THREE-A

Here we will look at a completely new facility in Form. The idea is to simplify output expressions. We define simplification as writing the expression with as small a number of numerical operations as possible. This is an old and not very easy problem. Of course we can take the approach that this has been studied thoroughly by compiler builders. There is however a difference between what a compiler may do and what we may do: For a compiler $(a+b)+c$ is not the same as $a+(b+c)$ due to numerical precision problems. Also, using a high level of optimization with the compiler may need very much time, and on some of the bigger routines I had it even crash by lack of address space.

One of our test cases is an expression for a one loop four-point Feynman diagram inside a $2 \rightarrow 4$ reaction, multiplied by the conjugate of a tree graph. This diagram contains 639727 terms with about 4.8×10^6 arithmetic operations and it definitely is not the worst. The file is anywhere from 25 to 30 Mbytes depending on whether we can use tabulator characters. Our task is now to do better than this, because if we have to do more than 200,000 diagrams the eventual Monte Carlo program becomes a bit unwieldy.

Let us take this formula with its 639727 terms and see how much we can squeeze it. We have put the formula in a file named testx3.sav and we can read it with the program

```
    Load testx3.sav;
 Sigma loaded
    Local test = Sigma;
    .sort

Time =       0.66 sec    Generated terms =     639727
            test         Terms in output =     639727
                         Bytes used      =   27495480
    Delete storage;
    .end

Time =       0.92 sec    Generated terms =     639727
            test         Terms in output =     639727
                         Bytes used      =   27495480
```

When we print the output it looks like

```
  Delete storage;
  Format 00;
  Bracket x1,x3,x4,xlevi,zk;
  Print +f;
  .end

Time =       1.40 sec    Generated terms =     639727
            test         Terms in output =     639727
                         Bytes used      =   22747648

   test=
      +xlevi*(-24*amel2*amdq2^2*es1235*xcp5-24*amel2*amdq2^2*es1235
         *xcp1+48*amel2*amdq2^2*es1234*xcp5+72*amel2*amdq2^2*es1234
         *xcp1+48*amel2*amuq2*amdq2*es1235*xcp5+48*amel2*amuq2*
         amdq2*es1235*xcp1-48*amel2*amuq2*amdq2*es1234*xcp6-48*
         amel2*amuq2*amdq2*es1234*xcp5-24*amel2*amuq2^2*es1235*xcp5
         -24*amel2*amuq2^2*es1235*xcp1+48*amel2*amuq2^2*es1234*xcp6
```

```
     -72*amel2*amuq2^2*es1234*xcp1+96*amel2*ammu2*amdq2*es2345*
     xcp7+192*amel2*ammu2*amdq2*es2345*xcp5+48*amel2*ammu2*

   and more than 400000 extra lines.
```

The format statement is a basically a dummy statement because O0 is the default. The output is bracketted in the Feynman parameters, xlevi (which indicates whether there is a Levi-Civita tensor and hence the terms are imaginary), and the parameter zk which indicates whether there are powers of Q^2 in the answer. Each two powers of zk indicate one power of Q^2. The denominator with its $Q.Q$ and its Feynman parameters is an overal factor and taken for granted. The x_1, x_3, x_4 are Feynman parameters and we want to have the expressions for the 47 different brackets we have in this expression. We want to use a simultaneous optimization of these 47 expressions, which is done by using the bracket statement.

