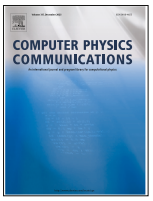
Computer Physics Communications 320 (2026) 109935

Contents lists available at ScienceDirect

Computer Physics Communications

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



Computer Programs in Physics

Automation of a matching on-shell calculator

Javier López Miras, Fuensanta Vilches *

Departamento de Física Teórica y del Cosmos, Universidad de Granada, E-18071, Granada, Spain

ARTICLE INFO

Editor: Prof. Z. Was

Keywords:
Effective field theory
Matching
On-shell methods
Field redefinitions

ABSTRACT

We introduce `mosca`, a MATHEMATICA package designed to facilitate on-shell calculations in effective field theories (EFTs). This initial release focuses on the reduction of Green's bases to physical bases, as well as transformations between arbitrary operator bases. The core of the package is based on a diagrammatic on-shell matching procedure, grounded in the equivalence of physical observables derived from both redundant and non-redundant Lagrangians. `mosca` offers a complete set of tools for performing basis transformations, diagram isomorphism detection, numerical substitution of kinematic configurations, and symbolic manipulation of algebraic expressions. Planned future developments include extension to one-loop computations, thus providing support for EFT renormalization directly in a physical basis and automated computation of one-loop finite matching, including contributions from evanescent operators.

PROGRAM SUMMARY

Program Title: mosca
CPC Library link to program files: https://doi.org/10.17632/xf8zs7hnhm.1
Developer's repository link: https://gitlab.com/matchingonshell/mosca
Licensing provisions: GPLv3
Programming language: Mathematica
Nature of problem: Matching calculations in effective field theories are traditionally performed off-shell, involving complicated basis reductions through non-trivial field redefinitions to eliminate redundant operators. This process is algebraically intensive and prone to errors. Although on-shell matching, which focuses directly on physical observables, could simplify these steps by avoiding field redefinitions, it has been considered impractical due to the presence of apparent non-localities that must cancel precisely. Automating on-shell matching has therefore been a long-standing challenge.
Solution method: Our approach is based on a numerical solution of the on-shell matching equations, which naturally and effortlessly enforces the delicate cancellation of non-local terms between the full theory and the effective theory. By employing rational on-shell kinematics, the method achieves an exact analytic solution despite using numerical techniques. This allows the matching to be performed entirely within a physical operator basis.
Additional comments including restrictions and unusual features: The workflow for handling Lagrangians and Feynman diagrams in mosca is based on the integration of FeynArts and FeynCalc. Consequently, users need to provide specific FeynArts model files patched for compatibility with FeynCalc. Additionally, a specialized input format is required to define Wilson coefficients along with their corresponding EFT order (EFTOrder). These requirements ensure the correct processing of models and coefficients.

1. Introduction

Effective field theories (EFTs), which have been used since the very foundations of quantum field theory, have become one of the clearest directions to choose in beyond the Standard Model (SM) research. With the rise in prominence of this field, more and more ultraviolet (UV) models are to be matched to their respective EFTs, *e.g.* in order to study their physical implications at low energy levels (see [1–8] for some applications). Furthermore, after an initial understanding of their effects based on rough approximations, studies have shifted towards the inspection of the consequences when the full set of effective operators is considered. Well-known complete bases for some common EFTs in the literature can

* Corresponding author.
E-mail address: fuenvilches@ugr.es (F. Vilches).

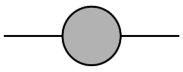
https://doi.org/10.1016/j.cpc.2025.109935
Received 30 July 2025; Received in revised form 28 October 2025; Accepted 1 November 2025
Available online 19 November 2025

be found in [9–30]. This includes accounting for their precise coupling dependencies and exploring the impact of extending the EFT to higher dimensions. Hence, a consistent, reliable and efficient matching procedure is essential in order to progress with the EFT program.

In the functional approach to matching, a functional integration over the heavy states allows the extraction of the local contributions that are relevant for the description of the low-energy dynamics. These local contributions correspond to operators which are typically not in a physical basis and require reduction via field redefinitions [31,32], which can be tedious at higher orders but can be automated with tools like `Matchete` [33]. On the other hand, the diagrammatic approach (automated in `MatchMakerEFT` [34]) offers two distinct implementation strategies. The first involves performing an off-shell matching, where off-shell Green's functions are computed in both the UV model and the EFT; and their difference is matched by adjusting the Wilson coefficients (WCs) of the local operators in the EFT. Although this first method is advantageous due to the smaller number of diagrams (only the 1-light-particle-irreducible should be considered), it still requires a Green's basis with redundant operators that must be reduced. The second strategy, in contrast, avoids the need for a Green's basis and allows for the direct use of physical operators by performing an on-shell matching between both theories [35–37]. However, it faces challenges due to the larger number of diagrams and the non-local nature of light bridges, making it difficult to manage analytic cancellations. In [38] it was showed the implementation of a numerical (but exact) approach that efficiently addresses these challenges, proving the potential of on-shell matching.

In this work, we adopt this latter approach and automate it in a MATHEMATICA code. For the sake of completeness, we shall first summarize here the algorithm for on-shell matching, as presented in [38]:

1. Determination of the physical masses and residues that fix the on-shell conditions: The light propagators in the UV theory read



$$= \frac{i}{p^2 - m^2 - \Pi(p^2)} \approx \frac{iZ}{p^2 - m^2_{phys}}\,, \tag{1.1}$$

   where $\Pi(p^2)$ is the 1-particle-irreducible (1PI) contribution to the 2-point functions. In the last equality, we approximate the propagator in the vicinity of the physical mass, which corresponds to the pole of the 2-point function. The parameter Z represents the residue at the physical pole and can be determined by expanding $\Pi(p^2)$ near $p^2 = m^2_{phys}$ in Eq. (1.1). Through this expansion, we find that $Z = \left(1 - \Pi'(m^2_{phys})\right)^{-1}$. In the EFT, there are no 1PI corrections to the 2-point function so $m^2_{phys} = m^2$ and $Z = 1$.
2. Calculation of amplitudes: Evaluate all necessary connected, amputated amplitudes, up to the correct order in mass dimension, in both the UV model and the EFT. Then, substitute into the amplitudes the physical mass and complete propagators, and multiply by a factor $\sqrt{Z}$ for each external leg, according to the LSZ reduction formula.
3. Solve for the WCs: Start with the simplest amplitudes (fewest external particles) and solve for the EFT WCs using on-shell randomly generated kinematic configurations. At tree level, matching equates the tree-level amplitudes of the EFT and the UV theory.

Furthermore, the generation of on-shell numerical kinematics with rational values is based on the algorithm presented in [39]. This method offers two key advantages: first, the use of rational kinematics ensures exact solutions; second, it allows for the enforcement of independent symbolic masses for each particle, a crucial feature to capture the mass dependence in the matching procedure.

The implementation of such algorithm, which makes use of the so-called *spinor-helicity* formalism [40–43], can be found in a MATHEMATICA package in https://github.com/StefanoDeAngelis/SpinorHelicity/tree/main/Wolfram/NumericalKinematics.wl. This package outputs both the set of physical momenta and the corresponding associated spinors. However, for processes involving massless particles of spin-1, we also require polarization vectors. Using the spinor-helicity formalism, we derive polarization vectors as follows:

$$\epsilon^\nu_+ = \frac{1}{\sqrt{2}} \frac{\lambda^\alpha \widetilde{\mu}^{\dot\alpha} \sigma^\nu_{\alpha\dot\alpha}}{\lambda_\beta \mu^\beta} \quad \text{and} \quad \epsilon^\nu_- = \frac{1}{\sqrt{2}} \frac{\mu^\alpha \widetilde{\lambda}^{\dot\alpha} \sigma^\nu_{\alpha\dot\alpha}}{\widetilde{\lambda}_{\dot\beta} \widetilde{\mu}^{\dot\beta}},$$

where $(\mu, \widetilde{\mu})$ are auxiliary spinors, $(\lambda, \widetilde{\lambda})$ are the spinors corresponding to the momentum of the spin-1 particle, and $\sigma^\nu = (1_{2\times2}, \sigma^I)$, with σ^I the Pauli matrices. These expressions inherently satisfy the transversality conditions, ensuring they represent physical polarization states. The spinors lower and rise their indices as $\lambda^\alpha \equiv \epsilon^{\alpha\beta}\lambda_\beta$, with $\epsilon^{\alpha\beta}$ the totally antisymmetric tensor with $\epsilon^{12} = 1$. Furthermore, for spin-$\frac{1}{2}$ particles, spinors satisfying the Dirac equation are also required. In the case of incoming massless fermions, the relationship between left- and right-handed Dirac spinors and the Weyl spinor basis is established through the following relations:

$$P_L u(p) = \begin{pmatrix} \widetilde{\lambda}_{\dot\alpha} \\ 0 \end{pmatrix}, \quad P_R u(p) = \begin{pmatrix} 0 \\ \lambda^\alpha \end{pmatrix}, \quad \overline{v}(p) P_R = \begin{pmatrix} \widetilde{\lambda}^{\dot\alpha} \\ 0 \end{pmatrix} \quad \text{and}$$

$$\overline{v}(p) P_L = \begin{pmatrix} 0 \\ \lambda_\alpha \end{pmatrix},$$

where $(\lambda, \widetilde{\lambda})$ represent the spinors associated with the four-momentum p of the spin-1/2 particle. Once all the kinematic structures are generated, we substitute them into the amplitudes, incorporating the numerical values of the gamma matrices in the Dirac representation. This yields a final expression where the WCs—the couplings of local operators in the effective Lagrangian of the EFT—are the only remaining unknowns.

After having set the details of the on-shell matching procedure, we turn now to the scope of this manual: `mosca`. The MATHEMATICA code `mosca` (`Matching On-Shell Calculator`) is engineered to proceed with the aforementioned routine to implement on-shell matching. The general lines of the workflow of `mosca` are summarized in Fig. 1.

Until now, the discussion of the on-shell matching procedure has focused on its traditional application in the construction of an EFT that reproduces the low-energy behavior of a more fundamental theory. However, in this initial version, `mosca` sets the usual matching of UV models aside and focuses on the matching of different Lagrangian representations of the same theory. The foundational principle of on-shell matching relies in explicitly ensuring that two Lagrangians provide identical S-matrix elements for all computable processes, up to the corresponding order in the EFT expansion. In the context of basis reduction, this same logic can be applied, since this process involves merely redefining the Lagrangian representation of the theory without altering the underlying physics. In this reinterpretation, instead of comparing the Lagrangians of a UV theory and an EFT, we compare two Lagrangian representations of the same theory expressed in terms of different operator bases. As argued more formally in [38], a tree-level on-shell matching can be used to relate the WCs of a Lagrangian $\mathcal{L}_1$ with respect to other $\mathcal{L}_2$, as long as there exists a local redefinition of the fields transforming $\mathcal{L}_1$ into $\mathcal{L}_2$.

As previously noted, computations with EFTs (*e.g.* renormalization group evolutions or finite matching) often require a (on-shell redundant) Green's basis which is later to be reduced to the much smaller and manageable physical basis. Traditionally this has been done by simply replacing the equations of motion (EOMs) into the redundant Lagrangian [44–47]. However, this is known to be just a mere approximation beyond the linear order in the EFT cut-off Λ [32]. The right way to do this is via field redefinitions[1] which, while straightforward in principle, can turn very tedious and cumbersome. Thus, a simple tree-level on-shell matching between the Lagrangians expressed in terms of some bases is a promising alternative to automate such process. In light of this, `mosca`

[1] Other methods such as modified equations of motions can also be considered [48].

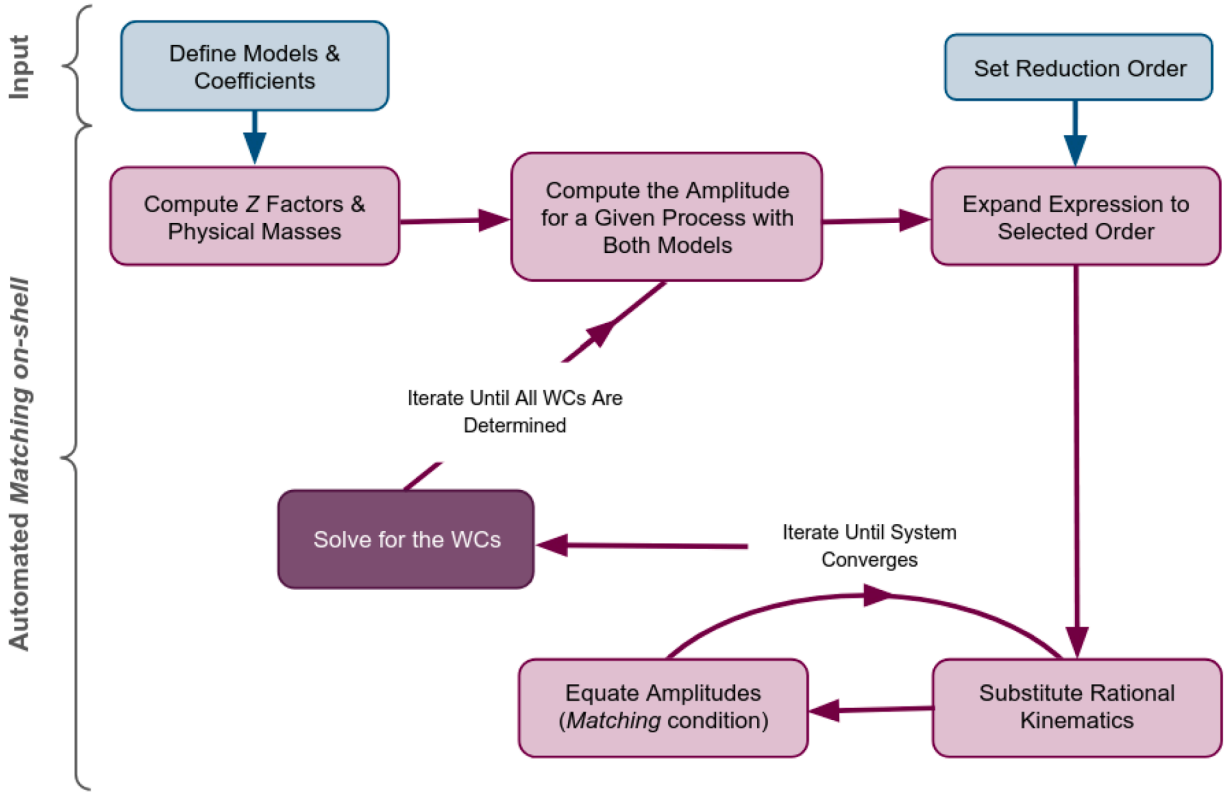


Fig. 1. Schematic representation of `mosca` roadmap.

is designed to facilitate the process of reducing Lagrangians by performing a matching based on on-shell techniques.

