On the choice of variables in IBP reductions

J. Vermaseren
Nikhef

- Introduction
- First attempt
- A study of the bebe topology
- Comparing with Forcer
- Conclusions (10-sep-2019)

Introduction

One of the methods used in a large category of precision calculations is the reduction of integrals to a small set of 'master integrals' by means of an IBP reduction scheme.
There are two different ways to set up such a scheme:

1. A parametric scheme, as advocated originally by Chetyrkin and Tkachov.
2. A 'build up' scheme as introduced by Laporta.

Each of these ways has advantages and disadvantages: A parametric scheme gives usually faster programs and can go to much more complicated cases than the Laporta schemes. For the Laporta schemes generic programs exist. Hence they gain big on the time to program them.

In all but the simplest systems one does not only have variables that are propagators, but also variables that occur as numerators and the elimination of such numerators is usually far more complicated than the elimination of excess denominators.
The choice of the numerator variables is usually not unique and it has been noticed independently by the author and by Rutger Boels (and maybe others) that it can make a big difference in the complexity of the reduction.
Such complexity manifests itself in the size of the expressions and the polynomials in the rational coefficients of the terms during the intermediate stages of the reduction. In particular these rational coefficients are a severe bottleneck in the whole process, because the computation of GCD's of two multivariate rational polynomials can be extremely time consuming.

This brings us to the central question of this talk:

"Is there a way to select a set of variables in which the reduction scheme is either optimal or close to optimal?"

Here I define optimal as using fewest computer resources.

It should be noticed that 'non-trivial reductions' which have a number of kinematical variables in their answers are to some extend special cases of the complete reductions that are needed for massless propagator diagrams. In the intermediate stages the rational polynomials are of a similar type. In the case of massless propagators one has to keep eliminating variables until the only parameter that remains is the dimension (or ϵ). The 'non-trivial reductions' stop at an earlier moment.

In this talk I will concentrate on the parametric method as part of a project to study the automatic generation of a 5-loop massless propagator reduction program. Such a program is still in the far future. Hence I will take examples from 4-loop reductions, even though a program for those exists already (Forcer). This gives the benefit that results can be compared. It may also improve the speed of Forcer, which then can lead to more Mellin moments for splitting functions in DIS.

It should come as no surprise that all programs used for this talk are written in the Form language.

Finally, the definition of a reduction scheme:

First one takes the number of denominators.

Next one defines complexity as the number of excess numerators and denominators.

Next one defines an order of the variables.

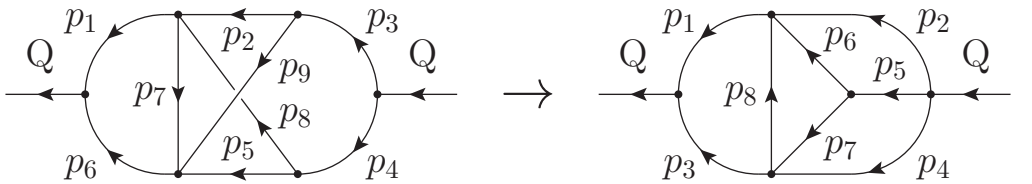
In a reduction scheme each step either removes a denominator, lowers the complexity, or moves integrals to integrals with the same complexity but with raised variables further in the order of variables.

This gives a sequence of statements that will eventually either kill a denominator, placing the integral in a different category, or leave only integrals that cannot be reduced (master integrals).

A kickback is when a lowering of a variable gives terms with a lower complexity, but closer to the beginning of the elimination tree, because one of the earlier variables, that had already been brought down to its canonical value gets raised again. It still means a finite program (the complexity was lowered) but many more elimination steps then when such kickbacks are not present.

