Natural Termination*

Nachum DERSHOWITZ and Charles HOOT

Department of Computer Science, University of Illinois at Urbana-Champaign, 1304 West Springfield Ave., Urbana, IL 61801–2987, U.S.A.

nachum,hoot@cs.uiuc.edu

Communicated by
Received
Revised

Abstract. Two techniques are examined for showing termination of rewrite systems when simplification ordering are insufficient. The first approach generalizes the various path orderings and the conditions under which they work. Examples of its use are given and a brief description of an implementation is presented. The second approach uses restricted derivations, called "forward closures", for proving termination of orthogonal and overlaying systems. Both approaches allow the use of "natural" interpretations under which rules rewrite terms to terms of the same value.

1. Introduction

Rewrite systems are sets of directed equations used to compute by repeatedly replacing terms in a given formula with equal terms, as long as possible. The theory of rewriting is an outgrowth of the study of the lambda calculus and combinatory logic, and has important applications in abstract data type specifications, functional programming, symbolic computation, and automated deduction. For surveys of the theory of rewriting, see Dershowitz and Jouannaud [9], Klop [17] and Plaisted [33].

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 (written, $s \rightarrow t$). As suggested by Manna and Ness [24], it is often convenient to express reduction orderings as a homomorphism from terms to an algebra equipped with a well-founded ordering. The use, in particular, of *polynomial interpretations* which map terms into the natural numbers was developed by Lankford [19]. For a survey of termination methods, see Dershowitz [7].

*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 and a Meyerhoff fellowship at the Weizmann Institute of Science. This is a revised and expanded version of *Topics in Termination* [8].

The rule

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

is terminating. This can be shown by interpreting $\times$ as multiplication, $+$ as $\lambda xy.x + y + 1$, and constants as 2. Since $x \geq 2$ implies $x(y + z + 1) > xy + xz + 1$, the rule is terminating. It can also be proved terminating 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 [10] for similar examples. Syntactic "path" orderings (see Dershowitz [7]) work in this case, too. Lipton and Snyder [22] gave a particular method for proving termination with interpretations (order-isomorphic to ω) for which rules are "value-preserving", as this example is for the natural interpretation.

Virtually all orderings used in practice are *simplification orderings* [6], satisfying the *replacement* property, that $s \succ t$ implies that any term containing s as a subterm is at least as large (under $\succ$) as the same term with s replaced by t, and the *subterm* property, that any term containing s is at least as large as s. Simplification orderings are surveyed by Steinbach [34]; their well-foundedness is a consequence of Kruskal's Tree Theorem. (See Dershowitz [6].) A *non-simple* rewrite system (such as $ffx \rightarrow fgfx$) is one for which no simplification ordering will show termination.

Knuth and Bendix [18] designed a particular class of well-orderings which assigns a weight to a term that 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. If they are headed by different symbols, a "precedence" ordering determines which term is larger. Another class of simplification orderings, the *path orderings* introduced in Dershowitz [6], 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 [16] to compare subterms lexicographically and to allow for a semantic component; see Dershowitz [7].

We use quasi-orderings (reflexive-transitive binary relations), rather than partial orderings, to prove termination of rewrite systems. If $\succsim$ is a quasi-ordering and $\precsim$ is its inverse, then its strict part $\succ$ ($\succsim - \precsim$) is a partial order. Its associated equivalence relation $\approx$ is defined as $\succsim \cap \precsim$. 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 can 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 [6, 16, 21] are syntactic; the Knuth-Bendix and polynomial orderings are not.

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. For example, consider the following contrived system for computing factorial in unary arithmetic (expanding on one in Kamin and Lévy [16]):

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

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 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 $fact(s(x)) \to s(x) \times fact(p(s(x)))$.

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. To that end, in Section 2, we present the general path ordering and prove that it is a quasi-order. In Section 3 we use the general path ordering to generalize all the above-mentioned orderings and the conditions under which they work so that they can also handle some non-simple systems. Examples, special cases, and a brief description of an implementation of the general ordering are included in Section 4.

We also look at methods of proving termination of *orthogonal* (left-linear non-overlapping) systems, such as (2), and related issues in Sections 5 and 6. These may be compared with ordinary structural induction proofs used for recursively-defined functions; see Burstall [3] and Manna [23]. In particular, we employ the notion of restricting the set of forward closures (Dershowitz [5]) to those conforming with some particular rewrite strategy, and give conditions under which the restricted set suffices.

2. The General Path Ordering

The general path ordering combines mappings from terms to well-founded sets.

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

a. a homomorphism from terms to an algebra (set of values) $\mathcal{A}$, where $\theta(f(s_1, \ldots, s_n)) = f_\theta(\theta(s_1), \ldots, \theta(s_n))$, and f_θ is a function from $\mathcal{A}^n$ to $\mathcal{A}$ for n-ary function symbol f;

b. an extraction function from terms to multisets of selected immediate subterms, that is $\theta(f(s_1, \ldots, s_n)) = \{s_{j_1}, \ldots, s_{j_m}\}$, such that $j_1, \ldots, j_m \in \{1, \ldots, n\}$ where the choice of the subterms depends on the function symbol f.

Definition 2 (Component Order). Let $\mathcal{T}$ be a set of variable-free terms (over some alphabet). A *component order* $\phi = \langle \theta, \geq \rangle$ consists of a termination function $\theta : \mathcal{T} \to \mathcal{A}$, from terms to an algebra $\mathcal{A}$ along with an associated well-founded quasi-order $\geq$ over $\mathcal{A}$.

The following definitions are useful ($\simeq$ denotes the equivalence part of $\geq$):

- A homomorphism θ is *value-preserving* with respect to the ordering $\geq$ and rewrite system $\mathcal{R}$ if $\theta(l\sigma) \simeq \theta(r\sigma)$ for all rules $l \to r$ in $\mathcal{R}$ and ground substitutions σ.

- A homomorphism θ is *monotonic* with respect to the ordering $\geq$ if for all function symbols f, $f_\theta(\ldots x \ldots) \geq f_\theta(\ldots y \ldots)$ whenever $x > y$.

- A homomorphism θ is *strictly monotonic* with respect to the ordering $\geq$ if for all function symbols f, $f_\theta(\ldots x \ldots) > f_\theta(\ldots y \ldots)$ whenever $x > y$.

- A homomorphism θ has the *strict subterm* property with respect to the ordering $\geq$ if for all function symbols f, $f_\theta(\ldots x \ldots) > x$.

- An equivalence relation $\simeq$ is a *congruence* with respect to a homomorphism θ if for all function symbols f, $x \simeq y$ implies $f_\theta(\ldots x \ldots) \simeq f_\theta(\ldots y \ldots)$.

- The multiset $R_i(S)$ of terms of rank i ($i \geq 0$) with respect to the ordering $\geq$ on terms in a multiset of terms S, is inductively defined as

$$R_i(S) = \{u \ : \ u \text{ is maximal with respect to } \geq \text{ in } L_i(S)$$

where

$$L_i(S) = S - \bigcup_{0<j<i} R_j(S).$$

Definition 3. Some important classes of component orders are:

a. $\langle \theta, \geq \rangle$ is a *precedence* when θ is a homomorphism which returns the outermost function symbol of a term and $\geq$ is a precedence ordering;

b. $\langle \theta, \geq \rangle$ is *value-preserving* when θ is a value-preserving homomorphism with respect to $\geq$ and $\geq$ is a well-founded quasi-order;

c. $\langle \theta, \geq \rangle$ is *monotonic* when θ is a monotonic homomorphism with the strict subterm property (with respect to $\geq$) and $\geq$ is a well-founded quasi-order;

d. $\langle \theta, \geq \rangle$ is *strictly monotonic* when θ is a strictly monotonic homomorphism with the strict subterm property (with respect to $\geq$) and $\geq$ is a well-founded quasi-order;

e. $\langle \theta, \geq \rangle$ is *multiset extracting* when θ is an extraction function which depending on the outermost function symbol returns a multiset of immediate subterms $\mathcal{IS}(t) = \{t_1, t_2, \ldots\}$ of a term t, of the following types:

   1. a multiset (including the empty multiset) containing the immediate subterms at specified positions $\mathcal{K}$ ($P_{\mathcal{K}}(t) = \{t_i : i \in \mathcal{K}\}$),
   2. a multiset containing the immediate subterms of rank k, $R_k(\mathcal{IS}(t))$, or
   3. a multiset containing the immediate subterms of rank k or less ($R_{\leq k}(\mathcal{IS}(t)) = \bigcup_{i=1}^{k} R_j(\mathcal{IS}(t))$)

   and $\geq$ is the multiset ordering $\succnsim_{\mathcal{M}}$ induced by a well-founded ordering $\succnsim$ on terms. (See Dershowitz and Manna [10] for more on multiset orderings.)