This paper serves as a concise introduction to `mosca`, providing an overview of the package's structure, main functions and practical examples. With this objective, the rest of this article is organized as follows: First, Section 2 presents the installation process and an overview of the structure of the package, including key aspects and guidance for setting up the input. Next, Section 3 introduces the main objects and functions, focusing on each part of the on-shell matching procedure. Finally, Section 4 is dedicated to discuss several aspects related to the use of `mosca`. In Section 5 we conclude.

2. Getting started: installation and package overview

2.1. First time running mosca

The `mosca` package can be downloaded from the GitLab repository: https://gitlab.com/matchingonshell/mosca
The `mosca` folder includes the main program (`mosca.m`) along with all the essential tools needed to run the package. Among the different files, you will find the `ExternalPackages/` directory, which includes the following dependencies:

- `FeynCalc` [49]: A MATHEMATICA library for symbolic manipulation of Feynman integrals and expressions in quantum field theory.
- `FeynArts` [50]: A package for generating Feynman diagrams, which is included within the `FeynCalc` folder.
- `NumericalKinematics`: The package that provides the framework necessary for defining and managing kinematic variables.

The `NumericalKinematics` package is used by `KinematicSubstitution`, another package located in a folder of the same name. This package constructs all the necessary physical kinematic configurations, as described in Section 1.

The minimum requirement for running `mosca` is having installed MATHEMATICA (version 8 or later, although it has been only tested from version 12 onward). The necessary `FeynCalc` and `FeynArts` are distributed within the package installation but, just for completeness, their versions are 10.1.0 *(stable version)* and 3.12, respectively. Note that the `FeynArts` version we use is the one directly provided with the `FeynCalc` installation and, therefore, it is already patched to `FeynCalc`.

Before running `mosca`, the user must specify the directory where the folder containing the package is located. You can do this by executing the following command:
After setting the directory, load `mosca` by running the command:
If everything is correct, you should see a GIF animation[2] on the screen drawing the `mosca` logo:
You are now ready to start flying your `mosca`[3]

2.2. Setting up the input

`mosca` relies on the computation of Feynman diagrams. Hence, the first step consists on specifying a field content and an interaction Lagrangian. This is what we call a *model*, and its structure follows that of `FeynArts`. The most straightforward way to create the models is to use the `WriteFeynArtsOutput[]` function from `Feynrules`. When doing so, it is essential to have set the `CouplingRename` option to `False`. By default, this option is set to `True`, which causes all amplitudes to be expressed in terms of unknown new variables that the package cannot process correctly. Additionally, to ensure proper handling of bilinear terms in the Feynman rules, such as 2-point functions, make sure that `FR$Loop=True` is also enabled. Once the models are generated, the `FAPatch[]` function from `FeynCalc` can be used to patch them, ensuring compatibility with `FeynCalc`. If the version of `FeynArts` is already

[2] You can disable the GIF animation by typing `$LogoAnimation=False` before loading `mosca`.
[3] In Spanish, *mosca* means "fly".

```
In[1]:= SetDirectory["path/to/mosca"];
```

```
In[2]:= << mosca`
```

Extract of the 2-pt interaction kinematics in the `.gen` file

```
(* SS *)
AnalyticalCoupling[s1 S[j1, mom1], s2 S[j2, mom2] ] == G[+1][s1 S[j1], s2
S[j2]].{
   1,
   FAScalarProduct[mom1,mom2],
   FAScalarProduct[mom1,mom1]FAScalarProduct[mom2,mom2] }
```

Extract of the 2-pt interaction coupling in the `.mod` file

```
C[ S[1] , S[1] ] == {
   {0, (-I)*m^2},
   {0, -I},
   {0, (2*I)*rD\[Phi]} }
```

patched, the option PatchModelsOnly−>True can be used to apply the patch exclusively to the model files.

By default, the 2-point function generation at tree-level via Feynrules requires a special modification of the models. These modifications have been fully automated within mosca. In particular, the list M$CouplingMatrices from the model file (.mod), must be adapted. Each element of this list consists of a set of couplings corresponding to a certain Feynman rule or vertex. The set of couplings is arranged in turn in another list where each entry corresponds to a counter-term (CT) order. By default, the FeynArts model output generated from Feynrules does not include couplings for the 2-point function, even when FR$Loop is set to True (this option adds the kinematic structures in the .gen file, but not the coupling vector counterpart in the .mod). However, there is a minor change that one can make in the FeynArtsInterface.m file of Feynrules: wherever the comment (*Remove the tree-level component for the vertices with two legs or less*) appears, one has to replace > with >= in the subsequent $If[Length[\#1] > 2, \ldots]$ command. By doing this, the coupling vectors of 2-point vertices will be automatically written in the .mod file as 0th-order CTs. Otherwise, the user would have to provide these entries by hand.

Furthermore, the CreateTopologies[] function does not support a 1->1 process at 0th-order CT. To address this, the .mod file must be modified by placing all coupling vectors corresponding to 2-point vertices under the entry associated with CTs of order 1 in the M$CouplingMatrices. If we consider the following Lagrangian for a real scalar singlet ϕ with Z_2 symmetry:

$$\mathcal{L}_\phi = -\frac{1}{2}\phi(\partial^2 + m^2)\phi + r_{D\phi}\partial^2\phi\partial^2\phi, \tag{2.1}$$

the corresponding entry in the M$GenericCouplings found in the .gen file should be given by:
On the other hand, the associated entry in the M$CouplingMatrices list in the .mod file should be:

Here, the couplings for each 2-point interaction are placed in the second position of the list, ensuring they are treated as 1st-order CTs. The Patch2ptFunction[model] function in mosca automates this procedure, taking the 2-point coupling vectors from a given model.mod file and applying the necessary modification. This function can be executed alongside other patch functions using Patch2mosca[model]. The Patch2mosca[model] command also includes the functionalities of PatchMasses[model], which assigns a unique symbolic internal mass to every field (massless or not) to ensure proper differentiation within the calculations, together with the FeynCalc FAPatch[].

To ensure the correct operation of all functions, strict naming conventions must be followed. The .gen and .mod files must share the same base name, omitting the file extension when referring to them. To simplify the access, it is convenient to define a variable that specifies the path to these files. For instance, the command
can be used to reference the files model1.gen and model1.mod, which are stored in the model1 folder within the Models folder directory specified by path. From now on, when we refer to a model in a MATHEMATICA context (*e.g.*, as arguments of functions) we mean exactly this: the string containing the path where the model .gen and .mod files are found.

Once all these considerations are implemented, the final step is to define the couplings present in the models. This is achieved by using the following function:
which takes two inputs, namely, the name of a coefficient and the model it is associated with, and two optional arguments: the corresponding EFT order and whether the coefficient is complex. Hereafter, the EFT order X of the dimensionless coupling c, appearing in the Lagrangian as $\frac{c}{\Lambda^n}\mathcal{O}_i$, is understood to be $X = n + 4$, unless it is the WC of a renormalizable operator in which case $X = 4$ independently of the mass dimension of O_i. The function can also handle a list of coefficients, provided they all share the same EFTOrder and ComplexParameter settings. If these options are omitted, the function defaults to EFTOrder−>4 and ComplexParameter−>False. A more convenient way to define all the

```
In[3]:= model = FileNameJoin[{path, "Models", "model1", "model1"}];
```

```
DefineWC[coefficient, model, EFTOrder -> X, ComplexParameter -> True/False]
```

Example of parameter file to define the WCs of a model

```
paramList = {
  c1 == {
    ComplexParameter -> False,
    EFTOrder -> 4 },

  c2 == {
    ComplexParameter -> True,
    EFTOrder -> 6 },
  ...

};
```

```
DefineAllWCs[model, paramListPath]
```

```
FormatWC[model, ModelName -> "", VisibleModel -> True/False,
                VisibleOrder -> True/False]
```

coefficients in a model is by using a structured list. For instance, a real renormalizable coupling, c_1, and a complex dimension-6 coupling, c_2, can be defined via the following list:
This format is similar to the `M$Parameters` list from the `.fr` file used in `FeynRules` for model creation, which is very convenient in order to reuse its structure (note that any option other than `ComplexParameter` and `EFTOrder` is simply ignored). The name of the variable at which the list is assigned is unimportant (one could even write the list without any assignment), but this must be the only element present in the file. Then, the function
automates the process by taking the model and the path to the file containing `paramList`. It then extracts the necessary information for all coefficients. If any options are omitted, the function applies the default settings. Finally, by executing `WClist[model]` one can access the full list of defined WCs for the specified model. It is important to notice that WCs are interpreted in `mosca` as objects with head `WC[]` and three arguments, namely, the symbol of the coupling c, its EFT order X and the model it belongs to. We keep the models so that users can freely have equal coupling symbols in different Lagrangians, facilitating the creation of models and avoiding possible mix-ups. In the model creation, however, one should write the couplings simply with their symbol name c. The next section will cover how `mosca` converts symbols, c, into appropriate objects `WC[c,X,model]` and how to format and present the results in a readable manner.

2.3. Customizing the output

We first introduce various options for presenting the outputs. The primary function for this purpose is
which allows setting a specific format for all the WCs within a model. The function admits three optional inputs: the name of the model, provided as a string tagging the model of the coefficients; and two boolean options, `VisibleModel` and `VisibleOrder`. The first one controls whether the chosen model name is displayed as a subscript, while the second determines whether the EFT order of the coefficients is shown inside a parenthesis as a superscript. Both of these options are set to `True` by default. However, `VisibleModel` will not display any subscript if `ModelName` is not specified, since the default value for this input is an empty string.

As an example, let us print `WClist[modelPhi]` in a MATHEMATICA notebook, where `modelPhi` is the Lagrangian in Eq. (2.1). This list contains all the WCs of the Lagrangian, after having defined them via the `DefineWC[]` commands. If we do not run the `FormatWC[]` function, we obtain:
To properly format the WCs output, we use:
which now produces:
If we prefer to hide the EFT order in the output, we can set the `VisibleOrder` option to `False`:
Now, the coefficients are shown in the following way:
Similarly, to omit the model subscript, we can set `VisibleModel-> False`. In general, for the rest of the examples, we will assume both options are set to `False`, unless stated otherwise. One can always check the true form of the WCs with an `InputForm[]` command.

2.3.1. Treatment of coefficients

The function `RenameWC[model]` generates a substitution rule to rewrite the WCs of an expression into the proper format required for processing by `mosca`, this is, objects with head `WC[]`. For instance, suppose that a WC c has been defined, *e.g.* using `DefineWC[c, modelC, EFTOrder-> X]`. If we then write an expression containing c, say `Inputexp`, we can run:
to replace every occurrence of c in `exp` with the correctly formatted `WC[c, X, modelC]`.

Additionally, the function `ExplicitEFTOrder[exp]` multiplies each WC in `exp` by the corresponding powers of $\mathrm{inv}\Lambda$, based on its dimension. Applying both transformations in sequence to `exp = Ac + Bc^2` produces:
Here, we have left the output unformatted to clearly show that the WCs have been correctly replaced. This last function is described in detail in Section 3.6.

