Compactifying formulas with FORM

J. Vermaseren[*]
Nikhef
E-mail: t68@nikhef.nl

J. Kuipers
Nikhef, Google
E-mail: jkuipers@nikhef.nl

T. Ueda
Karlsruhe Institute of Technology
E-mail: takahiro.ueda@kit.edu

Starting with version 4.1 the symbolic manipulation program FORM has the capability to compactify output formula's to allow faster numerical evaluation. Here we discuss some of the methods used and show the results with a number of examples from automatically generated programs for one loop radiative correction calculations.

11th International Symposium on Radiative Corrections (Applications of Quantum Field Theory to Phenomenology)
22-27 September 2013
Lumley Castle Hotel, Durham, UK

[*]Speaker.

1. Introduction

The project for which FORM [1, 2] was originally designed was automated one loop radiative corrections. After a number of feasibility studies this project, named ESP (Experiment Simulation Program), was started in 1984. The first thing to construct was algebraic capability. It took however almost five years to complete the first version of this algebra program and in an attempt to show off its power I got ‘sidetracked’ into three loop QCD. I did however keep contact with the automated program scene which was coming up about then. This was mainly with the GRACE[3] group which started also in the 80’s. It kept me constantly aware of the needs in this field.

Unless very drastic shortcuts are taken, a big problem in automated loop calculations is the size of the expressions. And because the loop integrals are normally done numerically with the help of Monte Carlo techniques, such formulas have to be evaluated many times.

We see the following nasty development:

- Better understanding of nature needs the study of more complicated reactions.
- More particles in the final state asks for more diagrams and more complicated loops.
- More complicated loops give longer formulas.
- More particles give a more complicated phase space.
- More complicated phase space means that we need many more Monte Carlo points for any form of accuracy.

Conclusion: it is very important to squeeze the formulas to as short a representation as possible. This has several important consequences:

- Less computer time needed (or better accuracy with the same amount of resources).
- The compiler might be able to compile the formulas.
- The full compiled formula may fit inside the computer.

In this talk we will see this illustrated.

2. Horner schemes

The simplest technique to economize on the evaluation of a polynomial in a single variable is the Horner scheme[4]:

$$\begin{aligned} F(x) &= 2+3x+5x^2+7x^3+11x^4+13x^5 \\ &\to 2+x(3+x(5+x(7+x(11+13x)))) \end{aligned} \tag{2.1}$$

If we give the first representation to a non-optimizing compiler the most likely outcome will be that we need 5 additions and 5+8 multiplications. In the second representation even a poor compiler will need 5 additions and 5 multiplications.

Horner Order	Formula	Operations
x,y,z	$y+x(-3+5z+x(y(2z+y(z(-3+5z)))))$	8,5
x,z,y	$y+x(-3+5z+x(z(y(2-3y)+z(5y^2))))$	9,5
y,x,z	$x(-3+5z)+y(1+x^2(2z)+y(x^2(z(-3+5z))))$	11,5
y,z,x	$-3x+z(5x)+y(1+z(2x^2)+y(z(-3x^2+z(5x^2))))$	14,5
z,x,y	$y-3x+z(x(5+x(y(2-3y)))+z(x^2(5y^2)))$	11,5
z,y,x	$-3x+y+z(5x+y(2x^2+y(-3x^2))+z(y^2(5x^2)))$	14,5

Table 1: The various Horner schemes and their operational costs.

For multivariate polynomials the situation is more complicated. We can see the expression as

$$\begin{aligned} F(x,y,z) &= f_0(y,z)+f_1(y,z)x+f_2(y,z)x^2+f_3(y,z)x^3+\cdots \\ &\to f_0(y,z)+x(f_1(y,z)+x(f_2(y,z)+x(f_3(y,z)+\cdots))) \end{aligned} \quad (2.2)$$

and next make a similar Horner expansion of the f_i, each in terms of y and the coefficients $f_{i,j}(z)$, etc.

