FORM, the end of an era?

J.A.M. Vermaseren

- Introduction
- When, what and why
- What could the future bring?

Introduction

More than 10 years ago David Broadhurst asked me at a "Loops and Legs" workshop: "What happens with Form if you fall in an Amsterdam canal?" There are a few remarks to be made here.

1. The canal would not be a problem. I swim well enough to stay afloat until I get fished out, and nowadays the water is clean enough that you don't dissolve in it.

2. The proper Amsterdam question would be: "if you get overrun by a tram?" Not hypothetical as that is what happened to Antoni Gaudi in 1926, and similar to what happened to Pierre Curie in 1906.

3. A brutish answer to the Amsterdam version of the question: "That would not be my problem, hence you should not ask the question to me."

Fortunately I can still help the community to prepare for something like this.

In this talk I will outline, without going into too many details, a few of the things that went into Form to make it what it is, and why it is like that.

Then I will explain that my personal involvement will be strongly reduced in the future and that people who want Form to be maintained and developed further should somehow arrange for it. I can lend a hand occasionally but do not expect me to organise it.

When, what and why

In the beginning there were Schoonschip, Reduce and Ashmedai. If you are interested in the earlier history of these programs, please see the contribution of Ettore Remiddi in the Acta Physica Polonica in honour of M. Veltman.

My thesis adviser Jack Smith introduced me to Schoonschip in late 1975. At the time it was the most powerful for the calculations we were doing. Unfortunately it ran only on CDC computers, because it was written in the assembler language of the CDC computers. We had to run it at the CDC6600 at Brookhaven at the cost of one dollar per CPU second. When the CDC computers started to become too expensive and were phased out in the mid 1980's, the prospects for big calculations became rather bleak.

In 1984 I wanted to start a project to create a system of programs that would calculate standard model reactions at the tree and one-loop level automatically. The core of this had to be an algebraic capability that would go beyond what the existing programs could provide me with. And another requirement was that if I were to need new features, I would not have to wait years for them to be implemented, if at all. Hence the Form project was started. The planning was that it would take two years and Form would be programmed in Fortran, because that language was available in all physics institutes.

It turned out that this was a bad choice and I got more or less stuck, because Fortran77 did not support recursions. Hence I had to emulate them with ever more complicated multi-dimensional arrays. At a given moment I had to take a break and learn much more about computers. During this I did a big project in the assembler language of the Motorola 68000 processor of my Atari ST computer and after that I learned about the C language (there was not yet a Fortran compiler for the Atari ST). I redid then the assembler project in C and studied what all the crummy but available C compilers made of my code in terms of assembler language. This taught me how to program in C in such a way that even a bad compiler could make decent code of it. After that I restarted the development of Form in C. From that point on it went very fast and in a few months I got much further than 2 years in Fortran had gotten me. It also taught me that first making a version zero, and then restarting from scratch gives much better programs, because in the second attempt you know much more about the nature of the problems you are going to run into.

Example of a C construct to gain speed: Global variables.

If you have global variables like

```
char *a;
long b;
int c;
```

and in other files you need

```
extern char *a;
extern long b;
extern int c;
```

the compilation wil usually create absolute addresses.

If however you program this as

```
typedef struct {
  char *a;
  long b;
  int c;
} structA;
structA A;
```

and

```
extern structA A;
```

there is only need for one absolute address and the variables are picked up as offsets to this one address. This means that the optimiser can put this one address in a register and now dealing with these variables is a bit faster. It also leaves more address registers available for other optimisations. A trick like this made Form a few percent faster. And at a later stage it made the parallelisation much easier.

When starting a big project, another consideration should be to set up a decent memory management first. Few people do this, relying on the memory management of the OS.

Very bad idea.

The OS has no idea what you are doing and hence gives more or less random swapping results. If you use knowledge about what you are doing and you implement it correctly you have a superior hit/miss ratio. This is at the basis of the ability of Form to deal rather fast with very big expressions.

In the same period Veltman had rewritten Schoonschip into the assembler language of the Motorola 68000 family. This became available when the Form project was already well on its way. People could buy it for their Atari ST computer.

Finally after 5 years the first version of Form could be released (1989).

Because everywhere I had used the best algorithms I could find cq. think of, it turned out that it was far more powerful than originallly anticipated. Even on Veltmans favorite benchmarks it would beat Schoonschip and there was a much higher degree of flexibility. And it was fully portable because by then nearly every computer had a C compiler, although IBM mainframes were a bad problem: too many compiler errors. This prevented optimisation of several files.