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. A simple example of a precedence uses the ordering $+ > s > 0$ with $+_\theta = \lambda x.\text{“+”}$, $s_\theta = \lambda x.\text{“s”}$, and $0_\theta = \lambda x.\text{“0”}$. (The subterm property is guaranteed for strictly monotonic homomorphisms into well-ordered sets [6].) An example of a multiset component ordering is $\theta = R_1$; it extracts the maximal immediate subterms in $\succ$. Another example is $\theta = P_{\{1\}}$ which gives the leftmost subterm.

Definition 4 (General Path Ordering). Let $\phi_0 = \langle \theta_0, \geq_0 \rangle$, ..., $\phi_k = \langle \theta_k, \geq_k \rangle$ be component orders, where for multiset extraction θ_x component orders, $\geq_x$ is the general path ordering $\succsim$ itself. The induced *general path ordering* $\succsim$ is defined as follows:

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

if either of the two following cases hold:

(1) $s_i \succsim t$ for some s_i, $i = 1, \ldots, m$, or

(2) $s \succ t_1, \ldots, t_n$ and $\Theta(s) >_{lex} \Theta(t)$, where $\Theta(s) = \langle \theta_0(s), \ldots, \theta_k(s) \rangle$, and $>_{lex}$ is the lexicographic combination of the component orderings $>_x$,

while

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

in the general path ordering if

(3) $s \succ t_1, \ldots, t_n$, $t \succ s_1, \ldots, s_m$ and $\theta_0(s) \simeq_0 \theta_0(t), \ldots, \theta_k(s) \simeq_k \theta_k(t)$.

Note that $\succsim$ is the union of $\succ$ and $\approx$, which are mutually recursive. Lemmas 6, 7, 8 and 9 (below) guarantee that $\succ$ is the strict part of $\succsim$, while $\approx$ is the equivalence part.

Lemma 1 (Symmetry). *If $s \approx t$ then $t \approx s$.*

Proof. This is trivial, since $\simeq_x$ is reflexive for the component quasi-orders $\geq_x$. When $\simeq_x$ is the multiset extension of $\approx$, induction on the combined size of the terms s and t is required. □

We use the notation $t|_p$ to refer to the subterm of t at position p and the notation $u[s]$ (or $u[s]_p$) to indicate that u contains s as a subterm (at position p).

Lemma 2. *For the general path ordering, $s \succsim t$ implies $s \succ t|_p$ for each proper subterm $t|_p$ of t.*

Proof. Assume that the lemma holds for any pair of terms smaller in combined size than $\langle s, t \rangle$.

Suppose $s \succ t$ by Case (1) of the ordering. Then for some i, $s_i \succsim t$. By the induction hypothesis, however, $s_i \succ t|_p$. We may then apply Case (1) resulting in $s \succ t|_p$.

Suppose $s \succsim t$ by Case (2) or (3). Thus we know $s \succ t_1, \ldots, t_n$. Suppose that $t|_p$ is a subterm of some t_i. Then, we can apply induction on the pair $\langle s, t_i \rangle$. □

The following two lemmata must be shown by simultaneous induction over the height of a term.

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

Proof. By inductive application of reflexivity (Lemma 4) to the subterm s_i we have $s_i \approx s_i$, and Case (1) applies. □

Lemma 4 (Reflexivity). *The general path ordering* $\succsim$ *is reflexive.*

Proof. Assume that $\succsim$ is reflexive for all terms with height less than k. Consider a term $f(t_1, \ldots, t_n)$ of height k. By the strict subterm property (Lemma 3) for terms of height k, $f(t_1, \ldots, t_n)$ is strictly greater than each of its subterms. Therefore the first and second conditions for equivalence are satisfied. Since each of the θ's is a function, $\theta_f(t_1, \ldots, t_n) \simeq_x \theta_f(t_1, \ldots, t_n)$ as long as each of the component orderings is reflexive. The only non-trivial case is the multiset ordering on immediate subterms. But by the induction hypothesis, for every subterm t_i, we have $t_i \succsim t_i$ and therefore the multiset ordering on immediate subterms is reflexive ($\{t_{j_1}, \ldots, t_{j_m}\} \approx_{\mathcal{M}} \{t_{j_1}, \ldots, t_{j_m}\}$ for any $j_1, \ldots, j_m \in \{1, \ldots, n\}$). Consequently, the third condition is satisfied and $f(t_1, \ldots, t_n) \approx f(t_1, \ldots, t_n)$. □

Lemma 5. *For the general path ordering,* $s \succsim t$ *implies* $u[s] \succ t$ *for each non-empty enclosing context* $u[\cdot]$ *of* s.

Proof. Consider the subterm $u|_p$ which contains s as an immediate subterm. By Case (1), $u|_p \succ t$. Repeated application of the preceding argument leads to $u[s] \succ t$. □

Lemma 6 (Transitivity). *For terms* s, t, *and* u *and general path ordering* $\succsim$:

(i) $s \succ t \succ u$ *implies* $s \succ u$;

(ii) $s \approx t \succ u$ *implies* $s \succ u$;

(iii) $s \succ t \approx u$ *implies* $s \succ u$;

(iv) $s \approx t \approx u$ *implies* $s \approx u$.

Proof. The proof proceeds by induction over the triple of terms $\langle s, t, u \rangle$ with respect to the sum of the heights of the three terms.

(i) Suppose that $s \succ t$ by Case (1) of the ordering, then $s_i \succsim t$ for some i. Now if $t \succ u$, we can apply induction on the triple $\langle s_i, t, u\rangle$ to get $s_i \succ u$. By Case (1) of the ordering we have $s \succ u$.

Suppose that $s \succ t$ by Case (2) of the ordering, then $s \succ t_1, \ldots, t_n$ and $\Theta(s) \geq_{lex} \Theta(t)$. Now if $t \succ u$ by Case (1) of the ordering, then $t_j \succ u$ for some j. But we may apply induction to the triple $\langle s, t_j, u\rangle$ to show $s \succ u$. If $t \succ u$ by Case (2) of the ordering, then $t \succ u_1, \ldots, u_m$ and $\Theta(t) \geq_{lex} \Theta(u)$. We may apply induction to each of the triples $\langle s, t, u_k\rangle$ to show that $s \succ u_k$ for each k. If each of the component orders is transitive then $\Theta(s) \geq_{lex} \Theta(u)$. When $\geq_x$ is a well-founded quasi-order there is no problem; when $\geq_x$ is a multiset ordering on immediate subterms, the induction hypothesis is needed.

(ii) We know that $s \succ t_1, \ldots, t_m$ and $\Theta(s) \simeq_{lex} \Theta(t)$.

Suppose that $t \succ u$ by Case (1) of the ordering, then $t_i \succsim u$ for some i. By induction on the triple $\langle s, t_i, u\rangle$, we have $s \succ u$.

Suppose that $t \succ u$ by Case (2) of the ordering, then $t \succ u_1, \ldots, t_m$ and $\Theta(t) \geq_{lex} \Theta(u)$. But for each triple $\langle s, t, u_k\rangle$ we have $s \succ u_k$. To show $s \succ u$, we only need to demonstrate the second condition of Case (2). But this holds for the quasi-orders and multiset orders by induction.

(iii) Essentially the same argument as for (i).

(iv) We know that $t \succ s_1, \ldots, s_l$, $\Theta(s) \simeq_{lex} \Theta(t)$, $t \succ u_1, \ldots, u_n$, and $\Theta(t) \simeq_{lex} \Theta(u)$. For each triple $\langle s, t, u_i\rangle$ we can apply (ii) to get $s \succ u_i$. For each triple $\langle u, t, s_j\rangle$ we can apply (ii) to get $u \succ s_j$. The lexicographic part holds for the quasi-orderings and, by induction, for the multiset orderings. Therefore all three conditions of Case (3) hold and $s \approx u$.

□

Lemma 7 (Irreflexivity). *For any s, $s \not\succ s$.*

Proof. Apply induction on the height of terms. Assume on the contrary that $s \succ s$ for some s.

Suppose that $s \succ s$ by Case (1) of the ordering, then $s_i \succsim s$ for some i. But by transitivity and the strict subterm property we have $s_i \succ s_i$. But by induction $s_i \not\succ s_i$, and we have a contradiction.

We cannot have $s \succ s$ by Case (2) of the ordering, since $\Theta(t) \simeq_{lex} \Theta(u)$, (using induction for the multiset components).

Therefore neither case is applicable and $s \not\succ s$. □

Lemma 8. *If $s \approx t$ then $t \not\succ s$.*

Proof. Were $t \succ s$, then by transitivity $s \succ s$ contradicting the previous lemma (Lemma 7). □

The converse follows from:

Lemma 9. *If $s \succ t$, then $t \not\succsim s$.*

Proof. Were $t \succsim s$, then by transitivity $s \succ s$, contradicting Lemma 7. □

Theorem 1. *The general path ordering is a quasi-ordering.*

Proof. By the previous lemmata we know that $\succsim$ is reflexive and transitive. □

3. Termination Proofs

The general path ordering can be used to prove termination if certain general conditions are met. The first lemma we present guarantees a strict decrease in the multiset ordering induced by a quasi-ordering. We then show general conditions under which the general path ordering is well-founded. Finally, we give specific conditions for the component orderings which satisfy these general conditions.

