Well-Founded Orderings for Proving Termination of Systems of Rewrite Rules

David A. Plaisted

University of Illinois
Urbana, Illinois

Report R-78-932 July 1978

Abstract: Well-founded partial orderings can be used to prove the termination of programs, and can also be used for algebraic simplification. A new class of well-founded orderings is presented which can be used to prove the termination of programs expressed as sets of rewrite rules. The orderings are syntactically defined in terms of a lexicographic ordering and an ordering on multisets and require an ordering on the function and constant symbols to be specified. This technique of proving termination appears to be of practical importance, because it is able to handle rewrite rules that arise from typical recursive programs. Several efficient algorithms are presented which allow the termination of a set of rewrite rules to be verified in linear time, in cases to which this method applies. These results can be viewed in terms of an incomplete system of logic in which short termination proofs exist. The well-founded orderings may also be useful for proofs by mathematical induction in various areas of mathematics. A general characterization of a class of rewrite rules is presented, for which termination can be proved using these orderings.

Key Words and Phrases: proof of termination, total correctness, well-founded sets, rewrite rules, term rewriting systems, tree replacement systems, equational systems, reductions, mathematical induction, sorting, algebraic simplification, theorem proving, recursive programs, recursive schemata, partial orderings.

CR Categories: 4.22, 5.21, 5.24, 5.7.

This research was supported in part by the National Science Foundation under Grant MCS 77-22830.

Authors' address: Department of Computer Science, University of Illinois, Urbana, Illinois 61801.

1. Introduction

Many programs can conveniently be expressed as sets of term rewriting rules. For example, a program to compute the factorial function can be expressed as follows:

$$fact(s(x)) \rightarrow s(x) * fact(x)$$

$$fact\ (0) \rightarrow 1$$

The append function in LISP can be expressed this way:

$$append\ (cons(x,y),z) \rightarrow cons(x,append(y,z))$$

$$append\ (NIL,z) \rightarrow z$$

We are interested in efficient, general methods for proving that such sets of rewrite rules always terminate. Although this is in general an undecidable problem [6], we have found a method that works well in many cases of practical interest. For related work, see [4], [5], [7], [10], [11].

Suppose that

$$s_1 \rightarrow t_1$$

$$\cdots$$

$$s_n \rightarrow t_n$$

is a set of rewrite rules. We have found a class of partial orderings on terms with the following properties:

1. If $t_i < s_i$ in the ordering for $1 \leq i \leq n$, then the set of rewrite rules is guaranteed to always terminate.
2. Given s_i and t_i, there is an efficient procedure for deciding if $t_i < s_i$ in the ordering.

This paper presents an efficient, simple method for proving termination of sets of rewrite rules in many cases of practical interest. Common cases that our method cannot deal with are also presented.

Partial orderings of the kind we investigate are useful in devising general methods for algebraic simplification. Some of the most powerful theorem provers currently in use rely heavily on general simplification procedures ([1], [2], [3], [12], [13]). Also Lankford [9] has described a complete theorem proving strategy for the first-order predicate calculus which is based on the concept of simplification. See [8] for another application of such orderings.

Our approach is related to that of Dershowitz [4]. He presents many specialized methods, some of which can provide termination proofs that our approach cannot provide. We present a single, general method which can handle cases not covered by Dershowitz's techniques. Huet [5] is primarily concerned with showing that term rewriting systems are "confluent", and not with exhibiting particular well-founded orderings. Iturriaga's work [7] is hard to understand and not generally available. Lipton and Snyder [10] give another method for proving termination.

Of all the above approaches, only Dershowitz's approach and the approach presented here are suitable for proving termination of rewrite rules of the kind obtained when writing recursive programs. Dershowitz's nested multiset ordering is directly applicable only to one function symbol at a time, whereas the ordering presented here is directly applicable to many function symbols at the same time. Also, we give explicit programs for computing the ordering and analyze their time complexity. In addition, we give a general characterization of a class of rewrite rules all of which can be handled by our ordering. Both of these techniques seem to be new in this area, although Dershowitz may use the latter. Perhaps Iturriaga's ordering can be adapted to our purposes, but he gives neither an explicit program nor a general characterization for his ordering.

2. Simplification Orderings

Definition A term is an expression formed from function symbols, constant symbols, and variables, properly combined. Thus f(x,g(c,y)) is a term. We consider constant symbols as function symbols having no arguments. A term without variables in it is a ground term. We use the usual notion of substitution. A substitution is considered as a multiple replacement, simultaneously replacing all variables of a term by terms. Thus $\{x \leftarrow f(x), y \leftarrow g(c)\}$ is a substitution. We assume all function and constant symbols are taken from some set F of symbols.

Definition: We say an ordering "<" is well-founded if there is no infinite sequence $t_1, t_2, t_3, \ldots$ such that $t_i > t_{i+1}$ for all $i \geq 1$. (Such a sequence $t_1, t_2, t_3, \ldots$ is called an infinite descending sequence.)

Definition: A partial ordering "<" on terms is a simplification ordering if it has the following four properties:

1. It is a total ordering on ground terms.

2. It is a well-founded ordering. (That is, there are no infinite descending sequences.)

3. (Consistency with respect to substitution) If t_1 and t_2 are terms and $t_1 < t_2$ in the ordering, then for all substitutions θ, $t_1\theta < t_2\theta$ in the ordering.

4. (Consistency with respect to subterm replacement) If s_1 is a subterm of t_1, and t_2 obtained from t_1 by replacing s_1 by s_2, and $s_1 < s_2$ in the ordering, then $t_1 < t_2$ in the ordering.

Intuitively, these are desirable properties for a simplification ordering to have. Property 2 guarantees that the simplification process

must terminate. Property 4 guarantees that simplifying a subterm will also simplify the whole term, so simplification can be done "locally."

Definition: A partial ordering on terms is a partial simplification ordering if it has properties 2, 3, and 4 as above. Thus it need not be a total ordering on ground terms.

Theorem 2.1. If s is a subterm of t then s is never greater than t in any partial simplification ordering.

Theorem 2.2. If s is a ground term and t is a non-ground term, then s is never greater than t in any partial simplification ordering.

A replacement is an equation that can only be used in one direction. We write $s_1 \rightarrow s_2$ to mean that any instance of s_1 can be replaced by the corresponding instance of s_2.

Definition: A replacement $s_1 \rightarrow s_2$ is a simplification with respect to a simplification ordering if s_2 is less than s_1 with respect to the ordering.

Definition: A term t is obtained from term s using the replacement $s_i \rightarrow t_i$ if there is some substitution θ with the following properties:

$s_i\theta$ is a subterm of s, and t is obtained from s by replacing one occurrence of $s_i\theta$ in s by $t_i\theta$.

Note that if $t_i < s_i$ in some partial simplification ordering then $t_i\theta < s_i\theta$ also by consistency with respect to substitution. Hence $t < s$ by consistency with respect to subterm replacement.

We say that a set $s_1 \to t_1$, $s_2 \to t_2$, ..., $s_n \to t_n$ of rewrite rules fails to terminate on input u_1 if there is an infinite sequence $u_1, u_2, u_3, \ldots$ such that for all $i \geq 1$, u_{i+1} is obtained from u_i using some replacement in the set. If no such infinite sequence exists, we say the set of rewrite rules terminates on input u_1.

Theorem 2.3. Suppose $s_1 \to t_1$, ..., $s_n \to t_n$ is a set of rewrite rules. Suppose there is a partial simplification ordering "<" such that $t_i < s_i$ in the ordering for $1 \leq i \leq n$. Then the set of rewrite rules terminates on all inputs.

