Topics in Termination*

Nachum Dershowitz and Charles Hoot

Department of Computer Science, University of Illinois, Urbana, IL 61801, U.S.A.
nachum,hoot@cs.uiuc.edu

Abstract. We generalize the various path orderings and the conditions under which they work, and describe an implementation of this general ordering. We look at methods for proving termination of orthogonal systems and give a new solution to a problem of Zantema's.

1 Introduction

If no infinite sequences of rewrites are possible, a rewrite system is said to have the *termination* property. In practice, one usually guarantees termination by devising a well-founded (strict partial) ordering $\succ$, such that $s \succ t$ whenever s rewrites to t. As suggested in [Manna and Ness, 1970], it is often convenient to separate reduction orderings into a homomorphism from terms to an algebra with a well-founded ordering. The use, in particular, of *polynomial interpretations* which map terms into the natural numbers, was developed by Lankford [1979]. For a survey of termination methods, see [Dershowitz, 1987].

Virtually all orderings used in practice are *simplification orderings* [Dershowitz, 1982], satisfying the *replacement* property, that $s \succ t$ implies that any term containing s is not less (under $\succ$) than the same term with that occurrence of s replaced by t, and the *subterm* property, that any term containing s is greater or equal to s. Simplification orderings cannot be used to prove termination of "self-embedding" systems, that is, when a term t can be derived in one or more steps from a term t', and t' can be obtained by repeatedly replacing subterms of t with subterms of those subterms.

Knuth and Bendix [1970] designed a particular class of well-orderings which assigns a weight to a term which is the sum of the weights of its constituent function symbols. Terms of equal weight and headed by the same symbol have their subterms compared lexicographically. Another class of simplification orderings, the *path orderings* [Dershowitz, 1982], is based on the idea that a term u should be bigger than any term that is built from smaller terms, all held together by a structure of function symbols that are smaller in some precedence ordering than the root symbol of u. The notion of path ordering was extended by Kamin and Lévy [1980] to compare subterms lexicographically and to allow for a semantic component; see [Dershowitz, 1987]. Here, we generalize these orderings and the conditions under which they work. In the appendix, we describe an implementation of the general ordering.

* This research was supported in part by the U. S. National Science Foundation under Grants CCR-90-07195 and CCR-90-24271. The first author was also supported by a Lady Davis fellowship at the Hebrew University of Jerusalem.

We also look at methods of proving termination of *orthogonal* (left-linear non-overlapping) systems and related issues. These may be compared with ordinary structural induction proofs used for recursively-defined functions; see [Burstall, 1969; Manna, 1974]. In particular, we give a solution to a problem posed by Zantema [personal communication].

2 Path orderings

We use quasi-orderings (reflexive-transitive binary relations) to prove termination of rewrite systems. A quasi-ordering is *well-founded* if it has no infinite strictly descending sequences of elements. A *precedence* is a well-founded quasi-ordering of function symbols. An ordering might be called *syntactic* if it is based on a precedence and is invariant under shifts of symbols. In other words, we require that consistently replacing function symbols in two terms with others of the same arity and with the same relative ordering has no effect on the ordering of the two. The recursive path orderings [Dershowitz, 1982; Kamin and Lévy, 1980; Lescanne, 1990] are syntactic; the Knuth-Bendix and polynomial orderings are not.

The rule

$$x \times (y + z) \quad \rightarrow \quad (x \times y) + (x \times z) \tag{1}$$

is terminating. This can be shown by considering the multiset of "natural" interpretations of all products in a term, letting $+$ and $\times$ stand for addition and multiplication, and assigning some fixed value to constants; see [Dershowitz and Manna, 1979] for similar examples. Syntactic "path" orderings (see [Dershowitz, 1987]) work in this case, too. Lipton and Snyder [1977] gave a method for proving termination with interpretations (order-isomorphic to ω) for which rules are "value-preserving", as this example is for the natural interpretation.

Consider the following contrived system for computing factorial in unary arithmetic (expanding on one in [Kamin and Lévy, 1980]):

$$\begin{array}{rcl} p(s(x)) & \rightarrow & x \\ fact(0) & \rightarrow & s(0) \\ fact(s(x)) & \rightarrow & s(x) \times fact(p(s(x))) \\ 0 \times y & \rightarrow & 0 \\ s(x) \times y & \rightarrow & (x \times y) + y \\ x + 0 & \rightarrow & x \\ x + s(y) & \rightarrow & s(x + y)\ . \end{array} \tag{2}$$

It would be nice were we able to use a natural interpretation, but that does not prove termination, since the rules preserve the value of the interpretation, rather than cause a decrease. Nor can we use multisets of the values of the argument of $fact$, since some rules can multiply occurrences of that symbol. Though path orderings [Dershowitz, 1987] have been successfully applied to many termination proofs, they suffer from the same limitation as do all simplification orderings: they are not useful when a rule embeds as does $fact(s(x)) \rightarrow s(x) \times fact(p(s(x)))$.

What is needed is a way of combining the semantics given by a natural interpretation with a non-simplification ordering that takes the structure of terms into account.