Lemma 10. *If $\succsim$ is a quasi-order with the strict subterm property,*

$$s \to t \text{ and } s \succsim t \quad \text{imply} \quad f(\ldots, s, \ldots) \succsim f(\ldots, t, \ldots) \ ,$$

for all terms s, t, ... and function symbols f, and $l\sigma \succ r\sigma$ for all rules $l \to r$ and substitutions σ, then for any rewrite step $u \to v$ $U_{\mathcal{M}} \succ_{\mathcal{M}} V_{\mathcal{M}}$ where $\succ_{\mathcal{M}}$ is the multiset ordering induced by $\succsim$, $U_{\mathcal{M}} = \{t | t \text{ is a subterm of } u\}$, and $V_{\mathcal{M}} = \{t | t \text{ is a subterm of } v\}$.

Proof. To begin, note that given a position p, the multiset of subterms can be split into three parts: the subterms at or below p, the subterms above p, and the subterms disjoint from p.

If $u \to v$ then there is some subterm $u|_p$ of u such that $u|_p = l\sigma$. Therefore

$$u = u[l\sigma]_p \to u[r\sigma]_p = v.$$

By assumption $l\sigma \succ r\sigma$. In addition, repeated application of the strict subterm property with transitivity gives $r\sigma \succ r\sigma|_p$ for all proper subterms of $r\sigma$. Thus the subterm $l\sigma$ in $U_{\mathcal{M}}$ is replaced in $V_{\mathcal{M}}$ by the strictly smaller $r\sigma$ and its subterms.

The only other subterms which are affected by the rewrite are those rooted at symbols on the path from $l\sigma$ to the top of u. We can show that $w[l\sigma]_p \succsim w[r\sigma]_p$ for all contexts w by induction on the depth of position p in w. If w is the empty context, we are given that $l\sigma \succ r\sigma$. Otherwise, let $w = f(\cdots s[l\sigma]_q \cdots)$. By induction $s[l\sigma]_q \succ s[r\sigma]_q$, and by the given implication $w[l\sigma]_p = f(\cdots s[l\sigma]_q \cdots) \succsim f(\cdots s[r\sigma]_q \cdots) = w[r\sigma]_p$. □

Theorem 2. *Let $\succsim$ be a general path ordering. A rewrite system $\mathcal{R}$ terminates if*

- *$l\sigma \succ r\sigma$ for all rules $l \to r$ in $\mathcal{R}$ and substitutions σ and,*
- *$s \to t$ and $s \succsim t$ implies $f(\ldots, s, \ldots) \succsim f(\ldots, t, \ldots)$.*

Proof. The proof of this theorem is akin to Kamin and Lévy [16] and uses a minimal counter-example argument.

To prove the well-foundedness of $\succ$, suppose the contrary and 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 sequence there are only finite descending sequences. (By the subterm property, we can replace any term in a descending sequence by any proper subterm that initiates an infinite descending sequence. Thus we can always construct a minimal descending sequence from an arbitrary descending sequence.) Case (1) of the definition of $\succ$ cannot be the justification for any pair $t_j \succ t_{j+1}$, since then $t_{j-1} \succ t_j|_p \succ t_{j+2}$, for some proper subterm $t_j|_p$ of the jth term in the example, and the example would not be minimal. Therefore every pair must use Case (2) and consequently $\Theta(t_j) >_{lex} \Theta(t_{j+1})$. But a lexicographic combination of well-founded orderings (including $\succ$ on multisets of proper subterms which by assumption are well-founded), is well-founded, and the descending sequence cannot be infinite.

Since the general path ordering is a quasi-order with the strict subterm property, by Lemma 10 we know that each rewrite results in a strict decrease in $\succ_{\mathcal{M}}$. Since $\succ$ is well-founded, $\succ_{\mathcal{M}}$ is as well and termination follows. □

Theorem 3. *Let $\phi_0, \ldots, \phi_{i-1}$ ($i \geq 0$) be monotonic, all but possibly the last strict, and let $\phi_i, \ldots, \phi_k$ be precedence, value-preserving, or multiset extraction component orders. A rewrite system terminates if $l\sigma \succ r\sigma$ in the corresponding general path ordering $\succsim$ for all rules $l \to r$ and ground substitutions σ, provided*

(i) if $\theta_x = R_k$ there is some $y < x$ such that $\theta_y = R_{k-1}$ or $\theta_y = R_{\leq k-1}$; and

(ii) $\simeq_x$ is a congruence for each homomorphism θ_x.

Note that whenever $\geq_x$ is a partial-order, congruence is guaranteed.

Before giving a proof, consider the following examples illustrating the need for restrictions on the components: (We omit parentheses for the unary function symbols 0, 1, f, g.)

Consider the non-terminating two rule rewrite system

$$\begin{array}{rcl} 0011x & \to & 111000x \\ 0x & \to & 11x\ . \end{array} \tag{3}$$

A general path ordering with first component, the precedence $0 > 1$, and the second, the strictly monotonic homomorphism which counts the number of symbols in a term, shows a decrease for both rules. But this violates the condition requiring monotonic homomorphisms to precede the other types of component orderings.

Consider the non-terminating two rule rewrite system

$$\begin{array}{rcl} ffx & \to & fgfx \\ gx & \to & x\ . \end{array} \tag{4}$$

A general path ordering with first component, a monotonic homomorphism θ_{ff} which counts the number of pairs of f's, and second, the precedence $f > g$, shows a decrease for both rules. But this violates the condition requiring that homomorphisms be congruences, since $\theta_{ff}(f(g(a))) \neq \theta_{ff}(f(f(a)))$ even though $\theta_{ff}(g(a)) = \theta_{ff}(f(a))$.

Consider the non-terminating two rule rewrite system

$$\begin{array}{rcl} h(a,b) & \to & h(a,a) \\ a & \to & b\ . \end{array} \tag{5}$$

A general path ordering with first component, the precedence $f > a > b$, and second, the multiset extraction of rank two, shows a decrease for both rules, since $\{b\} > \emptyset$. But this violates the condition requiring that the rank extracting component be preceded by a rank extracting component which extracts terms of rank one.

Proof. By Theorem 2, it suffices to show

$$s \to t \text{ and } s \gtrsim t \text{ imply } u = f(\ldots s \ldots) \gtrsim f(\ldots t \ldots) = v\ ,$$

for all terms s, t, ... and function symbols f.

If $s \approx t$, then $\Theta(s) \simeq_{lex} \Theta(t)$. We demonstrate that $u \approx v$. For each of the subterms $v_i \neq t$, we have $u_i = v_i$. For the subterm t, we have $s \approx t$, and consequently $u = f(\ldots s \ldots) \succ v_i$ for each i by Case (1) of the ordering. Similarly, we have $v = f(\ldots t \ldots) \succ u_j$ for each j. We just need to show

that $\theta_x(u) \simeq_x \theta_x(v)$ for each component order. For precedence and value-preserving component orders this is trivial. For monotonic component orders, the extra condition guarantees that $\simeq_x$ is a congruence and hence $\theta_x(f(\ldots s \ldots)) = f_{\theta_x}(\ldots \theta_x(s) \ldots) \simeq_x f_{\theta_x}(\ldots \theta_x(t) \ldots) = \theta_x(f(\ldots t \ldots))$.

For θ_i that return multisets, we need to consider each of the extraction functions separately:

1. Extract subterms at positions $\mathcal{K}$. If $s \neq u_k$ for any $k \in \mathcal{K}$, then each $u_k = v_k$ and $P_{\mathcal{K}}(u) = P_{\mathcal{K}}(v)$. Otherwise, the multisets are identical except that s is replaced by t and therefore $P_{\mathcal{K}}(u) \approx_{\mathcal{M}} P_{\mathcal{K}}(v)$.

2. Extract subterms of rank k. Since s is equivalent to t, they have the same rank. Therefore $R_k(\mathcal{IS}(u)) \approx_{\mathcal{M}} R_k(\mathcal{IS}(v))$ for all k.

3. Extract terms of rank k or less. As in the previous case, $R_{\leq k}(\mathcal{IS}(u)) \approx_{\mathcal{M}} R_{\leq k}(\mathcal{IS}(v))$.

Now we focus on the strict case, $s \succ t$. As before we can show $u \succ v_i$ for each i. So we just need to show that $\Theta(u) >_{lex} \Theta(v)$. Note that for the recursive definition to give $s \succ t$, there must be some subterm $s|_p$ of s such that $s|_p \succsim t$ by Case (2) of the ordering and hence $\Theta(s|_p) >_{lex} \Theta(t)$. Consider a monotonic homomorphism ϕ_x. There are two cases:

Case A ($s|_p = s$): Suppose that θ_y with $y \leq x$ is the first monotonic homomorphism which shows an increase. For each of the preceding homomorphisms $\theta_z(s) \simeq_z \theta_z(t)$ and therefore $\theta_z(f(\ldots, s, \ldots)) \simeq_z \theta_z(f(\ldots, t, \ldots))$ by congruence for $z \leq y$, while for the yth homomorphism $\theta_y(s) >_y \theta_y(t)$. If the homomorphism is strict, this implies $\theta_y(f(\ldots, s, \ldots)) >_y \theta_y(f(\ldots, t, \ldots))$ and the lexicographic comparison is strictly greater. If the homomorphism is not strict, then $\theta_y(f(\ldots, s, \ldots)) \geq_y \theta_y(f(\ldots, t, \ldots))$ and the status of the lexicographical comparison may depend on the succeeding component orderings.

