TTP22-005
P3H-22-009
FERMILAB-PUB-22-025-T

tapir
A tool for topologies, amplitudes, partial fraction decomposition and input for reductions

Marvin Gerlach[a], Florian Herren[b,*], Martin Lang[a]

[a] *Institut für Theoretische Teilchenphysik*
Karlsruhe Institute of Technology (KIT)
Wolfgang-Gaede-Straße 1, D-76128 Karlsruhe, Germany

[b] *Fermilab, PO Box 500, Batavia, Illinois 60510, USA*

Abstract

The demand for precision predictions in the field of high energy physics has dramatically increased over recent years. Experiments conducted at the LHC, as well as precision measurements at the intensity frontier such as Belle II require equally precise theoretical predictions to make full use of the acquired data. To match the experimental precision, two-, three- and, for certain quantities, even higher-loop calculations are required.

To facilitate such calculations, computer software automating as many steps as possible is required. Yet, each calculation poses different challenges and thus, a high level of configurability is required. In this context we present `tapir`: a tool for identification, manipulation and minimization of Feynman integral families. It is designed to integrate in `FORM`-based toolchains which is common practice in the field. `tapir` can be used to reduce the complexity of multi-loop problems with cut-filters, topology mapping, partial fraction decomposition and alike.

Keywords: Feynman integrals, Higher order calculations

*Corresponding author.
E-mail address: florian.s.herren@gmail.com

PROGRAM SUMMARY

Program Title: `tapir`
CPC Library link to program files: (to be added by Technical Editor)
Developer's repository link: gitlab.com/F.Herren/tapir
Code Ocean capsule: (to be added by Technical Editor)
Licensing provisions: GPLv3
Programming language: `python` 3, `C++`
Nature of problem:
Multi-loop computations require the automatization of a large number of different tasks related to Feynman integral topologies. Among them are the identification and minimization of integral topologies, partial fraction decomposition of topologies in the case of linearly dependent propagtors as well as mapping scalar products of loop momenta to scalar functions.
Solution method:
The minimization of topologies is performed by comparison of their respective Nickel indices [1], even further minimization utilizes Pak's algorithm [2]. To efficiently map scalar products of loop momenta to scalar functions `FORM` [3] code is generated.
Additional comments including restrictions and unusual features:
Minimization based on Pak's algorithm slows down for many lines and scales. A coarser minimization using the Nickel indices, however, is still possible.

References

[1] B. Nickel, D. Meiron, G. A. J. Baker, Compilation of 2-pt and 4-pt graphs for continuous spin model, University of Guelph Report (1977).

[2] A. Pak, The Toolbox of modern multi-loop calculations: novel analytic and semi-analytic techniques, J. Phys. Conf. Ser. 368 (2012) 012049. arXiv:1111.0868, doi:10.1088/1742-6596/368/1/012049.

[3] B. Ruijl, T. Ueda, J. Vermaseren, FORM version 4.2 (7 2017). arXiv:1707.06453.

1. Introduction

The evaluation of multi-loop scattering amplitudes is a challenging, yet indispensable, task to obtain precise predictions for scattering processes in high energy physics.

To obtain the scattering amplitude for a particular process all contributing Feynman diagrams are generated based on a set of Feynman rules. Following diagram generation, the numerator structure of each diagram is simplified before the resulting loop integrations are carried out. The potentially large number of diagrams and the nontrivial numerator structures arising in gauge theories lead to a large number of individual Feynman integrals. In general, it is not feasible to directly compute each individual integral. To overcome this issue, it is advantageous to group diagrams with similar underlying Feynman integrals by their topologies. In the context of Feynman diagrams, topologies are also called *integral families*. Identifying a minimal set of topologies is crucial for higher loop calculations as the reduction of Feynman integrals to a set of basis functions, so-called *master integrals*, is a tedious and expensive task in terms of computing time. Thus, duplicating work through a non-minimal set of topologies needs to be avoided.

Individual scalar expressions which appear in a typical l-loop amplitude can be expressed as a linear combination of Feynman integrals from different integral families. Each family is characterized by a set of propagators and irreducible scalar products, $\{D_i\}$, composed of masses, loop momenta as well as external momenta. Each Feynman integral is then specified by the family and a set of *indices* $\{a_i\}$[1]:

$$I(a_1, ..., a_N) = \int ... \int \frac{\mathrm{d}^d k_1 ... \mathrm{d}^d k_l}{(2\pi)^{ld}} \prod_{i=1}^{N} \frac{1}{D_i^{a_i}} \,. \tag{1}$$

In practical calculations, the form of Eq. (1) can always be reached by proper use of projectors or other kinds of tensor reduction. The scalar Feynman integrals can then be reduced to *master integrals* using integration-by-parts relations [1, 2]. Subsequently, the *master integrals* need to be evaluated.

In calculations based on the method of reverse unitarity [3] or in calculations comparing amplitudes in an effective and a full theory, special kinematics may often arise and as a consequence give rise to linearly dependent propagators in Feynman integrals. As no unique set of integration-by-parts relations can be derived for families involving linearly dependent propagators, new families with linearly independent propagators need to be derived by partial fraction decomposition.

Each of the aforementioned steps is a challenge in its own right and requires automatization for all but the most simple processes. As a consequence, a large amount of programs have been developed that automatize one or more of these steps.

[1] In this paper, we always assume dimensional regularization with $d = 4 - 2\epsilon$.

Feynman diagrams can be generated using qgraf [4] or FeynArts [5, 6], while their corresponding symbolic expressions can be simplified using programs such as FeynCalc [7, 8, 9], FormCalc [6] or a plethora of non-public codes relying on computer algebra systems such as Mathematica or FORM [10]. The relevant Feynman rules for a given Lagrangian can be automatically obtained using FeynRules [11, 12] which supports a variety of output formats, such as UFO [13].

In the case of one-loop calculations, the Passarino-Veltman method [14] allows to express any one-loop diagram with propagators that depend quadratically on the momentum as a linear combination of a limited set of basis functions, allowing to completely automate the symbolic computation of one-loop diagrams. This algorithm is implemented in FeynCalc, FormCalc, Package-X [15], HEPMath [16] and the resulting basis functions can be numerically evaluated using libraries such as Collier [17] or LoopTools [5]. Furthermore, programs such as Madgraph [18], Grace [19] or GoSam [20, 21] allow for completely automated computation of one-loop amplitudes.

As such an algorithm is not known at two loops and beyond, dedicated tools are required for the different parts of such a calculation. Generating symbolic expressions for diagrams, identifying and minimizing topologies as well as rewriting propagators and scalar products involving loop momenta in terms of scalar functions as defined in Eq. (1) have been implemented in programs such as DIANA [22], q2e and exp [23, 24, 25], TopoID [26], ALibrary [27], Feynson [28] or LIMIT [29]. Partial fraction decomposition of linearly dependent integrands can be performed by TopoID, APart [30], Multivariate Apart [31] and LIMIT.

Integration-by-parts relations can be solved in a symbolic manner for certain integral families, either by hand or by programs such as LiteRed [32]. Prominent examples are massive tadpole or massless two-point integrals. Both types of integrals have been implemented in the dedicated computer programs MINCER [33], MATAD [34], FORCER [35] and FMFT [36]. For arbitrary families, where no such algorithm is known, programs including FIRE [37, 38], Reduze [39, 40] or KIRA [41, 42] generate systems of linear equations by evaluating integration-by-parts relations for fixed indices.

While a number of programs implementing different parts of multi-loop calculations exist, most of them only implement one specific task. As topology identification and manipulation, partial fraction decomposition and the generation of symbolic expressions all rely on the underlying graphs associated with the Feynman diagrams under consideration, combining these tasks in a single program allows for a unified treatment without interoperability issues that can arise when combining several individual programs.

With this in mind, we introduce tapir: a tool for topologies, amplitudes, partial fraction decomposition and input for reductions. tapir is written in order to replace two of the aforementioned programs and expand on their scope: q2e and LIMIT. q2e is written to generate FORM expressions for Feynman diagrams generated by qgraf and providing information about their graph structure to exp. As a consequence, tapir inherits input formats for Feynman diagrams and Feynman rules from q2e while generating compatible

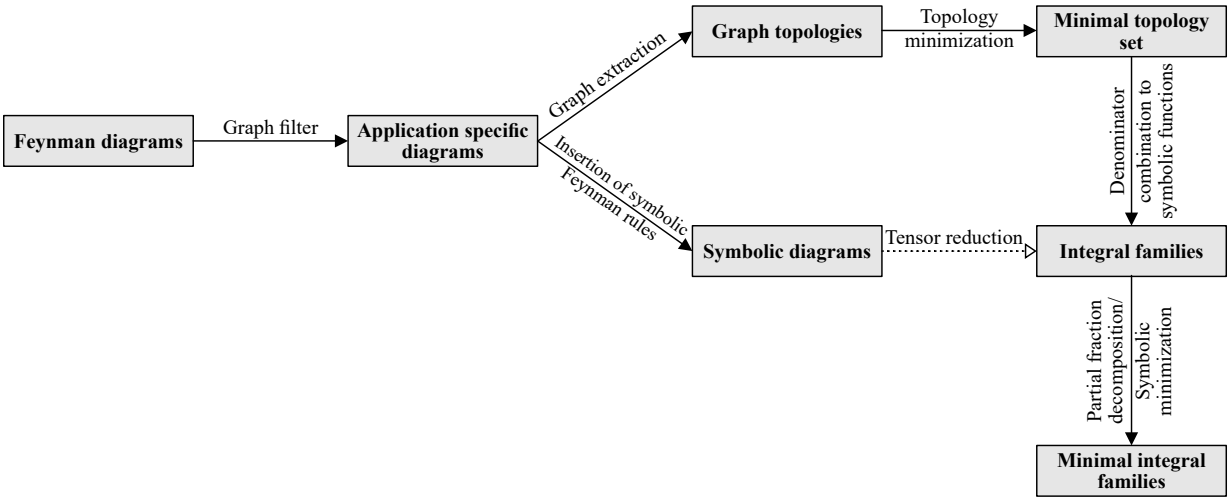


Figure 1: The application of `tapir` is manifold, although the common workflow is oriented on well-established `FORM`-based setups for multi-loop computations. Tensor reduction is not part of `tapir` and must be applied externally.

output files. In contrast to `q2e`, `LIMIT` works at the level of Feynman integrals, allowing for partial fraction decomposition and minimization of integral families. `tapir` can use the information about the graphs associated to the Feynman diagrams for a given problem to not only generate symbolic expressions but also to generate `FORM` code to perform the same tasks `LIMIT` performs.

