A PATH ORDERING FOR PROVING TERMINATION OF TERM REWRITING SYSTEMS

D. Kapur, P. Narendran
Computer Science Branch
Corporate Research and Development
General Electric Company
Schenectady, NY

G. Sivakumar†
Dept. of Computer Science
University of Illinois,
Urbana-Champaign, IL

ABSTRACT

A new partial ordering scheme for proving uniform termination of term rewriting systems is presented. The basic idea is that two terms are compared by comparing the paths through them. It is shown that the ordering is a well-founded simplification ordering and also a strict extension of the recursive path ordering scheme of Dershowitz. Terms can be compared under this path ordering in polynomial time.

1. INTRODUCTION

Term rewriting systems have been found to be widely applicable in many areas of computer science and mathematics, including word problems, unification problems, decision procedures for equational theories, theorem proving, program transformation and synthesis, polynomial simplification, analysis and design of specifications, proving properties by induction, etc. In most of these applications, the Knuth-Bendix completion procedure [11] and its extensions discussed in [1,8,9,13,15] play a crucial role. However, the successful use of the completion procedure crucially depends upon the ability to prove the termination of term rewriting systems that are generated during the course of the completion procedure to obtain a canonical set of rewrite rules. In this connection, many termination orderings have been proposed in the literature including the original Knuth-Bendix ordering based on weights [11], paths of subterm ordering [16], polynomial ordering [12], recursive path ordering (RPO) [2] and its extension based on lexicographic status (LRPO) [7], and recursive decomposition ordering (RDO) [6] and its extension based on lexicographic status (RDOS) [14].

In RRL, a rewrite rule laboratory under development at the Computer Science Branch at General Electric Corporate Research and Development Center, we have implemented RPO as a

† This work was done when Sivakumar was a graduate student in the Dept. of Mathematical Sciences, Rensselaer Polytechnic Institute, Troy, NY. Kapur and Sivakumar were partially supported for this research by the NSF grant MCS-8211621.

way to establish the termination of term rewriting systems [10]. By and large, our experience has been positive with LRPO in terms of its performance and its applicability to a wide range of term rewriting systems; however in some examples, we have found LRPO to be weak to handle terms which are intuitively simple to handle [10]. In this paper, we develop a new termination ordering based on paths in terms which is intuitively simple to understand and which is an extension of RPO.

In the next section, we study examples illustrating what is lacking in RPO. We analyze the definition of RPO pointing out weaknesses in different aspects of the definition of RPO. Section 3 introduces the ordering based on paths. In Section 4, it is shown that the path ordering is a simplification ordering and has the substitution property, which implies that it can be used for proving termination of term rewriting systems [2]. We also show that the new ordering is a strict extension of recursive path ordering. In Section 5, we prove that comparison of two terms under the path ordering can be done in polynomial time; we show an upper-bound of $O(|s|^5 * |t|^5)$, where $|s|$ and $|t|$ are the size of terms s and t being compared. Section 6 is a brief discussion of how the proposed ordering can be extended to incorporate lexicographic status of function symbols [7]. In Section 7, we outline an extension to the path ordering which further generalizes it. In Section 8, we discuss how the path ordering relates to RDO and RDOS.

2. A DISCUSSION OF RECURSIVE PATH ORDERING (RPO)

Consider the equation $a(b(x)) = c(d(x))$ with the precedence $a > d$ and $b > c$. Under RPO the terms $a(b(x))$ and $c(d(x))$ are incomparable, since $b(x) \not\gtrsim c(d(x))$ and $d(x) \not\gtrsim a(b(x))$. But, in our mind, it is clear to us that this equation should be oriented as $a(b(x)) \rightarrow c(d(x))$, since a 'takes care of' d and b 'takes care of' c. In the next section we make this notion of 'taking-care-of' more precise and thereby define the new ordering.

At this point it will be helpful to take a look at 'recursive path ordering' (RPO) and discuss its weaknesses.

2.1 DEFINITION OF RPO

Let $\gtrsim$ denote a quasi-ordering of a set E (i.e, a reflexive and transitive relation on E) and $\sim$ be the equivalence relation $\gtrsim \cap \lesssim$. Let $>$ denote the partial ordering $\gtrsim - \lesssim$; in other words, $a > b$ if and only if $a \gtrsim b$ and $\neg(b \gtrsim a)$. A partial ordering $>$ is *well-founded* if there exists no infinite descending sequence $e_1 > e_2 > \ldots$ of elements of E. A well-founded partial order $>$ on a set E can be extended to the set of *multisets* on E as follows:

$M_1 \gg M_2$ if and only if for each $x \in M_2 - M_1$, there is a $y \in M_1 - M_2$ such that $y > x$.

A finite multiset M is written by enumerating its elements and enclosing them within braces.

Let $\gtrsim$ be a quasi-ordering of a set of function symbols F. Then terms $s = f(s_1, \cdots, s_m)$ and $t = g(t_1, \cdots, t_n)$ are equivalent (denoted by $s \sim t$) if and only if $f \sim g$, $m = n$ and there is a permutation π of the set {1, ... ,n} such that $s_i \sim t_{\pi(i)}$ for all $1 \leq i \leq n$.

Given a quasi-ordering $\gtrsim$ of a set of function symbols F, the *recursive path ordering* $(>_{rpo})$ on terms generated by F can be defined as follows:

(a) $s = f(s_1, \cdots, s_m) >_{rpo} t = x$ if and only if $x \in Var(s)$, the set of variables in s.

