tcol(i,j,a)`	T^a_{ij}	generator of the fundamental representation
`Fcol(a,b,c)`	f^{abc}	totally antisymmetric structure constants
`Cold(a,b,c)`	d^{abc}	totally symmetric tensor

These tensors are defined such that $\mathrm{Tr}\{T^aT^b\} = \frac{1}{2}\delta^{ab}$ and $\mathrm{Tr}\{T^aT^bT^c\} = \frac{1}{4}(d^{abc} + if^{abc})$.

These functions require external or internal leg numbers as arguments, which will then be substituted in terms of unique indices by `docolour`. For instance, group indices for a quark propagator in the fundamental representation could be generated by the following substitution:

```
id propagator(xx1?,xx2?,?a) = propagator(xx1,xx2,?a)*Dcol(xx1,xx2) ;
```

`docolour` then needs to be called to generate the appropriate indices and subsequently resolve the $SU(N)$ algebra. Fundamental indices are called `i1`, `i2`, etc. for internal legs and `j1`, `j2`, etc. for external legs, and have dimension `NcolF` (= 3 for QCD), while adjoint indices are called `a1`, `a2`, etc. for internal legs and `b1`, `b2`, etc. for external legs, and have dimension `NcolA` (= 8 for QCD). It is recommended to include the `#include docolour` statement only after all group structures are generated – either directly at the end of the INSERTVERTICES fold of the `FORM` model file (see below), or within the FOLD1 fold in the problem file.

Generation of all other indices is left to the user (for instance, $SU(2)$ gauge indices for the SM in the unbroken phase). It is recommended to use the leg numbers to generate all group indices. As an example, consider the assignment of some flavour indices `idx'i'` with dimension `Nf`. This can be done by defining a `set`

```
Symbol Nf;
dimension Nf;
Index idx1,...,idx99;
Set idcs: idx1,...,idx99;
```

which aids in matching indices to leg numbers later. For instance, a propagator could be substituted by the following expression:

```
id propagator(xx1?pos_,xx2?pos_,?a) = d_(idcs[xx1],idcs[xx2])
                                    * propagator(xx1,xx2,?a);
```

External legs have negative indices and thus need to be handled separately.

The appropriate fold to generate group-theory structures is the GROUPTHEORY fold in the `FORM` model file.

```
*--#[ GROUPTHEORY :
*   Possible place to factor out index contractions from propagators and vertices.

*   For instance, use the docolour module.
*   You only have to pass the leg numbers that QGRAF has assigned.

*   Example: insert the SU(n) generator for the quark-gluon vertex.
  id Vqqg(xx1?,xx2?,xx3?,?a) = Vqqg(xx1,xx2,xx3,?a)*Tcol(xx1,xx2,xx3);

*   Note that the index contractions may also be handled in later folds.
*--#] GROUPTHEORY :
```

4.2.3. Generation of fermion lines

`MaRTIn` does not use any explicit spinor indices for fermionic variables, and instead uses its own as well as `FORM`'s built-in expressions for Dirac matrices. This entails that all fermionic variables (spinor functions and Dirac matrices) have to be treated as non-commuting objects, and they have to be arranged in the correct order. We will briefly describe the algorithm here, so the user can implement their own fermionic vertices.

`QGRAF` requires that all fermionic vertices are bilinear in the fermion fields. Therefore, fermionic vertices and propagators can be used to construct `FL'i'` objects representing open and closed fermion lines, enumerated by an integer `'i'`. These objects represent products of Dirac matrices arising from both fermion propagators and vertices in the correct order. After `gengraph` has collected all fermion propagators and vertices into the fermion-line objects, the corresponding propagator and vertex functions no longer appear in the amplitude. Instead, each propagator is represented as an argument inside `FL'i'`, of the form

```
FL1(?a,prop,mom,p,mass,m,?b)
```

Here, the separator `prop` indicates the presence of a propagator factor; p is any linear combination of loop and external momenta flowing through the propagator, and m is its mass. Each vertex is identified by a symbol representing the vertex type, followed by the arguments of the corresponding vertex function. If `vertex` is the symbol associated with the vertex function `VERTEX`, this vertex would be represented as follows:

```
   FL1(?a,vertex,lorentz,v_{i_1},...,v_{i_N},mom,p_1,...,p_N,?b) * vertex
```

Note that the symbol `vertex` is used both as a label inside the fermion line object (and will later be used by FORM's pattern matcher to substitute the vertex by the appropriate Dirac structure), and as a symbol multiplying the term (this instance of the symbol can later be replaced by some value for the corresponding Feynman rule, e.g., for the $\bar{e}\not{A}e$ vertex in QED this could be `vAee` $\to ieQ_e$).

For the algorithm to work properly, all fermionic vertices must be defined in the QGRAF model file with the antifermion field as the first entry and the fermion field as the second entry. Moreover, there must be a one-to-one correspondence between fermionic vertex functions and the associated symbols. This is achieved by providing a list of all fermionic vertices in the FF fold of the FORM model file, and a list of all corresponding vertex symbols in the VERTICES fold, *in exactly the same order*. This could look as follows:

```
*--#[ FF :
*   In order to generate spinor contractions, the vfunct of each vertex with
*   fermions have to be listed here, comma separated, e.g.:
  Vqqg, Vqqs,
*--#] FF :

*--#[ VERTICES :
*   For each vertex from the fold FF above, a symbol representing the coupling
*   at this vertex has to be provided. The order has to match.
*   MaRTIn will match these symbols to substitute the Feynman rules.
  vqqg, vqqs,
*--#] VERTICES :
```

where, in the example above, `Vqqg` and `Vqqs` are declared as commuting functions and `vqqg` and `vqqs` as symbols. Note that the trailing "`,`" is required. If any vertex is not included in these lists, the code will exit with an error message. If the ordering of functions and symbols is not in exact correspondence, the Feynman rules will be inserted in a wrong way. The actual substitution of the vertices by the Feynman rules is discussed in Sec. 4.2.5.

