1 An unfinished project

This lecture is based on the following papers:

- D.J. Broadhurst and D.Kreimer: "Association of multiple zeta values with positive knots via Feynman diagrams up to 9 loops", Phys.Lett.B 393(1997) 403-412, ePrint: hep-th/9609128.
- J. Blümlein, D.J. Broadhurst and J.A.M.Vermaseren, Comput.Phys.Commun. 181(2010)582-625. ePrint: 0907.2557
- J.Kuipers and J.A.M.Vermaseren: "About a conjectured basis for multiple zeta values": ePrint: 1105.1884 math-ph.
- Francis C.S. Brown: "On the decomposition of motivic multiple zeta values", ePrint 1102.1310 Math.NT

During this talk we will assume that the results for motivic multiple zeta values of the Francis Brown paper also hold for regular multiple zeta values. This has however not been proven (yet?).

One type of constants we encounter in Feynman diagram calculations are the Multiple Zeta Values (MZV's) and some of their generalisations like the Euler sums. When we have more loops, these objects occur at a higher weight, and a higher weight means more nested sums, and hence a much larger variety of them. For the MZV's there are basically 2^{w-3} different ones and for the Euler sums $4\ 3^{w-3}$ different ones. If one considers higher roots of unity, the number of which we have to take the power is $r+1$ when the 'alphabet' is the r-th root of unity. And there are forcasts that sooner or later we will run into the 6-th root of unity.

Fortunately not all those MZV's are linearly independent. There are relations between them, based on two algebras. The result is that a basis for these MZV's is usually relatively small, and if one can express the MZV's that occur in a calculation in terms of such a basis the formula's simplify very much. The main problem here is however how to express the MZV's and Euler sums in terms of such a basis. And how to find such a basis?

One method is to write down all relations that are a consequence of the two algebras, and then solve them into a set of independent variables. Although rather primitive, this was for a while the only known method. It also proves that all coefficients in the solution must be rational numbers.

A bit later Broadhurst started experimenting with numerical methods, in which he evaluated the sums numerically, guessed a basis, and then used an LLL algorithm to fit these values to the basis. Also the determination of the basis was of course due to the fact that the LLL algorithm could not find relations, if you did have a linearly independent set. This is how Broadhurst came to a guess about the size of the bases.

A more recent breakthrough was a paper by Francis Brown, in which he managed to find a method to

1. determine a basis
2. express any MZV in terms of a given basis.

The only weakness of his method is that he can obtain all coefficients in such an expression as rational numbers, with the exception of one: the coefficient of the unique depth 1 basis element. But if it is only a single number, and the first method of solving the equations based on the algebras shows that all coefficients must be rational numbers, one can evaluate the missing number first as a floating point number and, provided there is enough precision, convert it to a rational number in the end. This is the method we are going to study here.

While David Broadhurst was experimenting with his numerical methods, he noticed something irregular at weight 12. He thought that he had found a rule for the construction of the basis by means of Lyndon words. The assumption was that the indices should be Lyndon words of odd integers greater than one. In the case of weight 12, those would be $MZV(9,3)$ and $MZV(7,5)$, but one of those elements was not in the basis. instead he needed one with 4 indices: $MZV(6,4,1,1)$. It became even stranger when he embedded the whole in terms of Euler sums, because the missing element could be expressed as an Euler sum with only two negative indices. He called this a pushdown. When investigating further he found some more pushdowns at higher weights. Together with D. Kreimer he found a formula that would tell in more detail when such pushdowns would occur. Unfortunately they did not consider the possibility of double pushdowns. Later investigations for the datamine suggested that there might exist double pushdowns, starting at weight 27. It turned out that the Kreimer and Broadhurst formula could be repaired (it took David one evening). It now predicts the first triple pushdown at weight 39. A few years later the first two double pushdowns were indeed found at weights 27 and 28. The whole mechanism of these pushdowns remains mysterious, even though a notation for them has been found.

The MZV datamine runs complete relations to weight 22 for the MZV's and partial relations modulus a 31-bit prime number up to weight 27 and 28. The problem here is that all equations based on the algebras had to be generated and solved, which becomes a horrendous task. The run to determine the relations of depth 9 at weight 27 took at the time more than 80 days with 8 processors on the computer at Karlsruhe.

The method of Francis Brown seems much better to determine bases and create individual relations. This was first implemented by Oliver Schnetz and he managed to construct bases to weight 33 (or even 34?) for the MZV's, and lower weights for higher roots of unity. This was about as far as he could go, but he hoped that with FORM we might be able to get a bit further. This hope was based on the fact that already, by using the FORM implementation for the evaluation of MZV's he could improve his programs, and the manipulation of FORM of large expressions is also quite good. The hope is of course to make it to weight 39. This would show whether indeed there will be the triple pushdown, and of course what all intermediate pushdowns look like. This in the hope to understand them better.