The problem with such expansions is that the result depends on the order in which we select the variables for the rewriting [5]. We can see that in the following example:

$$F(x,y,z) = y-3x+5xz+2x^2yz-3x^2y^2z+5x^2y^2z^2 \quad (2.3)$$

Direct evaluation of this polynomial takes 18 multiplications and 5 additions (18,5). Depending on the order of the Horner scheme we have the results of Tab. 1.

So, how do we know which scheme to use?

If the number of variables is rather limited (when $n!$ is not very large) we can try all orderings. For, say, 30 variables this is obviously not practical. In that case it is most common to use what is called occurrence ordering: the variables are ordered by the number of occurrences in the formula.

When we were programming this we decided to try a number of random orderings just to see how much better this occurrence ordering was. As an example we used a relatively simple GRACE formula. The result was that occurrence ordering was better than average, but by no means optimal. Hence the big question is how to obtain a (near) optimal ordering.

3. Monte Carlo tree search

The various multivariate Horner schemes can be represented as a tree in which the first time we have to select one out of n variables, the next time one out of $n-1$ variables, etc. In the end we have a complete ordering with a number that tells us the cost of the evaluation of the formula. This is very similar to games like for instance Go. The main difference is of course that in games we have usually an opponent, but if we consider the selection of one variable as two ply, one move followed by a move of the opponent, the simularity should be obvious.

In 2006 there was a breakthrough in game theory w.r.t. how to search through such trees[6, 7]. It is called Monte Carlo tree search or shortly MCTS. It is based on a weight formula that tells which branch in the tree to select. For equal weights a random number determines which branch to select. The tree is evaluated to the end (also in Go or Chess) and then an evaluation is made. For each branchpoint the average score and the number of visits in the branches are remembered.

The formula most commonly used is (UCT stands for Upper Confidence level for Trees):

$$UCT_i = \langle x_i \rangle + 2C_p \sqrt{\frac{2\log n}{n_i}} \tag{3.1}$$

The first term in the equation favours trying previously successful branches in the tree. This is called exploitation. The second term favours branches that have not been visited much before (if never, the term is even infinite). This is called exploration. The value of C_p determines the balance between the two.

This approach can be successful if positive outcomes are clustered in the tree.

In games this often works because a good move will usually leave many more favourable end-positions than a bad move. When the value of C_p is too small, we will only sample one seemingly good branch in the tree and eventually end up in a local optimum. When the value of C_p is too big, we will basically be sampling randomly and forget to pursue branches that seem promising.

The proper value of C_p is problem dependent. It usually requires some experimentation[8]. It would be a separate branch of investigation to see how its best value can be determined automatically.

4. Other improvements

The above selection of the Horner scheme would not be very spectacular if we would not apply some more techniques, like common subexpression elimination. Such subexpression eliminations come in varieties of which some take more time than others. Take for instance:

$$\begin{aligned} F(x,y,z) &= y + x(-3+5z+x(y(2z+y(z(-3+5z))))) \\ &= y + x(Z_1 + x(y(2z+y(zZ_1)))) \\ Z_1 &= -3+5z \end{aligned} \tag{4.1}$$

This insertion saves us one multiplication and one addition.

In FORM [9] we have two types of such insertions:

- Common subexpressions.
- Greedy optimizations.

The second type is far more sophisticated than the first type, but also needs much more CPU type (it uses basically a quadratic algorithm). When FORM works out a Horner scheme it always combines this with at least the first type. The second type is optional.

Using such insertions, one might accumulate a large number of intermediate variables. This is rather compiler unfriendly (if we want the compiler to also optimize). Hence FORM will try to reuse variables.

We will show some examples of the various optimization levels first. Without MCTS the Horner ordering is by ‘occurrence’ which means that the variables are ordered by the number of occurrences in the formula. Actually the program tries two orderings: occurrence and anti-occurrence.

O1 Occurrence+anti-occurrence followed by common subexpression elimination.

O2 Occurrence+anti-occurrence followed by common subexpression elimination. After that a 'greedy' optimization is used.