First Attempt

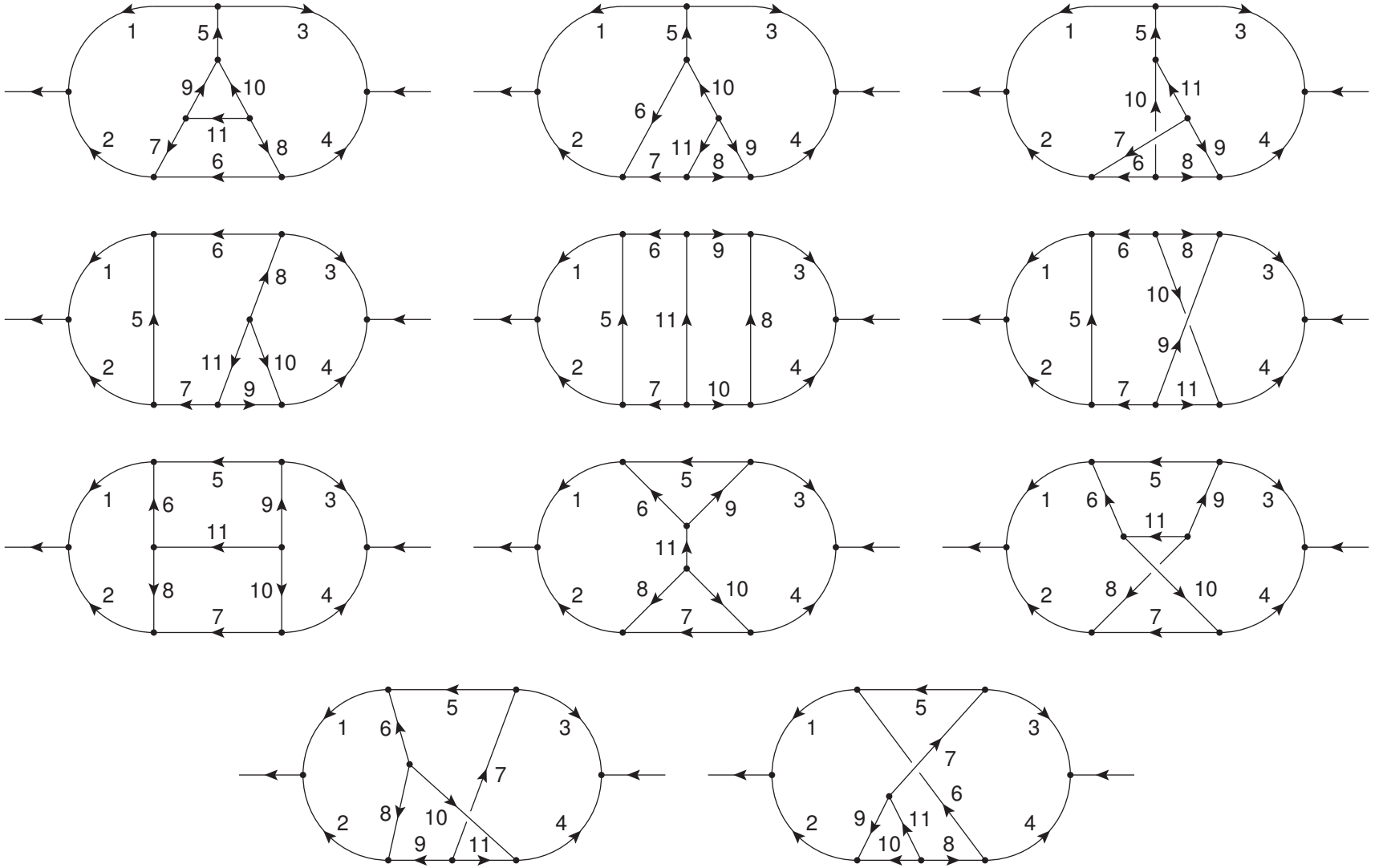
A simple study of running statistics of the Forcer program reveals that much time is spent converting notations when the program switches from one topology to a simpler topology. Such rewriting is necessary because a topology that contains a number of 4-point vertices can come from different 'higher level' topologies with fewer 4-point vertices. In addition it can come from different missing lines in the same higher level topology. Example:



The topology on the right (called bebe) can come from the left topology (called nono) either by eliminating its p_3-line or by eliminating its p_4-line.