4.2.4. Substitution of propagators

After the discussion of the fermion line objects, we next describe the substitution of propagators. In general, the internal function `Deno`$(p, M) = 1/(p^2 - M^2)$ is employed to represent the propagator denominators. The *numerators* $(\not{p} + m)$ of all fermionic propagators are substituted automatically, in the correct order, by `gengraph` using the fermion-line objects. For this to work properly, a substitution of the following form is required in the INSERTPROPAGATORS fold. In the example below, `Q` is the quark-propagator function eventually corresponding to $i\delta^{ab}(\not{p} + m)/(p^2 - m^2)$:

```
*--#[ INSERTPROPAGATORS :
*   Substitution of a quark propagator:
  id Q(xx1?,xx2?,lorentz,?x,mom,?p,mass,M?,field,fname?)
     = i_*F(xx1,xx2,mom,?p,mass,M,field,fname)*Deno(?p,M)*Dcol(xx1,xx2);

*   The internal function, F, must be used to properly account
*   for the γ-matrices and spinor structure of the numerator.
*   Note the use of the predefined commuting function Deno(p,M) = 1/(p^2 - M^2).
*   Here, the colour factor has been supplied together with the denominator.
*--#] INSERTPROPAGATORS :
```

The use of the internal `MaRTIn` function, `F`, for the numerator part of any fermionic propagator is essential for `gengraph` to work.

It remains to substitute all the bosonic propagators using the INSERTPROPAGATORS fold:

```
*--#[ INSERTPROPAGATORS :
*   For instance, the substitution of a gluon propagator in R_ξ gauge:
  id Dg(xx1?,xx2?,lorentz,nu1?,nu2?,mom,p?,mass,M?,field,g)
     = -i_*(d_(nu1,nu2) - p(nu1)*p(nu2)*Deno(p,M)*[1-xi])
        *Deno(p,M)*Dadcol(xx1,xx2);

*   [1-xi] is a symbol representing (one minus) the gauge-fixing parameter, i.e., 1-ξ.
*--#] INSERTPROPAGATORS :
```

4.2.5. Substitution of fermionic vertices

The substitution of fermionic vertices appearing in the `FL'i'` objects can be done in two different ways. A large number of frequently occurring vertices has been implemented in a generic way; these vertices can be substituted by simply populating the corresponding folds in the FORM model file with the vertex symbols of the VERTICES fold. The following generic cases are implemented:

Vector coupling: For all vertex symbols `vv`, provided as a comma-separated list in the VVCOUP fold, the Feynman rule γ^μ is substituted.

```
*--#[ VVCOUP :
*   For each vertex represented by the symbol listed here,
*   a vector-fermion-fermion will be susbtituted (e.g.: vv↦vvγ^μ.)
  vv, ...
*--#] VVCOUP :
```

Note that after the substitution, the vertex is multiplied by the corresponding vertex symbol. This symbol can then be used to insert coupling constants or symmetry factors, see Sec. 4.2.8.

Chiral vector couplings: For all vertex symbols `va`, `vl`, `vr`, provided as three comma-separated lists in the VACOUPV, VACOUPL, and VACOUPR folds, respectively, a Feynman rule is substituted as follows:

```
*   VACOUPV contains a list of vertex symbols, e.g. va, which must
*   also be contained in the VERTICES fold.
*   VACOUPL and VACOUPR contain corresponding lists of symbols,
*   vvl and vvr, for each VACOUPV symbol.
```

```
*   Thus, the number and order of elements must match in VACOUPV, VACOUPL, VACOUPR.
*   va gets replaced by vl, vr : va ↦ γ^μ (vl P_L + vr P_R).

*--#[ VACOUPV :
  va,
*--#] VACOUPV :
*--#[ VACOUPL :
  vl,
*--#] VACOUPL :
*--#[ VACOUPR :
  vr,
*--#] VACOUPR :

*   These three folds belong together and should have the same number of entries.
```

Here, $P_L = \frac{1}{2}(1-\gamma_5)$ and $P_R = \frac{1}{2}(1+\gamma_5)$ denote the projectors onto the left- and right-handed chiral states. Again, the symbols `vl` and `vr` can be replaced later.

Scalar couplings: For all vertex symbols `vs`, provided as a comma-separated list in the SCOUPV fold, a unit matrix in spinor space $\mathbb{1}$ is substituted.

```
*--#[ SCOUPV :
*   Generic scalar-fermion-fermion operator, a unit matrix is substituted.
*   Same syntax as in the examples above: sv ↦ sv 𝟙
vs,
*--#] SCOUPV :
```

Chiral scalar couplings: For all vertex symbols `sav`, `slv`, `srv`, provided as three comma-separated lists in the SACOUPV, SACOUPLV, and SACOUPRV folds, respectively, a Feynman rule is substituted as follows:

```
*   SACOUPV contains a list of vertex symbols; e.g. sav
*   SACOUPLV and SACOUPRV contain corresponding lists of symbols,
*   slv and srv, for each SACOUPV symbol.
*   Thus, the number and order of elements must match in SACOUPV, SACOUPLV, SACOUPRV.
*   sav gets replaced by slv, srv : sav ↦ slv P_L + srv P_R.

*--#[ SACOUPV :
  sav,
*--#] SACOUPV :
*--#[ SACOUPLV :
  slv,
*--#] SACOUPLV :
*--#[ SACOUPRV :
  srv,
*--#] SACOUPRV :
```

This concludes the list of implemented fermion vertices. We would like to remind the reader that, while folds may be empty, they should never be removed from the model file.

Alternatively, fermion vertices can by substituted on a case-by-case basis, using the fermion line objects. This has to be done in the INSERTFERMIONVERTICES fold. All substitution rules provided in this fold will be looped over the number of fermion line objects by `gengraph`; the corresponding loop variable (fermion line index) must be denoted as `'i'`. Care should be taken to keep the correct ordering of the spinorial objects. For instance, the vectorial quark-gluon interaction vertex could be implemented as follows:

```
*--#[ INSERTFERMIONVERTICES :
*   Fermionic vertices, represented by fermion line (FL) objects
*   that are enumerated by 'i', can be substituted here.

*   Below is an example for the gluon - quark vertex, with vertex symbol vqqg.
  id FL`i'(?a,vqqg,lorentz,uu1?,mom,v1?,v2?,v3?)
     = FL`i'(?a) * g_(`i',uu1);
*--#] INSERTFERMIONVERTICES :
```

An extended example showing how to implement an effective four-quark vertex is included in Appendix A.

