Manuals for **feyngen** and **feyncop** - Two programs for calculations with Feynman graphs

Michael Borinsky
Institutes of Physics and Mathematics
Humboldt-Universität zu Berlin
Unter den Linden 6
10099 Berlin, Germany
E-mail address: borinsky@physik.hu-berlin.de

November 3, 2014

1 Feyngen and Feyncop

feyngen is a program to generate Feynman graphs for the use in perturbative calculations of quantum field theory.

feyncop is a program to calculate the coproduct of Feynman graphs in the scope of the Hopf algebra of Feynman graphs.

Both programs are designed to be used at relatively high loop orders where traditional programs like QGRAF are not applicable. For instance the established nauty package is used to ensure high performance.

The theoretical foundation of the programs with details to validation and implementation is outlined in my paper M. Borinsky, *Feynman graph generation and calculations in the Hopf algebra of Feynman graphs*, Computer Physics Communications (2014) and in my master's thesis on Algorithmization of the Hopf algebra of Feynman graphs.

Please cite M. Borinsky, *Feynman graph generation and calculations in the Hopf algebra of Feynman graphs*, Computer Physics Communications, Volume 185, Issue 12, December 2014, Pages 33173330 if you want to refer to the programs or this work.

1.1 Download

The source code for both programs can be downloaded from github. A pre-built version is also available on my website, where you can also find the newest version of this manual.

1.2 Prerequisites

To use either of the programs Python 2.7 with development files must be installed on your machine. For information on how to install Python please consult `http://www.python.org/`.

Additionally if you do not use the pre-built version, the nauty package by Brendan McKay is needed. The newest version can be downloaded from:

`http://pallini.di.uniroma1.it/`

1.3 Installation

This step can be skipped if you want to use the pre-built version.

Copy the **feyncop** and nauty archives into the same directory and extract them:

```
$ tar xzf feyncop.tar.gz
$ tar xzf nautyXXXX.tar.gz
```

Where XXXX are some letters representing the current nauty version.

Next, change the name of the folder containing the nauty package:

```
$ mv nautyXXXX/ nauty/
```

and build the nauty package:

```
$ cd nauty/
$ ./configure && make
$ cd ../
```

Now, **feyncop** and **feyngen** can be build:

```
$ cd feyncop/
$ make
```

The two python programs **feyngen** and **feyncop** in the `feyncop/` directory should know be working as expected.

An overview of the parameters of the two programs is displayed with

```
$ ./feyngen --help
```

or

```
$ ./feyncop --help
```

.

1.4 Testing

To test **feyngen** run,

```
$ ./feyngen 2 -j2
```

in the `feyncop/` directory. The output should be:

```
phi4_j2_h2 :=
+G[[0,0],[0,0],[1,1],[1,1],[3,2]]/128
+G[[0,0],[1,1],[1,1],[2,0],[3,0]]/16
+G[[1,0],[1,0],[1,1],[2,0],[3,0]]/4
+G[[0,0],[1,0],[1,0],[1,1],[3,2]]/16
+G[[1,0],[1,0],[1,0],[1,0],[3,2]]/48
+G[[0,0],[1,0],[1,1],[2,0],[3,1]]/4
+G[[1,0],[1,0],[1,0],[2,0],[3,1]]/6
;
```

Corresponding to the sum of all 2-point, 2-loop diagrams in φ^4-theory. To test **feyncop** run,

```
$ ./feyngen 2 -j2 -p | ./feyncop -u
```

the output should be:

```
phi4_j2_h2_red_cop_unlab :=
+ 1/4 * T[ G[[0,0],[1,0],[2,0]], G[[0,0],[1,0],[2,0]] ]
+ 3/4 * T[ G[[1,0],[1,0],[2,0],[3,0],[4,1],[5,1]], G[[0,0],[1,0],[2,0]] ]
;
```

This output corresponds to the coproduct of the sum of all 1PI, 2-point, 2-loop diagrams in φ^4-theory.

