Path Orderings for Termination of Associative-Commutative Rewriting*

Nachum Dershowitz, Subrata Mitra
Department of Computer Science
University of Illinois at Urbana-Champaign
1304 West Springfield Avenue
Urbana, Illinois 61801, USA.
{nachum, mitra}@cs.uiuc.edu

Abstract

We show that a simple, and easily implementable, restriction on the recursive path ordering, which we call the "binary path condition," suffices for establishing termination of extended rewriting modulo associativity and commutativity.

1 Introduction

Rewrite systems find application to various aspects of theorem proving and programming language semantics. The essential idea in rewriting is to use an asymmetric directed equality ($\rightarrow$), rather than the usual symmetric equality relation ($\approx$).

Termination of a system consisting of such directed equations means that no infinite sequences of left-to-right replacements are possible for any term. Termination is important for using rewriting as a computational tool, and for simplification in theorem provers. One popular way of proving termination of a rewrite system is to use *path orderings*, based on a precedence relation on the function symbols of the system. Another common approach interprets function symbols as multivariate polynomials. For a survey of these techniques, see [Der87].

A binary function f is said to be associative and commutative (AC for short) if f obeys the equations

$$\begin{aligned} f(x, f(y, z)) &= f(f(x, y), z) \\ f(x, y) &= f(y, x). \end{aligned}$$

Since it is not possible to orient the second equation without losing termination, rewriting modulo such a congruence has to be handled in a special way. In essence, we rewrite AC equivalence classes, rather than terms.

*This research was supported in part by the U. S. National Science Foundation under Grants CCR-90-07195 and CCR-90-24271.

Polynomials can be used to prove termination of rewriting modulo AC when AC-equivalent terms have the same interpretation. But this severely restricts the degree of polynomial that can be used. (See [Lan79] and [BL87].) Path orderings have been commonly used in theorem provers, even for AC-rewriting (see the discussion in [Bjo82, page 350]), despite the fact that they do not establish termination in the AC case (see the counterexamples in [DHJP83]). Extensions of path orderings have been proposed that do handle associative and commutative functions properly ([BP85], for example), most recently in [KSZ90]. However, the ordering of [KSZ90] is difficult to implement, because it requires many nondeterministic operations (like *pseudocopying*; see Section 2).

In this paper, we show that if a rewrite system can be proved terminating using the recursive path ordering (RPO), then it is also AC-terminating—provided that when comparing two terms with the same (or equivalent) AC symbol at their roots, we compare subterms componentwise, rather than as multisets. This criterion can be easily implemented.

We write $s \sim_{ac} t$ to denote that s and t are rearrangements using the AC axioms. AC-rewriting ($\rightarrow_{R/AC}$) can be defined as follows: $u[s]_\pi \rightarrow_{R/AC} u[t]_\pi$, for terms s, t, context u and position π, if $s \sim_{ac} s'$, $s' \rightarrow_R t'$ and $t' \sim_{ac} t$. When dealing with AC systems, it is often convenient to treat AC symbols as functions with variable arity by considering only *flattened* terms. We use $\overline{t}$ to denote the flattened version of t. An ordering $\succ$ is *AC-compatible* if, for all terms s, s', t, t', $s \sim_{ac} s' \succ t' \sim_{ac} t$ implies $s \succ t$, in which case, we can also say that $\overline{s} \succ \overline{t}$. A rewrite system is *AC-terminating* if and only if the relation $\rightarrow_{R/AC}$ is contained in an AC-compatible reduction ordering. In general, we use standard terminology and notation for rewrite systems. For a survey of the field, refer to [DJ90].

2 Binary Path Condition

In this section we develop a restricted version of RPO—called "binary path condition"—which can be extended to an AC-compatible reduction ordering.

We first show that RPO, in general, is not AC-compatible. Consider the rule

$$f(a, f(a, b)) \quad \rightarrow \quad f(b, f(a, a))$$

If we consider $b \succ_f a$, then we can show that $f(a, f(a, b)) \succ_{rpo} f(b, f(a, a))$, assuming multiset status for f. However, we also have that $f(a, f(a, b)) \sim_{ac} f(b, f(a, a))$. Clearly, RPO with lexicographic status is not compatible with the commutativity axiom:

$$f(a, b) \quad \rightarrow \quad f(b, a)$$

If we now have $a \succ_f b$, then using left-to-right status for f, we have $f(a, b) \succ_{rpo} f(b, a) \sim_{ac} f(a, b)$, which violates irreflexivity. Finally, we show that RPO on flattened terms is not AC-compatible:

$$\begin{array}{rcl} f(a, b) & \rightarrow & g(a, b) \\ f(a, g(a, b)) & \rightarrow & f(a, a, b) \end{array}$$

Here $f \succ_f g$, and f is AC. Now, we have $f(a, a, b) = \overline{f(a, f(a, b))} \succ_{rpo} f(a, g(a, b)) \succ_{rpo} f(a, a, b)$, which violates irreflexivity.