(b) $s = f(s_1, \cdots, s_m) >_{rpo} t = g(t_1, \cdots, t_n)$ if and only if

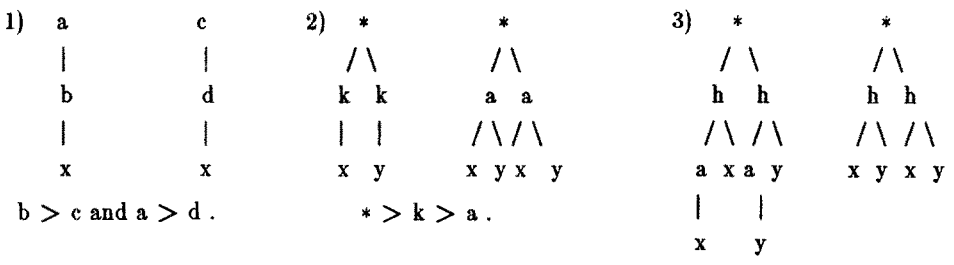
(1) $f > g$ and $s >_{rpo} t_i$ for all i, $1 \leq i \leq n$, or,

(2) $f \sim g$ and $\{ s_1, \ldots, s_m \} \gg_{rpo} \{ t_1, \ldots, t_n \}$, or,

(3) $s_i \gtrsim_{rpo} t$ for some i, $1 \leq i \leq m$.

2.2 WHERE RPO DOES NOT WORK

We give below a few examples of terms which can definitely be ordered by the intuitive concept of paths but which RPO cannot order.



Let $t = f(t_1, \cdots, t_n)$, $s = g(s_1, \cdots, s_m)$, $M_1 = \{t_1, \cdots, t_n\}$ and $M_2 = \{s_1, \cdots, s_m\}$. Now when RPO tries to order these terms it first compares root-symbols. There are 3 cases:

1) $f > g$. Then RPO needs $t >_{rpo} s_i$. This is most powerful and needs no change.

2) $f \sim g$. Then RPO needs $M_1 \gg_{rpo} M_2$.

This is not really the best case for this demands that for every s_i in $M_2 - M_1$ there is a single term t_j in $M_1 - M_2$ such that $t_j >_{rpo} s_i$. This may not be suited when for each path in s_i there are paths in different t_j's which are "bigger". (See examples 2, 3). So we need to treat the terms collectively.

3) $f \not\gtrsim g$. Here again RPO is not ideal for it needs a single term t_j such that $t_j \gtrsim_{rpo} s$. We

also do not remember the root (topmost) operator f of t any more even though it may be needed to make a path "bigger".(See example 1).

3. PATH ORDERING

Let F be a finite set of function symbols of fixed arity and V be a denumerable set of variables. By $T(F,V)$ we denote the set of all possible terms that can be constructed using F and V. For a term t, $Var(t)$ denotes the set of all variables that occur in t. For example, $Var(f(x,y,g(y))) = \{x,y\}$. The *size* of a term s is the total number of function and variable symbol occurrences in s and is denoted by $|s|$. By $f(\ldots t \ldots)$ we denote a term that contains the term t as an immediate (top-level) subterm.

A *term rewriting system* P over a set of terms $T(F,V)$ is a finite set of rewrite rules of the form $l_i \rightarrow r_i$ where $l_i, r_i \in T(F,V)$. We write $t \Rightarrow t'$ to indicate that the term t' can be derived from the term t by a single application of a rule P to one of its subterms. P is said to be *uniformly terminating* if there exist no infinite sequences of terms $t_i \in T(F,V)$ such that $t_1 \Rightarrow t_2 \Rightarrow \ldots$.

3.1 PATHS

A *path* is a sequence of two-tuples ending possibly in a variable, with the following properties: Let $P = <f_1, t_1> \ldots <f_n, t_n>\{x\}$ be a path. Then

1. f_i is the top level function symbol of t_i for $1 \leq i \leq n$.
2. t_{i+1} is an immediate subterm of t_i for $1 \leq i \leq n-1$, and
3. if P ends in x then x is an immediate subterm of t_n.

A path $P = <f_1, t_1> \ldots <f_n, t_n>\{x\}$ is a *full path* in a term t if and only if $t_1 = t$ and either (i) t_n is a constant, or (ii) P ends in a variable x. (See Figure 1.)
Full paths can be formally defined as follows:

(a) If $t = x$, a variable, then x is the only full path in t.
(b) If $t = b$, a constant, then $<b, b>$ is the only full path in t.
(c) If $t = f(t_1, \cdots, t_m)$ then a full path in t is $<f, t> . p$, where p is a full path in some t_i.

Examples:

(1) $t = f(x,y)$. $<f, f(x,y)>x$ and $<f, f(x,y)>y$ are the full paths.
(2) $t = f(g(x),h(b,y))$. The full paths are

i. $<f, f(g(x),h(b,y))><g, g(x)>x$,

ii. $<f, f(g(x),h(b,y))><h, h(b,y)><b,b>$,

iii. $<f, f(g(x),h(b,y))><h, h(b,y)>y$.

3.2 PATH COMPARISON

i. Paths ending in different variables are incomparable.

ii. A variable is incomparable with any two-tuple. (This ensures that a path ending in a constant is never greater than a path ending in a variable.)