`tapir` is written in `python` 3 and is developed in a test-driven, as well as self-documented style to ensure reliability and maintainability. The workflow of `tapir` is illustrated in Fig. 1.

In a first step, diagrams are read from the output of `qgraf`, the so-called `qlist` file. Next, diagram filters, in addition to the `qgraf` built-in filters, can be used to extract the diagram classes of interest. Of particular interest is the *cut filter*, based on an algorithm discussed in Ref. [43], which removes diagrams with unwanted Cutkosky cuts. Diagram filters act in general non-destructively, i.e. they only include or exclude diagram classes with specific features, but do not alter the diagrams themselves. The symbolic diagram representation is then built from a set of Feynman rules which are defined in `vrtx` files for interactions and `prop` files for propagators. Symbolic expressions in `tapir` are represented in syntax for the computer algebra system `FORM` and written to so-called `dia` files. Note that these files are compatible with `q2e` input and output files and we also use the same naming scheme. Alternatively, Feynman rules can be read from `UFO` models generated by `FeynRules`, allowing for the automated generation of Feynman rules for many different kinds of quantum field theory models.

In addition, `tapir` offers destructive topology filters which change the topology structure. An example use case is the removal of auxiliary particle lines from the topology, whose propagators do not transfer momentum.

The graph information of the diagrams is later used to identify the topologies in which the propagators, also called denominator functions, $\{D_i\}$ resemble the edges of the underlying graph. Other objects such as numerators or eikonal propagators cannot be expressed in this graph representation. We can only account for them as symbolic objects of the scalar integral family function.

For every graph, the so-called *Nickel index* [44] is then computed. It is a unique naming convention which can be used to identify similar graphs. The Nickel index is used to minimize the graph topologies by building a hash-table with the index as hashing function. Together with the information on how the graph edges must be named to be canonically ordered with respect to the index, we can also give a mapping description between line momenta of similar topologies.

Instead of using the Nickel index for momentum mapping, the program `exp` can be used. It has the advantage of finding mappings on (sub-)topologies rather quickly for a small amount of target topologies. For larger problems, on the other hand, matching Nickel indices has a better algorithmic scaling and enables minimization of topologies in a highly parallelizable manner. For the usage with `exp`, an `edia` file as well as a `topsel` file can be generated which include the topological information of the diagrams. Also these file formats were adopted from `q2e` and `exp`.

The remainder of this article is structured as follows: in Section 2 we introduce the key ideas and algorithms implemented in `tapir`. In Section 3 we discuss the format of the input and output files, while in Section 4 we present explicit examples for the usage of `tapir`. We summarize the features of `tapir` in Section 5. Finally, Appendix A provides details on the interface between `tapir` and `UFO` models.

2. Key concepts and algorithms

In `tapir` we incorporated a few noteworthy algorithms and ideas, summarized in this section.

2.1. Nickel index

To compare different Feynman graph topologies the use of a canonical graph naming scheme is suited best. This has the advantage of efficient pre-computation which can be highly parallelized. The canonical label can afterwards be used as a unique identifier to quickly find graph topologies. Such labels are, for example, implemented in the program pair `nauty` and `Traces` [45]. Although these programs are publicly available and widely used, their graph labeling for our purposes[2] is not directly applicable without additional effort.

[2] In the language of graph theory Feynman graphs are edge-colored graphs with multiple edges and self-loops.

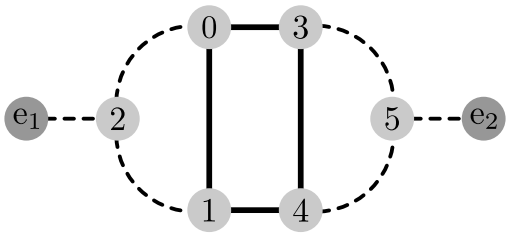


Figure 2: The Nickel index gives a canonical labeling of the vertices of a Feynman graph. The dashed lines denote massless propagators and solid lines propagators with mass `M1`.

To have control over the labeling algorithm we decided to implement a version of the *Nickel index* as a canonical graph label. The algorithm for its computation is described in detail in Ref. [46].

An example topology is given in Fig. 2. Its Nickel index is given by

```
123|24|e|45|5|e| : M1__M1|_M1|q1|M1_||q2|
```

and consists of two parts. The first part uniquely describes how the vertices are connected. The information for each vertex is written as the numeric labels of its adjacent vertices, but in such a way that only the vertex with the smaller number provides this information.

For example, the first entry `123` describes vertex 0. It is connected to vertices 1, 2 and 3. Vertex 1 is connected to 0, 2 and 4. But since the entry for vertex 0 already described the former connection we skip it and get the entry `24` for vertex 1.

Connections to external lines are denoted by "`e`". The different vertex entries are separated by a vertical line "`|`". Thus self-loops and multiple edges can be easily represented.

The second part of the Nickel index describes the edge coloring. In the case of Feynman diagram topologies, this is either the mass of the propagator for internal lines or the momentum for external lines. For massless internal lines the entry is kept empty. The order is the same as in the first part, but the individual edge entries are separated by an underscore "`_`". This notation has the advantage that topologies with special kinematics can be identified, e.g. when two external momenta are equal.

Vertex 0 in this example is connected to vertices 1 and 3 by a line with mass `M1` and to vertex 2 by a massless line. The corresponding second part of the Nickel index to vertex 0 is thus `M1__M1`.

Although the vertex numbering is not unique, the Nickel index is. To obtain a unique label, all possible *Nickel notations* according to different vertex enumerations have to be computed. This notation is simply a string combination of the two described parts. Then all notations are compared and the one with the lowest lexicographical order is kept as the Nickel index. Hence, the full algorithm has at least a complexity of $\mathcal{O}(v!)$, where v is the number of vertices.

Our implementation of the algorithm as a `C++` sub-module provides additional information how the edges are ordered in the Nickel index. We can use this information to find mappings of line momenta of graphs with the same topologies.

A drawback of this approach is the impossibility of mapping subleading topologies, e.g. a topology where one line is missing could in general fit into a "larger" topology with more lines. To overcome this problem, the program `exp` or Pak's algorithm, described in the next section, can be used.

2.2. Pak's algorithm

With the method described above only graphs can be compared. A more general procedure to compare different integral families with or without graph representation is provided by an algorithm by A. Pak [47].

To make use of it, Eq. (1) is written as a Feynman parameter integral. Integration over loop momenta finally leads to

$$I(a_1,...,a_N) = \frac{i^l}{(4\pi)^{dl/2}} \frac{\Gamma\left(\sum_i a_i - \frac{ld}{2}\right)}{\prod_i \Gamma(a_i)} \int_0^\infty ... \int_0^\infty \left(\prod_i \mathrm{d}x_i \; x_i^{a_i-1}\right) \delta\left(\sum_i x_i - 1\right) \frac{\mathcal{U}^{\sum_i a_i - \frac{(l+1)d}{2}}}{\mathcal{F}^{\sum_i a_i - \frac{ld}{2}}} \,. \quad (2)$$

The so-called *Symanzik polynomials* $\mathcal{U}$ and $\mathcal{F}$ can be computed in many different ways (see Ref. [48] for an elaborate overview). We found the most efficient way to be the method implemented in Ref. [49].

The idea of Pak's algorithm is to find a canonical enumeration of Feynman parameters such that the monomials of $\mathcal{U} \cdot \mathcal{F}$ are lexicographically maximal. This is done by iterative renaming of the x_i to find the lexicographically maximal name for x_1. After that follows the search for the maximal x_2 with x_1 fixed, and so on.

As an example we start with the simple integral family

$$I(a_1,a_2) = \int \frac{\mathrm{d}^d k}{(2\pi)^d} \; \frac{1}{(k^2)^{a_1} \; ((k+q)^2 - m^2)^{a_2}} \,. \quad (3)$$

The Symanzik polynomials entering Eq. (2) for this family are given by

$$\begin{aligned} \mathcal{U} &= x_1 + x_2 \,, \\ \mathcal{F} &= x_2^2 m^2 + x_1 x_2 (m^2 - q^2) \,. \end{aligned} \quad (4)$$

x_1 and x_2 can be renamed at will. For a canonical labeling we have to compare the follwing two possibilities lexicographically:

$$\begin{aligned} \mathcal{U}\cdot\mathcal{F} &\overset{\text{lex.}}{>} (\mathcal{U}\cdot\mathcal{F})_{x_1\leftrightarrow x_2} \\ x_1^2x_2(m^2-q^2)+x_1x_2^2(2m^2-q^2)+x_2^3m^2 &\overset{\text{lex.}}{>} x_1^3m^2+x_1^2x_2(2m^2-q^2)+x_1x_2^2(m^2-q^2) \end{aligned} \tag{5}$$

The lexicographic comparison can be a simple string comparison of the individual (properly ordered) monomials. The global definition of "maximal" is irrelevant in this context, since only the relative ordering matters when comparing integral families.

The algorithm starts with expressing the polynomial $\mathcal{U}\cdot\mathcal{F}$ as a matrix, with rows representing vectors of the powers of the x_i for each monomial. Then, the maximal labeling according to x_1 is found by iterative switching of the first with the i'th column. The matrix rows are sorted according to their entries, and the variant with the lexicographically maximal first column vector is kept. Then the same procedure is applied to the second column, keeping x_1 fixed. This continues until all x_i names are determined.

For this example the matrix is given by

$$\begin{pmatrix} 2 & 1 & m^2-q^2 \\ 1 & 2 & 2m^2-q^2 \\ 0 & 3 & m^2 \end{pmatrix} . \tag{6}$$

An additional column is added to take also the prefactors into account. The renaming of $x_1 \leftrightarrow x_2$ now gives the following column-sorted matrix:

$$\begin{pmatrix} 3 & 0 & m^2 \\ 2 & 1 & 2m^2-q^2 \\ 1 & 2 & m^2-q^2 \end{pmatrix} . \tag{7}$$

Comparing the vectors of the first columns, we see that the renaming leads to a lexicographically larger result.