Form was planned to be suitable for my automation project and for 2-loop gravity, because when the project was started it was not yet clear whether that was finite. By 1985 however that issue was resolved by Goroff and Sagnotti with a dedicated C program. Of course a recalculation of something that complicated is desirable and was indeed done by Anton van der Ven with the first version of Form.

Automated calculations were at that moment also undertaken by various groups, most notably the GRACE group in Japan, CompHEP in Moscow and later the FeynArts/FeynCalc group in München. Hence I decided to first see how powerful Form really was by looking at what was at that moment the biggest algebraic calculation: the total crossection of e^+e^- into hadrons to order α_S^3 by Gorishnii, Kataev and Larin. I wanted to see what more could be done at the three loop level in QCD by making a Form version of the Mincer program for three-loop massless propagator diagrams.

Out of this came a collaboration with Sergei Larin which took many years and led to many papers, but most important, dictated many improvements and new features in Form.

The generic formula for one-loop massless two-point functions is:

$$\int \frac{d^D P}{(2\pi)^D} \frac{\mathcal{P}_n(P)}{P^{2\alpha}(P-Q)^{2\beta}} =$$
$$\frac{1}{(4\pi)^2}(Q^2)^{D/2-\alpha-\beta} \sum_{\sigma\geq 0}^{[n/2]} G(\alpha,\beta,n,\sigma) Q^{2\sigma} \left\{ \frac{1}{\sigma!} \left(\frac{\Box}{4} \right)^\sigma \mathcal{P}_n(P) \right\}_{P=Q},$$

in which

$$\mathcal{P}_n(P) = P_{\mu_1} P_{\mu_2} \cdots P_{\mu_n}.$$

D is the dimension of space-time and is also given by $D = 4 - 2\epsilon$, $\Box = \partial^2/\partial P_\mu \partial P_\mu$ and G can be expressed in terms of Γ-functions:

$$G(\alpha,\beta,n,\sigma) = (4\pi)^\epsilon \frac{\Gamma(\alpha+\beta-\sigma-D/2)\Gamma(D/2-\alpha+n-\sigma)\Gamma(D/2-\beta+\sigma)}{\Gamma(\alpha)\Gamma(\beta)\Gamma(D-\alpha-\beta+n)}.$$

The problem here is formed by the powers of the d'Alembertians, which can become rather high for high Mellin moments. The naïve algorithm makes that many terms get generated many times and have to be added.

With the addition of two new functions and two new commands this problem was solved and each different term is generated only once with the proper combinatorics factor.

In the following program we work out

$$\Box^{10}(P.p_1 P.p_2 P.p_3)^{10}$$

one thousand times. This gives us a good idea about the timings.

```
#define MAX "10"
Vector P,p1,p2,p3;
Tensor T,dd;
Off Statistics;
.global
#do i = 1,1000
#if ( `i' == 1000 )
  On Statistics;
```

```
    #endif
    L   F = P.p1^`MAX'*P.p2^`MAX'*P.p3^`MAX';
    ToTensor,nosquare,T,P;
    id  T(?a) = 2^`MAX'*distrib_(1,2*`MAX',dd,T,?a);
    ToVector,T,P;
    id  dd(?a) = dd_(?a);
    .store
    #enddo

Time =       2.91 sec    Generated terms =        1701
               F         Terms in output =        1701
                         Bytes used      =      157108
    .end
  2.91 sec out of 2.93 sec
```

One can see that each time exactly the right number of terms is generated. This made the Mincer program much faster.

In the mid 90's I was contacted by Hans Kühn from Karlsruhe. The group there was planning big calculations for the LHC and figured that if a parallel version of Form could be made, it would greatly enhance the possibilities in this field. The question was whether they could get access to the Form sources and collaborate on such a project. The answer was "yes!"

At the time I had already some experience with parallelisation, because in 1991 I had been visiting Fermilab and they had a special computer there with 257 cores, called "the Fermilab machine". With the help of George Hockney, who had written part of the operating system in one week we managed to put together a primitive parallel version that worked much better than expected. Unfortunately access to that computer was rather limited. Hence that project was abandoned, but the knowledge gained made sure that all development after that had already parallelisation in mind. Therefore the Karlsruhe request was very welcome. Mostly only the rewriting of those parts of Form that involved the typical bottlenecks for parallel algebra programs was necessary.