Now we switch to a higher format level. This may make the program crash, because it needs a rather big amount of workspace and it does not warn about that (still needs some attention). Hence we use a special file that defines a number of setup parameters to tune FORM to the computer at hand and to the problem:

```
LargeSize 1G
SmallSize 200M
ScratchSize 500M
TermsInSmall 1M
MaxTermSize 50K
WorkSpace 40M
```

The workspace of 40 Mbytes should be more than enough. We have now

```
    Delete storage;
    Format O1;
    Bracket x1,x3,x4,xlevi,zk;
    Print +f;
    .end

Time =       1.40 sec    Generated terms =     639727
            test         Terms in output =     639727
                         Bytes used      =   22747648
      Z1_=5*xnlb;
      Z2_=-6+Z1_;
      Z3_=xnlb*Z2_;
      Z4_=11+Z3_;
      Z5_=ammu2*Z4_;
      Z6_=3-xnlb;
      Z7_=xnlb*Z6_;
      Z8_=3*Z7_;
      Z9_=-8+Z8_;
```

```
Z10_=e3e1*Z9_;
   .
   .
   .   almost 350000 lines
   .
   .
Z50_=4*Z54_+Z50_;
Z50_=ammu2*Z50_;
Z30_=Z30_+Z37_+Z50_+Z48_;
Z30_=xcp5*Z30_;
Z2_=Z51_+Z44_+2*Z24_+2*Z43_+2*Z52_+Z47_+2*Z7_+4*Z49_+2*Z31_+2
*Z2_+4*Z14_+Z19_+2*Z8_+2*Z15_+Z28_+2*Z30_;
Z2_=24*Z2_;
test=xlevi*Z5_+x4*Z26_+x4*xlevi*Z226_+x4^2*Z167_+x4^2*xlevi*
Z53_+x4^3*Z166_+x4^3*xlevi*Z68_+x3*Z18_+x3*xlevi*Z35_+x3*x4*
Z82_+x3*x4*xlevi*Z126_+x3*x4^2*Z139_+x3*x4^2*xlevi*Z87_+x3^2*
Z96_+x3^2*xlevi*Z40_+x3^2*x4*Z25_+x3^2*x4*xlevi*Z23_+x3^3*
Z132_+x3^3*xlevi*Z111_+x1*Z9_+x1*xlevi*Z21_+x1*x4*Z69_+x1*x4*
```

```
    xlevi*Z20_+x1*x4^2*Z57_+x1*x4^2*xlevi*Z39_+x1*x3*Z17_+x1*x3*
    xlevi*Z11_+x1*x3*x4*Z29_+x1*x3*x4*xlevi*Z12_+x1*x3^2*Z41_+x1*
    x3^2*xlevi*Z1_+x1^2*Z6_+x1^2*xlevi*Z10_+x1^2*x4*Z3_+x1^2*x4*
    xlevi*Z34_+x1^2*x3*Z16_+x1^2*x3*xlevi*Z27_+x1^3*Z32_+x1^3*
    xlevi*Z4_+zk^2*Z42_+zk^2*xlevi*Z13_+zk^2*x4*Z22_+zk^2*x4*
    xlevi*Z36_+zk^2*x3*Z33_+zk^2*x3*xlevi*Z38_+zk^2*x1*Z45_+zk^2*
    x1*xlevi*Z46_+Z2_;

  20.09 sec out of 20.12 sec
```

If we have variables outside brackets, they are not part of the optimizations. This is needed like this if we need their coefficients. In that case we speak of ‘simultaneous optimization’. The variables Z_ are temporary variables that have been introduced by FORM for this calculation. They are called ‘extra symbols’ and we do have control over their representation:

```
    ExtraSymbols,vector,w;
    Delete storage;
    Format O1,stats=on;
    Bracket x1,x3,x4,xlevi,zk;
    Print +f;
    .end

Time =        1.43 sec    Generated terms =     639727
            test          Terms in output =     639727
                          Bytes used      =   22747648
      w(1)=5*xnlb;
      w(2)=-6+w(1);
      w(3)=xnlb*w(2);
      w(4)=11+w(3);
      w(5)=ammu2*w(4);
      w(6)=3-xnlb;
      w(7)=xnlb*w(6);
         .
```

```
           .
           .
      w(16)+x1^2*x3*xlevi*w(27)+x1^3*w(32)+x1^3*xlevi*w(4)+zk^2*
      w(42)+zk^2*xlevi*w(13)+zk^2*x4*w(22)+zk^2*x4*xlevi*w(36)+zk^2
      *x3*w(33)+zk^2*x3*xlevi*w(38)+zk^2*x1*w(45)+zk^2*x1*xlevi*
      w(46)+w(2);
*** STATS: original  33660P 4130889M 639679A : 4837888
*** STATS: optimized 2P 281714M 252897A : 534615

  20.00 sec out of 20.02 sec
```

