Some steps towards improving IBP calculations

and related topics

J.A.M. Vermaseren

- Introduction
- Few legs and many loops
- A little bit about Rstar
- Mathematical aspects
- Computer algebra
- Outlook

1 Introduction

We all know the principle of the IBP relations. They are based on the equation

$$\int d^D p \frac{\partial}{\partial p^\mu} I^\mu \; = \; 0$$

and working out the derivative. This sounds simple, but when applied to Feynman diagrams it can lead to great complications. This working out was first done in a nontrivial way by Chetyrkin and Tkachov , but it took about 10 years before their algorithms could actually be used in an automated way to compute the $\mathcal{O}(\alpha_S^3)$ contribution to the reaction $e^+e^- \to hadrons$. This is a principle we are still struggling with: there is much time involved in going from an idea to obtaining physics results. In particular more and more mathematics and computer science is needed.

The general problem is of course how to compute reactions by whatever means. The 'classical' way is to write down all Feynman diagrams and try to work them out. Before the days of LHC data relatively few people were involved at the technological edge of this approach, but the need of accurate theoretical results for very large numbers of reactions has changed this and nowadays this is a very active field with very many very smart people making steady progress.

The development knows two main branches:

- many legs and few loops
- few legs and many loops.

There are of course more subdivisions. Inclusion of mass parameters creates horrendous complications, just to name one.

Personally I work in the 'few legs and many loops' branch and in addition I try to keep the particles massless. There are some similarities here with the 'many legs and few loops' approach, because in principle the equations have the same structure, but in for instance massless propagator-type diagrams the equations can/should be reduced until only the dimension parameter remains in the coefficients, which I call a complete reduction, while otherwise the reduction steps finish with still other parameters remaining, resulting in very complicated final answers. In many cases one is already happy with numerical results.

Deriving and applying the IBP relations cannot be done without the use of sophisticated computer programs, either of the generic computer algebra type or more dedicated programs. One computer algebra program that has been developed with this kind of work in mind is Form . As mentioned later in this talk, it is still being extended.

There are different approaches to solving the IBP relations.

The oldest method is to work the integrals down to simpler and simpler integrals until the IBP relations cannot go any further. This requires a rather lengthy development time for creating good programs. Currently the attempt is to automate this approach.

In Laporta-style methods one starts with the easiest integrals and uses the IBP relations to create more and more complicated integrals until one has all the integrals that are needed for a given problem. This generates usually many more integrals than needed and hence needs much storage. Its advantage is that it can (and has been) automated much easier.

In this talk I will concentrate on the first method.

Of course, to get a full answer for a Feynman diagram involves more than 'just' solving a set of IBP equations. There are also the master integrals. These have become a specialization by themselves, attracting also attention from the more mathematically minded scientists as we will see in this talk.

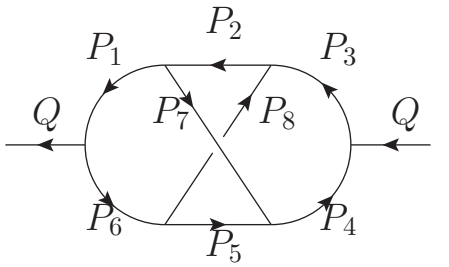
In this talk I will highlight what I see as several paths that are followed in the 'few legs and many loops' branch to make it more powerful. Next I will discuss some mathematical aspects in the field of multiple zeta values (MZV's) and finally some new features in Form to support this.

There will not be very many formulas in this talk. In this subject it is either not many or too many.

2 Few legs and many loops

The program with which this all started was Mincer , developed for Schoonschip in the 1980's in Russia, and later reprogrammed into Form at Nikhef. It managed to reduce 3-loop massless propagator-type diagrams into two master integrals plus convolutions of well known one-loop integrals. With ever increasing computer power it could be used for many more useful physics calculations.

The most complicated topology in Mincer was what was called the NO topology (for non planar):



The reduction scheme can be worked out by hand, but you would not want it to be much more complicated than it is. The diagram in which all denominators have one power is a master diagram. Originally it was worked out to its finite term as

$$NO \;=\; 20\zeta_5 + \mathcal{O}(\epsilon)$$

and a few improvements later it was worked out all the way to the ϵ^7 term (Roman Lee and the Smirnovs).