Another useful function is `TrueEFTOrder[]`, which determines the EFT order associated with a given WC. While this function may initially

```
In[4]:= WClist[modelPhi]

Out[4]= { WC[m, 4, "path-to-mosca/Models/modelPhi/modelPhi"],
          WC[rDϕ, 6, "path-to-mosca/Models/modelPhi/modelPhi"] }
```

```
In[5]:= FormatWC[modelPhi, ModelName->"modelϕ"];
```

```
In[6]:= WClist[modelPhi]

Out[6]= { m^(4)_modelϕ, rDϕ^(6)_modelϕ }
```

```
In[7]:= FormatWC[modelPhi, ModelName->"modelϕ", VisibleOrder->False];
```

```
Out[7]= { m_modelϕ, rDϕ_modelϕ }
```

```
In[8]:= exp /. RenameWC[modelC]
```

```
In[9]:= exp /. RenameWC[modelC] //ExplicitEFTOrder

Out[9]= A invΛ^X-4 WC[c, X, modelC] + B (invΛ^X-4)^2 WC[c, X, modelC]^2
```

seem unnecessary, its purpose becomes clear when understanding how `mosca` systematically solves the system of equations to find the redefinition of the WCs. As a brief preview, `mosca` performs the reduction process by solving for each coefficient order-by-order in the EFT expansion. Specifically, instead of solving directly for $c^{(X)}_{\text{modelC}}$, we express the coefficient as a sum of contributions at different orders:

$$\frac{1}{\Lambda^{X-4}}c^{(X)}_{\text{modelC}} \to \frac{1}{\Lambda^{X-4}}\left(c^{(X)}_{\text{modelC}} + \frac{1}{\Lambda}c^{(X+1)}_{\text{modelC}} + \frac{1}{\Lambda^2}c^{(X+2)}_{\text{modelC}} + \cdots + \frac{1}{\Lambda^{N-X-1}}c^{(N-1)}_{\text{modelC}} + \frac{1}{\Lambda^{N-X}}c^{(N)}_{\text{modelC}}\right),$$

where each $c^{(n)}_{\text{modelC}}$ (with $n \in [X, N]$, being N the order at which we truncate the EFT expansion) represents a distinct contribution to the redefinition of the WC c at dimension n. However, the EFT order of a WC is determined solely by the operator it accompanies and is independent of n. Then, given a specific coefficient $c^{(n)}_{\text{modelC}}$, we can retrieve its true EFT order by using:

2.4. Practical basic usage of mosca

The `mosca` package is distributed together with a set of example notebooks designed to illustrate its practical use. This section provides a quick-start guide focused on performing the basic calculations, namely,the reduction of a redundant basis. For detailed explanations of the algorithms and other standalone useful functions, refer to later sections of this manual.

Below, we present the minimal set of commands required to reduce a redundant basis to a physical one. In the first block of commands we initialize the program and the models, following the previous subsections. In `modelFull` we load a `FeynArts` model where the Lagrangian contains redundant operators (possibly a Green's basis of an EFT), while in `modelPhys` the Lagrangian involves only physical operators. The WCs information is loaded from the associated `paramList.txt` files. In the last line, we call `RedBasis[]`, the main function of `mosca`, which directly provides the expression of the WCs of the physical model in terms of the WCs in `modelFull`. The third argument of `RedBasis[]` specifies the order at which the EFT expansion is truncated during the reduction. The function is discussed in more detail in Section 3.4, together with an example of its output. We refer the reader to that section for further details.

It is worth noting that there are several other commands that are relevant in their own right. These include `PropagatorAttributes[]`, `AmplitudeComputation[]`, `AmplitudeMatching[]`, `ReplaceKinematics[]` or `FindIsomorphisms[]`, among others. However, since all of them are encompassed in the main function `RedBasis[]`, and the reduction of a redundant basis is mostly the purpose of this package, we will leave the demonstration of these functions to Sections 3 and 4.

```
In[10]:= TrueEFTOrder[WC[c, n, modelC]]

Out[10]= X
```

```
In[11]:= SetDirectory["/path/to/mosca/folder/"];
         << mosca`

         path = "/path/to/models/folder/";
         modelFull = FileNameJoin[{path, "name_model_full", "name_model_full"}];
         modelPhys = FileNameJoin[{path, "name_model_phys", "name_model_phys"}];

         PatchToMosca[modelFull]
         PatchToMosca[modelPhys]
```

3. A guide to mosca: objects and functions

3.1. 2-point functions calculation

A key feature that distinguishes on-shell matching is the need of computing what in the following will be referred to as 2-point attributes, namely, physical masses, wavefunction factors and explicit forms of Feynman propagators in momentum space. The precise expressions of the masses are needed for the correct generation of on-shell momenta, while the wavefunction factors appear in the computation of scattering amplitudes according to the LSZ formula. Meanwhile, propagators should account for the 2-point interactions in internal propagator lines of Feynman diagrams. In a given model, all operators involving two fields contribute to these functions, which means that these attributes will generally differ for each case. Hence, it is essential to adopt a consistent approach to incorporate these effects.

Propagators are obtained (up to factors of i) as the inverse of the 2-point functions, which we have access to by computing the amplitude $\phi \to \phi$ at tree-level and CT order 1 in FeynArts, with ϕ a generic field. While the propagator for scalar fields can be computed straightforwardly by inverting the 2-point function, the process becomes more complex for vector and fermion fields due to the tensor nature of these propagators.

First, in the case of a scalar, if we add arbitrary 2-point (local) operators we will have a 2-point Green's function of the form $\Gamma_S = p^2 - R(p^2)$, where R is a polynomial (*e.g.*, in a massive free scalar theory we would have $R = m^2$). Therefore, the scalar propagator is simply

$$D(p) = \frac{i}{p^2 - R(p^2)}. \tag{3.1}$$

Now, let us move on to vectors. Given a generic 2-point function $\Gamma_V^{\mu\nu}$ of a vector field, the corresponding propagator $\Pi_{\mu\nu}$ is determined by imposing $\Gamma_V^{\mu\rho}\Pi_{\rho\nu} = i\delta_\nu^\mu$. In general, $\Gamma_V^{\mu\nu}$ and $\Pi_{\mu\nu}$ can depend on all possible Lorentz structures:

$$\Gamma_V^{\mu\nu} = A_1 g^{\mu\nu} + A_2 p^\mu p^\nu, \qquad \Pi_{\mu\nu} = B_1 g_{\mu\nu} + B_2 p_\mu p_\nu,$$

where all coefficients can possibly depend on p^2. A_1 and A_2 can be separated into two distinct components: a canonical part coming from the kinetic and gauge-fixing terms, and a second part resulting from the contribution of other (effective) operators. We will denote the non-canonical parts as R_1 and R_2, so we have that

$$\Gamma_V^{\mu\nu} = \left(-p^2 + R_1\right) g^{\mu\nu} + \left(1 - \frac{1}{\xi} + R_2\right) p^\mu p^\nu.$$

Doing now

$$i\delta_\nu^\mu = \Gamma_V^{\mu\rho}\Pi_{\rho\nu} = B_1\left(-p^2 + R_1\right)\delta_\nu^\mu + \left[B_1\left(1 - \frac{1}{\xi} + R_2\right) + B_2\left(R_1 + p^2 R_2 - \frac{p^2}{\xi}\right)\right] p^\mu p_\nu,$$

we obtain that

$$\Pi_{\mu\nu} = -\frac{i g_{\mu\nu}}{p^2 - R_1} - p_\mu p_\nu \frac{i(1 - \xi + R_2\xi)(\xi R_1 + p^2 \xi R_2 - p^2)^{-1}}{p^2 - R_1}.$$

By setting $\xi = 0$, we find that the non-canonical contributions to the numerator vanish, and so does every dependence on R_2 in $\Pi_{\mu\nu}$. Then, in the Landau gauge, the modification in the vector propagators only manifests as a modification in the denominator given by the component of the 2-point function proportional to $g^{\mu\nu}$:

$$\Pi_{\mu\nu}(p) = \frac{-i\left(g_{\mu\nu} - \frac{p_\mu p_\nu}{p^2}\right)}{p^2 - R_1(p^2)}. \tag{3.2}$$

This makes the Landau gauge a very convenient choice for simplification of amplitudes.

For fermion fields, the general form of the 2-point function can be expressed as

$$\Gamma_F = \not{p} A + B,$$

where A, B are 4×4 Dirac structures functions of the squared-momentum. Due to the absence of free Lorentz indices, without loss of generality A and B must be linear combinations of the chirality projectors P_R and P_L. Therefore, we can write

$$A = R_1 P_R + \widetilde{R}_1 P_L, \qquad B = R_2 P_R + \widetilde{R}_2 P_L,$$

where $R_i, \widetilde{R}_i$ are polynomials of p^2. Inverting this expression yields the fermion propagator:

$$S(p) = \frac{i}{\not{p}(R_1 P_R + \widetilde{R}_1 P_L) + (R_2 P_L + \widetilde{R}_2 P_R)} = \frac{(\not{p} R_1 - R_2) P_R + (\not{p}\widetilde{R}_1 - \widetilde{R}_2) P_L}{R_1 \widetilde{R}_1 p^2 - R_2 \widetilde{R}_2}. \tag{3.3}$$

Here, both the numerator and denominator of the propagator are influenced by the non-canonical contributions to R_i and $\widetilde{R}_i$.

Once we know how to determine the explicit forms for propagators from the 2-point Green's functions, physical masses are obtained as the poles of such expressions while the wavefunction Z factors are the residues at these poles, as it was explained in Section 1.

```
DefineAllWCs[modelFull,
   FileNameJoin[{path, "name_model_full", "paramList.txt"}]]
DefineAllWCs[modelPhys,
   FileNameJoin[{path, "name_model_phys", "paramList.txt"}]]

FormatWC[modelFull, ModelName -> "full"]
FormatWC[modelPhys, ModelName -> "phys"]


In[12]:= RedBasis[modelFull, modelPhys, 6];
```

```
PropagatorAttributes[field, model, EFTOrder, Output -> True/False]
```

```
In[13]:= {mass,Zwf,prop} = PropagatorAttributes[S[1], modelPhi, 6]

         Propagator Attributes of S[1]:
         { m_phys^2 → m^2 - 2 invΛ^2 m^4 rDϕ, Z → 1 - 4 invΛ^2 m^2 rDϕ,
            Prop → 1/(p^2-m^2) - 2 p^4 invΛ^2 rDϕ/(p^2-m^2)^2 }

Out[13]= { m^2 - 2 invΛ^2 m^4 rDϕ, 1 - 4 invΛ^2 m^2 rDϕ, Prop$23001 }
```

In mosca we incorporated all this theoretical framework to automatically find the physical mass, the wave function renormalization factor and the propagator for each field from the 2-point Feynman rules of the model. By running
one can retrieve all the attributes associated with the 2-point function of a given field in the specified model, up to the dimension determined by EFTOrder. The Output option, enabled by default (True), formats and displays the result. For the Lagrangian of Eq. (2.1) we would have
The function returns the physical mass, the wavefunction renormalization and the propagator (notice that the list appearing before the *Out* cell is simply a *Print* display). The latter is a MATHEMATICA *pure function* whose explicit form can be recovered upon evaluation at the momentum and the momentum squared[4]:
Not only does PropagatorAttributes[] compute, show and return the 2-point attributes, but it also updates the reserved keywords mphys2[field,model], Z[field,model] and Prop[field,model], which can be called to access to the different parameters of a field in a specific model.

It should be emphasized what we mean by "propagator" in this context. Actually, the object Prop[field,model] is thought to be the replacement for the generic FeynAmpDenonimators FeynCalc structures in amplitudes. Thus, for scalars and vectors (in the Landau gauge), where only the denominator changes, Prop[] is simply the corresponding denominator. However, for fermions, according to Eq. (3.3) we need to modify the numerator too. Hence, we identify every $\not{p} \pm m$ (coming from a fermion propagator) and replace it—keeping its position in the fermionic chain—by the propagator dictated by Eq. (3.3) (up to the factor of i). The precise expressions for the Prop objects for scalar, spinor and vector fields can be found in Table 1.

Table 1
Expressions for Prop in mosca. These yield the correct propagators for scalars, vectors and fermions upon multiplication by i, $-i(g_{\mu\nu} - p_\mu p_\nu/p^2)$ and i, respectively. Refer to the beginning of this section for further details in the definition of R, R_i and $\widetilde{R}_i$ in each case.

Scalar	Vector	Fermion
$\frac{1}{p^2 - R}$	$\frac{1}{p^2 - R_1}$	$\frac{(\not{p}R_1 - R_2)P_R + (\not{p}\widetilde{R}_1 - \widetilde{R}_2)P_L}{R_1\widetilde{R}_1 p^2 - R_2\widetilde{R}_2}$

If the user wants to compute all these attributes for every field within a model at the same time, then it is enough to execute
The ExcludeParticles option allows to omit any field present in the model for the computation. For example, we extend the Lagrangian of Eq. (2.1) to

$$\mathcal{L}_{full} = \mathcal{L}_\phi - \frac{1}{4}F_{\mu\nu}F^{\mu\nu} + \overline{\psi}(i\not{D} - m_\psi)\psi - \frac{1}{2}r_{2F}\partial_\mu F^{\mu\nu}\partial^\rho F_{\rho\nu} + r_{\psi D}\overline{\psi}i\{D^2, \not{D}\}\psi + \mathcal{L}_{int}, \tag{3.4}$$

where $\mathcal{L}_{int}$ will be defined later, since it contains operators with more than two fields which are not relevant now. Here, the fermion ψ is coupled to a $U(1)$ gauge symmetry and the corresponding field-strength tensor is given by $F_{\mu\nu}$. Thus, in the FeynArts model we have a scalar S[1], a fermion F[1] and a vector V[1]. Naming modelFull the model associated with this Lagrangian, we would obtain the result by simply running

[4] In practice it would only be necessary to specify the momentum. Adding the squared momentum as a second variable is only a matter of convenience to correctly deal with SP[mom,mom] structures instead of mom^2. Even so, it is possible to call also the propagator with just one variable, say p, which will be understood internally as a call with p and p^2. Furthermore, a non-squared momentum p can only appear in fermion propagators, as we will see.

```
In[14]:= prop[p, p2]
```

$$\text{Out[14]=}\quad \frac{1}{\text{p2}-\text{m}^2}-\frac{2\,\text{p2}^2\,\text{inv}\Lambda^2\text{r}D\phi}{(\text{p2}-\text{m}^2)^2}$$

```
AllPropagatorAttributes[model, EFTOrder, ExcludeParticles -> {},
Output -> True/False]
```

```
In[15]:= AllPropagatorAttributes[modelFull, 6];

Propagator Attributes of F[1]:
{ m^2_phys → mψ^2 + 2invΛ^2mψ^4rψD, Z → 1 + 3invΛ^2mψ^2rψD,
Prop → ((γ · p)+mψ)/(p^2-mψ^2) + (invΛ^2(p^4(γ · p) + 2p^4mψ + p^2(γ · p)mψ^2)rψD)/(p^2-mψ^2)^2 }

Propagator Attributes of S[1]:
{ m^2_phys → m^2 - 2 invΛ^2m^4rDϕ, Z → 1 - 4 invΛ^2m^2rDϕ,
  Prop → 1/(p^2- m^2) - 2 p^4invΛ^2rDϕ/(p^2-m^2)^2 }

Propagator Attributes of V[1]:
{ m^2_phys → 0, Z → 1, Prop → 1/p^2 - invΛ^2r2F }
```

```
Bare2PhysMass[field, EFTOrder, model]
```