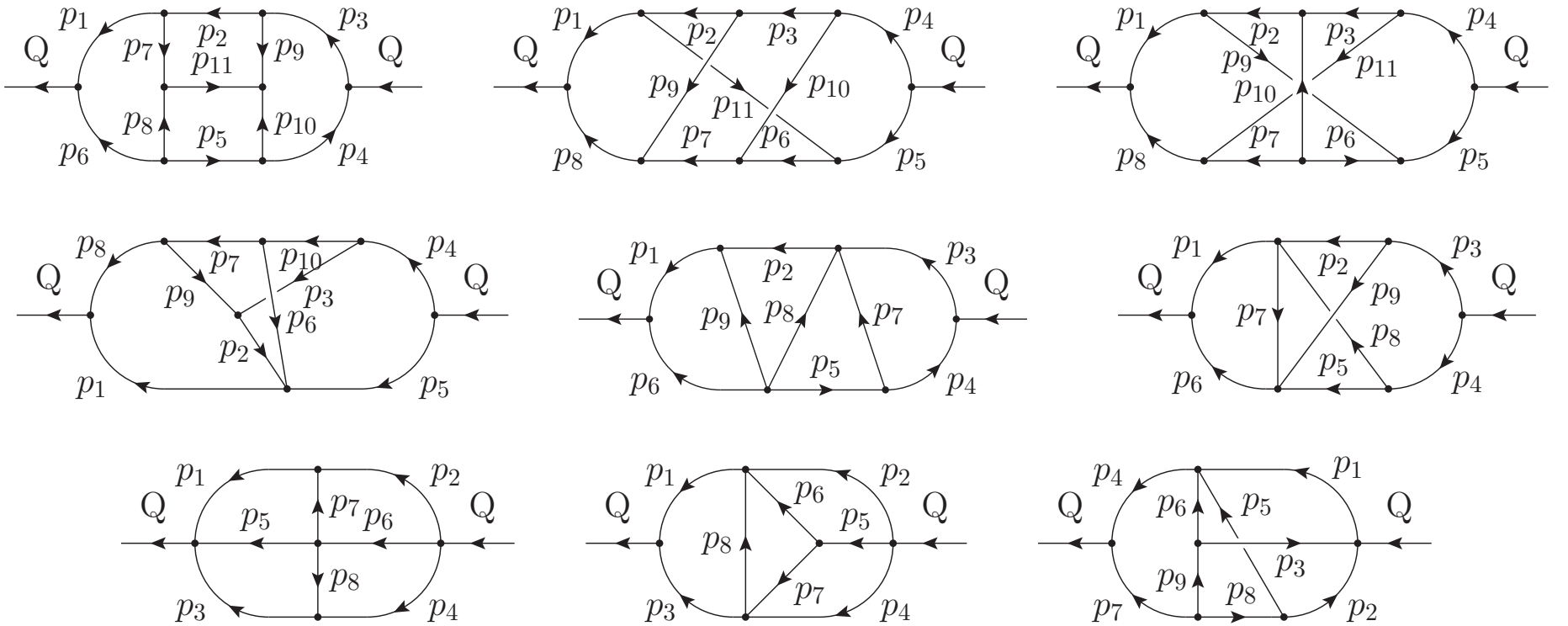
The fact that Forcer has only a single bebe routine forces such rewrites. This single bebe routine is a direct consequence of the hand guided derivation which was difficult enough already.
If one can make good quality automatically generated reduction schemes this limitation might be removed. One could have for instance 34 versions of the bebe routine, each with its own notation.
One might even need more versions: if a certain variable occurs to a high power, it is more economical when it is the last variable to be reduced, because if it is at the beginning of the reduction tree it will generate an enormous avalanche of terms. This would mean that it is most economical to have schemes that are tailor-made for particular diagrams.

Let us look at the following setup:
Take all non-trivial top-level topologies. Non-trivial is interpreted as that there is no immediate subintegral that can be worked out, like an internal 2-point function. There are 11 of such topologies:



Notation from the Kaneko diagram generator in Form.