Most of the work was done by Albert Retey (PhD student), Denny Fliegner (postdoc) and Hans Staudenmeier (professor) and after a few years and many frustrating hardware/software upgrades of the computer, they managed to get the system going. It goes by the name ParForm and has the various processors/computers communicate by means of MPI.

By the late 90's it was clear that parts of Form were not up to standard for very big calculations. Specially the compiler part and the variable management were only designed for a few hundred objects or lines of code. Hence this was reprogrammed from scratch. It also forced a few changes in the syntax based on some user suggestions, making the language more consistent. The better techniques made the system much easier to maintain and to expand with new types of capabilities, statements or functions. In addition there was a whole new type of variables that could give feedback from algebraic information to the compiler part. This allows for very meaningful flowcontrol. It is also possible for Form programs to write pieces of Form code that can be used at later stages in the same program. It was all released as version 3 in the year 2000.

The project in which all Mellin moments of the three loop splitting and coefficient function were determined required ways to deal with gigantic tables. If you have 3 Gbytes of tables and the compiler compiles 2 Mbytes of code per second, it is no fun waiting while your program is starting up. It led to the concept of the tablebase, a crossbreed between tables and databases. In no time York Schröder, implementing the Laporta algorithm, used this feature with tables that were at least 10 times the size of ours (Sven Moch, Andreas Vogt and me), showing that this feature was well appreciated.

In the 2000's most computers started having multiple cores which would share memory. In ParForm it is assumed that the computers are separate and have no direct access to each others memory. Of course having memory access makes it much faster to combine the information of various 'workers'. This led to the development of a multi-threaded version called tform, which is currently used in most big Form calculations. The model is rather simple and makes use of POSIX threads. With the use of some clever macro's not many changes had to be made to most of the sources. Nearly all of the important multithreading code is in a single file. It keeps the sources rather uniform and much easier to maintain.

As of yet there is no combination of tform and ParForm to run on multiple computers, each with multiple cores. It is unlikely that it will ever materialise.

Another original feature was developed for the programs that solved the millions of equations for the MZV datamine. The problem here is that at a given moment one has sums with many indices as in

```
H(3,2,1,5,4,3,4)
```

which has to be used in sum notation as well as in integral notation. The transformation can be programmed as

```
repeat id H(?a,n?{>1},?b) = H(?a,0,n-1,?b);
```

which gives

```
H(0,0,1,0,1,1,0,0,0,0,1,0,0,0,1,0,0,1,0,0,0,1)
```

but this is very slow because the pattern matching had to be done many times. In addition it was a good idea to interpret the integral notation as a binary number in which case there would be only a single index during the solving of the equations. This can be done by

```
repeat id H(x?,n?,?a) = H(2*x+n,?a);
```

but again one needs to execute many statements. and the same holds for the inverse transformations.

The transform statement with its many substatements is made for manipulating such fields of arguments. The above two transformations can even be executed in two substatements of a single transform statement:

```
CFunction   H;
Symbols x,n;
Off Statistics;
.global
#do i = 1,1000000
#if ( `i' == 1000000 )
    On Statistics;
#endif
L   F = H(3,2,1,5,4,3,4);
repeat id H(?a,n?{>1},?b) = H(?a,0,n-1,?b);
repeat id H(x?,n?,?a) = H(2*x+n,?a);
#if ( `i' == 1000000 )
```

```
      Print;
    #endif
    .store
    #enddo

Time =       82.57 sec    Generated terms =           1
                F         Terms in output =           1
                          Bytes used      =          40

   F =
      H(723089);

    Off Statistics;
    #do i = 1,1000000
    #if ( `i' == 1000000 )
```

```
      On Statistics;
    #endif
    L   G = H(3,2,1,5,4,3,4);
    Transform H,ToIntegralnotation(1,last),
                Encode(1,last):base=2;
    #if ( `i' == 1000000 )
      Print;
    #endif
    .store
    #enddo

Time =      93.11 sec    Generated terms =          1
               G         Terms in output =          1
                         Bytes used      =         40
```

```
  G =
     H(723089);

   .end
 93.11 sec out of 93.14 sec
```

This made a big difference.