A few comments are in order. All Lorentz indices corresponding to the fermion field indices are automatically deleted by `gengraph`; hence, the remaining Lorentz indices correspond to the fields appearing after the two fermion fields in the definition of the vertex in the `QGRAF` model file. In particular, the antifermion / fermion fields must always appear at the first / second position in the definition of the vertices in the `QGRAF` model files. The preprocessor variable `'i'` enumerating the fermion line is tracked also throughout the spinor matrices in the Feynman rule, which can be `FORM`'s internal non-commuting functions for spinor objects, such as the identity in spinor space (`gi_`), the γ^μ matrix (`g_`), γ_5 (`g5_`). Alternatively, `MaRTIn`'s predefined objects `PL`i'` and `PR`i'` for the chiral projectors and `G5`i'` for the γ_5 matrix can be used.

If there are any remaining fermion line objects left after vertex susbtitution, `MaRTIn` will exit with an error message.

4.2.6. Substitution of bosonic vertices

As for fermionic vertices, there are several generic bosonic vertices implemented that can be substituted in a simple manner via predefined folds. They are listed in the following. For each of the implemented cases, the first fold should contain the `vfunct` vertex function corresponding to a given vertex, and the other folds should contain corresponding user-defined symbols, as needed. These symbols need to be declared in the `FORM` model file (see Sec. 4.2.8). Again, we would like to remind the reader that, while folds may be empty, they should never be removed from the model file. Folds corresponding to a given vertex class need to be populated with lists of functions or symbols of the same length, and in one-to-one correspondence to each other.

Vector-vector interaction: This vertex can be used, for instance, to implement gauge-boson propagator counterterm vertices. Here, q is the incoming momentum of the first vector field.

```
*   Generic vector-vector vertex: Vv ↦ vv1 q^2 g^{μν} + vv2 g^{μν} + vv3 q^μ q^ν .

*--#[ VVgenCOUP :
  Vv,
*--#] VVgenCOUP :
*--#[ vVVgenCOUP1 :
  vv1,
*--#] vVVgenCOUP1 :
*--#[ vVVgenCOUP2 :
  vv2,
*--#] vVVgenCOUP2 :
*--#[ vVVgenCOUP3 :
  vv3,
*--#] vVVgenCOUP3 :
```

Scalar-scalar interaction: This vertex can be used, for instance, to implement scalar propagator counterterm vertices. Here, q is the incoming momentum of the first scalar field.

```
*   Generic scalar-scalar vertex: Ss ↦ ss1 q^2 + ss2.

*--#[ SSgenCOUP :
  Ss
*--#] SSgenCOUP :
*--#[ vSSgenCOUP1 :
  ss1,
*--#] vSSgenCOUP1 :
*--#[ vSSgenCOUP2 :
  ss2,
*--#] vSSgenCOUP2 :
```

Vector-scalar interaction: This vertex can be used, for instance, to implement vector-scalar mixing. Here, q is the incoming momentum of the vector field. Note that the vector field has to precede the scalar field in the definition of the vertex in the `QGRAF` model file.

```
*   Generic vector-scalar (with momentum q) vertex: Vs ↦ vs1 q^μ .

*--#[ VSgenCOUP :
  Vs,
*--#] VSgenCOUP :
*--#[ vVSgenCOUP1 :
  vs1,
*--#] vVSgenCOUP1 :
```

Triple vector interaction: Here, q_1, q_2, q_3 are the incoming momenta of the first, second, and third vector field, respectively, while μ_1, μ_2, μ_3 are their Lorentz indices (ordering as in the `QGRAF` model file). Here and below, $g^{\mu\nu}$ denotes the metric tensor of Minkowski space, and ϵ_{ijk} is the totally antisymmetric Levi-Civita tensor in three dimension with $\epsilon_{123} = 1$.

```
*   Generic triple vector vertex: Vvv ↦ vvv ε_{ijk} g^{μ_i μ_j} (q_j)^{μ_k} .

*--#[ VVVgenCOUP :
  Vvv,
*--#] VVVgenCOUP :

*--#[ vVVVgenCOUP :
  vvv,
*--#] vVVVgenCOUP :
```

Scalar-vector-vector interaction: Here, μ, ν denote the Lorentz indices of the two vector fields. The scalar field must be at the first position in the `QGRAF` vertex definition.

```
*   Generic scalar-vector-vector vertex: Svv ↦ svv g^{μν} .

*--#[ SVVgenCOUP :
  Svv,
*--#] SVVgenCOUP :
*--#[ vSVVgenCOUP :
  svv,
*--#] vSVVgenCOUP :
```

Triple scalar interaction:

```
*   Generic triple scalar vertex: Sss ↦ sss.

*--#[ SSSgenCOUP :
  Sss,
*--#] SSSgenCOUP :
*--#[ vSSSgenCOUP :
  sss,
*--#] vSSSgenCOUP :
```

Quartic scalar interaction:

```
*   Generic quartic scalar vertex: Ssss ↦ ssss.

*--#[ SSSSgenCOUP :
  Ssss,
*--#] SSSSgenCOUP :
```

```
*--#[ vSSSSgenCOUP :
  ssss,
*--#] vSSSSgenCOUP :
```

Vector-vector-scalar-scalar interaction: Here, μ, ν denote the Lorentz indices of the two vector fields. The two vector fields must be at the first and second positions in the QGRAF vertex definition.

```
*   Generic vector-vector-scalar-scalar vertex: Vvss ↦ vvss g^μν .

*--#[ VVSSgenCOUP :
  Vvss,
*--#] VVSSgenCOUP :
*--#[ vVVSSgenCOUP :
  vvss,
*--#] vVVSSgenCOUP :
```