2 Manual of feyngen

2.1 Overview

In this section the commands of **feyngen** together with input and output format is described. Because **feyngen** is a program optimized to generate non-isomorphic high loop Feynman diagrams, the only mandatory parameter for **feyngen** is the order in $\hbar$ in the perturbation series of the class of diagrams to be generated. This corresponds to the number

$$h_1(\gamma) - f_\gamma + 1,$$

where f_γ is the number of connected components of the graphs γ to generate. For connected graphs, $f_\gamma = 1$, this number is equivalent to the loop number $h_1(\gamma)$.

For example, the call to **feyngen**,

```
$ ./feyngen 3 4 5
```

will generate all connected 3, 4 and 5 loop vacuum diagrams and all disconnected diagrams which correspond to these loop orders in respect to their order in $\hbar$ in the perturbation series of φ^4 theory.

2.2 Options and Parameters

Additionally to the loop number, the generation can be controlled by various options. **feyngen** called with the option **--help** prints the list of all possible program options together with a short description.

```
$ ./feyngen --help
```

For instance, only graphs with certain properties can be generated. These restrictions can be set by the following options:

-c / --connected
: Generate only connected graphs.

-p / --1PI
: Generate only 1PI graphs.

-v / --vtx2cntd
: Generate only 2-vertex connected graphs.

-t / --notadpoles
: Generate only non-tadpole graphs.

By default, **feyngen** generates φ^4 graphs. The four options:

-k# / --valence=#
: Generate graphs with vertex valence # for a scalar $\varphi^{\#}$ theory.

--phi34
: Generate scalar graphs with 3 or 4 valent vertices ($[\varphi^3 + \varphi^4]$-theory).

--qed
: Generate graphs for QED (with neglect of Furry's theorem).

--qed_furry
: Generate graphs for QED (respecting Furry's theorem).

--ym
: Generate graphs for Yang-Mills theory.

can be used to alter the type of graphs generated. The external leg structure of the graphs is determined by the parameters,

-j# / --ext_legs=#
: Set the total number of external legs # of φ^k graphs.

-b# / --ext_boson_legs=#
: Set the number of external boson legs # of QED or Yang-Mills graphs.

-f# / --ext_fermion_legs=#
: Set the number of external fermion legs # of QED or Yang-Mills graphs.

-g# / --ext_ghost_legs=#
: Set the number of external ghost legs # of Yang-Mills graphs.

depending on whether graphs for φ^k-theory, for QED or for Yang-Mills are generated. By default, only graphs without external legs are given as output.

Additionally, the behaviour under graph isomorphisms of the external legs can be controlled:

-u / --non_leg_fixed
: Generate non-leg-fixed graphs. External legs of graphs are not considered as fixed if this option is set. This option influences isomorphism testing and symmetry factor calculation of graphs.

If not stated otherwise, leg-fixed diagrams are generated.

2.3 Output of φ^k-graphs

Representation of graphs Graphs are represented as edge lists. An edge is represented by a pair of vertices or a triple of two vertices and a letter, representing the type of the edge. The types used by **feyngen** are `f`, `A` and `c` representing fermion, gauge boson and ghost edges. The pairs or triples are embraced by brackets. Vertices are labeled by integers.

So for example

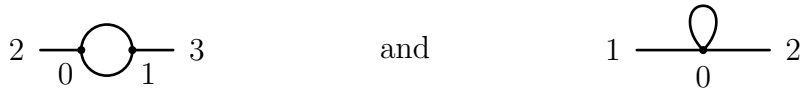
`[2,3]` and `[6,4,f]`

represent one edge without specific type between the vertices 2 and 3 and one fermion edge between the vertices 4 and 6. For ϕ^k graphs no edge type will be specified. For QED and Yang-Mills graphs fermions and ghosts will be oriented edges and gauge bosons will be non oriented edges. This way, `[6,4,f]` can be interpreted as an fermion pointing from vertex 6 to vertex 4.

External edges are not distinguished from internal edges, except that they are incident to an external one-valent vertex.

