PROCEEDINGS OF SCIENCE

New features of FORM and future plans

J.A.M.Vermaseren[*†]
Nikhef Amsterdam
E-mail: t68@nikhef.nl

In this talk development criteria of FORM are discussed and clarified with examples. This is done in the form of an evolution path. It ends with the discussion of potential new features that should be implemented over the coming years.

13th International Symposium on Radiative Corrections
24-29 September, 2017
St. Gilgen, Austria

[*]Speaker.
[†]With help from Ben Ruijl and Takahiro Ueda.

PoS(RADCOR2017)022

1. Introduction

To determine a direction we need to have a look over a period. Hence in this talk I will start with looking at the development of FORM in the past and the present before starting to speculate about the future. Even though we may not be able to predict the future accurately, it will give a feeling of what to expect. The users can however influence the future by coming up with useful suggestions of their own.

2. The past

FORM started as a successor of SCHOONSCHIP [1]. a symbolic system that M. Veltman wrote in a period that people worried about tree level reactions or about divergences of one-loop integrals with no more than 4 legs. Typical SCHOONSCHIP programs looked like:

- Type in/punch diagrams (not very many).
- Substitute Feynman rules.
- Sort out index contractions and traces.
- Make a number of (by current standards simple) substitutions.
- Print output or punch to cards.

In 1984 it was already clear that for automated one-loop calculations with up to 6 legs much more would be needed. In addition portability was going to be important. This led to FORM, designed by reverse engineering much of the functionality of SCHOONSCHIP. It was first programmed in Fortran77 and then reprogrammed in C. Version 1 came out in 1989. In the same year the FF program, a more generic version of the program FORMF by M. Veltman, based on [2], was completed by Geert Jan van Oldenborgh [3, 4]. It could evaluate one-loop integrals numerically.

In FORM the strong points of SCHOONSCHIP were taken over and a number of weak points were either omitted or replaced. The internal structure however was completely different, most notably the memory management, the sorting and the representation of numbers. Until now these have not needed any essential improvements, even though what was designed for O(1 Mbyte) is now used for O(Terabytes). Another early improvement was the way function arguments were treated. The larger variety of patterns in the substitutions gave greater flexibility. The family of if-statements was also an enormous improvement on the 'flags' and conditional actions in SCHOONSCHIP. And finally there was the beginning of a potentially very powerful preprocessor, inspired by the preprocessor of the C language.

Until this point the design was based on my experience with the use of SCHOONSCHIP and what I considered necessary for the future. Having version 1 and using it for calculating massless 3-loop propagators (Mincer [5, 6]) and in particular, trying to make those programs faster and faster, led to many new inventions in FORM. Also the color [7] program stimulated a number of new facilities, some of which may be rather unknown, because the current ways of computing diagrams does not involve them (yet?). A little example is the Heawood graph [8] in figure 1. It will show up when computing 7-loop propagators, or

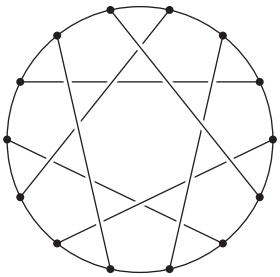

Figure 1: One redition of the the Heawood graph which has girth 6.

6-loop vertices. Maybe some of the younger people here will run into it during their career. Here we look at the color structure:

```
Tensor f(antisymmetric),ff(rcyclic);
AutoDeclare Index j;
CFunction num;
Symbol x;
Off Statistics;
.global
*
*   The Heawood graph has girth 6. Prove this.
*
L   g14 = f(j1,j2,j3)*f(j1,j4,j5)*f(j2,j6,j7)*f(j3,j8,j9)
        *f(j4,j10,j11)*f(j5,j12,j13)*f(j6,j14,j15)*f(j7,j16,j17)
        *f(j8,j18,j19)*f(j9,j20,j21)*f(j10,j21,j15)
        *f(j13,j19,j14)*f(j17,j11,j18)*f(j12,j16,j20);
*
*   Speed test. Do this 10000 times. Otherwise it is too fast.
*
Sum j1,...,j21;
Multiply sum_(x,1,10000,num(x))/10000;
Repeat;
    ReplaceLoop,f,arguments=3,loopsize<7,outfun=ff;
Endrepeat;
id  num(x?) = 1;
.sort
Renumber;
Print +f +s;
.end

 g14 =
     - f(N1_?,N2_?,N3_?)*f(N1_?,N4_?,N5_?)*
       ff(N2_?,N6_?,N7_?,N4_?,N8_?,N9_?)*
       ff(N3_?,N7_?,N9_?,N5_?,N6_?,N8_?);

0.39 sec out of 0.39 sec
```