The formal complexity of this procedure grows at least as $\mathcal{O}(e!)$, where e is the number of propagators. If at some point two or more Feynman parameters acquire the same (maximal) lexicographic order, a symmetry is found. When comparing $\mathcal{U}\cdot\mathcal{F}$ of different integral families, one has to iterate over all symmetries and compare each possible x_i enumeration individually. This slows the procedure down and can only be parallelized in exchange for an increased memory usage.

In the original definition of the algorithm, copies of the matrix are created if a symmetry appears. Then the algorithm continues for all subsequent matrices. Keeping only a single matrix while memorizing the symmetries is also known as the *Light Pak algorithm* [50].

Pak's algorithm is nevertheless an outstanding tool for our purposes, as we use it to compare integral families after partial fraction decomposition, which do not necessarily possess a graph representation.

2.3. Partial fraction decomposition

In general, the denominator functions $\{D_i\}$ in an integral family can be linearly dependent. Thus, some sort of partial fraction decomposition must be performed before the reduction with programs such as `FIRE` is possible.

For a generic partial fraction decomposition, we employ the idea of Ref. [43], where the problem was reformulated to finding an appropriate Gröbner basis [51]. The method starts by identifying linear relations between propagators. The easiest way to construct these relations is by representing each denominator function in terms of loop momenta k_i, external momenta q_i and masses m_i. For this purpose we construct a tuple of scalar products and masses that appear in the denominator functions D_i. The ordering in the tuple is chosen such that terms with and without loop momenta are split, e.g.

$$\mathbf{P} = \left(\begin{array}{c} k_1^2 \\ k_1.q_1 \\ k_1.q_2 \\ \hline q_1^2 \\ q_2^2 \\ q_1.q_2 \\ m^2 \end{array}\right) \equiv \begin{pmatrix} \mathbf{P_l} \\ \mathbf{P_n} \end{pmatrix} . \tag{8}$$

Then a coefficient matrix M is constructed which expresses the inverse denominator functions linearly in terms of $\mathbf{P}$:

$$M \cdot \mathbf{P} = \begin{pmatrix} D_1^{-1} \\ D_2^{-1} \\ D_3^{-1} \\ \vdots \\ D_N^{-1} \end{pmatrix} \equiv \tilde{\mathbf{D}} \, . \tag{9}$$

The linear relations between the D_i^{-1} are found by computing a left-inverse M_L^+ of M and performing Gaussian reduction such that M_L^+ takes a row echelon form. This is easily achieved by transforming the matrix

$$\left(M \,\middle|\, \mathrm{diag}\left(\tilde{\mathbf{D}}\right)\right) \tag{10}$$

to row echelon form. The rows containing only zeros in the M-side of this matrix lead to linear equations of the form

$$0 = \sum_i a_{ji} D_i^{-1} \, , \tag{11}$$

whereas rows containing only non-zero entries of $\mathbf{P_n}$ (i.e. loop-momenta independent) give rise to linear equations of the form

$$b_j = \sum_i a_{ji} D_i^{-1} \,. \tag{12}$$

Both, Eq. (11) as well as Eq. (12), can be used iteratively to reduce the number of denominator functions of an integral family. Gröbner bases can help to apply these relations in such a way that only families with linear independent denominator functions remain. Thus, we find the partial fraction reduction steps directly, when we use the following polynomial basis as input:

$$K = \left\{ \left\{ \sum_i a_{ji} D_i^{-1} \right\}, \left\{ \sum_i a_{ki} D_i^{-1} - b_k \right\}, \left\{ D_l \cdot D_l^{-1} - 1 \right\} \,\middle|\, \forall\, j,k,l \right\} . \tag{13}$$

Applying the Buchberger algorithm with respect to all denominators and inverse denominators $\{D_i, D_i^{-1}\}$ gives the iterative (and always deterministic) prescription for partial fraction decomposition with arbitrary indices for a given integral family we were looking for. The Buchberger algorithm was found together with the idea of Gröbner bases in Ref. [51].

The flavor of the actual implemented Buchberger algorithm defines how it scales with the number of linearly dependent denominator functions. However, the worst possible scaling behavior is double exponential [52]. At present, different approaches using heuristics and advanced monomial orderings are widely available in many computer algebra systems. As for all symbolic manipulations within `tapir`, we use `sympy` [53] for this purpose with its implementation of the *improved Buchberger algorithm* [54].

2.4. Cutkosky cut filter

In some scattering problems in quantum field theory it is useful to utilize the unitarity constraint to apply the optical theorem. With it, it is possible to relate quantities such as total cross sections or decay rates to the imaginary part of a scattering amplitude. To compute the latter, it is common to use the *Cutkosky cutting rules* [55], which state that the imaginary part of a Feynman diagram equals the sum of all subsequent "cut" diagrams. A cut is defined as a set of loop-propagators which separates external vertices when removed. These cut propagators are then treated as on-shell. The Cutkosky rules allow to exclude diagrams that only contain cuts leading to final states that are not allowed kinematically. In addition, diagrams with specific cuts, e.g. corresponding to a specific decay mode, can be singled out.

For these purposes `tapir` offers a *cut filter* option to collect only diagrams with certain cut properties. It allows to effectively reduce the number of diagrams in an early calculational

stage, and hence to reduce the complexity of problems in which decay processes play a crucial role.

The cut filter is based on an algorithm by A. Pak, described in great detail in Ref. [43]. Several instructions need to be provided to the algorithm, such as whether a filter shall be applied inclusively or exclusively. Also the lines playing the start and end points for the cut search need to be specified, since the goal is to separate both. Furthermore, the types and numbers of particles that can be cut for a kinematically allowed configuration need to be specified. Several cut filter rules can be applied simultaneously to enable complex filter combinations.

The mentioned algorithm to find the diagrams that are allowed according to the given rules, can be described as follows:

0) Invert all disallowed ranges of cut particles and reformulate them as an allowing rule, e.g. $\{\texttt{false}, [1,1] \vee [4,6]\} \;\rightarrow\; \{\texttt{true}, [0,0] \vee (1,4) \vee (6,\infty)\}$.

1) Categorize, a priori, all lines of a diagram either in the groups `M` (must be cut), `C` (can be cut) or `N` (must not be cut) according to the filter arguments.

2) Colorize all source vertices identically (label `0`), as well as all sink vertices (label `1`).

3) Specify the colors of all other internal vertices by the following rules as far as possible: Adjacent vertices that are separated by an `N` line have the same color; the ones connected by an `M` line must have different colors. Introduce new ancillary colors if the coloring is ambiguous.

4) Iterate over all possible assignments of the the newly introduced colors to the color of the sources (`0`) or of the sinks (`1`) respecting the rules of step 3 and the defined filter restrictions in every iterative step.

5) Successful bi-coloring of a graph indicates that the separation between sources and sinks is allowed and the diagram is kept. On the other hand, diagrams for which no bi-coloring can be applied are dropped.

Step 0 must be applied only once, whereas the rest is done for each diagram individually. Splitting this vertex coloring procedure into different steps allows to drop diagrams already in step 2 if vertex clusters cannot be sufficiently separated.

Due to this heuristic exclusion approach, the algorithm does not suffer as much from combinatorially expensive recoloring as the naive brute force approach would. Hence, it is well suited for large scale problems.

The cut filter has been tested in the case of real-virtual and double-real corrections to Higgs boson pair production. These corrections were calculated in Refs. [56, 57] using the program `gen` [58] for cut filtering.

3. Definitions

To use `tapir` several file formats need to be introduced. The `config`, `prop` and `vrtx` files provide options and Feynman rules to `tapir` and need to be specified by the user. All other files discussed in this section provide additional information about the analyzed diagrams and topologies, or have specific formats to be directly read from `FORM` or `Mathematica` programs. Most of the files described here follow the conventions and implementations of `q2e` and `exp` to ensure compatibility with existing setups and workflows.

In future releases, additional files and options might be introduced. Hence, we always refer to the project repository and the user documentation therein as a single source of reference.

3.1. `config` files

To hand complex instructions to `tapir`, the command line alone is not sufficient due to the vast set of options. Instead, a `config` file has to be provided with instructions of the following form:

```
* tapir.[option keyword] {: [option argument 1]} {: [option argument 2]}
   ...
```

Lines without the `* tapir` tag at the beginning are treated as comments. Also note the mandatory whitespace after the asterisk.

To use `tapir` in a pipeline setup, i.e. different program calls from the same directory, it is often useful to write multiple `config` files and provide each of them separately as a command line argument. In addition, `tapir` has further command line options which can be provided via the `config` file as well, with prioritization of the former.

The following example instructs `tapir` to read a `qgraf` output file `qlist.2` and draw some diagrams thereof with representative topologies:

```
Propagator and vertex Feynman rules
* tapir.propagator_file qcd.prop
* tapir.vertex_file qcd.vrtx

qgraf input
* tapir.qlist qlist.2

Output
* tapir.repart representatives.tex

Define mass of top quark
* tapir.mass t : M1

Drawing options
* tapir.draw_particle t : fermion : $t$
* tapir.draw_particle g : gluon : $g$
```

```
* tapir.draw_particle h : scalar : $h$
```

The output `representatives.tex` is a LaTeX file using the `TikZ-Feynman` package [59] to draw the corresponding diagrams. It can be compiled using the broadly available program `lualatex`. The drawing options at the end specify how the propagators of the individual particles are drawn, and what label should be printed next to them. The Feynman rule files `qcd.prop` and `qcd.vrtx` are mandatory because they include the particle and anti-particle information of the theory.