Proof: Assume the rules fail to terminate on input u_1. Let $u_1, u_2, u_3, \ldots$ be an infinite sequence of terms such that for all $i \geq 1$, u_{i+1} is obtained from u_i by using some replacement in the set of rewrite rules. We showed above that $u_{i+1} < u_i$ in the ordering, for all i. Hence $u_1, u_2, u_3, \ldots$ is an infinite descending sequence in the ordering. But this is impossible, because a partial simplification ordering is well-founded. Thus the given set of rewrite rules is guaranteed to terminate on all inputs.

3. Multisets

A multiset is a set in which an element can occur more than once. We write multiset union as $\uplus$ or usually $\cup$. The number of occurrences of an element in a multiset $A \cup B$ is the sum of the number of its occurrences in A and B.

Definition: A sequence of function symbols (or sequence) is a sequence of function and constant symbols from F.

Definition: If term t is of the form $f(t_1, \ldots, t_n)$ then f is the top-level function symbol of t. Also, $\{t_1, \ldots, t_n\}$ are top-level subterms of t.

Definition: If t is a ground term of form $f(t_1, \ldots, t_n)$ then a path in t is a sequence beginning with f and followed by a path from some top-level subterm of t.

Definition: If t is a ground term then Paths (t) is the multiset of paths of t. Thus

$$\text{Paths}(f(t_1, \ldots, t_n)) = \{f\} \overset{n}{\underset{i=1}{\Psi}} \text{Paths}(t_i).$$

Paths(c) = {c} for constant symbols c. Thus Paths (f(c,g(a,b))) = {fc, fga, fgb}.

Definition: If α is a path then Subseq(α) is the multiset of subsequences of α.

$$\text{Thus Subseq}(f\alpha) = \{f,\Lambda\}\ \text{Subseq}(\alpha)$$

$$= \{f\}\ \text{Subseq}(\alpha) \Psi\ \text{Subseq}(\alpha)$$

where Λ denotes the empty sequence and concatenation denotes subsequence concatenation. Hence Subseq(fg) = {f,g,fg,Λ}.

Definition: If S is a finite multiset of elements ordered by some total order relation, let List (S) be a list $\{s_1, s_2, \ldots, s_m\}$ of the elements of S in non-increasing order. Each element must appear the same number of times in List (S) as in S.

Given a total ordering on some set S, we define a total ordering on finite multisets of elements of S as follows:

Suppose U and V are multisets of elements of S. Suppose $U \neq V$. Let $\{u_1, u_2, \ldots u_k\}$ be List (U) and let $\{v_1, v_2, \ldots, v_\ell\}$ be List(V).

If List (U) is a proper prefix of List(V), we say U < V. If List (V) is a proper prefix of List (U), we say V < U. Otherwise, let j be min{i: $u_i \neq v_i$}. Then we say U < V if $u_j < v_j$, U > V if $u_j > v_j$.

Note that if the ordering on S is well-founded, so is the ordering on multisets of elements of S.

We assume that there is a total ordering on the function and constant symbols F appearing in terms t that we are dealing with. If g > f in this ordering, we intuitively think of g as being more complicated than f, and we say g is larger than f.

We order sequences of symbols lexicographically. Suppose $\overline{F} = (f_0, f_1, \ldots f_k)$ and $\overline{G} = (g_0, g_1, \ldots, g_\ell)$ are two such sequences of symbols. Suppose $\overline{F} \neq \overline{G}$. If $\overline{F}$ is a prefix of $\overline{G}$, we say that $\overline{F} < \overline{G}$ in the lexicographic ordering. If $\overline{G}$ is a prefix of $\overline{F}$, we say $\overline{G} < \overline{F}$. Otherwise, let j be min{i: $f_i \neq g_i$}. We say $\overline{F} < \overline{G}$ if $f_j < g_j$, $\overline{F} > \overline{G}$ otherwise. Although this is <u>not</u> a well-founded ordering on sequences, useful well-founded orderings can be obtained from it.

4. A Subsequence Ordering on Paths

We order <u>multisets</u> of sequences by extending the sequence ordering to multisets as indicated above. If α and β are paths, we say $\alpha < \beta$ in the subsequence ordering if Subseq(α) < Subseq(β) in the multiset ordering for subsequences.

Note that the subsequence ordering on paths is a total ordering.

Theorem 4.1. For all paths α, β and for all function symbols f, Subseq(α) < Subseq(β) iff Subseq(fα) < Subseq(fβ).

The proof is quite simple.

Definition: If α is a path and g is a function symbol, let $\#_g(\alpha)$ be the number of occurrences of g in α.

Definition: If α is a path, let maxf(α) be the maximal function symbol in α. Let maxf(α,β) be the maximal function symbol in α or β, if α and β are paths.

Let $R(\alpha,\beta)$ be the following relation on paths α and β:

Suppose g is maxf(α,β). Suppose α is $\alpha_1 g\alpha_2 g \ldots \alpha_m g\alpha_{m+1}$ and β is $\beta_1 g\beta_2 g \ldots \beta_n g\beta_{n+1}$ where $\#_g(\alpha)=m$ and $\#_g(\beta)=n$. Then $R(\alpha,\beta)$ is true if $m<n$ or if ($m=n$ and $R(\alpha_j,\beta_j)$ is true, where $j=\max\{i:\alpha_i \neq \beta_i\}$).

Theorem 4.2. Suppose α and β are paths. Then

a) $\alpha < \beta$ in the path ordering iff Subseq(α){f} < Subseq(β){f} in the path ordering and

b) $\alpha < \beta$ in the path ordering iff $R(\alpha,\beta)$ is true.

Proof: By induction on maxf(α,β). If these values are the same, then by induction on $\max(\#_g(\alpha),\#_g(\beta))$.

Note incidentally that it is easy to show that $R(\alpha,\beta)$ implies $R(\alpha f,\beta f)$ for all function symbols f.

Assume that α is $\alpha_1 g\alpha_2 g \ldots \alpha_m g\alpha_{m+1}$ and β is $\beta_1 g\beta_2 g \ldots \beta_n g\beta_{n+1}$ where $\#_g(\alpha)=m$ and $\#_g(\beta)=n$ and g is maxf(α, β).

We show $R(\alpha,\beta)$ iff $\alpha < \beta$. If $m < n$, then Subseq(α) < Subseq(β) easily. If $m=n$, then List(α) begins with g^m List(α_{m+1}') and List (β) begins with g^m List(β_{m+1}).

Let j be $\max\{i: \alpha_i \neq \beta_i\}$. If $j = m+1$ then $\alpha < \beta$ iff $\alpha_{m+1} < \beta_{m+1}$. Suppose $m = n$ and $j < m + 1$. Then $\text{Subseq}(\alpha) = \text{Subseq}(\alpha'\alpha_j g\gamma)$ and $\text{Subseq}(\beta) = \text{Subseq}(\beta'\beta_j g\gamma)$ for some γ where $\#_g(\gamma) = m - j$. Suppose $\alpha_j < \beta_j$ in the path ordering. Now, $\text{Subseq}(\alpha) = \text{Subseq}(\alpha'\alpha_j\gamma) \uplus \text{Subseq}(\alpha'\alpha_j)\{g\}\text{Subseq}(\gamma)$ and $\text{Subseq}(\beta) = \text{Subseq}(\beta'\beta_j\gamma) \uplus \text{Subseq}(\beta'\beta_j)\{g\}\ \text{Subseq}(\gamma)$. We can assume inductively that $R(\alpha'\alpha_j\gamma, \beta'\beta_j\gamma)$ iff $\alpha'\alpha_j\gamma < \beta'\beta_j\gamma$ since $\alpha'\alpha_j\gamma$ and $\beta'\beta_j\gamma$ either have no g at all or else have fewer g than α and β do. Now, $\alpha'\alpha_j\gamma$ is $\alpha_1 g\alpha_2 g\ldots g\alpha_j\alpha_{j+1}g\ldots\alpha_m g\alpha_{m+1}$ and $\beta'\beta_j\gamma$ is $\beta_1 g\beta_2 g\ldots g\beta_j\beta_{j+1}g\ldots\beta_m g\beta_{m+1}$. Also, $\alpha_{m+1} = \beta_{m+1}$, $\ldots$, $\alpha_{j+1} = \beta_{j+1}$. Hence $R(\alpha'\alpha_j\gamma, \beta'\beta_j\gamma)$ is true iff $R(\alpha_j\alpha_{j+1}, \beta_j\beta_{j+1})$ is true, i.e., iff $R(\alpha_j\alpha_{j+1}, \beta_j\alpha_{j+1})$ is true. By repeated application of the note following the theorem, $R(\alpha_j\alpha_{j+1}, \beta_j\alpha_{j+1})$ is true. Hence $R(\alpha'\alpha_j\gamma, \beta'\beta_j\gamma)$ is true and so by inductive application of b), $\alpha'\alpha_j\gamma < \beta'\beta_j\gamma$ in the ordering on paths.