The edge list is embraced by brackets and prefixed by a `G` to simplify the usage of the output with **maple**. This output format of graphs will be referred to as the `G`-format.

Example The graphs



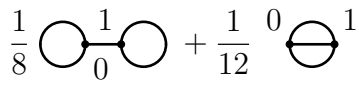
are represented as

`G[[1,0],[1,0],[2,0],[3,1]]` and `G[[0,0],[1,0],[2,0]]`,

where in the first diagram 0 and 1 are the internal vertices and 2 and 3 are external vertices. In the second diagram 0 is the only internal vertex with 1 and 2 depicting external vertices. The labeling of the vertices is auxiliary and is used to give a representative of the isomorphism class of the graph. The labeling is assigned using **nauty**, which chooses a canonical labeling unique for every isomorphism class of graphs. The labeling is chosen, such that the external vertices carry the highest labels.

Output of graph sums The output format of **feyngen** is designed to be readable by a **maple** program. Therefore, the graphs generated by **feyngen** are written as a sum of graphs with every graph weighted by its symmetry factor.

For instance, the sum of all φ^3 2-loop graphs without external legs, with each graph weighted by its symmetry factor, depicted diagrammatically as



can be generated by **feyngen** as follows:

```
$ ./feyngen 2 -k3
phi3_j0_h2 :=
+G[[0,0],[1,0],[1,1]]/8
+G[[1,0],[1,0],[1,0]]/12
;
```

Where the **2** in the command line stands for diagram generation of order $\hbar^2$ and **-k3** for φ^3-theory, such that only graphs with three valent vertices are generated.

Symmetry factors As can be seen in the above example, the graphs are given weighted by their symmetry factor. The format is

```
G[...]/Aut,
```

where `Aut` is the order of the automorphism group of the graph.

Note that in general, the calculation of the symmetry factor depends on the **-u** option if external legs are present, depending on whether external legs are fixed or not. An explicit example for the behaviour of the **-u** option is given in section 2.5.

Distinguished name for maple usage The sum of graphs is given a name, which indicates the loop number(s), the number of external edges and the theory type.

That means, the output is always of the form:

```
phi(k)_j(m)_h(L) :=
+G[...]/Aut1
+G[...]/Aut2
+...
...
;
```

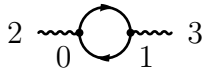
Where `(k)` is replaced with the appropriate theory degree, `(m)` is substituted by the number of external edges and `(L)` is given by the order in $\hbar$ of the graphs.

2.4 Output of QED and Yang-Mills graphs

QED diagrams carry additional information, because they have two different edge types. Consequently, an QED edge `[v1,v2,t]` is depicted as a pair of vertices `v1,v2` together with the edge's type `t`. Possible edge types for QED graphs are `f` for fermions and `A` for photons. For Yang-Mills graphs the possible types are `f`, `A` and `c` for fermions, gauge bosons (e.g. gluons) and Faddeev-Popov ghosts respectively. Fermions and ghosts edges are oriented - gauge boson edges are not.

QED and Yang-Mills Feynman diagrams are represented as edge lists as in the last section. Because they are oriented, fermion and ghost edges are depicted as ordered pairs of vertices.

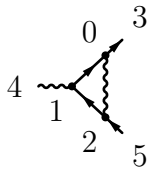
Example For example, the graph



is represented as

```
G[[0,1,f],[1,0,f],[2,0,A],[3,1,A]]
```

and



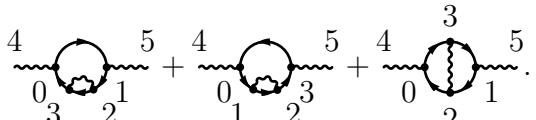
is depicted as

```
G[[1,0,f],[2,1,f],[2,0,A],[0,3,f],[5,2,f],[4,1,A]].
```

The orientation of the fermion lines matches the ordering of the vertices in the edges. **feyngen** only generates graphs with valid QED vertex types.