3.2. `prop` and `vrtx` files

The `prop` and `vrtx` files define the Feynman rules of the propagators and vertices that are declared in `qgraf` `lagrangian` files. They contain symbolic `FORM` code for each tuple of particles that correspond to the specified Feynman rule, which can be used to produce `dia` (see Subsection 3.3) files from `qgraf`'s output `qlist` file. It is possible to have multiple Feynman rules for the same tuple of particles, for example in theories with multiple Feynman rules corresponding to four-fermion operators defined by the same set of particles. In this case, a separate entry has to be added to the `vrtx` file for each Feynman rule of the respective vertex. All entries of `prop` and `vrtx` files are of the same generic form:

```
{particle1,particle2(,particle3,...):Lorentz|QCD|QED|EW}
```

In the case of propagators, only `particle1` and `particle2` are used, vertices contain at least three particles. The particles `particle1` and `particle2` which share a propagator are interpreted as anti-particles of each other. The particle content is separated from the Feynman rule by a ":", and the Feynman rule itself is divided into four different folds, separated by "|". The `QED` and `EW` folds are usually empty, as they are historical structures that do not serve any special purpose anymore. The `Lorentz` and `QCD` folds serve as a possibility for the user to isolate different parts of the calculation, in particular the color algebra and the Dirac structure, and tackle them separately, which is particularly useful for large-scale calculations. Of course, the decision whether to split the Feynman rules or not is entirely up to the user, in particular the `QED` and `EW` folds can be used to further split a calculation into smaller pieces. In most cases, the Feynman rules involve template variables such as `<lorentz_index_particle_3>`, which refers to the Lorentz index of the third particle entering the vertex, that will be replaced by a definite index or symbol once `tapir` generates the `dia` file. These placeholder variables can be used to construct products of tensors in different spaces, e.g. for spinor and color structures. The entry of each fold of a Feynman rule must begin with a multiplication asterisk "*", as the symbolic diagram expression results from concatenation (i.e. multiplication) of several strings from vertices and propagators.
Examples of typical `prop` and `vrtx` entries are given in Subsection 4.1 and Appendix A.

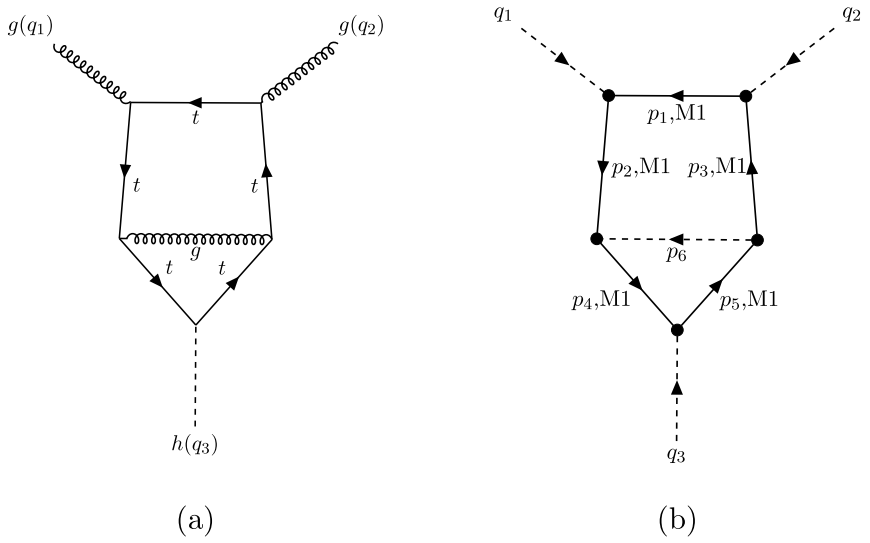


Figure 3: The graphic representations of the diagram (a) and topology (b) of the examples in Section 3 were generated using the `--diagramart` and `--topologyart` command line options. The line labels were shifted for better readability. Note: external momenta are always treated as incoming.

3.3. `dia` files

Based on the Feynman rules given in `prop` and `vrtx` files `tapir` generates four `FORM` code folds for each diagram: `Lorentz`, `QCD`, `QED` and `EW` fold. These folds are written to the `dia` file. Note that the `dia` file is compatible with `dia` files produced by `q2e` and can serve as an input to `exp`.

As an example, consider the two-loop diagram of Fig. 3 (a) contributing to Higgs boson production in gluon fusion:

```
*--#[ d2l3 :

  (-1)*1*nh
  *FT1(mu2)
  *FT1(p3)
  *FT1(nu17)
  *FT1(p5)
  *FT1(M1)
  *FT1(p4)
  *FT1(nu18)
  *FT1(p2)
  *FT1(mu1)
  *FT1(p1)
  *Dg(nu17,nu18,p6)
  ;

  #define TOPOLOGY "arb"
```

```
  #define INT1 "arb"

*--#] d2l3 :

*--#[ fqcd2l3 :

  1
  *GM(a(2),j7,j12)
  *d_(j12,j11)
  *GM(b(17),j11,j16)
  *d_(j16,j15)
  *d_(j15,j14)
  *d_(j14,j13)
  *GM(b(18),j13,j10)
  *d_(j10,j9)
  *GM(a(1),j9,j8)
  *d_(j8,j7)
  *prop(b(17),b(18))
  ;

*--#] fqcd2l3 :

*--#[ fqed2l3 :
  1
*--#] fqed2l3 :

*--#[ few2l3 :
  1
*--#] few2l3 :
```

The two numbers in the name of the `Lorentz` fold denote the number of loops and the number of the diagram, respectively. All other folds follow this pattern, but have additional prefixes to denote the respective fold. Each fold can be loaded using `FORM`'s `#include` preprocessor directive.

The first line in the `Lorentz` fold contains the symmetry factor provided by `qgraf` (`(-1)` in this case), factors for counting closed fermion loops as defined in the `config` file[3] (`*nh` in this case) and a `*1` in the case of the previous factors being trivial and thus absent. The following lines contain the Feynman rules for all occuring propagators and vertices, filled with the respective open indices (e.g. `mu2`), line momenta (e.g. `p1`) and masses (e.g. `M1`). The demonstrated Feynman rules incorporate a non-commuting object `FT1` which corresponds to a Dirac gamma matrix, either with an open index or contracted with a line momentum. Similarly, the `QCD` fold contains the color part of the Feynman rules, in which `GM` refers here to the Gell-Mann matrix.

[3]To obtain information about closed fermion loops, the option `tapir.contract_fermion_lines` must be used and all fermion names must start with "f" to indicate them as such. With this option it is not possible to directly define e.g. four-fermion interactions.

In this example the other two folds are empty, but they can be populated depending on the Feynman rules.

3.4. `edia` and `topsel` files

The information necessary for topology identification of each diagram is contained in the `edia` file. Each entry takes the form

```
{Name; Number of lines; Number of loops; Number of independent ext. legs;
  Number of masses; List of scales; List of line momenta}
```

where the name is the same as the name of the `Lorentz` fold in the `dia` file, the list of scales is taken from the `config` file and the list of line momenta contains entries of the form `(Momentum,Mass:v1,v2)` where `v1` and `v2` are the vertices the momentum flows in between. For external momenta, the two numbers `v1` and `v2` denote where the momentum enters and leaves the graph, respectively.

In the case of the diagram "2l3" of the previous section, the corresponding topology is visualized in Fig. 3 (b). Its `edia` entry takes the following form:

```
{d2l3;6;2;2;1;M1,q1,q2;(q1:1,3)(q2:2,3)(p1,M1:2,1)(p2,M1:1,4)(p3,M1:5,2)
  (p4,M1:4,3)(p5,M1:3,5)(p6:5,4)}
```

As a consequence of momentum conservation, $q_3 = -(q_1 + q_2)$ is not an independent external leg, and hence only two external legs are counted in the above example.

When performing a naive expansion in a scale with an external program like `exp`, the corresponding scale will not be listed in the list of scales. Furthermore an `,e` will be added to it in all relevant entries in the list of line momenta. As an example consider the previous diagram, but in a Taylor expansion for a small internal quark mass `M1`:

```
{d2l3;6;2;2;1;q1,q2;(q1:1,3)(q2:2,3)(p1,M1,e:2,1)(p2,M1,e:1,4)
  (p3,M1,e:5,2)(p4,M1,e:4,3)(p5,M1,e:3,5)(p6:5,4)}
```

Similarly, a Taylor expansion for a large quark mass and small external momenta can be performed:

```
{d2l3;6;2;2;1;M1;(q1,e:1,3)(q2,e:2,3)(p1,M1:2,1)(p2,M1:1,4)(p3,M1:5,2)
  (p4,M1:4,3)(p5,M1:3,5)(p6:5,4)}
```

For programs such as `exp` a list of topology selection (`topsel`) entries can be generated. These take a similar form as the `edia` entries:

```
{Name; Number of lines; Number of loops; Number of independent ext. legs;
  Number of masses; List of options; List of line momenta;
  Mass assignment}
```

The first five entries resemble those of an edia entry. They are followed by a list of options that can be filled for use with an external program. The last two entries contain a list of line momenta and the mass assignment of each line. In contrast to the line momenta in edia entries the list of line momenta does not contain information regarding the masses. This information is contained in a number with one digit per internal line. Each digit is either 0, in case the line is massless, or a number between 1 and 9 corresponding to the masses M1 to M9 to be defined as tapir.mass option in the config file.

For example, a topology entry generated by tapir resembling the edia entry of d2l3 takes the form:

```
{Tri2la;6;2;2;1;;(q1:1,3)(q2:2,3)(p1:2,1)(p2:1,4)(p3:5,2)(p4:4,3)(p5:3,5)
   (p6:5,4);111110}
```

As in the case of the dia files, the format of the edia and topsel files is equivalent to those produced by q2e and used by exp.

3.5. topology files

tapir can also generate FORM code to rewrite scalar products of loop momenta and propagators in terms of scalar functions. Given a topsel entry, a subset of the line momenta is selected as the loop momenta and all other line momenta are decomposed as a sum of external momenta and the loop momenta.

As an example, in the case of the topsel entry discussed above, the corresponding FORM code for rewriting numerators takes the form

```
* Reducible numerator momentum replacements
id p6 = -p3 + p5;
id p1 = p3 + q2;
id p2 = p3 + q1 + q2;
id p4 = p5 + q1 + q2;

.sort
```

Here p3 and p5 have been chosen as loop momenta.

In a next step, tapir rewrites scalar products of loop momenta and external momenta in terms of scalar products of external momenta and inverse denominators.