It is not difficult to see that $\text{Subseq}(\alpha'\alpha_j)\{g\}\ \text{Subseq}(\gamma) < \text{Subseq}(\beta'\beta_j)\{g\}\ \text{Subseq}(\gamma)$ iff $\text{Subseq}(\alpha_j)\{g\}\ \text{Subseq}(\gamma) < \text{Subseq}(\beta_j)\{g\}\ \text{Subseq}(\gamma)$.

This is because these two expressions are the same on all subsequences beginning with j or more g's. Their ordering will therefore be determined by subsequences beginning with exactly (j-1) g's. Of these, all subsequences will be the same in both expressions except those beginning with the <u>first</u> j-1 g's.

Now, we can assume inductively by a) that $\mathrm{Subseq}(\alpha_j)\{g\} < \mathrm{Subseq}(\beta_j)\{g\}$. (Since we assumed $\alpha_j < \beta_j$.) We want to show that for all sequences x, $\mathrm{Subseq}(\alpha_j)\{gx\} < \mathrm{Subseq}(\beta_j)\{gx\}$. Note that for $y,z \in \mathrm{Subseq}(\alpha_j) \cup \mathrm{Subseq}(\beta_j)$, $ygx < zgx$ iff $yg < zg$. This is because g does not occur in y or z, hence neither of yg and zg is a prefix of the other.

We thus have a 1-1, order-preserving mapping between $\mathrm{Subseq}(\alpha_j)\{g\} \cup \mathrm{Subseq}(\beta_j)\{g\}$ and $\mathrm{Subseq}(\alpha_j)\{gx\} \cup \mathrm{Subseq}(\beta_j)\{gx\}$. Hence $\mathrm{Subseq}(\alpha_j)\{gx\} < \mathrm{Subseq}(\beta_j)\{gx\}$ iff $\mathrm{Subseq}(\alpha_j)\{g\} < \mathrm{Subseq}(\beta_j)\{g\}$.

Since this is true for each x in $\mathrm{Subseq}(\gamma)$, we obtain that $\mathrm{Subseq}(\alpha_j)\{g\}\mathrm{Subseq}(\gamma) < \mathrm{Subseq}(\beta_j)\{g\}\mathrm{Subseq}(\gamma)$. Hence $\mathrm{Subseq}(\alpha'\alpha_j)\{g\}\mathrm{Subseq}(\gamma) < \mathrm{Subseq}(\beta'\beta_j)\{g\}\mathrm{Subseq}(\gamma)$. Therefore $\mathrm{Subseq}(\alpha) < \mathrm{Subseq}(\beta)$, and we have shown b) by induction.

We now show a). Assume j is as above. If $\#_g(\alpha) < \#_g(\beta)$ and $f \leq g$, then a) follows easily. Suppose $\#_g(\alpha) < \#_g(\beta)$ and $f > g$. Then $\mathrm{List}(\alpha)$ begins with $f, m*gf, \binom{m}{2}*ggf, \ldots$ and $\mathrm{List}(\beta)$ begins with $f, n*gf, \binom{n}{2}*ggf, \ldots$. Hence if $m < n$ and $f > g$, $\mathrm{Subseq}(\alpha)\{f\} < \mathrm{Subseq}(\beta)\{f\}$. Suppose $\#_g(\alpha) = \#_g(\beta)$. Suppose $f > g$. Then $\mathrm{List}(\mathrm{Subseq}(\alpha)\{f\})$ and $\mathrm{List}(\mathrm{Subseq}(\beta)\{f\})$ will both begin with $f, m*gf, \binom{m}{2}*ggf, \ldots, \binom{m}{m-1}g^{m-1}f, g^m f$. After this, they will agree on all sequences having j or more g's at the beginning. Also, they will agree

on all sequences having j-1 g's not all of them chosen from the first j-1 g's. Hence the ordering on α and β will be determined by the ordering on $Subseq(\alpha_j g\gamma)(f)$ and $Subseq(\beta_j g\gamma)(f)$ for suitable γ. As before, $Subseq(\alpha_j g\gamma)(f) = Subseq(\alpha_j\gamma)(f) \Psi\ Subseq(\alpha_j)(g)\ Subseq(\gamma)(f)$ and $Subseq(\beta_j g\gamma)(f) = Subseq(\beta_j\gamma)(f) \Psi\ Subseq(\beta_j)(g)\ Subseq(\gamma)(f)$. Using induction and an argument as above, we obtain that $Subseq(\alpha)(f) < Subseq(\beta)(f)$.

Suppose $\#_g(\alpha) = \#_g(\beta)$ and $f \leq g$. In this case, $Subseq(\alpha)(f)$ and $Subseq(\beta)(f)$ agree on all subsequences beginning with j or more g's. They also agree on all subsequences beginning with j-1 g's not all chosen from among the first j-1 g's. Hence the ordering on $Subseq(\alpha)(f)$ and $Subseq(\beta)(f)$ is determined by the ordering on $Subseq(\alpha_j g\gamma)(f)$ and $Subseq(\beta_j g\gamma)(f)$, as above. Using induction and an argument as above, we obtain that $Subseq(\alpha)(f) < Subseq(\beta)(f)$. This completes the proof.

Corollary 1: For all paths α,β and all function symbols f, $Subseq(\alpha) < Subseq(\beta)$ iff $Subseq(\alpha f) < Subseq(\beta f)$.

Proof: $Subseq(\alpha f) = Subseq(\alpha) \Psi\ Subseq(\alpha)(f)$ and $Subseq(\beta f) = Subseq(\beta) \Psi\ Subseq(\beta)(f)$. Also, $Subseq(\alpha)(f) < Subseq(\beta)(f)$ iff $Subseq(\alpha) < Subseq(\beta)$ by the above theorem.

Corollary 2: The subsequence ordering on paths is a well-founded ordering.

Proof: By the above theorem, $Subseq(\alpha) < Subseq(\beta)$ iff $R(\alpha,\beta)$. But it is easy to see that the relation R is a well-founded partial ordering on paths.

This result is slightly surprising, since the multiset ordering on arbitrary multisets of sequences is not well-founded. However, if we restrict ourselves to multisets of sequences obtained from paths, this ordering is well-founded.

Definition: If α is a path, let Desc(α) be the multiset of non-increasing subsequences of α.

Theorem 4.3. For all paths α and β, Subseq(α) < Subseq(β) iff Desc(α) < Desc (β).

Proof: By induction on maxf(α,β). Let α be $\alpha_1 g \alpha_2 g \ldots \alpha_n g \alpha_{n+1}$ and let β be $\beta_1 g \beta_2 g \ldots \beta_n g \beta_{n+1}$, where $g = \text{maxf}(\alpha,\beta)$ and $\#_g(\alpha) = m$ and $\#_g(\beta) = n$.