Case B ($s|_p \neq s$): Consider θ_0. By repeated application of the strict subterm property of the monotonic homomorphism components, we have $\theta_0(s) >_0 \theta_0(s|_p) \geq_0 \theta_0(t)$. If θ_0 is strict, this implies $\theta_0(f(\ldots, s, \ldots)) >_0 \theta_0(f(\ldots, t, \ldots))$ and the lexicographic comparison is strictly greater. If θ_0 is not strict, then $\theta_0(f(\ldots, s, \ldots)) \geq_0 \theta_0(f(\ldots, t, \ldots))$ and the status of the lexicographical comparison may depend on the succeeding component orderings.

In either case, any component orderings following a non-strict homomorphism need not show an increase for s or $s|_p$, respectively, compared with t. As a consequence, none of the succeeding component orderings may safely rely on the lexicographic status of s or its subterms. In addition, since the monotonic homomorphisms depend on the lexicographical status of subterms, it is not safe to have other types of component orders preceding. This is the reason for the restrictions:

- there may only be one non-strict monotonic homomorphism and each of the strict monotonic homomorphisms must precede it, and

- no other type of component ordering may precede a monotonic homomorphism.

Consider now a value-preserving homomorphism and a rewrite $s = c[l\sigma] \rightarrow c[r\sigma] = t$. We are given that $\theta(l\sigma) \simeq_x \theta(r\sigma)$. Combined with congruence of the ordering this results in $\theta(f(\ldots, s, \ldots)) \simeq_x \theta(f(\ldots, t, \ldots))$.

When the termination function is a precedence, its value does not depend on subterms and trivially $\theta(f(\ldots, s, \ldots)) \simeq_x \theta(f(\ldots, t, \ldots))$.

Now consider component orderings that compare multisets of subterms:

1. Extract subterms at positions in $\mathcal{K}$. If $s \neq u_k$ for all $k \in \mathcal{K}$, then each $u_k = v_k$ and $P_{\mathcal{K}}(u) = P_{\mathcal{K}}(v)$. Otherwise the multisets are identical except that s is replaced by t and therefore $P_{\mathcal{K}}(u) \succ_{\mathcal{M}} P_{\mathcal{K}}(v)$.

2. Extract subterms of rank k. Suppose that $s \in R_i(\mathcal{IS}(u))$. Then there is no change in multisets of rank less than i. For the multiset of rank i, the only possible new members are t and terms from R_{i+1} that were dominated by s. Thus we have $R_i(\mathcal{IS}(u)) \succ_{\mathcal{M}} R_i(\mathcal{IS}(v))$. If $k > i$, there may be an increase, but we are guaranteed that either R_i or some $R_{\leq j}$ containing rank i is before θ_x lexicographically, and either of these will show an increase.

3. Extract subterms of rank less than or equal k. Suppose $s \in R_i(u)$. By an argument similar to that above, $R_{\leq k}(\mathcal{IS}(u)) = R_{\leq k}(\mathcal{IS}(v))$ for $k < i$ and $R_{\leq k}(\mathcal{IS}(u)) \succ_{\mathcal{M}} R_{\leq k}(\mathcal{IS}(v))$ for $k = i$. One just needs to consider the case $k > i$. Think of the process of going from $R_{\leq k}(\mathcal{IS}(u))$ to $R_{\leq k}(\mathcal{IS}(v))$ as adding t to the set of immediate subterms then removing s. When t is added other terms may move to higher rank, but not lower rank. So the only possible new term in $R_{\leq k}(\mathcal{IS}(u) \cup \{t\})$ is t. When s is removed, terms may be added from rank $k+1$ (note that terms may only move one rank position when a single term is added or deleted). Consider a term w of rank $j + k + 1$ which is a member of $R_{\leq k}(\mathcal{IS}(v))$, but was not a member of $R_{\leq k}(\mathcal{IS}(u) \cup \{t\})$. It must have been added because a term x_k of rank k moved to rank $k-1$ and $x_k \succ w$. Inductively, we can construct a chain of terms such that $x_i \succ x_{i+1} \succ \cdots \succ x_k \succ w$. But there was only the single term s which was removed at level i and therefore $s = x_i \succ w$. In combination with $s \succ t$, it must be that $R_{\leq k}(\mathcal{IS}(u)) \succnsim R_{\leq k}(\mathcal{IS}(v))$.

□

Whereas we have only used lexicographic and multiset mappings in the general path ordering, in [16], Kamin and Lévy consider the more general case of orderings based on a mapping $\triangleright$ from well-founded quasi-orderings to well-founded quasi-orderings. They allow a component order $\theta t = \langle t_1, \ldots, t_n \rangle$ and $\geq = \trianglerighteq$, where $\triangleright$ recursively makes finitely many comparisons of subterms. In particular, one can use weighted multisets, as in Martin [25].

Theorem 4 (Incrementality). *If a general path ordering $\gtrsim$ with a component ordering $\phi_i = \langle \theta, \geq \rangle$ proves termination of a set of rules $\mathcal{R}$, then the general path ordering $\gtrsim'$ which is the same as $\gtrsim$ except for $\phi_i' = \langle \theta, \geq' \rangle$, where $\geq'$ is an extension of the ordering $\geq$, also proves termination of $\mathcal{R}$.*

Proof. For any termination proof that uses the ith component ordering, the same proof can be constructed, since the mapping is identical and orderings $\geq$ and $\geq'$ are the same for any pair of values $\theta(t_1)$ and $\theta(t_2)$ used to show termination. □

Incrementality is important when an ordering is sought to orient a set of equations. Thus, as a special case, with a precedence one can delay deciding whether $f > g$ or $f < g$, or $f \simeq g$ until necessary to establish the ordering of two terms, (as for the standard recursive path ordering). In general, one can successively refine the well-founded ordering of a homomorphism component.

4. Specific Path Orderings

The following ordering is a special case of the general path ordering to which Theorem 2 applies:

Semantic path ordering (Kamin and Lévy [16]) θ_0 is the identity homomorphism; $\geq_0$ is a well-founded ordering; $\phi_1, \ldots, \phi_n$ give a permutation of the subterms.

For this ordering, one must separately insure that $s \rightarrow t$ implies $s \geq_0 t$. Indeed any terminating system can be (uninterestingly) proven terminating in this way [16], by taking $\geq_0$ to be the reflexive-transitive closure of $\rightarrow$.

The following simplification orderings are special cases of the general path ordering for which the conditions of Theorem 3 hold:

Knuth-Bendix ordering (Knuth and Bendix [18]) θ_0 gives the sum of (non-negative integer) "weights" of the function symbols appearing in a term; $\geq_0$ is the $\geq$ ordering on the natural numbers; ϕ_1 gives a (total) precedence; $\phi_2, \ldots, \phi_{n+1}$ give (a permutation of) the immediate subterms.

Polynomial path ordering (Lankford [19]) θ_0 is a strict monotonic homomorphism with each f_θ a polynomial with positive integer coefficients; $\geq_0$ is the $\geq$ ordering on the natural numbers; ϕ_1 gives a precedence; $\phi_2, \ldots, \phi_{n+1}$ give a permutation of the immediate subterms.

Multiset path ordering (the original version of the "recursive path ordering", Dershowitz [6]) ϕ_0 is a precedence; ϕ_1 extracts the multiset of immediate subterms.

Extended path ordering (Dershowitz [6]) ϕ_0 extracts one of the immediate subterms; ϕ_1 extracts a multiset of the remaining immediate subterms.

Lexicographic path ordering (Kamin and Lévy [16]) ϕ_0 is a precedence; $\phi_1, \ldots, \phi_n$ give a permutation of the subterms.

Recursive path ordering ("with status", Lescanne [21]) ϕ_0 is a total precedence; $\phi_1, \ldots, \phi_n$ give a permutation of the subterms or multisets of subterms, depending on the function symbol.

Extended Knuth-Bendix ordering (Dershowitz [6], Steinbach and Zehnter [35]) ϕ_0 is a monotonic interpretation; ϕ_1 is a precedence; $\phi_2, \ldots, \phi_{n+1}$ give the subterms in order, permuted, or multisets of immediate subterms, depending on the function symbol.

For a system like

$$\begin{array}{rcl} fsx & \rightarrow & shdfx \\ f0 & \rightarrow & 0 \\ d0 & \rightarrow & 0 \\ dsx & \rightarrow & ssdx \\ hssx & \rightarrow & shx \ , \end{array} \tag{6}$$

a precedence ($f > h > d > s > 0$) ought to be considered first, before looking at subterms, as with a lexicographic path ordering.