For the case under consideration, the code is given by

```
* Numerator momentum product replacements
id p5.q1 = p4.p4/2 - p5.p5/2 - p5.q2 - q1.q1/2 - q1.q2 - q2.q2/2;
id p3.q1 = -p1.p1/2 + p2.p2/2 - q1.q1/2 - q1.q2;
id p3.q2 = p1.p1/2 - p3.p3/2 - q2.q2/2;
id p3.p5 = p3.p3/2 + p5.p5/2 - p6.p6/2;
```

```
.sort

* Define massive propagators
id p1.p1 = -1/s1m1 + M1^2;
id p2.p2 = -1/s2m1 + M1^2;
id p3.p3 = -1/s3m1 + M1^2;
id p4.p4 = -1/s4m1 + M1^2;
id p5.p5 = -1/s5m1 + M1^2;

.sort
```

Here, the first block of code rewrites scalar products in terms of the momenta of the propagators, while the second block takes into account masses of propagators. The symbols `sImJ` denote the propagators $1/(m_j^2 - p_i^2)$.[4]

In the next step, the massive denominator factors and remaining scalar products are rewritten in terms of a scalar function.

For the given example, the following `FORM` code is generated:

```
* Combine to scalar topology function
id s5m1^n0? * s3m1^n1? * s4m1^n2? * s2m1^n3? * s1m1^n4? * 1/p6.p6^n5? *
   1/p5.q2^n6? =
   (-1)^n5 * (-1)^n6 *
   Tri2la(n0,n1,n2,n3,n4,n5,n6);

.sort
```

Furthermore, if the problem involves a partial fraction decomposition, the corresponding code is appended here.

3.6. `topology list` files

`tapir` also generates a `Mathematica` readable `topology list` file containing the definition of the propagators of each resulting scalar topology function of the `topology` files. The entries have the form of

```
{"Name", List of denominators, List of loop momenta}
```

In the case of the topology discussed above, the entry is given by

```
{"Tri2la", {M1^2 - p5^2, M1^2 - p3^2, M1^2 - (p5 + q1 + q2)^2,
  M1^2 - (p3 + q1 + q2)^2, M1^2 - (p3 + q2)^2, -(p3 - p5)^2, -p5*q2},
  {p3, p5}}
```

[4] This choice of the sign allows for direct use of the output through programs working with Euclidean loop momenta, such as `MATAD` or `MINCER`.

Note that regular multiplication is used instead of a scalar product.

4. Usage and example

The installation of `tapir` is simple once the repository has been cloned. We recommend the usage of a pre-packed release version for a defined and well tested set of features. New features can be found on the development (master) branch. The installation is done with

```
$ make install
```

The shell-executable `tapir` can now be used. To get a hint on the command line options, the instruction

```
$ /path/to/tapir --help
```

is useful. For further reference we strongly recommend the user manual in the `doc` folder of the repository.

To illustrate the basic features of `tapir` we follow some basic examples which may occur in loop calculations. The first example highlights the multi-loop aspect and advantages of `tapir`, whereas the second and third examples show features which help to address more specific problems. In the fourth example, we show how `tapir` can be used to convert pre-generated Feynman rules into `tapir` and `qgraf` input files. All examples can be found in the `example` subdirectory.

4.1. Example 1: The 3-loop gluon self-energy

Generating symbolic diagrams

Computations of this order are usually only possible with a suited `FORM` setup due to the large size of intermediate terms. Let us start with `qgraf` using the `tapir` style file `qgraf-tapir.sty`. Our model (called `lagrangian` file in `qgraf`) contains all QCD-relevant interactions with gluons, ghosts and quarks. In addition, we split the four-gluon interaction into three sub-diagrams with only three-particle interactions using a so-called *sigma particle* (see e.g. Ref. [60]). This enables the factorization of color and Lorentz related parts of the Feynman rules, which we can hence compute separately for the whole diagram.

`qgraf` produces $\mathcal{O}(1600)$ diagrams if all six quark flavors are treated individually. The resulting `qlist` file is a valid input for `tapir`. Example diagrams are shown in Fig. 4.

To obtain a symbolic expression of each diagram, we have to specify Feynman rules for each propagator and interaction that was used in the `lagrangian` file. The symbolic Feynman rules have to be provided in the `prop` and `vrtx` files, respectively. For example, the gluon propagator looks like

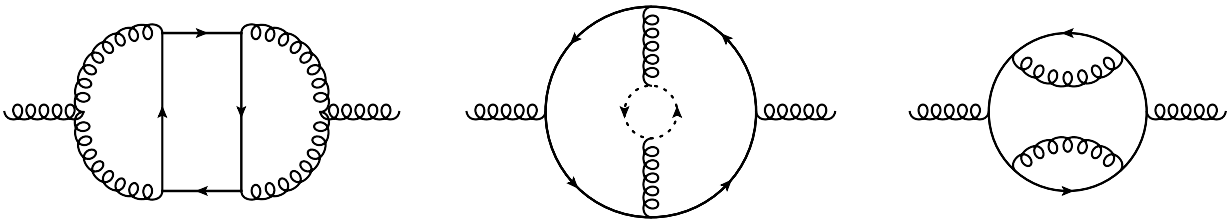


Figure 4: `qgraf` generates $\mathcal{O}(1600)$ diagrams for the gluon self-energy at 3-loop order. We included ghosts and sigma particles in our Lagrangian definition.

```
{g,g:*Dg(<lorentz_index_vertex_1>,<lorentz_index_vertex_2>,<momentum>)
  |*prop(<colour_index_vertex_1>,<colour_index_vertex_2>)||}
```

A quark-gluon vertex is given by

```
{fU,fu,g:*ffgVertex(<lorentz_index_particle_3>,
    <spinor_index_particle_1>,<spinor_index_particle_2>)
  |*GM(<colour_index_particle_3>,
    <spinor_index_particle_1>,<spinor_index_particle_2>)
  ||}
```

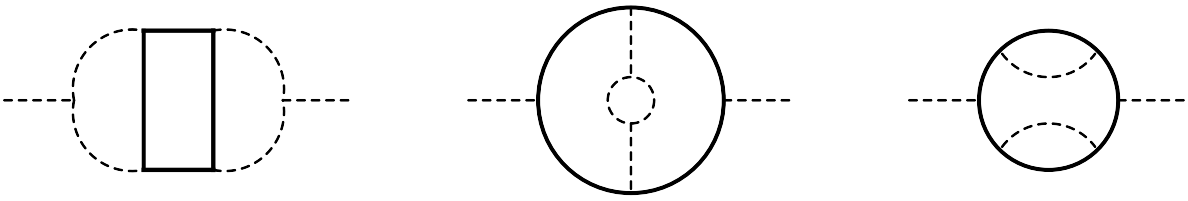
The structure of `prop` and `vrtx` entries is defined in Subsection 3.2.

All configurations and options of `tapir` can be provided in the `config` file. If we simply want to insert all Feynman rules per diagram, the `config` file reads:

```
Define propagator Feynman rules
* tapir.propagator_file qcd.prop

Define vertex Feynman rules
* tapir.vertex_file qcd.vrtx

Declare and assign masses to particles
* tapir.scales M1
* tapir.mass ft:M1

The following options can also be given via command line.
qgraf input
* tapir.qlist qlist.3

Output
* tapir.diaout gg3l.dia
* tapir.ediaout gg3l.edia
```

The options here are more or less self-explanatory. To specify massive particles the `tapir.mass` option for the corresponding particle needs to be provided. As an output we specify two files: `dia` and `edia` files are discussed in Subsection 3.3 and Subsection 3.4

The last three options can also be provided via the command line. But here we specify



```
123|24|e|45|5|e|
: M1__M1|_M1|q1|M1_||q2|
```

```
112|3|45|45|e|e|
: __||M1_M1|M1_M1|q1|q2|
```

```
112|3|e4|e5|55|
: M1__M1|M1|q1_M1|q2_M1|_M1|
```

Figure 5: The topology structure of the diagrams of Fig. 4 shows that only the propagator connections and the masses are relevant for the Nickel index. The dashed lines indicate massless propagators and the solid lines massive ones.

everything in the `config` file such that we can run `tapir` simply with

```
$ /path/to/tapir -c myconf.conf
```

Topology analysis

`tapir` is also able to map and minimize the topologies of the generated diagrams. For this, we add the following lines to the `config` file:

```
* tapir.minimize
* tapir.topselout reducedTopologies.topsel
```

The first option computes the Nickel index (see Subsection 2.1) for every diagram. Examples are shown in Fig. 5. Afterwards, the Nickel indices between all diagrams are compared and only unique ones are kept. The remaining topologies are then written to a `topsel` file which has a similar format as the `edia` file. The former is used to map onto target topologies with `exp`. We adapted the file format here to express topologies in general. It is also possible to use a `topsel` file as input and, for example, minimize its content with respect to another `topsel` file. This explicit handling of topologies in in- and output allows for the realization of different minimization and mapping approaches that fit best to the problem at hand.

At the end of the minimization we are left with 60 unique topologies. The following option gives us the possibility to express diagrams in terms of a scalar topology function:

```
* tapir.topologyfolder topologies
```

It creates a `FORM` file for every remaining topology, which reduces the possible numerator and denominator structures assuming common Feynman rules (i.e. quadratic massive or massless denominators). More specialized propagators, e.g. eikonal ones, can also be requested by a different option. These `topology` files reside in the directory `topologies` together with a `Mathematica` readable `topology list` file which can be used as an input for reduction programs such as `FIRE`.

For visualization the option

```
* tapir.topologyart topologies.tex
```

can be used to generate a `TikZ-Feynman` representation for every remaining topology. The generated LaTeX file can be compiled using `lualatex`.

4.2. Example 2: Partial fraction decomposition

This example is concerned with the partial fraction decomposition of Feynman integrals with linearly dependent propagators. Such integrals occur whenever kinematic configurations are chosen in which external momenta are nullified or become proportional to other external momenta. As an example consider QCD corrections to $B_s - \overline{B}_s$ mixing. This requires the computation of $\bar{b} + s \to b + \bar{s}$ in forward scattering diagrams (see e.g. Ref. [61]). If terms suppressed by the b-quark mass are neglected, these diagrams can be computed for vanishing strange-quark momentum. Thus, we can set $p_2 = p_4 = 0$ and $p_1 = -p_3$. In the following we will use `tapir` to identify all relevant topologies for this problem, nullify p_2 and p_4, find linearly independent topologies and generate `FORM` code to map linearly dependent integrals onto them.

After generating the relevant four-point diagrams with `qgraf` we can instruct `tapir` to nullify the two momenta using the options

```
* tapir.external_momentum q2:0
* tapir.external_momentum q4:0
```

in the configuration file (see `example/Bmix/Bmix.conf`).

