Manual for

DECO

Version 1.0

Simon Calò, Coenraad Marinissen, Rudi Rahn

November 25, 2022

Quick reference guide for `DECO` 1.0

Step 0 Install `FORM` (chapter 2)

Step 1 Set global flags and parameters (chapter 3)

Step 2 Define field content (chapter 4)

Step 3 Assign charges (chapter 5)

Step 3 Run `DECO` (chapter 6)

Step 4 (optional) Process output (chapter 7)

Step 5 (very optional) Request additional symmetries and representations (chapter 8)

Step 6 (hopefully unnecessary) Commonly encountered problems (chapter 9)

Available fields

Label	Lorentz representation
`Scalar`	$(0,0)$
`LHFermion`	$(\frac{1}{2},0)$
`RHFermion`	$(0,\frac{1}{2})$
`DiracFermion`	$(\frac{1}{2},0)\oplus(0,\frac{1}{2})$
`FieldStrength`	$(1,0)\oplus(0,1)$
`Gravity`	$(2,0)\oplus(0,2)$

Available groups and representations

Group	Label	Representation	Representation label
U(1)	`U1`	charge (integer)	`...,-3,-2,-1,0,1,2,3,...`
SU(2)	`SU2`	trivial	`1`
		fundamental	`2`
		adjoint	`3`
SU(3)	`SU3`	trivial	`1`
		fundamental	`3`
		antifundamental	`B`
		adjoint	`8`
A_4	`A4`	1	`1`
		$1'$	`p1`
		$1''$	`pp1`
		3	`3`
S_4	`S4`	1	`1`
		$1'$	`p1`
		2	`2`
		3	`3`
		$3'$	`p3`
U(1) with residue	`U1R`	residue X, charge C (integers)	`R(X,C)`
$\mathbb{Z}_n$	`Zn`	cycle X, charge C (integers)	`R(X,C)`

Contents

Chapter 1

Overview

This document serves as the M in RTFM for users of `DECO` wishing to enumerate operator bases in weakly-coupled (SMEFT-like) EFTs with continuous and discrete symmetries.

It is intended for first-time users, but structured in the hope that repeat users can quickly find the solutions to their problems.

We include a section on commonly encountered problems, if you encounter an interesting one, please get in touch!

Chapter 2

Initial installation and setup

`DECO` uses the `FORM` symbolic manipulation system, which can be found at `https://www.nikhef.nl/~form/`. No particular optimisations for parallelisation (i.e. `ParFORM` or `TFORM`) are implemented in `DECO`, but parallelisation may still speed up the calculations somewhat. Whether to use it is up to the user.

`DECO` itself consists of two files:

- A `.frm` file containing the information about the EFT, i.e. a file which contains the user input of fields, charges, and external parameters.
- `DECO.h`, which contains the procedures, functions, and definitions that are used for all steps along the Hilbert Series evaluation, i.e. the definition of the plethystic exponentials, Haar measures, characters, etc. It is not particularly interesting for a typical user.

Multiple template `.frm` files, corresponding in particular to the models investigated in the main paper, are included in the repository. These can be used and edited as needed.

To use `DECO`, a user must modify a `.frm` according to the EFT they wish to consider, and run it using `FORM`[1]. The `.frm` file can be renamed freely, edited as desired, and copied as needed, but `DECO.h` should be left untouched.

[1]i.e. for a model file called `mymodel.frm` the command to run `DECO` is `form mymodel.frm`

Chapter 3

Global flags and parameters

The common flags and parameters that must be set for every run of `DECO` sit near the beginning of the `.frm` file, in the block (using `SMEFT.frm` as a template here)

```
#define MASSDIM "6"
#define EOM "1"
#define IBP "1"
#define numFermGen "1"
```

These parameters encode the following:

`MASSDIM` must be set to the mass dimension, i.e. the EFT order that the user is interested in. The out-of-the-box setting here is `"6"`, corresponding to the second subleading operator basis (4 being the leading order Lagrangian, of course). In the case of the SMEFT this would return the number and field content of the operators in the Warsaw basis.

`EOM` is set to either `1` or `0`, and switches equation of motion redundancies on or off. Setting `EOM` to `"0"` would not remove operators that can be eliminated by substituting the leading order equations of motion, setting it to `"1"` removes them.