Definition 1 (Termination Function). A *termination function* τ takes a term as argument and is of one of the following types:

a. a function that returns the outermost function symbol of a term to be compared using a precedence;
b. a homomorphism from terms to some well-founded set of values (that is, $\tau(f(s_1, \ldots, s_n)) = f_\tau(\tau(s_1), \ldots, \tau(s_n))$, for each function symbol f);
c. a monotonic homomorphism from terms to some well-founded set with the strict subterm property $(f_\tau(\ldots x \ldots)) > x)$ (a homomorphism is *monotonic* with respect to the given ordering $\geq$ if $f_\tau(\ldots x \ldots) \geq f_\tau(\ldots y \ldots)$ whenever $x > y$; it has the *strict subterm property* if $f_\tau(\ldots x \ldots) > x$);
d. a strictly monotonic homomorphism from terms to some well-founded set which has the strict subterm property (it is *strictly monotonic* if $f_\tau(\ldots x \ldots) > f_\tau(\ldots y \ldots))$ whenever $x > y$);
e. a function that extracts the immediate subterm at a specified position (which position can depend on the outermost function symbol of the term);
f. a function that extracts the immediate subterm of a specified rank (the kth largest in the path ordering defined recursively below); or
g. a constant function.

Simple examples of homomorphisms from terms to the natural numbers are size (number of function symbols, including constants), depth (maximum nesting of function symbols), and weight (sum of weights of function symbols). Size and weight are strictly monotonic; depth is monotonic. (The subterm property is guaranteed for strictly monotonic homomorphisms into well-ordered sets [Dershowitz, 1982].)

Definition 2 (General Path Ordering). Let $\tau_0, \ldots, \tau_n$ be termination functions. The induced *path ordering* $\succ$ is as follows:

$$s = f(s_1, \ldots, s_m) \succeq g(t_1, \ldots, t_n) = t$$

if either of the following cases (1 or 2) hold:

(1) $s_i \succeq t$ for some s_i, $i = 1, \ldots, m$; or
(2) $s \succ t_1, \ldots, t_n$ and $\langle \tau_1(s), \ldots, \tau_k(s) \rangle$ is lexicographically greater than or equal to $\langle \tau_1(t), \ldots, \tau_k(t) \rangle$, where function symbols are compared according to their precedence, homomorphic images are compared in the corresponding well-founded ordering, and subterms are compared recursively in $\succ$.

As usual, $s \succ t$ if $s \succeq t$, but $s \not\preceq t$.

Lemma 3. *The path ordering satisfies the strict subterm property $f(\ldots, s_i, \ldots) \succ s_i$, for all i.*

Proof. By (1) $f(\ldots, s_i, \ldots) \succeq s_i$, but $s_i \not\succeq f(\ldots, s_i, \ldots)$, since the first part of (2) cannot hold for the ith subterm on the right. □

Thus, the ordering is strict when Case (1) applies, or, for Case (2), if the lexicographic comparison is strictly greater.

Lemma 4. *For the path ordering, $s \succeq t$ implies $s \succ t|_i$ for each proper subterm $t|_i$ of t and implies $u[s] \succ t$ for each immediately enclosing context $u[\cdot]$ of s.*

Lemma 5. *The path ordering is a quasi-ordering.*

Proof. Reflexivity is an easy induction. For transitivity, we show that $s \succeq t \succeq u$ implies $s \succeq u$ and that $s \succeq t \succ u$ or $s \succ t \succeq u$ implies $s \succ u$, simultaneously, by induction on the size of the three terms and a case analysis. This requires the preceding lemma. □

Theorem 6. *Let $\tau_0, \ldots, \tau_{i-1}$ ($i \geq 0$) be monotonic homomorphisms, all but possibly the last strict, and let $\tau_i, \ldots, \tau_k$ be any other kinds of termination functions. A rewrite system terminates if $l\sigma \succ r\sigma$ in the corresponding path ordering $\succ$ for all rules $l \rightarrow r$ and ground substitutions σ, and also $\tau(l\sigma) = \tau(r\sigma)$ for each of the non-monotonic homomorphisms among the τ_i.*

The proof of this theorem is akin to [Kamin and Lévy, 1980] and uses a minimal counter-example argument.

Proof. First we show that

$$s \rightarrow t \text{ and } s \succ t \ \text{ imply } \ f(\ldots, s, \ldots) \succeq f(\ldots, t, \ldots) \ ,$$

for all terms s, t, ... and function symbols f. Then, $l\sigma \succ r\sigma$ will imply a decrease with each rewrite.

Other than for the monotonic homomorphisms, we have $\tau_i(f(\ldots, s, \ldots)) \geq \tau_i(f(\ldots, t, \ldots))$: For τ, a precedence, value-preserving homomorphism, specified subterm, or constant, $s \rightarrow t$ clearly implies $\tau(f(\ldots, s, \ldots)) \geq \tau(f(\ldots, t, \ldots))$ in the relevant ordering. For a τ that extracts the kth largest subterm u of $f(\ldots, s, \ldots)$: if $u \succ s$ or $t \succ u$, then replacing s by t has no impact on rank k and $\tau(f(\ldots, s, \ldots)) = u = \tau(f(\ldots, t, \ldots))$; if $s \succeq u \succeq t$, then $\tau(f(\ldots, s, \ldots)) \succeq \tau(f(\ldots, t, \ldots))$.