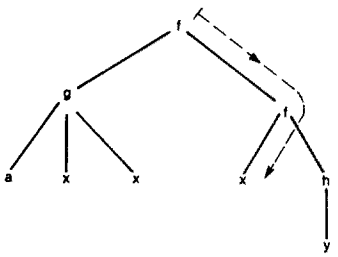
iii. To compare paths ending with the same variable we drop the variable from the end of both paths and compare the remaining sequences of two-tuples.

Let $P = <f_1, t_1><f_2, t_2> \ldots <f_m, t_m>\{x\}$ be a path. With every two-tuple $<f_i, t_i>$ we can associate a *left-context* (LC) and a *right-context* (RC) defined as:

$$RC(<f_i, t_i>, P) = \begin{cases} <f_{i+1}, t_{i+1}> \ldots <f_m, t_m>\{x\} & \text{where } i < m, \text{ and} \\ \{x\} & \text{if } i = m. \end{cases}$$

$$LC(<f_i, t_i>, P) = \begin{cases} <f_1, t_1> \ldots <f_{i-1}, t_{i-1}>, & \text{where } i > 1, \text{ and} \\ \lambda & \text{if } i = 1. \end{cases}$$

(See Figures 1 and 2. One can visualise the right-context of a tuple as *below* and the left-context as *above* the corresponding node in the tree representation of a term.)



The path shown here is

⟨f, f(g(a,x,x), f(x,h(y))⟩ ⟨f, f(x,h(y))⟩ x

Figure 1.

P = _____ <f, t> _____

<- - - LC - - -> <- - - RC - - ->

Figure 2.

Let $P_1 = <f_1, t_1> \ldots <f_m, t_m>$ and $P_2 = <g_1, s_1> \ldots <g_n, s_n>$ be two sequences of two-tuples. $P_1 \underset{P}{\geq} P_2$ if and only if for all $<g_j, s_j>$ in P_2 there exists $<f_i, t_i>$ in P_1 such that

a. $f_i > g_j$ or

b. $f_i \sim g_j$ and

1. $RC(<f_i, t_i>, P_1) \underset{P}{\geq} RC(<g_j, s_j>, P_2)$ or

2. $RC(<f_i, t_i>, P_1) \sim RC(<g_j, s_j>, P_2)$ and $t_i \underset{T}{\geq} s_j$ or

3. $RC(<f_i, t_i>, P_1) \sim RC(<g_j, s_j>, P_2)$, $t_i \sim s_j$ and

$LC(<f_i, t_i>, P_1) \underset{P}{\geq} LC(<g_j, s_j>, P_2)$

where $P_1 \sim P_2$ (the paths are *equivalent*) if $m = n$ and $f_i \sim g_i$ and $t_i \sim s_i$ for all $1 \leq i \leq n$, and term comparison ($\underset{T}{>}$) is defined in the next section. (The subscript P in $\underset{P}{>}$ will be omitted whenever it is obvious from the context.)

We often express this as '$<f_i, t_i>$ *takes care of* $<g_j, s_j>$'. Clearly, if $<f_i, t_i>$ takes care of $<g_j, s_j>$, then $f_i \gtrsim g_j$. Checking whether a two-tuple in one path takes care of a two-tuple in another is done in the following sequence: First compare the two function symbols, then the right contexts, then the terms in the two-tuples and finally the left-contexts.

Lemma 1: Let P_1, P_2, P_3 be paths such that $P_3 = <f, t> P_1$ for some term t and $P_1 \gtrsim P_2$. Then $P_3 > P_2$.

Lemma 2: Let P_1, P_2, P_3 be paths such that $P_1 = <f, t> P_1'$, $P_3 = <g, s> P_2$, $P_1 > P_2$ and $f > g$. Then $P_1 > P_3$.

Sketch of the Proof: Assume the contrary. Let $Q_1 = <f, t> Q_1'$ and Q_2 constitute a shortest counterexample in terms of $|Q_1| + |Q_2|$. Let $Q_3 = <g, s> Q_2$ with $f > g$ such that $Q_1 \not> Q_3$. Then there exists a tuple $<h, u>$ in Q_3 which is not taken care of by any tuple in Q_1.

Clearly $<h, u> \neq <g, s>$ since $<f, t>$ takes care of $<g, s>$. Thus there exists j such that $<h, u> = <g_j, s_j>$ in Q_2. But since $Q_1 > Q_2$ there must exist a tuple $<f_i, t_i>$ in Q_1 such that either

(a) $f_i > g_j$, or

(b) $f_i \sim g_j$ and $\mathrm{RC}(<f_i, t_i>, Q_1) > \mathrm{RC}(<g_j, s_j>, Q_2)$, or

(c) $f_i \sim g_j$, $\mathrm{RC}(<f_i, t_i>, Q_1) \sim \mathrm{RC}(<g_j, s_j>, Q_2)$ and $t_i > s_j$, or

(d) $f_i \sim g_j$, $\mathrm{RC}(<f_i, t_i>, Q_1) \sim \mathrm{RC}(<g_j, s_j>, Q_2)$, $t_i \sim s_j$ and $\mathrm{LC}(<f_i, t_i>, Q_1) > \mathrm{LC}(<g_j, s_j>, Q_2)$.