`IBP` is set to either `1` or `0`, and switches integration by parts redundancies on or off. Setting `IBP` to `"0"` would leave total derivatives (or operators differing from others by total derivatives) as part of the basis, setting it to `"1"` removes these surface terms.

`numFermGen` acts as a blanket multiplier for the fermions in the basis. Setting `numFermGen` to any integer value other than `"1"` simply replaces every fermion that is defined as a dynamic field in the EFT with the `numFermGen` identical copies. This replacement is made implicitly, i.e. the output still uses the single label for the fields that is defined the the list of fields. Instead it should be read as every fermion field label carrying an index, and the increased number of operators we find corresponds to the possible index combinations. We illustrate this below.

Form also allows us to set `numFermGen` to a variable (e.g. `nf`), in which case the number of operators is given as a polynomial in the variable.

As an example for these settings, consider the SMEFT: If we are interested in the Weinberg operator, we set `MASSDIM "5"`, and include a Higgs doublet `h`, and a charged lepton doublet `l` in the list of fields (see 4 on how to do that), alongside their conjugates `hd` and `ld`, with the usual charges. Setting `numFermGen "1"` results in the `DECO` output

```
HS =
      + hd^2*ld^2
      + h^2*l^2
     ;

Number of operators at mass dimension 5 is 2.
```

corresponding to the Weinberg operator $Q_{\nu\nu}$ (in Warsaw basis notation) of two lepton fields and two Higgs fields, as well as its hermitian conjugate, which is counted independently. `DECO` finds two independent real operators, in other words: $Q_{\nu\nu} + Q_{\nu\nu}^\dagger$ and $\mathrm{i}(Q_{\nu\nu} - Q_{\nu\nu}^\dagger)$.

Setting `numFermGen "3"` by contrast replaces the lepton with three identical copies implicitly, and results in

```
HS =
      + 6*hd^2*ld^2
      + 6*h^2*l^2
     ;

Number of operators at mass dimension 5 is 12.
```

corresponding to the six flavour combinations possible with three fermion generations (ee, $\mu\mu$, $\tau\tau$, $e\mu$, $e\tau$, $\mu\tau$), for each of the two real Weinberg operators.

Finally, setting `numFermGen "nf"` (and adding `nf` to the line of symbols) results in

```
HS =
      + 1/2*hd^2*ld^2*nf
      + 1/2*hd^2*ld^2*nf^2
      + 1/2*h^2*l^2*nf
      + 1/2*h^2*l^2*nf^2
     ;

Number of operators at mass dimension 5 is 2.
```

corresponding to $\frac{n_f(1+n_f)}{2}$ flavour combinations for each of the two independent operators, reducing to the values we found previously for $n_f = 1$ and $n_f = 3$.

As is readily visible, we've also stumbled across a limitation of `DECO`:

The `counting` procedure that returns the total number of operators in a basis cannot accommodate parameters, and implicitly sets any such variable (here `nf`) to 1.

Switching `EOM` or `IBP` on or off does not make a difference here, as the dimension-five basis does not contain any derivatives. However, looking at the Warsaw basis instead, we find with settings `MASSDIM`

`"6"`, `EOM "1"`, `IBP "1"`, `numFermGen "1"` a grand total of 84 operators, as we expect. If we switch EOM redundancies off by setting `EOM "0"`, this increases to 183, and we find entries such as `h*hd*p^4` in the list, which corresponds necessarily to the operator $\Box^2\varphi^\dagger\varphi$. There is only one such entry, as all other operators with the same field and derivative content are related to this one by IBP relations. Consequently, if we also switch IBP relations off by setting `IBP "0"`, the total number of operators increases to 298, and we find the entry `6*h*hd*p^4` in the list of operators. This corresponds to the six possible ways of distributing four derivatives onto two fields: $\varphi^\dagger\Box^2\varphi$, $(\Box^2\varphi^\dagger)\varphi$, $(\partial_\mu\varphi^\dagger)\partial^\mu\Box\varphi$, $(\partial_\mu\Box\varphi^\dagger)\partial^\mu\varphi$, $(\Box\varphi^\dagger)\Box\varphi$, $(\partial^\mu\partial^\nu\varphi^\dagger)\partial_\mu\partial_\nu\varphi$.