These counterexamples show that RPO with status cannot be extended to an AC-compatible ordering. We therefore define a restricted version of it ($\succ_{bpc}$), which uses RPO

with status for the non-AC symbols, but uses RPO without status to compare terms which have equivalent top-level AC operators. Here we use $t \succsim_{bpc} s$ to mean $t \sim_{ac} s$ or $t \succ_{bpc} s$.

Definition 2.1 (Binary Path Condition). Let $\succ_f$ be a well-founded precedence ordering on the function symbols. We have $t = f(t_1, \ldots, t_n) \succ_{bpc} g(s_1, \ldots, s_m) = s$ iff one of the following holds:

1. $t_i \succsim_{bpc} s$ for some i, $1 \leq i \leq n$.
2. $f \succ_f g$, and $t \succ_{bpc} s_j$ for all j, $1 \leq j \leq m$.
3. $f \sim_f g$, f and g are non-AC, and have the same status, and either
   - f has multiset status, and $\{t_1, \ldots, t_n\} \succ_{mul} \{s_1, \ldots, s_m\}$, or,
   - f has lexicographic status, and
     - $(t_1, \ldots, t_n) \succ_{lex} (s_1, \ldots, s_m)$, and
     - $t \succ_{bpc} s_j$ for all j, $1 \leq j \leq m$.
4. $f \sim_f g$, f, g are AC, $t = f(t_1, t_2)$ and $s = g(s_1, s_2)$, and either $(t_1, t_2) \succ_{comp} (s_1, s_2)$ or $(t_1, t_2) \succ_{comp} (s_2, s_1)$, where $(t_1, t_2) \succ_{comp} (s_1, s_2)$ iff either $t_1 \succ_{bpc} s_1$ and $t_2 \succsim_{bpc} s_2$, or, $t_1 \succ_{bpc} s_2$ and $t_2 \succsim_{bpc} s_1$.

To compare terms with variables, we can use the fact that a ground term $t\sigma$ is greater under $\succ_{bpc}$ than $x\sigma$ (x is a variable), for any substitution σ, whenever x occurs in t.

Theorem 2.2. *Let R be a rewrite system. If for each rule $l \to r \in R$ we have $l \succ_{bpc} r$, then R is AC-terminating.*

We first recall the definition of AC-RPO ($\succ_{ac}$), on ground terms, due to [KSZ90]. This ordering compares flattened terms.

Definition 2.3. Let $\succ_f$ be a well-founded precedence ordering on the function symbols. We have $t = f(t_1, \ldots, t_n) \succ_{ac} g(s_1, \ldots, s_m) = s$ iff one of the following holds:

1. $t_i \succsim_{ac} s$ for some i, $1 \leq i \leq n$, where $t_i \succsim_{ac} s$ iff $t_i \sim_{ac} s$ or $t_i \succ_{ac} s$.
2. $f \succ_f g$, and $t \succ_{ac} s_j$ for all j, $1 \leq j \leq m$.
3. $f \sim_f g$, f and g are non-AC and have the same status, and either
   - f has multiset status, and $\{t_1, \ldots, t_n\} \succ_{mul} \{s_1, \ldots, s_m\}$, or,
   - f has lexicographic status, and
     - $(t_1, \ldots, t_n) \succ_{lex} (s_1, \ldots, s_m)$, and
     - $t \succ_{ac} s_j$ for all j, $1 \leq j \leq m$.
4. $f \sim_f g$, f, g are AC, $t = f(T)$, $s = g(S)$, $S' = S - T = \{s'_1, ..., s'_k\}$ (where "$-$" denotes the multiset difference performed using $\sim_{ac}$, i.e., terms equivalent with respect to $\sim_{ac}$ can be dropped from both T and S), and either

- $k = 0$ and $n > m$ (i.e., $S - T = \emptyset$ and $T - S \neq \emptyset$), or
- $f(T - S) \Rightarrow^* f(T')$, and $T' = T_1 \cup \ldots \cup T_k$ and for all i $(1 \leq i \leq k)$ either
  - $T_i = \{u\}$ and $u \succsim_{ac} s'_i$, or
  - $T_i = \{u_1, \ldots, u_l\}$ and $f(u_1, \ldots, u_l) \succsim_{ac} s'_i$.

  Also, in this case, either $t \Rightarrow^+ f(T')$, or, for at least one i, we have instead a strict decrease in $\succ_{ac}$.