Let $s \succ t$ because the τ_i for some subterm $s|_p$ of s are lexicographically greater than for t. If the first point of difference between the τ_i is a strict homomorphism, then this (with the subterm property) implies a strict decrease $\tau_i(f(\ldots, s, \ldots)) \succ \tau_i(f(\ldots, t, \ldots))$ and, therefore, $f(\ldots, s, \ldots) \succ f(\ldots, t, \ldots)$. If it's at the non-strict homomorphism, then $\tau_i(f(\ldots, s, \ldots)) \succeq \tau_i(f(\ldots, t, \ldots))$ and $f(\ldots, s, \ldots) \succeq f(\ldots, t, \ldots)$.

To prove well-foundedness of $\succ$, consider a minimal infinite descending sequence $t_1 \succ t_2 \succ \cdots$, minimal in the sense that from all proper subterms of each term in the example there are only finite descending sequences. (By the subterm property, if $t_j \succ t_{j+1}$ then t_j is also greater than the subterms of t_j.) Case (1) of the definition of $\succ$ could not be the justification for any pair $t_j \succ t_{j+1}$, since then we would have $t_{j-1} \succ t_j|_i \succ t_{j+2}$, for some proper subterm $t_j|_i$ of the jth term in the example, and the example would not be minimal. Since Case (2) uses a lexicographic combination of well-founded orderings (including $\succ$ on proper subterms), it, too, is well-founded, and the descending sequence could not be infinite. □

For System (2), let τ_0 interpret everything naturally: *fact* as factorial, s as successor, p as predecessor, $\times$ as multiplication, $+$ as addition, and 0 as zero. Let all constants be interpreted as natural numbers, making all terms non-negative. Let the precedence τ_1 be $fact \succ \times \succ + \succ s$. Each rule causes a strict decrease with respect to $\succ$.

One must also make sure that all terms and subterms in any derivation are interpretable as natural numbers; otherwise a rule like $fact(x) \rightarrow fact(p(x))$ would give pretense of being terminating.

The following orderings are special cases of the general path ordering. For all but one, the conditions of the theorem hold:

Knuth-Bendix ordering [Knuth and Bendix, 1970]. τ_0 gives the sum of (non-negative integer) "weights" of the function symbols appearing in a term; τ_1 gives a (total) precedence; $\tau_2, \ldots, \tau_{n+1}$ give a permutation of the subterms.

Polynomial path ordering [Lankford, 1979]. τ_0 is a strictly monotonic homomorphism (each f_τ is a polynomial with positive coefficients); τ_1 gives a precedence; $\tau_2, \ldots, \tau_{n+1}$ give a permutation of the subterms.

Multiset path ordering [Dershowitz, 1982]. τ_0 is a total precedence; $\tau_1, \ldots, \tau_n$ give the subterms in non-increasing order. (The multiset path ordering is also defined for partial precedences; that would require comparing the τ_i as a multiset, rather than lexicographically .)

Lexicographic path ordering [Kamin and Lévy, 1980]. τ_0 is a precedence; $\tau_1, \ldots, \tau_n$ give a permutation of the subterms.

Semantic path ordering [Kamin and Lévy, 1980; Plaisted, 1979]. τ_0 is the identity function (a non-monotonic homomorphism), with terms compared in some well-founded ordering; τ_1 gives a precedence; $\tau_2, \ldots, \tau_{n+1}$ give a permutation of the subterms. *(For this ordering, one must separately insure that $s \rightarrow t$ implies $\tau_0(s) \geq \tau_0(t)$.)*

Recursive path ordering [Lescanne, 1990]. τ_0 is a total precedence; $\tau_1, \ldots, \tau_n$ give a permutation of the subterms or give the subterms in non-increasing order, depending on the function symbol.

Extended Knuth-Bendix ordering [Dershowitz, 1982; Steinbach and Zehnter, 1990]. τ_0 is a monotonic interpretation; τ_1 gives a precedence; $\tau_2, \ldots, \tau_{n+1}$ give the subterms in order, permuted, or sorted, depending on the function symbol.

For a system like

$$\begin{array}{rcl} f(s(x)) & \rightarrow & s(h(d(f(x)))) \\ f(0) & \rightarrow & 0 \\ d(0) & \rightarrow & 0 \\ d(s(x)) & \rightarrow & s(s(d(x))) \\ h(s(s(x))) & \rightarrow & s(h(x)) \,, \end{array} \qquad (3)$$

a precedence ($f > h > d > s$) ought to be considered first, before looking at subterms, as with a lexicographic path ordering. In a system like

$$\begin{aligned} f(s(x)) &\rightarrow p(s(f(f(x)))) \\ f(0) &\rightarrow 0 \\ p(s(x)) &\rightarrow x\ , \end{aligned} \tag{4}$$

with nested defined symbols on the right, an interpretation ($f_\tau(x) = 0, s_\tau(x) = x+1, p_\tau(x) = x-1$) could be considered first, followed by a precedence ($f > s, p$), as with an extended Knuth-Bendix ordering. (With $f(0) \rightarrow s(0)$, instead of 0, the system would be nonterminating.)