Chapter 4

Defining the field content

Below the global flags and parameters, the EFT's dynamic fields are specified. This is done by virtue of a sum of terms, with one term for each field in the EFT.

As an example, the SMEFT.frm file lists

```
Symbol h,hd,q,qd,u,ud,d,dd,l,ld,e,ed,B,W,G,p;

Local Input = Scalar(h,SU2(2),U1(3))
       + Scalar(hd,SU2(2),U1(-3))
       + LHFermion(q,SU3(3),SU2(2),U1(1))
       + RHFermion(qd,SU3(B),SU2(2),U1(-1))
       + RHFermion(u,SU3(3),U1(4))
       + LHFermion(ud,SU3(B),U1(-4))
       + RHFermion(d,SU3(3),U1(-2))
       + LHFermion(dd,SU3(B),U1(2))
       + LHFermion(l,SU2(2),U1(-3))
       + RHFermion(ld,SU2(2),U1(3))
       + RHFermion(e,U1(-6))
       + LHFermion(ed,U1(6))
       + FieldStrength(B)
       + FieldStrength(W,SU2(3))
       + FieldStrength(G,SU3(8));
```

as its input, and thus corresponds to the SMEFT field content (and charges, which will be detailed in section 5 below):

At the top, every field is included (with its respective label) in the list of symbols that FORM operates on. The label p for derivatives must always be included, and for every new field that a user defines, another entry must be added to this line.

Forgetting to add a field symbol is by far the most common error made by DECO users. Always make sure all fields you've added to the sum are also represented among the symbols.

We can in particular also note here that there are more entries than there are fields in the Standard

Model, as each conjugate field which differs from the field itself must be defined as its own entry in the sum, and represented by its own label in the list. Hence we find `q` and `qd` representing the lefthanded quark q_L and its conjugate $\bar{q}_L$, respectively, but only one label `W`, as the W field strength is hermitian: $W^\dagger_{\mu\nu} = W_{\mu\nu}$.

The syntax for the fields added is schematically of the form

```
LorentzRepresentation(label,charges)
```

where `label` can be freely chosen (constrained by the fact that `FORM` must be able to recognise it as a symbol) but should not be in use by `DECO` already (please don't call your fields "`SU3`", "`p`", "`MASSDIM`", or something similar that might be in use already).

`charges` is a comma-separated list of the various charges and will be detailed below in section 5.

The Lorentz representations available for the fields are

Label	Lorentz representation
`Scalar`	$(0,0)$
`LHFermion`	$(\frac{1}{2},0)$
`RHFermion`	$(0,\frac{1}{2})$
`DiracFermion`	$(\frac{1}{2},0) \oplus (0,\frac{1}{2})$
`FieldStrength`	$(1,0) \oplus (0,1)$
`Gravity`	$(2,0) \oplus (0,2)$

`DiracFermion` is de facto a short hand, it amounts to defining both a lefthanded and righthanded fermion with the same name and charges.

If you have an exotic representation you'd like to include which we don't allow for yet, please get in touch, the list can be extended.

Chapter 5

Setting the charges

The symmetry groups available, as well as the representations under which fields can transform are

Group	Label	Representation	Representation label
U(1)	`U1`	charge (integer)	`...,-3,-2,-1,0,1,2,3,...`
SU(2)	`SU2`	trivial	`1`
		fundamental	`2`
		adjoint	`3`
SU(3)	`SU3`	trivial	`1`
		fundamental	`3`
		antifundamental	`B`
		adjoint	`8`
A_4	`A4`	1	`1`
		$1'$	`p1`
		$1''$	`pp1`
		3	`3`
S_4	`S4`	1	`1`
		$1'$	`p1`
		2	`2`
		3	`3`
		$3'$	`p3`
U(1) with residue	`U1R`	residue X, charge C (integers)	`R(X,C)`
$\mathbb{Z}_n$	`Zn`	cycle X, charge C (integers)	`R(X,C)`

A field can transform under any combination of arbitrarily many of these, but a few base rules are important.