Calling `tapir` with

```
$ /path/to/tapir -c Bmix.conf -q qlist.2 -m -t Bmix.topsel -f topologies
    -pm
```

loads the configuration file `Bmix.conf` and the output of `qgraf` (`example/Bmix/qlist.2`). The `-m` switch leads to the minimization of integral topologies based on the given diagrams, followed by the generation of a `topsel` file for `exp`. The `-f` switch instructs `tapir` to generate `FORM` `topology` files for rewriting propagators and scalar products in terms of scalar integral familes. Finally, the `-pm` switch triggers the partial fraction decomposition of linearly dependent denominator functions of the integral families and a subsequent minimization. The respective code is added to the `topology` files.

4.3. Example 3: Filtering cuts

In computations based on the method of reverse unitarity [3] phase space integrals over squared amplitudes are rewritten as cut loop integrals in forward scattering kinematics (see

Subsection 2.4). This method allows the application of techniques developed for regular loop integrals to phase space integrals. However, diagram generators such as `qgraf` are not capable to only generate diagrams with the correct cuts. As a consequence, we need to filter the output of `qgraf` using `tapir` before doing further manipulations.

As an example consider higher order corrections to Higgs boson pair production in the large top quark mass expansion. In this expansion top quark loops reduce to effective gluon-Higgs vertices and as a consequence only diagrams with massless quarks, gluons and Higgs bosons need to be considered. Example diagrams can be found in Refs. [56, 57]. To filter the output of `qgraf` we add

```
* tapir.filter cuts : true : q1,q2 : h : 2,2
* tapir.filter cuts : true : q1,q2 : g,c,fq : 1,2
```

to the respective configuration file. These options select all diagrams with an s-channel cut through two Higgs bosons (`h`) and any combination of one or two gluons (`g`), light quarks (`fq`) or ghosts (`c`).

Should we only be interest in final states with exactly two quarks we can modify the second line to read

```
* tapir.filter cuts : true : q1,q2 : fq : 2,2
```

Alternatively, we could filter diagrams with cuts through gluons and ghosts but no quarks by the following combination of options:

```
* tapir.filter cuts : true : q1,q2 : h : 2,2
* tapir.filter cuts : true : q1,q2 : g,c : 1,2
* tapir.filter cuts : false : q1,q2 : fq :
```

The option arguments are described as follows. The first boolean argument specifies whether the diagrams fulfilling the stated cutting restriction are exclusively kept (`true`) or excluded (`false`) from further evaluation.

The second argument describes which external momenta are treated as "sources". To find cuts the external lines need to be divided into "sources" and "sinks", such that a cut is defined as a cut through non-bridge lines (i.e. lines that are part of loops) that separates sinks from sources. By specifying some external lines (according to their momenta) as sources, all other external lines are treated as sinks.

The third argument (if provided) specifies for which particles the cutting restriction shall apply. Thus, cuts through different particle types can be cumulatively accounted for. An empty argument is equivalent to specifying all occurring particles.

The last argument defines the ranges of how many cuts of the particles of the given kind are allowed. Several allowed ranges can be provided at the same time. Alternatively, the range $[1, \infty)$ can be specified by leaving the argument empty.

As shown, several filter options can be provided in a single `config` file to confine the filtered diagram subset even more. Hence, the filter system allows logical "∧" and "¬" operations. A logical "∨" is only partly supported but can be induced by multiple `config` files.

4.4. Example 4: Using an UFO model to provide Feynman Rules

Here, we give a short example of how to use the `UFOReader` to import Feynman rules corresponding to the Standard Model `FeynRules UFO` files that are provided together with `tapir`.

The `UFOReader` requires a minimal configuration file (e.g. `example/UFO/SM/ufo.conf`), consisting of only two options

```
Directory containing the input UFO files
* tapir.ufo_dir Standard_Model_UFO/

Directory for the output (.lag, .vrtx, .prop, .inc) files
* tapir.fr_pref tapir_SM_UFO/
```

which specify the input and output directories, respectively. `tapir` is then called by simply executing

```
$ /path/to/tapir -ufo -c ufo.conf -fr
```

and will read the `UFO` files and produce a number of output files for later use with `qgraf`, `tapir` and `FORM`.
Note that the `lagrangian` and `prop` files will have two particle components, a "transversal" and a "longitudinal" one, for each massive vector boson, cf. Appendix A.1 for details. Some vertices in the `vrtx` file carry an explicit gauge-parameter dependence in the `Lorentz` fold, and the four-gluon vertex involves the function `nonfactag()`, both of which are explained in Appendix A.2. In addition, in the files `UFOdecl.inc` and `UFOrepl.inc`, we find declarations and definitions of coupling constants in terms of parameters of the theory, which can be used in `FORM`-based setups.
An exemplary use case of the `UFOReader` would be to first convert a `FeynRules UFO` module into the previously mentioned files, then use `qgraf` to generate all diagrams for a given process with the `lagrangian` file as input. Then, use the Feynman rule definitions in the usual way using `tapir` a second time.

5. Summary and Outlook

We present `tapir`, a program for processing multi-loop Feynman diagrams and working with Feynman integral families. `tapir` allows for identification and minimization of topologies, partial fraction decomposition of linearly dependent topologies, filtering diagrams

allowing for any given Cutkosky cut, and it provides an interface to UFO for importing Feynman rules. Furthermore, FORM code can be generated for the symbolic computation of Feynman diagrams. We provide four examples showcasing the features of tapir and discuss the relevant input and output files. tapir already came to use in [62].

The source code, further examples as well as more detailed documentation can be obtained from https://gitlab.com/F.Herren/tapir. Furthermore, additional features and performance improvements might be added to tapir in the future.

Acknowledgements

We thank Fabian Lange, Vladyslav Shtabovenko and Matthias Steinhauser for careful reading of the manuscript.

This research was supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under grant 396021762 — TRR 257 "Particle Physics Phenomenology after the Higgs Discovery". M.L. is supported by the BMBF grant 05H15VKCCA.

F.H. acknowledges the support of the Alexander von Humboldt Foundation. This document was prepared using the resources of the Fermi National Accelerator Laboratory (Fermilab), a U.S. Department of Energy, Office of Science, HEP User Facility. Fermilab is managed by Fermi Research Alliance, LLC (FRA), acting under Contract No. DE-AC02-07CH11359.

Appendix A. The UFOReader module

Given a representation of the Feynman rules of a model in terms of vrtx, prop, and lagrangian files, tapir is able to generate FORM expressions from qgraf-generated [4] diagrams. The vrtx, prop, and lagrangian files can be manually written by the user. If the model at hand is complicated, however, the number and complexity of Feynman rules grows, and such an approach can be prone to errors. FeynRules [12] is a convenient tool to automatically generate Feynman rules for a given theory. The Universal FeynRules Output [13] (UFO) provides a complete, easy-to-use set of FeynRules-generated Feynman rules in terms of python files. In this section, we describe the UFOReader module that allows to read in UFO files and convert them into appropriate vrtx, prop, and lagrangian files.

The python module produced by UFO contains a number of different files with information about the particles (particles.py) and vertices (vertices.py, couplings.py and lorentz.py) of the given model, with additional information about the parameters given in parameters.py. An UFO module also contains the file propagators.py with information about spinless, spin-$\frac{1}{2}$ and spin-1 propagators, either in Feynman ($\xi = 1$) or unitary

($\xi \to \infty$) gauge. We emphasize here that the `UFOReader` should be used with models exported in the Feynman gauge $\xi = 1$. Note that currently `UFO` files are produced as `python` 2 modules; in order to use such a module, the user has to convert it into a `python` 3 module. At present, the program `2to3` [63] seems to work well.

The `UFOReader` module will import the model package from the path that is specified with the directive

```
* tapir.ufo_dir PATH_TO_MODEL
```

where `PATH_TO_MODEL` is the path to the directory containing the `__init__.py` file of the model. In addition, a directory `PATH_TO_OUTPUT_DIR` has to be given for the resulting files to be placed in, which can be specified with:

```
* tapir.fr_pref PATH_TO_OUTPUT_DIR
```

`tapir` will then import and parse the `UFO` files and produce the `lagrangian`, `prop` and `vrtx` files that can be used with `qgraf` and `tapir` again, as well as two additional files `UFOdecl.inc` and `UFOrepl.inc`, intended for the use with `FORM`-based setups to take care of automated declaration and replacement of symbols, respectively.

In order to treat massive gauge bosons in general R_ξ gauge, a list of massive spin-1 bosons is extracted from the list of all particles. We will describe the treatment of massive vector bosons in Appendix A.1. Note that the `UFOReader` and resulting `prop` and `vrtx` files inherit the use of spinor, Lorentz and color indices from the original `UFO` files.

The `UFOReader` module has been tested on models containing spin-0, spin-$\frac{1}{2}$ as well as spin-1 particles in the trivial, fundamental or adjoint representation of $\mathrm{SU}(3)_\mathrm{C}$, respectively, and has been used in the context of calculations within the Standard Model and the Two-Higgs-Doublet Model. Higher spin particles and particles in different representations of $\mathrm{SU}(3)_\mathrm{C}$ have also been implemented, but the primary use case is for the Standard Model and sufficiently similar theories.

The functions appearing in the Feynman rules are defined in the user documentation. In the following we focus on two special cases that need further discussion: propagators of massive vector bosons and vertices for which the color and Lorentz structure does not factorize.