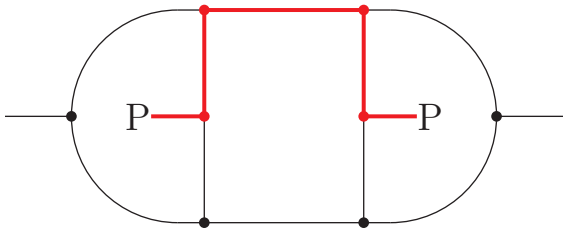
This topology shows the beginning of a very annoying effect. Whereas there are 8 propagators/edges, there are 9 variables (not considering Q^2 which is used for scaling). One of these can only occur in the numerator. There are various possibilities to choose this variable, but all run into the problem that much of the reduction scheme deals with reducing its power to zero. Once this has been done, the remaining part is much simpler.

One may wonder why we need reductions of higher powers of the denominators. Normally diagrams give only one power, and the diagram with all propagators at one is a master integral. Well, there are at least two cases for which higher powers occur:

First: When we want to make gauge checks gluon propagators look like

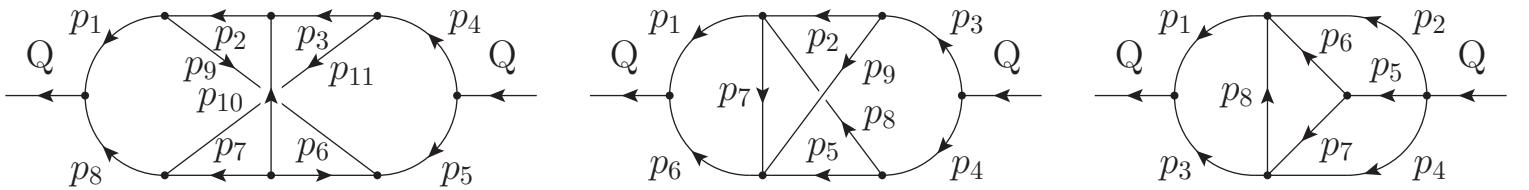
$$-i(g_{\mu\nu} - \xi q_\mu q_\nu/q^2)/q^2$$

and hence we can obtain two powers of $1/q^2$.

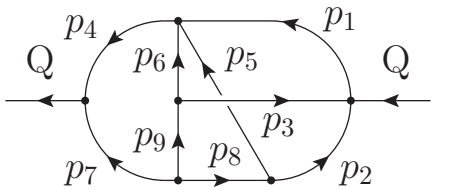


Second: When we study Mellin moments, we have diagrams with 4 legs, two of which have momentum P. This P flows through the diagram and we expand in P. This means that for the N-th moment we have to take N derivatives in P, after which we set P to zero. This can give up to $N + 2$ extra powers of denominators in the remaining propagator-type diagram. The $+2$ comes from what remains after putting P to zero.

When we go to four loops things become much more complicated. Now there are 14 variables and at most 11 propagators. This gives very problematic reduction schemes. Some of the worst ones are:



This was all implemented in the Forcer program. In the Forcer program actually the worst topology was

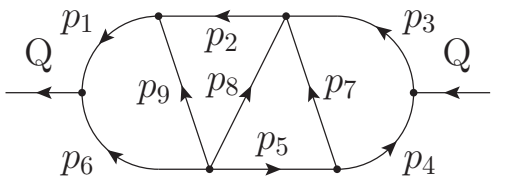


More recent research has shown that the scheme for this topology can be vastly improved by making a different selection for its 5 numerator variables. This brings us to the crucial question, relevant for all IBP solving systems: what is the optimal set of variables and how do we find it?

I have been involved in trying to find such variables in a systematic way. This is by no means easy for several reasons:

- There are many ways to select an independent set of variables. At the four-loop level, there can already be hundreds and sometimes even thousands.
- Once you have a set, there are many orders in which you can eliminate variables.
- Many orders of elimination become very slow and obtain rational polynomial coefficients that can crash the program.
- Some nasties that I will mention below.

Considering the above restrictions a nearly exhaustive search program has been executed only for one topology until now:



It did give indeed a much better reduction scheme than what was in Form. The more important BEBE topology however has still resisted a complete investigation due to some extra complications which reduce the number of available equations and hence make most selections run into very long execution times (to be mentioned when we define complexity).