A family is defined as a single top-level topology and all subtopologies that can be derived of it by removing one or more lines. Hence there are 11 families.
Most of the subtopologies are of course trivially zero. Many are trivial in the sense that one of the loop integrals can be done immediately.
There are also integrals that can be treated with a (generalized version) of the triangle rule to eliminate a line and create simpler subtopologies.
Finally there are integrals remaining that need a complete reduction scheme. These are the 9 4-loop topologies mentioned in the Forcer paper for which reductions had to be crafted by hand guided computer programs. In the case of the 4-loop families there are in total 88 such (sub)topologies (identical subtopologies with different notations are considered different here).



The 9 4-loop topologies that need a complete reduction scheme. The notation is the one used in the Forcer program.

The different (sub)topologies for family 10 (also called no1)

Topology	nr	1	2	3	4	5	6	7	8	9	10	11
no1	58	x	x	x	x	x	x	x	x	x	x	x
no6	59	x	x	x	x	-	x	x	x	x	x	x
lala	60	x	x	x	x	x	-	-	x	x	x	x
lala	61	x	x	x	x	x	-	x	x	x	x	-
lala	62	x	x	x	x	x	x	-	-	x	x	x
nono	63	x	x	x	x	x	-	x	x	-	x	x
nono	64	x	x	x	x	x	x	-	x	x	-	x
bubu	65	x	-	x	x	-	x	x	x	x	x	x
bubu	66	x	-	x	x	x	x	x	x	x	-	x
bubu	67	x	x	x	-	-	x	x	x	x	x	x
bubu	68	x	x	x	-	x	x	x	x	-	x	x
bebe	69	-	x	x	x	x	x	-	x	x	-	x
bebe	70	x	-	x	x	-	x	x	x	x	-	x
bebe	71	x	-	x	x	-	x	x	x	x	x	-
bebe	72	x	-	x	x	x	x	-	x	x	-	x
bebe	73	x	x	-	x	x	-	x	x	-	x	x
bebe	74	x	x	x	-	-	x	x	-	x	x	x
bebe	75	x	x	x	-	-	x	x	x	-	x	x
bebe	76	x	x	x	-	x	-	x	x	-	x	x
cross	77	-	x	x	-	x	x	x	x	-	x	x
cross	78	x	-	-	x	x	x	x	x	x	-	x
cross	79	x	-	x	-	-	x	x	x	x	x	x

and a simpler one (family 2)

Topology	nr	1	2	3	4	5	6	7	8	9	10	11
bebe	3	x	x	x	-	-	x	-	x	x	x	x
cross	4	x	-	x	-	x	x	x	x	x	-	x

The challenge is now to find a single notation for each family that allows the creation of efficient reduction schemes for all its (sub)topologies.

This involves:

Determine all possible notations by finding all possible independent choices of either dotproducts or invariants in the numerators for the top level topologies, because the remaining variables for the subtopologies are determined by the missing lines. This runs us into a problem already: the missing lines are defined by squares of composite momenta, while working with dotproducts makes the 1-loop subintegrals and the application of the triangle rule simpler. Probably the best would be a hybrid scheme, but that has not been tried yet.

For each notation, try to find a decent solution for all nontrivial (sub)topologies. This means that a program is needed that can try to solve the set of IBPs and make a parametric reduction scheme for it. This turns out to be rather problematic, because when there are 14 variables to be eliminated, in principle there are 14! potential orderings. In addition there may be more than one equation that can be used for a given variable. Fortunately at each step the choice is usually limited and barring other inconveniences one can get through all of these possibilities in a time that is of the order of days. But this still means that if each notation needs a few days CPU time and there are many thousands of notations to study, more computer time is needed than I have access to.

The next biggest problem here is that most attempts at finding a reduction scheme crash, because at a given moment in a given order of the variables with an unfavorable selection of the equation to be used, the rational coefficients become too complicated for FORM (limit set at a few Mbytes). Alternatively the computation of the GCD of two multivariate polynomials takes too long and it is better to interrupt the program to look at the next notation. Hence each notation is run as a separate program with a time limit. Of course, schemes that run into these problems would not be very compact anyway and should not be used.