We show Subseq(α) < Subseq(β) implies Desc(α) < Desc(β). It is clear that Subseq(α) = Subseq(β) implies Desc(α) = Desc (β), and Subseq(α) > Subseq(β) implies Desc(α) > Desc(β) by symmetry.

Assume that Subseq(α) < Subseq(β). If $m < n$, then the maximal element of Desc(α) is of the form $g^m\alpha'$ and the maximal element of Desc(β) is of the form $g^n\beta'$ where $\#_g(\alpha') = 0$ and $\#_g(\beta') = 0$. Since $g^m\alpha' < g^n\beta'$ in the lexicographic ordering on sequences, Desc(α) < Desc(β).

Suppose that $m=n$. Let j be $\max\{i:\alpha_i \neq \beta_i\}$. We know then that Subseq(α_j) < Subseq(β_j). Also, Desc(α) and Desc(β) will agree on all subsequences beginning with j or more g's, and on all subsequences beginning with j-1 g's not all chosen from the first j-1 g's of α and β. Hence the largest subsequences on which Desc(α) and Desc(β) will differ will begin with g^{j-1} followed by a subsequence not containing any g's. Hence Desc(α) < Desc(β) iff $g^{j-1}\text{Desc}(\alpha_j\alpha_{j+1}\ldots\alpha_{m+1}) < g^{j-1}\text{Desc}(\beta_j\beta_{j+1}\ldots\beta_{m+1})$.

We know that $\alpha_{j+1} = \beta_{j+1}, \ldots, \alpha_{m+1} = \beta_{m+1}$. Also, Subseq($\alpha_j$)< Subseq($\beta_j$). Hence $\text{Subseq}(\alpha_j\alpha_{j+1}\ldots\alpha_{m+1}) < \text{Subseq}(\beta_j\beta_{j+1}\ldots\beta_{m+1})$. We can assume inductively, therefore, that $\text{Desc}(\alpha_j\alpha_{j+1}\ldots\alpha_{m+1}) < \text{Desc}(\beta_j\beta_{j+1}\ldots\beta_{m+1})$. Hence $g^{j-1}\text{Desc}(\alpha_j\alpha_{j+1}\ldots\alpha_{m+1}) < g^{j-1}\text{Desc}(\beta_j\beta_{j+1}\ldots\beta_{m+1})$ and the theorem is proven.

Note that the lexicographic ordering on non-increasing sequences of function symbols is a well-founded ordering. Hence the induced multiset ordering on multisets of such sequences is also well-founded. This gives us another way to show that the subsequence ordering on paths is well-founded.

We now derive some results which are related to an efficient algorithm for computing the subsequence ordering on paths.

Theorem 4.4. Suppose x is the maximal subsequence of a path α. Then x is a non-increasing sequence and the last symbol of x is the same as the last symbol of α.

Theorem 4.5. Suppose x and y are the maximal subsequences of α and β, respectively. Suppose the last symbols of α and β are different. Then x and y differ, and so Subseq(α) < Subseq(β) iff x < y.

Theorem 4.6. Suppose α and β are paths, and f and g are symbols. Suppose that $\alpha < \beta g$ in the path ordering, and $f < g$ in the symbol ordering. Suppose that the last symbol of α is not g. Then $\alpha f < \beta g$ in the path ordering.

Proof: Let $f_1 f_2 \ldots f_m$ be the maximal subsequence of α, and let $g_1 g_2 \ldots g_n$ be the maximal subsequence of βg. Note that both subsequences will be non-increasing sequences. Since the last symbols of α and βg differ, the sequences $f_1 f_2 \ldots f_m$ and $g_1 g_2 \ldots g_n$ differ, so $f_1 f_2 \ldots f_m < g_1 g_2 \ldots g_n$ in the lexicographic ordering. Since g_n is g and $g_1 g_2 \ldots g_n$ is a non-increasing sequence, $g_i \geq g$ for all i, $1 \leq i \leq n$. The maximal subsequence of αf will be $f_1 f_2 \ldots f_j f$ for some j, $0 \leq j \leq m$. We know that $f_i < f$ for all i, $j < i \leq m$. If $g_1 g_2 \ldots g_j$ is not identical to $f_1 f_2 \ldots f_j$ then $g_1 g_2 \ldots g_n > f_1 f_2 \ldots f_j f$ since $g_1 g_2 \ldots g_n > f_1 f_2 \ldots f_m$.

Suppose $g_1g_2\ldots g_j$ is identical to $f_1f_2\ldots f_j$. Then $j < n$, because $j = n$ implies $\alpha > \beta g$ in the path ordering. Since $f < g$, we have $f < g_{j+1}$ and so $f_1f_2\ldots f_jf < g_1g_2\ldots g_jg_{j+1}\ldots g_n$. Thus $\alpha f < \beta g$ in the path ordering.

Theorem 4.7. Suppose that α and β are paths. Let $\alpha = \alpha'\gamma$ and $\beta = \beta'\gamma$ where γ is the longest common suffix of α and β. Let x and y be the maximal elements of $Desc(\alpha')$ and $Desc(\beta')$, respectively, in the lexicographic ordering on subsequences. Then $Subseq(\alpha) < Subseq(\beta)$ iff $x < y$ in the lexicographic ordering on subsequences. (In fact, x and y are the maximal elements of $Subseq(\alpha')$ and $Subseq(\beta')$, respectively.)

5. A Path Ordering on Terms

Given ground terms t_1 and t_2, we order them by the multiset ordering on their paths. That is, we order paths α and β by the subsequence ordering, i.e., $\alpha < \beta$ iff $Subseq(\alpha) < Subseq(\beta)$. We then order t_1 and t_2 by the multiset ordering on $Paths(t_1)$ and $Paths(t_2)$. Thus, if $List(Paths(t_1)) = (\alpha_1, \alpha_2, \ldots, \alpha_m)$ and $List(Paths(t_2)) = (\beta_1, \beta_2, \ldots, \beta_n)$, we say $t_1 < t_2$ if $List(Paths(t_1))$ is a proper prefix of $List(Paths(t_2))$ or if $Subseq(\alpha_j) < Subseq(\beta_j)$ where $j = \min\{i: \alpha_i \neq \beta_i\}$. Hence we are really using two levels of multisets to order t_1 and t_2: multisets of paths, each path considered as a multiset of sequences.

Note that this gives a well-founded ordering on ground terms, since we are ordering multisets of paths and the ordering on paths is a well-founded ordering.

Definition: The path ordering on terms is defined as follows:

a) If t_1 and t_2 are ground terms, $t_1 < t_2$ if $Paths(t_1) < Paths(t_2)$ in the above ordering.

b) If t_1 and t_2 are non-ground terms, $t_1 < t_2$ if for all θ such that $t_1\theta$ and $t_2\theta$ are ground terms, $Paths(t_1\theta) < Paths(t_2\theta)$ in the above ordering.

Theorem 5. The path ordering on terms is a partial simplification ordering. Note that we may have $Paths(t_1) = Paths(t_2)$ even if t_1 and t_2 are not identical. For example, f(a,b) and f(b,a) have the same multiset of paths. If t_1 and t_2 are ground terms and have unequal multisets of paths, then either $t_1 < t_2$ or $t_2 < t_1$ in the path ordering on terms.