Programs like the above depend very much on built in smartness to reject hopeless cases as soon as possible. Such rules have to be found heuristically and usually involve much time for experiments.
Of course one can derive reduction schemes automatically without such optimizations, but the result will be that the actual calculations will be extremely slow due to the inefficiency of the scheme.
The worst reduction equations are typically about halfway the reduction, when about half the number of parameters is still present, but already the powers of the monomials can become rather high. This is comparable to final results in the 'many legs and few loops' case where there are many physical parameters remaining. In the case of the 'few legs and many loops' diagrams and higher powers of numerators or denominators, one may have to apply such reduction formulas many times, hence it is important to keep such formulas as short as possible.

The Forcer program has been used already for a number of calculations. Some of them are

- Two different ways to compute the five-loop beta function.
- Higgs decay.
- A number of Mellin moments of the splitting functions in DIS at the 4-loop level. This is needed for the 3-loop Higgs production at the LHC.
- Even some Mellin moments of the splitting functions in DIS at the 5-loop level.

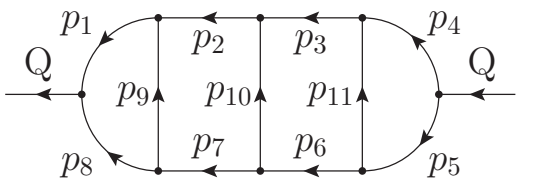
Currently the splitting functions for the non-singlet are being run at $N = 10$ and the singlet at $N = 8$. These are enormous tasks with individual diagrams that can take weeks on 16 cores. Sven Moch will tell us more about it.
The hope is that when all topologies can be optimized some more Mellin moments can be computed, giving a higher precision in the determination of the PDF's at the LHC.

Having a four loop program, the next question is: what about five loops? The Forcer program as it exists was derived by hand guided computer programs. Reduction schemes were needed for 21 topologies and took three people several months to derive. For five loops there are about 10 times as many of such topologies, and most have 20 variables that need to be reduced. This is only feasible with an even higher level of automatization.

The flow from topology to topology is not so much the problem. For Forcer this gluing together of the topologies was done with a Python program, but the newest version of Form has the Kaneko diagram generator built in, making it at least as easy to do this in Form itself. For a five (or more) loop program most of this has now been set up in Form. There are however a few unresolved issues. The most important are:

- The reduction scheme of the 'nontrivial' topologies.
- The selection of the notation for such topologies.

In the Forcer program it turns out that often most of the time is spent when transitioning from one topology to simpler topologies (with one fewer edge). They may have different notations. This is shown by:



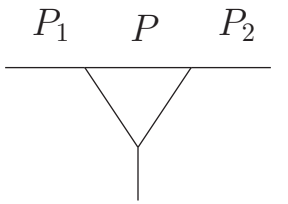
If we eliminate either of the lines 1,4,5,8 we obtain identical topologies, but the transitions to the notation of this topology are different. The same holds when we eliminate either of the lines 2,3,6,7, and similar when we eliminate one of the lines 9 or 11.
It means that the numerators have to be rewritten, unless we have a reduction scheme for each notation that can be encountered. This would need for instance 34 different reduction schemes for the bebe topology. It has already been noticed that not all of those schemes would be very efficient. Hence one should create a number of efficient schemes, and rely for the others on a minimal amount of rewriting. It should be clear that designing a program that can do all this automatically is not trivial. Some progress has been made though.

We did find a much better bebe reduction scheme than the one in Forcer. Replacing the Forcer routine was not easy because it was all handwork to change the notations at the input and the output. The results was:

- A much faster overal running time.
- Fewer spurious poles (the equivalent of better numerical stability).

What are spurious poles?

IBP reduction algorithms have the tendency to create poles in ϵ whenever a line/edge is eliminated. The result is that diagrams with for instance the 3-loop ladder topology at the three loop level will give individual terms with up to 6 powers of $1/\epsilon$ during the reduction, while, once all terms are added, the leading divergence is at most $1/\epsilon^3$. This is the normal situation. Spurious poles can occur when there are powers of numerators and we take a close look at the most common reduction equation which is called the triangle relation:



$$
\begin{aligned}
I(n,\alpha_0,\beta_1,\beta_2,\alpha_1,\alpha_2) = (\\
&+\beta_1(I(n,\alpha_0-1,\beta_1+1,\beta_2,\alpha_1,\alpha_2) - I(n,\alpha_0,\beta_1+1,\beta_2,\alpha_1-1,\alpha_2))\\
&+\beta_2(I(n,\alpha_0-1,\beta_1,\beta_2+1,\alpha_1,\alpha_2) - I(n,\alpha_0,\beta_1,\beta_2+1,\alpha_1,\alpha_2-1))\\
&)/(n+4-2\epsilon-2\alpha_0-\beta_1-\beta_2).
\end{aligned}
$$