The program selects the smallest loop it can find in the diagram and 'takes it out'. It can find two six-point functions. Doing this with the pattern matcher is considerably more work, both in programming effort and in CPU time.

```
Tensor f(antisymmetric),ff(rcyclic);
AutoDeclare Index j;
CFunction num;
Symbol x;
Off Statistics;
.global
*
L   g14 = f(j1,j2,j3)*f(j1,j4,j5)*f(j2,j6,j7)*f(j3,j8,j9)
        *f(j4,j10,j11)*f(j5,j12,j13)*f(j6,j14,j15)*f(j7,j16,j17)
        *f(j8,j18,j19)*f(j9,j20,j21)*f(j10,j21,j15)
        *f(j13,j19,j14)*f(j17,j11,j18)*f(j12,j16,j20);
*
*   Speed tests. Do this 10000 times. Otherwise it is too fast.
*
Sum j1,...,j21;
Multiply sum_(x,1,10000,num(x))/10000;
repeat;
id,once,ifmatch->endloop,f(j1?,j2?,j11?)*f(j2?,j1?,j12?)
     = ff(j11,j12);
id,once,ifmatch->endloop,f(j1?,j2?,j11?)*f(j2?,j3?,j12?)*
       f(j3?,j1?,j13?) = ff(j11,j12,j13);
id,once,ifmatch->endloop,f(j1?,j2?,j11?)*f(j2?,j3?,j12?)*
       f(j3?,j4?,j13?)*f(j4?,j1?,j14?) = ff(j11,j12,j13,j14);
id,once,ifmatch->endloop,f(j1?,j2?,j11?)*f(j2?,j3?,j12?)*
       f(j3?,j4?,j13?)*f(j4?,j5?,j14?)*f(j5?,j1?,j15?)
     = ff(j11,j12,j13,j14,j15);
id,once,ifmatch->endloop,f(j1?,j2?,j11?)*f(j2?,j3?,j12?)*
       f(j3?,j4?,j13?)*f(j4?,j5?,j14?)*f(j5?,j6?,j15?)*
       f(j6?,j1?,j16?) = ff(j11,j12,j13,j14,j15,j16);
Label endloop;
endrepeat;
id  num(x?) = 1;
.sort
Renumber;
Print +f +s;
.end

 g14 =
     - f(N1_?,N2_?,N3_?)*f(N1_?,N4_?,N5_?)*
       ff(N2_?,N6_?,N7_?,N4_?,N8_?,N9_?)*
       ff(N3_?,N7_?,N9_?,N5_?,N6_?,N8_?);

15.52 sec out of 15.52 sec
```

In version 3 [9] a qualitative jump was made with the introduction of the $ variables.

This gives a new level of control. The ensum program (calculating all moments of DIS splitting [10] and structure functions [11]) led to the table bases [12] which is a convenient way to store very large tables without the need to read complete tables in each program that uses them. The transform family [13] of statements to quickly manipulate complicated sequences of function arguments was invented to construct the MZV data mine [14].

3. The present

The programs that have influenced recent developments very much are the FORCER program [15] and the Rstar program [16]. This comes as no surprise, because they have been some of the major achievements of the HEPGAME project (ERC advanced grant 320651). Other major results have been the use of these programs for physics calculations as explained in the talks by Ben Ruijl, Giulio Falcioni and Andreas Vogt in these proceedings. And there are of course many calculations by other groups. The continuous need for greater speed and convenience of programming often points directly at what type of commands are needed, although a certain level of abstraction is mandatory to make new features more generally applicable.

