Detecting Looping Simplifications

by

Paul W. Purdom, Jr.

Computer Science Department
Indiana University
Bloomington, Indiana 47405

TECHNICAL REPORT NO. 208

Detecting Looping Simplifications

by

Paul W. Purdom, Jr.

December, 1986

Detecting Looping Simplifications

Abstract: A generalization of tree matching and unification algorithms is presented. Given the equation $s = t$, this algorithm can often quickly determine that the rewrite rule $s \rightarrow t$ leads to an infinite sequence of "simplifications". The rule $t \rightarrow s$ can be tested in the same way. Rules leading to infinite simplifications should not be included in a rewrite system. In general, the problem of deciding whether a set of rewrite rules leads to infinite simplifications is undecidable. The algorithm that is used for this problem is a cross between a unification algorithm for terms with overlapping variables and a matching algorithm. In the simplest case it attempts to find a, σ_M and σ_U such that $\sigma_M \sigma_U s = \sigma_U t/a$. The substitution σ_M is used like the substitution that occurs in matching problems and the substitution σ_U is used like the one that occurs in unification problems. If $\sigma_M \sigma_U s = \sigma_U t/a$ has a solution, then the rule $s \rightarrow t$ leads to infinite simplifications. The same basic algorithm can be used to test more complex cases of looping involving the interaction of several rules, but it is limited to those cases where each application of a rule occurs inside of the previous rule application. Experiments suggest that the simplest form of the algorithm is about 80 percent effective in eliminating bad orientations of rules. The algorithm never rules out a good orientation of a rule, and so it is most useful when one wants to consider all possible rule orientations.

Keywords: Knuth-Bendix, Looping, Matching, Rewrite rules, Simplification, Unification.

1. Introduction

One basic problem that arises in the Knuth-Bendix procedure [6, 10] is to orient equations: convert an equation $s = t$ into the rule $s \rightarrow t$ or the rule $t \rightarrow s$. The decision of how to orient each rule can be critical to the operation of the procedure. Two variations of the procedure obtain the same final answer if they use the same method for orienting rules [11]. On the other hand, one orientation method may lead to a finite set of rewrite rules, while another may lead to infinite calculations.

Often when a person runs the Knuth-Bendix system, he wants to obtain a finite set of rewrite rules so long as it is not too much trouble. Various termination orderings have been devised for such people. Dershowitz [2] gives a survey of such orderings. Sometimes, however, a person wishes to obtain a finite set of rewrite rules if at all possible. Then he may wish to try every possible ordering until he finds one that works. He will have to be careful not to spend too much time on any one ordering because some of them probably lead to infinite computations. At such times he would like an orientation method that eliminates those orientations which are *known* to lead to infinite computations but that does not eliminate any which might lead to finite computations. A perfect system would eliminate *all* orientations that lead to infinite computations and no others, but there is no such algorithm — the problem is undecidable.

This paper gives an algorithm for the elimination of most orientations that lead to infinite simplifications while not eliminating any that do not. In what follows infinite

simplifications will be called *looping* to save space. The Knuth-Bendix procedure can diverge due to looping or due to starting to generate an infinite set of rules. The current paper attacks the looping problem. Herman and Prívara [4] attack the infinite number of rules problem. They eliminate some orientations that are known to lead to an infinite number of rules. Thus, their algorithm complements the current one; the Knuth-Bendix procedure needs to consider only those rules that pass both test.

The present algorithm is apparently the first algorithm to reject (some) looping rules while not ever rejecting non-looping rules. Plaisted [13] gives an algorithm based on a similar idea: he eliminates orientations that are not consistent with any simplification ordering while not eliminating any that are consistent. His method does not allow rules such as $f(f(X)) \rightarrow f(g(f(X)))$, which do not loop, because such rules are not consistent with a simplification ordering.