We have turned on the statistics of the optimization and we see that in the input there were 33660 exponent calls of at least a third power (a square counts as a multiplication), 4130889 multiplications and 639679 additions (subtractions counted as additions). In the output these numbers are 2, 281714 and 252897 respectively. This is an improvement of almost an order of magnitude. Actually the compiler does not do that well.

In O2 it tries for more complicated common subexpressions. This is called greedy optimization. This is a search that is quadratic in the sizes of the subexpressions and also has to traverse the system more than once. As a consequence it is much slower.

When we apply this to our big formula the result is

```
    ExtraSymbols,vector,w;
    Delete storage;
    Format O2,stats=on;
    Bracket x1,x3,x4,xlevi,zk;
    Print +f;
    .end
```

```
Time =       1.40 sec    Generated terms =     639727
            test         Terms in output =     639727
                         Bytes used      =   22747648
      w(1)=2*xnlb;
      w(2)=w(1)+3;
      w(3)=2*amel2;
      w(4)=w(2)*w(3);
```

```
         .

         .

         .
      x1^2*w(21)+x1^2*xlevi*w(6)+x1^2*x4*w(7)+x1^2*x4*xlevi*w(25)+
      x1^2*x3*w(22)+x1^2*x3*xlevi*w(16)+x1^3*w(30)+x1^3*xlevi*w(23)
      +zk^2*w(33)+zk^2*xlevi*w(31)+zk^2*x4*w(41)+zk^2*x4*xlevi*
      w(45)+zk^2*x3*w(43)+zk^2*x3*xlevi*w(49)+zk^2*x1*w(26)+zk^2*x1
      *xlevi*w(50)+w(1);
*** STATS: original  33660P 4130889M 639679A : 4837888
*** STATS: optimized 2P 186598M 220298A : 406900

  3828.65 sec out of 3838.53 sec
```

and the number of operations has gone down from 534615 to 406900. But the execution time has shot up enormously. This may be worthwhile, but only if the function has to be evaluated very many times.

With the O3 option we use a method that changes the order of variables in the multivariate Hornerscheme that we use. Because there are N! orderings when there are N variables we cannot try all orderings and we use a Monte Carlo technique that comes from game theory, called MCTS (Monte Carlo tree search). Because it is a Monte Carlo technique the outcome depends on the random number seed (and various parameters that the use can set).

```
#do i = 1,5
ExtraSymbols,vector,w;
#message mctsconstant = 0.‘i’
Format O3,mctsconstant=0.‘i’,hornerdirection=backward,
         method=csegreedy,mctsnumexpand=1000,stats=on;
#message CPU time till now: ‘time_’ sec.
Local test = Sigma;
B   x1,x3,x4,xlevi,zk;
.sort
#Optimize test
.store
#enddo
.end
```

The results are

```
*** STATS: optimized 2P 163477M 207304A : 370785
*** STATS: optimized 3P 175169M 209676A : 384851
*** STATS: optimized 2P 176962M 210156A : 387122
*** STATS: optimized 2P 172261M 203804A : 376069
*** STATS: optimized 3P 170929M 212534A : 383469
```

for ‘mctsconstant’ is 0.1, 0.2, 0.3, 0.4, and 0.5 respectively. The number of operations is bouncing around a bit and we do not see a pattern in which value of C_p is to be preferred. This also took much CPU time. The five determinations took a bit more than 9 hours on a slightly faster computer.

We have compared the current implementation with results from the literature and programs we had access to.