It is easy to see that (d) is the only possibility. But then $\mathrm{LC}(<g_j, s_j>, Q_3) = <g, s> \mathrm{LC}(<g_j, s_j>, Q_2)$, and $|\mathrm{LC}(<f_i, t_i>, Q_1)| + |\mathrm{LC}(<g_j, s_j>, Q_2)| < |Q_1| + |Q_2|$ and the minimality of Q_1 and Q_2 leads us to a contradiction. □

Lemma 3: Let P_1, P_2, P_3, P_4 be paths such that $P_3 = <f, t> P_1$, $P_4 = <f, s> P_2$ for some s, t. Then $P_1 \gtrsim P_2$ and $t \underset{T}{>} s$ implies $P_3 > P_4$.

Lemma 4: Let P_1, P_2, Q_1 and Q_2 be paths such that $Q_1 = <f, t> P_1$, $Q_2 = <f', s> P_2$ and $f \sim f'$. Then $Q_1 > Q_2$ and $t \not\gtrsim s$ implies $P_1 > P_2$.

Lemma 5: Let P_1, P_2, P_3, P_4, P_5 be paths such that $P_4 = P_1 P_3$ and $P_5 = P_2 P_3$. Then $P_1 > P_2$ implies $P_4 > P_5$.
(Intuitively, attaching equivalent paths to the right does not alter the ordering relation.)

3.3 TERM COMPARISON

Let $M = \{ s_1, s_2, \ldots, s_m \}$ be a multiset of terms. By $MP(M)$ we denote the union of the multisets of all full paths in each s_i.

Example: Let $M = \{ f(a,b), f(a,b) \}$.
$MP(M) = \{ <f, f(a,b)><a, a>, <f, f(a,b)><b, b>,$
$<f, f(a,b)><a, a>, <f, f(a,b)><b, b> \}$.

We deliberately abuse the notation when M is a singleton set and write $MP(t)$ instead of $MP(\{t\})$. Note that $MP(s) \sim MP(t)$ if and only if $s \sim t$.

Definition: (i) If s is a non-variable term and t is a variable then $s \underset{T}{>} t$ if and only if s contains t. (ii) $s = f(s_1, \ldots, s_m)$ and $t = g(t_1, \ldots, t_n)$. Let $M_1 = \{ s_1, \ldots, s_m \}$, $M_2 = \{ t_1, \ldots, t_n \}$. Then $s \underset{T}{>} t$ if and only if

a. $f > g$ and $s \underset{T}{>} t_i$ for all i, $1 \leq i \leq n$, or

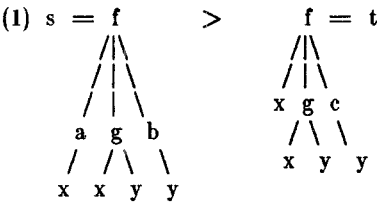
b. $f \sim g$ and $MP(M_1) \underset{P}{\gg} MP(M_2)$, or

c. $f \not\sim g$ and $MP(s) \underset{P}{\gg} MP(t)$.

Note that the path-comparisons that are done during term-comparison may necessitate further comparisons of terms. Let u_i and v_j be two terms that have to be compared while comparing s and t. It can be seen that either u_i must be a proper subterm of s or v_j must be a proper subterm of t. The only case where this may not be obvious is (ii)c, where $s = f(s_1, \ldots, s_m)$, $t = g(t_1, \ldots, t_n)$ and $f \not\sim g$. But, even though every full path of s starts with the tuple $<f, s>$ and every full path of t starts with the tuple $<g, t>$, we have no occasion to compare s and t again, since $<f, s>$ cannot ever take care of $<g, t>$. Thus the algorithm for term-comparison terminates, or, in other words, the ordering scheme is well-defined.

Examples:

(1)



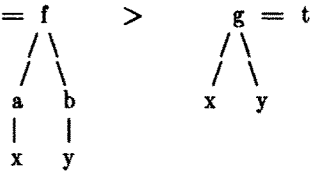
with precedence b > c.

Since the top-level function symbols are the same, we have to compare the multisets of all full paths in the immediate subterms, $MP(\{ a(x), g(x,y), b(y) \})$ and $MP(\{ x, g(x,y), c(y) \})$.

$MP(\{a(x), b(y)\}) = \{ <a, a(x)>x, <b, b(y)>y \}$. $MP(\{x, c(y)\}) = \{ x, <c, c(y)>y \}$.

Clearly $<a, a(x)>x \underset{P}{>} x$. $<b, b(y)>y \underset{P}{>} <c, c(y)>y$ since $<b, b(y)>$ takes care of $<c, c(y)>$.

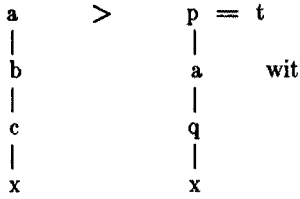
(2) s = f > g = t with a > g and b > g.



Since f and g are incomparable, we have to compare $MP(s)$ and $MP(t)$, where $MP(s) = \{<f,s><a,a(x)>x, <f,s><b,b(y)>y\}$ and $MP(t) = \{<g,t>x, <g,t>y\}$.

$<f,s><a,a(x)> \underset{P}{>} <g,t>$ since $<a,a(x)>$ takes care of $<g,t>$. Similarly $<f,s><b,b(y)> \underset{P}{>} <g,t>$. (Note also that s and t are incomparable under RPO.)