The next special case is not a simplification ordering, but the conditions of Theorem 3 hold for it as well.

Value-preserving path ordering (Plaisted [31], Kamin and Lévy [16]) θ is a value-preserving homomorphism and $\geq$ is a well-founded quasi-order; ϕ_0 is a precedence; θ_1 is θ applied to the first subterm and $\geq_1$ is $\geq$; θ_2 is θ applied to the second subterm and $\geq_2$ is $\geq$; and so forth.

As an example of the use of the value-preserving path ordering, consider System 2. The precedence is $fact >_0 \times >_0 + >_0 s$; θ_1 interprets everything

naturally: $fact$ as factorial, s as successor, p as predecessor, $\times$ as multiplication, $+$ as addition, and 0 as zero. The ordering $\geq_1$ is the well-founded greater-than relation on natural numbers. Let all constants be interpreted as natural numbers, making all terms non-negative. Each rule causes a strict decrease with respect to the general path ordering and the rewrite system terminates. This approach works for primitive-recursive functions in general.

Note that to use a natural interpretation, one must always 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.

We enlarge on the idea embodied in the value-preserving ordering in the following way, intended to mirror the standard structural induction proof method for recursive programs:

Definition 5 (Natural Path Ordering). A *natural path ordering* is a special case of the general path ordering with two component orderings: ϕ_0 is a precedence and ϕ_1 is defined for each f (of arity n), as $\theta_1 f(t_1, \ldots, t_n) = f_{\theta_1}(\theta_1 t_1, \ldots, \theta_1 t_n)$, where θ_1 is a value-preserving homomorphism to some arbitrary algebra $\mathcal{A}$, and f_{θ_1} a mapping from $\mathcal{A}^n$ to a well-founded set $(W, \geq)$.

Theorem 2 applies.

As an example, consider the following rewrite system for computing the average of two integers:

$$\begin{array}{rcl} a(sx, y) & \rightarrow & a(x, sy) \\ a(x, sssy) & \rightarrow & sa(sx, y) \\ a(0, 0) & \rightarrow & 0 \\ a(0, s0) & \rightarrow & 0 \\ a(0, ss0) & \rightarrow & s0\ . \end{array} \tag{7}$$

A multiset path ordering will not work for the arguments of a in the first rule and a lexicographical path ordering will not work for the first two rules. The natural path ordering is sufficient for proving termination with ϕ_0 as $a >_0 s >_0 0$ and ϕ_1 given by $\theta_1(a(x, y)) = 2\theta(x) + \theta(y)$, where θ is the value-preserving homomorphism: $a_\theta = \lambda xy.\lfloor \frac{x+y}{2} \rfloor$, $s_\theta = \lambda x.x + 1$, and $0_\theta = \lambda x.0$.

A more complicated example using the general path ordering is the following rewrite system which sorts a list of natural numbers into decreasing order via an insertion sort:

$$sort(nil) \rightarrow nil \tag{5.1}$$

$$
\begin{aligned}
sort(cons(x,y)) &\rightarrow insert(x, sort(y)) && (5.2)\\
insert(x, nil) &\rightarrow cons(x, nil) && (5.3)\\
insert(x, cons(v,w)) &\rightarrow choose(x, cons(v,w), x, v) && (5.4)\\
choose(x, cons(v,w), y, 0) &\rightarrow cons(x, cons(v,w)) && (5.5)\\
choose(x, cons(v,w), 0, s(q)) &\rightarrow cons(v, insert(x,w)) && (5.6)\\
choose(x, cons(v,w), s(p), s(q)) &\rightarrow choose(x, cons(v,w), p, q)\ . && (5.7)
\end{aligned}
$$

Four component orders are used. They are

$$
\begin{aligned}
\phi_0 = &\quad \text{the precedence } sort > insert \simeq choose > cons\\
\phi_1 = &\quad \text{the extraction based on the outermost symbol } f\\
&\quad \theta_1 = \begin{cases} P_{\{1\}} & f = sort\\ P_{\{2\}} & f = choose, insert\\ \emptyset & \text{otherwise} \end{cases}\\
\phi_2 = &\quad \text{the precedence } sort > insert > choose > cons\\
\phi_3 = &\quad \text{the extraction based on the outermost symbol } f\\
&\quad \theta_3 = \begin{cases} P_{\{1\}} & f = sort\\ P_{\{2\}} & f = insert\\ P_{\{3\}} & f = choose\\ \emptyset & \text{otherwise .} \end{cases}
\end{aligned}
$$

The ordering interleaves precedences with recursive comparisons of subterms and thus is unlike either the semantic path ordering [16] or semantic labeling [36]. No semantic interpretation of the function symbols is required to prove termination in this example.

If one were to use an ordering just based on the precedence ϕ_2, all of the rules except for the seventh would be oriented in the appropriate direction. Unfortunately, the fourth and seventh rules interact with each other. In particular, there is a *choose* and an *insert* on opposite sides of each rule. The precedence ϕ_0 is chosen to guarantee a decrease in the lexicographical part when ordering Rule 6 by Case (2) of the general path ordering, while leaving Rule 4 equal. The first condition for Case (2) requires that the left-hand side of Rule 6 be strictly greater than each of the two subterms on the right. The non-trivial comparison is $choose(x, cons(v,w), 0, s(q))$ with $insert(x,w)$. These terms are equal under the precedence ordering ϕ_0, but by selecting the second subterm of both *choose* and *insert* we achieve the needed decrease, and Rule 6 is correctly ordered.

Now consider Rule 4. Fortunately, the second subterm on both sides of Rule 4 is identical, leaving the lexicographical order unaffected. The precedence ordering ϕ_2 breaks that tie. Verifying the first condition of Case (2) for Rule 4 is easy.

Rule 1 is a trivial application of Case (1). Rule 2 is nearly as trivial. The only observation to make is that the first condition for Case (2) requires $sort(cons(x, y)) \succ sort(y)$, which itself requires an application of Case (2) where the lexicographic part requires the extraction and comparison of $cons(x, y)$ with y. Rules 3 and 5 are also straightforward.

Rule 7 meets the first conditions for Case (2), but is equal for the lexicographical part with respect to the first three component orderings. The addition of a fourth component breaks the tie by extracting the third subterm for *choose* (the fourth subterm would also have worked).

Therefore, by the general path ordering, this system of rules terminates.

5. 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:

$$\begin{array}{rcl} fsx & \rightarrow & sffpsx \\ f0 & \rightarrow & 0 \\ psx & \rightarrow & x \;. \end{array} \tag{6}$$

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. 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.

2. For rewriting in general, one must consider the possibility that the x to which the definition of $f(x)$ is applied is itself a term containing occurrences of the defined function f (or of mutually-recursive defined functions), something usually ignored in the (sufficiently complete) functional case.

3. One cannot use a syntactic simplification ordering like the simple path ordering [30], 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.

First a few definitions: A *non-overlapping* system is one where no left-hand side of a rule unifies with any non-variable subterm of the left-hand

side of another rule or with a non-variable proper subterm of itself, with variables in the two rules renamed apart. A *left-linear* system has no repeated variables on the left-hand side of a rule. Similarly, a *right-linear* system has no repeated variables on the right-hand side of a rule. An *orthogonal* system is non-overlapping and left-linear. An *overlaying* system is one whose only overlaps are at the topmost position, that is, no left-hand side unifies with a non-variable proper subterm of any left-hand side.

As an example of an orthogonal system, consider:

$$\begin{array}{rcl} fsx & \to & sfpsx \\ f0 & \to & 0 \\ psx & \to & x\ . \end{array} \tag{7}$$

The general path ordering works with component orders ϕ_0 and ϕ_1, where ϕ_0 is a precedence with $f >_0 s, p$, and ϕ_1 is a natural interpretation with $f_\theta = \lambda x.x$, $p_\theta = \lambda x.x - 1$, $s_\theta = \lambda x.x + 1$, and $0_\theta = \lambda x.0$.

The following is overlaying and locally confluent:

$$\begin{array}{rcl} x \times 0 & \to & 0 \\ x \times sy & \to & (x \times y) + x \\ x + 0 & \to & x \\ 0 + x & \to & x \\ x + sy & \to & s(x + y) \\ sx + y & \to & s(x + y)\ , \end{array} \tag{8}$$

Proposition 1 (Gramlich [14]). *A locally confluent overlaying system is terminating if, and only if, innermost rewriting always leads to a normal form.*

A *locally confluent* system is one for which $u \to s, t$ implies $s, t \to^* v$, for some v, where $\to^*$ is the reflexive transitive closure of the rewrite relation.

An *innermost* derivation is one in which the redex chosen at every rewrite step contains no rewritable proper subterm. In particular, orthogonal systems are locally confluent and have no (non-trivial) overlays; the proposition for this case was shown by O'Donnell [29]. Geupel [13] showed that left-linearity is unnecessary, that is, a non-overlapping system is terminating if, and only if, innermost rewriting always leads to a normal form.