The `mphys2[]` object gives the expression for the physical mass in terms of the bare mass and other coefficients of the Lagrangian. It is also important to have the inverse relation, namely, the bare mass in terms of the physical mass (up to a given EFT order). The function that handles this is

It accepts either a single field or a list of fields as input. The redefinition (returned as a replacement rule) is given in terms of `MPhysSymbol[field]`, which represents the symbolic physical mass assigned internally to each field during the different computations. This function applied to the scalar field in the model yields the following result:

On the other hand, the functions `ReplaceBareMass[expression, EFTOrder, model]` and `ReplacePhysMass[expression, EFTOrder, model]` are responsible for rewriting an expression by translating bare masses of a given model in terms of physical masses, and *vice versa*. They are specifically designed to handle expressions that depend on `MPhysSymbol[]`, which is useful since, in general, the partial results arising from `mosca`'s matching functions are expressed using this symbolic physical mass. Both `ReplacePhysMass[]` and `ReplaceBareMass[]` also automatically expand the result up to the dimension specified by `EFTOrder`. For example, given an expression `exp`, applying `ReplacePhysMass[]` produces the following result:

3.2. Amplitude computation

Once the tools regarding 2-point amplitudes have been discussed we move on to S-matrix elements of general processes. The amplitudes can be computed using the following command:

This function calculates the physical amplitude for the specified process, considering all particles incoming, up to the given `EFTOrder` within the chosen model. The option `ExcludeParticles` allows for the exclusion of specific particles by specifying the corresponding fields. By default, no particles are excluded, as `ExcludeParticles` is set to an empty list. Additionally, the `SeparateCrossings` option is set to `False` by default, meaning that the amplitude is returned as a single expression. If `SeparateCrossings` is set to `True`, the function will return the result separating each diagram and all possible permutations of its external legs (*crossings*). We will cover this in detail in the next subsection.

Before presenting some examples, we must introduce the interaction term $\mathcal{L}_{int}$ from the Lagrangian in Eq. (3.4), where

```
In[16]:= Bare2PhysMass[S[1], 6, modelFull]

Out[16]= m → MPhysSymbol[S[1]] + MPhysSymbol[S[1]]^3 rDϕ
```

```
In[17]:= exp = a MPhysSymbol[S[1]] +b MPhysSymbol[S[1]]^2;
          ReplacePhysMass[exp, 6, modelFull]

Out[17]= a (m - m^3 rDϕ) + b (m^2 - 2 m^4 rDϕ)
```

```
AmplitudeComputation[process, EFTOrder, model, ExcludeParticles -> {},
                     SeparateCrossings -> True/False]
```

```
In[18]:= AmplitudeComputation[{F[1], -F[1], V[1]}, 6, modelPhys] //TraditionalForm
```

Out[18]//TraditionalForm=

$$g\left(-(\varphi(-\overline{P2}))\cdot(\overline{\gamma}\cdot\overline{\varepsilon}(P3))\cdot\overline{\gamma}^6\cdot(\varphi(\overline{P1}))\right)-g\left((\varphi(-\overline{P2}))\cdot(\overline{\gamma}\cdot\overline{\varepsilon}(P3))\cdot\overline{\gamma}^7\cdot(\varphi(\overline{P1}))\right)$$

Out[19]//TraditionalForm=

$$\left(\begin{array}{c}\{g\left(-(\varphi(-\overline{P2}))\cdot(\overline{\gamma}\cdot\overline{\varepsilon}(P3))\cdot\overline{\gamma}^6\cdot(\varphi(\overline{P1}))\right)-g\left((\varphi(-\overline{P2}))\cdot(\overline{\gamma}\cdot\overline{\varepsilon}(P3))\cdot\overline{\gamma}^7\cdot(\varphi(\overline{P1}))\right)\}\\ \{\{1\to1,2\to2,3\to3,4\to4\}\}\end{array}\right)$$

```
In[20]:= AmplitudeComputation[{F[1],F[1],-F[1],-F[1]},6,modelPhys,
```

$$\mathcal{L}_{int}=a_{\psi F}F_{\mu\nu}\overline{\psi}\sigma^{\mu\nu}\psi+a_{\phi}\phi^6+a_{\psi}\overline{\psi}\gamma^{\mu}\psi\overline{\psi}\gamma_{\mu}\psi+r_{\phi D}\phi^3\partial^2\phi$$
$$+r_{DF\psi}D^{\mu}F_{\mu\nu}\overline{\psi}\gamma^{\nu}\psi. \tag{3.5}$$

Additionally, we define modelPhys as the model containing the Lagrangians Eqs. (3.4) and (3.5) with all redundant operators (those with WCs r_i) set to zero.

Given then the following input:
The function generates the amplitude[5]:
If we set SeparateCrossings– >True, the result would rather be:
In this case, there is only one diagram, and the only existing permutation is the identity.

If we consider now another process, running the input
generates a result similar to the previous one, but now the first element of the output is a list of the amplitudes associated with the two different diagrams we can draw at tree level for this process:
These correspond to the four-fermion contact interaction and the diagram with a gauge boson propagating in an internal line. The second element of the output is a list containing the different allowed permutations of external legs for each diagram. For the contact interaction, there are no permutations. However, for the second diagram, the amplitude

$$\left\{-\sum_{i=6,7}\sum_{j=6,7}\frac{g^2(\varphi(-\overline{P3}))\cdot\overline{\gamma}^{\mathrm{Lor2}}\cdot\overline{\gamma}^{i}\cdot(\varphi(\overline{P1}))(\varphi(-\overline{P4}))\cdot\overline{\gamma}^{\mathrm{Lor2}}\cdot\overline{\gamma}^{j}\cdot(\varphi(\overline{P2}))}{(\overline{P2}+\overline{P4})^2}\right\},$$

calculated using modelPhys ($r_i\to0$), admits two possible permutations, *i.e.*, the identity and transposing particles 3 and 4:

$$\{\{1\to1,2\to2,3\to3,4\to4\},\{1\to1,2\to2,3\to4,4\to3\}\}.$$

The output displayed in TraditionalForm within the MATHEMATICA notebook appears as follows:

$$\left(\begin{array}{cc}\{A_1\} & \{A_2\}\\ \{\{1\to1,2\to2,3\to3,4\to4\}\} & \{\{1\to1,2\to2,3\to3,4\to4\},\{1\to1,2\to2,3\to4,4\to3\}\}\end{array}\right),$$

where A_1 and A_2 are the amplitudes corresponding to each diagram.
If we specify the option ExcludeParticles– >{V[1]}, the second diagram is omitted, resulting in the following output:

[5] In the remaining examples of this subsection, for brevity, we will suppose that $a_{\psi F}=0$.

```
In[21]:= {amps,perms}=AmplitudeComputation[process,EFTOrder,model,
```

```
RecoverFullAmplitude[amps, perms, fermionList]
```

```
In[22]:= finalAmp = Plus @@ RecoverFullAmplitude[amps, perms]
```

```
FindIsomorphisms[topologyList, allowedPermsList]
```

$$\begin{pmatrix} \{A_1\} \\ \{\{1 \to 1, 2 \to 2, 3 \to 3, 4 \to 4\}\} \end{pmatrix}.$$

For convenience, we can store the computed amplitudes and their associated permutations in separate lists using the following command: The function
applies the specified permutations to the partial amplitudes, generating a list of amplitudes with a length equal to the number of existing diagrams. For instance, in the case of Eq. (3.2), this list contains three elements. The function takes, as a third argument, a list of positive integers identifying which external lines are fermions, in order to correctly take into account "minus" relative signs between odd and even permutations. If the last argument is an empty list or it is omitted, it will be considered that there are no fermions as external particles. The total amplitude can then be reconstructed by executing the following command:

3.3. Finding isomorphisms

In `FeynArts`, there are several levels in the representation of a diagram. To find isomorphisms, `mosca` focuses on the most basic level: *topologies*. A topology is simply a set of lines (propagators) connecting a set of points (vertices), where no information about particles and Feynman rules is encoded. This is essentially a graph, so we can make use of graph theory tools to identify such isomorphisms.

To make the notation clear we briefly expose how we deal with diagrams and graphs. We can uniquely identify a graph with a set of pairs $\{\{v_1, v_2\}, \{v_3, v_4\}, \dots\}$, where each v_i represents a vertex and each pair a line connecting both vertices.[6] In this graph representation of a diagram, external lines are those which are connected to a vertex of degree 1.[7] Then, every vertex of degree 1 unambiguously identifies an external leg (i.e., an external particle). A permutation of a graph is essentially a permutation over the set $\{v_1, v_2, \dots\}$, as long as it does not exchange vertices of different degrees. This way, a permutation of a graph reliably represents a *crossing* of external particles in a diagram. Finally, two graphs will be called to be *isomorphic* if there exists a permutation that brings one into another, which is nothing but the mathematical way to express that two diagrams are related by a crossing permutation of the external particles.

The function
is designed to identify and classify isomorphic topologies while incorporating constraints on allowable permutations of external legs. It refines the identification of isomorphisms by considering only physically meaningful permutations —meaning those that only permute identical particles—, ensuring a precise classification of equivalent diagrams via graph representation theory.

In its initial approach, the function FindIsomorphisms[topologyList] classifies isomorphic graphs without considering permutation constraints, that is, taking all external lines to represent the same particle. It does so by comparing the canonical representations of the diagrams, as isomorphic graphs share the same canonical form.[8] This process identifies a set of representative unique graphs, their corresponding isomorphism classes, and the permutations that map isomorphic graphs to one another.

Given a canonical diagram D_0, its isomorphic graphs or, in group-theory terms, its equivalence class, consists of all the diagrams $\{D_i\}$ satisfying $g_i D_0 = D_i$, where g_i represents any permutation over external legs. Notice that if the diagrams have n external legs, g_i will be in general an element of S_n.

The next step consist on the further division of equivalence classes attending to the symmetries permitted by allowedPerms. This argument should be a list of lists of numbers with the form `{{1,…,i}, {i+1,…,j},…,{k,…,n}}`, stating that the external legs 1 to i are identical, equally for the legs $i+1$ to j and so on. With this, we aim to restrict the equivalence classes of diagrams related by a S_n permutation to those related by an element of $H = S_{\{1,\dots,i\}} \times S_{\{i+1,\dots,j\}} \times \dots \times S_{\{k,\dots,n\}}$, where S_I refers to the group of possible permutations over the set I.

However, additional considerations are necessary due to graph automorphisms, which introduce extra symmetries that impact the classification. These are all permutations of a graph which leave the graph invariant, which turn out to form a group A. In particular, there exists a degree of freedom in first rotating the representative graph D_0 by an arbitrary automorphism $a_0 \in A_0$. This means that in the initial classification, we store the permutation g_i for each diagram in the equivalence class, but, unfortunately, it is not unique. Specifically, since

$$D_i = g_i D_0 = g_i a_0^{(i)} D_0 = g_i' D_0,$$

it follows that g_i is only determined up to automorphisms $a_0^{(i)} \in A_0$.

To account for this, we must refine the isomorphism classes further. Given two diagrams D_i and D_j within the equivalence class of D_0, they are considered truly equivalent if there exists an element $h \in H$ such that

$$h D_i = D_j.$$

Expanding this condition, we obtain:

$$h D_i = D_j \iff h g_i a_0^{(i)} D_0 = g_j a_0^{(j)} D_0 \iff h g_i a_0^{(i)} = g_j a_0^{(j)}.$$

[6] If the graph is *undirected* the order of the pairs is unimportant.

[7] In graph theory, the degree of a vertex is the number of edges that are connected to that vertex.

[8] In graph theory, the canonical form of a graph is a standardized way of representing the graph so that structurally identical graphs (called isomorphic graphs) have the same representation, regardless of how their vertices are labeled. In MATHEMATICA, canonical forms are computed by default using the Bliss algorithm.

```
In[23]:= {time, topo} = CreateTopologies[0, 8->0] // Timing;
         time
         topo // Length

Out[23]= 5.28852
         39208
```

```
In[24]:= {reprDiag, classDiags, perms} = FindIsomorphisms[topo,{{1,2,3,4,5,6,7,8}}];
```

```
In[25]:= reprDiag // Length
         reprDiag

Out[25]= 32
         {1, 2, 9, 30, 121, 126, 127, 131, 243, 709, 2039, 2042, 2044, 2046, 2063,
         2092, 2110, 2111, 2303, 3267, 11489, 11490, 11496, 11513, 11514, 11546,
         11548, 11828, 28814, 28817, 28819, 28895}
```

Rearranging, and considering that $a_0^{(n)}$ are elements of a group, we get that

$$g_j = h g_i a_0^{(i)} \left(a_0^{(j)}\right)^{-1} = h g_i a_0^{(k)}. \tag{3.6}$$

From this, we read that we must find equivalence subclasses within $\{g_i\}$ by means of the equivalence relation Eq. (3.6). This process effectively constructs equivalence classes using *double cosets* HgA_0, which account for the symmetries induced by two subgroups: the right cosets of the subgroup of allowed permutations and the left cosets of the automorphism group of the representative diagram.