Another major improvement was a long wished extension to have Form optimise its output such that, if one passes from the algebraic phase of a calculation to the numerical phase, the resulting formula's take as few numerical operations as possible. It is very important for projects like the GRACE project in which one diagram can cause formula's that take many Mbytes, and there are many diagrams. This is something that can hurt twice: first the formula needs many operations for evaluation and second the compiler cannot optimise because the formula is too big. In many cases only -O0 could be used. Making use of Monte Carlo Tree Search techniques a first version was produced together with Jan Kuipers(postdoc), Jaap van den Herik(CS) and Aske Plaat(CS) which gave spectacular improvements. This was later taken further by Ben Ruijl as part of his thesis work. To my knowledge it is still unique in its generic efficiency. This is all part of version 4.

Of course over the years many more additions were made. The pattern matching was made more and more powerful and there are rather interesting functions that can help making many processes far more efficient etc.

A more recent addition that is present only as some form of beta version is the diagram generator of Toshiaki Kaneko. This is originally the diagram generator of the GRACE system, but it has been reprogrammed in C++ as a stand-alone program or as a library. As such it has been attached to Form. Internally there is much more information about the diagrams than Form currently uses but that can easily be extracted by new Form commands if the need arises because Kaneko has documented the generator very well. At the moment the main question is still whether extra graphical instructions should be implemented. One would be canonicalisation of diagrams/topologies. Kaneko was going to visit Nikhef to work on that when Covid destroyed all our travel plans. We hope it can still be done in the future. It would for instance be very useful for a faster implementation of the R* operation.

The most recent extension is the addition of an arbitrary precision floating point capacity, not so much for doing big numerical calculations but more for calculations in the style of the Francis Brown paper about motivic multiple zeta values. This allows the construction of MZV relations to much higher weights than those available in the datamine. Example at weight 15:

According to the Francis Brown paper one can find a basis.

$$\begin{gathered}\zeta_2, \zeta_3, \zeta_5, \zeta_7, \zeta_{5,3}, \zeta_9, \\ \zeta_{7,3}\zeta_{11}, \zeta_{9,3}, \zeta_{6,4,1,1}, \\ \zeta_{13}, \zeta_{7,3,3}, \zeta_{5,5,3}, \zeta_{11,3}, \\ \zeta_{9,5}, \zeta_{5,3,3,3}, \\ \zeta_{15}, \zeta_{9,3,3}, \zeta_{7,3,5}, \zeta_{6,4,3,1,1}\end{gathered}$$

in which products of lower weight basis elements belong to the basis provided the combined weight is exactly 15.

One can find the coefficients for each of the basis elements by algebraic means, with the exception of the depth-1 element, in this case: ζ_{15}. This means that one obtains for instance:

$$\begin{aligned}\zeta_{4,2,1,1,2,4,1} &= 101074859/2702700\zeta_2^6\zeta_3 + 715422133/9702000\zeta_2^5\zeta_5 - 2704607/201600\zeta_2^4\zeta_7 \\ &\quad +538/15\zeta_2^3\zeta_3^3 - 21012281/26460\zeta_2^3\zeta_9 + 101/10\zeta_2^2\zeta_3^2\zeta_5 \\ &\quad -1613/60\zeta_2^2\zeta_3\zeta_{5,3} + 483517807/201600\zeta_2^2\zeta_{11} - 1676/75\zeta_2^2\zeta_{5,3,3} \\ &\quad -3851/8\zeta_2\zeta_3^2\zeta_7 - 52109/672\zeta_2\zeta_3\zeta_5^2 - 33345/224\zeta_2\zeta_3\zeta_{7,3} \\ &\quad -308/5\zeta_2\zeta_5\zeta_{5,3} + 1999971329/115200\zeta_2\zeta_{13} - 689/96\zeta_2\zeta_{7,3,3} \\ &\quad +1787/1200\zeta_2\zeta_{5,5,3} - 809/120\zeta_3^5 + 59279/144\zeta_3^2\zeta_9 \\ &\quad -6199/18\zeta_3\zeta_5\zeta_7 + 99041/432\zeta_3\zeta_{9,3} + 319/6\zeta_3\zeta_{6,4,1,1} \\ &\quad -206405/1008\zeta_5^3 - 34717/4032\zeta_5\zeta_{7,3} + 77047/480\zeta_7\zeta_{5,3} \\ &\quad +253/27\zeta_{9,3,3} + 49885/4032\zeta_{7,3,5} - 20/3\zeta_{6,4,3,1,1} \\ &\quad +c_1\zeta_{15};\end{aligned}$$