O3 MCTS with common subexpression elimination and greedy optimizations afterward.

Examples of the three basic levels of optimization:

```
    Symbols x,y,z;
    Local F = 6*y*z^2+3*y^3-3*x*z^2+6*x*y*z-3*x^2*z+6*x^2*y;
    Format O1,stats=on;
    Print;
    .end
      Z1_=y*z;
      Z2_= - z + 2*y;
      Z2_=x*Z2_;
      Z3_=z^2;
      Z1_=Z2_ - Z3_ + 2*Z1_;
      Z1_=x*Z1_;
      Z2_=y^2;
      Z2_=2*Z3_ + Z2_;
      Z2_=y*Z2_;
      Z1_=Z2_ + Z1_;
      F=3*Z1_;
*** STATS: original  1P 16M 5A : 23
*** STATS: optimized 0P 10M 5A : 15
```

```
    Symbols x,y,z;
    Local F = 6*y*z^2+3*y^3-3*x*z^2+6*x*y*z-3*x^2*z+6*x^2*y;
    Format O2,stats=on;
    Print;
    .end
      Z1_=z^2;
      Z2_=2*y;
      Z3_=z*Z2_;
      Z2_= - z + Z2_;
      Z2_=x*Z2_;
      Z2_=Z2_ - Z1_ + Z3_;
      Z2_=x*Z2_;
      Z3_=y^2;
      Z1_=2*Z1_ + Z3_;
      Z1_=y*Z1_;
      Z1_=Z1_ + Z2_;
      F=3*Z1_;
```

```
*** STATS: original  1P 16M 5A : 23
*** STATS: optimized 0P 9M 5A : 14

    Symbols x,y,z;
    Local F = 6*y*z^2+3*y^3-3*x*z^2+6*x*y*z-3*x^2*z+6*x^2*y;
    Format O3,stats=on;
    Print;
    .end
      Z1_=x + z;
      Z2_=2*y;
      Z3_=Z2_ - x;
      Z1_=z*Z3_*Z1_;
      Z3_=y^3;
      Z2_=x^2*Z2_;
      Z1_=Z1_ + Z3_ + Z2_;
      F=3*Z1_;
*** STATS: original  1P 16M 5A : 23
*** STATS: optimized 1P 6M 4A : 12
```

With simple expressions that have only a few variables the MCTS can use brute force and try all possibilities. When there are more variables this is of course not possible and it tries a default 1000 times, unless the user specifies a different number.

The variables `Z1_` etc. are so-called extra symbols, generated by FORM. It is possible to have control over their representation as in

```
    ExtraSymbols,array,w;
    Symbols x,y,z;
    Format FORTRAN;
    Format nospaces;
    Format O3,stats=on;
    Local F = 6*y*z^2+3*y^3-3*x*z^2+6*x*y*z-3*x^2*z+6*x^2*y;
    .sort
    #optimize F
    #write <f.f> "      REAL*8 FUNCTION F(x,y,z)"
    #write <f.f> "      REAL*8 x,y,z,w(`optimmaxvar_')"
    #write <f.f> "      %O"
    #write <f.f> "      F=%e",F
    #write <f.f> "      RETURN\n      END"
    .end
```

This results in a file f.f with the contents

```
      REAL*8 FUNCTION F(x,y,z)
      REAL*8 x,y,z,w(3)
```

	Operations	Time
Original	142711	0.00 sec
O1	20210	0.41 sec
O2	16398	5.37 sec
O3	11171	183.61 sec

Table 2: Results for a 12x12 resultant, using various optimization levels.

```
      w(1)=x+z
      w(2)=2*y
      w(3)=w(2)-x
      w(1)=z*w(3)*w(1)
      w(3)=y**3
      w(2)=x**2*w(2)
      w(1)=w(1)+w(3)+w(2)

      F=3*w(1)
      RETURN
      END
```

There are a few subtle points here that I cannot go into in such a short presentation. It is a good idea to read the manual.

5. Results

After the above toy examples we go to some serious stuff.