First, the syntax for the charges under the groups is such that the whole list of charges is collected by group, i.e. for a Lorentz scalar `s` you define

```
Scalar(s,group1(charge1a,charge1b,...),group2(charge2a,charge2b,...),...)
```

where `group1`, `group2`, ... are the labels for the particular mathematical group to which the symmetries correspond, and `charge1a`, `charge2a`, ... are the charges under the relevant groups of a given mathematical type. So a Lorentz scalar s, charged under two distinct SU(3) groups, under which it transforms as a triplet and anti-triplet, respectively, as well as charged under three SU(2)s as a triplet, singlet, and doublet, respectively, would be defined as

```
Scalar(s,SU3(3,B),SU2(3,1,2))
```

The representations that are available, as well as the representation labels you can use (like e.g. the `B` for the anti-triplet in the example) are listed in the table above.

Note in particular that for U(1) groups you can only use integer values. So rescale your U(1) groups accordingly — see the hypercharge in the `SMEFT.frm` template (rescaled by 6) as an example.

Two types of groups are special here, the $\mathbb{Z}_n$ family, as well as U(1) groups with residue (denoted in the following as residual-U(1)). The latter is a special type of U(1), inspired by supersymmetry concerns, for which we do not seek *singlet* operators, but operators with an overall charge. In other words, if you equip a field with `U1R` charge, you demand that all operators in the basis should have an overall residual charge, which you must specify.

It should become clear by now why the $\mathbb{Z}_n$ and residual-U(1) groups are special: In addition to the charge itself, the user must specify which member of a family of groups the charge corresponds to. In the case of $\mathbb{Z}_n$ the cycle length (is it a $\mathbb{Z}_2$, or $\mathbb{Z}_3$,...?) must be specified, in the case of residual-U(1) the value of the residue must be specified.

To do this, the representations are now tuples of two numbers: the charge itself, as well as the cycle length or residue. The syntax for (again) a scalar `s` is thus

```
Scalar(s,Zn(R(N1,C1),R(N2,C2),...),U1R(R(X1,C1),R(X2,C2),...))
```

where `C1`, `C2`, ... are the charges under the respective groups, `N1`, `N2`,... are cycle lengths for $\mathbb{Z}_n$, and `X1`, `X2`,... are residual charges for the residual-U(1) groups. Note that each representation is not merely given as a tuple, but carries the letter `R` in front of it.

As an example, a scalar `s` that is charged under a $\mathbb{Z}_2$, as well as two $\mathbb{Z}_3$ symmetries, for the latter with charges 1 and 2, respectively[1], and which carries charge 1 under a U(1) for which we demand a residual charge for the operators of 4, is defined as

```
Scalar(s,Zn(R(2,1),R(3,1),R(3,2)),U1R(R(4,1)))
```

It should be clear that for many different groups this can become difficult to oversee. So while we recommend that you try to stick to one particular ordering of symmetries and charges, we add a few explanatory rules and guidelines following the obligatory warning:

We strongly recommend that you specify all charges for a field, even if it is a singlet under some symmetry group(s), to avoid accidentally assigning the wrong charges, and that you choose the same order for groups and representations.

[1]i.e. s transforms as $s \to \tau s$ under the first $\mathbb{Z}_3$, and as $s \to \omega^2 s$ under the second, with $\tau^3 = \omega^3 = 1$.

So here are the internal rules for `DECO`:

