IdSolver notes

June 7, 2017

Contents

1 General guidelines

- `lorentz3p.prc` is needed for 3-point functions
  `lorentz4p.prc` is needed for 4-point functions
- Edges in prototype file must be declared following some convention.
- Temporary `form` files are stored as `.number.id.frm`. The corresponding `form` log file in `.number.id.log`.

2 Tools directory

2.1 generate_identities

```
./generate_identities prototype nscalars ndeno d n symbols
```

- `prototype`: name of the prototype
- `nscalars`: number of indices `n1, n2,...nscalars`
- `ndeno`: number of denominators; notice that `nscalars >= ndeno`

Contrary to its name, this program requires the full set of administrative files (symmetries, matchings, identities, integrals, kinematics, etc.). What it does is the following It applies IBPs, which are stated in general in those files, in particular in `PR?identities.prc`, to a concrete set of integrals specified by the command line arguments, *i.e.* integrals with specific indices. The result ends up in `idprototype.dat` and `PR?inc.dat` files are created as well. All these are binary database files that contain IBPs, LIs etc. in the FORM format, which look like this

```
id4 =
    + PR0(1,2,1) * ( - 1/2 )
    + PR0(1,2,2) * ( 1/2 )
    + PR0(2,1,1) * ( 1/2 )
    + PR0(2,1,2) * ( - 1/2 );
```

2.2 solve_integrals

```
./solve_integrals integrals output symbols
```

- `integrals`: file with list of integrals in the format like `PR1(3,0,0)`
- `output`
- `symbols`: usual `ep`

This program will use databases created by `generate_identities` and will generate `fill` statements (like Substitutions file from the `solve_prototypes`). Depending on what is found the input databases, the output will be more or less simplified.

2.3 read_identities

```
./read_identities prototype begin end firstid symbols
```

Read identities stored in FORM log files. The files need to be called as: `0.log`, `1.log`... `n.log`. Then, `begin` and `end` are the starting and ending numbers of the set of files that should be read. `firstid` is the number of the identity we want to start from.

2.4 print_identities

```
./print_identities prototype output
```

Print identities stored in `idprototype.dat` database and print them in the text formatn into the `output` file.

3 test directory

This directory contains an example, which can be run as

```
$./solve_prototypes proto 1 1 ep
```

`ep` above means ϵ. The two numbers mean the sum of denominator and numerator powers.

Options:

`-p` prints postscript pictures for all the prototypes

`-v` verbose

This will produce the file `Substitutions` that contains all the simplified `fill` statements and the file `Masters` with all the masters.

Exactly the same effect can be achieved by the following sequence

```
$ ./generate_identities PR0 3 3 1 1 ep
$ ./generate_identities PR1 3 1 1 1 ep
$ ./generate_identities PR2 3 2 1 1 ep
$ ./generate_identities PR3 3 2 1 1 ep

$ ./solve_integrals  inputint out ep

$ ./determine_masters PR0 3 3 1 1 1 1 ep
$ ./determine_masters PR1 3 1 1 1 1 1 ep
$ ./determine_masters PR2 3 2 1 1 1 1 ep
```

The masters that come out in stdout will be exactly those from `Masters` and the file `out` will be identical with `Substitutions`.

3.1 determine_masters

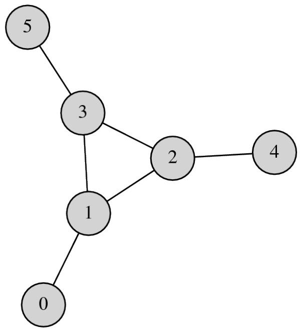
```
usage: ./determine_masters [-n] [-f prototypes | prototype nscalars
        ndeno d n] dm nm symbols
```

This program does not need databases of integrals. It only needs FORM procedures like `PR0identities.prc`, `PR0symmetries.prc` and corresponding include files. Witn `nscalars` = `ndeno` the value of `n`, the maximal sum of numerators is irrelevant.

3.2 Input files

proto

As shown in the example below, diagrams are specified by setting the number of nodes `n`, then `e` *ni nj* describes the edges between nodes *ni* and *nj*. The lines `m -p1 4` specifies the momentum $p1$ on edge 4 and outgoing.




```
---------- PR0 ----------

0: 0 -> 1, x, p3
1: 1 -> 2, 1, p1+k
2: 1 -> 3, 1, p2-k
3: 2 -> 3, 0, k
4: 2 -> 4, 1, p1
5: 3 -> 5, 1, p2
```

Figure 1: *Left:* Graphical representation of the prototype `PR0`. *Right:* The actual content of `PR0` obtained by printing the `PrototypeMap` object. The left column is the node name, then the edges connecting the nodes, then the masses of the lines and then the 4-momenta of the lines. Printout on the r.h.s. comes from temporary `form` files created and used by `IdSolver`.

Listing 1: proto

```
n 6      # number of nodes
e 0 1 x  # edge from node 0 to node 1 with mass x
e 1 2 1
e 1 3 1
e 2 3 0
e 2 4 1
e 3 5 1
m +p3 0  # ingoing external momentum p3 on edge 0
m -p1 4  # outgoing external momentum p1 on edge 4
m -p2 5
```

The above listing corresponds to the configuration shown in Fig. 1. The nodes are marked with circles and the edges correspond to lines connecting the circles. The incoming/outgoing momenta are themselves the edges. The RHS of Fig. 1 is what is in the actual `Prototype` object graphically represented on the LHS.

The diagram from Fig. 1 is a triangle. Three of the edges correspond to propagators and three other to the external particles.

userkinematics

That file contains kinematic identities. Comments with # are not recognized.

Listing 2: userkinematics

```
id p1.p1 = 0;
id p2.p2 = 0;
id p1.p2 = 1/2;
```

ibp.prc

- `SS` functions are the scalar products.

3.3 Output files

Masters

Here is where the set of master integrals ends up. The prototypes are called with the names going like `PR?`.

Listing 3: Masters

```
PR1(1,0,0)
PR2(1,1,0)
PR0(1,1,1)

3 master integral(s) identified
```

Substitutions

Integrals expressed in terms of masters like for example

Listing 4: Substitutions

```
fill PR0(1,1,2) =

 + PR0(1,1,1) * (ep)

 + PR1(1,0,0) * (2*ep-2)

 + PR2(1,1,0) * (-2*ep+1)
;
```

4 The code

PrototypeMap.cpp: PrototypeMap::insert(NamedPrototype* p)

This is one of the main functions, where a lot is being done. In particular, all the prototypes are derived here.

DiaGen/Prototype.cpp

The original `Prototype` class is defined in `DiaGen`.

IdentityGenerator::StoreIdentities

If this option is set identities are saved in databases. This option is enabled by (hard coded) in `generate_identities` and can be chosen at run time in `solve_prototypes` with the command line argument `-c`.

decls

Prototypes like `PR0(n1,n2,n3)` are effectively sparse tables where the table indices run over indices of the topology.

SolveNumerators

It is a switch in `IdSolver.hpp`. When set to 0, results will contain irretudible numerators, when set to 1, those numerators are solved into masters with dots (double denominators).

Errors

- When the program gets stuck at `Creating administrative and identity files...` most probably, the path to `fermat` is not correct
- Some errors end up in the files `PR?identities.prc` and in the variable `current_identity` of `IdentityGenerator.cpp`.