6. Computing the Orderings

Given paths α and β, we present an algorithm to compute whether $Subseq(\alpha) < Subseq(\beta)$. Let $|\alpha|$ be the length of α, i.e., the number of occurrences of symbols in α. Let $\alpha[i]$ be the i^{th} symbol in α, and assume that the first symbol is $\alpha[1]$.

```
procedure order(α,β);
   m ← |α|;
   n ← |β|;
      while α[m] = β[n] and m > 0 and n >0 do (m ← m - 1; n ← n - 1);
      if m = 0 and n = 0 then return ("α = β");
      i ← 1; j ← 1
      while i ≤ m and j ≤ n do
         (if α[i] > β[j] then j ← j + 1 else
         if α[i] < β[j] then i ← i + 1 else
            (i ← i + 1; j ← j + 1););
      if i > m then return ("β > α") else return ("α > β");
end order;
```

This method requires at most $|\alpha| + |\beta| - 1$ comparisons to compute the ordering on α and β. It is easy to show that any comparison-based method of computing this ordering requires at least $|\alpha|$ comparisons on inputs α, β such that $\alpha < \beta$ in the subsequence ordering on paths.

We now show that the procedure "order" is correct. Let γ be the longest common suffix of α and β. Suppose $\alpha=\alpha'\gamma$ and $\beta=\beta'\gamma$. At the end of the first while statement, $m=|\alpha'|$ and $n=|\beta'|$. If $\alpha=\beta$ then $m=n=o$ and the procedure will return "$\alpha=\beta$".

We show inductively that the second while statement, started on strings α' and β' which do not have the same last character, will work correctly. If $\alpha'=\Lambda$ and $\beta'\neq\Lambda$ or if $\alpha'\neq\Lambda$ and $\beta'=\Lambda$, it is clear that the program works correctly. Suppose $\alpha'\neq\Lambda$ and $\beta'\neq\Lambda$ Let α' be $\alpha_1 g_1 \alpha_2$ and let β' be $\beta_1 g_2 \beta_2$, where g_1 is $maxf(\alpha')$ and $\#_{g_1}(\alpha_1)=o$. Also, g_2 is $maxf(\beta')$ and $\#_{g_2}(\beta_1)=o$. Thus we have specified the first occurrences of g_1 and g_2 in α' and β', respectively.

Suppose $g_1 > g_2$. Then i will stop scanning at $|\alpha_1| + 1$ but j will continue all the way through β' and so eventually $j > n$ and the algorithm will return "$\alpha>\beta$", which is correct. Similarly, if $g_2 > g_1$ the algorithm will return "$\beta>\alpha$", which is correct.

Suppose $g_1=g_2$. Then $\alpha>\beta$ iff $\alpha_2 > \beta_2$, by previous results and the fact that $\alpha_2 \neq \beta_2$. We eventually have $i = |\alpha_1| + 1$ and $j = |\beta_1| + 1$. This causes the statement (i←i+1; j←j+1) to execute. From then on, the while statement behaves as if it were started on α_2 and β_2. We can assume inductively, therefore, that the program will output "$\alpha > \beta$" if $\alpha_2 > \beta_2$, and "$\alpha < \beta$" if $\alpha_2 < \beta_2$. But $\alpha_2 > \beta_2$ iff $\alpha > \beta$ and $\alpha_2 < \beta_2$ iff $\alpha < \beta$, so the program is correct. This completes the proof.

The following algorithm orders two paths α and β in the subsequence ordering on paths. It scans α and β from back to front rather than from front to back. Notation is as in the previous algorithm.

```
procedure orderb(α,β);
    i←|α|;
    j←|β|;
    while α[i] = β[j] and i>0 and j>0 do (i←i-1; j←j-1);
    if i=0 and j=0 then return("α=β");
    if i=0 and j>0 then return("α<β");
    if i>0 and j=0 then return("α>β");
L1: if i>0 then
      (i←i-1;
      if α [i]<β[j] then goto L1 else goto L2)
      else return("α<β");
L2: if j>o then
      (j←j-1;
      if α[i]>β[j] then goto L2 else goto L1)
      else return("α>β");
    end orderb;
```

The reason this algorithm works is the following:

Let $N(\gamma_1,\gamma_2)$ be the following relation on paths γ_1 and γ_2: $\gamma_1 < \gamma_2$ in the subsequence ordering, but $\gamma_1 >$ (γ_2 with the first symbol removed) in the subsequence ordering.

Then for all paths γ_1, γ_2 we have that $N(\gamma_1, g\gamma_2)$ and $f < g$ implies $N(f\gamma_1, g\gamma_2)$. $N(\gamma_1, g\gamma_2)$ and $f \geq g$ implies $N(g\gamma_2, f\gamma_1)$. In the above algorithm, let γ be the longest common suffix of α and β. Suppose $\alpha=\alpha'\gamma$ and $\beta=\beta'\gamma$. Let α_1 be $\alpha[i]\alpha[i+1]..\alpha[m]$ and let β_1 be

$\beta[j]\beta[j+1]\ldots\beta[n]$, where $m=|\alpha'|$ and $n=|\beta'|$. Then at L1, $N(\alpha_1, \beta_1)$ is always true and at L2, $N(\beta_1, \alpha_1)$ is always true.

Given ground terms t_1 and t_2, we compute the path ordering on t_1 and t_2 as follows:

Let S1 be Paths(t_1) and let S2 be Paths(t_2). If S1 = S2 then $t_1 = t_2$ in the ordering. If S1 $\neq$ S2 and S1 $\subset$ S2 then $t_1 < t_2$ in the ordering. If S1 $\neq$ S2 and S2 $\subset$ S1 then $t_2 < t_1$ in the ordering. Otherwise, sort S1 and S2 in non-increasing order using the above procedure to compute the subsequence ordering on paths. We thus obtain List(S1) = $(\alpha_1, \alpha_2, \ldots, \alpha_m)$ and List(S2) = $(\beta_1, \beta_2, \ldots, \beta_n)$. Let j be min{i: $\alpha_i \neq \beta_i$}. Then $t_1 < t_2$ in the path ordering on terms if $\alpha_j < \beta_j$ in the subsequence ordering on paths. Otherwise $t_1 > t_2$.

Let Size(t) be the number of occurrences of function and constant symbols in a term t. Suppose Size(t_1) = q_1 and Size(t_2) = q_2. Then S1 has at most q_1 elements. Also, each element of S1 is of length at most q_1. Hence each comparison done while sorting S1 requires at most $2q_1 - 1$ comparisons of symbols, and we can sort S1 in $O((q_1 \log q_1)(2q_1 - 1))$ or $O(q_1^2 \log q_1)$ comparisons. Similarly we can sort S2 in $O(q_2^2 \log q_2)$ comparisons. Finding j requires at most $\min(q_1^2, q_2^2)$ comparisons and comparing α_j and β_j requires at most $q_1 + q_2 - 1$ comparisons. So the entire algorithm is of complexity $O(q_1^2 \log q_1 + q_2^2 \log q_2)$ comparisons.

It is possible to do better than this. We present a method which can sort n paths in time O(L min(logk,logn)) where L is the sum of the lengths of the paths and k is the number of distinct function symbols. We present a related method for computing the path ordering on ground

terms t_1 and t_2. The time required is O(Lmin(logk,logL)) where L is size (t_1) + size (t_2) and K is the number of distinct function symbols. Hence this is a linear time method if the set of function symbols is fixed.

Given in paths, the following algorithm will output them in order, smallest first, according to the subsequence ordering on paths. The time required is O(L min(logk, logn)) where L is the sum of the lengths of the paths and k is the number of distinct function symbols.

The algorithm considers v(j) to be the j^{th} path, for $1 \leq j \leq n$, and v(j) is considered to be a stack. Thus f←v(j) removes the first function symbol from v(j) and assigns it to f. The function symbols are identified with the integers {1,2,...,k}. The function symbols are ordered by the usual ordering on the integers. Thus 1<2, 2<3 et cetera. The algorithm makes use of k queues Q_1, Q_2, ..., Q_k together with another queue T. The empty queue (or stack) is denoted by Λ. Stack and queue operations are indicated in the usual way by ← . Thus j←T means remove an element from the queue and assign it to j. Statements such as T←Λ and T←Q_i are not queue operations but ordinary assignment statements. Thus T←Λ sets T to be the empty queue, and T←Q_i sets T to be the current configuration of the queue Q_i. The algorithm outputs a list a_1, a_2, ..., a_n of integers such that the initial values of $v(a_1)$, $v(a_2)$, ..., $v(a_n)$ constitute the list of paths in nondecreasing order.