Case 4 of this definition uses the operation $\Rightarrow$, which may be one of the following: *pseudo-copying*, *elevation* or *flattening*. Here we briefly explain these notions; for details refer to [KSZ90]. Pseudo-copying is used to allow a single *big* (that is, with a top-level function which is higher than f in the precedence relation $\succ_f$) subterm on the left-hand side to handle multiple subterms on the right. For example, while comparing the terms $t_1 = f(g(x))$ and $t_2 = f(h(x), h(x))$, where f is AC, and $g \succ_f f \succ_f h$, we can say that $t_1 \succ_{ac} t_2$, since $t_1 \Rightarrow f(gg(x), gg(x)) \succ_{ac} t_2$, where $gg(x)$ is a pseudo-copy of $g(x)$. Note that pseudo-copying is allowed only for big terms which are immediate subterms of the top-level AC operator of the left-hand side term. At times, a big term may be nested further down, in which case elevation is used to bring it up. For example, in comparing $f(c(g(x)))$ with $f(h(x), h(x))$, where $g \succ_f f \succ_f h \succ_f c$, we can use the following steps: $f(c(g(x))) \Rightarrow f(g(x)) \succ_{ac} f(h(x), h(x))$. Finally, flattening can be used to remove immediate nesting of different AC functions which have the same precedence. For example, we could say $f(g(x), y) \Rightarrow f(x, y)$, if $f \sim_f g$, and f and g are AC. The essential idea in this ordering is to partition the subterms of the AC functions and compare the components, using $\Rightarrow$ to make the relation transitive. It is shown in [KSZ90] that this ordering is well-founded and AC-compatible.

We are ready for a proof of the theorem:

Proof. We show that for any two terms s and t, if $t \succ_{bpc} s$ then $\overline{t} \succ_{ac} \overline{s}$, by induction on the sizes of s and t. There are several cases to be considered, depending on the possibile reasons why $t \succ_{bpc} s$. We assume that t is of the form $f(t_1, \ldots, t_n)$ and s is $g(s_1, \ldots, s_m)$.

1. If $t_i \succsim_{bpc} s$, then, by the inductive hypothesis we have $t_i \succsim_{ac} s$, and hence $\overline{t} \succ_{ac} \overline{s}$, by Case 1 of Definition 2.3.

2. If $f \succ_f g$ and $t \succ_{bpc} s_j$, then by the inductive hypothesis, we have $\overline{t} \succ_{ac} \overline{s_j}, 1 \leq j \leq m$. There are two further possibilities:

   - g is not AC. In this case, $\overline{s} = g(\overline{s_1}, \ldots, \overline{s_m})$, and therefore we have $\overline{t} \succ_{ac} \overline{s}$, by Case 2 of Definition 2.3.
   - Suppose g is AC (thus $m = 2$). In this case, there are various possibilities for s, for example we could have: $s = g(g(s_{1.1}, s_{1.2}), s_2)$, or $s = g(s_1, g(s_{2.1}, s_{2.2}))$, or $s = g(g(s_{1.1}, s_{1.2}), g(s_{2.1}, s_{2.2}))$, and so forth. However, in each case, we have $\overline{s} = g(s'_1, \ldots, s'_k)$, where each $s'_j, 1 \leq j \leq k$, is either a subterm of s_1 or of s_2. Therefore, we have $\overline{t} \succ_{ac} \overline{s}$.

3. If f and g are both non-AC, and $f \sim_f g$, then we can use the inductive hypothesis on the flatten subterms of s, and then the proposition follows.

4. If f and g are both AC, and $f \sim_f g$ then $n = m = 2$. Furthermore, without loss of generality, we can assume that $t_1 \succ_{bpc} s_1$, *and* $t_2 \succsim_{bpc} s_2$. (The other cases admit similar proofs.) There are two further possibilities:

   - If $t_2 \succ_{bpc} s_2$, then by the inductive hypothesis we have: $\overline{t_1} \succ_{ac} \overline{s_1}$ and $\overline{t_2} \succ_{ac} \overline{s_2}$. Thus, we could use this partitioning of t to show that $\overline{t} \succ_{ac} \overline{s}$.
   - If $t_2 \sim_{ac} s_2$, then again the proposition holds, because we could ignore $\overline{t_2}$ and $\overline{s_2}$ when comparing $\overline{t}$ with $\overline{s}$.

□

Since $\succ_{ac}$ is AC-compatible, we have $\overline{t} \succ_{ac} \overline{s}$, not only when $t \succ_{bpc} s$, but also if $t \sim_{ac} t' \succ_{bpc} s$. In order to prove termination of a system using $\succ_{bpc}$, it is therefore sufficient to use any rearrangement of the left- and right-hand side terms. We also have, for any AC function symbol f and terms s and t, that if $t \succ_{bpc} s$, then $f(t, X) \succ_{bpc} f(s, X)$ (and therefore, $\overline{f(t, X)} \succ_{ac} \overline{f(s, X)}$).