The parameter n is the number of times the vector P occurs in the numerator. The parameters α_1 and α_2 are the powers of the two lines outside the triangle with the momenta P_1 and P_2, while β_1 and β_2 are the powers of the adjacent lines inside the triangle.
If all denominators have power 1 and there are no numerators, we see indeed that eliminating one line gives us a power of $1/\epsilon$. But we can obtain more than one power when there are more powers of numerators and denominators because now $(n+4-2\alpha_0-\beta_1-\beta_2)$ can pass through zero more than once during the full reduction.

If one works with rational polynomials in ϵ this is not so much of a problem, but this is rather slow. It is faster to work with finite expansions in ϵ, thereby avoiding rational polynomial arithmetic. This is similar to rational numbers versus floating point numbers which have a finite accuracy and are sensitive to numerical instabilities.

In the case of the Mincer program the spurious poles were avoided by a summation of the recursion and properly adding all resulting terms. This is however either not feasible or not practical when more loops are involved. Hence, whenever powers of numerators are involved we have to be prepared for extra powers of $1/\epsilon$. And it can be rather difficult to predict how many. There are two solutions around this:

1. Work with exact rational coefficients.
2. Work with a cut off series but maybe run more than once to get an idea how many powers are needed cq. sufficient.

The first solution turned out to be rather expensive. Hence we worked with the second solution and eventually we got a heuristic formula to how many powers in ϵ we had to go to be able to trust the results. This formula depended linearly on the number of the Mellin moment.

As it turns out, with the new bebe routine this dependence disappeared. Let us see how that can be, because this will be important for the automated derivation of reduction schemes.

In order to have a reduction scheme one needs an ordering of the integrals in the form of a complexity. If each equation in the scheme lowers this complexity, and below a certain level at least one line/edge vanishes creating a simpler topology, eventually the scheme will terminate. The most important parameter is of course the number of edges. We cannot possibly accept relations that increase the number of edges. We do not include this parameter in our definition of complexity.

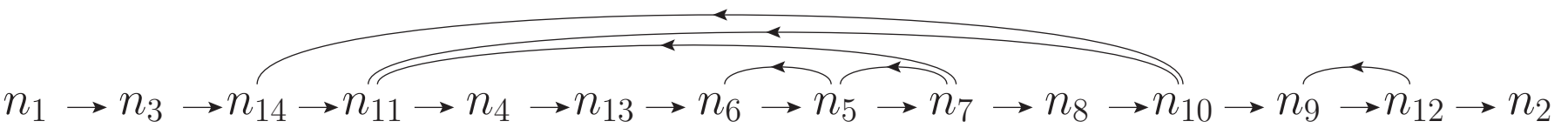
We have to proceed as follows:

1. The most important part of the complexity is the sum of the deviations from their minimal value for all parameters combined. We call this the sum-complexity.
2. Next we can lower indivual parameters , but this may go at the cost of raising others, provided the sum complexity is not increasing.
3. Then we have to find an ordering of these reductions that does not create loops in the scheme.

The above three steps are in principle sufficient, but not necessarily efficient.

For the next step we need the concept of kickbacks. When we lower the sum complexity by one, it may well be that we lower several variables at the cost of several others. And if one of those that are raised is at the top of the scheme, we have to start all over again, be it with one sum complexity less. We call this a large kickback. The better schemes have as few of such kickbacks as possible, and the ones they have are only very few steps back. In particular, if whatever exists in kickbacks does not involve numerators, the number of spurious poles will be severely limited. Also the rational coefficients will be comparatively mild and so will be the number of terms at the end of the reduction.

Hence the ordering of the reduction of the variables should take these kickbacks into account.



This is exactly what happened with the bebe routine. The old bebe routine, created by handguided computing, but without much experience, had a few rather bad kickbacks. The new one, designed with much more automation, gave already shorter code, and by exchanging the order of some of the variables by hand, the code became only marginally longer, but kickback-free. This made a big difference, both in speed and in the number of terms in the final answer.