In the appendix, we describe how an implementation of this ordering performs on a sorting example.

3 Orthogonal systems

Consider a recursive definition like

$$f(x) = \text{if } x > 0 \text{ then } f(f(x-1)) + 1 \text{ else } 0\ .$$

By a straightforward use of structural induction, one can prove that the least fixpoint (over the natural numbers) is the always-defined identity function. This definition translates into the rewrite system:

$$\begin{aligned} f(s(x)) &\rightarrow s(f(f(p(s(x))))) \\ f(0) &\rightarrow 0 \\ p(s(x)) &\rightarrow x\ . \end{aligned} \tag{5}$$

It would be nice to be able to mimic the proof for the recursive function definition in the rewriting context, but several issues arise:

1. One cannot use a syntactic simplification ordering like the simple path ordering [Plaisted, 1978], since the first rule is embedding. In fact, we must combine termination with the semantics ($f(x) = x$), as one must for the functional proof.
2. In the functional case, one can show that call-by-value terminates, which implies that all fixpoint computation rules also terminate. We will see under what conditions the same holds for rewriting.
3. For rewriting in general, one must consider the possibility that the x in the definition of $f(x)$ us itself a term containing occurrences of the defined function f (or of mutually recursive defined functions), something usually ignored in the functional case.

Consider the system:

$$\begin{aligned} f(s(x)) &\rightarrow s(f(p(s(x)))) \\ f(0) &\rightarrow 0 \\ p(s(x)) &\rightarrow x\ . \end{aligned} \tag{6}$$

The general path ordering works with a natural interpretation of the argument of f and a precedence $f > s, p$.

Alternatively, one can employ the following result:

Proposition 7 [O'Donnell, 1977]. *A non-erasing orthogonal system is terminating if and only if every term has a normal form.*

Therefore, the offending rule may be immediately followed by an application of the last rule, effectively replacing the former with $f(s(x)) \rightarrow s(f(x))$. Now termination can be shown with a standard recursive path ordering, demonstrating that the orginal system is normalizing, and, hence, terminating.

This method does not apply to a system like

$$\begin{array}{rcl} x \times 0 & \rightarrow & 0 \\ x \times s(y) & \rightarrow & (x \times y) + x \\ x + 0 & \rightarrow & x \\ x + s(y) & \rightarrow & s(x + y) \ , \end{array} \tag{7}$$

with its erasing rule (the first one).

Still, we can employ the following:

Proposition 8 [Gramlich, 1992]. *A locally confluent overlaying system is terminating if and only if innermost rewriting always leads to a normal form.*

An *overlaying* system is one whose only critical pairs are obtained from an overlap at the topmost position. In particular, orthogonal systems are locally confluent and have no (non-trivial) critical pairs; the proposition for this case was shown in [O'Donnell, 1977].

We turn now to the question of when termination of ground constructor instances of left-hand sides suffices for establishing termination in all cases.

Definition 9 [Dershowitz, 1981]. The set of *forward closures* for a given rewrite system is inductively defined as follows:

- Every rule $l \rightarrow r$ is a forward closure.
- If $c \rightarrow c'$ and $d \rightarrow d'$ are forward closures such that $c' = u[s]$ for nonvariable s and $s\mu = d\mu$ for most general unifier μ, then $c\mu \rightarrow u\mu[d'\mu]$ is also a forward closure.

The idea is to restrict application of rules to that part of a term created by previous rewrites. In the same way, we can define *innermost* and *outermost* forward closures—restricting the position at which unification is performed so that the derivations captured by closure are of the desired type.

Proposition 10 [Geupel, 1989]. *A non-overlapping rewrite system is terminating if, and only if, no right-hand side of a forward closure initiates an infinite derivation.*

In general, though, a term-rewriting system need not terminate even if all its forward closures do [Dershowitz, 1981].

Consider the following system for symbolic differentiation with respect to t (proving termination of the first five of these rules was one of the problems on a qualifying exam given at Carnegie-Mellon University in 1967):

$$
\begin{array}{rcl}
D_t\, t & \to & 1 \\
D_t\, a & \to & 0 \\
D_t\,(x+y) & \to & D_t\, x \;+\; D_t\, y \\
D_t\,(x \cdot y) & \to & y \cdot D_t\, x \;+\; x \cdot D_t\, y \\
D_t\,(x-y) & \to & D_t\, x \;-\; D_t\, y \\
D_t\,(-x) & \to & -D_t\, x \\
D_t\,(x/y) & \to & D_t\, x/y \;-\; x \cdot D_t\, y/y^2 \\
D_t\,(\ln\, x) & \to & D_t\, x/x \\
D_t\,(x^y) & \to & y \cdot x^{y-1} \cdot D_t\, x \;+\; x^y \cdot (\ln x) \cdot D_t\, y\;,
\end{array}
\tag{8}
$$

where a is any constant symbol other than t. It is orthogonal (hence, non-overlapping), so the above method applies. Since D's are not nested on the right, forward closures cannot have nested D's. Since the arguments to D on the left are always longer than those on the right, all forward closures must lead to terminating derivations; hence, regardless of the rewriting strategy and initial term, rewriting terminates.