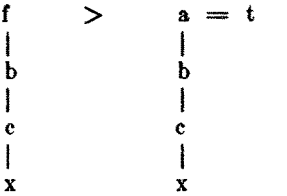
(3) s = a > p = t with a > p and b ~ q.



Since $a > p$, we have to compare $a(b(c(x)))$ and $a(q(x))$ and then again since the top-level symbols are both a's, we end up comparing $<b,b(c(x))><c,c(x)>$ and $<q,q(x)>$. It can be seen that $<b,b(c(x))>$ takes care of $<q,q(x)>$ since the former has a bigger right context.

It should be noted here that in the case when all function symbols are monadic, we can treat paths merely as *strings* formed by the function symbols. Consider the two-tuples $<f,s>$ in path P and $<g,t>$ in path Q. Now it is not hard to observe that if $f \sim g$ and the right-contexts are equivalent, then $s \sim t$ also. Hence it is quite unnecessary to keep the terms around.

(4) s = f > a = t with f > a.



We compare $f\ b\ c$ and $a\ b\ c$. a is taken care of by f. The b in $a\ b\ c$ is taken care of by the b in $f\ b\ c$ since the latter has a bigger left-context. Similarly, $f\ b > a\ b$ and thus the c in $f\ b\ c$ takes care of the c in $a\ b\ c$.

Lemma 6: $s \underset{T}{>} t$ if and only if $MP(s) \underset{P}{>>} MP(t)$.

$MP(s) \underset{P}{\gg} MP(t)$ implies that $MP(s) \cap MP(t) = \emptyset$.

Corollary 6.1: $s \underset{T}{>} t$ implies $f(\ldots s \ldots) \underset{T}{>} t$.

Theorem 1: (a) $\underset{T}{>}$ is a partial ordering on terms.

(b) $\underset{P}{>}$ is a partial ordering on paths.

4. PROPERTIES OF THE ORDERING

A partial ordering $>$ on terms is a ***simplification ordering*** if it satisfies the following properties (See [2]):

(1) $s > t$ implies $f(\ldots s \ldots) > f(\ldots t \ldots)$, (*replacement*)

(2) $f(\ldots t \ldots) > t$, (*subterm*)

for any terms $f(\ldots s \ldots)$ and $f(\ldots t \ldots)$.

Theorem 2: $\underset{T}{>}$ satisfies the subterm and replacement properties.

Proof: Follows from Lemma 6. □

Theorem 3: $\underset{T}{>}$ is closed under substitutions.

Basic Idea: A major share of the effort is in proving the following two propositions:

(1) For all substitutions σ, $s_1 \underset{T}{>} t_1$ implies $\sigma(s_1) \underset{T}{>} \sigma(t_1)$.

(2) Let $P_1 = <f_1, s_1> \ldots <f_m, s_m>$ and $P_2 = <g_1, t_1> \ldots <g_n, t_n>$ be two variable-free paths. Then for all substitutions σ,

$$\sigma(P_1) = <f_1, \sigma(s_1)> \ldots <f_m, \sigma(s_m)> \underset{P}{>} <g_1, \sigma(t_1)> \ldots <g_m, \sigma(t_n)> = \sigma(P_2).$$

We prove this by simultaneous induction on $|s_1| + |t_1|$. □

Theorem 4: $\underset{T}{>}$ is an extension of the recursive path ordering (RPO).

Sketch of the Proof: It can be easily observed that $s \underset{rpo}{\sim} t$ implies $s \underset{T}{\sim} t$. We prove that $s \underset{rpo}{>} t$ implies $s \underset{T}{>} t$ by induction on $|s| + |t|$.

The case when t is a variable and s is a term containing t is trivial.

Let $s = f(s_1, \ldots, s_m)$, $t = g(t_1, \ldots, t_n)$, $M_1 = \{ s_1, \ldots, s_m \}$ and $M_2 = \{ t_1, \ldots, t_n \}$. The following cases have to be considered:

1. $f > g$. Then, by the definition of RPO, $s \underset{rpo}{>} t_i$ for all i, $1 \leq i \leq n$. Since $|s| + |t_i| < |s| + |t|$ for every i, we get $s \underset{T}{>} t_i$ for $1 \leq i \leq n$. Thus $s \underset{T}{>} t$ by definition.

2. $f \sim g$. RPO requires that for every t_j in $M_2 - M_1$ there exist s_i in $M_1 - M_2$ such that $s_i \underset{rpo}{>} t_j$. Again, by induction hypothesis, $s_i \underset{T}{>} t_j$ implying $MP(s_i) \underset{P}{\gg} MP(t_j)$ by Lemma 6. Thus $MP(M_1 - M_2) \underset{P}{\gg} MP(M_2 - M_1)$ and the result follows.

3. $f \not\gtrsim g$. Therefore there exists i such that $s_i \underset{\widetilde{rpo}}{>} t$. If $s_i \underset{rpo}{\sim} t$ then $s_i \underset{T}{\sim} t$, $MP(s_i) \underset{P}{\sim} MP(t)$ and by Lemma 1, $MP(s) \underset{P}{\gg} MP(t)$. Hence the result.

If $s_i \underset{rpo}{>} t$, then by induction hypothesis $s_i \underset{T}{>} t$ and the rest of the proof can be carried out in a similar fashion. □

The path ordering also has the incrementality property desired of a termination ordering that is built on the fly to ensure the termination of rewriting systems generated by the Knuth-Bendix completion procedure in the process of computing a canonical system. The following theorem states that as the new precedence relations between function symbols are added, they do not upset the old ordering relation among terms; only terms which were not comparable earlier can be compared.