Output of graph sums The output of graph sums is similar to the output of φ^k graphs. As for the treatment of φ^k graphs, an example for the case of QED and Yang-Mills diagram generation is given.

Examples Consider the sum of all two loop, photon propagator residue type, 1PI, QED diagrams:

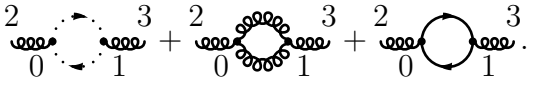


feyngen generates them if it is called with the command line

```
$ ./feyngen --qed 2 -b2 -p
qed_f0_b2_h2 :=
+G[[0,1,f],[1,2,f],[2,3,f],[3,0,f],
    [3,2,A],[4,0,A],[5,1,A]]/1
+G[[0,1,f],[1,2,f],[2,3,f],[3,0,f],
    [2,1,A],[4,0,A],[5,3,A]]/1
+G[[0,3,f],[1,2,f],[2,0,f],[3,1,f],
    [3,2,A],[4,0,A],[5,1,A]]/1
;
```

--qed indicates QED graph generation, **2** stands for 2-loop diagrams ($\hbar^2$), **-b2** makes **feyngen** generate graphs with 2 photon legs and the **-p** option filters out non 1PI graphs.

For the sum of all one loop, gauge boson propagator residue type, 1PI, Yang-Mills diagrams,



the call,

```
$ ./feyngen --ym 1 -tp -b2
```

where the generation of Yang-Mills graphs is triggered with the **--ym** option, gives the desired result:

```
ym_f0_g0_b2_h1 :=
+G[[0,1,c],[1,0,c],[2,0,A],[3,1,A]]/1
+G[[0,1,f],[1,0,f],[2,0,A],[3,1,A]]/1
+G[[1,0,A],[1,0,A],[2,0,A],[3,1,A]]/2
;
```

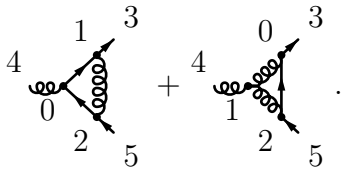
Another example for Yang-Mills graph generation is the command,

```
$ ./feyngen --ym 1 -p -f2 -b1
```

which generates the two one loop, 1PI graphs with two fermion and one gluon leg:

```
ym_f2_g0_b1_h1 :=
+G[[0,1,f],[2,0,f],[2,1,A],[1,3,f],[5,2,f],[4,0,A]]/1
+G[[2,0,f],[1,0,A],[2,1,A],[0,3,f],[5,2,f],[4,1,A]]/1
;
```

Written diagrammatically as,



Distinguished name for maple usage The name of the graph sum for QED diagrams is slightly modified:

```
qed_f(m1)_b(m2)_h(L) :=
...
;
```

For the output of graphs respecting Furry's theorem (see **--qed_furry** option) the line will read:

```
qed_furry_f(m1)_b(m2)_h(L) :=
...
;
```

Instead of `j(m)` indicating the total number of legs as in the case of φ^k graphs, `f(m1)` and `b(m2)` are given with `(m1)` being replaced by the number of fermion legs and `(m2)` by the number of photon legs. For the output of Yang-Mills graphs the pattern is modified in the following way,

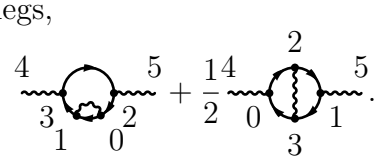
```
ym_f(m1)_g(m3)_b(m2)_h(L) :=
...
;
```

with the same numbers and `(m3)`, the number of ghost legs, to be plugged in.

2.5 Labeled and unlabeled legs

Without the **-u** option the external legs are considered as fixed and leg-fixed diagrams are generated. For this reason the first and the second graphs in the last QED example are not isomorphic and all diagrams carry a symmetry factor of 1. The **-u** option controls this behaviour and influences the generation of non-isomorphic graphs and the calculation of symmetry factors appropriately.