Let us take an example. In the parametric reduction approach of the FORCER program internal multi-loop propagators are treated as insertions and hence represented by a single line (with a denominator that has a non-integer power). How to combine all the diagrams that contribute to such a propagator?

In the past, with the Mincer program [6] which only went to three loops, one could do this easily with pattern matching trying all possible propagator topologies. For four loops or more this becomes too slow. Hence another method is called for. The method we came up with is:

1. Select a representative for a one-loop propagator. A representative is a single diagram that occurs in this propagator. For the ghost and the quark propagators this is trivial, since there is only a single diagram. For the gluon we select the diagram with the ghost loop (not forgetting the minus sign).

2. In the propagators we indicate the number of loops with an extra parameter. Adjacent loop representatives are combined and their number of loops is the sum of those parameters. This means that the representative of a three-loop gluon propagator is a chain of three one-loop diagrams, each with a ghost loop as in figure 2.

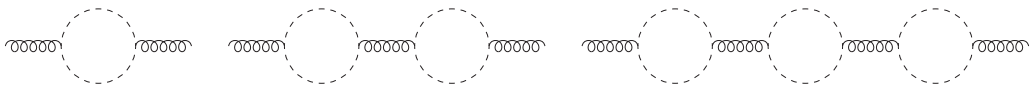

Figure 2: The chains of diagrams that represent 1-loop, 2-loop and 3-loop gluon propagators.

3. Next we make a copy of all remaining vertices into a function `acc`. In this function we remove all vertices that have an external line (figure 3).

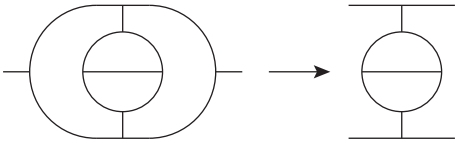

Figure 3: Removal of vertices with external lines.

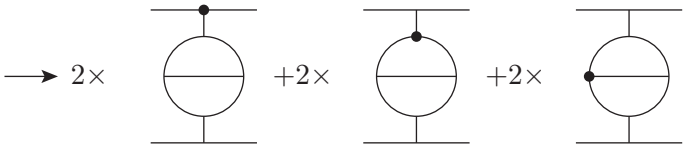

Figure 4: Selecting one vertex in all possible ways.

4. In the function `acc` we start selecting one vertex in all possible ways (see figure 4).

5. If this special vertex has more than two lines, it 'consumes' in all possible ways one of its neighbouring vertices, removing the connecting momentum. If the same momentum connects twice to the new vertex, it is removed as well (figure 5).

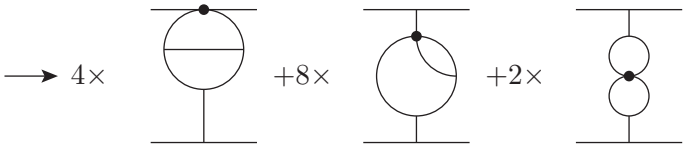

Figure 5: Special vertex absorbs in all possible way one adjacent vertex.

6. We keep doing this until either the super-vertex in one of the terms has two lines remaining in which case we can eliminate the whole diagram as it is part of a propagator, or we cannot remove any more lines. If all possibilities end in the last way we keep the diagram (figure 6).

After the code that introduces the representatives of the propagators the complete code, for any number of loops and any number of legs, looks like:

```
id  vx(?a,QQ?externals,?b) = 1;
Symmetrize vx;
id  vx(?a) = acc(vx(?a));
repeat id acc(x1?)*acc(x2?) = acc(x1*x2);
$x = 0;
Argument acc;
    id,all,vx(?a) = dV(?a);
EndArgument;
Repeat;
  Argument acc;
    id,all,dV(?a,p1?,?b)*vx(?c,p1?,?d) = dV(?a,?c,?b,?d);
```