Theorem 5: Let $\overset{a}{>}$ and $\overset{b}{>}$ be two partial orderings of the set of function symbols F such that $\overset{b}{>}$ properly contains $\overset{a}{>}$. Then for all terms s, t in $T(F,V)$ $s \underset{T}{\overset{a}{>}} t$ implies $s \underset{T}{\overset{b}{>}} t$.

5. COMPLEXITY OF COMPARING TERMS USING PATH ORDERING

Given two terms s and t and a quasi-ordering $\underset{\sim}{>}$ of the set of function symbols F, it can be determined whether $s \underset{T}{>} t$ or $s \underset{T}{\sim} t$ in time $0(|s|^5 * |t|^5)$. We show this upper-bound by doing the comparison by a method similar to dynamic programming.

Assume that all subterms of s have already been compared to all proper subterms of t and all subterms of t have already been compared to all proper subterms of s. (In other words, we have done everything short of comparing s and t themselves.) Assume also that the results are stored in a 2-dimensional array A that can be accessed easily. For instance, if s_i is a subterm of s at position p and t_j is a subterm of t at position q ($p \neq \lambda$ or $q \neq \lambda$), then $A(p,q)$ tells you whether $s_i \underset{T}{>} t_j$ or $s_i \sim t_j$. Our aim is to determine $TCOMP(s,t)$, the additional time required to compare s and t.

Note that the worst case is the one in which *every* full path in s has to be compared with *every* full path in t. Hence we have to analyze the time complexity of path comparison. This seems at first to be difficult, since path comparison involves further term comparisons. But note that in those comparisons at least one of the terms must be a *proper* subterm of s or of t. Thus this involves only looking up in the array A. Let P and Q be full paths in s and t respectively. It can be shown, under the above assumption (namely that every subterm of s

has been compared with every subterm of t, with the exception of s and t themselves), that P and Q can be compared in time $O(|P|^3 * |Q|^3)$. We prove this later using, again, a dynamic programming-like method.

An upper bound for $TCOMP(s,t)$ can now be derived in terms of $|s|$ and $|t|$. The number of full paths in any term u has an upper bound of $|u|$. Hence the number of full-path-comparisons required to compare s and t has an upper bound of $|s| * |t|$. Similarly, the maximum length of a full path in a term u is also bounded above by $|u|$. Thus $TCOMP(s,t)$ takes $O(|s|^4 * |t|^4)$ time.

All that we now have to do is to sum up all possible $TCOMP(s_i, t_j)$s, where s_i is a subterm of s and t_j is a subterm of t. The number of all possible subterms of a term u is quite clearly $|u|$, since every position in the term corresponds to a subterm and vice versa. Each $TCOMP(s_i, t_j)$ is bounded above by $TCOMP(s,t)$ and therefore takes $O(|s|^4 * |t|^4)$ time. Thus

$$\sum TCOMP(s_i,t_j) = O(|s|^5 * |t|^5)$$

and the claim is proved.

It remains to be shown that two paths P and Q can be compared in time $O(|P|^3 * |Q|^3)$. Let $|P| = m$ and $|Q| = n$. For all i, j such that $1 \leq i \leq j \leq m$, let P_{ij} denote the subpath of P from the i-th tuple through the j-th. Similarly, let Q_{kl} denote the subpath of Q from the k-th tuple through the l-th.

Assume, as we did for comparing terms, that all proper subpaths of P have been compared with all proper subpaths of Q. Let B be a 4-dimensional array in which the results are stored: $B(i,j,k,l)$ gives you the result of comparing P_{ij} with Q_{kl}. Denote by $PCOMP(P,Q)$ the (additional) time required to compare P and Q under these assumptions.

For any tuple $<g, s_i>$ in P, we can find out whether there exists a tuple in Q that takes care of it in $O(n)$ time by a straightforward two-pass algorithm: in the first path, it is checked whether there is any tuple $<f, t_j>$ in Q such that $f > g$; if none, then in the second pass, we check whether $f \sim g$ in which case the arrays B and A are used to compare right-contexts, subterms and left-contexts. Thus $PCOMP(P,Q) = O(mn) = O(|P| * |Q|)$. Note that there are $O(m^2)$ subpaths of P and similarly $O(n^2)$ subpaths of Q. Therefore, $O(m^2 * n^2)$ path-comparisons are necessary in the worst case and each of these is bounded above by $PCOMP(P,Q)$. The overall time, therefore, is

$$O(m^3 * n^3) = O(|P|^3 * |Q|^3).$$

This completes the proof. □

6. INCORPORATING STATUS IN PATH ORDERING

While comparing terms we have to sometimes introduce the concept of *status* of an operator to make the terms comparable. The most common example is the associativity law of certain operators like + and *: $(x * y) * z = x * (y * z)$.

In order to orient such equations as rules we have to assign left-to-right (l-r) or right-to-left (r-l) status to one or more operators. (e.g. the * operator in the above equation.) Equivalent operators should have the same status. If the top-level operators of two terms are the same we compare the immediate subterms *not as multisets* but in a *lexicographical* way, either left-to-right or right-to-left.