Quartic vector interaction: Here, $\mu_1, \ldots, \mu_4$ denote the Lorentz indices of the first, second, third, fourth vector field as defined in the QGRAF model file.

```
*   Generic quartic vector vertex: Vvvv ↦ vvvv(2 g^μ1μ2 g^μ3μ4 − g^μ1μ3 g^μ2μ4 − g^μ1μ4 g^μ2μ3).

*--#[ VVVVgenCOUP :
  Vvvv,
*--#] VVVVgenCOUP :
*--#[ vVVVVgenCOUP :
  vvvv,
*--#] vVVVVgenCOUP :
```

Vector-scalar-scalar interaction: Here, μ_1 is the Lorentz index of the vector field (first field in the QGRAF model file), while q_2, q_3 are the incoming momenta of the first and second scalar field (second and third fields in the QGRAF model file).

```
*   Generic vector-scalar-scalar vertex: Vss ↦ vss(q2 − q3)^μ1 .

*--#[ VSSgenCOUP :
  Vss,
*--#] VSSgenCOUP :
*--#[ vVSSgenCOUP :
  vss,
*--#] vVSSgenCOUP :
```

Vector-ghost-ghost interaction: Here, μ_1 is the Lorentz index of the vector field (first field in the QGRAF model file), while q_2 is the incoming momentum of the first ghost field (second field in the QGRAF model file).

```
*   Generic vector-ghost-ghost vertex: Vhh ↦ vhh(q2)^μ1 .

*--#[ VGGgenCOUP :
  Vhh,
*--#] VGGgenCOUP :
*--#[ vVGGgenCOUP :
  vhh,
*--#] vVGGgenCOUP :
```

Alternatively, any bosonic vertices may be substituted on a case-by-case basis in the INSERTVERTICES fold. This may be required if other Lorentz structures appear in the Feynman rules, for instance, when implementing Feynman rules in a background field gauge, or in an effective field theory. The corresponding substitutions can be constructed from the arguments of the vertex functions, discussed in Sec. 4.2.1. An example is given in the following.

```
*--#[ INSERTVERTICES :
*   Use arguments of vertex functions to substitute Feynman rules.
*   The order of arguments correspond to the order
*   of how fields were defined in QGRAF.
*   For convenience, the arguments are separated by keywords lorentz and mom.

*   Consider for instance the three-gluon-vertex:
  id Vggg(?a,lorentz,nu1?,nu2?,nu3?,mom,v1?,v2?,v3?) =
    v3g * (
      +d_(nu1,nu3)*(v3(nu2)-v1(nu2))
      +d_(nu2,nu3)*(v2(nu1)-v3(nu1))
      +d_(nu1,nu2)*(v1(nu3)-v2(nu3))
    );
*   It is assumed that the colour structure was generated before.
*   v3g is a symbol representing the coupling constant.
*--#] INSERTVERTICES :
```

4.2.7. Substitution of external leg factors

Finally, external leg factors are substituted in the POLARIZATION fold. In addition to the polarization function discussed in Sec. 4.2.1, MaRTIn generates a non-commuting

```
dummyspinor(fermionlineindex, fieldindex, dualfieldindex)
```

function for each external fermion. An example of how these objects can be translated into meaningful external leg factors, i.e., the external spinor functions $u(p)$ and $v(p)$, is found in `model_SM`. A simple substitution rule for external photons could look like

```
*--#[ POLARIZATION :
*   Handle substitutions of polarization / dummyspinor functions here.
*   Below is example for a scalar field.
*   First we mark external leg numbers with nneg():

  id pol(xx1, xx2?neg_, q1?) = pol(xx1, nneg(-xx2), q1);

*   which can be used to build polarisation vectors, e.g. for photons (field symbol "a")

  id pol(a, nneg(xx2?odd_), q1?) = ee(a, mom, q1, lorentz, mmuu[xx2]);
  id pol(a, nneg(xx2?even_), q1?) = eestar(a, mom, q1, lorentz, mmuu[xx2]);

*   distinguishing incoming and outgoing photon legs.
*   Here, "mmuu" is the built-in set of external Lorentz indices.
*--#] POLARIZATION :
```

External fermions, as well as particles with additional quantum numbers, can be handled in the same way; see the supplied `model_SM` for examples. It is highly recommended to generate any spinor functions only in the POLARIZATION fold, to avoid interference with the internal routines that handle the Dirac algebra.

4.2.8. Other folds

Each `FORM` model file contains three `FORM` folds in addition to those discussed above.

The DEF fold contains all `FORM` definitions necessary for the implementation of the model. This includes all fields (as symbols), propagator and vertex functions (commuting functions) referenced in the `QGRAF` model file, but also anything else the user desires. Note that many variables are already defined in `prc/maindeclare`.

```
*--#[ DEF :
*   This fold contains all necessary declarations for the model file.

*   Here is a good place to define the integer spacetime dimension.
*   MaRTIn has been most tested for the case of "4" and less so for "3".
  #define DIMENSION "4"

*   Declare all your symbols, functions, indices and sets here.
*   Several variables are already defined in prc/maindeclare.
*   Fields and masses from the QGRAF model should be declared as symbols.
*   All pfunct and vfunct should be declared as commuting functions.

*--#] DEF :
```

The AMASSES fold defines a `FORM` set that fixes the hierarchy of masses in a given model, as well as a dollar variable set to the total number of masses. This is used to reduce the number of terms produced in some two-loop integrals when there are more than three masses present.