We give an alternate proof to the one in [14]. (See also Middeldorp [27].) It is similar in style to Geupel's proof [13] that forward closures suffice for showing termination of non-overlapping rewrite systems.

Proof. We say that a *term* t is *terminating* (and write $t \in \mathcal{T}_f$) if all derivations from t are finite; t is *non-terminating* ($t \in \mathcal{T}_\infty$) if some derivation from t is infinite; and t is on the *frontier* ($t \in \mathcal{FR}$) if t is non-terminating,

but every proper subterm of t is terminating. If a term has no frontier subterms, then it must be terminating. Conversely, if a term has a frontier subterm, it is non-terminating.

For a locally confluent rewrite system, any terminating term t has a unique normal form $\hat{t}$ by Newman's Lemma [28]. The *inner normalization* function N for a locally confluent rewrite system is defined as follows:

$$N(t) = \begin{cases} f(N(t_1), \ldots, N(t_n)) & \text{if } t = f(t_1, \ldots, t_n) \in \mathcal{T}_\infty \\ \hat{t} & \text{if } t \in \mathcal{T}_f. \end{cases}$$

Clearly, $t \to^* N(t)$.

If the rewrite system is non-terminating, we can construct an infinite derivation as follows: Let $t_1 = s_1$ be a frontier term. It initiates an infinite derivation of the form

$$t_1 = s_1 \to^*_{\text{below top}} s_1' \to_{\text{at top}} t_2 \to \cdots ,$$

where all the steps in $s_1 \to \ldots \to s_1'$ are below the top position and t_2 contains a frontier subterm s_2 at some position p_2. Continuing in this way we get the infinite derivation

$$t_1 \to^+ t_2 \to^+ t_3 \to^+ \cdots$$

where $t_i = u_2[u_3[\cdots u_i[s_i]_{p_i} \cdots]_{p_3}]_{p_2}$, each s_i is a frontier subterm of u_i, and

$$s_i \to^*_{\text{below top}} s_i' \to_{\text{at top}} u_{i+1}[s_{i+1}]_{q_{i+1}} ,$$

where $p_{i+1} = p_i \cdot q_{i+1}$. (This is a constricting derivation á la Plaisted [32], making the proof a little simpler.)

Notice that each redex in the infinite derivation is either terminating (those below p_i in s_i) or on the frontier (at p_i in s_i). Let us consider these cases separately.

- The redex is a terminating subterm: Since each of the terms in $s_i \to^* s_i'$ is on an infinite path, the position of the frontier is unaffected and hence by local confluence $N(s_i) = N(s_i')$. Since both s_i and s_i' are nonterminating, by the definition of N we have $N(t_i[s_i]) = N(t_i[s_i'])$.

- The redex is a frontier subterm: In this case we have $s_i' \to u_{i+1}[s_{i+1}]_{q_{i+1}}$ with some rule $g(c_1, \ldots, c_n) \to r$ and substitution σ. Since s_i' is on the frontier, each of its subterms must be terminating and therefore each of the terms in the image of σ is terminating as well. Since the rewrite system is overlaying, we know that each of the contexts $c_1, \ldots, c_n$ is in normal form, so the rewrites below p_i are all within the terminating terms introduced by σ. In

other words, $N(s_i) = g(c_1, \ldots, c_n)\hat{\sigma}$, where $\hat{\sigma}$ is σ with each of the terms in its image rewritten to normal form. By application of the same rule $N(s_i) \to r\hat{\sigma}$.

Consider $u_{i+1}[s_{i+1}] = r\sigma$. Since the terms in the image of σ are terminating, by the definition of N we have $N(r\sigma) = N(r\hat{\sigma})$. (By definition, we know that $r\hat{\sigma} \to^* N(r\hat{\sigma})$.) Since both s_i and $u_{i+1}[s_{i+1}]$ are frontier terms (in t_i and t_{i+1}, respectively), we have $N(t_i[s_i]) \to^+ N(t_{i+1}[s_{i+1}])$.

Thus from the infinite derivation $t_1 \to^+ t_2 \to^+ t_3 \to^+ \cdots$ we can construct an infinite derivation $N(t_1) \to^+ N(t_2) \to^+ N(t_3) \to^+ \cdots$. Each of the rewrite steps corresponding to a frontier redex in the original derivation will be innermost after the application of N. The remaining steps are all under the position of the immediately preceding frontier step and are applied to terminating subterms. By local confluence, we may rearrange these rewrites to be innermost as well. Thus, from any infinite derivation we can find some innermost infinite derivation. □

Notice that given any non-terminating term v, we can use the above construction to obtain the derivation $v[t_1] \to^+ v[N(t_1)] \to^+ v[N(t_2)] \to^+ \cdots$ and so each term is terminating if and only if it is innermost terminating.

As an example of the use of Proposition 1, consider System 8. We need to show that, under the assumption that variables are bound to normal forms, each rule leads to a normal form. Consider the second rule. If x and y are in normal form, then after applying the rule the innermost redex is the newly produced multiplication. But we can show that this will terminate since its second argument is smaller. Addition can be considered separately from multiplication, and it too terminates regardless of changes in the first summand. Therefore, every innermost derivation terminates, and hence the system terminates.

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 6. The *forward closures* of a given rewrite system are a set of derivations inductively defined as follows:

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

The idea, first suggested by Lankford and Musser [20], is to restrict application of rules to that part of a term created by previous rewrites. We can

define *innermost* (*outermost*) forward closures as those closures which are innermost (outermost) derivations. More generally, arbitrary redex choice strategies may be captured in an appropriate forward closure.

For example, the forward closures of System 7 are

$$\begin{array}{rcll} fs^nx & \rightarrow^+ & s^nfpsx & n>0 \\ fs^n0 & \rightarrow^+ & s^n0 & n\geq 0 \\ fs^nx & \rightarrow^+ & s^nfx & n>0 \\ psx & \rightarrow & x\ . & \end{array}$$

In fact, since there is only one possible redex in every forward closure, these are the innermost and outermost forward closures as well.

For an example where the innermost and outermost forward closures are not identical, consider the rewrite system:

$$\begin{array}{rcl} fsx & \rightarrow & sfpsfx \\ f0 & \rightarrow & 0 \\ psx & \rightarrow & x\ . \end{array} \tag{9}$$

The forward closure

$$fssx \rightarrow sfpsfsx \rightarrow sffsx \rightarrow sfpsfx$$

is outermost, but not innermost. The forward closure

$$fssx \rightarrow sfpsfsx \rightarrow sfpssfpsfx$$

is innermost, but not outermost.

Proposition 2 (Dershowitz [5]). *A right-linear rewrite system is terminating if, and only if, there are no infinite forward closures.*

In particular, forward closures suffice for string-rewriting systems.

Thus, for a system like

$$\begin{array}{rcl} fsx & \rightarrow & ssfpsx \\ f0 & \rightarrow & 0 \\ psx & \rightarrow & x\ , \end{array} \tag{10}$$

we can restrict our attention to forward closures. (This is not exactly a string rewriting system since the second rule applies only at the *end* of a string.) Since f's won't nest, termination can be shown by comparing the argument on the left, sx, with the one on the right, psx, using a natural semantic comparison.

Proposition 3 (Guepel [13]). *A non-overlapping rewrite system is terminating if, and only if, there are no infinite forward closures.*

This extends the result in [5] for orthogonal systems. In general, though, a rewrite system need not terminate even if all its forward closures do [5].

Consider the following system for symbolic differentiation with respect to t:

$$\begin{array}{rcll} D\ t & \to & 1 & \\ D\ z & \to & 0 & z \text{ does not contain } t \\ D\ (x+y) & \to & D\ x\ +\ D\ y & \\ D\ (x\cdot y) & \to & y\cdot D\ x\ +\ x\cdot D\ y & \\ D\ (x-y) & \to & D\ x\ -\ D\ y & \\ D\ (-x) & \to & -D\ x & \\ D\ (x/y) & \to & D\ x/y\ -\ x\cdot D\ y/y^2 & \\ D\ (\ln\ x) & \to & D\ x/x & \\ D\ (x^y) & \to & y\cdot x^{y-1}\cdot D\ x\ +\ x^y\cdot(\ln x)\cdot D\ y\ . & \end{array} \tag{11}$$

It is orthogonal, 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.

Theorem 5. *A rewrite system has an infinite innermost derivation if, and only if, it has an infinite innermost forward closure.*

Proof. Consider a term t which has an infinite innermost derivation. It must have a subterm $t|_p$ which has an infinite innermost derivation such that the top position is eventually rewritten:

$$t|_p = s_0 \to s_1 \to \cdots \to s_i \to_{\text{top}} s_{i+1} \to \cdots\ .$$

But for the top of s_i to be rewritten all of its immediate subterms must be in normal form. Therefore, the derivation from s_i is an instance of an innermost forward closure. □

Theorem 6. *A locally-confluent overlaying rewrite system is terminating if, and only if, it has no infinite innermost forward closure.*

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

Proof. From Proposition 1 we know that if the rewrite system is non-terminating it will have an innermost non-terminating derivation. But by Theorem 5 this implies the existence of an infinite innermost forward closure. □

This method applies to most of the previous examples. Since we need only consider innermost derivations, we can assume that problematic expressions like psx on the right of System 2 rewrite immediately to x (and that the x is in normal form). Since we need only consider forward closures, we can assume x contains no function symbols other than s and 0, without having to show that *fact* is sufficiently complete (which it would not be were the rule $fact(0) \to s0$ omitted). By "sufficiently complete", we mean that every ground term containing the symbol *fact* and constructors reduces to a term containing only constructors.