The idea will be to determine a basis weight by weight. But whenever we have to do the test whether something is a basis, the MZV's are split into products of MZV's with lower weights which are not necessarily basis elements. Hence we have to express MZV's at a lower weight in terms of their bases. Rather than storing complete relations, which takes enormous amounts of disk space, we will only store the rational number of the unknown coefficient, because that is the only very expensive object. All other coefficients are relatively easy to obtain. In addition, we only store the ones we actually need, once the weights become too big. To get the program started we take the relations of the MZV datamine up to weight 8. Anything higher we will generate successively and potentially store in files for later use.

Because running with very high floating point precision is very expensive, it seems best to adapt the precision to the weight.

Assume we work at weight w. What we have to do is determine a basis first. Of course the datamine gives us a decent guess at a basis by starting to form all Lyndon words of odd integers greater than one. Unfortunately the mechanism of pushdowns makes that some may not be part of a basis, but the mechanism that is a fair guess how to repair that, is that an n-fold pushdown involves taking the first 2n digits of the Lyndon word, lowering them by one each, and adding 2n ones to the Lyndon word. The only uncertainty about this is that we do not know which Lyndon words we have to treat. But it is not a prohibitive amount of work to try this out.

Example:

$$\begin{aligned} MZV(7,5,3) &\rightarrow MZV(6,4,3,1,1) \\ MZV(7,5,7,5,3) &\rightarrow MZV(6,4,6,4,3,1,1,1,1) \end{aligned}$$

The algorithm for determining whether a set of MZV's forms a basis is given by Francis Brown. We do not have time to go into its details. It suffices to say that it can be programmed in FORM. As mentioned, the main complication is that during the procedure MZV's are split into products of MZV's of a smaller weight and those may not be basis elements. Hence these MZV's at lower weights need to be expressed in terms of the lower weight basis elements. If we do not have those relations ready, it means that they have to be derived on the spot. This makes for some very careful programming, because the whole algorithm is of course recursive, but at the same time we do not want to blow up the expressions more than absolutely necessary, if we want to go to the limits of what is possible. Hence we tabulate the relations up to the point that we think that the tables become too big, and from that point on we have to work dynamically. With the coefficients of the depth one terms in relations we can continue a little bit further, but at a given point we also have to give up on tabulating all of them. From that point we create a tablebase that only contains the numbers we evaluated before.

Let us start with the program for constructing the bases. It starts with:

```
#ifndef ‘MAX’
    #define MAX "22"
#endif
#write "Running the basis for weight = ‘MAX’"
#include hmzv.h
*#include hh.h
off Statistics;
.global
#call StartFloat(‘MAX’)
#if ( ‘MAX’ >= 29 )
#call StartDepth1Table(28,0)
#else
#call StartDepth1Table({‘MAX’-1},0)
#endif
#call StartffbasisTable({‘MAX’-1})
#call StartBareTable({‘MAX’-1})
.global
#call CreateElements(‘MAX’,F,H)
.sort
```

```
L   FF = F;
if ( ( count(H,1) == 1 ) && ( count(z2,1) == 0 ) );
    if ( expression(FF) );
        Transform,H,IsLyndon>(1,last)=(1,0);
    else;
        Transform,H,IsLyndon>(1,last)=(0,1);
    endif;
else;
    if ( expression(FF) ) Discard;
endif;
id  H(?a) = pushdown(nargs_(?a),?a);
id  pushdown(n?,?a) = H(?a);
$count = count_(H,1);
if ( $count > 1 ) id H(n?)^$count = 0;
ModuleOption local $count;
.sort
```

The procedure StartFloat estimates the needed precision and starts up the floating point system, including the allocations for the auxiliary tables for the evaluation of the MZV's.

The StartDepth1Table reads the tables of the depth1 coefficients for all relevant MZV's, when expressed in terms of the previously determined bases. Up to weight 8 they are hard-coded, as taken from the datamine. It may seem that some are missing, but if one checks the paper by Francis Brown, one will see that there is a conjugation that makes that those missing ones are easily converted to elements that are in the table. Up to weight 28 we expect complete tables. Above that weight such tables would become too big and they will only contain elements that were used during previous parts of the project. Each time there are new ones they will be added to the tables. To this end, we make a backup of these partial tables at the start of the program to avoid loosing much work, if the program crashes. It should be noted, that because this project was unfinished, this last feature is currently not active.

The next procedure is StartffbasisTable. This procedure picks up the lower weight bases. Again, up to weight 8 it is hard-coded as taken from the datamine. Note that we leave out the depth 1 elements of the even weights. They are multiples of powers of ζ_2. Depth one elements play a special role as remarked before.