The first example, taken from ref.[10], is a 12x12 determinant with 14 variables. The expression contains 11380 terms. After some tests we decided to put the MCTS constant to 0.1. The results are shown in Tab. 2. In the literature[10] we found as best result 19148 operations obtained in 22 seconds on a probably slightly faster computer.

For a somewhat simpler determinant (11x11 with 2562 terms) we have some results for other systems, as shown in Tab. 3.

It should be mentioned that a Mathematica expert might do better than what we could get out of it. Because the runs were all on different computers we have omitted the CPU time needed.

Of course we are mainly interested in the running time of the compiled code. This is shown in Tab. 4.

FORM can also do simultaneous optimizations. If we take some objects outside brackets, only the contents of the brackets are optimized, but in a combined optimization:

```
    ExtraSymbols,array,w;
    Symbols x,y,z;
    Off Statistics;
    Format FORTRAN;
```

	Operations	Remarks
Original	29163	
FORM O1	4968	
FORM O2	3969	
FORM O3	3015	
Maple	8607	
Maple(Tryhard)	6451	Takes a very long time
Mathematica	19093	Can probably be better?
HG+cse[10]	4905	Best in literature
Haggies[11]	7540	

Table 3: Results for an 11x11 resultant, including other systems.

	Format O0	Format O1	Format O2	Format O3
Operations	587880	71262	55685	36146
FORM time	0.12	1.66	65.43	2398
gcc -O0 time	29.02	6.33	5.64	3.36
run	119.66	13.61	12.24	7.52
gcc -O1 time	3018.6	295.96	199.47	80.82
run	24.30	6.88	6.12	3.58
gcc -O2 time	3104.4	247.60	163.79	65.21
run	21.09	7.00	6.22	3.93
gcc -O3 time	3125.4	276.77	179.24	71.02
run	21.02	6.95	6.19	3.93

Table 4: FORM run time, compilation times and the time to evaluate the compiled formula 10^5 times (run) for a 13x13 determinant. All times are in seconds. The O3 option in FORM used $C_p = 0.07$ and 10×400 tree expansions.

```
Format nospaces;
Format O3,stats=on;
Local F2 = 6*y*z^2+3*y^3-3*x*z^2+6*x*y*z-3*x^2*z+6*x^2*y;
Bracket y;
Print;
.end
  w(1)=x**2
  w(2)=x+z
  w(2)=z*w(2)
  w(2)=w(1)+w(2)
  w(2)=6*w(2)
  w(3)=-z*x
  w(1)=-w(1)+w(3)
  w(1)=3*z*w(1)
  F2=y*w(2)+3*y**3+w(1)
```

	HEP(σ)	F_{13}	F_{24}
Variables	4+11	5+24	5+31
Expressions	35	56	56
Terms	5 717	105 114	836 010
Format O0	34 437	816 158	5 776 416
Format O1	5 764	72 013	400 044
Format O2	4 661	46 507	233 458
Format O3	3 387	42 912	181 871
Terms shifted	1 518	37 123	215 790
Format O0	9 141	286 162	1 520 117
Format O1	2 442	48 558	156 537
Format O2	2 013	39 003	116 442
Format O3	1 693	30 943	92 819

Table 5: Results for three physics formulas with simultaneous optimizations. The concept of shifting is explained in the text.

```
*** STATS: original  0P 12M 3A : 15
*** STATS: optimized 0P 6M 3A : 9
```

This is very convenient in the GRACE[3] system where we have the Feynman parameters outside the brackets and need to evaluate the coefficients.

For three outputs from the GRACE system we obtain the results in Tab.5.

Results for the physics formulas in the original and the shifted (explained below) versions. The counting of the number of operations takes the brackets in the Feynman parameters into account. The number of variables is the number of Feynman parameters plus the number of other variables.