Appendix A.1. Propagators

The propagators of massless particles, massive spin-zero particles and spin-$\frac{1}{2}$ fermions as well as ghosts are translated into corresponding expressions suitable for `prop` files in a straightforward manner. Massive gauge-boson propagators in R_ξ gauge are cumbersome to deal with, since the propagator contains two denominators with different masses, $k^2 - M^2$ and $k^2 - \xi M^2$, at the same time, which makes the automated calculation of loop integrals rather cumbersome. A standard solution is two split the massive propagator into two

separate propagating particles, the "transversal" and "longitudinal" components, which is implemented in the `UFOReader` module. The ξ-dependent propagators of massive gauge bosons are expanded as

$$D_{\mu\nu}(k, M, \xi) = \mathcal{D}_{\mathrm{T},\mu\nu}(k, M, \xi) + \mathcal{D}_{\mathrm{L},\mu\nu}(k, M, \xi)\,, \tag{A.1}$$

where the two components

$$\mathcal{D}_{\mathrm{T},\mu\nu}(k, M, \xi) = \frac{-\mathrm{i}\left(g_{\mu\nu} - \frac{k_\mu k_\nu}{M^2}\right)}{k^2 - M^2}\,, \tag{A.2}$$

$$\mathcal{D}_{\mathrm{L},\mu\nu}(k, M, \xi) = -\frac{\mathrm{i}}{M^2}\frac{k_\mu k_\nu}{k^2 - \xi M^2} \tag{A.3}$$

each feature only a single mass in the propagator denominator. The subscripts T ("transversal") and L ("longitudinal") refer to the fact that

$$k^\mu \mathcal{D}_{\mathrm{T},\mu\nu}(k, M, \xi) = 0 \tag{A.4}$$

if $k^2 = M^2$, that is if the massive gauge boson goes on-shell. At the technical level, this involves introducing two particles X_T and X_L for each massive vector boson X in the theory. The corresponding propagators `Dtran` and `Dlong` have to be replaced by the user with their setup-dependent implementation of Eq. (A.2) and Eq. (A.3) in a later step. In particular, it is easy to return to a specific choice of gauge, e.g. by simply setting `Dlong` to zero and

$$\mathcal{D}_{\mathrm{T},\mu\nu}(k, M, \xi = 1) = \frac{-\mathrm{i}g_{\mu\nu}}{k^2 - M^2} \tag{A.5}$$

in the case of Feynman gauge $\xi = 1$.

Apart from this conceptually important aspect, the `UFOReader` acts essentially as a `python`-to-`tapir` parser, i.e. it makes some notational changes to the objects and fields that appear.

Appendix A.2. Vertices

Similarly to the propagators, the vertices also almost exclusively undergo some parsing, with two noteworthy exceptions.

The first point concerns the general R_ξ gauge that is restored by the `UFOReader`. In order to restore the correct gauge dependence of vertices, a few vertices have to carry an overall factor of the corresponding gauge parameter ξ. This factor is simply prefixed to the `Lorentz` fold string of the corresponding vertex in the `vrtx` file. As an example, consider the vertex of the ghost corresponding to the Z-boson, the ghost corresponding to the W^+-boson and the negatively charged goldstone boson. This vertex is proportional to the gauge parameter of the Z-boson and takes the form

```
{cghZ,CghWp,Gp: *<gauge_parameter_xiZ> *( ufoGC95 * ( 1 )) | *(1) | | }
```

In addition, since massive vector bosons are split into two parts (see Appendix A.1), for each possible combination of "transversal" and "longitudinal" propagator components entering a vertex, an identical copy of the vertex Feynman rule is generated.

Secondly, the factorisation of a Feynman rule into a `Lorentz` part and a `QCD` part is not always trivial, e.g. the four-gluon vertex of QCD. A common approach is to introduce an auxiliary *sigma particle* (cf. Subsection 4.1) and write the four-gluon vertex as a linear combination of several sigma-exchange diagrams. In the `UFOReader`, we follow a less QCD-tailored approach, based on Ref. [64]. In Feynman rules of non-factorising vertices we introduce an auxiliary function `nonfactag()` with three arguments. For example, the rule for the four-gluon vertex reads

```
{g,g,g,g:*( ufoGC12 * (
    d_(<lorentz_index_particle_1>,<lorentz_index_particle_4>)
    *d_(<lorentz_index_particle_2>,<lorentz_index_particle_3>)
    -d_(<lorentz_index_particle_1>,<lorentz_index_particle_2>)
    *d_(<lorentz_index_particle_3>,<lorentz_index_particle_4>)
  )*nonfactag(0,1,<local_index_F>)
+  ufoGC12 * (
    d_(<lorentz_index_particle_1>,<lorentz_index_particle_4>)
    *d_(<lorentz_index_particle_2>,<lorentz_index_particle_3>)
    -d_(<lorentz_index_particle_1>,<lorentz_index_particle_3>)
    *d_(<lorentz_index_particle_2>,<lorentz_index_particle_4>)
  ) *nonfactag(0,0,<local_index_F>)
+  ufoGC12 * (
    d_(<lorentz_index_particle_1>,<lorentz_index_particle_3>)
    *d_(<lorentz_index_particle_2>,<lorentz_index_particle_4>)
    -d_(<lorentz_index_particle_1>,<lorentz_index_particle_2>)
    *d_(<lorentz_index_particle_3>,<lorentz_index_particle_4>)
  ) *nonfactag(0,2,<local_index_F>))
|*( ufocomplex(0,1)
  *V3g(<colour_index_particle_-1>,<colour_index_particle_1>,
    <colour_index_particle_2>)*ufocomplex(0,1)
  *V3g(<colour_index_particle_3>,<colour_index_particle_4>,
    <colour_index_particle_-1>)*nonfactag(0,0,<local_index_F>)
+ ufocomplex(0,1)
  *V3g(<colour_index_particle_-1>,<colour_index_particle_1>,
    <colour_index_particle_3>)*ufocomplex(0,1)
  *V3g(<colour_index_particle_2>,<colour_index_particle_4>,
    <colour_index_particle_-1>)*nonfactag(0,1,<local_index_F>)
+ ufocomplex(0,1)
  *V3g(<colour_index_particle_-1>,<colour_index_particle_1>,
    <colour_index_particle_4>)*ufocomplex(0,1)
  *V3g(<colour_index_particle_2>,<colour_index_particle_3>,
    <colour_index_particle_-1>)*nonfactag(0,2,<local_index_F>))
||}
```

There are three terms inside the `Lorentz` fold, each coming with a factor of `nonfactag()`. These factors differ in the second argument of `nonfactag()`, which labels the term inside

the Feynman rule, but share the same third argument that is replaced by an index, unique to the position of the vertex in a given Feynman diagram by `tapir`. The `QCD` fold is structured in the same way. The first argument of `nonfactag()` counts the number of different non-factorising Feynman rules, starting from zero. When computing diagrams involving a non-factorising vertex, both, the color factor and the `Lorentz` part, can be calculated independently. When multiplying both results, we can set products of `nonfactag()` to 0 if they correspond to the same vertex but different terms or to 1 if they correspond to the same vertex and the same term. This can be achieved by two simple `FORM` replacement rules:

```
*** First set matching products to 1
id nonfactag(a?,b?,c?)*nonfactag(a?,b?,c?) = 1;
*** All remaining products corresponding to the same vertex vanish
id nonfactag(a?,b?,c?)*nonfactag(d?,e?,c?) = 0;
```

References

[1] K. G. Chetyrkin, F. V. Tkachov, Integration by Parts: The Algorithm to Calculate beta Functions in 4 Loops, Nucl. Phys. B 192 (1981) 159–204. doi:10.1016/0550-3213(81)90199-1.

[2] F. V. Tkachov, A Theorem on Analytical Calculability of Four Loop Renormalization Group Functions, Phys. Lett. B 100 (1981) 65–68. doi:10.1016/0370-2693(81)90288-4.

[3] C. Anastasiou, K. Melnikov, Higgs boson production at hadron colliders in NNLO QCD, Nucl. Phys. B 646 (2002) 220–256. arXiv:hep-ph/0207004, doi:10.1016/S0550-3213(02)00837-4.

[4] P. Nogueira, Automatic Feynman graph generation, J. Comput. Phys. 105 (1993) 279–289. doi:10.1006/jcph.1993.1074.

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

[6] T. Hahn, Generating Feynman diagrams and amplitudes with FeynArts 3, Comput. Phys. Commun. 140 (2001) 418–431. arXiv:hep-ph/0012260, doi:10.1016/S0010-4655(01)00290-9.

[7] R. Mertig, M. Böhm, A. Denner, FEYN CALC: Computer algebraic calculation of Feynman amplitudes, Comput. Phys. Commun. 64 (1991) 345–359. doi:10.1016/0010-4655(91)90130-D.

[8] V. Shtabovenko, R. Mertig, F. Orellana, New Developments in FeynCalc 9.0, Comput. Phys. Commun. 207 (2016) 432–444. arXiv:1601.01167, doi:10.1016/j.cpc.2016.06.008.

[9] V. Shtabovenko, R. Mertig, F. Orellana, FeynCalc 9.3: New features and improvements, Comput. Phys. Commun. 256 (2020) 107478. arXiv:2001.04407, doi:10.1016/j.cpc.2020.107478.

[10] B. Ruijl, T. Ueda, J. Vermaseren, FORM version 4.2 (7 2017). arXiv:1707.06453.

[11] N. D. Christensen, P. de Aquino, C. Degrande, C. Duhr, B. Fuks, M. Herquet, F. Maltoni, S. Schumann, A Comprehensive approach to new physics simulations, Eur. Phys. J. C 71 (2011) 1541. arXiv:0906.2474, doi:10.1140/epjc/s10052-011-1541-5.

[12] A. Alloul, N. D. Christensen, C. Degrande, C. Duhr, B. Fuks, FeynRules 2.0 - A complete toolbox for tree-level phenomenology, Comput. Phys. Commun. 185 (2014) 2250–2300. arXiv:1310.1921, doi:10.1016/j.cpc.2014.04.012.

[13] C. Degrande, C. Duhr, B. Fuks, D. Grellscheid, O. Mattelaer, T. Reiter, UFO - The Universal FeynRules Output, Comput. Phys. Commun. 183 (2012) 1201–1214. arXiv:1108.2040, doi:10.1016/j.cpc.2012.01.022.

[14] 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) 151–207. doi:10.1016/0550-3213(79)90234-7.

[15] H. H. Patel, Package-X: A Mathematica package for the analytic calculation of one-loop integrals, Comput. Phys. Commun. 197 (2015) 276–290. arXiv:1503.01469, doi:10.1016/j.cpc.2015.08.017.

[16] M. Wiebusch, HEPMath 1.4: A mathematica package for semi-automatic computations in high energy physics, Comput. Phys. Commun. 195 (2015) 172–190. arXiv:1412.6102, doi:10.1016/j.cpc.2015.04.022.

[17] A. Denner, S. Dittmaier, L. Hofer, Collier: a fortran-based Complex One-Loop LIbrary in Extended Regularizations, Comput. Phys. Commun. 212 (2017) 220–238. arXiv:1604.06792, doi:10.1016/j.cpc.2016.10.013.

