Nikhef 2013-036
TTP13-031
SFB/CPP-13-80
October 2013

Code Optimization in FORM

J. Kuipers[a], T. Ueda[b] and J.A.M. Vermaseren[a]

[a]*Nikhef Theory Group*
Science Park 105, 1098 XG Amsterdam, The Netherlands

[b]*Institut für Theoretische Teilchenphysik, Karlsruhe Institute of Technology (KIT)*
D-76128 Karlsruhe, Germany

Abstract

We describe the implementation of output code optimization in the open source computer algebra system Form. This implementation is based on recently discovered techniques of Monte Carlo tree search to find efficient multivariate Horner schemes, in combination with other optimization algorithms, such as common subexpression elimination. For systems for which no specific knowledge is provided it performs significantly better than other methods we could compare with. Because the method has a number of free parameters, we also show some methods by which to tune them to different types of problems.

1 Introduction

One of the uses of computer algebra is to prepare potentially large formulas for frequent numerical evaluation. This is particularly the case in particle physics. The most widespread way to compute reactions in particle physics is by means of perturbative field theory. Even at the one loop level (usually the second term in the perturbative expansion) one may encouter large numbers of Feynman diagrams, each resulting in a lengthy formula[1]. Such calculations are undertaken to compare theories with experimental results. Hence such formulas have to be integrated over the region of sensitivity of the detectors that measure these reactions. This region is called the experimental acceptance and the only technique that is available to integrate over it is Monte Carlo integration. One may have to evaluate the formulas millions of times to obtain accurate results. Hence it is important to have a representation of the formulas that is as short as possible, even if this involves a non-negligible cost during the computer algebra phase of the calculation.

Optimizing the output of the formulas can be done in two different ways. The first is *domain specific*. This means that specific knowledge about the behavior of the formulas is provided to make the formulas shorter. An example is an equation in two variables x and y, but it is known that $x + y$ and $x - y$ are more natural variables and make the formulas shorter. For the second way either there is no domain specific knowledge, or it is too much work to obtain it. In that case the formula has to be treated by generic means. It should be clear that usually the best results are obtained when domain specific knowledge is applied first, followed by a generic method to clean up what is left.

In computer algebra the challenge is to make a system for the optimization of the output of expressions in the absence of domain specific knowledge. In addition this system should work reasonably fast, which we interpret as subquadratic in the length of the input expression. In the recent past several methods have been published in which two of the authors reported on new techniques to improve upon existing methods [**?**, **?**, **?**]. It turns out that an optimization method based on Monte Carlo tree search [**?**, **?**], a recent search method from artificial intelligence and game theory, performs best on the benchmarks that were tested. This method has caused much excitement in the field of game theory, because it has improved the strength of Go playing computer programs from advanced beginner to medium level players, and on small (9x9) boards they have reached top level strength. Application of this technique to the field of formula simplification has led to sufficiently positive results that we have decided to implement it in the computer algebra language FORM [**?**] in such a way that all its users can benefit from it.

Since this is such a new field, we do not yet have extensive experience with applications and how different types of formulas need different values for the controlling parameters. Hence we have made an implementation in which the user has access to these parameters and can tune them to whole categories of formulas. This means that at the moment we have a number of default settings which may be changed by the user. In later versions we may try to have the program tune these parameters for individual formulas automatically.

The outline of the paper is as follows. In section **??** we explain the algorithms that are used for the simplification. The syntax of the FORM implementation is explained in section **??** including all parameters that can be set. In section **??** we discuss a number of examples.

[1] In special cases other techniques can be used that lead to surprisingly simple formulas [**?**], but in general these are not applicable.

Section **??** is dedicated to studying the effects of some parameters and the determination of good settings for a number of formulas. We finish with remarks about potential future development. All programs that we use can be obtained from the FORM website at ref.[**?**].

2 Code optimization algorithms

2.1 Horner's method

For optimizing polynomials in a single variable, the textbook algorithm called *Horner's method* gives an efficient form for evaluating it [**?**]. It can be written as follows:

$$a(x) = \sum_{i=0}^{n} a_i x^i = a_0 + x(a_1 + x(a_2 + x(\ldots + x \cdot a_n))). \tag{1}$$

If the polynomial is of degree n and dense, this form takes n multiplications and n additions to calculate its value.

It is possible to generalize Horner's method for multivariate polynomials, but this generalization is not unique. First, one of the variables in the polynomial is selected and Eq. (**??**) is applied, thereby treating the other variables as constants. Next, a different variable is chosen and Horner's rule is applied again on the parts not containing the first variable. This method is repeated until all variables have been selected. As an example, we consider the polynomial $a(x,y,z) = y - 3x + 5xz + 2x^2yz - 3x^2y^2z + 5x^2y^2z^2$ and chose the variable x first, then y and finally z. This results in the following representation:

$$a(x,y,z) = y + x(-3 + 5z + x(y(2z + y(z(-3 + 5z))))). \tag{2}$$

This representation takes 8 multiplications and 5 additions to evaluate, while the original form takes 18 multiplications and 5 additions. This behavior is generic: Horner's method reduces the number of multiplications and leaves the number of additions constant.

For the multivariate Horner method it is important in which order the variables are processed. Different orders may lead to huge differences in the number of operations used to evaluate a polynomial [**?**]. Classically, simple greedy algorithms like sorting the variables by number of occurrences are used to determine the order [**?**]. Recently, two of the authors of this paper describedan algorithm based on Monte Carlo tree search to determine more efficient orders [**?**].

2.1.1 Occurrence order

In the *occurrence order* all variables are ordered with respect to their number of occurrences in the polynomial. The variable that appears most often is the first variable in the order [**?**]. At every step in the multivariate Horner's method this results in the largest decrease in the number of operations, because it is the most-occurring variable that is factored out of the polynomial. This greedy approach usually gives good results.

Another simple order is the *reverse occurrence order*. As the name suggests, this order contains the variables sorted with resprect to the number of occurrences, but with the least-occurring variable first. This method usually results in Horner schemes that use more operations than the normal occurrence order to evaluate the polynomial. However, since the