At any time in the execution of the algorithm, the list $Q_1Q_2...Q_k$ of paths is in increasing order, where each queue is listed from front to back and where we only consider the subsequence ordering on the portion of the path already seen.

It would be interesting to know if there is a similar algorithm which outputs the paths largest first.

```
procedure sort(v,n);
    flag←true;
    T←Λ;
    for j←1 to n do T←j;
    for j←1 to k do Q_j←Λ;
    while flag do
        (while T≠Λ do
            (j←T;
             if v(j)=Λ then output(j)
                       else (f←v(j); Q_f←j););

         if(∃j)(Q_j≠Λ) then
           (i←min{j:Q_j≠Λ}; [Use a priority queue for this]
           T←Q_i; Q_i←Λ)
           else flag←false;);

end sort;
```

To show that procedure "sort" works, it suffices to show that on any pair v(i), v(j) of input strings, it behaves the same as procedure "order". For strings with differing last characters, this is easy to do. For strings with common nontrivial suffixes, this is more difficult but still can be done by considering the relative position of v(i) and v(j) within a queue, if they are in the same queue.

The following algorithm also sorts n paths v(1), v(2), ..., v(n) and outputs the integers {1, 2, ..., n} in an order corresponding to a listing of the paths in non-decreasing order. Notation is as in procedure sort. The difference is that the paths are scanned from back to front rather than from front to back. That is, f←v(j) means "Let f be the last symbol in path v(j) and delete this symbol from the end of v(j)." However, the queue operations $j \leftarrow Q_i$ et cetera are as before. This algorithm is more useful than the preceding one in certain situations.

```
procedure sortb(v,n);
    for j=1 to k do Q_j←Λ;
    for j=1 to n do
        if v(j)=Λ then output(j) else
                (f←v(j); Q_f←j);
    while (∃j)(Q_j≠Λ) do
        (i←min{j:Q_j≠Λ}; [Use a priority queue for this]
        j←Q_i;
        if v(j)=Λ then output(j) else
                (f←v(j); Q_f←j););
end sortb;
```

We can verify that "sortb" works by showing that it treats input strings v(i), v(j) the same way that "orderb" does. Common nontrivial suffixes do not present a problem because the strings are scanned from back to front.

We present an algorithm which will, given a set $\{t_1,\ldots,t_n\}$ of ground terms, sort the multiset $\biguplus_{i=1}^{n}$ Paths (t_i) in time $O(L\min(\log k, \log L))$ where $L = \sum_{i=1}^{n}$ Size(t_i) and k is the number of distinct function symbols in the terms t_i. For purposes of this algorithm, we associate subterms of a term t with prefixes of paths in t. If the same subterm occurs more than once in t, we associate a prefix of a path with each occurrence of the subterm in t.

Suppose t is of the form $f(v_1,\ldots,v_m)$. With t itself we associate the path consisting of the single function symbol f. Suppose w is a subterm of v_i for sum i, $1\leq i\leq m$. Suppose that α is the path associated with w as a subterm of v_i. Then $f\alpha$ is the path associated with w as a subterm of t.

For example, if t is f(g(a,b),c) then the path associated with b is fgb, the path associated with g(a,b) is fg, and the path associated with t itself is f.

Sorting $\biguplus_{i=1}^{n}$ Paths(t_i) is therefore equivalent to sorting the occurrences of constant symbols in the terms t_i, $1\leq i\leq n$. This is because there is a 1-1 correspondence between elements of Paths(t) and constants in t, using the above association of paths and occurrences of subterms.

Recall that the top-level subterms of a term $f(v_1,\ldots,v_m)$ form a multiset $\{v_1,v_2,\ldots,v_m\}$. Assume the function symbols are the integers $\{1,2,\ldots,k\}$, and let notation be as in procedure sort.

The following algorithm will output in non-decreasing order the elements of $\biguplus_{i=1}^{n}$ Paths(t_i), each path represented by an occurrence of a constant symbol in some term t_i. We assume that enough information is kept with each occurrence of a constant symbol to identify which path it corresponds to when it is printed out. The last "for" statement must deal with each occurrence of a top-level subterm v of t separately.

```
procedure sortt({t_1,t_2,...,t_n});
    T←Λ;
    for j←1 to k do Q_j←Λ;
    for j←1 to n do
        (f←top-level function symbol of t_j;
         Q_f←t_j);
    while (∃_j)(Q_j≠Λ) do
        (i←min {j:Q_j≠Λ};   [Use a priority queue for this]
        T←Q_i; Q_i←Λ;
        while T≠Λ do
            (t←T;
            if t is a term consisting of a single constant symbol then
                output(t)
                else
                for all top-level subterms v of t do
                    (f←top-level function symbol of v;
                     Q_f←v);););
end sortt;
```

This algorithm can be used to compute the path ordering on two ground terms in linear time, assuming that the number of function symbols is bounded. We do not know whether this technique can be extended to sort a set of arbitrarily many ground terms in linear time, assuming that the number of function symbols is bounded.

The output of procedure sortt can be simplified, for most purposes. Suppose $\alpha_1, \alpha_2, \ldots, \alpha_\rho$ is the list of paths of $\bigcup_{i=1}^{n} \text{Paths}(t_i)$

in non-decreasing order. Suppose path α_j comes from term a_j, $a_j \in \{1, 2, \ldots, n\}$ for all $j, 1 \leq j \leq p$. Then all we usually need is the list $(a_1, b_1)(a_2, b_2) \ldots (a_p, b_p)$ where b_1 is arbitrary and $b_j = 1$ if α_j and α_{j-1} are identical, $b_j = 0$ otherwise. If path α_j occurs in more than one term, we are assuming that it is counted the right number of times as coming from each term. By scanning this list in reverse, we can easily find the largest term in $\{t_1, t_2, \ldots, t_n\}$ in linear time. It is not difficult to modify the procedure sortt so that it computes the bits b_j correctly.

Define Paths(t) for a non-ground term t as follows:

Paths(x) = {x} for variables x

Paths(c) = {c} for constant symbols c

$\text{Paths}(f(t_1, \ldots, t_n)) = \{f\} \underset{i}{\psi} \text{Paths}(t_i)$

Thus $\text{Paths}(f(g(x_1, c), x_2)) =$

$\{fgx_1, fgc, fx_2\}$.

Define Size(t) for a non-ground term t to be the number of occurrences of function and constant symbols and variables in it.

Given not necessarily ground terms t_1 and t_2, we compute the path ordering on t_1 and t_2 as follows:

Let $\text{Paths}(t_1)$ be $S_1^1 \cup S_2^1 \cup \ldots \cup S_k^1 \cup R_1$ and let $\text{Paths}(t_2)$ be $S_1^2 \cup S_2^2 \cup \ldots \cup S_k^2 \cup R_2$, where S_j^i and R_i are all multisets. Also, $\{x_1, \ldots, x_k\}$ is the set of variables appearing in t_1 or t_2, and $S_j^i = \{\alpha: \alpha x_j \in \text{Paths}(t_i)\}$. In addition, R_i is the set of paths of t_i ending with a constant symbol.

Thus, if $t_1 = f(g(x_1, c), x_2)$ then $S_1^1 = \{fg\}, S_2^1 = \{f\}, R_1 = \{fgc\}$.