We have shown that the relation $\succ_{bpc}$ is embedded in the AC-compatible reduction ordering $\succ_{ac}$. Therefore, the binary path condition is sufficient for proving AC-termination. It is important to note that the relation ($\succ_{bpc}$) defined here is not really an ordering, because it is not transitive. For example, if we have the precedence relation $g \succ_f f \succ_f h$, then we can show that (here f is AC, while g and h are non-AC)

$$f(g(x), g(y)) \succ_{bpc} f(f(x, x), f(y, y)) \sim_{ac} f(f(x, y), f(x, y)) \succ_{bpc} f(h(x, y), h(x, y)).$$

However, it is the case that $f(g(x), g(y)) \not\succ_{bpc} f(h(x, y), h(x, y))$. The interesting point about $\succ_{bpc}$ is that it is easy to implement; much easier than the ordering of [KSZ90].

3 Examples

The binary path condition developed in the previous section, like $\succ_{ac}$, and unlike [BP85], has no restriction on the precedence relation $\succ_f$, and can therefore be used to prove termination of a large class of rewrite systems.

Two examples follow:

Example 3.1 (Arithmetic over natural numbers). *Here* $*$ *and* $+$ *are AC, and* $* \succ_f + \succ_f s \succ_f 0$.

$$\begin{aligned} 0 + x &\rightarrow x \\ s(x) + y &\rightarrow s(x + y) \\ 0 * x &\rightarrow 0 \\ s(x) * y &\rightarrow y + (x * y) \\ (x + y) * z &\rightarrow (x * z) + (y * z) \end{aligned}$$

Example 3.2 (Free commutative ring). *Here* $*$ *and* $+$ *are AC, and* $* \succ_f - \succ_f + \succ_f 0$.

$$\begin{array}{rcl} 0+x & \to & x \\ -x+x & \to & 0 \\ -0 & \to & 0 \\ --x & \to & x \\ -(x+y) & \to & -x+-y \\ 0*x & \to & 0 \\ -x*y & \to & -(x*y) \\ x*(y+z) & \to & (x*y)+(x*z) \end{array}$$

4 Discussion

We have considered a simple restriction on RPO that can be extended to an AC-compatible ordering. The restriction disallows comparison of two terms with equivalent top-level AC functions when both subterms on the right-hand side are dominated by only one subterm on the left-hand side.

Independently, Bachmair [Bac92] has presented an AC-compatible rewrite-relation, also based on [KSZ90], and proved its termination using a minimal counterexample argument. Our termination condition is essentially the same as one rewrite step of [Bac92], with the possibility of multiset status added. It is believed that the transitive closure of this relation is identical to the ordering in [KSZ90], but this remains to be proved.

It will be interesting to be able to extend the relation defined here to cases where simple multiset comparisons may be allowed for subterms of the AC-terms.

References

[Bac92] Leo Bachmair. Associative-commutative reduction orderings. *Information Processing Letters*. To appear.

[BP85] Leo Bachmair and David A. Plaisted. Termination orderings for associative-commutative rewrite systems. *J. of Symbolic Computation*, vol. 1, pages 329–349 (1985).

[BL87] Ahlem Ben Cherifa and Pierre Lescanne. Termination of rewriting systems by polynomial interpretations and its implementation. *Science of Computer Programming*, vol. 9, pages 137–159 (1987).

[Bjo82] Dines Bjorner, editor. Proceedings of the IFIP Working Conference on Formal Description of Programming Concepts–II. Garmisch-Partenkirchen, West Germany, North-Holland 1982.

[Der87] Nachum Dershowitz. Termination of rewriting. *J. of Symbolic Computation*, vol. 3, pages 69–116 (1987).

[DHJP83] Nachum Dershowitz and Jieh Hsiang and N. Alan Josephson and David A. Plaisted. Associative-commutative rewriting. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence,* Karlsruhe, West Germany, pages 940–944, 1983.

[DJ90] Nachum Dershowitz and Jean-Pierre Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, chapter 6, pages 243–320, North-Holland, Amsterdam, 1990.

[KSZ90] Deepak Kapur, G. Sivakumar and Hantao Zhang. A new method for proving termination of AC-rewrite systems. In *Proceedings of the Tenth International Conference of Foundations of Software Technology and Theoretical Computer Science*, vol. 472 of *Lecture Notes in Computer Science*, pages 133–148, Springer-Verlag, Berlin, 1990.

[Lan79] Dallas S. Lankford. On proving term rewriting systems are Noetherian. Memo MTP-3, Mathematics Department, Louisiana Tech. University, Ruston, LA, 1979.