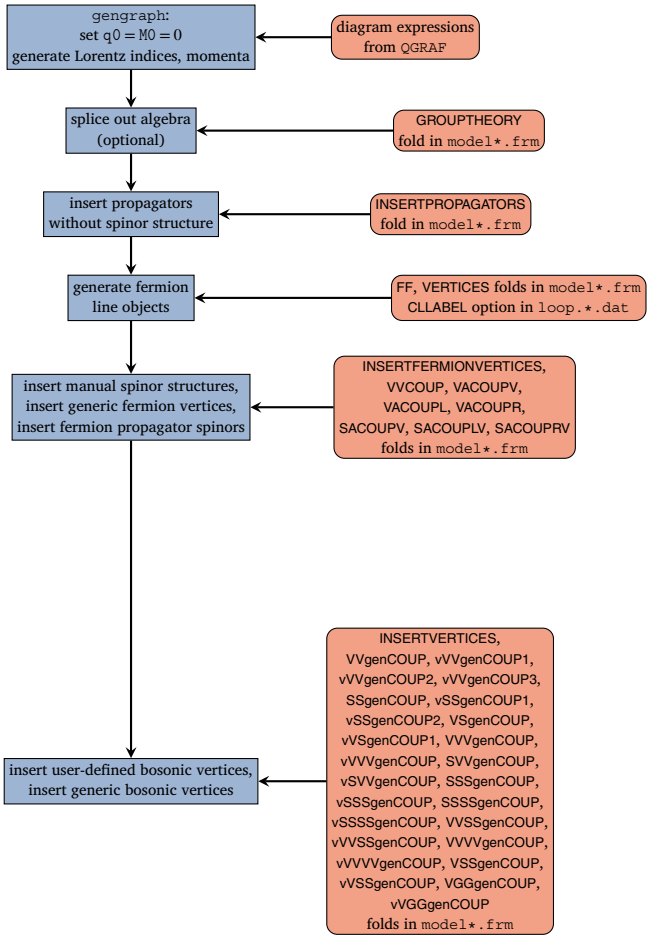
```
*--#[ AMASSES :
*   In this fold we need to define the hierarchy of masses.
*   This information is required by some two-loop master integrals.

*   Fill this set with ordered masses. For M1 < M2 < M3 < M4:
  set massorder:  M1, M2, M3, M4;

*   The length of massorder needs to be stored in a dollar variable.
  #$massnum = 4;

*--#] AMASSES :
```

Finally, vertex symbols may be replaced with actual couplings in the fold INSERTCOUPLINGS.

```
*--#[ INSERTCOUPLINGS :
*   At the very end of the calculation, insert the actual couplings.
*   Renormalisation may be implemented here.
  id vgqq = g + dg1/ep;

*--#] INSERTCOUPLINGS :
```

4.2.9. Summary

Most of the information contained in the `FORM` model file feeds into the internal module `gengraph`, which handles the insertion of Feynman rules, see Sec. 3.2. We close this section by giving a graphical overview of `gengraph`'s workflow, as the order of inclusion for each fold may be relevant to the user.



4.3. Add-ons for MaRTIn

Additional information on the models might be required for add-ons to MaRTIn.

Currently, this is only the case for richard_draw. This piece of software requires a model file models/rdraw/*qmodel*.json of the working directory, where *qmodel* has to match the name of the QGRAF files. This JSON formatted file contains the following style information:

```
{
  "fields": {
    "QGRAF name": ["TEX name", "linetype"],
    ...
  }
}
```

where "*QGRAF name*" has to match the field identifier in the QGRAF, "*TEX name*" contains valid LaTeX code, and "*linetype*" controls the appearance of the propagator lines. The default options for the latter include "fermion", "anti fermion", "scalar", "charged scalar", "anti charged scalar", "boson", "gluon" and "ghost". An example file is provided by user_template/models/rdraw/standardmodel.json in the source directory. MaRTIn may execute richard_draw via the rpdf target, e.g. the command

```
martin problems/SM/loop.1_uu.dat rpdf
```

will generate the file displayed in Fig. 1.

5. Example

In this section we briefly discuss the calculation of the divergent part of the QCD quark self energy, in order to illustrate the usage of MaRTIn. This example uses the following files that are distributed together with the code: The Standard Model FORM-model file model_SM, located in user_template/models/form, the Standard Model qgraf-model files standardmodel.prop.lag and standardmodel.vrtx.lag, located in user_template/models/qgraf, and the problem file loop.1_uu.dat located in user_template/problems/SM. The latter is calling the dolour procedure from the base code to simplify the colour structure. This example should give a good illustration of how to read the output of MaRTIn.

The corresponding call of MaRTIn, e.g., from the user_template directory, would be:

```
martin problems/SM/loop.1_uu.dat
```

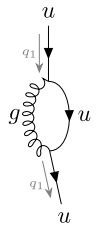
which would generate and compute all diagrams. In this example there is a single diagram, see Fig. 1.

This will result in a screen printout, which includes the following lines:

Diagrams (total # 1)

Richard_Draw.py

```
-----------------------------------------------------
 output = 'rgraphs.1_uu.dot';
 style  = 'richard_draw.sty';
 model  = 'model.lag';
 in =  fu[q1];
 out = fu[q1];
 loops = 1 ;
 loop_momentum = p;
 options =;
 true = bridge[a, g, z, G0, h, 0, 0] ;
 true = iprop[g, 1, 1] ;
-----------------------------------------------------
```



1
(symmetry: +1)

Fig. 1. Example of `richard_draw` output for the problem `SM/loop.1_uu.dat`.

```
------------------------------------------------------------------------
Massive Recursive Tensor Integration

            M a R T I n

Copyright (C) 2009-2023 J. Brod, L. Huedepohl, E. Stamou, T. Steudtner
MaRTIn is licensed under the GNU General Public License Version 3.
------------------------------------------------------------------------

Extracted default configuration from config file: "~/.martin.conf"

Computing using the input of the loop.dat:
   xxx/user_template/problems/SM/loop.1_uu.dat

The problem suffix of the loop.dat is:
   suffix = ".1_uu"

The problem directory is:
   xxx/user_template/problems/SM

The results directory is:
   [inferred from loop.dat] : xxx/user_template/results/SM

The user_prc directories are (in order of importance):
   [inferred from loop.dat] : xxx/user_template/prc

The FORM models are:
   [inferred from loop.dat] : xxx/user_template/models/form/model_SM

The QGRAF models are:
   [inferred from loop.dat] : xxx/user_template/models/qgraf/sm.prop.lag
   [inferred from loop.dat] : xxx/user_template/models/qgraf/sm.vrtx.lag

The JSON files for richard_draw.py are:
   [inferred from loop.dat] : xxx/user_template/models/rdraw/standardmodel.json
```

```
Computing based on GIT commit: <hash> (Modulo local/uncommited modifications)
```