The bebe reduction has another problem. The nasties mentioned before. The structure of the equations is such that the only solutions that we could find start off with eliminating two variables, but each step increases the sum complexity by one. This had to be done before the equations were generated that are used for the other variables, thereby avoiding kickbacks and hence infinite loops. It does however make the number of useful relations for the other reductions significantly smaller.

Such approaches are not needed for the other topologies, but can be used anyway in such a way that the results are improved. This is still being studied.

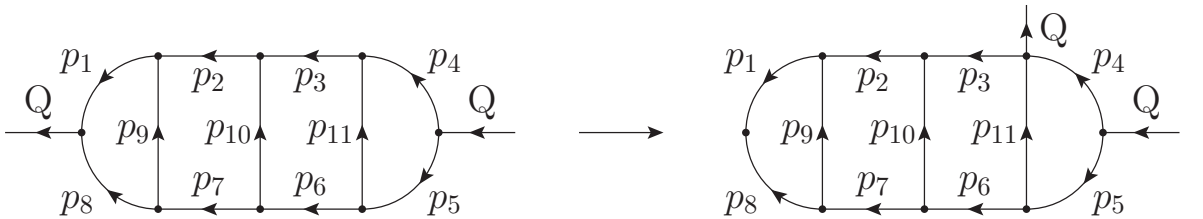
Unfortunately the situation with the other topologies that still need to be cleaned up is slightly more complicated. Also, it is not easy to replace topology routines inside Forcer, because of the changes in notation. A routine like bebe can be called in 34 different ways, and creates output in 8 different ways. This has to be changed by hand, because the original Python code had some global optimizations in which tiny changes anywhere, could change all numberings and notations of diagrams in the complete program, invalidating all partially built up databases. The future Form program should not have that problem. The hope is that with the new computer we just got we can make a new attempt at the other topologies.

Of course at the five loop level this will be even worse, because then 20 variables have to be eliminated, rather than 14, and in addition the equations will be lengthier. In the next section we will see how one can still obtain some five-loop results.

3 A little bit about Rstar

If one is only interested in the UV divergent terms of a diagram, there are a few theorems that can be applied. They make the calculation feasible with a program that can solve diagrams with one loop fewer.
When calculating only the UV divergences of a diagram there are a few theorems about extracting this divergence and rearranging external lines.
In the case of propagator diagrams one may extract the divergence by taking enough derivatives of the momentum to make the result dimensionless.

If a propagator diagram is dimensionless, we may as well set the momentum to zero, or move external legs to other points in the diagram. This does not change the ultraviolet divergence, but it may affect the IR divergences. This way we can change the diagram to one in which we can do one integral and reduce the problem to an integral that can be done at a simpler level.



Integrals with only one line between the two Q's we call carpet integrals and they can be done 'trivially'. But because now $p_1^2 = p_8^2$ we have introduced a potential infrared divergence. This is the main reason why it took so many years to do the four-loop QCD beta function and later the five-loop QCD beta function. Cannot we hunt down those IR divergences?

For the subtraction of divergences there exists a procedure, called Rstar. This has been used a lot in ϕ^4 theories. In QCD it is more complicated bacause the gluons introduce dotproducts in the numerators and hence complicated tensors. The whole is a very tricky interplay of determining subgraphs, locally eliminating tensors in the numerators by derivation and keeping the orders of limits and derivations correct. Details are in the paper by Herzog and Ruijl .