For System 6, we can compare the multiset of right-hand side arguments $\{fpsx, psx\}$ of the recursive function symbols with that of left-hand side, $\{sx\}$. Semantics are necessary for this comparison. If we let $psx = x$ and $fx = x$ (just as we would be using θ_0 with a natural path ordering), we have $\{sx\}$ 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 techniques ("proof by consistency"; see Bachmair and Dershowitz [1]). Indeed, adding $fx \to x$ to System 6 yields a terminating confluent overlay system.

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

$$\begin{array}{rcl} fsx & \to & ssffpsx \\ f0 & \to & 0 \\ psx & \to & x\ . \end{array} \tag{12}$$

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 (allowing one to prove $s0 = ss0$).

Proposition 4. *If a left-linear rewrite system is constructor-based, then all of its forward closures begin with constructor-based instances of left-hand sides of rules.*

A term is *constructor-based* if all of its proper subterms have only free constructors and variables. A rewrite system is constructor-based if its left-hand sides are constructor-based, and a forward closure is constructor-based if its initial term is constructor-based.

Proof. Since forward closures are only extended via substitution, a trivial induction shows that every forward closure's initial term is an instance of the left-hand side of some rule.

Consider the inductive definition of forward closures. For the base case, each rule is a forward closure which, trivially, is constructor-based. Assume that $c[\vec{x}] \to \cdots \to d[\vec{x}]$ is a constructor-based forward closure. It is extended by applying the substitution σ, found by unifying the left-hand side of a rule, $f(k_1[\vec{y}], \ldots, k_n[\vec{y}])$, with some subterm of d. Suppose that the extension is not constructor based. This can only happen if the substitution, σ, maps some $x_i \in \vec{x}$ to a term with a function symbol in it. We unify $f(c_1[\vec{x}], \ldots, c_n[\vec{x}])$ with $f(k_1[\vec{y}], \ldots, k_n[\vec{y}])$. Since the rule itself is constructor-based, the only source of a function symbol is one of the contexts, c_i, from d. But these can only unify with variables, $\vec{y}$, from the rule. Since the rules are left-linear, each occurrence is distinct and therefore, the only mappings in σ which have function symbols are for variables in $\vec{y}$, not $\vec{x}$. This is a contradiction, and the extension is constructor based. □

As a counter-example illustrating the need for left-linearity, consider the rewrite system:

$$\begin{array}{rcl} f(x,x) & \to & f(ga,x) \\ gb & \to & c\ . \end{array} \tag{13}$$

It is constructor-based, but the forward closure $f(ga,ga) \to \cdots \to f(ga,ga)$ is not.

A left-linear, locally confluent, constructor-based rewrite system is overlaying, and hence, by Theorem 6, is terminating if and only if its innermost forward closures are terminating. But by Proposition 4, 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.

6. Non-erasing Systems

We focus now on non-erasing rewrite systems. Recall that a system is *non-erasing* if any variable on the left-hand side of a rule is also on the right-hand side.

Proposition 5 (O'Donnell [29]). *A non-erasing orthogonal system is terminating if, and only if, it is normalizing (every term has a normal form).*

Therefore, the first rule of System 7 (which has a self-embedding) may be immediately followed by an application of the last rule, effectively replacing the former with $fsx \to sfx$. Now termination can be shown with

a standard recursive path ordering with precedence $f >_0 s$, demonstrating that the original system is normalizing, and, hence, terminating.

We can improve upon the previous proposition.

Lemma 11. *If a term has an infinite derivation in a non-erasing non-overlapping system, then all derivations from that term are infinite.*

Note that both non-overlapping string systems and non-erasing orthogonal rewrite system are special cases covered by this lemma.

Proof. We use the inner normalization function N. From the proof of Proposition 1, we know that if t is a frontier term, then $N(t)$ is also non-terminating. As a consequence, for an arbitrary non-terminating term t, it must be that $N(t)$ is non-terminating as well.

Consider an arbitrary non-terminating term t and an arbitrary rewrite step applied to that term at redex s. The rewrite must occur in one of the following positions:

- The redex $s = l\sigma$ is a terminating term. But $t[l\sigma] \to t[r\sigma] \to^* N(t)$ by local confluence and since $N(t)$ is non-terminating, $t[r\sigma]$ is as well.
- The redex is a frontier term. But we know that there is exactly one rule, $l \to r$, applicable at that redex. From our proof of Proposition 1, we know that the rule will still be applicable to $N(s)$. In addition, $N(s[r\sigma])$ is still non-terminating. Suppose that there was some other rule, $l' \to r'$, which was applicable, but led to a terminating term. This rule would also be applicable to $N(s)$. But since $N(s)$ is an instance of the right-hand sides of both rules they overlap, which is a contradiction. Therefore we know that $t[s[l\sigma]] \to t[s[r\sigma]] \to N(t[s[r\sigma]])$ and that $N(t[s[r\sigma]])$ is non-terminating.
- The redex is non-terminating, but is not a frontier term. We know that there is some subterm $s|_p$ which is the frontier. Suppose that the rule, $l \to r$ has the top symbol of $s|_p$ as part of its context $c[\cdot]$. Consider applying N to the entire term. The subterms of the context $c[\cdot]$ are terminating, so they must be preserved; the top symbol of $c[\cdot]$ heads the subterm $r|_p$ and won't be rewritten, either. Since N maps terminating terms to their unique normal forms, repeated variables will observe the same rewrite and the applicability of the rule is unaffected by N. But we know that there is some other rule, $l' \to r'$, which is applicable at the top of $N(s|_p)$. But that means there is an instance to which both rules apply and overlap. Therefore, the rule may only bind $s|_p$ by a variable. Since the system is non-erasing, the frontier term $s|_p$ must also be in the result of the rewrite, $t[r\sigma]$, which consequently must also be non-terminating.

Since there is no rule application which can lead to a term that is terminating, every derivation from a non-terminating term must be infinite. □

The following non-overlapping rewrite system shows that the non-erasing property is necessary:

$$\begin{array}{rcl} gx & \rightarrow & a \\ b & \rightarrow & gb. \end{array} \tag{14}$$

Clearly, the term b has both infinite and terminating derivations.

To see that this result can not be extended to non-erasing, locally-confluent overlaying systems consider:

$$\begin{array}{rcl} a & \rightarrow & a \\ a & \rightarrow & b. \end{array} \tag{15}$$

Unfortunately, the term a has both infinite and terminating derivations.

The following generalizes Proposition 5.

Theorem 7. *A non-erasing non-overlapping system is terminating if, and only if, it is normalizing.*

This is a corollary of Lemma 11. Gramlich [15] gives an independent proof of this.

Theorem 8. *A non-erasing non-overlapping system is terminating if, and only if, no right-hand side of an arbitrary strategy basic forward closure initiates an infinite derivation.*

A *basic* forward closure $l\sigma \rightarrow r\sigma \rightarrow \cdots$ is one for which the substitution σ, used in the first step of the closure, is irreducible.

is one where if t is the initial term and $l \rightarrow r$ is the initial rule, then $t = l\sigma$ with an irreducible substitution σ.

Proof. Suppose the system has an infinite derivation. Then we know from Theorem 6 that there is a innermost forward closure leading to an infinite derivation. But the left-hand side of the infinite forward closure is a term which has an infinite derivation, and hence all derivations from it must be infinite as well (by Lemma 11). Furthermore, all derivations from it are instances of basic forward closures. Therefore, for an arbitrary strategy there is a corresponding infinite basic forward closure of the appropriate type. □

As an example, consider the following system:

$$\begin{array}{rcl} fsx & \rightarrow & psffx \\ f0 & \rightarrow & 0 \\ psx & \rightarrow & x\ . \end{array} \tag{16}$$

Its outermost forward closures are:

$$\begin{array}{rcll} fs^n x & \to^+ & f^{n-1}psffx & n > i \\ fs^n x & \to^+ & f^{n+1}x & n > i \\ fs^n 0 & \to^+ & f^m 0 & n \geq 0, n \geq m \\ psx & \to & x\ . & \end{array}$$

For a forward closure which is an instance of $fs^n x \to^+ f^{n-1}psffx$, we only need to consider the extension with the rule $psx \to x$, since any other choice would not lead to an outermost forward closure. Verification of termination is easy now. Terms of the form $f^{n-1}psffx$ derive in one step $f^{n+1}x$ which is in normal form. Terms of the form $f^m 0$ derive 0 in m steps. Since no right-hand side admits a non-terminating rewrite sequence, the system is terminating.