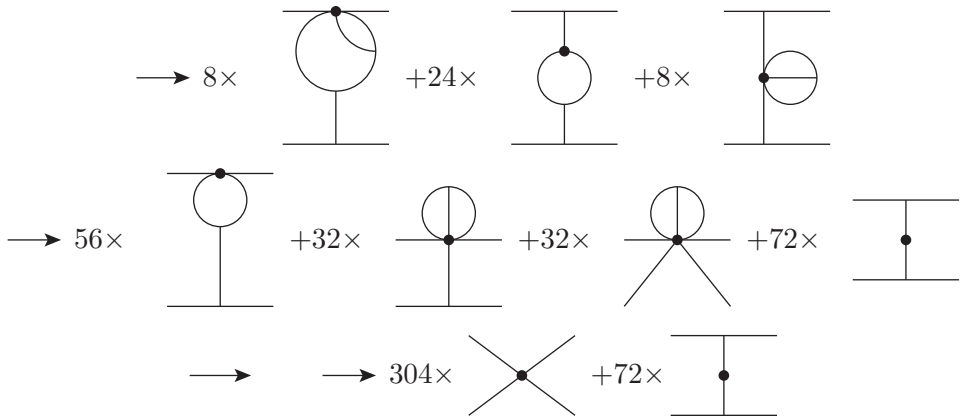

Figure 6: Continue absorbing adjacent vertices.

```
     Symmetrize dV;
    repeat id dV(?a,p1?,p1?,?b) = dV(?a,?b);
    if ( match(dV(p1?,p2?)) ) $x = 1;
  EndArgument;
  if ( $x == 1 ) Discard;
EndRepeat;
id  acc(x?) = 1;
```

That the id,all is indeed a good abstraction of the problem is shown by the possibility to also use it to generate all automorphisms of a given topology. Here it is shown for the three-loop propagator NO topology (figure 7).

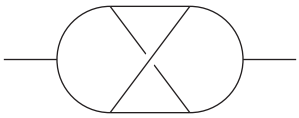

Figure 7: The NO topology.

```
CFunction v(s),auto;
Vector Q,p1,...,p8,QQ,q0,...,q8;
Set pp:<p1,-p1>,...,<p8,-p8>;
*
L F = v(-Q,p1,-p6)*v(-p1,p2,p7)*v(p3,-p2,p8)
     *v(-p3,p4,Q)*v(p5,-p4,-p7)*v(-p5,p6,-p8);
$x = term_;
id all,$x * replace_(Q,Q?{Q,-Q},<p1,p1?>,...,<p8,p8?>)
       = $x * auto(Q,QQ,<p1,q1>,...,<p8,q8>);
*
Bracket v;
Print +f +s;
.sort
```

```
F =

    + v(-Q,-p6,p1)*v(-p1,p2,p7)*v(-p2,p3,p8)*v(-p3,Q,p4)*v(-p4,-p7,p5)*v(
   -p5,-p8,p6) * (
       + auto(Q,QQ,p1,q1,p2,q2,p3,q3,p4,q4,p5,q5,p6,q6,p7,q7,p8,q8)
       + auto(Q,QQ,p1,q1,p7,q2,-p4,q3,-p3,q4,p8,q5,p6,q6,p2,q7,p5,q8)
       + auto(Q,QQ,-p6,q1,-p5,q2,-p4,q3,-p3,q4,-p2,q5,-p1,q6,-p8,q7,-p7,q8)
       + auto(Q,QQ,-p6,q1,-p8,q2,p3,q3,p4,q4,-p7,q5,-p1,q6,-p5,q7,-p2,q8)
       + auto(-Q,QQ,p4,q1,p5,q2,p6,q3,p1,q4,p2,q5,p3,q6,-p7,q7,-p8,q8)
       + auto(-Q,QQ,p4,q1,-p7,q2,-p1,q3,-p6,q4,-p8,q5,p3,q6,p5,q7,p2,q8)
       + auto(-Q,QQ,-p3,q1,p8,q2,p6,q3,p1,q4,p7,q5,-p4,q6,-p2,q7,-p5,q8)
       + auto(-Q,QQ,-p3,q1,-p2,q2,-p1,q3,-p6,q4,-p5,q5,-p4,q6,p8,q7,p7,q8)
       );
```