One practical problem is that the tensors need projectors to take them from inside the derivatives and the counterterm subtractions to outside of these.
Effectively one needs a projection operator $T^{\mu_1\cdots\mu_n}_{\nu_1\cdots\nu_n}$ that projects out products of Kronecker deltas like $\delta^{\mu_1}_{\nu_1}\cdots\delta^{\mu_n}_{\nu_n}$. As it turns out, it is an exercise in cosets of the symmetric group S_n in which each coset has a coefficient that is a rational polynomial in the dimension D. And it happens that the generalised Kronecker delta dd_ in Form is ideal for this kind of work because, when the indices are contracted with momenta, of which there are only a few different ones, Form gets the combinatorics factors right without having to generate the same term many times. We managed to determine these coefficients all the way to 16 pairs of indices (in the process of being written up).

```
    Vector p1,p2,p3;
    Local F = dd_(p1,p2,p3,p1,p2,p3,p1,p2,p3,p1,p2,p3);
    Print +f +s;
    .end

Time =       0.00 sec    Generated terms =          15
               F         Terms in output =          15
                         Bytes used      =         868
   F =
       + 1728*p1.p1*p1.p2*p1.p3*p2.p2*p2.p3*p3.p3
       + 1152*p1.p1*p1.p2*p1.p3*p2.p3^3
       + 216*p1.p1*p1.p2^2*p2.p2*p3.p3^2
       + 864*p1.p1*p1.p2^2*p2.p3^2*p3.p3
       + 864*p1.p1*p1.p3^2*p2.p2*p2.p3^2
```

```
 + 216*p1.p1*p1.p3^2*p2.p2^2*p3.p3
 + 216*p1.p1^2*p2.p2*p2.p3^2*p3.p3
 + 27*p1.p1^2*p2.p2^2*p3.p3^2
 + 72*p1.p1^2*p2.p3^4
 + 1152*p1.p2*p1.p3^3*p2.p2*p2.p3
 + 864*p1.p2^2*p1.p3^2*p2.p2*p3.p3
 + 1728*p1.p2^2*p1.p3^2*p2.p3^2
 + 1152*p1.p2^3*p1.p3*p2.p3*p3.p3
 + 72*p1.p2^4*p3.p3^2
 + 72*p1.p3^4*p2.p2^2
;
```

To do all possible diagrams in QCD one also needs to deal with indices on gamma matrices. It is still an outstanding problem when there is a mixture of Kronecker delta's and gamma matrices. The gamma matrices can be written in the antisymmetric 'σ' basis, but it ceases to be a simple application of the symmetric group. This is currently under study.

Also here the continuous changing of (sub)topologies and hence notations is a serious problem. Because in the local Rstar operation this is done term by term, such operations become very time consuming, unless specialized tables are constructed in advance. Part of this is already present in Form by means of the Kaneko diagram generator, but the canonicalization of the notations still has to be built in. Using external code is rather slow.

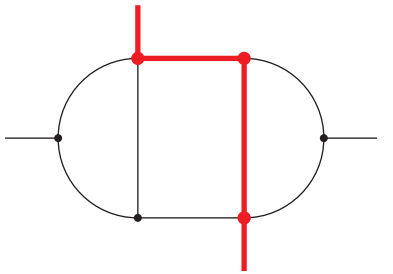
4 Mathematical aspects

In the sequel we ignore color factors. Each color channel can be seen as a separate object for the purpose of the current discussion.
Evaluating master integrals is a science by itself. In the case that we have massless propagator graphs, the answer should be 'just a number'. This holds also for beta functions. What numbers are involved?

Until now Multiple Zeta Values have been sufficient. There are indications that at more loops these may not be sufficient. Thus far this has been shown for individual diagrams cq. topologies. This is not yet a proof that it will hold for complete physical results. Personally I have encountered two cases in which the more complicated objects cancelled each other.

In the first computation of the four-loop beta function in QCD there were master integrals that we could not evaluate at the time, but they dropped out when the diagrams were added. Here the method was to blame. The infrared regularization was based on masses and this gave rather hard integrals. When done with a program like Forcer there are no mass parameters and this effect does not occur. Also not at five loops.

In the computation of all Mellin moments of the coefficient functions in three-loop DIS, one sum occurred that was not a simple harmonic sum. We just gave it a name when it survived in individual diagrams, but when all diagrams were added this object vanished.



It is of course likely that with extra loops such miracles do not occur.

When we have integrals or sums that end up in terms of MZV's or Euler sums there are usually very many different ones. There exist however many relations between them due to shuffle and stuffle relations. Such relations can be combined to reduce all different MZV's and Euler sums to an independent set. This set is usually rather restricted.

Example of a stuffle relation, which comes from the sum property of MZV's:

```
 CFunction S;
 L F = S(2,3)*S(5);
 Stuffle S-;
 Print +s;
 .end
F =
    + S(2,3,5) + S(2,5,3)
    - S(2,8) + S(5,2,3) - S(7,3);
```