For a system like

$$
\begin{array}{rcl}
f(s(x)) & \to & s(s(f(p(s(x))))) \\
f(0) & \to & 0 \\
p(s(x)) & \to & x\;,
\end{array}
\tag{9}
$$

we can also restrict our attention to forward closures. Since f's won't nest, termination can be shown by comparing the argument on the left, $s(x)$, with the one on the right, $p(s(x))$. This time we need to use a semantic comparison, making the left argument always larger.

Theorem 11. *A locally-confluent overlaying rewrite system is terminating if, and only if, no right-hand side of an innermost forward closure initiates an infinite derivation.*

In particular, orthogonal systems satisfy the prerequisites for application of this termination test; one need only prove termination of such innermost derivations.

The proof is similar to [Geupel, 1989]:

Proof. Consider a minimal example of nontermination $t_1 \to t_2 \to \cdots$, minimal in the sense that at each point any rewrite lower down in the term than the redex in the example would have to lead to a normal form. Replace the largest terminating subterms of each t_i with their unique normal form (which they have by local confluence). The fact that all overlaps occur at the top ensures that none of these replacements prevents application of a rule above the replaced terms. Hence, the result is an infinite derivation with the desired characteristics. □

This method applies to Systems 2 and 5: Since we need only consider innermost derivations, we can assume that the problematic $p(s(x))$ on the right rewrites immediately to x (and that the x is in normal form).

Suppose an orthogonal system is constructor-based, that is, all proper subterms of left-hand sides have only free constructors and variables. All its forward closures begin with constructor-based instances of left-hand sides. Thus, termination proofs need not consider initial terms containing nested defined function symbols (even when the symbol is not completely defined). That makes proving termination of such systems no more difficult than proving termination of ordinary recursive functions: the instances of rule variables can be presumed to be in normal form and the context can be ignored.

For System 5, say, we can compare the multiset of right-hand side arguments of the (mutually-)recursive function symbols $\{f(p(s(x))), p(s(x))\}$ with that of left-hand side, $\{s(x)\}$. Semantics are necessary for this comparison. If we let $p(s(x)) \rightarrow x$ and $f(x) \rightarrow x$, we have $\{s(x)\}$ greater (in the multiset ordering) than $\{x, x\}$. But one must ensure that the semantics are consistent with the rules (which is analogous to showing that $f(x) = x$ is a fixpoint of the definition). This can be done using standard rewriting technique ("proof by consistency").

It is instructive to compare the above examples with the following nonterminating rewrite system:

$$\begin{array}{rcl} f(s(x)) & \rightarrow & s(s(f(f(p(s(x)))))) \\ f(0) & \rightarrow & 0 \\ p(s(x)) & \rightarrow & x\,. \end{array} \qquad (10)$$

It is the rewriting analogue of the recursively-defined function

$$f(x) = \text{if } x > 0 \text{ then } f(f(x-1)) + 2 \text{ else } 0\,,$$

which does not terminate for 2. Indeed, $f(x) = x$ would be inconsistent with the rules.

The above results can be used to prove termination of systems that can be decomposed into two terminating systems that do not share defined symbols.

Proposition 12 [Dershowitz, 1993]. *Let R contain defined symbols and free constructors, and S contain defined symbols from a disjoint set of defined symbols and from the same set of constructors. If R and S are each non-overlapping and terminating, then so is their union.*

4 String rewriting

Proposition 13 [Dershowitz, 1981]. *A right-linear rewrite system is terminating if, and only if, no right-hand side of a forward closure initiates an infinite derivation.*

In particular, forward closures suffice for string-rewriting systems. String systems are also non-erasing.

Zantema's Problem (circulated via electronic mail) is to prove termination of the following one-rule string-rewriting system:

$$1100 \rightarrow 000111\,. \qquad (11)$$

It provides a nice example of termination proofs based on an analysis of restricted derivations.

Suppose it is nonterminating. Consider a minimal infinite derivation

$$t_1 \rightarrow t_2 \rightarrow \cdots\,,$$

minimal in the sense that no substring of t_1 is nonterminating and there is no infinite derivation from t_1 taking place at higher positions (further left). More specifically, $t_1 \rightarrow t_2$ takes place at the top (leftmost symbol) and among all infinite derivations beginning $t_1 \rightarrow t_2 \rightarrow \cdots \rightarrow t_i$, none starts higher than does $t_i \rightarrow t_{i+1} \rightarrow \cdots$.