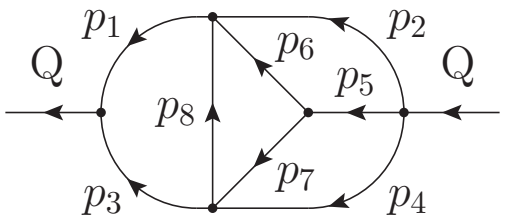
Of course one can try to minimize such problems by giving the program criteria that allow it to skip attempts that would have a big chance of crashing, or attempts that would lead to schemes that would be rejected as too complicated anyway.

In the end each notation leaves a list of its best results at the moment the program finished.

For the most difficult families (starting with a nonplanar diagram) the program could not find a single notation that would solve all its (sub)topologies completely. The cause is that most likely the program for solving a system was still rather primitive. The time limitation plays a big role as well. It should be made much smarter in its selection of things to try and things to reject. At this point there is already much progress, but the time was lacking to make complete runs with it.
I did take some cases in which almost all (sub)topologies had a solution and tried the remaining one(s) by hand guided computer programs and did not succeed to come up with anything reasonable. Again, my tools/methods may not have been powerful enough yet.

Conclusion for now: improve the search program before starting a new search.

A study of the bebe topology



The bebe topology

It was the bebe topology that gave most troubles in the previous section. It is the most-occurring subtopology (34 out of a total of 88). Hence it seems a good idea to improve the search program by trying it out on this subtopology. The game is now:

- Determine all (or most) potential notations for the bebe topology.
- For each of these notations we try to determine a best scheme.
- We may have to accept a limited amount of rewriting in a complete program if we get something that is much better than the routine that is currently used in Forcer.

A notation here is defined as:

1. Choose a set of 4 independent momenta plus the external momentum Q.
2. Choose all possible combinations of 6 independent dotproducts made from the above 5 momenta (the bebe topology has three missing lines and hence needs 6 numerator variables).

An other approach could be to select 3 invariants connected to the 3 4-point vertices (the s, t or u variable) and 3 other invariants to later build things back to the higher topologies. For now the dotproduct method is used.

With an improved ‘search program’ all possibilities were visited. Each solution was judged on how lengthy its resulting code would be. This gave eventually some notations that were markedly shorter than the one used in Forcer. The problems here were that the search program was slower than before and again eventually most programs either crashed, terminated without full solution or never finished. But then for each the best solution at the moment of termination was taken.
To give an example: ‘notation 312’ came out as best, but ‘notation 213’ was not far behind. Based on extra criteria, 213 is the one that was used eventually.
The next step was to see (because of the above limitations) whether hand guiding could improve such solutions. To understand this, let us look at how the program works for a given notation:

First it constructs the IBP's. These are called the A-equations. There are 20 of those.

Next it uses a Gaussian elimination to minimize the number of terms with the highest complexity (= number of excess numerators and/or denominators). These are complexity 2. This gives the B-equations. Still 20 equations but with a simpler structure in the most complex integrals.

Next in all B-equations the complexity is raised by one in all 14 possible ways. This results in a total of 300 equations. These are the G-equations.

In a Gaussian elimination all integrals of complexity 2 or more are eliminated (equations used for elimination are dropped). The remaining equations all have maximum complexity one.

Next pairs of equations are combined to create new equations with fewer terms. The resulting equations are called the H-equations and those are the ones that the program tries to 'solve'.

For the bebe topology the above is not enough. It turns out that there are two variables that cannot be solved this way. But there are two B-equations that can be used to eliminate these variables by raising the complexity (in other variables). When these equations are used on the G-equations before the Gaussian elimination the resulting H-equations can have these two parameters at their canonical value of one already. The drawback is that there are fewer equations remaining.

For solving the system the program makes at each step a table of possible next steps. This involves determining an equation that can be used for the elimination of a variable and putting a 'badness' to it which is determined by the number of terms in the equation, the number of terms of complexity one (ie terms that do not directly lower the complexity but move values to thus far untreated variables) and the size of the polynomials in the rational coefficient of the term that is used for the elimination. The possibilities are ordered by badness.