Example of a shuffle relation, which comes from the integral property of MZV's:

```
 CFunction H;
 L F = H(2,3)*H(5);
 Transform H,ToIntegralNotation(1,last);
 Print;
 .sort
F =
   H(0,0,0,0,1)*H(0,1,0,0,1);
```

```
 Shuffle H;
 Transform H,ToSumNotation(1,last);
 Print +s;
 .end
F =
    + H(2,3,5) + 3*H(2,4,4) + 7*H(2,5,3) + 15*H(2,6,2)
    + 30*H(2,7,1) + 2*H(3,3,4) + 8*H(3,4,3) + 20*H(3,5,2)
    + 40*H(3,6,1) + 6*H(4,3,3) + 18*H(4,4,2) + 36*H(4,5,1)
    + 5*H(5,2,3) + 12*H(5,3,2) + 24*H(5,4,1) + 10*H(6,1,3)
    + 5*H(6,2,2) + 10*H(6,3,1);
```

By writing down all shuffle and stuffle relations and 'solving' them, one can determine a basis and express all MZV's in terms of this basis.

Complete reduction relations have been worked out in the MZV datamine and go to weight 12 for the Euler sums and weight 22 for the MZV's. This was the limit of what could be reached with the existing computer resources at the time (2009). Recently I got a new computer with 64 cores and 1 Tbyte of memory. I managed to compute the weight 13 Euler sum reductions. The run took 11 days in which it used 637 days of CPU time (a pseudo-efficiency of 58). The run for weight 12 took 5 hours, using 161 hours of CPU time. It shows that it is not realistic to expect much of an extension to the datamine. Actually the situation can become much worse when the alphabet has more than 3 elements.

As Broadhurst showed, at a given moment some not very high loop diagrams may result in sums in which the alphabet involves the sixth root of unity. Let us consider how many finite sums exist for a given alphabet as a function of the number m of its elements. For the MZV's the alphabet contains the two elements 0 and 1, while for the Euler sums the alphabet contains the elements 0, 1 and -1. The generic formula is $(m-1)^2 m^{N-2}$, possibly divided roughly by two when there are duality considerations.

weight	2	3	7
2	1	4	36
3	2	12	252
4	4	36	1764
5	8	108	12348
6	16	324	86436
7	32	972	605052
8	64	2916	4235364
9	128	8748	
10	256	26244	
11	512	78732	
12	1024	236196	
13	2048	708588	
22	1048576		

In general one needs to solve only for about half the number of variables.

To do this for such a large number of variables requires special programs, as is explained in the paper about the datamine. One complication is that the rational coefficients become rather bad. Their complexity seems to be dictated by the number of variables. Of course one can try to solve the systems over a finite field, like modulus a 31 bit prime number and after doing this several times, one can reconstruct the coefficients. This was also done for the MZV's at weight 22, but the result was that the combined running time of the modulus programs took more time than working directly with the rational numbers.

If weight 13 for the alternating sums is already barely feasible, it should be clear that we can never reach such weights for the sixth root of unity. Hence another approach is called for, which is based on a paper by Francis Brown and a paper by David Broadhurst . Most advanced in applying this is Oliver Schnetz . The way it works is as follows:

Based on the theory of motivic multiple zeta values one can construct algebraic operators that can map any MZV onto elements in a space that has the same dimension d as a basis for the given weight w. Because this is all linear, one can determine whether, if one chooses d MZV's, these form a basis for this weight. This needs however already expressions for weights lower or equal to $w-3$, which means that the whole algorithm is effectively recursive. Once a basis is known one can obtain the coefficients for each individual MZV in terms of this basis with the exception of one coefficient: the coefficient of the depth 1 term, which, when the weight is even, can be expressed as a power of ζ_2.

This is where the Broadhurst paper comes in. He explains how one can determine any MZV numerically to very high precision. If necessary many thousands of digits. Having routines for this allows then to determine the last coefficient numerically in terms of a high precision floating point number, which then can be converted to a fraction.

The above method is explained for the MZV's but can also be applied to Euler sums or to alphabets based on higher roots of unity. In this way Oliver Schnetz has created already programs that can determine alternating sums to weight 21 and MZV's to weight 32. In addition he has already some programs for higher roots of unity, like up to weight 13 for the sixth roots of unity and up to weight 17 for the third and fourth roots of unity.

For the higher weights the programs can become a bit slow, because they will need to do calculations with many thousands of bits accuracy, and the evaluation of the MZV's needs as many steps as there are bits in the accuracy.

Currently those programs are mainly interesting from the mathematical viewpoint, but it is inevitable that once more loops can be dealt with, physics will need them as well.