In practice, we first gather every isomorphic graph no matter the permutation, then take a representative of each class and subdivide the class further attending to the double coset HgA_0, where H is the subgroup of allowed permutations and A_0 the automorphism group of the representative diagram. Once the refined isomorphism classes are obtained, the function determines new representative graphs for each subclass and computes the permutations required to map graphs onto their respective representatives. Finally, it translates the results back into permutation notation, ensuring that the output remains interpretable in terms of external leg exchanges. The function returns a tuple consisting of the indices of representative graphs, the full classification of graphs, and the corresponding permutations for each isomorphism class.

To see an explicit example, let us identify the isomorphic classes in the tree-level topologies of $n = 8$ external legs with every possible vertex degree from 3 to 8. These are 39 208 different topologies and takes about 5.3 seconds in this case.

If we run

we get the list of representative diagrams in `topo`, the list of classes gathering all equivalent diagrams and the list of permutations among them. In this example we set all external particles to be the same, so every permutation is allowed (in this case, we could have omitted the second argument of `FindIsomorphisms[]` as well). Graphs in `reprDiag` and `classDiags` will be identified with integers, corresponding to their position in the `TopologyList` object `topo`. Thus, the 39 208 graphs are divided into 32 isomorphic classes whose representatives are the following diagrams:

For instance, there are 27 diagrams isomorphic to diagram #2, occuppying the positions 3 to 8 and 100 to 120 in `topo`. We can also also check the external leg permutations mapping these diagrams to diagram #2:

Notice the difference in the result when not all external particles are identical. Indeed, if we intend to identify isomorphisms of diagrams in the process $\psi\overline{\psi}\eta\eta\phi\phi\phi\phi$, where only crossings between the η's (particles 3 and 4) or between the ϕ's (particles 5 to 8) are allowed, we now have where `reprDiag2` contains this time 1 951 different diagrams. The 28 diagrams which where isomorphic before (under a general permutation of S_8) are now divided in 9 different classes when the equivalence relation is restricted to permutations of $S_{\{3,4\}} \times S_{\{5,6,7,8\}}$. In this particular example, these classes are found from the 2nd to the 10th position of `classDiags2`, grouping together the following diagrams: $\{2\}, \{3,4\}, \{5,6,7,8\}, \{9,10\}, \{100,101\}, \{102,103,104,105\}, \{106\}$, $\{107,\ldots,114\}$ and $\{115,\ldots,120\}$.

Once the topologies are divided, we have a very reduced number of diagram topologies to compute. These bare topologies are afterwards dressed with the proper field content, as usual, with the `InsertFields[]` function from `FeynArts`. The advantage of performing the identification of isomorphisms before the dressing is that there is no need to worry about internal lines having the same field content, the information about what external lines can be crossed is enough. If there are more than one way to dress the same topology with different fields, then `InsertFields[]` will do the job, but the crossing permutations among the different diagrams remain identical. This very process is also automated by the function

which gives a list of two components. The first one is the list of Feynman diagrams which are inequivalent under permutations. Each entry is conformed by a `TopologyList[]` object and corresponds to one different topology, but it may contain more than one diagram if several distinct particles could be inserted therein. The second component returned is the list of all necessary permutations to each entry of the diagrams list in order to recover the whole diagram set. The function `DiagramComputation[]` takes care of computing the necessary topologies and applying `FindIsomorphisms[]`, followed by the `InsertFields[]` command.

As a final comment before leaving this subsection, notice that `FindIsomorphisms[]` is equally suitable for loop diagrams, correctly

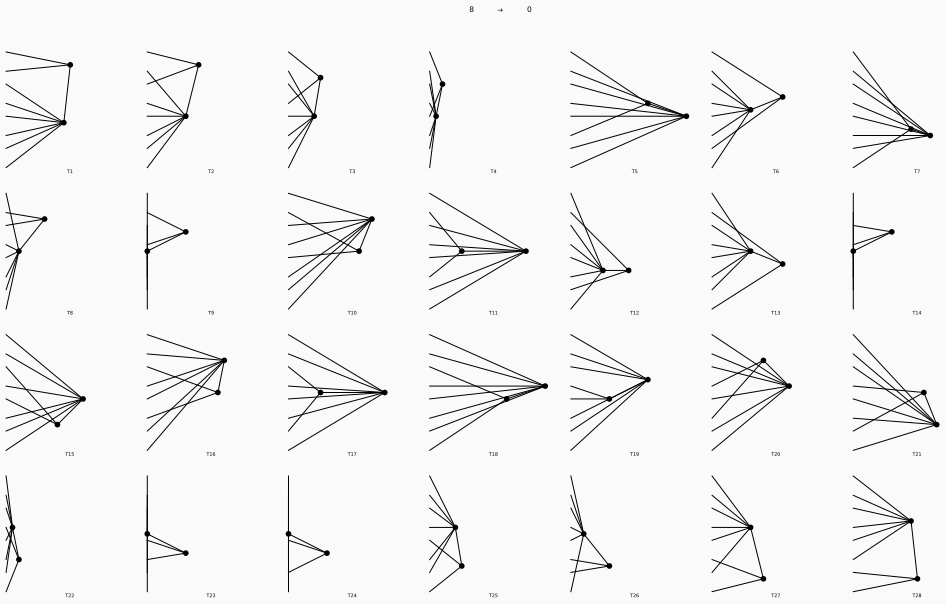
```
In[26]:= classDiags[[2]]
         perms[[2]] // Short

Out[26]= {2, 3, 4, 5, 6, 7, 8, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109,
         110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120}

Out[27]//Short=
```

```
{ <|1→1, 2→2, 3→3, 4→4, 5→5, 6→6, 7→7, 8→8|>,
<|1→1, 2→3, 3→2, 4→4, 5→5, 6→6, 7→7, 8→8|>, <<24>>,
<|1→6, 2→8, 3→1, 4→2, 5→3, 6→4, 7→5, 8→7|>,
<|1→7, 2→8, 3→1, 4→2, 5→3, 6→4, 7→5, 8→6|> }
```

```
In[28]:= Paint[topo[classDiags[[2]]], ColumnsXRows -> 7];
```



```
In[29]:= {reprDiag2,classDiags2,perms2} =
            FindIsomorphisms[topo,{{1},{2},{3,4},{5,6,7,8}}];
```

identifying crossings at the loop level. However, given the lack of knowledge about fermion flow at the topology level, with closed fermion loops there may be some diagrams which turn out to be related by a crossing permutation but they cannot be identified as such. This is because in `FeynArts` they correspond to the exact same topology, later becoming different diagrams with opposite fermion flow after the `InsertFields[]` command. For instance, this would happen with the closed fermion triangle loop diagrams, in which transposing two of the external legs is equivalent to the change of the direction of the fermion flow.

3.4. Rational kinematics

Once amplitudes are computed we have to replace every kinematic element with a numerical value. In the end, we expect every amplitude to be a complex polynomial in the WCs with, possibly, rational functions of the masses of the fields. Furthermore, we want to ensure that coefficients are rational so that no machine precision is lost over the whole process and the matching is exact.

To this end, we can exploit the *momentum twistor* theory introduced in [51] to generate rational-valued momenta satisfying momentum conservation and on-shell conditions, as shown in [39]. This is implemented in the `Mathematica` package found in https://github.com/StefanoDeAngelis/SpinorHelicity/tree/main/Wolfram/NumericalKinematics.wl, which we provide in the `mosca/ExternalPackages/NumericalKinematics/` folder with a slight modification to allow for the generation of momenta with symbolic masses m_i.

On the other hand, in the `KinematicSubstitution.m` file we implemented routines to transform the output of `NumericalKinematics.wl`

```
DiagramComputation[process, model, ExcludeParticles->{}]
```

```
KinematicConfigurations[nF, nV, nS, masses]
```

```
In[30]:= kin = KinematicConfigurations[2, 1, 0, {m1, m2, 0}];
```

```
In[31]:= kin["p",f1]

Out[31]= { -43831/47824 - 4 m1^2/61 , 77459/95648 - 40 m1^2/61 , -I(8429 + 62720 m1^2)/95648 , -2993/6832 - 4 m1^2/61 }
```

to facilitate the substitution of such rational kinematics into amplitudes. One of the main functions is
which generates a configuration of momenta, Dirac spinors and polarizations for nF fermions, nV vectors and nS scalars with masses $\{m_1, \ldots, m_{nF+nV+nS}\}$, where the first nF masses correspond to fermions and so on. Here, we recall that we do not address kinematic configurations for massive vectors, so the corresponding vector masses entries should be filled with0ŕs. The argument `masses` is to be provided with the list of masses, but alternatively also accepts either a single symbol (or number), in which case every particle is considered to have the same mass, or no argument whatsoever and then every particle is considered to be massless.

The output of `KinematicConfigurations[]` is an association. We can access the list of momenta, u Dirac spinors, $\bar{v}$ Dirac spinors, ϵ^+ positive polarizations and ϵ^- negative polarizations through the entries `"p"`, `"u"`, `"vbar"`, `"e+"` and `"e-"`, respectively.[9] Each of these is, in turn, another association where we have a unique key `fi,vi,si` for the i-th fermion, vector or scalar, respectively. Thereby, for instance, the momentum of the i-th scalar will be obtained through the entries `{"p",si}`, and the spinor of the j-th fermion through `{"u",fj}`.

To see an explicit example, let us generate a kinematic configuration with two fermions with masses m_1, m_2 and a massless vector.
The momentum (a four-vector) for the first fermion takes the following form:
To take the list of all momenta one can use the command `Values`. We can check momentum conservation and that every momentum is on-shell by running
Here, `MDot[a,b]` is the Minkowsi product $a \cdot b$ of the two four-vectors a and b.

Furthermore, Dirac equations for spinors, $(\not{p} - m)u = 0$, and for conjugate spinors, $\bar{v}(\not{p} + m) = 0$, are satisfied:
where pslash[a] "slashes" the four-vector a by applying $\not{a} = \sum_{\mu=0}^{3} \gamma^\mu a_\mu$.
Finally, both polarizations are also transverse
and consist of light-like vectors verifying $\epsilon^+ \cdot \epsilon^- = -1$:

The second important command is
This function is designed to receive an amplitude `amp` (or a list of amplitudes) and replace specific random rational kinematics in it. The format for specifying the number of particles and their respective masses is identical to the one in `KinematicConfigurations[]`. `ReplaceKinematics[]` effectively replaces values for momenta and polarization products, contractions with the Levi-Civita tensor $\epsilon_{\mu\nu\rho\sigma}$ and Dirac chains, thus transforming the amplitude into a complex polynomial over the WCs (admitting rational functions of the masses though). The `Momenta` option accepts a list which allows to specify, in order, the symbol for the momenta of the fermions, vectors and scalars in `amp`. If no option is given, it will be assumed that the n-th particle has momentum `Pn`.

Let us see how it works by computing a numerical value for the following expression:

$$g\ \bar{v}_1 \frac{\not{q}_1 - \not{q}_2}{(q_1 - q_2)^2} u_2,$$

where we have an antifermion and a fermion with momenta p_1, p_2, respectively, and 2 scalars with momentum q_1, q_2. Moreover, the fermions are massless and the scalar with momentum q_i has mass m_i. Then, one would have to type the following:

If vectors are present, then several numerical amplitudes will be returned, attending to the different combinations of + and - helicity polarizations. To see which amplitude comes from which polarization choice one can simply set the `PolarizationInfo` option to `True` (which is in `False` by default). For instance, we can compute the 4 gluon amplitude with `FeynArts+FeynCalc` and replace a physical configuration of momenta:
We can see that the only non-zero amplitudes are those having two gluons of the same helicity and the other two with the other one, in agreement with the very well-known fact that in an n-gluon amplitude at least two gluons need to have different helicity from the rest to give a non-vanishing result. Since every particle is massless, here we called `ReplaceKinematics[]` with just a 0 in the fifth argument, instead of a list of four0ŕs.

We finally comment some aspects. First, notice that, in the same call to `ReplaceKinematics[]`, the momenta is not being randomly recomputed in each case; it is computed once and used for every helicity configuration instead. This makes this function very useful for computing numerical values for cross-sections with sums over helicities. Second, for fermions we do not have distinction between $+1/2$ and $-1/2$ helicities because we consider all fermions to be Dirac. However, we can equally mimic configurations for Weyl fermions with well-defined helicites by adding the corresponding $P_{L/R}$ projectors in the spinor chains. Finally, if we want to replace the same rational kinematic configuration in several amplitudes, we can just provide them in a list as the first argument of `ReplaceKinematics[]`.