Here is a sample output for the startup of the program.

```
~~~topo = bebe, NNN = 011111101011000
For topology bebe:  n4 with B1  n3 with B14
~~~Going to do gauss at j = 4
~~~Going to do gauss at j = 3
~~~Going to do gauss at j = 2
 replace G23
 replace G116
 replace G133
 replace G35
 replace G133
 replace G133
 replace G263
 replace G268
 replace G183
 replace G226
 drop G273
 replace G292
 replace G263
 replace G297
 replace G292
 replace G297
 replace G183
 replace G297
 replace G263
 replace G263
```

```
 replace G297
   expressions=17, terms=891, one=196, time=16.82 sec;
 Topo = bebe. Signature = 01111101011000
$old = 0
There are 16 possibilities
 1: m = 3, expr = 13, var = n8, sp1 = 0, w = 390
 2: m = 3, expr = 12, var = n5, sp1 = 0, w = 419
 3: m = 3, expr = 3, var = n9, sp1 = 0, w = 424
 4: m = 3, expr = 16, var = n2, sp1 = 0, w = 434
 5: m = 3, expr = 17, var = n5, sp1 = 0, w = 480
 6: m = 3, expr = 5, var = n1, sp1 = 0, w = 982
 7: m = 4, expr = 13, var = n10, sp1 = 6, w = 390
 8: m = 4, expr = 13, var = n11, sp1 = 6, w = 390
 9: m = 4, expr = 3, var = n1, sp1 = 8, w = 424
 10: m = 4, expr = 3, var = n7, sp1 = 8, w = 424
 11: m = 4, expr = 2, var = n13, sp1 = 5, w = 718
 12: m = 4, expr = 9, var = n9, sp1 = 10, w = 766
 13: m = 4, expr = 4, var = n9, sp1 = 10, w = 793
 14: m = 4, expr = 10, var = n12, sp1 = 13, w = 958
 15: m = 4, expr = 10, var = n7, sp1 = 13, w = 958
 16: m = 4, expr = 5, var = n9, sp1 = 6, w = 982
 $nmax1 = 27,0,0,        $nnums1 = 145
 $nmax2 = 17,0,0,        $nnums2 = 84
 $nmax3 = 0,0,0,         $nnums3 = 0
 $nmax4 = 0,0,0,         $nnums4 = 0
```

```
$nmax5 = 21,0,0,        $nnums5 = 96
$nmax6 = 27,0,0,        $nnums6 = 153
$nmax7 = 33,0,0,        $nnums7 = 136
$nmax8 = 11,0,0,        $nnums8 = 89
$nmax9 = 21,0,0,        $nnums9 = 177
$nmax10 = 23,0,0,       $nnums10 = 152
$nmax11 = 50,0,0,       $nnums11 = 210
$nmax12 = 21,0,0,       $nnums12 = 150
$nmax13 = 21,0,0,       $nnums13 = 221
$nmax14 = 54,0,0,       $nnums14 = 159
H1: -1: 11  0: 7  rem: 18  tot: 76
H2: -1: 9  0: 8  rem: 17  tot: 61
H3: -1: 5  0: 2  rem: 7  tot: 28
H4: -1: 15  0: 11  rem: 26  tot: 49
H5: -1: 13  0: 3  rem: 16  tot: 85
H6: -1: 13  0: 8  rem: 21  tot: 42
H7: -1: 17  0: 7  rem: 24  tot: 101
H8: -1: 8  0: 4  rem: 12  tot: 61
H9: -1: 11  0: 9  rem: 20  tot: 58
H10: -1: 14  0: 5  rem: 19  tot: 76
H11: -1: 22  0: 12  rem: 34  tot: 104
H12: -1: 7  0: 4  rem: 11  tot: 21
H13: -1: 7  0: 0  rem: 7  tot: 20
H14: -1: 14  0: 2  rem: 16  tot: 30
H15: -1: 14  0: 4  rem: 18  tot: 34
```

```
H16: -1: 8  0: 3  rem: 11  tot: 20
H17: -1: 8  0: 3  rem: 11  tot: 25
```