5 Computer algebra

It should be clear that when diagrams are calculated the easiest would be if it can all be done in a single program. Unfortunately version 4 of Form cannot deal with floating point numbers. Hence for the necessary type of operations version 5 will be equipped with some new functions that allow access to the arbitrary precision floating point facilities of the GMP library. In addition there will be some built in functions like mzv_ and euler_ to evaluate these sums to a user defined precision. More functions (sqrt_, ln_, etc.) still need to be programmed. And of course there are some commands that allow conversions between rational and floating point numbers. This was not completely trivial, because this is rather specialized stuff that should not be in the way of the regular operations of Form.

Let me give an example: Assume we want to know the expression for the

```
L  F = mzv_(2,5,3);
```

The Francis Bown part of the program would give us

```
L   Fa = +20*mzv_(2)*mzv_(3)*mzv_(5)
         -34*mzv_(3)*mzv_(7)
         -447/14*mzv_(5)^2
          -22/7*mzv_(7,3);
```

but the coefficient of `mzv_(10)` or `mzv_(2)^5` is missing. The interesting part of the program becomes now

```
 #StartFloat 400
 *
 L   F = mzv_(2,5,3);
 L   Fa = +20*mzv_(2)*mzv_(3)*mzv_(5)
          -34*mzv_(3)*mzv_(7)
          -447/14*mzv_(5)^2
           -22/7*mzv_(7,3);
 L   Fb = mzv_(2)^5;
 Evaluate mzv_;
 Print;
 .sort

F =
    + 1.401734757585418072158234569433678567438982623228\
   5932441172391706367200419120404878187236028403844898392\
   7719621931729000000000000000000000000000000e-02
   ;
```

```
Fa =
    - 3.456140777152661943477201458825007526066031695778\
   8\
   586596697064256953964603926330089589639190454970585852\
   95600157098180000000000000000000000000000000e+01
   ;

Fb =
    + 1.204321598200656133921253731801946317335367248913\
   5\
   120129507773841717881690108952732589411184843702384696\
   64542013543780000000000000000000000000000000e+01
   ;

 L   FF = (F-Fa)/Fb;
 Print FF;
 .sort

FF =
    + 2.870946196660482374768089053803339517625231910946\
   1\
```

```
    9666048237476808905380333951762523191094619666048237 47\
    6808905380322000000000000000000000000000000e+00
    ;

 ToRational;
 Print FF;
 .end

FF =
     + 38686/13475
    ;
```

Checking with the datamine gives indeed this fraction.

The major work now is to figure out what is a good accuracy for the floating point numbers. This will be a function of the type of the sums and the maximal weight we would like to go to. This needs some experimentation and extrapolation. For now we can use the datamine of course.

For other roots of unity the experimental method is called for. If there is not enough accuracy, the conversion to a fraction will give rather 'unphysical' results. One can start with too much accuracy for a number of cases and then work it down to see how long the proper fractions are reconstructed. Then give it some extra accuracy and one is ready to go. And there is always the check of the stuffle and shuffle relations.

Of course, much speed can be gained if we tabulate the numerical value of the basis elements. This has one problem. These basis elements have different numbers of indices and tables have typically a fixed number of indices. Hence Form has now been extended with tables of which the number of indices does not have to be fixed. This gives yet another complication. Tables in Form can have arguments in addition to indices, and the number of arguments follows the rules of wildcarding, and hence the number of arguments could also not be fixed. Therefore the notation becomes:

```
Table,sparse,T1(<=5,arguments);
Table,sparse,T2(<=10,arguments);
```

in which 'arguments' represents a potential field of arguments as in the definition of regular tables. It needs a maximum number, because space is reserved for this maximum.

When such tables are used, the first index should be a number that tells how many real indices there are, because as mentioned before, the number of arguments does not have to be constants when `?a` style wildcards are involved, which pick up a whole field of arguments.

With the above we can make one single table for all basis elements of which we would like to store the numerical value. Such a table can be stored for future programs and its size is only a small fraction of the size of complete tables.

6 Outlook

To make progress in the field of few legs and many loops we need either completely new methods, or a number of very powerful new automated programs to break down the integrals to master integrals.

It is worth to put effort in this, because accurate calculations for reactions at the LHC need reliable 4-loop splitting functions.
Steady progress is made and more is needed, both in the fields of mathematics and computer algebra.
Dedicated programs for specific tasks may help, but have mainly been constructed for the 'many legs and few loops' calculations.