3.5. Green's basis reduction

The main function of `mosca` is

[9] Since we consider both fermions and antifermions always incoming, we refer to spinors as u and to conjugate spinors as $\bar{v}$. However, what we call here $\bar{v}$ would actually be a $\bar{u}$ in case of an outgoing fermion and *vice versa*.

```
In[32]:= plist = Values @ kin["p"];
         Plus @@ plist // Simplify
         (MDot[#, #] & /@ plist) // Simplify

Out[32]= { 0 , 0 , 0 , 0 }
         { m1^2 , m2^2 , 0 }
```

```
In[33]:= (pslash[kin["p",f1]] . kin["u",f1]) - m1 kin["u",f1] // Simplify
         (kin["vbar",f1] . pslash[kin["p",f1]]) + m1 kin["vbar",f1] // Simplify

Out[33]= { 0 , 0 , 0 , 0 }
         { 0 , 0 , 0 , 0 }
```

```
In[34]:= MDot[kin["e+",v1], kin["p",v1]] // Simplify
         MDot[kin["e-",v1], kin["p",v1]] // Simplify

Out[34]= 0
         0
```

```
In[35]:= MDot[kin["e+",v1], kin["e+",v1]] // Simplify
         MDot[kin["e-",v1], kin["e-",v1]] // Simplify
         MDot[kin["e+",v1], kin["e-",v1]] // Simplify

Out[35]= 0
         0
         -1
```

```
ReplaceKinematics[amp, nF, nV, nS, masses, Momenta -> {},
                  PolarizationInfo -> True/False]
```

```
In[36]:= amp = g SpinorVBar[p1] . (GS[q1-q2] / SP[q1-q2, q1-q2]) . SpinorU[p2];
         ReplaceKinematics[amp, 2, 0, 2, {0,0,m1,m2}, Momenta->{p1,p2,q1,q2}]
         //Simplify

Out[36]= { - g (447219682 + 172476999 m1^2 + 194337 m2^2) / (195 (42343 + 14205 m1^2 + 1661 m2^2)) }
```

```
In[37]:= ts = CreateTopologies[0, 4 -> 0];
         diags = InsertFields[ts, {V[1],V[1],V[1],V[1]} -> {},
          InsertionLevel -> {Particles}, Model -> modelGluon,
         GenericModel -> modelGluon];
         amp = FCFAConvert[CreateFeynAmp[diags], IncomingMomenta -> {P1,P2,P3,P4},
```

```
                List -> False, DropSumOver -> True];
amp = FeynAmpDenominatorExplicit[SUNSimplify[amp, SUNFJacobi -> True]];
ReplaceKinematics[amp, 0, 4, 0, 0, PolarizationInfo -> True] // Simplify

Polarization(s) for P1, P2, P3, P4 read: {++++, +++-, ++-+, ++--, +-++,
+-+-, +--+, +---, -+++, -++-, -+-+, -+--, --++, --+-, ---+, ----}
```

Out[37]=
$$\left\{0,0,0,\frac{768}{551}g_s^2\left(57\,f^{g1g4a}f^{g2g3a}+f^{g1g3a}f^{g2g4a}\right),0,\frac{3888}{551}g_s^2\left(57\,f^{g1g4a}f^{g2g3a}+f^{g1g3a}f^{g2g4a}\right),\right.$$
$$\frac{3}{8816}g_s^2\left(57\,f^{g1g4a}f^{g2g3a}+f^{g1g3a}f^{g2g4a}\right),0,0,\frac{16}{14877}g_s^2\left(57\,f^{g1g4a}f^{g2g3a}+f^{g1g3a}f^{g2g4a}\right),$$
$$\left.\frac{6859}{12528}g_s^2\left(57\,f^{g1g4a}f^{g2g3a}+f^{g1g3a}f^{g2g4a}\right),0,\frac{24389}{8208}g_s^2\left(57\,f^{g1g4a}f^{g2g3a}+f^{g1g3a}f^{g2g4a}\right),0,0,0\right\}$$

```
RedBasis[model1, model2, EFTOrder, OutputFormat -> {},
ExcludeParticles -> {}, StopAt -> {}]
```

```
ProcessList[model, ExcludeParticles -> {}, Sorted -> True/False]
```

This function returns the redefinition of the WCs of model2 in terms of the WCs of model1 up to the dimension fixed by EFTOrder. The function also accepts three optional parameters: (1) The OutputFormat option provides control over the display format of the result by specifying new values for ModelName, VisibleModel and VisibleOrder. If OutputFormat is left as an empty list, the output will follow the formatting settings previously defined through the FormatWC[] function. (2) The ExcludeParticles option, which defaults to an empty list, allows the user to exclude any particle present in the models from the calculation. (3) The StopAt option sets the process at which the computation should stop, if needed. Therefore, understanding how RedBasis[] determines the order of process computation is essential for effectively using this option. In this subsection, in order to give practical examples and see all these commands in action, we will take model1 → modelFull, corresponding to the Lagrangian Eqs. (3.4) and (3.5), and model2 → modelPhys, corresponding to the same Lagrangian where all redundant operators (those with WCs r_i) are set to zero.

In RedBasis[], the process list is first reduced by removing redundant processes that share similar Feynman rules for the contact interaction, as they do not provide additional information. Once this minimum set is obtained, the remaining processes are ordered according to specific criteria: priority is given to those with fewer total particles, a higher number of coefficients relative to kinematical structures, fewer distinct kinematical structures, and a greater number of identical particles. From this list, 2-point interactions are also excluded, as their contribution to the reduction is already accounted for by the AllPropagatorAttributes[] function.

The list of processes can be accessed by using the function
This function returns a list of computable processes within a given model, excluding the particles specified by ExcludeParticles. Since all particles are considered to be incoming, the output consists of lists representing the participating fields in each process. For example, running the command:
produces the output:
The Sorted option allows to output the list of processes used by RedBasis[] during the computation, following the same ordering criteria and excluding 2-point interactions. Having access to this ordered list can be helpful for selecting a specific process at which to stop the calculation in RedBasis[], using the StopAt option. For the previous example, setting Sorted– >True results in:

RedBasis[] makes use of two other functions designed to perform the matching between the amplitudes of two different models: MassReduction[] and AmplitudeMatching[].

First,
returns the expression that the bare mass of the given field in model2 must take, in terms of the bare mass and other 2-point WCs in model1, to require that the physical mass for such field is the same in both models, up to a given EFT order. The Output– >True (default) enables to display the result in a readable manner. For instance, by running
we obtain the physical mass of S[1] (ϕ) in terms of its bare mass in modelFull, up to dimension 6 in the EFT order. For the sake of clarity, we have set both VisibleModel– >True and VisibleModel– >True in the FormatWC[] function to explicitly show the model names and orders in the redefinition. The display option shows the full redefinition of the mass, while the output of the command itself is split order by order in the EFT expansion: $m = m^{(4)} + \frac{1}{\Lambda}m^{(5)} + \frac{1}{\Lambda^2}m^{(6)} + \dots$. This will be the *modus operandi* in the following commands. One can easily go back and forth between these two representations by applying JoinOrderWC[] and SplitOrderWC[] directly over the replacement rule list.

It is important to notice that here $\mathtt{m}_{\mathtt{phys}}$ is *not* the physical mass of ϕ, but rather the bare mass of ϕ appearing in the Lagrangian of modelPhys, whose WCs have been tagged with ModelName– >"phys" in the FormatWC[] command. However, in a Lagrangian with no 2-point operators, like the one in modelPhys, the bare mass naturally coincide with the physical mass at tree level, and hence the reduction looks identical to the one obtained when applying PropagatorAttributes[] of ϕ in modelFull (except for the fact that there we have the squared mass relation).

The function
computes the mass redefinition for all fields in model2. The Output option behaves the same as in MassReduction[], and ExcludeParticles allows to

```
In[38]:= ProcessList[modelPhys]

Out[38]= {{-F[1], F[1]}, {S[1], S[1]}, {V[1], V[1]}, {-F[1], F[1], V[1]},
         {-F[1], -F[1], F[1], F[1]}, {S[1], S[1], S[1], S[1]},
         {S[1], S[1], S[1], S[1], S[1], S[1]}}

In[39]:= ProcessList[modelPhys, Sorted->True]

Out[39]= {{-F[1], F[1], V[1]}, {S[1], S[1], S[1], S[1]}, {-F[1], -F[1], F[1], F[1]},
         {S[1], S[1], S[1], S[1], S[1], S[1]}}
```

```
MassReduction[field, model1, model2, EFTOrder, Output->True/False]
```

```
In[40]:= MassReduction[S[1], modelFull, modelPhys, 6]

         Mass redefinition for S[1]:
         m_phys^(4) → m_full^(4) - (m_full^(4))^3 rDϕ_full^(6)

Out[40]= { m_phys^(4) → m_full^(4), m_phys^(5) → 0, m_phys^(6) → -(m_full^(4))^3 rDϕ_full^(6) }
```

```
AllMassReduction[model1, model2, EFTOrder, Output->True/False,
                 ExcludeParticles -> {}]
```

skip any field in the computation. In this specific case we would further have:

After obtaining the relations of the bare masses of model2 in terms of the WCs of model1, it is necessary to find the relations regarding other WCs. The command responsible for this is:
This function returns the redefinition order by order for all the coefficients in model2 contributing to the given process up to the mass dimension fixed by EFTOrder, excluding the particles specified by ExcludeParticles. In some sense, AmplitudeMatching[] is a generalization of MassReduction[] for processes other than field− >field. For a matter of convenience, the redefinition will be given in terms of MPhysSymbol[] objects, so the ReplacePhysMass[] function can be useful to obtain a result depending on bare masses of model1.

RecalcPropAttributes is an option that controls whether 2-point function calculations are repeated. By default, it is set to True. Setting it to False skips this recalculation if the values have already been computed. Caution is advised when disabling this feature, especially when matching calculations are performed at different orders in the EFT expansion. Whenever the EFT order is increased, the propagator attributes must be extended accordingly to ensure consistency of the results.

On the other hand, the Replacements option, which defaults to a list of two empty lists, allows to apply any replacement rule in the amplitudes coming from model1 and model2. This is particularly useful to send some WCs at zero and to substitute known redefinitions for any of the coefficients.

Let us see this at work by considering two processes involving the scalar ϕ, {S[1], S[1], S[1]} and {S[1], S[1], S[1], S[1], S[1], S[1]}. Running AmplitudeMatching[] results in:
By applying ReplacePhysMass[] to the result with the following command we obtain:
We can see how the first process only matched the λ WC but the second one was enough to match both, λ and a_ϕ. We would like to highlight that this, as a general property of on-shell matching, constitutes one key difference with its off-shell counterpart, where one needs to deal with all the amplitudes, one by one, in order to match all WCs. Crucially, like two sides of the same coin, either matching on-shell allows to match many WCs with a unique amplitude, or otherwise it requires a high degree of redundancy which translates into strong cross-checks in the process.

In the latter case, in which one wants to match WCs with amplitudes of increasing number of fields, it is key to use the Replacements option in order to substitute already-known redefinitions. To use this option correctly, redefinitions must be expressed in terms of MPhysSymbol[]. In this context, the ReplaceBareMass[] function is particularly useful, as it automatically rewrites the input in terms of the symbolic physical mass. Imagine we know in advance the result from Out[39], then we can perform the matching for the 6-ϕ interaction as:

```
Mass redefinition for F[1]:
mψ_phys → mψ_full + mψ_full^3 rψD_full

Mass redefinition for V[1]:
0 → 0
```

```
AmplitudeMatching[model1, model2, process, EFTOrder, ExcludeParticles->{},
    RecalcPropAttributes-> True/False, Replacements -> {{},{}}]
```

```
In[41]:= sol1 = AmplitudeMatching[modelFull, modelPhys, {S[1], S[1], S[1], S[1]}, 6]

Out[41]= {lmbd_phys^(4) → lmbd_full^(4), lmbd_phys^(5) → 0,

  lmbd_phys^(6) → MPhysSymbol[S[1]]^2 (-8 lmbd_full^(4) rDϕ_full^(6) + rϕ D_full^(6))}

In[42]:= sol2 = AmplitudeMatching[modelFull, modelPhys,

Out[42]= {lmbd_phys^(4) → lmbd_full^(4), lmbd_phys^(5) → 0,

  aϕ_phys^(6) → aϕ_full^(6) + 4 lmbd_full^(4) (4 lmbd_full^(4) rDϕ_full^(6) - rϕ D_full^(6)),

  lmbd_phys^(6) → MPhysSymbol[S[1]]^2 (-8 lmbd_full^(4) rDϕ_full^(6) + rϕ D_full^(6))}
```

```
In[43]:= sol1BareMass = (#[[1]] -> ReplacePhysMass[#[[2]], 6, modelFull]) & /@ sol1

Out[43]= {lmbd_phys^(4) → lmbd_full^(4),  lmbd_phys^(5) → 0,

  lmbd_phys^(6) → -8 lmbd_full^(4) (m_full^(4))^2 rDϕ_full^(6) + (m_full^(4))^2 rϕ D_full^(6)}

In[44]:= sol2BareMass = (#[[1]] -> ReplacePhysMass[#[[2]], 6, modelFull]) & /@ sol2

Out[44]= {lmbd_phys^(4) → lmbd_full^(4), lmbd_phys^(5) → 0,

  aϕ_phys^(6) → aϕ_full^(6) + 16 (lmbd_full^(4))^2 rDϕ_full^(6) - 4 lmbd_full^(4) rϕ D_full^(6)),

  lmbd_phys^(6) → -8 lmbd_full^(4) (m_full^(4))^2 rDϕ_full^(6) + (m_full^(4))^2 rϕ D_full^(6)}
```

This iterative, process-by-process substitution procedure constitutes the core workflow of the main function RedBasis[]. Considering $\mathcal{L}_{full}$, the result for the reduction can be obtained by using this function as follows:

3.6. Useful functions

This section is dedicated to a set of auxiliary functions that, while not directly involved in the reduction of a Lagrangian, play a supportive role in the main workflow of `mosca`. These can be useful for users to carry out algebraic transformations, expansions, and other symbolic manipulations on expressions, providing greater flexibility and control in intermediate steps.