It indicates that the variables n3 and n4 are used in the complexity raising operation. The replace messages are given during the minimization of the number of terms. The signature tells which variables are numerators (0) and which are denominators (1). It tells that for the first step there are 16 possibilities. A value of 3 for m indicates a direct reduction and the value 4 indicates that the reduction should be followed in the next step by the reduction of the variable indicated by the 'spectator'.
nmax indicates how often an integral with a raised value of that variable occurs. This is an indicator of how big a mess will be created when eliminating this variable. It has acually three numbers: the numberof $n+1$ occurrences, the number of $n+2$ occurrences and the number of $n+3$ occurrences. The last two need multiple applications of the lowering equation and hence create an even bigger mess.
nnums indicates how often the given variable occurs in the combined equations. This tells us about how much simplification occurs when this variable is replaced by its canonical value (1 or 0).
Finally the program tells how many terms each equation has and a breakdown of their types.

The program will try all possibilities, starting with the smallest badness, and continue to the next step which will be similar. Etc.
Of course it can happen that at any given moment there are no possibilities remaining after which the program goes back to the next possibility for the previous level. Because the program keeps track of what it tried and still has to try, in the end we can see what were the best elimination sequences. It also keeps track about how 'expensive' each (partial) solution is and once a possibility is too expensive it is removed because otherwise the administration becomes too lengthy. For simple toplogies like lala this works really well. This topology does not lead to crashes and hits on full solutions almost 50% of the time. Many of these are significantly shorter than the Forcer solution. Unfortunately lala is not a very expensive topology anyway, which is why most practicing went into bebe which in actual physics programs is often the most expensive (sub)topology.

Because the program starts with the best prospect, and then tries the next etc. there is of course a good chance that the best solution is somewhere near the beginning of what is attempted. In addition many partial attempts that are much worse than the best solution already found, can be abandoned. The bebe program ends in a crash nearly every time and most of the time still before the early stages have reached an advanced possibility. But anyway, the hand guiding gets a great boost by starting from the best solution and then slowly trying improvements at the earlier possibilities.

Even though nearly all automated programs ended in crashes or timeouts, it still indicated a number of promising solutions, which were then further investigated by guiding the program by hand. The result was a few solutions that were significantly shorter than the Forcer solution.

In the next section the best of these is compared with the Forcer solution.

Comparing with Forcer

To replace the existing bebe routine inside Forcer by a new one that uses a different notation is not entirely trivial. On all potential inputs the 'notation transformation' parts have to be changed and the same holds for the potential outputs. And because of technical reasons the original Python program that generated Forcer cannot be used anymore. A new run would change all notations and hence the existing databases would become useless. Hence the change of these transformations had to be implemented by hand.

First test. One bebe integral in which all 14 parameters are raised by 1.

	whole program	bebe step	no shortcuts	no shortcuts bebe step
old	44 sec	34 sec	1937 sec	1562 sec
new	30 sec	18 sec	724 sec	377 sec

What are these shortcuts? One can use the B-equations to lower complexity. This can be done for a limited number of combinations of one excess numerator plus one excess denominator. As it turns out, such reductions work better for the 'old' program than for the 'new' program. It is not yet clear though how to take this into account when looking for the ideal scheme.

The second test is with a single one of the parameters raised by 10.

nr	new	new bebe	old	old bebe
1	280 sec	114 sec	304 sec	241 sec
2	4 sec	-	18 sec	-
3	85 sec	19 sec	304 sec	241 sec
4	4 sec	-	18 sec	12 sec
5	6 sec	-	16 sec	10 sec
6	7 sec	-	9 sec	-
7	7 sec	-	9 sec	-
8	10 sec	4 sec	5 sec	-
9	118 sec	34 sec	734 sec	633 sec
10	40 sec	2 sec	734 sec	633 sec
11	100 sec	13 sec	770 sec	666 sec
12	371 sec	131 sec	83 sec	31 sec
13	380 sec	106 sec	908 sec	675 sec
14	30 sec	1 sec	788 sec	672 sec

In this case the shortcuts can only become active after some substitutions have taken place already, because in the beginning there are either no numerators or no denominators. This way the better structure of the new scheme is superior when there are many numerators.