	7-4 resultant	7-5 resultant	7-6 resultant	HEP(σ)
Original	29163	142711	587880	47424
FORM O1	4968	20210	71262	6099
FORM O2	3969	16398	55685	4979
FORM O3	3015	11171	36146	3524
Maple	8607	36464	-	17889
Maple tryhard	6451	$\mathcal{O}(27000)$	-	5836
Mathematica	19093	94287	-	38102
HG + cse	4905	19148	65770	-
Haggies	7540	29125	-	13214

Number of operations after optimization by various programs. The number for the 7-5 resultant with 'Maple tryhard' is taken from Leiserson et al. For the 7-4 resultant they obtain 6707 operations, which must be due to a different way of counting. The same holds for the 7-6 resultant where they start with 601633 operations. The FORM O3 run used $C_p = 0.07$ and 10×400 tree expansions.

A better approach is to first use some 'domain specific' knowledge. In this case we have a procedure that tries what happens if we shift variable as with

$$\begin{aligned} p1 \cdot p_2 &\rightarrow y_1 \\ m^2 &\rightarrow y_2 \\ y_1 &\rightarrow y_1 + y_2/2 \end{aligned}$$

Often in terms of the new y_1 the expression may be shorter and by shifting we do not increase the number of variables. This way we can make shifts in subsystems of coupling constants, Feynman parameters, Levi-Civita tensors, gauge terms and dotproducts plus masses. The result is that the number of terms in the expression can be reduced to

```
Multiply replace_(xcp8,xcp8-xcp4*(-1)/(1));
   time = 1274.70: New number = 125497
Multiply replace_(xcp8,xcp8-xcp4*(2)/(1));
   time = 1275.58: New number = 125129
~~~xcp1 to xcp8
doshift(Sig) pass 2 at time = 1322.63 sec
doshift finished at time = 1393.71 sec
```

After this we can see what happens next in the optimizations.

```
.store
Format O1,stats=on;
Local test1 = test;
Bracket x1,x3,x4,xlevi,zk;
.sort
#optimize test1
.sort
#clearoptimize
.store
Format O2,stats=on;
Local test2 = test;
Bracket x1,x3,x4,xlevi,zk;
.sort
#optimize test2
.sort
#clearoptimize
.store
Format O3,hornerdirection=backward,mctsconstant=0.05,
          mctsnumexpand=1000,stats=on;
Local test3 = test;
Bracket x1,x3,x4,xlevi,zk;
.sort
#optimize test3
.sort
#clearoptimize
.end
```

which results in:

```
Time =    1405.12 sec    Generated terms =     125129
            Sig1         Terms in output =     125129
                         Bytes used      =    6789144

Time =    1409.03 sec    Generated terms =         48
            Sig1         Terms in output =         48
                         Bytes used      =       2324
*** STATS: original  1157P 793459M 125081A : 920854
*** STATS: optimized 2P 84402M 59353A : 143759
```

```
Time =    1409.18 sec    Generated terms =     125129
            Sig2         Terms in output =     125129
                         Bytes used      =    6789144

Time =    1597.21 sec    Generated terms =         48
            Sig2         Terms in output =         48
                         Bytes used      =       2324
*** STATS: original  1157P 793459M 125081A : 920854
*** STATS: optimized 2P 52250M 53857A : 106111

Time =    1597.36 sec    Generated terms =     125129
            Sig3         Terms in output =     125129
                         Bytes used      =    6789144

Time =    3682.29 sec    Generated terms =         48
            Sig3         Terms in output =         48
                         Bytes used      =       2324
```

```
*** STATS: original  1157P 793459M 125081A : 920854
*** STATS: optimized 1P 43142M 49629A : 92773
```

and we see that the number of operations has decreased dramatically, as well as the time for the optimization. Which optimization level is the best depends on the number of function evaluation you need. On the whole we have reduced the number of operations by about a factor 50. And of course we will gain time back by having much shorter compilation times.