[18] J. Alwall, R. Frederix, S. Frixione, V. Hirschi, F. Maltoni, O. Mattelaer, H. S. Shao, T. Stelzer, P. Torrielli, M. Zaro, The automated computation of tree-level and next-to-leading order differential cross sections, and their matching to parton shower simulations, JHEP 07 (2014) 079. arXiv:1405.0301, doi:10.1007/JHEP07(2014)079.

[19] G. Belanger, F. Boudjema, J. Fujimoto, T. Ishikawa, T. Kaneko, K. Kato, Y. Shimizu, Automatic calculations in high energy physics and Grace at one-loop, Phys. Rept. 430 (2006) 117–209. arXiv:hep-ph/0308080, doi:10.1016/j.physrep.2006.02.001.

[20] G. Cullen, N. Greiner, G. Heinrich, G. Luisoni, P. Mastrolia, G. Ossola, T. Reiter, F. Tramontano, Automated One-Loop Calculations with GoSam, Eur. Phys. J. C 72 (2012) 1889. arXiv:1111.2034, doi:10.1140/epjc/s10052-012-1889-1.

[21] G. Cullen, et al., G*OSAM*-2.0: a tool for automated one-loop calculations within the Standard Model and beyond, Eur. Phys. J. C 74 (8) (2014) 3001. arXiv:1404.7096, doi:10.1140/epjc/s10052-014-3001-5.

[22] M. Tentyukov, J. Fleischer, A Feynman diagram analyzer DIANA, Comput. Phys. Commun. 132 (2000) 124–141. arXiv:hep-ph/9904258, doi:10.1016/S0010-4655(00)00147-8.

[23] 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) 125–132. arXiv:hep-ph/9712228, doi:10.1016/S0370-2693(98)00220-2.

[24] 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.

[25] R. Harlander, T. Seidensticker, M. Steinhauser, q2e/exp (1997).
URL http://sfb-tr9.ttp.kit.edu/software/html/q2eexp.html

[26] J. Hoff, The Mathematica package TopoID and its application to the Higgs boson production cross section, J. Phys. Conf. Ser. 762 (1) (2016) 012061. arXiv:1607.04465, doi:10.1088/1742-6596/762/1/012061.

[27] V. Margerya, Alibrary (2021).
URL https://github.com/magv/alibrary

[28] V. Margerya, feynson (2020).
URL https://github.com/magv/feynson/

[29] 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). doi:10.5445/IR/1000125521.

[30] F. Feng, `Apart`: A Generalized Mathematica Apart Function, Comput. Phys. Commun. 183 (2012) 2158–2164. arXiv:1204.2314, doi:10.1016/j.cpc.2012.03.025.

[31] M. Heller, A. von Manteuffel, MultivariateApart: Generalized partial fractions, Comput. Phys. Commun. 271 (2022) 108174. arXiv:2101.08283, doi:10.1016/j.cpc.2021.108174.

[32] R. N. Lee, LiteRed 1.4: a powerful tool for reduction of multiloop integrals, J. Phys. Conf. Ser. 523 (2014) 012059. arXiv:1310.1145, doi:10.1088/1742-6596/523/1/012059.

[33] S. A. Larin, F. V. Tkachov, J. A. M. Vermaseren, The FORM version of MINCER (9 1991).

[34] M. Steinhauser, MATAD: A Program package for the computation of MAssive TADpoles, Comput. Phys. Commun. 134 (2001) 335–364. arXiv:hep-ph/0009029, doi:10.1016/S0010-4655(00)00204-6.

[35] 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. arXiv:1704.06650, doi:10.1016/j.cpc.2020.107198.

[36] A. Pikelner, FMFT: Fully Massive Four-loop Tadpoles, Comput. Phys. Commun. 224 (2018) 282–287. arXiv:1707.01710, doi:10.1016/j.cpc.2017.11.017.

[37] A. V. Smirnov, Algorithm FIRE – Feynman Integral REduction, JHEP 10 (2008) 107. arXiv:0807.3243, doi:10.1088/1126-6708/2008/10/107.

[38] A. V. Smirnov, F. S. Chukharev, FIRE6: Feynman Integral REduction with Modular Arithmetic, Comput. Phys. Commun. 247 (2020) 106877. arXiv:1901.07808, doi:10.1016/j.cpc.2019.106877.

[39] C. Studerus, Reduze - Feynman Integral Reduction in C++, Comput. Phys. Commun. 181 (2010) 1293–1300. arXiv:0912.2546, doi:10.1016/j.cpc.2010.03.012.

[40] A. von Manteuffel, C. Studerus, Reduze 2 - Distributed Feynman Integral Reduction (1 2012). arXiv:1201.4330.

[41] P. Maierhöfer, J. Usovitsch, P. Uwer, Kira—A Feynman integral reduction program, Comput. Phys. Commun. 230 (2018) 99–112. arXiv:1705.05610, doi:10.1016/j.cpc.2018.04.012.

[42] J. Klappert, F. Lange, P. Maierhöfer, J. Usovitsch, Integral reduction with Kira 2.0 and finite field methods, Comput. Phys. Commun. 266 (2021) 108024. arXiv:2008.06494, doi:10.1016/j.cpc.2021.108024.

[43] J. S. Hoff, Methods for multiloop calculations and Higgs boson production at the LHC, Ph.D. thesis, KIT, Karlsruhe (2015). doi:10.5445/IR/1000047447.

[44] B. Nickel, D. Meiron, G. A. J. Baker, Compilation of 2-pt and 4-pt graphs for continuous spin model, University of Guelph Report (1977).

[45] B. D. McKay, A. Piperno, Practical graph isomorphism, ii, Journal of Symbolic Computation 60 (2014) 94–112. doi:https://doi.org/10.1016/j.jsc.2013.09.003.

[46] D. Batkovich, Y. Kirienko, M. Kompaniets, S. Novikov, GraphState - a tool for graph identification and labellingarXiv:1409.8227.

[47] A. Pak, The Toolbox of modern multi-loop calculations: novel analytic and semi-analytic techniques, J. Phys. Conf. Ser. 368 (2012) 012049. arXiv:1111.0868, doi:10.1088/1742-6596/368/1/012049.

[48] C. Bogner, S. Weinzierl, Feynman graph polynomials, Int. J. Mod. Phys. A 25 (2010) 2585–2618. arXiv:1002.3458, doi:10.1142/S0217751X10049438.

[49] A. V. Smirnov, UF.
URL https://www.ttp.kit.edu/~asmirnov/Tools-UF.htm

[50] G. Heinrich, S. Jahn, S. P. Jones, M. Kerner, F. Langer, V. Magerya, A. Pöldaru, J. Schlenk, E. Villa, Expansion by regions with pySecDec (8 2021). arXiv:2108.10807.

[51] B. Buchberger, A theoretical basis for the reduction of polynomials to canonical forms, SIGSAM Bull. 10 (3) (1976) 19–29. doi:10.1145/1088216.1088219.

[52] E. W. Mayr, A. R. Meyer, The complexity of the word problems for commutative semigroups and polynomial ideals, Advances in Mathematics 46 (3) (1982) 305–329. doi:https://doi.org/10.1016/0001-8708(82)90048-2.

[53] A. Meurer, C. P. Smith, M. Paprocki, O. Čertík, S. B. Kirpichev, M. Rocklin, A. Kumar, S. Ivanov, J. K. Moore, S. Singh, T. Rathnayake, S. Vig, B. E. Granger, R. P. Muller, F. Bonazzi, H. Gupta, S. Vats, F. Johansson, F. Pedregosa, M. J. Curry, A. R. Terrel, v. Roučka, A. Saboo, I. Fernando, S. Kulal, R. Cimrman, A. Scopatz, Sympy: symbolic computing in python, PeerJ Computer Science 3 (2017) e103. doi:10.7717/peerj-cs.103.

[54] B. Sturmfels, Grobner Bases—a Computational Approach to Commutative Algebra (Thomas Becker and Volker Weispfenning), SIAM Review 36 (2) (1994) 323–323. arXiv:https://doi.org/10.1137/1036089, doi:10.1137/1036089.

[55] R. E. Cutkosky, Singularities and discontinuities of Feynman amplitudes, J. Math. Phys. 1 (1960) 429–433. doi:10.1063/1.1703676.

[56] J. Davies, F. Herren, G. Mishima, M. Steinhauser, Real-virtual corrections to Higgs boson pair production at NNLO: three closed top quark loops, JHEP 05 (2019) 157. arXiv:1904.11998, doi:10.1007/JHEP05(2019)157.

[57] J. Davies, F. Herren, G. Mishima, M. Steinhauser, Real corrections to Higgs boson pair production at NNLO in the large top quark mass limit (10 2021). arXiv:2110.03697.

[58] A. Pak, `gen`, unpublished.

[59] J. Ellis, TikZ-Feynman: Feynman diagrams with TikZ, Comput. Phys. Commun. 210 (2017) 103–123. arXiv:1601.05437, doi:10.1016/j.cpc.2016.08.019.

[60] A. Grozin, Lectures on QED and QCD, in: 3rd Dubna International Advanced School of Theoretical Physics, 2005. arXiv:hep-ph/0508242.

[61] M. Beneke, G. Buchalla, C. Greub, A. Lenz, U. Nierste, Next-to-leading order QCD corrections to the lifetime difference of B(s) mesons, Phys. Lett. B 459 (1999) 631–640. arXiv:hep-ph/9808385, doi:10.1016/S0370-2693(99)00684-X.

[62] M. Gerlach, U. Nierste, V. Shtabovenko, M. Steinhauser, Two-loop QCD penguin contribution to the width difference in B_s–$\overline{B}_s$ mixing, JHEP 07 (2021) 043. arXiv:2106.05979, doi:10.1007/JHEP07(2021)043.

[63] 2to3 - Automated Python 2 to 3 code translation.
URL https://docs.python.org/3/library/2to3.html

[64] J. Davies, F. Herren, G. Mishima, M. Steinhauser, NNLO real corrections to $gg \to HH$ in the large-m_t limit, PoS RADCOR2019 (2019) 022. arXiv:1912.01646, doi:10.22323/1.375.0022.