Theorem 6.1. $t_1 < t_2$ in the path ordering on terms iff both of the following are true:

a) For all j, $1 \leq j \leq k$, $S_j^1 \leq S_j^2$ in the ordering on multisets of paths.

b) $t_1\theta < t_2\theta$ in the ordering on terms, where θ is the substitution replacing all variables by the minimal constant symbol of F.

The direct method of applying this theorem yields an algorithm whose worst case complexity is $O(q_1^2 \log q_1 + q_2^2 \log q_2)$ where $q_i = \text{Size}(t_i)$.

Using techniques similar to those mentioned above, we can compute the ordering on non-ground terms in linear time, assuming that the set of function symbols is fixed.

We now develop some results leading up to a proof of the above theorem, and present an efficient algorithm for computing the path ordering on non-ground terms.

Theorem 6.2. Suppose $S_1 \uplus T_1 > S_2 \uplus T_2$ in a multiset ordering, where S_1, T_1, S_2, and T_2 are multisets of elements from some totally ordered set. Suppose $T_1 \geq T_2$ in the multiset ordering. Suppose α is a mapping on elements of T_1 and T_2 which is 1-1, order preserving, and monotone increasing on $T_1 \uplus T_2$. Then $S_1 \uplus (T_1\alpha) > S_2 \uplus (T_2\alpha)$ in the multiset ordering.

Proof: It suffices to consider the case in which $S_1 \cap S_2 = \emptyset$ and $T_1 \cap T_2 = \emptyset$ since elements which occur the same number of times in S_1 and S_2 or T_1 and T_2 don't affect the argument. Assume then that $S_1 \cap S_2 = \emptyset$ and $T_1 \cap T_2 = \emptyset$.

Let s_1, s_2, t_1, and t_2 be maximal elements in S_1, S_2, T_1, and T_2, respectively. Then $t_1 > t_2$ since $T_1 \geq T_2$. Also, $t_1\alpha > t_2\alpha$

since α is 1-1 and order preserving. In addition, $t_1\alpha > t_1$ and $t_2\alpha > t_2$ since α is monotone increasing. Furthermore, $t_1\alpha$ and $t_2\alpha$ are the maximal elements of $T_1\alpha$ and $T_2\alpha$, respectively.

Suppose that $s_1 \geq t_1$. Then $s_1 > s_2$ since $\max(s_1, t_1) \geq \max(s_2, t_2)$ and $S_1 \cap S_2 = \emptyset$. Also, $t_1\alpha > t_2\alpha$ as noted above. Hence $\max(s_1, t_1\alpha) > \max(s_2, t_2\alpha)$ so $S_1 \uplus T_1\alpha > S_2 \uplus T_2\alpha$.

Suppose that $s_1 < t_1$. Then $t_1 \geq s_2$ since $\max(s_1, t_1) \geq \max(s_2, t_2)$. Hence $t_1\alpha > s_2$ since α is monotone increasing. Also, $t_1\alpha > t_2\alpha$ since α is 1-1 and order preserving. Hence $t_1\alpha > \max(s_2, t_2\alpha)$ so $S_1 \uplus T_1\alpha > S_2 \uplus T_2\alpha$.

Corollary: Suppose $S_1 \uplus V_1 \uplus \ldots \uplus V_n > S_2 \uplus W_1 \uplus \ldots \uplus W_n$ in a multiset ordering as above. Suppose $V_j \geq W_j$ for $j = 1, 2, \ldots, n$. Suppose the mappings α_j are 1-1, order preserving, monotone increasing mappings on $V_j \uplus W_j$ for $j = 1, 2, \ldots, n$. Then $S_1 \uplus V_1\alpha_1 \uplus V_2\alpha_2 \uplus \ldots \uplus V_n\alpha_n > S_2 \uplus W_1\alpha_1 \uplus W_2\alpha_2 \uplus \ldots \uplus W_n\alpha_n$.

Proof: By repeated application of the above theorem.

Proof of Theorem 6.1.: Assume $t_1 < t_2$. Then by definition of the path ordering on non-ground terms, $t_1\theta < t_2\theta$ where θ is as in the theorem. So b) is true.

Suppose that for some j, $S_j^1 > S_j^2$ in the ordering on multisets of paths. Choose substitution α to replace x_j by a very large term and x_i by very small terms, for $i \neq j$. Then $t_1\alpha > t_2\alpha$. Hence a) is true if $t_1 < t_2$.

Assume a) and b) are true. We show that $t_1 < t_2$ in the path ordering on terms. Let θ' be a substitution replacing all variables

$x_1, \ldots, x_k$ by ground terms $v_1, \ldots, v_k$, respectively. We show that $t_1\theta' < t_2\theta'$ in the path ordering on terms.

Suppose c is the minimal constant symbol in F. We know by b) that $\text{Paths}(t_1)\{c\} < \text{Paths}(t_2)\{c\}$, where $\text{Paths}(t_1)\{c\}$ denotes $\{xc : x \in \text{Paths}(t_1)\}$ and similarly for $\text{Paths}(t_2)\{c\}$. By a), $S_j^1\{c\} \leq S_j^2\{c\}$ for $1 \leq j \leq k$. Hence $n_j * S_j^1\{c\} \leq n_j * S_j^2\{c\}$ for $1 \leq j \leq k$, where n_j is the number of paths in $\text{Paths}(v_j)$. (Here $n_j * S_j^1\{c\}$ means n_j copies of $S_j^1\{c\}$ unioned together, and $S_j^1\{c\}$ means $\{xc : x \in S_j^1\}$.)

Thus $\Psi_{j=1}^{k} n_j * S_j^1\{c\} \Psi R_1 < \Psi_{j=1}^{k} n_j * S_j^2\{c\} \Psi R_2$ by the above remark and b). Thus $\Psi_{j=1}^{k} S_j^1 \text{Paths}(v_j) \Psi R_1 < \Psi_{j=1}^{k} S_j^2 \text{Paths}(v_j) \Psi R_2$ by the above corollary. (The mappings we are using are those that replace a path of form βc by a path of form $\beta\gamma$ for some $\gamma \in \text{Paths}(v_j)$, $\gamma \neq \{c\}$. Such a mapping is 1-1, order-preserving, and monotone increasing, as required.) Note that $\text{Paths}(t_1\theta') = \Psi_{j=1}^{k} S_j^1 \text{Paths}(v_j) \Psi R_1$ and $\text{Paths}(t_2\theta') = \Psi_{j=1}^{k} S_j^2 \text{Paths}(v_j) \Psi R_2$. Hence $t_1\theta' < t_2\theta'$ in the path ordering on terms. Since this is true for all θ' such that $t_1\theta'$ and $t_2\theta'$ are ground terms, $t_1 < t_2$ in the path ordering on terms. This completes the proof.

If t_1 and t_2 are not necessarily ground terms, then we can still compute the path ordering on t_1 and t_2 in linear time using the above theorem, assuming that the number of function and constant symbols occurring in t_1 and t_2 is bounded. We perform a simple modification to the algorithm for ground terms, as follows: Let x_j, S_j^i, and θ be as in the above theorem. We modify the algorithm so that it prints out two lists:

1. The paths of $\text{Paths}(t_1\theta) \Psi \text{Paths}(t_2\theta)$ in non-decreasing order, each path identified as coming from t_1 or t_2.

2. The paths in $\left[\bigcup_{j=1}^{k} S_j^1\right] \Psi \left[\bigcup_{j=1}^{k} S_j^2\right]$ in non-decreasing order, where each path is identified as belonging to a particular set S_j^i. That is, the values of i and j are also given.

Both of these lists can be computed during the same scan of t_1 and t_2. Also, by reading these lists in reverse order, it can easily be determined in linear time whether the conditions of the theorem are true. Hence we can compute the ordering in linear time, assuming that the number of function symbols is bounded.