Example Consider again the two loop, photon propagator, 1PI, QED diagrams, but without fixed external legs,



Because of the additional freedom in permuting the edges and vertices, the first two diagrams of the last QED example belong to the same isomorphism class. Furthermore, the second diagram in this example gains an additional symmetry generator of index 2, such that the order of the automorphism group increases from 1 to 2. This can be reproduced using **feyngen** with the **-u** option:

```
$ ./feyngen --qed 2 -b2 -p -u
qed_f0_b2_h2_nlf :=
+G[[0,1,f],[1,3,f],[2,0,f],[3,2,f],
    [1,0,A],[4,3,A],[5,2,A]]/1
+G[[0,2,f],[1,3,f],[2,1,f],[3,0,f],
    [3,2,A],[4,0,A],[5,1,A]]/2
;
```

The text `nlf`, standing for non-leg-fixed, succeeding the name of the graph sum acknowledges this behaviour for later reproducibility.

3 Manual of feyncop

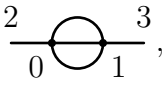
3.1 Overview

This section shall introduce the commands and the output format of **feyncop**.

A graph or a sum of graphs, for which the reduced coproduct shall be calculated, must piped as input into **feyncop**. For example,

```
$ echo "G[[1,0],[1,0],[1,0],[2,0],[3,1]]"
   | ./feyncop -D4
```

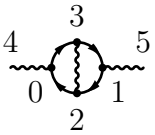
where the **echo** command is used to pipe the input into **feyncop**, will give the three proper non empty subgraphs of the diagram



,

which are, in four dimensions, composed of superficially divergent 1PI graphs. Details to the output format will be given in section 3.4 and the following ones with elaborate examples.

QED graphs are handled the same way, except for the weights that must be given with the edges. For instance, the subgraphs for the reduced coproduct $\widetilde{\Delta}_4$ of the graph



or given in the G-format as

```
G[[0,3,f],[1,2,f],[2,0,f],[3,1,f],
    [3,2,A],[4,0,A],[5,1,A]]
```

can be calculated using the command line

```
$ echo "G[[0,3,f],[1,2,f],[2,0,f],[3,1,f],[3,2,A],
[4,0,A],[5,1,A]]" | ./feyncop -D4
```

.

3.2 Option and Parameters

Similar to **feyngen**,

```
$ ./feyncop --help
```

prints a list of the available options and parameters with a short summary.

The parameter D, altering the dimension parameter of the reduced coproduct Δ_D, is controlled by the option

-D# / --dimension=#
Set the dimension for the calculation of the coproduct.

By default $D = 4$ is assumed.

By default, **feyncop** calculates only the subgraphs, which are composed of superficially divergent 1PI graphs, of the input graphs. A sum of graphs with a list of the subgraphs, composed of superficially divergent, 1PI graphs is given as output. This behaviour can be changed by the options:

-c / --cographs
Calculate and print the cographs additionally to the corresponding subgraphs. Output format: Sum of graphs with a list of the subgraphs, composed of superficially divergent, 1PI graphs, and their cograph.

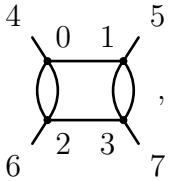
-u / --unlabeled
Transform the subgraphs and the cographs to unlabeled graphs and identify similar tensor products. Output format: Sum of tensor products.

-p / --primitives
Only filter the input graphs for primitive graphs. Output format: Sum of graphs.

3.3 Reference to edges

feyncop represents subgraphs as a set of edges of the original graph. This has the advantage that the location of the subgraph in the original graph is implicitly included in the output. This information is crucial for some applications of the coproduct of a Feynman diagram. The edges are referred to by their index in the edge list of the `G`-format starting with 0.

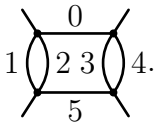
Example The graph, with an auxiliary vertex labeling,



represented in the `G`-format by

```
G[[1,0],[2,0],[2,0],[3,1],[3,1],
    [3,2],[4,0],[5,1],[6,2],[7,3]],
```

will be assigned the following internal edge labels,



Here, labels for external legs were omitted for simplicity. They are not of interest for the description of subgraphs.