Kamin and Levy [7] extend RPO to LRPO (lexicographic RPO) in the following way:

$s = f(s_1, \ldots, s_m) \underset{lrpo}{>} t = g(t_1, \ldots, t_n)$ if and only if one of the following three conditions hold:

(1) $f > g$ and $s \underset{lrpo}{>} t_i$ for all i, $1 \leq i \leq n$ (same as for RPO).

(2) $f \sim g$ and (a) if f and g have l-r status then there exists j such that $s_1 \sim t_1$, $s_{j-1} \sim t_{j-1}$, $s_j \underset{lrpo}{>} t_j$ and $s \underset{lrpo}{>} t_i$ for $j+1 \leq i \leq n$, (Similarly for r-l status.) whereas

(b) if f and g have no status then it must be that $\{ s_1, \ldots, s_m \} \underset{lrpo}{\gg} \{ t_1, \ldots, t_n \}$.

(3) $s_i \underset{lrpo}{\geq} t$ for some i, $1 \leq i \leq n$ (same as for RPO).

To incorporate status in the path ordering we have to modify path comparison and term comparison as follows:

Path Comparison (LP):

Let $P_1 = <f_1, t_1><f_2, t_2> \ldots <f_m, t_m>$ and

$P_2 = <g_1, s_1><g_2, s_2> \ldots <g_n, s_n>$ be two sequences of two-tuples. $P_1 \underset{LP}{>} P_2$ if and only if for all $<g_j, s_j>$ in P_2 there exists $<f_i, t_i>$ in P_1 such that

a. $f_i > g_j$ or

b. $f_i \sim g_j$ and if they have no status then

1. $\mathrm{RC}(<f_i, t_i>, P_1) \underset{LP}{>} \mathrm{RC}(<g_j, s_j>, P_2)$ or
2. $\mathrm{RC}(<f_i, t_i>, P_1) \sim \mathrm{RC}(<g_j, s_j>, P_2)$ and $t_i \underset{LT}{>} s_j$ or
3. $\mathrm{RC}(<f_i, t_i>, P_1) \sim \mathrm{RC}(<g_j, s_j>, P_2)$, $t_i \sim s_j$ and $\mathrm{LC}(<f_i, t_i>, P_1) \underset{LP}{>} \mathrm{LC}(<g_j, s_j>, P_2)$.

If they have l-r status, say, then

1. $t_i \underset{LT}{>} s_j$ or
2. $t_i \sim s_j$ and $\mathrm{LC}(<f_i, t_i>, P_1) \underset{LP}{>} \mathrm{LC}(<g_j, s_j>, P_2)$

Term Comparison (LT):

Let $s = f(s_1, \ldots, s_m)$, $t = g(t_1, \ldots, t_n)$, $M_1 = \{ s_1, \ldots, s_m \}$ and $M_2 = \{ t_1, \ldots, t_n \}$.

Then $s \underset{LT}{>} t$ if and only if one of the following three conditions hold:

a. $f > g$ and $s \underset{LT}{>} t_i$ for all i, $1 \leq i \leq n$.

b. $f \sim g$ and if f and g have l-r status, say, then there exists j such that $s_1 \sim t_1$, ... $s_{j-1} \sim t_{j-1}$, $s_j \underset{LT}{>} t_j$ and $s \underset{LT}{>} t_i$ for $j+1 \leq i \leq n$. (Similarly for r-l status.)

If f and g have no status, then $\mathrm{MP}(M_1 - M_2) \underset{LP}{\gg} \mathrm{MP}(M_2 - M_1)$.

c. $f \not> g$ and $MP(s) \underset{LP}{\gg} MP(t)$.

7. A FURTHER EXTENSION OF PATH ORDERING

A possible way of extending the path ordering even further is by allowing tuples from *different* paths to take care of tuples along a path. Thus we could remove the restriction that a path must be taken care of by another path and instead let tuples from a collection of paths take care of the tuples from another path. An example where such an idea would work is the following:

Example: $s = f(a(x), b(x))$, $t = g(h(x))$ with $a > g$ and $b > h$.
The tuple $<a, a(x)>$ takes care of $<g, g(h(x))>$ and $<b, b(x)>$ takes care of $<h, h(x)>$. Note that neither the path ordering nor RDO can compare s and t.

It must be noted of course that this new scheme is not clearly defined yet. But this looks like an interesting problem for future research and we are working on the details.

8. RELATIONSHIP TO RECURSIVE DECOMPOSITION ORDERING

Jean-Pierre Jouannaud [personal communication] has pointed out to us the similarity between our ordering scheme and the Recursive Decomposition Ordering Scheme (RDO) [6,14]. The way our ordering is formulated is similar to the "entire choice" scheme of RDO, since essentially *all* the paths in a term and *all* the tuples in a path are taken into account. (The reader is referred to [6,14] for the formal definition of RDO and further details.) However, the path ordering is *not* quite the same as RDO, since there are terms that the path ordering can compare but RDO cannot. An example is given below.

Example: $s = h(a(z), g(a(a(x)), x), g(b(b(y)), y))$ and
$t = h(a(z), g(a(x), b(y)), g(a(x), b(y)))$ with no precedence among the function symbols.

While comparing s with t, we eventually end up comparing the paths through the multisets $M_1 = \{ g(a(a(x)), x), g(b(b(y)), y) \}$ and $M_2 = \{ g(a(x), b(y)), g(a(x), b(y)) \}$. It can be easily seen that $\mathrm{MP}(M_1) \underset{P}{\gg} \mathrm{MP}(M_2)$.