I recommend that you first use the Knuth-Bendix procedure with some termination ordering. If the termination ordering leads to a finite set of rules, you have obtained your answer the easy way. If it does not seem to be leading to a finite set of rules, then you can use the current method and order the rules whichever way seems best. Eventually you will orient your rules correctly or give up on the problem. The method does not prevent you from trying out any promising orientation. So far I have not followed my own advice; the loop detection algorithm is easy to program and is fairly effective, so I have used it alone until I get around to building a simplification ordering routine. Before I had any assistance with orientation, looping was a major problem. After I added the loop testing algorithm, looping has seldom been a problem.

It is undecidable whether or not a set of rewrite rules loops [7]. The problem remains undecidable for sets with 2 rules [2]; the complexity of one rule systems is not known. As a result, no ordering method is completely satisfactory if one wishes to exhaustively consider possible orientations for rules. Compared to other published orientation methods the current algorithm gives the most freedom but the least help in solving orientation problems. The user is left with the problem of proving that his final set of rules do not loop.

2. Some Simple Loops

The rewrite rule $*(e, X) \rightarrow *(X, e)$ loops on the term $*(e, e)$. The reason for this is that the left and right sides of the rule unify with each other — for $X = e$ the two sides are the same. This generalizes to: $s \rightarrow t$ loops when s unifies with any part of t.

For many applications of unification, one first modifies one of the terms so that they have distinct variables. This is incorrect for this problem; the original variables must be retained. For example, the rule $+(X) \rightarrow X$ does not loop even though X' unifies with $+(X)$.

The rule $*(X, Y) \rightarrow *(i(i(X)), Y)$ loops. In this case the two sides do not unify. Here the reason for looping is that the left side matches the right — if on the left side X is replaced with $i(i(X))$ then the left and right sides are the same. This generalizes to: $s \rightarrow t$

loops when s matches part of t. For this case, the variables of t can be considered as distinct from those of s.

These two observations can be generalized. Let σ be a substitution of values for variables. The individual substitutions in σ will be written in the form $X \leftarrow t$, where X is a variable and t is a term. The term t may contain variables. Those variables that are not assigned values in σ retain their original value. I will say $\sigma[X]$ is *empty* in such cases. The notation σt means replace each variable in the term t with its value in σ. Do not apply σ to itself, i.e., do not assign values to variables that appear in the right side of the assignments in σ. The notation $\sigma^2 t$ means $\sigma(\sigma t)$ and t/a means the subtree at position a in t. The position is usually represented as a number where the first digit says which child of the root to follow, the second digit says which child of that node to follow, etc. The notation a^i means to follow the path consisting of i a's.

Theorem 1. The rewrite rule $s \rightarrow t$ loops if there exists a position a in t and substitutions σ_M and σ_U such that

$$\sigma_M \sigma_U s = \sigma_U(t/a).$$

(The substitution σ_M is similar to matching, and the substitution σ_U is similar to unification.)

Proof: Let

$$u_i = (\sigma_U t)[a \leftarrow \sigma_M \sigma_U(t)] \cdots [a^i \leftarrow \sigma_M^i \sigma_U(t)].$$

Applying the rule $s \rightarrow t$ to u_i at a^{i+1} gives u_{i+1}, leading to an infinite sequence of simplifications.

Theorem 1 provides a way to test a single rule for looping, but it does not always detect looping when it is present. Dershowitz [2] gives a number of cases where it fails. In practice, however, it does quite well. For example, when the Knuth-Bendix procedure is run on the standard 3 rule representation of group theory [10], theorem 1 suggests a unique orientation for all equations that arise except for two: $*(*(X,Y),Z) = *(X,*(Y,Z))$ and $i(*(X,Y)) = *(i(Y),i(X))$. These two equations can be oriented either way without causing looping, although the second rule must be oriented properly to avoid generating an infinite number of rules.

The following table summarizes the results of a few tests based on theorem 1.

	number of orientations		
Initial rules	0	1	2
groups (three rules) [10]	0	17	2
dihedral group of order 8	0	22	24
central groupoid (second version) [10]	0	43	13
groups (one rule) [5]	9	137	14

The three columns of numbers show the number of rules for which it permitted zero orientations, one orientation, and two orientations. In most cases the algorithm was able