The first function, used inside most `mosca` functions, is which performs a series expansion in the perturbative parameter `invΛ` to a given EFT dimension, that is, to order $\texttt{inv}\Lambda^{(\texttt{EFTOrder}-4)}$.[10] It is important to notice that, by default, it applies an

[10] If the EFT order is set to an integer lower than 4, it computes the "renormalizable" (or EFT-independent) part of the expression.

```
In[45]:= reduction = (#[[1]] -> ReplaceBareMass[#[[2]], 6, modelFull]) & /@ sol1BareMass

Out[45]= {lmbd(4)phys → lmbd(4)full, lmbd(5)phys → 0,

  lmbd(6)phys → MPhysSymbol[S[1]]^2 (-8 lmbd(4)full rDϕ(6)full + rϕ D(6)full)}

In[46]:= sol2bis = AmplitudeMatching[modelFull, modelPhys,
     {S[1],S[1],S[1],S[1],S[1],S[1]},
   6, Replacements->{{},reduction}]

Out[46]= {aϕ(6)phys → aϕ(6)full + 4 lmbd(4)full (4 lmbd(4)full rDϕ(6)full - rϕ D(6)full)}
```

```
In[47]:= finalReduction = RedBasis[modelFull, modelPhys, 6];

>> Number of solved WCs: 5/5
>> List of solved WCs: {m, g, lmbd, aψ, aϕ}
>> Showing the reduction
mphys → mfull - m3full rϕDfull

mψphys → mψfull + mψ3full rψDfull

lmbdphys → lmbdfull - 8 lmbdfull m2full rDϕfull + m2full rϕDfull

gphys → gfull

aϕphys → aϕfull + 16 lmbd2full rDϕfull - 4 lmbdfull rϕDfull

aψ(6)phys → aψfull - 1/2 g2full r2Ffull + gfull rDFψfull

aψF(6)phys → aψFfull - 1/2 gfull mψfull rψDfull

All done!

>> Elapsed time: 3.594948
```

EFTSeries[expression, EFTOrder]

before doing the series expansion. This last function, which we encountered before, first ignores every occurrence of invΛ in expression and then gets the right power of the EFT parameter according to the products of different WCs and their EFT order, independently of the model they belong to. For instance, we would have the following behaviors:
If one wants to compute a series in the EFT parameter, explicitly leaving the powers of invΛ already present in expression, it is possible to skip the ExplicitEFTOrder[] command by setting ExplicitEFTOrder− >False in the EFTSeries.

Finally, the matching equations eqs obtained after replacing numerical kinematics are solved by means of the command, which automates the process of solving the system of equations eqs in the variables solveVars order by order in invΛ. It solves the system by identifying and matching the coefficients of the different powers of invΛ, in a strictly increasing order and replacing the results of lower

ExplicitEFTOrder[expression]

powers in the following matching equations. This significantly simplifies the system, because WCs appearing in products in higher order equations, *e.g.* $\texttt{c1}^{(6)}\texttt{c2}^{(6)}$, could have already been solved in more simple linear equations at lower orders. Indeed, this function intends to solve the system of equations in a linear way, unless a non-linear equation is encountered, in which case it raises an error but continues with the resolution of the system with the MATHEMATICA built-in function Solve. This procedure ensures the finding of a solution, while a brute-force Solve[eqs,solveVars] approach could fail in systems of really long equations, taking for hours and finding no solution, even when a solution does exist.

4. Using mosca

While the reader is already equipped with all the necessary tools to run and use mosca, we would like to devote this section to comment on the capabilities of mosca and show some possible issues and other aspects.

First, despite mosca being designed to facilitate the reduction of bases and implement the on-shell matching approach, we had to include a couple of functionalities of a much broader usage. Indeed, the automation of finding Feynman diagrams related by isomorphisms in order to reduce as much as possible the number of diagrams to compute is completely suitable for the everyday use of FeynArts, as far as all particles are considered incoming. This can be really convenient for both memory and time efficiency, as well as for facilitating several analysis aspects. Certainly, regarding this last statement, in the inspection of the different contributions to a given amplitude, it is much easier to visualize and understand unique Feynman diagrams (in the sense that they are not related by permutations) than to face all the—possibly hundreds of—graphs returned by FeynArts.

As far as time and memory efficiency are considered, we can take the electroweak (EW) sector of the SMEFT Green's basis for dimension 6 (model_SMEFT_HBW, provided within mosca) and compare some results if we use standard FeynArts techniques (A) or our isomorphisms-identification approach (B). In Fig. 2 we show the comparison between methods in the computation time for diagrams and amplitude expressions in 76 different processes (the first 40 at tree-level and the following 36 at one-loop level).

Regarding the time it takes to just compute diagrams we can see a similar behavior, with slight performance differences depending on the process. However, a notable improvement of method (B) with respect to method (A) is found after using the FCFAConvert[CreateFeynAmp[#]] command to compute the associated amplitude. This obviously extrapolates to every subsequent manipulation performed over the amplitudes, saving considerable time and memory if those are only applied to the minimal set of permutation-unrelated diagrams (and afterwards reconstructing the whole amplitude with the permutations, if needed), instead of applying them directly over all diagrams.

Note that the Y-axis in the graphs is set to logarithmic scale. Thus, there are some processes for which the amplitude computation time varies from 1 to 30 minutes, depending on the employed method. Moreover, the reason why there are only 36 one-loop processes instead of 40 is that the full amplitude expressions for the last 4 chosen processes were not even manageable by the available computer memory. With approach (B) this does not represent an issue, as long as one works with the minimal set of permutation-unrelated amplitudes.

Other general-purpose tools than can be used beyond on-shell matching context are the commands to perform rational kinematic substitutions. Although there are some package implementations to generate random-valued kinematics (see refs. [40,43]), to the best of our knowledge, there is no such tool to explicitly replace rational-valued kinematics satisfying on-shell conditions in analytic Feynman amplitudes, with even the possibility to keep masses symbolic. Indeed, although the ReplaceKinematics[] is absolutely designed to translate expressions from FeynCalc, the KinematicConfigurations[] function is completely general, so that a generalization of ReplaceKinematics[] to interpret expressions from other packages, whether private or public, such as FeynArts itself or FormCalc [52], should be rather straightforward.

After these comments on general uses of different commands, we would like to give some advice when using mosca. First, performing a FormatWC[] is almost mandatory if one does not want to decipher unintelligible outputs in terms of WC[] objects. Following this, while setting VisibleOrder and VisibleModel options to True can blur things a bit, they can be very useful to check, for example, whether a WC has been completely reduced or not after an AmplitudeMatching[]. Indeed, it could happen than we enocunter an output like c1− >c1 + r c2, coming from a command similar to AmplitudeMatching[model1,model2,process,X]. In light of this, it may seem that the WC c_1 has been matched. But then, after a VisibleModel− >True setting, it could turn out that $\texttt{c1}_{model2} - > \texttt{c1}_{model1} + \texttt{r}_{model1}\texttt{c2}_{model2}$, revealing that the matching procedure was not able to disentangle c_2 from c_1. In such a case, it would yet be necessary another process to fully have the expressions for c_1 and c_2 only in terms of WCs from model1.

Another useful tip is regarding the RedBasis[] function. This command does not stop until it either matches every WC in the model or computes every available process in ProcessList[]. Yet, and despite all efforts to speed up computations, for amplitudes with a high number of external legs the procedure is sometimes remarkably slow. When facing one of those processes, one could be tempted to abort the evaluation, at the cost of losing the collected solutions of all already-matched WCs. To avoid this, every call to RedBasis[model1,model2,X] internally appends the solution of the matching after every process to a list, under the name of the reserved keyword Reduction[model1,model2,X]. Hence, if one aborts the evaluation, there is no information lost.

Bear in mind that Reduction[model1,model2,X] essentially stores the results given by a series of AmplitudeMatching[] calls, so that: (1) they are returned in the order-by-order expansion of WCs, (2) they are written in terms of MPhysSymbol[] and (3) it could happen that $c_{k\,\text{model2}} \to f(c_{i\,\text{model1}}, c_{j\,\text{model2}})$ where f is a function of WCs from both model1 and model2, as noted previously. For (1) and (2), there exist built-in functions such as JoinOrderWC[] and ReplacePhysMass[] to deal with these "inconveniences". For (3), it could be the case that some of the $c_{j\,\text{model2}}$ appearing in the redefinition of $c_{k\,\text{model2}}$ have been solved later on; hence, it is necessary to perform a repeated replacement of the list of reductions over itself (of course, only over the values, not the keys). In this vein, we also would like to recommend the users that, in case they want to save results to a text file, do so with a replacement rule, in order to avoid the heavily formatted WC structures.

Finally, the mosca folder includes a collection of example notebooks (mosca-notebook-X.nb), which demonstrate the usage of the package. The models used in these examples are stored in the Models/ directory. Additionally, the mosca-paper-example.nb notebook is provided, allowing you to try out some of the commands discussed throughout this paper.

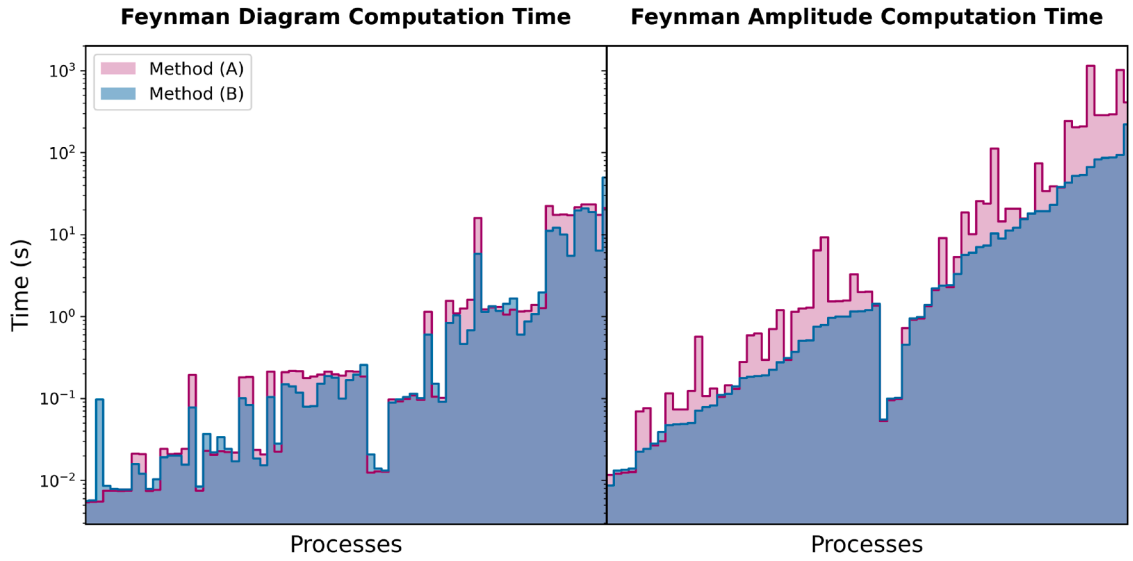


Fig. 2. Comparison of diagram and amplitude computation times for 76 different processes within the EW sector of dimension-6 SMEFT Green's basis with two different methods: standard `FeynArts` techniques (A) and our isomorphisms-identification approach (B). The amplitude computation time in method (B) corresponds to the total time after applying a `RecoverFullAmplitude[]` command.

```
In[48]:= ExplicitEFTOrder[invΛ^2 α]
         ExplicitEFTOrder[invΛ^2 α WC[c1, 6, model1] WC[c2, 6, model2]]

Out[48]= α
         invΛ^4 α c1(6)model1 c2(6)model2
```

5. Conclusions and outlook

In this article, we have introduced the first version of `mosca`, which implements an efficient on-shell matching algorithm. The toolkit includes several additional features designed to facilitate the manipulation of symbolic expressions. Among these, there are functions that implement isomorphism-identification algorithms to simplify the computation of Feynman diagrams, as well as utilities for performing algebraic transformations, series expansions, and other symbolic manipulations. As demonstrated in Section 4, the automation of identifying isomorphic Feynman diagrams results in a non-negligible reduction in computation time. Furthermore, the main function of the package in this initial release is dedicated to automatically perform reduction of redundant bases to physical bases in effective field theories, via the on-shell matching procedure.

As practical examples we have shown how to use `mosca` in several different situations, including: the calculation of physical masses, wave-function factors and explicit forms of Feynman propagators; the computation of amplitudes for distinct processes involving scalar, fermion and vector fields, with the corresponding identification of isomorphic diagrams and substitution of kinematic structures; and finally, the reduction of a Green's basis to a physical one by means of the main function of `mosca`.

Looking ahead, there are several directions in which `mosca` can naturally evolve. First, in this initial release the package cannot deal with flavor indices, so we will make it compatible. Second, we will also consider the possibility of having 2-point operators with mixing fields. Finally, from a broader perspective, we plan to extend the capabilities of `mosca` to the loop level, which includes the direct renormalization of effective Lagrangians in a physical basis, as well as the automated computation of finite matching contributions, including those from evanescent operators. These extensions have been previously discussed in [38], which demonstrates the effectiveness of numerical on-shell methods even in these scenarios. While implementing these features is expected to be relatively straightforward in principle, special care must be taken in treating the soft contributions to amplitudes in the full and effective theories, as they might be different due to evanescent effects.

`mosca` represents a significant development in the emerging field of on-shell methods, a framework that remains far less developed and barely adopted compared to the traditional off-shell approach. Once all planned functionalities are implemented, `mosca` will fully support a complete on-shell matching procedure based solely on a physical operator basis. This eliminates the need to include redundant or evanescent operators and avoids the often cumbersome process of basis reduction.