and the coefficient c_1 remains unknown. So how do we obtain it? We need a program that can evaluate the MZV's in the equation over the floating point numbers to a given precision. A good algorithm has been provided by Bradley, Borwein, Broadhurst and Lisonek. In that case we can obtain the floating point value of c_1 and if the precision is sufficient it allows us to reconstruct the rational value of c_1:

```
#startfloat 150
```

```
 L   F1 = mzv_(4,2,1,1,2,4,1);
 L   F2 = 101074859/2702700*mzv_(2)^6*mzv_(3)
     +715422133/9702000*mzv_(2)^5*mzv_(5)
              .
              .
     +49885/4032*mzv_(7,3,5)
     -20/3*mzv_(6,4,3,1,1);
 L   F3 =  mzv_(15);
 Evaluate mzv_;
 Print +f;
 .sort

F1 =
   1.38348122320335752921181736956954059486 2e-07;

F2 =
   3.21389798241888199375641773362430087142 1e+04;

F3 =
   1.00003058823630702049355172851064506256 8e+00;

 Drop;
 L   F = (F1-F2)/F3;
 Print +f;
 .sort
```

```
F =
    - 3.213799677941054894179894179894179894197e+04;

 ToRational;
 Print +f;
 .end

F =
    - 62198593447/1935360;

0.04 sec out of 0.04 sec
```

which is identical to the coefficient we can find in the datamine.

To find bases at higher weight one needs relations for lower weight MZV's and hence one can set up a whole bootstrapping procedure to construct the basis for any weight, provided of course that one has sufficient computer resources. And having the bases, one can produce the decomposition for any MZV of that weight or lower. Oliver Schnetz has created programs (not in Form) to do this for MZV's to weight 34, Euler sums to weight 22 and also for sums

with an alphabet involving 4-th and sixth roots of unity. The hope is that with Form we can go even further, because the 'interesting' values are weight 39 for the MZV's and weight 27 for the Euler sums. The last one would answer some intriguing questions about pushdowns and doubling formulas (see the datamine paper), but it seems a very tough challenge.

This should all be part of version 5, if it will ever be released.

What could the future bring?

At the moment I am 72 years old. This means that it is realistic to assume that my capabilities might decline over the coming years, independent of whether this is caused by a tram. If it is deemed necessary that Form will be maintained and extended when new types of calculations ask for new capabilities, somebody, or preferably more than one person, should take it over. The problem with this is that such a person(s) should either have a tenure position, or a proper career perspective to eventually obtain such a position. Here we are talking about things that are outside my power. This has to come from users pressuring their administrations.

When you need a new computer, it is mostly a one time expense. A postdoc or a graduate student is a commitment for just a few years. These are much easier to obtain than tenure positions which are long term commitments.

Such jobs belong in physics. We want the director of for instance Nikhef to be a physicist rather than a professional manager or a politician. Similarly the developer(s) of Form should be in physics, capable of working on calculations that move boundaries and from that learn how to improve Form to make it more powerful in that direction.

Currently Ben Ruijl is working on making a newer version of Form, called reForm. It is to be programmed in the language Rust, which is designed to severely cut down on programming errors. As of yet it is however inconclusive whether it can ever match Form in speed. In addition Ben does not have a tenure position and had to put the whole project on ice for now.

The problem with multidisciplinary proposals and skills (in this case physics, computer science and mathematics) is that they will be judged separately by physicists, computer scientists and/or mathematicians with the result that each might say: "it is good, but not super top in my field." ignoring the other fields and not realising that the combination is unique and very innovative.

One other thing I have run into was that our granting agency told me after a failed proposal: "look for money from the subgroup computational physics". This is however a much smaller group which does not want to share their limited resources with yet another person (= intruder).

Hence this problem needs strong support from very high up in a laboratory, physics department, science faculty or university. The generic rule seems to be: high administrators talk a lot about multidisciplinarity, but once specific projects get looked at at a lower level their chances diminish rapidly.

Currently the Form sources reside in the github. Most of the infrastructure there has been set up by Takahiro Ueda. He also answers many questions that are raised by users. Sometimes I also answer a few questions, and there are also other users who join in the discussions. Also Takhiro Ueda does not have a tenure position yet.

As far as my personal involvement concerns: I am not a born organiser. It seems to me that the users should determine how the story continues. I am slowly withdrawing from maintenance and development. What I am still willing to do is help people to familiarise themselves with the sources of Form. This can be done in many different ways, provided I do not have to organise it.