These lines just explicitly list the files used by MaRTIn in that particular run. (In practice, xxx will correspond to the base directory that is actually used.) The git commit version is also indicated. This is followed by information on the generation of the diagrams:

```
Running "make"...

Generating xxx/user_template/results/SM/qgraf.1_uu.dat ...
Generating xxx/user_template/results/SM/qlist.1_uu.dat ...
Running QGRAF ...

  ------------------------------------------------------
                       qgraf-3.1.4
  ------------------------------------------------------

 output = 'qlist.1_uu.dat';
 style  = 'main.sty';
 model  = 'model.lag';
 in =  fu[q1];
 out = fu[q1];
 loops = 1 ;
 loop_momentum = p;
 options =;
 true = bridge[a, g, z, G0, h, 0, 0] ;
 true = iprop[g, 1, 1] ;

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

   24P  ---  7+  17-  ---  5N+  2C+  17C-

   163V  ---  3^136  4^27

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

    -   4^1  ---  0 diagrams
   3^2   -   ---  1 diagram

   total =  1 diagram

Generating xxx/user_template/results/SM/make.1_uu.info ...
rm xxx/user_template/results/SM/qlist.1_uu.dat
Generating xxx/user_template/results/SM/qgraf.1_uu.dat ...
Generating xxx/user_template/results/SM/form.1_uu.dat ...
```

Finally, FORM is invoked to perform the actual calculation:

```
Computing xxx/user_template/results/SM/form.1_uu/dia1.sav ...
FORM 4.2.1 (Feb  6 2019, v4.2.1-3-g558b01f) 64-bits  Run: <date and time>
    #-

   dia1 =

    + ep^-1 * (
       - 3/4*UbarSp(fu,su3col,j1,mom,q1)*DIRAC(1,one)*USp(fu,su3col,j1,mom,
      q1)*i_*pi^-1*alphas*Mup*CF
       - 1/4*UbarSp(fu,su3col,j1,mom,q1)*DIRAC(1,one)*USp(fu,su3col,j1,mom,
      q1)*i_*pi^-1*xiqg*alphas*Mup*CF
       + 1/4*UbarSp(fu,su3col,j1,mom,q1)*DIRAC(1,q1)*USp(fu,su3col,j1,mom,
      q1)*i_*pi^-1*xiqg*alphas*CF
       );

  0.10 sec out of 0.10 sec
Done computing xxx/user_template/results/SM/form.1_uu/dia1.sav.

MaRTIn finished.
```

The following code in the FORM folds in the problem file has been used to simplify the output

```
*--#[ USERDEF :
s CF,alphas;
*--#] USERDEF :

*--#[ FOLD1 :
#include docolour
*--#] FOLD1 :

*--#[ FOLD5 :
id eM(mIRA) = 1;
id nc = 2*CF + 1/nc;
id gs^2 = 4*pi*alphas;
*--#] FOLD5 :
```

This is used to replace eM(mIRA) by 1 and gs by $\sqrt{4\pi\alpha_s}$, as well as powers of N_c in terms of $C_F \equiv (N_c^2-1)/2N_c$. Note that the spinorial objects UbarSp, DIRAC, and USp are non-commuting. The final output then corresponds to the familiar expression

$$\texttt{dia1} = -\frac{i\alpha_s}{4\pi} C_F \frac{1}{\epsilon} \bar{u}(\boldsymbol{q}_1, j_1)\left[\xi_G \not{q}_1 + m_u(3+\xi_G)\right] u(\boldsymbol{q}_1, j_1)\,. \tag{4}$$

The result is saved as `dia1.sav` in the directory `user_template/results/SM/form.1_uu` and can be loaded using FORM's `load` command. A Mathematica-readable (text-)file `dia1.m` is saved in the directory `user_template/results/SM/math.1_uu`. Its contents are

```
(* dia1 *)      { - 3/4*MathDirac[UbarSp[fu,su3col,j1,mom,q1],USp[fu,su3col,
      j1,mom,q1]]*I*Pi^(-1)*ep^(-1)*alphas*Mup*CF - 1/4*MathDirac[
      UbarSp[fu,su3col,j1,mom,q1],USp[fu,su3col,j1,mom,q1]]*I*Pi^(-1)*
      ep^(-1)*xiqg*alphas*Mup*CF + 1/4*MathDirac[UbarSp[fu,su3col,j1,
      mom,q1],q1,USp[fu,su3col,j1,mom,q1]]*I*Pi^(-1)*ep^(-1)*xiqg*
      alphas*CF}
```

CRediT authorship contribution statement

Joachim Brod: Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Software, Resources, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Data curation, Conceptualization. **Lorenz Hüdepohl:** Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Software, Resources, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Data curation, Conceptualization. **Emmanuel Stamou:** Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Software, Resources, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Data curation, Conceptualization. **Tom Steudtner:** Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Software, Resources, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Data curation, Conceptualization.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Joachim Brod reports financial support was provided by US Department of Energy. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgements

We would like to thank Martin Gorbahn for many cross checks of our code during various fruitful collaborations. J.B. acknowledges support in part by DOE grant DE-SC1019775.

Appendix A. Implementation of four-fermion vertices

QGRAF does not allow for the generation of local four-fermion vertices. They can, however, be implemented in a straightforward way using auxiliary propagators. The following example shows how to implement the four-quark operator $Q = (\bar{s}_L \gamma^\mu c_L)(\bar{u}_L \gamma_\mu d_L)$. Here, $q_L \equiv \frac{1}{2}(1-\gamma_5)q$ denote the left-handed quark fields ($q = u, d, s, c$).