CRediT authorship contribution statement

Javier López Miras: Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Software, Project administration, Methodology, Investigation, Formal analysis, Conceptualization; **Fuensanta Vilches:** Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Software, Resources, Project administration, Methodology, Investigation, Formal analysis, Conceptualization.

Data availability

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

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

We would like to acknowledge the valuable contributions of Mikael Chala and José Santiago during the development of the package. As `mosca` is the result of a previously shared collaboration, the insightful discussions and ideas exchanged with them were essential to its success. We also acknowledge support from the MCIN/AEI (10.13039/501100011033) and ERDF (grants PID2022-139466NB-C21 and PID2022-139466NB-C22), from the Junta de Andalucía grant P21-00199 and from Consejería de Universidad, Investigación e Innovación, Gobierno de España and Unión Europea – NextGenerationEU under grants AST22 6.5 and CNS2022-136024. JLM is further supported by an FPU grant (FPU23/02028) from Consejería de Universidad, Investigación e Innovación, Gobierno de España.

References

[1] N.P. Hartland, F. Maltoni, E.R. Nocera, J. Rojo, E. Slade, E. Vryonidou, C. Zhang, A Monte Carlo global analysis of the standard model effective field theory: the top quark sector, JHEP 04 (2019) 100. arXiv:1901.05965. https://doi.org/10.1007/JHEP04(2019)100

[2] R. Aoude, T. Hurth, S. Renner, W. Shepherd, The impact of flavour data on global fits of the MFV SMEFT, JHEP 12 (2020) 113. arXiv:2003.05432. https://doi.org/10.1007/JHEP12(2020)113

[3] S. Dawson, S. Homiller, S.D. Lane, Putting standard model EFT fits to work, Phys. Rev. D 102 (5) (2020) 055012. arXiv:2007.01296. https://doi.org/10.1103/PhysRevD.102.055012

[4] Anisha, S. Das Bakshi, J. Chakrabortty, S.K. Patra, Connecting electroweak-scale observables to BSM physics through EFT and Bayesian statistics, Phys. Rev. D 103 (7) (2021) 076007. arXiv:2010.04088. https://doi.org/10.1103/PhysRevD.103.076007

[5] A. Falkowski, M. González-Alonso, O. Naviliat-Cuncic, Comprehensive analysis of beta decays within and beyond the standard model, JHEP 04 (2021) 126. arXiv:2010.13797. https://doi.org/10.1007/JHEP04(2021)126

[6] J. Ellis, M. Madigan, K. Mimasu, V. Sanz, T. You, Top, higgs, diboson and electroweak fit to the standard model effective field theory, JHEP 04 (2021) 279. arXiv:2012.02779. https://doi.org/10.1007/JHEP04(2021)279

[7] J.J. Ethier, G. Magni, F. Maltoni, L. Mantani, E.R. Nocera, J. Rojo, E. Slade, E. Vryonidou, C. Zhang, SMEFiT, Combined SMEFT interpretation of higgs, diboson, and top quark data from the LHC, JHEP 11 (2021) 089. arXiv:2105.00006. https://doi.org/10.1007/JHEP11(2021)089

[8] J. de Blas, Y. Du, C. Grojean, J. Gu, V. Miralles, M.E. Peskin, J. Tian, M. Vos, E. Vryonidou, Global SMEFT fits at future colliders, in: Snowmass 2021, 2022. arXiv:2206.08326

[9] B. Grzadkowski, M. Iskrzynski, M. Misiak, J. Rosiek, Dimension-six terms in the standard model lagrangian, JHEP 10 (2010) 085. arXiv:1008.4884. https://doi.org/10.1007/JHEP10(2010)085

[10] F. del Aguila, S. Bar-Shalom, A. Soni, J. Wudka, Heavy majorana neutrinos in the effective lagrangian description: application to hadron colliders, Phys. Lett. B 670 (2009) 399–402. arXiv:0806.0876. https://doi.org/10.1016/j.physletb.2008.11.031

[11] A. Aparici, K. Kim, A. Santamaria, J. Wudka, Right-handed neutrino magnetic moments, Phys. Rev. D 80 (2009) 013010. arXiv:0904.3244. https://doi.org/10.1103/PhysRevD.80.013010

[12] S. Bhattacharya, J. Wudka, Dimension-seven operators in the standard model with right handed neutrinos, Phys. Rev. D 94 (5) (2016) 055022. [Erratum: Phys.Rev.D 95, 039904 (2017)]. arXiv:1505.05264. https://doi.org/10.1103/PhysRevD.94.055022

[13] Y. Liao, X.-D. Ma, Operators up to dimension seven in standard model effective field theory extended with sterile neutrinos, Phys. Rev. D 96 (1) (2017) 015012. arXiv:1612.04527. https://doi.org/10.1103/PhysRevD.96.015012

[14] J. Aebischer, T. Kapoor, J. Kumar, wilson: a package for renormalization group running in the SMEFT with Sterile Neutrinos (2024). arXiv:2411.07220

[15] E.E. Jenkins, A.V. Manohar, P. Stoffer, Low-energy effective field theory below the electroweak scale: operators and matching, JHEP 03 (2018) 016. arXiv:1709.04486. https://doi.org/10.1007/JHEP03(2018)016

[16] I. Bischer, W. Rodejohann, General neutrino interactions from an effective field theory perspective, Nucl. Phys. B 947 (2019) 114746. arXiv:1905.08699. https://doi.org/10.1016/j.nuclphysb.2019.114746

[17] M. Chala, A. Titov, One-loop matching in the SMEFT extended with a sterile neutrino, JHEP 05 (2020) 139. arXiv:2001.07732. https://doi.org/10.1007/JHEP05(2020)139

[18] T. Li, X.-D. Ma, M.A. Schmidt, General neutrino interactions with sterile neutrinos in light of coherent neutrino-nucleus scattering and meson invisible decays, JHEP 07 (2020) 152. arXiv:2005.01543. https://doi.org/10.1007/JHEP07(2020)152

[19] T. Li, X.-D. Ma, M.A. Schmidt, Constraints on the charged currents in general neutrino interactions with sterile neutrinos, JHEP 10 (2020) 115. arXiv:2007.15408. https://doi.org/10.1007/JHEP10(2020)115

[20] B. Gripaios, D. Sutherland, An operator basis for the standard model with an added scalar singlet, JHEP 08 (2016) 103. arXiv:1604.07365. https://doi.org/10.1007/JHEP08(2016)103

[21] M. Chala, G. Guedes, M. Ramos, J. Santiago, Running in the ALPs, Eur. Phys. J. C 81 (2) (2021) 181. arXiv:2012.09017. https://doi.org/10.1140/epjc/s10052-021-08968-2

[22] M. Bauer, M. Neubert, S. Renner, M. Schnubel, A. Thamm, The low-energy effective theory of axions and ALPs, JHEP 04 (2021) 063. arXiv:2012.12272. https://doi.org/10.1007/JHEP04(2021)063

[23] C. Grojean, J. Kley, C.-Y. Yao, Hilbert series for ALP EFTs, JHEP 11 (2023) 196. arXiv:2307.08563. https://doi.org/10.1007/JHEP11(2023)196

[24] C.W. Murphy, Dimension-8 operators in the standard model effective field theory, JHEP 10 (2020) 174. arXiv:2005.00059. https://doi.org/10.1007/JHEP10(2020)174

[25] C.W. Murphy, Low-Energy effective field theory below the electroweak scale: dimension-8 operators, JHEP 04 (2021) 101. arXiv:2012.13291. https://doi.org/10.1007/JHEP04(2021)101

[26] H.-L. Li, Z. Ren, M.-L. Xiao, J.-H. Yu, Y.-H. Zheng, Low energy effective field theory operator basis at $d \leq 9$, JHEP 06 (2021) 138. arXiv:2012.09188. https://doi.org/10.1007/JHEP06(2021)138

[27] H.-L. Li, Z. Ren, J. Shu, M.-L. Xiao, J.-H. Yu, Y.-H. Zheng, Complete set of dimension-eight operators in the standard model effective field theory, Phys. Rev. D 104 (1) (2021). https://doi.org/10.1103/physrevd.104.015026

[28] H.-L. Li, Z. Ren, M.-L. Xiao, J.-H. Yu, Y.-H. Zheng, Operator bases in effective field theories with sterile neutrinos: $d \leq 9$, JHEP 11 (2021) 003. arXiv:2105.09329. https://doi.org/10.1007/JHEP11(2021)003

[29] H.-L. Li, Z. Ren, M.-L. Xiao, J.-H. Yu, Y.-H. Zheng, Operators for generic effective field theory at any dimension: on-shell amplitude basis construction, JHEP 04 (2022) 140. arXiv:2201.04639. https://doi.org/10.1007/JHEP04(2022)140

[30] R.V. Harlander, M.C. Schaaf, AutoEFT: automated operator construction for effective field theories, Comput. Phys. Commun. 300 (2024) 109198. arXiv:2309.15783. https://doi.org/10.1016/j.cpc.2024.109198

[31] C. Arzt, Reduced effective lagrangians, Phys. Lett. B 342 (1995) 189–195. arXiv:hep-ph/9304230. https://doi.org/10.1016/0370-2693(94)01419-D

[32] J.C. Criado, M. Pérez-Victoria, Field redefinitions in effective theories at higher orders, JHEP 03 (2019) 038. arXiv:1811.09413. https://doi.org/10.1007/JHEP03(2019)038

[33] J. Fuentes-Martín, M. König, J. Pagès, A.E. Thomsen, F. Wilsch, A proof of concept for matchete: an automated tool for matching effective theories, Eur. Phys. J. C 83 (7) (2023) 662. arXiv:2212.04510. https://doi.org/10.1140/epjc/s10052-023-11726-1

[34] A. Carmona, A. Lazopoulos, P. Olgoso, J. Santiago, Matchmakereft: automated tree-level and one-loop matching, SciPost Phys. 12 (6) (2022) 198. arXiv:2112.10787. https://doi.org/10.21468/SciPostPhys.12.6.198

[35] H. Georgi, On-shell effective field theory, Nucl. Phys. B 361 (1991) 339–350. https://doi.org/10.1016/0550-3213(91)90244-R

[36] X. Li, S. Zhou, One-loop Matching and Running via On-shell Amplitudes (2023). arXiv:2309.10851

[37] S. De Angelis, G. Durieux, EFT matching from analyticity and unitarity, SciPost Phys. 16 (2024) 071. arXiv:2308.00035. https://doi.org/10.21468/SciPostPhys.16.3.071

[38] M. Chala, J.L. Miras, J. Santiago, F. Vilches, Efficient on-shell matching, 2024. arXiv:2411.12798

[39] S. De Angelis, Amplitude bases in generic EFTs, JHEP 08 (2022) 299. arXiv:2202.02681. https://doi.org/10.1007/JHEP08(2022)299

[40] M. Accettulli Huber, SpinorHelicity4D: a mathematica toolbox for the four-dimensional spinor-helicity formalism (2023). arXiv:2304.01589

[41] N. Arkani-Hamed, T.-C. Huang, Y.-t. Huang, Scattering amplitudes for all masses and spins, JHEP 11 (2021) 070. arXiv:1709.04891. https://doi.org/10.1007/JHEP11(2021)070

[42] C. Cheung, D. O'Connell, Amplitudes and spinor-helicity in six dimensions, JHEP 07 (2009) 075. arXiv:0902.0981. https://doi.org/10.1088/1126-6708/2009/07/075

[43] D. Maitre, P. Mastrolia, S@m, a mathematica implementation of the Spinor-Helicity formalism, Comput. Phys. Commun. 179 (2008) 501–574. arXiv:0710.5559. https://doi.org/10.1016/j.cpc.2008.05.002

[44] H. Kluberg-Stern, J.B. Zuber, Renormalization of nonabelian gauge theories in a background field gauge. 2. gauge invariant operators, Phys. Rev. D 12 (1975) 3159–3180. https://doi.org/10.1103/PhysRevD.12.3159

[45] C. Grosse-Knetter, Effective lagrangians with higher derivatives and equations of motion, Phys. Rev. D 49 (1994) 6709–6719. arXiv:hep-ph/9306321. https://doi.org/10.1103/PhysRevD.49.6709

[46] J. Wudka, Electroweak effective lagrangians, Int. J. Mod. Phys. A 9 (1994) 2301–2362. arXiv:hep-ph/9406205. https://doi.org/10.1142/S0217751X94000959

[47] A.V. Manohar, Introduction to effective field theories (2018). arXiv:1804.05863. https://doi.org/10.1093/oso/9780198855743.003.0002

[48] U. Banerjee, J. Chakrabortty, C. Englert, S.U. Rahaman, M. Spannowsky, Integrating out heavy scalars with modified equations of motion: matching computation of dimension-eight SMEFT coefficients, Phys. Rev. D 107 (5) (2023) 055007. arXiv:2210.14761. https://doi.org/10.1103/PhysRevD.107.055007

[49] N. Brambilla, H.S. Chung, V. Shtabovenko, A. Vairo, Feynonium: using feyncalc for automatic calculations in nonrelativistic effective field theories, JHEP 11 (2020) 130. arXiv:2006.15451. https://doi.org/10.1007/JHEP11(2020)130

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

[51] A. Hodges, Eliminating spurious poles from gauge-theoretic amplitudes, JHEP 05 (2013) 135. arXiv:0905.1473. https://doi.org/10.1007/JHEP05(2013)135

[52] 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. https://doi.org/10.1016/S0010-4655(98)00173-8