When RDO compares terms s and t, it has to compare decompositions along the paths that end in z. (They are the 'leftmost' paths, expressed as '1.1' in positional notation.) Clearly the decomposition corresponding to h in s must take care of the decomposition corresponding to h in t. In other words, the decomposition

$$<h, a(z),\{ g(a(a(x)), x), g(b(b(y)), y) \}, \Box > = <h, a(z), M_1, \Box >$$

must be greater than

$$<h, a(z), \{ g(a(x), b(y)), g(a(x), b(y)) \}, \Box > = <h, a(z), M_2, \Box >.$$

But this is impossible since M_1 and M_2 are not comparable as multisets of *terms*.

Jouannaud has suggested a way to overcome this in RDO by introducing a varyadic function symbol, the details of which still need to be worked out. It will be interesting to examine whether the proposed extension of RDO is equivalent to the path ordering.

Jouannaud has also remarked that that the path ordering may not compare well with RDO in efficiency. There is no way to do this comparison at present; we are not aware of any complexity analysis of RDO; there do not exist implementations of path ordering and RDO on the same machine and in the same language either to compare their experimental performance.

9. CONCLUSION

We have found that recursive path ordering (RPO), though simple and elegant, cannot order certain equations which we intuitively 'feel' we can order. The example (given in Section 2.2) $a(b(x)) = c(d(x))$ with $b > c$ and $a > d$ is one such. We have devised a new ordering which compares terms using the paths in them. This ordering properly contains RPO and eliminates many of RPO's drawbacks. We also feel it is easy to understand.

ACKNOWLEDGEMENTS: We thank Nachum Dershowitz and Jean-Pierre Jouannaud for their comments on earlier drafts of this paper.

10. REFERENCES

[1] Buchberger, B., "A Theoretical Basis for the Reduction of Polynomials to Canonical Forms," *ACM-SIGSAM Bulletin*, 39, August 1976, pp. 19-29.

[2] Dershowitz, N., "Orderings for Term Rewriting Systems," *Theoretical Computer Science* 17 (1982) 279-301.

[3] Guttag, J.V., Kapur, D., and Musser, D.R., "On Proving Uniform Termination and Restricted Termination of Rewriting Systems," *SIAM Journal on Computing*, Vol. 12, No. 1, February, 1983, pp. 189-214.

[4] Huet, G., "Confluent Reductions: Abstract Properties and Applications to Term Rewriting Systems," *JACM*, Vol. 27, No. 4, Oct., 1980, pp. 797-821.

[5] Huet, G., and Lankford, D.S., "On the Uniform Halting Problem for Term Rewriting Systems," Rapport Laboria 283, INRIA, Paris, March 1978.

[6] Jouannaud, J.-P., Lescanne, P., Reinig, F., "Recursive Decomposition Ordering," *Conf. on Formal Description of Programming Concepts*, Garmisch, 1982.

[7] Kamin, S., and Levy, J-J., "Attempts for Generalizing the Recursive Path Ordering," Unpublished Manuscript, Feb. 1980.

[8] Kandri-Rody, A. and Kapur, D., "Algorithms for Computing the Grobner Bases of Polynomial Ideals over Various Euclidean Rings," Proceedings of *EUROSAM* '84, Cambridge, England, July, 1984, Springer Verlag Lecture Notes in Computer Science LNCS 174, (ed. J. Fitch), pp. 195-206.

[9] Kapur, D., and Narendran, P., "The Knuth-Bendix Completion Procedure and Thue Systems," *Third Conference on Foundation of Computer Science and Software Engg.*, Bangalore, India, December 1983, pp. 363-385.

[10] Kapur, D., and Sivakumar, G., "Architecture of and Experiments with RRL, a Rewrite Rule Laboratory," *Proceedings of the NSF Workshop on Rewrite Rule Laboratory*, Rensselaerville, NY, September 4-6, 1983.

[11] Knuth, D.E. and Bendix, P.B., "Simple Word Problems in Universal Algebras," in *Computational Problems in Abstract Algebras* (ed. J. Leech), Pergamon Press, 1970, pp. 263-297.

[12] Lankford, D.S., "On Proving Term Rewriting Systems are Noetherian," Memo MTP-3, Mathematics Department, Louisiana Technical University, Ruston, LA (1979).

[13] Lankford, D.S., and Ballantyne A.M., "Decision Procedures for Simple Equational Theories with Commutative-Associative Axioms: Complete Sets of Commutative-Associative Reductions," Memo ATP-39, Dept. of Mathematics and Computer Sciences, Univ. of Texas, Austin, TX (1979).

[14] Lescanne, P., "How to Prove Termination? An Approach to the Implementation of a New Recursive Decomposition Ordering," Proceedings of an *NSF Workshop on the Rewrite Rule Laboratory* Sept. 6-9 Sept. 1983, (eds. Guttag, Kapur, Musser), General Electric Research and Development Center Report 84GEN008, April, 1984, pp. 109-121.

[15] Peterson, G.E., and Stickel, M.E., "Complete Sets of Reductions for Some Equational Theories," *JACM*, Vol. 28, No. 2, pp. 233-264.

[16] Plaisted, D.A., "A Recursively Defined Ordering for Proving Termination of Term Rewriting Systems," Report R78-943, Dept. of Computer Science, Univ. of Illinois, Urbana, IL.