The QGRAF model file should include

```
  *   Artificial propagator:
  [sp,sm,+;pfunct='Dmu',m='M0']

  *   Vertices for the two quark bilinears:
  [fS,fc,sp;vfunct='FSc']
  [fU,fd,sm;vfunct='FUd']
```

We assume that all functions and symbols have been properly declared, and colour indices have been generated. The vertex can then be substituted in two steps. If vertex symbols `vFSc` and `vFUd` (together with the corresponding vertex functions `FSc` and `FUd`) have been defined and included in the FF and VERTICES folds, we include the following substitution rule in the INSERTFERMIONVERTICES fold:

```
*--#[ INSERTFERMIONVERTICES :
*

id FL`i'(?a,vFUd,lorentz,mu1?,mom,q1?,q2?,q3?)
   = FL`i'(?a)*g_(`i',mu1)*PL`i';

id FL`i'(?a,vFSc,lorentz,mu1?,mom,q1?,q2?,q3?)
   = FL`i'(?a)*g_(`i',mu1)*PL`i';

*
*--#] INSERTFERMIONVERTICES :
```

Here, we made use of the unique Lorentz indices that have been generated automatically for the artificial "propagator field". (This trick does not work for operators with longer strings of Dirac matrices, in which case the Lorentz indices have to be inserted/generated "by hand".) Finally, we use the substitution of the artificial propagator to contract the correct pairs of Lorentz indices, by simply replacing it by a metric tensor:

```
*--#[ INSERTPROPAGATORS :
*

id Dmu(xx1?,xx2?,lorentz,mu1?,mu2?,mom,?p,mass,M?,field,fname?)
   = d_(mu1,mu2) ;

*
*--#] INSERTPROPAGATORS :
```

Note that `gengraph` will automatically supply a factor `vFSc*vFUd`, which can then be substituted by the Wilson coefficient of the operator. For instance, if the Lagrangian in the current example is $\mathcal{L} = \sqrt{2} G_F C Q$, one would include the following substitution rule in the INSERTCOUPLINGS fold:

```
*--#[ INSERTCOUPLINGS :

  id vFSc*vFUd = i_*sqrt_(2)*GF*C ;

*--#] INSERTCOUPLINGS :
```