System 6 can be shown terminating via similar reasoning (though the expressions for the forward closures are more complicated).

Zantema's Problem [37] is to prove termination of the following one-rule string-rewriting system:

$$1100 \to 000111, \tag{17}$$

corresponding to the term-rewriting rule $1100x \to 000111x$. (Theorem 7 applies as well, since string rewriting systems are non-erasing and this rule is non-overlapping.)

First note that for any term of the form $\alpha 00\beta$, if $\alpha 00$ is a normal form then any term derived from $\alpha 00\beta$ must have the form $\alpha 00\gamma$. Consider the right-hand side of the rule. It has the above form with suffix $\beta = 111$. There are two ways to construct a new outermost forward closure from 111:

$$\alpha 0011100 \to \alpha 001000111 = \alpha' 00111$$

and

$$\alpha 00111100 \to \alpha 00\underline{11000}111.$$

Since there is a redex (underlined) in the right hand side of the second forward closure, any outermost forward closure extending it must rewrite the redex:

$$\alpha 00111100 \to \alpha 000001110111 = \alpha' 001110111.$$

This gives us a new possibility $\beta = 1110111$, which can be used to construct a new outermost forward closure as:

$$\alpha 00111011100 \to \alpha 0011101000111 = \alpha' 00111$$

and

$$\alpha 001110111100 \to \alpha 001110\underline{11000}111.$$

As before we need to reduce the right hand side for any outermost forward closure:

$$\begin{aligned} \alpha 001110111100 &\rightarrow \alpha 001\underline{1100}01110111 \\ &\rightarrow \alpha 0010001\underline{1100}1110111 \\ &\rightarrow \alpha 00100010001111110111 = \alpha' 001111110111 \ . \end{aligned}$$

The third possibility is $\beta = 1111110111$, which can be used to construct a new outermost forward closure as:

$$\alpha 00111111011100 \rightarrow \alpha 0011111101000111 = \alpha' 00111$$

and

$$\alpha 001111110111100 \rightarrow \alpha 00111111\underline{1100}0111 \ .$$

The second of these has a redex which must be rewritten:

$$\begin{aligned} \alpha 001111110111100 &\rightarrow \alpha 001111\underline{1100}01110111 \\ &\rightarrow \alpha 00111\underline{1100}0111001110111 \\ &\rightarrow \alpha 00\underline{1100}01110111001110111 \\ &\rightarrow \alpha 000001110111\underline{1100}1110111 \\ &\rightarrow \alpha 0000011101110100011111110111 \\ &= \alpha' 001111110111 \ . \end{aligned}$$

For termination, it must be the case that no right-hand side of an outermost forward closure initiates a non-terminating derivation. Each of the right-hand sides of the form $\alpha 00111$, $\alpha 001110111$, and $\alpha 001111110111$ are already in normal form. Consider the right-hand side $\alpha 00111011000111$. It has only one possible derivation, leading to the normal form $\alpha 001111110111$. The right-hand side $\alpha 001111\underline{1100}01110111$ is a little more complicated. The next term in the sequence is $\alpha 00111\underline{1100}01\underline{1100}1110111$, which has two possible rewrites. But notice that each of the succeeding terms in the outermost derivation preserve the inner rewrite. Therefore they can be performed independently and $\alpha' 001111110111$ is the final form of all possible rewrites. None of the right-hand sides initiates an infinite rewrite, so the system is terminating.

Note that all derivations of a non-overlapping string-rewriting system have the same length. Hence, we have shown (as Zantema conjectured) that $2n$ is an upper-bound on the length of any derivation from a string of size n (in worst case six steps are needed to decrease the size of the suffix β by three). Other solutions to this problem are due to Geser [11] and Bittar [2]. See also McNaughton [26] who considers termination of semi-Thue systems such as this example.

7. Conclusion

The general path ordering we have defined provides a powerful general purpose tool for demonstrating termination of rewrite systems. It can be applied in situations in which the more familiar simplification orderings cannot, as when the rewrite system is self-embedding. It encompasses virtually all popular methods, including polynomial (and other) interpretations, the Knuth-Bendix ordering and its extensions, and the recursive path orderings and its variants. Geser [12] has suggested a weakening of the subterm conditions, thereby strengthening the general path ordering.

Several examples, including 1, were mechanically verified by our general path ordering termination code (GPOTC). 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. The 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 [19], Ben Cherifa and Lescanne [4], or Steinbach and Zehnter [35].

Forward closures provide a more specialized method for showing termination, applicable to locally-confluent overlaying or right-linear systems. Special cases of interest are orthogonal and string rewrite systems which are terminating whenever their forward closures are. In addition, when the rewrite system is non-erasing (as for string systems) the set of forward closures can be restricted to just the innermost forward closures, easing proof of termination. Furthermore, if the system is non-overlapping, any rewrite strategy will suffice to restrict the set of forward closures.

Work is currently under way on an implementation of forward closures.

Both methods can often lead to more natural proofs, using arguments similar to those used for recursive definitions.

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

References

[1] Leo Bachmair and Nachum Dershowitz. Equational inference, canonical proofs, and proof orderings. *J. of the Association for Computing Machinery*, 41(2):236–276, March 1994.

[2] Elias Tahhan Bittar. Non erasing, right linear, orthogonal term rewrite systems application to zantema's problem. Technical Report RR 2202, INRIA, 1993.

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

[4] Ahlem Ben Cherifa and Pierre Lescanne. Termination of rewriting systems by polynomial interpretations and its implementation. *Science of Computer Programming*, 9(2):137–159, October 1987.

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

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

[7] Nachum Dershowitz. Termination of rewriting. *J. Symbolic Computation*, 3(1&2):69–115, February/April 1987. Corrigendum: *4*, 3 (December 1987), 409–410; reprinted in *Rewriting Techniques and Applications*, J.-P. Jouannaud, ed., pp. 69—115, Academic Press, 1987.

[8] Nachum Dershowitz and Charles Hoot. Topics in termination. In C. Kirchner, editor, *Proceedings of the Fifth International Conference on Rewriting Techniques and Applications (Montreal, Canada)*, Lecture Notes in Computer Science, Berlin, June 1993. Springer-Verlag.

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

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

[11] Alfons Geser. A solution to zantema's problem. Technical Report MIP-9314, Universität Passau, Passau, Germany, December 1993.

[12] Alfons Geser. An improved general path order. Technical Report MIP-9407, Universität Passau, Passau, Germany, June 1994.

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

[14] Bernhard Gramlich. Relating innermost, weak, uniform and modular termination of term rewriting systems. In A. Voronkov, editor, *Proceedings of the Conference on Logic Programming and Automated Reasoning (St. Petersburg, Russia)*, volume 624 of *Lecture Notes in Artificial Intelligence*, pages 285–296, Berlin, July 1992. Springer-Verlag.

[15] Bernhard Gramlich, personal communication.

[16] 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.

[17] Jan Willem Klop. Term rewriting systems. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2, chapter 1, pages 1–117. Oxford University Press, Oxford, 1992.

[18] Donald E. Knuth and P. B. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, Oxford, U. K., 1970. Reprinted in *Automation of Reasoning 2*, Springer-Verlag, Berlin, pp. 342–376 (1983).

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

[20] Dallas S. Lankford and David R. Musser. A finite termination criterion. May 1978.

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

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

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

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

[25] Ursula Martin. A geometrical approach to multiset orderings. *Information Processing Letters*, 67:37–54, May 1989.

[26] Robert McNaughton. The uniform halting problem for one-rule semi-thue systems. Technical Report 94-18, Rensselaer, Department of Computer Science, Rensselaer Polytechnic Institute,Troy, New York 12180-3590, August 1994.

[27] Aart Middeldorp. A simple proof to a result of bernhard gramlich. Unpublished note, February 1994.

[28] M. H. A. Newman. On theories with a combinatorial definition of 'equivalence'. *Annals of Mathematics*, 43(2):223–243, 1942.

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

[30] 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.

[31] David A. Plaisted. Personal communication, 1979. Department of Computer Science, University of Illinois.

[32] David A. Plaisted. Polynomial time termination and constraint satisfaction tests. In Claude Kirchner, editor, *Proceedings of the Fifth International Conference on Rewriting Techniques and Applications*, pages 405–420, Montreal, Canada, June 1993. Vol. 690 of *Lecture Notes in Computer Science*, Springer , Berlin.

[33] David A. Plaisted. Term rewriting systems. In D. M. Gabbay, C. J. Hogger, and J. A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 4, chapter 2. Oxford University Press, Oxford, 1993. To appear.

[34] Joachim Steinbach. Simplification orderings - history of results. to be submitted.
[35] Joachim Steinbach and Michael Zehnter. Vade-mecum of polynomial orderings. Report SR-90-03, Fachbereich Informatik, Universität Kaiserslautern, Kaiserslautern, West Germany, 1990.
[36] Hans Zantema. Termination of term rewriting by semantic labelling. Technical Report RUU-CS-92-38, Utecht University, Utrecht, the Netherlands, December 1992.
[37] Hans Zantema, personal communication.