- All groups defined anywhere apply to all fields. So if you define two fields charged under SU(3) and SU(2) in the sum as `Scalar(s,SU3(3),SU2(1))+Scalar(t,SU3(8),SU2(2))`, this is processed as both fields being charged under *the same* SU(3) and SU(2) groups: s is a triplet and t is an adjoint under the same SU(3), and s is a singlet under the very SU(2) under which t is a doublet.
- The order of the groups is arbitrary, the order of charges is not (more on the latter below). With this we mean that e.g. the definition `Scalar(s,SU3(8),SU2(3))` (an adjoint under an SU(3) and a triplet under an SU(2)) is identical to `Scalar(s,SU2(3),SU3(8))`. The groups are processed in the order Lorentz→U(1)→ U(1)$_R$ →SU(2)→SU(3)→ $\mathbb{Z}_n$ → A_4 → S_4, but the order in which the blocks for the mathematical groups (i.e. the expressions like `SU3(charge1,charge2,...)`) are put in the definition is irrelevant.
- A missing charge is interpreted as a singlet. So for two fields `Scalar(s,SU3(8))+Scalar(t)` in the sum of fields there is one SU(3) taken to be at play: s is an adjoint under it, and t a singlet.
- While the order of groups is irrelevant, the order of representations is very relevant: Generally the left-most representation in a charge definition is processed first! (see comment about `Zn` and `U1R` below) This means that a field carrying a charge definition of `SU3(8,3)` is taken as an adjoint under the first SU(3) to be processed, and a triplet under the second SU(3) to be processed.
- This means that if you have two fields transforming under *distinct* groups *of the same mathematical type*, you *must* include the singlet charge assignments. As an example, if you have a left-right symmetric model in which lefthanded fermions transform as doublets under an SU(2)$_L$, and righthanded fermions under an SU(2)$_R$, these must be defined as `LHFermion(l,SU2(2,1))` and `RHFermion(r,SU2(1,2))`, respectively. Here the SU(2)$_L$ corresponds to the first entry in the `SU2(.,.)` block, and the SU(2)$_R$ to the second.
- The previous points together imply that *trailing singlets* ***can*** *be omitted.* We consider doing this, however, to be a ***REALLY BAD IDEA***™. So technically the two fermions from the previous example *could* also be defined as `LHFermion(l,SU2(2))` and `RHFermion(r,SU2(1,2))`. The charges are processed from the left, so the SU(2)$_L$ is processed first, after which only the SU(2)$_R$ remains, under which the lefthanded field is a singlet. And that is identical to a field in which the charge is not specified. The reason this is not a good idea comes from the next item:
- For the $\mathbb{Z}_n$ and residual-U(1) symmetries, the same rules apply, with one additional one: The order in which the various *members* of the group family appear is machine- and version-dependent, as it depends on FORM's internal ordering. As an example, consider the input

  ```
  Scalar(s,Zn(R(3,1),R(2,1)))+Scalar(t,Zn(R(2,1)))
  ```

  Depending on how the fields are ordered in an expression, FORM may first process the $\mathbb{Z}_3$ which is the first entry in the charge list for `s`, or the $\mathbb{Z}_2$ which is the first entry in the charge list for `t`.

  In the former case, the $\mathbb{Z}_3$ symmetry is processed first, and t is assumed to be a singlet under it (as the first charge entry for t is a $\mathbb{Z}_2$ entry, i.e. a different symmetry altogether). Subsequently only a $\mathbb{Z}_2$ remains, and according to the rules we just listed the fields are assumed to both transform under it (the same group family member is first in line in both fields, so `DECO` assumes it's the same symmetry).

  Alternatively the $\mathbb{Z}_2$ of `t` is processed first, and `s` is taken to be a singlet under it (because the first entry is a $\mathbb{Z}_3$, i.e. a different symmetry). Then in turn the $\mathbb{Z}_3$ and $\mathbb{Z}_2$ of `s` are processed, and these only affect `s` (because the only family member specified for `t` was already covered). The three symmetries all act on only one field, in other words. Which is of course a different pattern than the first case we discussed.

To mitigate this risk, we have included some pattern matching procedures which should check whether the $\mathbb{Z}_n$ and residual-U(1) symmetries are implemented correctly, but it is not a good idea to tempt fate.

Thus, following our ***VERY SENSIBLE RECOMMENDATION***™ in the blue box above, reiterated in the red box below, eliminates any potential ambiguity: Defining

```
Scalar(s,Zn(R(3,1),R(2,1)))+Scalar(t,Zn(R(3,0),R(2,1)))
```

makes it clear that the $\mathbb{Z}_3$ for s does not act on t, and both transform under the same $\mathbb{Z}_2$, and it does not matter how any expressions are ordered: the $\mathbb{Z}_3$ is done first, followed by the $\mathbb{Z}_2$.

To recap this last point in a short sentence: While the groups SU(3), SU(2), U(1), A_4, and S_4 are already bad enough with their implied *trailing* singlets, the groups $\mathbb{Z}_n$ and residual-U(1) may potentially suffer from implied *leading* singlets, which is even worse, as it leads to undefined behaviour.