References

[1] P. Nogueira, Automatic Feynman graph generation, J. Comput. Phys. 105 (1993) 279, https://doi.org/10.1006/jcph.1993.1074.
[2] T. Hahn, M. Perez-Victoria, Automatized one loop calculations in four-dimensions and D-dimensions, Comput. Phys. Commun. 118 (1999) 153, https://doi.org/10.1016/S0010-4655(98)00173-8, arXiv:hep-ph/9807565.
[3] T. Hahn, Generating Feynman diagrams and amplitudes with FeynArts 3, Comput. Phys. Commun. 140 (2001) 418, https://doi.org/10.1016/S0010-4655(01)00290-9, arXiv:hep-ph/0012260.
[4] J.A.M. Vermaseren, New features of FORM, arXiv:math-ph/0010025, 2000.
[5] J. Kuipers, T. Ueda, J.A.M. Vermaseren, J. Vollinga, FORM version 4.0, Comput. Phys. Commun. 184 (2013) 1453, https://doi.org/10.1016/j.cpc.2012.12.028, arXiv:1203.6543.
[6] R. Mertig, M. Bohm, A. Denner, FEYN CALC: computer algebraic calculation of Feynman amplitudes, Comput. Phys. Commun. 64 (1991) 345, https://doi.org/10.1016/0010-4655(91)90130-D.
[7] M. Tentyukov, J. Fleischer, A Feynman diagram analyzer DIANA, Comput. Phys. Commun. 132 (2000) 124, https://doi.org/10.1016/S0010-4655(00)00147-8, arXiv:hep-ph/9904258.
[8] R. Harlander, T. Seidensticker, M. Steinhauser, Complete corrections of Order alpha alpha-s to the decay of the Z boson into bottom quarks, Phys. Lett. B 426 (1998) 125, https://doi.org/10.1016/S0370-2693(98)00220-2, arXiv:hep-ph/9712228.
[9] T. Seidensticker, Automatic application of successive asymptotic expansions of Feynman diagrams, in: 6th International Workshop on New Computing Techniques in Physics Research: Software Engineering, Artificial Intelligence Neural Nets, Genetic Algorithms, Symbolic Algebra, Automatic Calculation, 1999, arXiv:hep-ph/9905298.
[10] F. Herren, Precision Calculations for Higgs Boson Physics at the LHC - Four-Loop Corrections to Gluon-Fusion Processes and Higgs Boson Pair-Production at NNLO, Ph.D. thesis, KIT, Karlsruhe, 2020.
[11] M. Gerlach, F. Herren, M. Lang, tapir: a tool for topologies, amplitudes, partial fraction decomposition and input for reductions, Comput. Phys. Commun. 282 (2023) 108544, https://doi.org/10.1016/j.cpc.2022.108544, arXiv:2201.05618.
[12] J. Hoff, The Mathematica package TopoID and its application to the Higgs boson production cross section, J. Phys. Conf. Ser. 762 (1) (2016) 012061, https://doi.org/10.1088/1742-6596/762/1/012061, arXiv:1607.04465.
[13] V. Margerya, Alibrary, https://github.com/magv/alibrary, 2021.
[14] V. Margerya, Feynson, https://github.com/magv/feynson, 2020.
[15] G. Passarino, M.J.G. Veltman, One loop corrections for e+ e- annihilation into mu+ mu- in the Weinberg model, Nucl. Phys. B 160 (1979) 151, https://doi.org/10.1016/0550-3213(79)90234-7.
[16] R.N. Lee, Presenting LiteRed: a tool for the Loop InTEgrals REDuction, arXiv:1212.2685, 2012.
[17] R.N. Lee, LiteRed 1.4: a powerful tool for reduction of multiloop integrals, J. Phys. Conf. Ser. 523 (2014) 012059, https://doi.org/10.1088/1742-6596/523/1/012059, arXiv:1310.1145.
[18] A.V. Smirnov, Algorithm FIRE – Feynman Integral REduction, J. High Energy Phys. 10 (2008) 107, https://doi.org/10.1088/1126-6708/2008/10/107, arXiv:0807.3243.
[19] A.V. Smirnov, V.A. Smirnov, FIRE4, LiteRed and accompanying tools to solve integration by parts relations, Comput. Phys. Commun. 184 (2013) 2820, https://doi.org/10.1016/j.cpc.2013.06.016, arXiv:1302.5885.
[20] A.V. Smirnov, FIRE5: a C++ implementation of Feynman Integral REduction, Comput. Phys. Commun. 189 (2015) 182, https://doi.org/10.1016/j.cpc.2014.11.024, arXiv:1408.2372.
[21] A.V. Smirnov, F.S. Chuharev, FIRE6: Feynman Integral REduction with modular arithmetic, Comput. Phys. Commun. 247 (2020) 106877, https://doi.org/10.1016/j.cpc.2019.106877, arXiv:1901.07808.
[22] C. Studerus, Reduze-Feynman Integral Reduction in C++, Comput. Phys. Commun. 181 (2010) 1293, https://doi.org/10.1016/j.cpc.2010.03.012, arXiv:0912.2546.
[23] A. von Manteuffel, C. Studerus, Reduze 2 - Distributed Feynman Integral Reduction, arXiv:1201.4330, 2012.
[24] P. Maierhöfer, J. Usovitsch, P. Uwer, Kira—a Feynman integral reduction program, Comput. Phys. Commun. 230 (2018) 99, https://doi.org/10.1016/j.cpc.2018.04.012, arXiv:1705.05610.
[25] J. Klappert, F. Lange, P. Maierhöfer, J. Usovitsch, Integral reduction with Kira 2.0 and finite field methods, Comput. Phys. Commun. 266 (2021) 108024, https://doi.org/10.1016/j.cpc.2021.108024, arXiv:2008.06494.
[26] S.A. Larin, F.V. Tkachov, J.A.M. Vermaseren, The FORM version of MINCER, 1991.
[27] M. Steinhauser, MATAD: a Program package for the computation of MAssive TADpoles, Comput. Phys. Commun. 134 (2001) 335, https://doi.org/10.1016/S0010-4655(00)00204-6, arXiv:hep-ph/0009029.
[28] B. Ruijl, T. Ueda, J.A.M. Vermaseren, Forcer, a FORM program for the parametric reduction of four-loop massless propagator diagrams, Comput. Phys. Commun. 253 (2020) 107198, https://doi.org/10.1016/j.cpc.2020.107198, arXiv:1704.06650.
[29] A. Pikelner, FMFT: Fully Massive Four-loop Tadpoles, Comput. Phys. Commun. 224 (2018) 282, https://doi.org/10.1016/j.cpc.2017.11.017, arXiv:1707.01710.
[30] M. Misiak, M. Munz, Two loop mixing of dimension five flavor changing operators, Phys. Lett. B 344 (1995) 308, https://doi.org/10.1016/0370-2693(94)01553-O, arXiv:hep-ph/9409454.
[31] K.G. Chetyrkin, M. Misiak, M. Munz, Beta functions and anomalous dimensions up to three loops, Nucl. Phys. B 518 (1998) 473, https://doi.org/10.1016/S0550-3213(98)00122-9, arXiv:hep-ph/9711266.
[32] A.I. Davydychev, J.B. Tausk, Tensor reduction of two loop vacuum diagrams and projectors for expanding three point functions, Nucl. Phys. B 465 (1996) 507, https://doi.org/10.1016/0550-3213(96)00033-8, arXiv:hep-ph/9511261.
[33] A.I. Davydychev, J. Tausk, Two loop selfenergy diagrams with different masses and the momentum expansion, Nucl. Phys. B 397 (1993) 123, https://doi.org/10.1016/0550-3213(93)90338-P.
[34] C. Bobeth, M. Misiak, J. Urban, Photonic penguins at two loops and m(t) dependence of BR[B → X(s) lepton+ lepton-], Nucl. Phys. B 574 (2000) 291, https://doi.org/10.1016/S0550-3213(00)00007-9, arXiv:hep-ph/9910220.
[35] Wolfram Research, Inc., Mathematica, Version 13.2, https://www.wolfram.com/mathematica, 2022, Champaign, IL.
[36] Free Software Foundation, GNU general public license, version 3, http://www.gnu.org/licenses/gpl.html, June 29th, 2007.
[37] F. Jegerlehner, Facts of life with gamma(5), Eur. Phys. J. C 18 (2001) 673, https://doi.org/10.1007/s100520100573, arXiv:hep-th/0005255.
[38] J.C. Collins, Renormalization: An Introduction to Renormalization, the Renormalization Group, and the Operator Product Expansion, Cambridge Monographs on Mathematical Physics., vol. 26, Cambridge University Press, Cambridge, 1986, ISBN 978-0-521-31177-9, 978-0-511-86739-2.
[39] K.G. Chetyrkin, M.F. Zoller, Three-loop \beta-functions for top-Yukawa and the Higgs self-interaction in the Standard Model, J. High Energy Phys. 06 (2012) 033, https://doi.org/10.1007/JHEP06(2012)033, arXiv:1205.2892.
[40] A.V. Bednyakov, A.F. Pikelner, V.N. Velizhanin, Yukawa coupling beta-functions in the Standard Model at three loops, Phys. Lett. B 722 (2013) 336, https://doi.org/10.1016/j.physletb.2013.04.038, arXiv:1212.6829.
[41] G. 't Hooft, M.J.G. Veltman, Regularization and renormalization of gauge fields, Nucl. Phys. B 44 (1972) 189, https://doi.org/10.1016/0550-3213(72)90279-9.
[42] P. Breitenlohner, D. Maison, Dimensionally renormalized Green's functions for theories with massless particles. 1, Commun. Math. Phys. 52 (1977) 39, https://doi.org/10.1007/BF01609070.

[43] P. Breitenlohner, D. Maison, Dimensionally renormalized Green's functions for theories with massless particles. 2, Commun. Math. Phys. 52 (1977) 55, https://doi.org/10.1007/BF01609071.
[44] P. Breitenlohner, D. Maison, Dimensional renormalization and the action principle, Commun. Math. Phys. 52 (11) (1977), https://doi.org/10.1007/BF01609069.
[45] S.A. Larin, The renormalization of the axial anomaly in dimensional regularization, Phys. Lett. B 303 (1993) 113, https://doi.org/10.1016/0370-2693(93)90053-K, arXiv:hep-ph/9302240.
[46] T. van Ritbergen, A.N. Schellekens, J.A.M. Vermaseren, Group theory factors for Feynman diagrams, Int. J. Mod. Phys. A 14 (1999) 41, https://doi.org/10.1142/S0217751X99000038, arXiv:hep-ph/9802376.