3.4 Output of subgraphs

By default, only the subgraphs composed of superficially divergent 1PI graphs are outputted. The output is given as pairs of the original graph in the `G`-format and of the subgraphs, each split into its connected components. These pairs are preceded by a `D` to mark a new object suitable to be read by **maple**. Therefore, giving a single graph as input, the output will take the form:

```
D[ ( original graph ), [
{ { 1. subgraph's 1. connected component's edges },
{ 1. subgraph's 2. connected component's edges },
...
},
{ { 2. subgraph's 1. connected component's edges },
    ... }
...
] ]
```

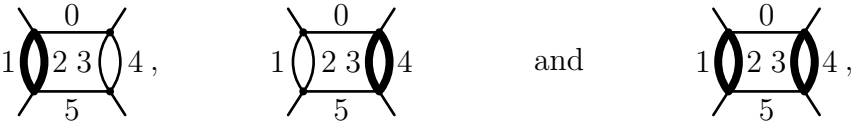
Example The graph  , represented in the `G`-format as above, the output to the call,

```
$ echo "G[[1,0],[2,0],[2,0],[3,1],[3,1],[3,2],
[4,0],[5,1],[6,2],[7,3]]" | ./feyncop -D4
```

will look as follows:

```
+ D[G[[1,0],[2,0],[2,0],[3,1],[3,1],
    [3,2],[4,0],[5,1],[6,2],[7,3]],
[{{1,2}}, {{3,4}}, {{1,2},{3,4}}]]
;
```

The original graph is paired with the information about the subgraphs composed of superficially divergent components in `[{{1,2}}, {{3,4}}, {{1,2},{3,4}}]`. The three sets in the list correspond to the three subgraphs, indicated by thick lines,



represented as the sets of sets,

`{{1,2}}`, `{{3,4}}` and `{{1,2},{3,4}}`.

The subgraphs are split into their connected components and every connected component is given as a set of edge references.

3.5 Output of cographs

Called with the **-c** option, **feyncop** also outputs the cographs, obtained by contracting the corresponding subgraphs in the original graph, for the reduced coproduct. With this option the output is a triple with the original graph, the subgraphs as described above and with the corresponding cograph in the `G`-format. The contracted edges of the cographs are not removed from the original edge list, but replaced by the dummy edge `[-1,-1]` to simplify reference to the edges by index. The output will be of the form

```
D[ ( original graph ), [
[ { { 1. subgraph's 1. connected component's edges },
{ 1. subgraph's 2. connected component's edges },
...
},
 ( cograph to 1. subgraph ) ],
[ { { 2. subgraph's 1. connected component's edges },
    ... },
 ( cograph to 2. subgraph ) ],
...
] ].
```

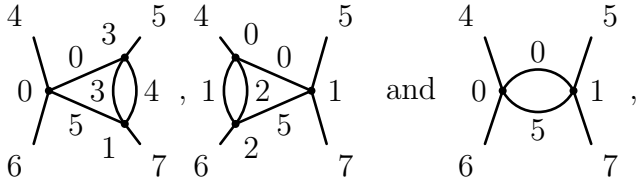
Example With  as input,

```
$ echo "G[[1,0],[2,0],[2,0],[3,1],[3,1],[3,2],
[4,0],[5,1],[6,2],[7,3]]" | ./feyncop -D4 -c
```

the following outcome will be produced:

```
+ D[G[[1,0],[2,0],[2,0],[3,1],[3,1],[3,2],
[4,0],[5,1],[6,2],[7,3]],[
[{{1,2}},
G[[1,0],[-1,-1],[-1,-1],[3,1],[3,1],[3,0],
[4,0],[5,1],[6,0],[7,3]]],
[{{3,4}},
G[[1,0],[2,0],[2,0],[-1,-1],[-1,-1],[1,2],
[4,0],[5,1],[6,2],[7,1]]],
[{{1,2},{3,4}},
G[[1,0],[-1,-1],[-1,-1],[-1,-1],[-1,-1],[1,0],
[4,0],[5,1],[6,0],[7,1]]]]
]
;
```