We therefore reiterate our warning:

We strongly recommend that you specify all charges for a field, even if it is a singlet under some symmetry group(s), to avoid accidentally assigning the wrong charges, and that you choose the same order for groups and representations.

Chapter 6

Running DECO

To run DECO on a file called mymodel.frm, simply excute

```
form mymodel.frm
```

The output is a sum of all operator structures, ordered by field content. Integer multiples denote multiple operators with the same field content but different index contractions. A welcome message, some machine- and version-dependent FORM info, the total number of operators, and the runtime of the program are also printed to the screen, here as an example for the SMEFT at dimension five, i.e. the Weinberg operators:

```
FORM 4.2 (Oct 17 2022) 64-bits              Run: Fri Nov 25 10:52:10 2022
    #-
Running DECO 1.0.

   HS =
       + hd^2*ld^2
       + h^2*l^2
      ;

Number of operators at mass dimension 5 is 2.
  0.06 sec out of 0.07 sec
```

Chapter 7

Processing the output

Here we discuss some FORM commands that can be useful in studying the output of DECO. Of course, this is just a small subset of all the FORM commands that can be used in the study of the operator basis, but we suspect that these are the most important ones. We expect that users with FORM experience are aware of these commands, but nevertheless find it useful to indicate what is possible with the code.

7.1 Counting

A procedure used in all the template files is the `counting` procedure, which — as the name suggests — counts the number of operators in a Hilbert Series output. It is called explicitly on the expression by appending the command to the relevant `.frm` file (e.g. here for `2HDM.frm`):

```
...

#call HilbertSeries(p)
Print +s;
.sort

#call counting(HS)
.end
```

It takes as input a `FORM` expression, and returns the number of elements in it. The default Hilbert series output is `HS`, so `#call counting(HS)` returns the total number of operators. Below we will introduce the `id` and `Brackets` commands, which can be used to select subsets of operator bases (by e.g. selecting all operators with a certain number of fields of one type, or without a given field altogether) and storing them in `FORM` expressions. The `counting` procedure can then be used to count the elements in these subsets, as e.g. demonstrated in 7.2 and 7.3.3.

7.2 Brackets

The `Brackets` statements can be used to reorganize the output to place a FORM object outside brackets and keep all remaining objects inside these brackets. For example, we can easily sort by the number of derivatives (with symbol `p`) by adding the line `Brackets p;` to the code. For a simple example with only one scalar and one fermion field the input then might be

```
...
#define MASSDIM "8"
#define EOM "1"
#define IBP "1"
#define numFermGen "1"

Symbol s,sd,q,qd,p;

Local Input = Scalar(s,SU2(2),U1(1))
   + Scalar(sd,SU2(2),U1(-1))
   + LHFermion(q,SU3(3))
   + RHFermion(qd,SU3(B));
.sort

#call HilbertSeries(p)
Brackets p;
Print;
.sort

#call counting(HS)
.end
```

resulting in the output

```
   HS =
       + p * ( s^2*sd^2*q*qd )

       + p^2 * ( 2*q^2*qd^2 + 2*s^3*sd^3 )

       + p^3 * ( 2*s*sd*q*qd )

       + p^4 * ( 3*s^2*sd^2 )

       + s*sd*q^2*qd^2 + s^4*sd^4;
```

The bracket information can now be used to, for example, store all operators with one and two derivatives in another `Local` expression by modifying the file's last lines to include the following code (note that the definition of a new local expression has to happen in a new module, i.e. after the `.sort` command):

```
...

#call HilbertSeries(p)
Brackets p;
```

```
.sort

Local HS2 = p*HS[p] + p^2*HS[p^2];
Print;
.sort

#call counting(HS)
#call counting(HS2)
.end
```

resulting in

```
   HS =
      2*q^2*qd^2*p^2 + 2*s*sd*q*qd*p^3 + s*sd*q^2*qd^2 + 3*s^2*sd^2*p^4 + s^2*
      sd^2*q*qd*p + 2*s^3*sd^3*p^2 + s^4*sd^4;

   HS2 =
      2*q^2*qd^2*p^2 + s^2*sd^2*q*qd*p + 2*s^3*sd^3*p^2;

Number of operators at mass dimension 8 is 12.
Number of operators at mass dimension 8 is 5.
```

where we also demonstrated the `counting` procedure in action on two distinct expressions: The full Hilbert Series output `HS`, and the reduced subset `HS2`.