In the StartBareTable procedure we evaluate the MZV's in the bases to the required precision. This way we do not have to evaluate them more than once. The notation here is that we write

```
id  mzv_(?a) = mzv_(?a)*H(nargs_(?a),?a);
```

and then evaluate the mzv_ outside the function, while the arguments of the function tell us which MZV this is. We could try to do this the other way around, but we would rather not have large floating point numbers inside function arguments. Maybe this is an irrational fear.

Because of the work with the datamine, we know all pushdown bases up to weight 28. We add those here. This means that actually all the way up to weight 28 the choice of the bases will be rather trivial, because by undoing these pushdowns, we can tell immediately which Lyndon words do not belong to the bases.

Finally we produce the potential basis elements for the given weight. This is done in the procedure CreateElements:

```
#procedure CreateElements(W,F,H)
*
*   Creates the full set of potential basis elements for a
*   given weight W, excluding the ones that involve products with H(2)
*   because those should already be in the tables.
*   The complete set will be in the expression F.
*   We assume that all lower bases exist already.
*
*   This algorithm is not very efficient.
*   It can be improved by building it up slowly, using tabulation.
*   To be done later.....
*
L   `F' = `H'x(`W');
#$ncall = 0;
#do iCE = 1,1
    #$ncall = $ncall+1;
    #$catch = 0;
    id  `H'x(3,?a) = `H'(3,?a)+`H'(3)*`H'(?a);
    id  `H' = 0;
    id  `H'x(n?,?a) = sum_(j,3,n,2,`H'x(n-j,j,?a))
                +sum_(j,3,n,`H'x(n-j,?a)*`H'x(j));
```

```
        id  ‘H’x(0,?a) = ‘H’(?a);
        id  ‘H’x(n?{1,2,4},?a) = 0;
        if ( count(‘H’x,1) ) $catch = 1;
        ModuleOption maximum,$catch;
        .sort: CE-‘$ncall’;
        #if ( ‘$catch’ == 1 )
            #redefine iCE "0"
        #endif
    #enddo
    DropCoefficient;
    Transform ‘H’,IsLyndon>(1,last)=(1,0);
    .sort:CE-end;
    #endprocedure
```

Effectively it creates all products of Lyndon words of odd integers greater than one for the specified weight.

The DoBasisElement applies the Francis Brown algorithm. After that we substitute the coefficients of the depth-1 terms that are in the tables. It will now become clear that some of the Lyndon words are not in the basis. Hence we have to try pushdowns of such Lyndon words. This is however not unique, because if there is a relation between some Lyndon words, which one is the one to try for a pushdown. One of the (arbitrary) requirements we pose, is that the pushdown version is also a Lyndon word. This cuts it down somewhat. One may add more heuristics here, but in practise we just try, first with single pushdowns, and if we run out of those we can go to multiple pushdowns. Hence, by looping, we will eventually find a basis. It does help that we know the size of the basis. Originally this size was just conjectured by Broadhurst, but at some point Hoffman suggested a basis that consisted of MZV's with all combinations of indices with the values 2 and 3. Francis Brown managed to prove that this is indeed a basis and hence the size of the basis is now know. For practical purposes the Hofman basis is not one we like to use though.

The above allows us to go to an ever higher weight, provided we have enough resources to work out the intermediate steps. And these intermediate steps involve expressing any lower weight MZV in terms of its basis elements in an economical way. Hence that is the next problem to solve. The paper by Brown gives an explicit example, but alas the example would have been more explicit, if the weight had been chosen to be a bit higher, making the use of MZV's with more than one index needed in the basis.

The first step is to program the various basic operations and put those in the library:

```
*--## Rule0 :
*--## Rule1 :
*--## Rule2 :
*--## Rule3 :
*--## Rule4 :
*--## applyHduality :
*--## GetHl :
*--## MakeDual :
*--## OldMissingHl :
*--## MissingHl :
*--## HlFill :
*--## DeriveI :
*--## dophi :
```

An example is the ‘derivative’:

```
#procedure DeriveI(D,I)
*
*   Take the derivative and split in IL,IR.
*
id  ‘D’(rDI?)*‘I’(?a) =
        sum_(jDI,1,nargs_(?a)-1-rDI,
            ‘I’L(jDI,jDI+rDI+1,?a)*‘I’M(jDI+1,jDI+rDI,?a));
*
*   To get the proper ranges of arguments we have to work with $-variables.
*   Don’t forget to declare them local in the ModuleOption statement!
*
id  ‘I’L(jDI1?$j1,jDI2?$j2,?a) = ‘I’L(?a);
Transform ‘I’L,SelectArgs($j1,$j2);
id  ‘I’M(jDI1?$j1,jDI2?$j2,?a) = ‘I’M(?a);
Transform ‘I’M,DropArgs($j1,$j2);
*
*   Eliminate obvious zeroes
*
#call Rule1(‘I’L)
```

```
*
*   and get the IL to standard form.
*
#call Rule3(1,‘I’L)
*
*   now absorb the leading zeroes.
*
#call Rule2(‘I’L)
id  ‘I’M(1,0) = 1;
while ( match(‘I’L(1,1,?a)) );
    id,once,‘I’L(1,1,?a) = ‘I’La(1,1,?a);
    #call Rule4(‘I’La)
    #call Rule2(‘I’La)
    id  ‘I’La(?a) = ‘I’Lb(?a);
endwhile;
id  ‘I’Lb(?a) = ‘I’L(?a);
*
#endprocedure
```

The use of the transform statement with carefully selected dollar variables is quite efficient here.