Divide each string t_i into three parts (from left to right): dead, active, and passive. The dead part never develops a redex; the passive part is a residual substring of the initial string which has not yet been touched; the active part contains letters introduced by right-hand sides. The dead part is in normal form and for (11) always ends in 000.

To start off, t_1 is all passive, except for its first letter. This minimal derivation must be leftmost (outermost). Suppose this were not the case. Either the outer redex is eventually rewritten, or it never is. In the former case, the derivation

$$t_1 \rightarrow \cdots \rightarrow u1100v \rightarrow u1100v' \rightarrow \cdots \rightarrow u1100v'' \rightarrow u000111v'' \rightarrow \cdots ,$$

where $v \rightarrow v' \rightarrow \cdots \rightarrow v''$, can be rearranged to

$$t_1 \rightarrow \cdots \rightarrow u1100v \rightarrow u000111v \rightarrow u000111v' \rightarrow \cdots \rightarrow u000111v'' \rightarrow \cdots ,$$

and, therefore, is not minimal. In the latter case, rewriting the outer redex doesn't preclude nontermination, and the smaller alternative is also nonterminating.

Similarly, redexes are always in the active part. For suppose the minimal derivation did have some steps in the passive part. There would have to be a subsequent step in the active part (or else that passive proper substring of t_1 would be nonterminating), which is perhaps enabled by the step in the passive part:

$$t_1 = sw \rightarrow \cdots \rightarrow uvw \rightarrow uvw' \rightarrow \cdots \rightarrow uvw'' \rightarrow uv'w'' \rightarrow \cdots ,$$

where u is dead, v is active, and w is passive. Since the alternate derivation

$$sw'' \rightarrow \cdots \rightarrow uvw'' \rightarrow uv'w'' \rightarrow \cdots$$

(starting out after the rewriting of the passive part) is smaller (the v redex is higher up than the w one), the given derivation can not be minimal.

More generally:

Proposition 14. *A non-erasing orthogonal system terminates if and only if no right-hand side of an outermost forward closure initiates an infinite derivation.*

System (9) is of this form (all its forward closures are outermost anyway.)

For this specific system, we need only consider three active parts: 111, 1110111, or 1111110111, since it takes only finitely many steps to get from one of these to another. Call these states A, B, and C, respectively.

For there to be a redex in the active part, the passive part must begin with 00 or with 100. The leftmost derivations (with redex underlined, and dead parts bracketed) of the six cases are shown in Fig. 1. In each case, termination follows from the fact that the passive part decreases in size.

The same approach works for other examples of the form $1^i0^j \rightarrow 0^k1^l$.

$A00 = 1\underline{1100} \rightarrow [1000]111 \in A$

$B00 = 11101\underline{1100} \rightarrow [11101000]111 \in A$

$C00 = 11111101\underline{1100} \rightarrow [11111101000]111 \in A$

$A100 = 11\underline{1100} \rightarrow \underline{1100}0111 \rightarrow [000]1110111 \in B$

$B100 = 111011\underline{1100} \rightarrow 11101\underline{1100}0111 \rightarrow 1\underline{1100}001110111 \rightarrow$
$[1000]1\underline{1100}1110111 \rightarrow [10001000]1111110111 \in C$

$C100 = 111111011\underline{1100} \rightarrow 1111110\underline{1100}0111 \rightarrow 1111\underline{1100}001110111 \rightarrow$
$11\underline{1100}01110 01110111 \rightarrow \underline{1100}0111011100 1110111 \rightarrow$
$[000]111011101\underline{1100}1110111 \rightarrow [000111011101000]1111110111 \in C$

Fig. 1. Derivations for Zantema's problem

Appendix

Our general path ordering termination code (GPOTC) is implemented in Common Lisp on a Macintosh. (No special features of Macintosh Common Lisp were used, so the code should be capable of running under any Common Lisp with just a few minor changes.)[2] The implementation supports termination functions for precedence, term extraction (given, minimum, and maximum), and homomorphisms.

Interpretations involving addition, multiplication, negation, and exponentiation are expressible. Currently, the burden of proving that functions are either value-preserving or monotonic is placed on the user. As is usual for such functions, one often ends up needing to know if a given function is positive over some range. When the functions are rational polynomials, this is decidable, but time consuming. Our code does not attempt a full solution, but merely applies some quick and dirty heuristics, such as testing the function at endpoints and checking coefficients of polynomials. In cases where the code cannot make a determination, it will query the user for an authoritative answer. The part of the code that does this testing could be upgraded to provide heuristics such as those described in [Lankford, 1979; Ben Cherifa and Lescanne, 1987; Steinbach and Zehnter, 1990]. We are also in the process of implementing Paul Cohen's decision procedure [Cohen, 1969] for the first-order theory of real polynomials within Mathematica®.