To get the zeroth order term in `p`, one can use `Hilbert[1]`. Note that the bracket information is only active during the next module. Furthermore, only a single object can be placed outside the brackets.

The `AF_A4.frm` template for the Altarelli-Feruglio A_4-based model includes this command as a live example.

7.3 Id

7.3.1 Discarding operators

In many cases it can be useful to study a subset of the generated operator basis, or to replace symbols to get around the `Brackets` restriction on single objects outside the brackets. In this case the `id` (identify) statement can be useful. For example, we can study all operators that are purely fermionic in the previous section's model by executing

```
...

#call HilbertSeries(p)
Brackets p;
.sort

id s = 0;
id sd = 0;
```

```
Print;
.sort

#call counting(HS)
.end
```

resulting in

```
   HS =
      2*q^2*qd^2*p^2;

Number of operators at mass dimension 8 is 2.
```

where the `id` statement set all occurrences of `s` and `sd` equal to 0.

7.3.2 Splitting operator bases

The `id` statement can also be used to replace objects by a more complicated structure. For example, we could use the `id` command to equip all fields `s` with a factor: `id s = k*s` (don't forget to declare the symbol). If we then set all squares of `k` to 1: `id k^2 = 1`, the output is split by appearance or non-appearance of `k`.

In terms of code, this means if we start with

```
...
#call HilbertSeries(p)
Brackets p;
Print;
.sort

Symbol k;
id s = k*s;
id k^2 = 1;
Brackets k;

Print;
.sort

#call counting(HS)
.end
```

we find as output

```
...

HS =
      + k * ( 2*s*sd*q*qd*p^3 + s*sd*q^2*qd^2 + 2*s^3*sd^3*p^2 )

      + 2*q^2*qd^2*p^2 + 3*s^2*sd^2*p^4 + s^2*sd^2*q*qd*p + s^4*sd^4;
```

which is the output ordered by odd vs. even powers of the field `s` — the poor man's implementation of $\mathbb{Z}_2$ behaviour.

Note that `id` is not identical to the Mathematica replacement rule "`->`": `id k^2 = 1` also replaces *powers* of `k^2`, i.e. expressions such as `k^4` by 1.

7.3.3 Mass dimension gymnastics

Finally, `id` can be used to implement some crude but necessary hacks. As an example, in the Altarelli-Feruglio A_4-based model some scalars should have mass dimension 2. In most cases a dedicated mass dimension variable is not needed, as elementary fields have fixed mass dimensions, which we hardcode.

We can, however, still implement these variable mass dimensions using the `id` command: We include the field which should have higher mass dimension as a bog-standard field. So e.g. for an SU(2) triplet scalar s with mass dimension 3 we'd include the usual `Scalar(s,SU2(3))`. We however *also* include an auxiliary inert field s_{aux} as `Scalar(saux)`. We now use the `id` feature to isolate all expressions containing powers of the combination `s*saux^2`, which has the same charges as `s`, and mass dimension 3. If we rescale $s \to k^2 s$ and $s_{\text{aux}} \to s_{\text{aux}} k^{-1}$, then the combination `s*saux^2` is unchanged, while all mismatched combinations (like $s^2 s_{\text{aux}}$) pick up powers of `k`.

So we use the `id` feature in our code and our input looks like this:

```
...
Symbol saux,s,p;

Local Input = Scalar(s,SU2(3))+Scalar(saux);
.sort

#call HilbertSeries(p)
.sort

Symbol k;

id s = k^2*s;
id saux = saux/k;
id saux = 1;
Brackets k;
.sort

Local MassdimCorrected = HS[1] ;
...
```

where we also removed the auxiliary field for aesthetic purposes. At mass dimension 6 we thus now get the output

```
  HS =
    k^-6 + s^2 + s^2*p^2*k^2 + s^4*k^6 + s^4*p^2*k^8 + s^6*k^12;

  MassdimCorrected =
    s^2;
```

where `MassdimCorrected` now lists the single operator $s^T s$, which has mass dimension 6 if s has mass dimension 3, and is an SU(2) singlet.