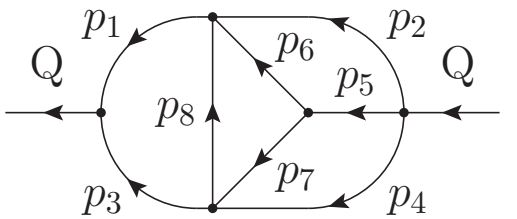
The final test is a test in which the power of each of the numerators is raised by two. This means that there are 12 excess numerators.

new	new bebe	old	old bebe
305 sec	30 sec	2052 sec	1541 sec

The tests with only excess numerators come closest to running Mellin moments of operator vertices in which there are many numerators and only a single excess denominator (if it does not get cancelled by the dotproducts coming from the Feynman rules).
High Mellin moments in DIS via $\gamma^* + q \to \gamma^* + q$ have a large excess of both numerators and denominators.

What do we learn from the above?

Let us have a good look at the topology again:



In the old program the numerators are: $Q.p_2, Q.p_4, Q.p_6, p_1.p_2, p_2.p_6, p_1.p_4$
In the new program (notation 213) they are $Q.p_2, Q.p_4, -Q.p_8, -p_1.p_2, -p_1.p_4, p_1.p_8$
The choice of the loop variables is of course also different, but should not be a big factor at this point. All tests with solving systems with the same numerators but different loop variables show very similar equations and solutions (although not exactly the same). At first it seems good to have the momentum Q to occur as often as possible. This is however not confirmed by early studies of the nono-topology.

Somehow the selection of these new numerators allowed a better solution. In another very good (but somewhat less favorable) solution (notation 312) the program indicated the numerators $Q.p_4, Q.p_8, -p_3.p_4, p_4.p_5, -p_3.p_5, -p_3.p_8$

Considering that there are many hundreds of possible solutions, it is not yet possible to derive rules for good choices unless somebody has a deeper insight. This is a great pity, because it turns out that there are more (sub)topologies in need of improvement for which the searches would take far more effort.

There is however the coincidence that all good solutions found have a very low number of complexity-one terms per H-equation. The opposite does not hold though. It might however speed up the search for good solutions. In particular, notation 213 comes out on top in this category. The number 2 and 3 in this category (notations 3 and 73) however does not get very far. Notation 312 is number 20 out of 982.

```
+ f(1153,196,-17,891,213)
+ f(1162,209,-18,923,3)
+ f(1177,200,-17,1002,73)
+ f(1184,213,-18,1589,937)
+ f(1200,216,-18,1728,960)
+ f(1236,210,-17,1259,255)
+ f(1239,223,-18,1553,568)
+ f(1242,211,-17,1748,946)
+ f(1248,212,-17,1003,635)
+ f(1259,214,-17,1389,367)
+ f(1262,227,-18,1210,356)
+ f(1262,227,-18,1685,957)
+ f(1275,204,-16,1210,656)
+ f(1279,243,-19,1321,643)
+ f(1282,205,-16,1134,739)
+ f(1282,205,-16,1300,46)
+ f(1289,232,-18,1142,601)
+ f(1295,220,-17,993,595)
+ f(1300,208,-16,1313,313)
+ f(1300,221,-17,1038,312)
+ f(1306,222,-17,1045,250)
     + 961 more
```

In addition there are the potential shortcuts that have not been taken into consideration and there is yet another factor which I call ‘kickbacks’ that plays a role and has not (yet) been incorporated in the search program. They are the reason that notation 213 was preferred, because it has no kickbacks at all while notation 312 has a few minor ones and the Forcer solution has a few very bad ones.

Conclusions (10-sep-2019)

The study of ideal notations is still in its infancy.

Some results exist, in particular in the bebe and the lala notation.

Only some weak correllations have been seen yet by lack of a sufficient number of completed examples.

We need a better search program that is more intelligent/powerful.

Better notations can make a big difference because the rational polynomials in the intermediate stages are simpler.