Again we will not go into too many details, except for that we implement lots of local sorting by using the term/endterm environments. This will also make running with TFORM more efficient.

Having made some runs one can see already one of the problems:

```
668M May  4  2021 Htable20.sav
1.8G May  4  2021 Htable21.sav
4.7G May  4  2021 Htable22.sav
 13G May  4  2021 Htable23.sav
 33G May  5  2021 Htable24.sav
6.5K May  4  2021 mzvbare-22.sav
8.0K May  4  2021 mzvbare-23.sav
9.9K May  4  2021 mzvbare-24.sav
 13K May  6  2021 mzvbare-25.sav
 17K May  7  2021 mzvbare-26.sav
134M May  4  2021 mzvdepth1-22.sav
280M May  4  2021 mzvdepth1-23.sav
629M May  5  2021 mzvdepth1-24.sav
1.4G May  6  2021 mzvdepth1-25.sav
2.3G May 12  2021 mzvdepth1-26.sav
1.8M May  4  2021 mzvffbasis-22.sav
3.2M May  4  2021 mzvffbasis-23.sav
6.0M May  4  2021 mzvffbasis-24.sav
 12M May  6  2021 mzvffbasis-25.sav
```

`22M May 7 2021 mzvffbasis-26.sav`

Some of the files are becoming very big. In particular the Htable files which contain all relations for expressing MZV's in terms of basis elements. The basis routines are already far more sympathetic in size, although, if each extra weight makes them twice as big, at weight 39 the file will be very large as well. The depth1 files contain all coefficients of the depth1 basis element for all MZV's. Also this grows rather fast, and hence we should only store the ones that actually take part in the calculation when we go to larger values of the weight. The files mzvbare contain only the MZV's that occur in the basis of a given weight as in the case of $f(3)^3 f(5)$ it would contain the $f(3)$ and the $f(5)$ only.

The above indicates what strategy we will have to use.

```
        .......
tform -w32 -D MAX=22 Cscan > Cscan22.log
tform -w32 -D MAX=22 Depth1 > Depth1-22.log
tform -w32 -D MAX=22 Htab > Htab22.log
tform -w32 -D MAX=23 Cscan > Cscan23.log
tform -w32 -D MAX=23 Depth1 > Depth1-23.log
tform -w32 -D MAX=23 Htab > Htab23.log
tform -w32 -D MAX=24 Cscan > Cscan24.log
tform -w32 -D MAX=24 Depth1 > Depth1-24.log
tform -w32 -D MAX=24 Htab > Htab24.log
tform -w32 -D MAX=25 Cscan > Cscan25.log
tform -w32 -D MAX=25 Depth1 > Depth1-25.log
tform -w32 -D MAX=26 Cscan > Cscan26.log
tform -w32 -D MAX=26 Depth1 > Depth1-26.log
         ......
```

First of all, we build up the various tables in which Cscan determines the basis for a given weight,

Depth1 determines the coefficients for the depth one element for all MZV's of the given weight. This is the tail of the Depth1-24.log file:

```
      +Hl(21,1)*(238642176/11986349375)
      +Hl(21,1,1)*(77)
      +Hl(21,1,1,1)*(373229390371392/1139841893815625)
      +Hl(21,1,2)*(-846655033013536977 92/490655903211145546875)
      +Hl(21,2)*(-127)
      +Hl(21,2,1)*(-192664371400453086208/490655903211145546875)
      +Hl(21,3)*(zero)
      +Hl(22,1)*(11)
      +Hl(22,1,1)*(85786275890688/1139841893815625)
      +Hl(22,2)*(-19120909505536/346908402465625)
      +Hl(23,1)*(2178331462656/162834556259375);
  319.04 sec + 1151247.84 sec: 1151566.89 sec out of 37090.00 sec
```

and the Htab program makes the complete Htable .sav files.

Clearly we cannot continue as in the script. We need adaptations to these programs in which Cscan must be able to run without complete Htable and Depth1 results and compose only the missing table elements after which the new elements are added to the corresponding (partial) files. This is not completely trivial, and during the debugging of this step the project was interrupted due to personal disasters. Hopefully in the future the development can be picked up again because it is an interesting problem that will take FORM to its limits.