7.4 Saving the output

The output can be studied in the `.frm` file with the commands discussed above. However, sometimes it can be useful to run the `.frm` file several times with just a few minor changes in the input and study the differences. In that case it can be useful to save the output/local expressions such that these can later be opened in another FORM file.

7.4.1 Save

In order to save the output of the main program, we need to make a `Global` expression out of `Local Hilbert`. The difference between a `Local` and `Global` expression is that the former are dropped at the end of a run, whereas the latter are stored when FORM encounters a `.store`. The stored expressions in the `store` file are not active (see FORM manual), but they can be copied to external files for later use by the `Save` command in FORM.

We will now look at an example in which we compare the operator bases in which EOM relations are used to simplify the basis or not. Therefore, we run the `.frm` file once with the variable `EOM` equal to 1 and once equal to 0, store the result and compare the result in another file. To do this, we insert the code fragment

```
Global HilbertEOM‘EOM’ = HS;
.store

Save HilbertEOM‘EOM’.sav;
.sort
```

at a suitable place between the `#call HilbertSeries(p)` and `.end` lines. Here we named the `Global` expression `HilbertEOM‘EOM’` and the preprocessor of FORM will replace the variable `‘EOM’` by its current value. The `Global` expression is stored in the file `HilbertEOM‘EOM’.sav`. After both runs we will thus have two files, `HilbertEOM1.sav` and `HilbertEOM0.sav`, with the respective operator bases stored.

7.4.2 Load

We can load saved expressions in another FORM file with the `Load` command of FORM. Just as with stored expressions, loaded expressions are not active. They can be made active by defining a new `Local` expression to which we copy the loaded expressions.

```
Load HilbertEOM1.sav;
Load HilbertEOM0.sav;
.sort

Local Difference = HilbertEOM0 - HilbertEOM1;
Print;
.end
```

In this case we have copied HilbertEOM0 and subtracted the operators where EOM relations are removed. Therefore the printed output yields all operators which can be related to other operators by EOM relations.

7.4.3 Save to Mathematica

In addition to saving FORM output to use with other FORM programs, you can also save the Hilbert series in Mathematica syntax using the saveto procedure:

Adding the line #call saveto(expression,file) to your .frm file in the same manner as e.g. the counting procedure saves the contents of the FORM expression expression to a file called file in Mathematica syntax. You can then load it into Mathematica using the Get command — or just old-fashioned Ctrl-C & Ctrl-V.

As an example, our non-standard mass dimension case from subsection 7.3.3 might end in

```
...
Local MassdimCorrected = HS[1] ;
Print;
.sort
#call saveto(MassdimCorrected,MathOutput.m)
.end
```

The file MathOutput.m then contains the single remaining operator:

```
(
     s^2
)
```

This procedure is included by default in the SMEFT.frm file, and saves the relevant operator basis at mass dimension MASSDIM in the file SMEFTdimMASSDIM.m.

Chapter 8

Requesting additional features

The groups and representations in `DECO` have been included based on necessity: They correspond to groups and representations we encountered when playing with discrete flavour symmetries and the Hilbert series. If you have a specific EFT based on additional representations (higher spins, higher representations for the SU(N) groups, additional SU(N) groups, additional finite groups,...) please get in touch! To add additional Lie groups we merely need the Haar measures and characters for the representations to be implemented, for additional discrete groups we need the eigenvalues of the representations. All of these are usually somewhere findable in the literature, or can be derived. So if you've got a request, don't be shy!

Chapter 9

Commonly encountered problems

First, check the following:

- Have you declared all symbols?
- Have you defined all charges correctly, and in particular also included all singlets?
- Are your U(1) charges all integers?
- If you play with additional features, make sure you are in a new module (i.e. by including a `.sort`)

If you get an error message `"Program terminated during symmetries of X. Check user input, e.g.  charges etc."`, then you defined the charges for the group 'X' incorrectly. Typically this means you used fractional U(1) charges, or an undefined representation (something like `SU2(8)`, which does not exist).

If the first error message contains `Unmatched ()`, you probably have forgotten to open or close a bracket.

`Illegal position` usually means you've forgotten a comma.