A slight subtlety in this and the preceding algorithm is that when scanning the output in reverse, we don't really need to know the whole path for each path on the list. All we need to know is whether the path is identical to the preceding element of the list. Hence the lists can be printed and scanned in linear time, since we only need a constant amount of information per path (assuming that the relevant integer indices don't get too big).

The replacements that the path ordering can handle are very similar to those which Dershowitz's nested multiset ordering can deal with [4]. However, the path ordering can deal with more than one function symbol being "pushed in" at the same time.

7. Examples

The following replacements are all simplifications in the path ordering on terms. Assume function symbols are ordered alphabetically. Thus a < b, b < c et cetera.

$g(f(x,y)) \rightarrow f(g(x),g(y))$

$e(d(x,y)) \rightarrow d(e(x),e(y))$

$g(f(x,y)) \rightarrow ff(g(e(x)),g(e(y)))$

$g(f(x,y)) \rightarrow f(f(g(x),g(y)),f(g(x),g(y)))$

$g(e(x)) \rightarrow f(x,g(x))$

$e(c(x)) \rightarrow d(x,e(x))$

Note that the same ordering can handle all of these replacements at the same time.

8. General Characterization

We now present a method of obtaining a fairly general class of simplifications in the path ordering on terms.

Definition: If s_1 and s_2 are ground terms, then the replacement $s_1 \rightarrow s_2$ is a strong simplification in the path ordering if the maximal element of Paths (s_2) is less than the maximal element of Paths (s_1) in the ordering on paths.

Definition: If s_1 and s_2 are not necessarily ground terms, then the replacement $s_1 \rightarrow s_2$ is a strong simplification in the path ordering if for all substitutions σ such that $s_1\sigma$ and $s_2\sigma$ are ground terms, $s_1\sigma \rightarrow s_2\sigma$ is a strong simplification in the path ordering.

Examples: The following replacements are all strong simplifications in the path ordering on terms, assuming $a < b$, $b < c$, et cetera:

$f(x,y) \rightarrow x$

$e(d(x,y)) \rightarrow e(x)$

$e(d(x,y)) \rightarrow e(c(x))$

In general, if s_2 is a subterm of s_1 then $s_1 \rightarrow s_2$ is a strong simplification in the path ordering. Also, the preceding six examples of simplifications are all strong simplifications in the path ordering.

Theorem 8.1. Let notation be as in the theorem characterizing the path ordering on non-ground terms. Then $t_2 \rightarrow t_1$ is a strong simplification in the path ordering iff both of the following conditions are true:

a) For all j, $1 \leq j \leq k$, the maximal element of S_j^1 is less than the maximal element of S_j^2 in the ordering on paths.

b) The maximal element of Paths $(t_1 0)$ is less than the maximal element of Paths $(t_2 0)$ in the ordering on paths.

We have the following characterization of a frequently occurring class of replacements, all of which are simplifications in the path ordering on terms:

Theorem 8.2. Suppose $t_1, t_2, \ldots, t_n$ and t are terms, possibly with variables in them. Suppose h is the top-level function symbol of t. Also, suppose that $t \rightarrow t_i$ is a strong simplification in the path ordering, for $i = 1,2,\ldots,n$. Suppose $(f_1,\ldots,f_k)$ is a set of function and constant symbols all of which are simpler than h in the symbol ordering. Let u be any term formed from any number of occurrences of $t_1, t_2, \ldots, t_n$ and any number of occurrences of the symbols $(f_1,\ldots,f_k)$. Then the replacement $t \rightarrow u$ is a simplification (in fact a strong simplification) in the path ordering on terms.

Thus we can deal with replacements such as the following:

```
fact(s(x)) → s(x)*fact(x)
count(cons(x,y)) → count(x) + count(y)
h(g(x),g(y)) → h(x,y)
h(g(x),g(y)) → f(h(x,y),h(x,y))
h(g(x),g(y)) → f(h(e(x),e(y)),h(e(x),e(y)))
```

We still cannot deal with replacements in which only one argument of h gets simpler, however. Here is an example of such a replacement:

$$h(g(x),g(y)) \rightarrow e(h(g(x),y))$$

If we could find a general ordering in which such replacements were simplifications, then we could probably handle the distributive replacements of multiplication over addition.

The following result helps us to prove the above theorem:

Theorem 8.3. Suppose α and β are paths such that Subseq (α) < Subseq (β). Let f be a function symbol which is less than the first function symbol of β in the symbol ordering. Then Subseq ($f\alpha$) < Subseq (β).

Corollary: Suppose α and β are paths such that Subseq (α) < Subseq (β). Suppose γ is a path composed entirely of function symbols which are less than the first symbol of β in the symbol ordering. Then Subseq ($\gamma\alpha$) < Subseq (β).

We still cannot deal with the following two useful replacements using this ordering:

$$(x+y) + z \rightarrow x + (y+z)$$
$$x * (y+z) \rightarrow x * y + x * z$$

We are developing other methods which can handle these replacements in addition to replacements such as those discussed above. In fact, we have recently developed methods which can handle the second of the above replacements, in addition to replacements similar to those included in the above characterization.

9. Summary and Conclusions

We have described a simple class of well-founded partial orderings which can be used to obtain short proofs of termination of systems of rewrite rules, in many cases which appear to be of practical interest. These "path of subterm" orderings are based on multisets of sequences of function symbols from the terms we are working with. We have given explicit programs which can decide whether $s < t$ in such an ordering, given terms s and t. These programs run in linear time, assuming that the set of function symbols is fixed. Also, we have described a general class of replacements, all of which are simplifications in the ordering presented. We have presented some limitations of this method of partial ordering. In future work we plan to discuss recent results extending these ideas to overcome some of the limitations of the path of subterms ordering, at the expense of some simplicity.

References

[1] Ballantyne, A. M., and Bledsoe, W. W., Automatic proofs of theorems in analysis using nonstandard techniques. J. ACM 24:3 (1977), pp. 353-374.

[2] Bledsoe, W. W., Non-resolution theorem proving. Artificial Intelligence 9:1 (1977), pp. 1-35.

[3] Boyer, Robert S., and Moore, Strother J., A lemma driven automatic theorem prover for recursive function theory. Proceedings of the 5th International Joint Conference on Artificial Intelligence, Massachusetts Institute of Technology (1977).

[4] Dershowitz, N., personal communication.

[5] Huet, Gerard, Confluent reductions: abstract properties and applications to term rewriting systems. Proceedings of the 18th Annual Symposium on Foundations of Computer Science (1977).

[6] Huet, Gerard and Lankford, Dallas,On the uniform halting problem for term rewriting systems. Report No. 283, IRIA (March 1978).

[7] Iturriaga, R., Contributions to mechanical mathematics. Ph.D. Thesis, Carnegie-Mellon University (1967).

[8] Knuth, D. E., and Bendix, P. B., Simple Word Problems in Universal Algebras. Computational Problems in Abstract Algebra, Leech, Ed., Pergamon Press (1970), pp. 263-297.

[9] Lankford, Dallas S., Canonical algebraic simplification in computational logic. Report No. ATP-25, Southwestern University, Department of Mathematics, Georgetown, Texas (May 1975).

[10] Lipton, R. J., and Snyder, L., On the halting of tree replacement systems. Proceedings of a Conference on Theoretical Computer Science, University of Waterloo, Canada (1977).

[11] Manna, Z., and Ness, S., On the termination of Markov algorithms. Proceedings of the Third Hawaii International Conference on System Sciences (1970).

[12] Suzuki, Norihisa, Automatic program verification II: verifying programs by algebraic and logical reduction. Report No. STAN-CS-74-473, Stanford University (1974).

[13] Weyhrauch, R. W., A users manual for FOL. Stanford Artificial Intelligence Laboratory Memo AIM-235.1 (1977).