The following brief example shows the use of GPOTC. The rewrite rules in Fig. 2 are an implementation of insertion sort over the natural numbers. The function `choose` is used to determine whether `X` should be inserted before or after the first element of the list which is the second argument to `insert`. Rule 2, for example, would be defined for the system as follows:

---

[2] Those interested in obtaining a copy of GPOTC should send electronic mail to `hoot@cs.uiuc.edu`.

```
(setf ins2 (make-production :lhs '(!Sort (!Cons ?X ?Y))
                            :rhs '(!Insert ?X (!Sort ?Y)))) .
```

The characters "!" and "?" are macro symbols indicating symbols and variables, respectively.

```
RULE 1:  sort(nil)  -->  nil
RULE 2:  sort(cons(X, Y))  -->  insert(X, sort(Y))
RULE 3:  insert(X, nil)  -->  cons(X, nil)
RULE 4:  insert(X, cons(V, W))  -->  choose(X, cons(V, W), X, V)
RULE 5:  choose(X, cons(V, W), 0, 0)  -->  cons(X, cons(V, W))
RULE 6:  choose(X, cons(V, W), s(P), 0)  -->  cons(X, cons(V, W))
RULE 7:  choose(X, cons(V, W), 0, s(Q))  -->  cons(V, insert(X, W))
RULE 8:  choose(X, cons(V, W), s(P), s(Q))  -->  choose(X, cons(V, W), P, Q)
```

Fig. 2. Rules for insertion sort.

The code for creating the ordering is

```
(setf SymOrd1 '(!Sort !Insert !Choose !Cons))
(setf SymOrd2 '(!Sort (!Insert !Choose) !Cons))
(makeorder ord1
  (list
    (make_prec_tau SymOrd2)
    (make_subterm_tau ((!Sort 1) (!Choose 2) (!Insert 2)) ord1)
    (make_prec_tau SymOrd1)
    (make_subterm_tau ((!Sort 1) (!Choose 3) (!Insert 2)) ord1)
    ))
```

Three termination functions are used; they are lexicographically compared from first to last. The macro `make_prec_tau` creates a precedence ordering based on its argument; `make_subterm_tau ((f n) ...) ord1` extracts the nth subterm for function symbol f and compares it using the ordering `ord1`. The `makeorder` macro creates a function with the name of the first argument which accepts two terms (s and t) and may return one of three values: `Ge` ($s \succeq t$), `Gr` ($s \succ t$), or `Un` (unknown).

If one uses a precedence ordering based on `SymOrd1`, all of the rules except for Rule 7 would be oriented in the appropriate direction. Unfortunately, Rules 4 and 7 interact with each other. In particular, there is a `choose` and an `insert` on opposite sides of each rule. The precedence order `SymOrd2` with (`sort` $\succ$ `insert = choose` $\succ$ `cons`) is chosen to guarantee that the lexicographical ordering of the terms in Rule 7 is from left to right, while leaving Rule 4 equal. This means that the left-hand side of Rule 7 is compared with each of the two subterms on the right. The comparison of interest is `choose(X, cons(V, W), 0, s(Q))` with `insert(X, W)`. These terms are equal under the precedence ordering `SymOrd2`, but by selecting the second subterm, the subterms `cons(V, W)` and `W` are recursively compared giving the necessary decrease. Fortunately, the second subterm on both sides of Rule 4 is identical, leaving the lexicographical ordering unaffected. The precedence ordering `SymOrd1` with (`sort` $\succ$ `insert` $\succ$ `choose` $\succ$ `cons`) breaks the tie, and all that remains is to verify that the left-hand side of Rule 4 is greater than the subterms on the right.

The code in Fig. 3 shows an example of a monotonic homomorphism where $F_f(X) = 2X + 4$, $F_g(X, Y) = 3Y + 6$, $F_a = 0$ and $F_b = 1$. The macro `make-fn` accepts a list of symbols and their associated functions. Notice that the expressions are essentially the equivalent Lisp expressions with `(arg n)` giving the nth argument.

```
(setq example-FNtau
      (make-fn ((!f (+ (* 2 (arg 1)) 4))
         (!g (+ (* 3 (arg 2)) 6))
         (!a 0)
         (!b 1))))
```

Fig. 3. Example code for creating a function τ.

To apply the ordering function `ord` to each of the rules in the list `InsSort` (containing the six rules in Fig. 2), one issues the command

`(term-cond InsSort #'ord1)` ,

with the result

`(:GR :GR :GR :GR :GR :GR :GR :GR)` .

Figure 4 displays the justification for Rule 4. The system is able to determine that `insert(X, cons(V, W))` is greater than `choose(X, cons(V, W), X, V)` by first showing that `insert(X, cons(V, W))` is strictly greater than each of the subterms of the right-hand side. These sub-proofs (for `X`, `cons(V, W)`, and `V`) are all similar: a sub-term of the left-hand side is found to be syntactically equal to the right-hand side, and Case (1) of the path ordering applies. Showing the lexicographic part of the ordering comes next: one of the termination functions must show a strict increase. The first two do not result in a strict decrease (they are equal). The third, however, compares `insert` with `compare` in the precedence given by `SymOrd1` where there is the desired strict decrease. That concludes Case (2) of Definition 2, showing that `insert(X, cons(V, W))` is strictly greater than `choose(X, cons(V, W), X, V)`.