The FORM optimization cannot use knowledge that we may have about the formulas. In a sense it sees the formula as chaotic. In terms of AI, it does not have 'domain specific knowledge'. Of course, often the best optimizations are based on knowledge about the system, like knowing that in some combination of variables the formula will be shorter. On the other hand, just introducing lots of new variables makes the work of the MCTS much harder, because the tree becomes bigger. As it turns out, the GRACE output can be greatly improved by making 'shifts' in variables. An example would be that

```
    w1 = 2*p1.p2
    w2 = me^2
    w1+w2 -> w1
```

or: instead of the original w1 we rewrite the formula in terms of the new w1, and very often the number of terms in the expression becomes smaller.

We have written a set of FORM procedures for this and use them before we apply the output optimizations. The results of this are the 'shifted' results in Tab. 5.

Hence, the best results are obtained when one first applies 'domain specific knowledge', and after that the FORM optimizations. This is actually also much faster in CPU time, because the shorter formulas after the shifts have an enormous impact on the speed of the O2 and O3 optimizations.

In the worst example shown here (diagram 4320x22 in the reaction $e^-e^+ \to \mu^-\overline{\nu}_\mu u\overline{d}$) we started with 2427746 terms. The shifts took this down to 193893 terms after which the O3 option of FORM reduced the number of operations from 1318539 to 119962. The shifting took 82000 sec and the O3 optimization 'only' 16700 sec. On the original expressions the O2 optimization alone cost already 70000 sec. and left more than 600000 operations.

As one can see, we are talking about enormous factors. And with outputs this size, there is very little chance that the compiler can perform any level of optimizations. It took for O0 17.2 sec and for O1 87.6 sec, but this is on an output that has been made very compiler friendly. On the unoptimized formulas, using the compiler flag O1, it crashed after a long time.

6. Outlook

It should be clear that code simplification is not a closed book. Improvements are likely to be found in the future, both in the field of domain specific knowledge and in the improvement of chaotic code.

I also expect that both Maple and Mathematica will improve their performance.

The application of MCTS in physics should not be restricted to code optimization. It should be considered in any field where one has to search though trees that are far too large for stepping through completely. One such field is the combination of recursion relations in the calculation of all Mellin moments in DIS as used in [12, 13]. Different orderings have completely different properties. They may lead to unacceptably slow programs, or to spurious poles. The use of MCTS to such systems is currently under study.

An interesting spinoff of the implementation of MCTS in the FORM optimization is that the method of MCTS can be studied rather cleanly with it. Much better than in the games of Go or chess. This may lead to improvements of the MCTS method as well.

References

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

[2] J. Kuipers, T. Ueda, J. A. M. Vermaseren, J. Vollinga, FORM version 4.0, Comput. Phys. Commun. **184** (2013) 1453, arXiv:1203.6543 (2012).

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

[4] W.G. Horner, Phil. Trans. Soc. London (1819) 308–305. *Reprinted in:* D.E. Smith, *A Source Book in Mathematics* McGraw-Hill (1959)

[5] M. Kojima, J. Oper. Res. Soc. Japan **51** (2008) 29-âĂŞ54.

[6] L. Kocsis, C. Szepesvári, LNCS **4212** (2006) 282–293.

[7] G.M.J.B. Chaslot, *Monte-Carlo Tree Search*, PhD thesis (2010).

[8] J. Kuipers, J.A.M. Vermaseren, A. Plaat and H.J. van den Herik, Comput. Phys. Commun **184** (2013) 2391, arXiv:1207.7079 (2012).

[9] J. Kuipers, T. Ueda and J.A.M. Vermaseren, arXiv:1310.7007 (2013).

[10] C.E. Leiserson, L. Li, M.M. Maza, Y. Xie, LNCS **6327** (2010) 342–353.

[11] T. Reiter, Comput. Phys. Commun. **181** (2010) 1301-1331.

[12] S. Moch and J.A.M. Vermaseren, Nucl. Phys. B573 (2000) 853, hep-ph/9912355

[13] S. Moch, J.A.M. Vermaseren and A. Vogt, Nucl. Phys. B688 (2004) 101, hep-ph/0403192