The cographs, with vertex and edge labeling, corresponding to the subgraphs in the last section



are represented in the G-format as

```
G[[1,0],[-1,-1],[-1,-1],[3,1],[3,1],[3,0],
[4,0],[5,1],[6,0],[7,3]]
,
G[[1,0],[2,0],[2,0],[-1,-1],[-1,-1],[1,2],
[4,0],[5,1],[6,2],[7,1]]
```

and

```
G[[1,0],[-1,-1],[-1,-1],[-1,-1],[-1,-1],[1,0],
[4,0],[5,1],[6,0],[7,1]].
```

In the course of the calculation of the cographs, vertices are removed. Therefore, the vertex labeling of the cographs is not compatible with the one of the original graph. Whenever an edge is contracted and two vertices are merged, the new vertex will carry the smaller label.

3.6 Output of tensor products

Called with the option **-u**, **feyncop** identifies the subgraphs and cographs with unlabeled graphs, groups them in tensor products and sums tensor products of similar

form. Called with this option **feyncop** will handle all graphs as non-leg-fixed graphs. Giving leg-fixed graphs as input will result in less terms with higher factors than expected. The tensor products are given as pairs of a product of superficially divergent connected components of the subgraphs and the cograph. The pairs are preceded by an `T` to indicate the tensor product type of output. The output is a sum of these tensor products.

The output of a single tensor product will be of the form

```
( factor ) * T[ ( product of subgraphs ), ( cograph ) ].
```

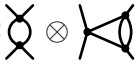
Example Giving  as input,

```
$ echo "G[[1,0],[2,0],[2,0],[3,1],[3,1],[3,2],
[4,0],[5,1],[6,2],[7,3]]" | ./feyncop -D4 -u
```

the output will be:

```
+ 2/1 * T[ G[[1,0],[1,0],[2,0],[3,0],[4,1],[5,1]],
G[[1,0],[1,0],[2,0],[2,1],[3,2],[4,2],[5,0],[6,1]] ]
+ T[ (G[[1,0],[1,0],[2,0],[3,0],[4,1],[5,1]])^2,
G[[1,0],[1,0],[2,0],[3,0],[4,1],[5,1]] ]
;
```

This output corresponds to the reduced coproduct calculation,

Where, 2  , is represented as

```
2/1 * T[ G[[1,0],[1,0],[2,0],[3,0],[4,1],[5,1]],
G[[1,0],[1,0],[2,0],[2,1],[3,2],[4,2],[5,0],[6,1]] ]
```

and , is denoted as

```
T[ (G[[1,0],[1,0],[2,0],[3,0],[4,1],[5,1]])^2,
    G[[1,0],[1,0],[2,0],[3,0],[4,1],[5,1]] ].
```

Note that the labelings of the vertices of the subgraphs and cographs in this mode of **feyncop** carry no resemblance to the vertex labeling of the input graph. The labelings are chosen using **nauty**'s canonical labeling algorithm. Therefore, the graphs are representatives of the corresponding graph isomorphism class.

3.7 Usage in conjunction with feyngen

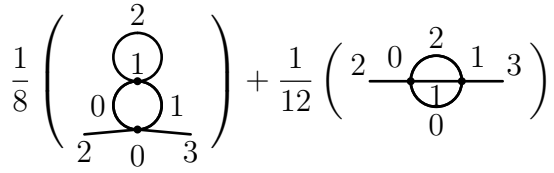
feyncop can be used in conjunction with **feyngen**. The coproduct of every input graph is computed and the sum of the coproducts in a format depending on the options is given as output. To do this the output of **feyngen** must be piped into **feyncop** as input. For instance,

```
$ ./feyngen 2 -j2 -k4 -pu | ./feyncop -D4
```

will yield the sum of the reduced coproducts, given in the D-format, of all non-leg-fixed two loop, φ^4, 1PI graphs with two external legs weighted by the symmetry factor of the original graph,

```
+ 1/8 * D[G[[1,0],[1,0],[1,1],[2,0],[3,0]],
[{{2}}, {{0,1}}]]
+ 1/12 * D[G[[1,0],[1,0],[1,0],[2,0],[3,1]],
[{{0,1}}, {{0,2}}, {{1,2}}]]
;
```

where the relevant graph sum with edge and vertex labels, of which the coproduct is calculated, is