References

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

[Burstall, 1969] Robert M. Burstall. Proving properties of programs by structural induction. *Computing J.*, 12(1):41–48, February 1969.

```
(term-cond (list ins4) #'ord1 :keep-causes t)
((:GR
 insert(X, cons(V, W)) > choose(X, cons(V, W), X, V) by case (2)
 Case 2a: Check that the LHS > all subterms of the RHS:
 | insert(X, cons(V, W)) > X by case (1)
 | | X is syntactically equal to term X
 | |
 | insert(X, cons(V, W)) > cons(V, W) by case (1)
 | | cons(V, W) is syntactically equal to term cons(V, W)
 | |
 | insert(X, cons(V, W)) > X by case (1)
 | | X is syntactically equal to term X
 | |
 | insert(X, cons(V, W)) > V by case (1)
 | | cons(V, W) >= V by case (1)
 | | | V is syntactically equal to term V
 Case 2b: Check that the LHS > RHS via lexicographic comparison:
 | 1:insert(X, cons(V, W)) >= choose(X, cons(V, W), X, V) by basic ordering
                                                           of a precedence tau
 | |
 | 2:immediate subterms insert|2 with choose|2: cons(V, W) >= cons(V, W)
 | | cons(V, W) is syntactically equal to term cons(V, W)
 | |
 | 3:insert(X, cons(V, W)) > choose(X, cons(V, W), X, V) by basic ordering
                                                          of a precedence tau
))
```

Fig. 4. Proof for a single rule.

[Cohen, 1969] Paul J. Cohen. Decision procedures for real and p-adic fields. *Comm. Pure and Applied Math*, 22(2): 279–301, March 1969.

[Dershowitz, 1981] Nachum Dershowitz. Termination of linear rewriting systems. In *Proceedings of the Eighth International Colloquium on Automata, Languages and Programming*, pages 448–458, Acre, Israel, July 1981. European Association of Theoretical Computer Science. Vol. 115 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin.

[Dershowitz, 1982] Nachum Dershowitz. Orderings for term-rewriting systems. *Theoretical Computer Science*, 17(3):279–301, March 1982.

[Dershowitz, 1987] Nachum Dershowitz. Termination of rewriting. *J. of Symbolic Computation*, 3(1&2):69–115, February/April 1987. Corrigendum: *4*, 3 (December 1987), 409–410.

[Dershowitz, 1993] Nachum Dershowitz. Hierarchical termination. Technical Report, Leibnitz Center for Research in Computer Science, Hebrew University, Jerusalem, Israel.

[Dershowitz and Manna, 1979] Nachum Dershowitz and Zohar Manna. Proving termination with multiset orderings. *Communications of the ACM*, 22(8):465–476, August 1979.

[Geupel, 1989] Oliver Geupel. Overlap closures and termination of term rewriting systems. Report MIP-8922, Universität Passau, Passau, West Germany, July 1989.

[Gramlich, 1992] Bernhard Gramlich. Relating innermost, weak, uniform and modular termination of term rewriting systems. *Proceedings of the Conference on Logic Programming and Automated Reasoning*, pp. 285–296, St. Petersburg, Russia, July 1992. Vol. 624 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin.

[Kamin and Lévy, 1980] Sam Kamin and Jean-Jacques Lévy. Two generalizations of the recursive path ordering. Unpublished note, Department of Computer Science, University of Illinois, Urbana, IL, February 1980.

[Knuth and Bendix, 1970] Donald E. Knuth and P. B. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pp. 263–297. Pergamon Press, Oxford, U. K., 1970.

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

[Lescanne, 1990] Pierre Lescanne. On the recursive decomposition ordering with lexicographical status and other related orderings. *J. Automated Reasoning*, 6:39–49, 1990.

[Lipton and Snyder, 1977] R. Lipton and L. Snyder. On the halting of tree replacement systems. In *Proceedings of the Conference on Theoretical Computer Science*, pp. 43–46, Waterloo, Canada, August 1977.

[Manna, 1974] Zohar Manna. *Mathematical Theory of Computation*. McGraw-Hill, New York, 1974.

[Manna and Ness, 1970] Zohar Manna and Steven Ness. On the termination of Markov algorithms. In *Proceedings of the Third Hawaii International Conference on System Science*, pp. 789–792, Honolulu, HI, January 1970.

[O'Donnell, 1977] Michael J. O'Donnell. *Computing in systems described by equations*, volume 58 of *Lecture Notes in Computer Science*. Springer, Berlin, West Germany, 1977.

[Plaisted, 1978] David A. Plaisted. Well-founded orderings for proving termination of systems of rewrite rules. Report R-78-932, Department of Computer Science, University of Illinois, Urbana, IL, July 1978.

[Plaisted, 1979] David A. Plaisted. Personal communication, 1979.

[Steinbach and Zehnter, 1990] Joachim Steinbach and Michael Zehnter. Vade-mecum of polynomial orderings. Report SR-90-03, Fachbereich Informatik, Universität Kaiserslautern, Kaiserslautern, West Germany, 1990.