The dollar variable makes a copy of the current term and then the id statement matches the term onto itself, but thanks to the replace function all variables have become wildcards and hence the statement generates all possible relabellings of the momenta. These relabellings are then stored in the function auto. The trick with the replace_ in the left hand side was invented by Takahiro Ueda. It did not work right away as intended and needed a considerable amount of debugging, because this potential use was never considered.

```
id  auto(?a) = replace_(?a)/8;
Print +f +s;
.end

F =
    + v(-QQ,-q6,q1)*v(-q1,q2,q7)*v(-q2,q3,q8)*v(-q3,QQ,q4)*v(-q4,-q7,q5)*v(
   -q5,-q8,q6)
   ;
```

In the end we have the program execute those symmetry operations and we see that we get the input back, except for that now we have vectors q instead of vectors p (and QQ instead of Q).

The function `dd_` in FORM is the generalized Kronecker delta. In the case of 10 (vector or index) arguments it generates $1 \times 3 \times 5 \times 7 \times 9$ terms, unless there are identical arguments in which case it manages to generate exactly the (smaller number of) terms with a combinatorics factor. This can speed up calculations by a significant factor.

During the development of the Rstar [16] program it seemed useful to have a similar function for objects with three arguments and without a restriction to vectors or indices. Once such a request is made, it is time to start thinking where this will end. It is better to solve this problem in greater generality. From this came the partitions_ function. It is a generalization of both the dd_ and the distrib_ functions, with the assumption that all functions involved are symmetric. The combinatorics is of course even more complicated than in the case of the dd_ and distrib_ functions, but Ben Ruijl managed to work it out. Here is an example:

```
    Symbol x1,x2,x3;
    CFunction f,g,h;
    Format nospaces;
    L   F = g(x1+1,x2,x3,x1+1,x2,x3,x2,x2,x1);
    L   G = g(x1+1,x2,x3,x1+1,x2,x3,x2,x2,x1);
    if ( expression(F) );
        id  g(?a) = partitions_(3,f,3,g,2,h,0,?a);
    else;
        id  g(?a) = partitions_(0,f,3,?a);
    endif;
    Print +f +s;
    .end

Time =       0.00 sec    Generated terms =         75
               F         Terms in output =         75
                         Bytes used      =      10256

Time =       0.00 sec    Generated terms =         16
               G         Terms in output =         16
                         Bytes used      =       2780

   F=
      +6*f(1+x1,1+x1,x1)*g(x2,x2)*h(x2,x2,x3,x3)
      +8*f(1+x1,1+x1,x1)*g(x2,x3)*h(x2,x2,x2,x3)
      +f(1+x1,1+x1,x1)*g(x3,x3)*h(x2,x2,x2,x2)
      +12*f(1+x1,1+x1,x2)*g(x1,x2)*h(x2,x2,x3,x3)
      +  .........
      ;

   G=
      +12*f(x1,x2,x2)*f(1+x1,x2,x2)*f(1+x1,x3,x3)
      +24*f(x1,x2,x2)*f(1+x1,x2,x3)^2
      +12*f(x1,x2,x2)*f(x2,x2,x3)*f(1+x1,1+x1,x3)
      +12*f(x1,x2,x2)*f(x2,x3,x3)*f(1+x1,1+x1,x2)
      +48*f(x1,x2,x3)*f(1+x1,x2,x2)*f(1+x1,x2,x3)
      +8*f(x1,x2,x3)*f(x2,x2,x2)*f(1+x1,1+x1,x3)
      +24*f(x1,x2,x3)*f(x2,x2,x3)*f(1+x1,1+x1,x2)
      +6*f(x1,x3,x3)*f(1+x1,x2,x2)^2
      +4*f(x1,x3,x3)*f(x2,x2,x2)*f(1+x1,1+x1,x2)
      +8*f(x2,x2,x2)*f(1+x1,x1,x2)*f(1+x1,x3,x3)
      +16*f(x2,x2,x2)*f(1+x1,x1,x3)*f(1+x1,x2,x3)
      +4*f(x2,x2,x2)*f(x2,x3,x3)*f(1+x1,1+x1,x1)
      +48*f(x2,x2,x3)*f(1+x1,x1,x2)*f(1+x1,x2,x3)
      +24*f(x2,x2,x3)*f(1+x1,x1,x3)*f(1+x1,x2,x2)
      +6*f(x2,x2,x3)^2*f(1+x1,1+x1,x1)
      +24*f(x2,x3,x3)*f(1+x1,x1,x2)*f(1+x1,x2,x2)
      ;
  0.00 sec out of 0.00 sec
```

4. The future

The above examples should give an impression of what some of the design criteria are. It is important that statements are as general as possible. In the ideal case the same statement can be used immediately for an unrelated problem or a different part of the same problem. Sometimes it is only anticipated that such re-use will occur, and the 'immediately' is replaced by 'some time in the future'.

Let us have a look at what is currently on the hitlist.

The major problem we (in the HEPGAME project) are experiencing at the moment is the lack of speed in the Rstar program. As seen in other talks in these proceedings (Ruijl and Falcioni) we recognize two types of R-star operations:

- Global R-star. This is rather fast but difficult from the viewpoint of renormalization and operator mixing.
- Local R-star. From the viewpoint of physics this is conceptually simpler, but rather slow.

The attack here is two-pronged: make the first method simpler and make the second method faster. It is always good strategy to develop both competing methods.

The 'making simpler' is not a FORM *problem, but the 'making faster' is.*

We need to know where the bottlenecks are. It turns out that one of the slowest parts is in topology recognition. The program gets a diagram, then substitutes only enough vertices to move external lines and to determine the necessary counterterms. Then it determines the topology of what remains and substitutes the remaining vertices. As it turns out, in this method the number of terms becomes already rather large when the topology has to be determined and the 'traditional method' of finding the topology involves much pattern recognition. Part of this pattern recognition is to determine the 'canonical' representation of the diagram, to make sure that equivalent diagrams have indeed exactly the same notation.

From the above it is clear that FORM will need some graph manipulation capabilities. Up to now, the procedure of calculating diagrams was

1. Prepare the appropriate QGRAF [17] files and run QGRAF to generate the diagrams.
2. Convert the QGRAF diagrams, write the diagrams with internal higher order propagators when possible (see example in the section about 'the present').
3. Determine their topology and fix the notation (momenta and indices).
4. Determine color factors.
5. Write the diagrams in a format that can be processed further.
6. Process the diagrams one by one, without the color factor. Multiply by the proper projectors.

7. Substitute the Feynman rules (make expansions if needed).

8. Call the appropriate program to reduce it to master integrals (like FORCER).

With Rstar the third step is eliminated and the seventh step step is broken up.

Observations:

- QGRAF does not give us much information about the topological properties of the diagrams.
- It also does not allow the use of its internal procedures.
- It is slower than the diagram generator of the GRACE system [18].

Solution: Get into a collaboration with the GRACE authors to have the necessary routines built into FORM. It will be an interesting challenge to come up with a proper syntax to manipulate diagrams and their topological properties.

Another recurring problem is the solving of large systems of linear equations. In the past several 'generic' methods were invented and implemented as procedures for FORM: the way to solve recursion relations in the case one could guess the solution space; the method to solve for the Multiple Zeta Values [14] where millions of equations that could not be held in memory needed solving, and more recently just ordinary Gaussian routines for simple parametric IBP relations. It would be nice to have some more standardized methods here that would be faster than external programs. An advantage over dedicated programs is the availability of many internal parts of FORM, in particular its memory management and potentially the tablebases.

Already version 4.1 of FORM [19] had capabilities to simplify polynomial expressions that are to be used for numerical evaluation. When these commands were applied to output of the GRACE system, it turned out that some knowledge of the polynomials allowed even better simplifications. But because of the built in optimizations, this particular 'application of knowledge' allowed only a limited number of operations. It was also rather slow. By adding new types of simplifications and applying new AI techniques we should be able to do better. This is under study, but one should not expect results in the immediate future.

5. Extra remarks

As one can see, the progress in FORM is driven by the need of its users, provided that the users make their needs known. It would be very constructive if more users could contribute as well, either by suggesting very useful commands/functions, or by implementing some of those themselves. The main requirements of new features are to be fast, to have an external syntax that blends in rather well, and a clear addition to the manual. A separate paper with good examples of their use would also be helpful. Whenever possible help can be provided, either to discuss the exact form of the commands or to explain the internal workings of FORM to the point that new features can be added without destroying any of the old ones. If it needs reprogramming of some of the existing routines, also that can be

considered. It will of course be important that the code for the new features is fast. In exchange the routines for the new features can use the internal routines of FORM, many of which are extremely fast and compact, like the memory management, or powerful, like the pattern matcher, etc.

Finally: the community should consider whether further development of FORM is desired and how it should be achieved. Such development will require at least one or two tenured positions somewhere.

This work was paid for by the ERC advanced grant 320651, HEPGAME. Figures were made with axodraw version 2 [20].

References

[1] A program by M. Veltman. See H. Strubbe, Comp. Phys. Commun. **8** (1974) 1.

[2] G. 't Hooft and M. Veltman, Nucl. Phys. B153, 365 (1979). G. Passarino and M. Veltman, Nucl. Phys. B160, 151 (1979).

[3] G. J. van Oldenborgh and J. A. M. Vermaseren, Z. Phys. C **46** (1990) 425.

[4] G. J. van Oldenborgh, Comput. Phys. Commun. **66** (1991) 1.

[5] S. G. Gorishnii, S. A. Larin, L. R. Surguladze, and F. V. Tkachev, Comp. Phys. Comm. **55** (1989) 381

[6] S. A. Larin, F. V. Tkachev, and J. A. M. Vermaseren, NIKHEF-H-91-18

[7] T. van Ritbergen, A. N. Schellekens and J. A. M. Vermaseren, Int. J. Mod. Phys, **A14** (1999) 41.

[8] P. J. Heawood, (1890), Quarterly J. Math. Oxford Ser. **24** (1890) 32239.

[9] J. A. M. Vermaseren, math-ph/0010025.

[10] S. Moch, J. A. M. Vermaseren, A.ZVogt, Nucl.Phys. **B688** (2004) 101-134. A. Vogt, S. Moch, J. A. M. Vermaseren, Nucl.Phys. **B691** (2004) 129-181.

[11] J. A. M. Vermaseren, A. Vogt, S. Moch, Nucl.Phys. **B724** (2005) 3-182.

[12] J. A. M. Vermaseren, Nucl.Phys.Proc.Suppl. **116** (2003) 343-347.

[13] J. A. M. Vermaseren, Nucl. Phys. Proc. Suppl. **205-206** (2010) 104.

[14] J. Blumlein, D. J. Broadhurst and J. A. M. Vermaseren, Comput. Phys. Commun. **181** (2010) 582.

[15] B. Ruijl, T. Ueda, J. A. M. Vermaseren, arXiv:1704.06650 [hep-ph].

[16] Franz Herzog, Ben Ruijl, JHEP **1705** (2017) 037.

[17] P. Nogueira, J. Comput. Phys. **105** (1993) 279

[18] F. Yuasa *et al.*, Prog. Theor. Phys. Suppl. **138** (2000) 18.

[19] J. Kuipers, A. Plaat, J. A. M. Vermaseren, H. J. van den Herik, Comput.Phys.Commun. **184** (2013) 2391-2395. 27. J. Kuipers, T. Ueda, J. A. M. Vermaseren, Comput.Phys.Commun. **189** (2015) 1-19.

[20] John C. Collins, J. A. M. Vermaseren, arXiv:1606.01177 [cs.OH].