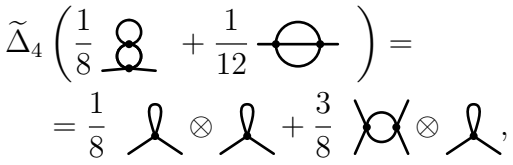
On the other hand,

```
$ ./feyngen 2 -j2 -k4 -pu | ./feyncop -D4 -u
```

will yield the sum of tensor products of unlabeled graphs corresponding to the reduced coproduct applied to the sum of all two loop, φ^4, 1PI graphs with two external legs weighted by their symmetry factor:

```
phi4_j2_h2_nlf_red_cop_unlab :=
+ 1/8 * T[ G[[0,0],[1,0],[2,0]],
   G[[0,0],[1,0],[2,0]] ]
+ 3/8 * T[ G[[1,0],[1,0],[2,0],[3,0],[4,1],[5,1]],
G[[0,0],[1,0],[2,0]] ]
;
```

This corresponds to the calculation



which can be validated using an identity on the Hopf algebra of Feynman graphs.

3.8 Filtering for primitive graphs

feyncop has the ability to filter the input graphs for primitive ones. This behaviour is triggered by the **-p** option. A convenient usage pattern is to pipe the output of **feyngen** into **feyncop** to obtain a set of primitive graphs with the desired properties.

Example If all primitive, QED, vertex diagrams with three loops are desired, the call to **feyngen**,

```
$ ./feyngen 3 --qed -b1 -f2 -p
```

to generate the 100 not necessarily primitive QED diagrams is needed. To filter these diagrams for primitive ones, they can be piped into **feyncop**:

```
$ ./feyngen 3 --qed -b1 -f2 -p | ./feyncop -D4 -p
```

This will result in the output

```
qed_f2_b1_h3_proj_to_prim :=
+ G[[1,4,f],[2,3,f],[3,6,f],[4,5,f],[5,0,f],[6,1,f],
[4,3,A],[5,2,A],[6,0,A],[0,7,f],[8,2,f],[9,1,A]]
+ G[[1,4,f],[2,3,f],[3,6,f],[4,5,f],[5,0,f],[6,1,f],
[4,2,A],[5,3,A],[6,0,A],[0,7,f],[8,2,f],[9,1,A]]
+ G[[1,2,f],[2,6,f],[3,4,f],[4,5,f],[5,1,f],[6,0,f],
[3,2,A],[4,0,A],[6,5,A],[0,7,f],[8,3,f],[9,1,A]]
+ G[[1,6,f],[2,5,f],[3,4,f],[4,1,f],[5,0,f],[6,2,f],
[3,2,A],[5,4,A],[6,0,A],[0,7,f],[8,3,f],[9,1,A]]
+ G[[1,6,f],[2,4,f],[3,2,f],[4,5,f],[5,1,f],[6,0,f],
[4,0,A],[5,3,A],[6,2,A],[0,7,f],[8,3,f],[9,1,A]]
+ G[[1,6,f],[2,5,f],[3,4,f],[4,0,f],[5,1,f],[6,3,f],
[4,2,A],[5,3,A],[6,0,A],[0,7,f],[8,2,f],[9,1,A]]
+ G[[1,4,f],[2,6,f],[3,5,f],[4,0,f],[5,1,f],[6,3,f],
[4,3,A],[5,2,A],[6,0,A],[0,7,f],[8,2,f],[9,1,A]]
;
```

corresponding to the sum of the seven primitive, three loop, vertex diagrams in QED: