The theory of total unary rpo is decidable

Paliath Narendran[1] and Michael Rusinowitch[2]

[1] Institute of Programming and Logics, Department of Computer Science,
State University of New York at Albany, Albany, NY 12222, USA
dran@cs.albany.edu

[2] LORIA-INRIA Lorraine, 615, rue du jardin botanique, BP 101,
54602 Villers les Nancy cedex, France
rusi@loria.fr

The Recursive Path Ordering (*rpo*) is a syntactic ordering on terms that has been widely used for proving termination of term-rewriting systems [7, 20]. How to combine term-rewriting with ordered resolution and paramodulation is now well-understood and it has been successfully applied in many theorem-proving systems [11, 16, 21]. In this setting an ordering such as rpo is used both to orient rewrite rules and to select maximal literals to perform inferences on. In order to further prune the search space the ordering requirements on conditional inferences are better handled when they are treated as constraints [12, 18]. Typically a non-orientable equation $s = t$ will be split as two constrained rewrite rules: $s \rightarrow t \mid\ s > t$ and $t \rightarrow s \mid\ t > s$. Such constrained rules are useless when the constraint is unsatisfiable. Therefore it is important for the efficiency of automated reasoning systems to investigate decision procedures for the theory of terms with ordering predicates.

Other types of constraints can be introduced too such as disunification constraints [1]. It is often the case that they can be expressed with ordering constraints (although this might be inefficient).

We prove that the first-order theory of the recursive path ordering is decidable in the case of unary signatures with total precedence. This solves a problem that was mentioned as open in [6]. The result has to be contrasted with the undecidability results of the lexicographic path ordering [6] for the case of symbols with arity ≥ 2 and total precedence and for the case of unary signatures with partial precedence. We recall that lexicographic path ordering (lpo) and the recursive path ordering and many other orderings such as [13, 10] coincide in the unary case.

Among the positive results it is known that the existential theory of total lpo is decidable [3, 17]. The same result holds for the case of total rpo [8, 15].

The proof technique we use for our decidability result might be interesting by itself. It relies on encoding of words as trees and then on building a tree automaton to recognize the rpo relation.

Key words: Recursive path ordering, first-order theory, ground reducibility, tree automata, ordered rewriting.

1 The recursive path ordering on words

We assume that A is a finite alphabet, A^* the set of words on A and ϵ is the empty word. We shall often identify a letter with the corresponding word of length one. There is a correspondance between words and terms on a unary alphabet: every letter can be considered as a unary function and every word $a_1, a_2, \ldots, a_n$ can be considered as a term $a_1(a_2(\ldots a_n(x)\ldots))$ where x is an element of the set of variables X.

The *Recursive Path Ordering* (*rpo*) originally introduced by Dershowitz [7] is defined as follows on words:

Given a finite total precedence $\succ^A$ on A,

$$s \succ^A_{rpo} t$$

if and only if one of the following holds:

1. $s \neq \epsilon$ and $t = \epsilon$
2. $s = as'$ and $t = bt'$ and either
   (a) $a \succ^A b$ and $s \succ_{rpo}^A t'$ or
   (b) $a = b$ and $s' \succ_{rpo}^A t'$ or
   (c) $b \succ^A a$ and $s' \succeq_{rpo}^A t$

where $s' \succeq_{rpo}^A t$ is an abbreviation for $s' \succ_{rpo}^A t$ or $s' = t$.

The following properties of rpo are well-known and their proofs can be found in the literature (e.g. [7]). We omit the superscript A when it is clear from context.

Proposition 1. *The relation $\succ_{rpo}$ is antireflexive, transitive, monotonic (i.e. $aw \succ_{rpo} aw'$ if $w \succ_{rpo} w'$), total, well-founded and has the subterm property (i.e. $w \succ_{rpo} w'$ when w' is a proper subterm of w).*

When the precedence is not total, all properties but totality of $\succ_{rpo}$ remain valid.

We now give a few properties that will be useful in the sequel.

Lemma 1. *If $s \succ_{rpo} t$ then for all $w \in A^*$, $ws \succ_{rpo} wt$.*

Proof: by induction on the length of w and by the monotonicity property. □

Lemma 2. *If $a \in A$ and $w \succ_{rpo} w'$ then for all $w'' \in A^*$ such that $a \succ_{rpo} w''$, we have $aw \succ_{rpo} w''aw'$.*

Proof: by induction on the length of w''. If w'' is empty it is obvious. If $w'' = bu$ where $b \in A$, then by definition of rpo we must have $a \succ^A b$ and $a \succ_{rpo} u$. Since u is shorter than w'', by induction hypothesis we have $aw \succ_{rpo} uaw'$. By the definition of rpo, we also have $aw \succ_{rpo} buaw'$ and the result follows. □

The next lemma shows that when terms s, t have the same number of maximal symbols a then the comparison can be done by considering only the rightmost (innermost) occurrence of a whose arguments are different in s and t.

Lemma 3. *Let $w_1, w_2, \ldots, w_k$, $u_1, u_2, \ldots, u_k$ be two sequences of words such that each letter in each word is strictly smaller than $a \in A$. If $w \succ_{rpo} w'$ then we have:*

$$aw_1aw_2 \ldots aw_kaw \succ_{rpo} au_1au_2 \ldots au_kaw'$$

Proof: By induction hypothesis we can assume that:

$$aw_2 \ldots aw_kaw \succ_{rpo} au_2 \ldots au_kaw'$$

by the previous lemma we have:

$$aw_2 \ldots aw_kaw \succ_{rpo} u_1au_2 \ldots au_kaw'$$

by the subterm property and transitivity we have:

$$w_1aw_2 \ldots aw_kaw \succ_{rpo} u_1au_2 \ldots au_kaw'$$

The result follows by monotonicity. □

We denote by $max(w, A)$ the maximal letter of A that occurs in word w and by $mul(w, A)$ the number of occurrences of this letter in w. As a consequence of the previous lemmas we get:

Lemma 4. *$w \succ_{rpo} w'$ iff one of the following holds:*

1. $max(w, A) \succ max(w', A)$

2. $max(w, A) = max(w', A)$ *and* $mul(w, A) > mul(w', A)$
3. $a = max(w, A) = max(w', A)$, $mul(w, A) = mul(w', A)$, $w = w_0 a w_1 a w_2 \ldots a w_k$, $w' = u_0 a u_1 a u_2 \ldots a u_k$ *and there exists* $0 \leq i \leq k$ *such that* $w_i \succ_{rpo} u_i$ *and for all* $j > i$ *we have* $w_j = u_j$.

A first idea would be to try to use word automata in order to recognize the relation $\succ_{rpo}$. However this is not possible. Consider $A = \{a, b\}$ with $a \succ b$. We introduce the product alphabet $A^2 = \{(u, v) \mid u, v \in A \cup \{\perp\}\}$, where $\perp$ is a new symbol. Classically we can associate to every couple of words $U = u_1.u_2 \ldots u_m \in A^*$ and $V = v_1.v_2 \ldots v_n \in A^*$ a word $U \otimes V = (u_1, v_1), (u_2, v_2), \ldots$ on A^2 by completing the shortest word (if any) by some $\perp$'s in order to get two words of the same length. Assume that the relation $R = \{U \otimes V \mid U, V \in A^* \text{ and } U \succ V\}$ is recognizable on A^2 by an automaton $\mathcal{A}$ with N states. Then $b^N a^{N+1} \otimes a^N b^{N+1}$ belongs to R. By a pigeon-hole argument the automaton will enter twice the same state when reading the second half of the word. Then by pumping between the corresponding positions, $\mathcal{A}$ should accept also $b^N a^m \otimes a^N b^m$, with $m \leq N$. Since $a^N b^m \succ b^N a^m$, this raises a contradiction.

2 Coding words as trees

We shall now define a tree representation of words so that comparison of words can be performed by an automaton. Our goal is to represent a word $w = w_1 a w_2 \ldots a w_k a w_{k+1}$ by a binary tree $a(t_{k+1}, a(t_k, \ldots, a(t_1, \epsilon) \ldots))$, where t_i represents w_i and $a \in A$ is maximal in w w.r.t. $\succ^A$. First we introduce a new signature F that contains a binary symbol for each element of the alphabet A. The binary symbol associated with $a \in A$ will be denoted also a, by abuse of notation. We shall introduce also a constant symbol ϵ_a for each $a \in A$. We shall denote by a' the successor of a with respect to the total precedence $\succ^A$ of A. We denote by o (resp. m) the minimal (resp. maximal) element of A.

The translation function τ from A^* to $T(F)$ is defined using auxiliary functions τ_a, $a \in A$.

$$\begin{aligned}
\tau_{a'}(w_1 a' w_2) &= a'(\tau_a(w_2), \tau_{a'}(w_1)) && \text{when } a' \notin w_2 \\
\tau_{a'}(w) &= a'(\tau_a(w), \epsilon_{a'}) && \text{when } a' \notin w \\
\tau_o(w.o) &= o(\epsilon_o, \tau_o(w)) \\
\tau_o(\epsilon) &= o(\epsilon_o, \epsilon_o)
\end{aligned}$$

Now we define $\tau(w) = \tau_m(w)$.

Example 1. Assume $A = \{a, b, c\}$ with $a \succ^A b \succ^A c$. Then $\tau(\epsilon) = a(b(c(\epsilon_c, \epsilon_c), \epsilon_b), \epsilon_a)$.

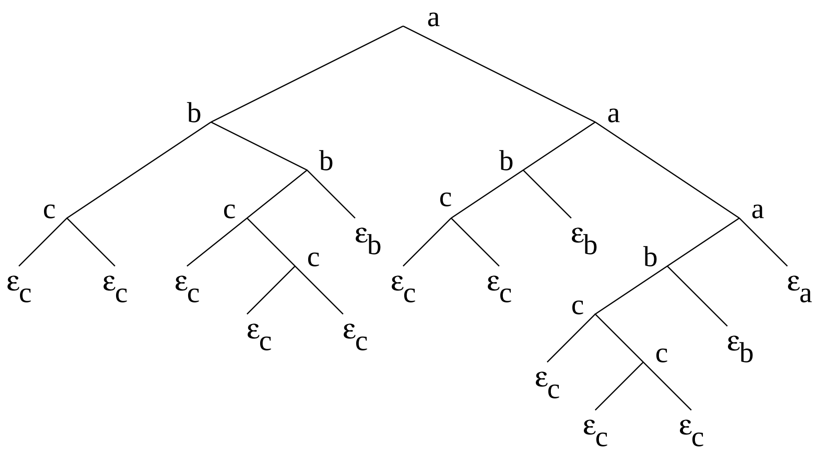
Example 2. Assume $A = \{a, b, c\}$ with $a \succ^A b \succ^A c$. Then $\tau(caacb) =$

$$a(b(c(\epsilon_c, \epsilon_c), b(c(\epsilon_c, c(\epsilon_c, \epsilon_c), \epsilon_b))), a(b(c(\epsilon_c, \epsilon_c), \epsilon_b), a(b(c(\epsilon_c, c(\epsilon_c, \epsilon_c)), \epsilon_b), \epsilon_a)))$$

Note that τ is a real encoding. In other terms, two different words are never coded as the same tree.

Lemma 5. *The function τ is injective.*

Proof: Let $I_a = \{c \preceq^A a\}$. We prove by induction on a that τ_a is injective on I_a^*. Since $I_m = A$ this will imply the result. If $a = o$ then $\tau_o(o^k) = \tau_o(o^j)$ clearly entails $j = k$. Now we assume by induction hypothesis that I_a is injective. Consider two words $w_1, w_2 \in I_{a'}$ such that $\tau_{a'}(w_1) = \tau_{a'}(w_2)$. If neither of w_1, w_2 contains a' we have $a'(\tau_a(w_1), \epsilon_{a'}) = a'(\tau_a(w_2), \epsilon_{a'})$ and therefore $\tau_a(w_1) = \tau_a(w_2)$ which implies $w_1 = w_2$ by the induction hypothesis. The case when $w_1 = u_1 a' v_1$ (for some words $u_1 \in I_{a'}, v_1 \in I_a$) and a' does not occur in w_2 is impossible since $\tau_{a'}(w_1)$ should contain in that case more a' that $\tau_{a'}(w_2)$. Now assume that $w_1 = u_1 a' v_1$ and $w_2 = u_2 a' v_2$ with $u_1, u_2 \in I_{a'}, v_1, v_2 \in I_a$. We prove by induction on the sum of the lengths of w_1 and w_2 that $\tau_{a'}(w_1) = \tau_{a'}(w_2)$ implies $w_1 = w_2$. The base case is trivial. $a'(\tau_a(v_1), \tau_{a'}(u_1)) = a'(\tau_a(v_2), \tau_{a'}(u_2))$



which implies by decomposition $\tau_a(v_1) = \tau_a(v_2)$ and $\tau_{a'}(u_1) = \tau_{a'}(u_2)$. From the induction hypothesis we have $v_1 = v_2$, $u_1 = u_2$ and therefore $w_1 = w_2$. □

A *finite tree automaton* over a signature $\mathcal{F}$ is a tuple $\mathcal{A} = (Q, Q_f, \Delta)$ where Q is a finite set of states, $Q_f \subseteq Q$ is a subset of accepting states and Δ is a set of transition rules of type:

$$f(q_1, \ldots, q_n) \rightarrow q$$

where $n \geq 0$, f is symbol of $\mathcal{F}$ of arity n and $q, q_1, \ldots, q_n \in Q$. We consider here bottom-up tree automata: they are applied to ground terms inductively from the leaves to the root. A ground term t is accepted by $\mathcal{A}$ if $t \rightarrow^*_\Delta q$ for some state $q \in Q_f$.

For more details about tree automata a recent reference is [4].

Lemma 6. *The set of trees $\{\tau(w) |\ w \in A^*\}$ is recognizable by a tree automaton $\mathcal{C}$.*

Proof: The following automaton does the work. Let $Q = \{q_a |\ q \in A\} \cup \{qs_o\}$, $Q_f = \{q_m\}$ where m is the maximal element in the precedence. The transitions are, for all $a \in A$:

$$\begin{aligned} \epsilon_a &\rightarrow q_a, \quad a \neq o \\ \epsilon_o &\rightarrow qs_o \\ o(qs_o, qs_o) &\rightarrow q_o \\ o(qs_o, q_o) &\rightarrow q_o \\ a(q_a, q_{a'}) &\rightarrow q_{a'} \end{aligned}$$

□

3 Tree automata

We define now a tree automaton for comparing two words. First let us extend F with a constant $\perp$. By abuse of notation we denote by fg a new function symbol associated with the couple $(f, g) \in (F \cup \{\perp\}) \times (F \cup \{\perp\})$. We denote by F^2 the signature $\{fg \,|\, f, g \in F \cup \{\perp\}\}$ (product alphabet), where the arity of fg is equal to the maximum of the arities of f and g ($\perp$ has arity 0). The automaton will traverse in a bottom-up way the tree obtained by gluing together the tree structures associated to the two words.

The coding of a couple of trees $(t_1, t_2) \in T(F)^2$ as a tree $t_1 \otimes t_2$ on the product alphabet F^2 is defined recursively as follows:

$$
\begin{aligned}
f(s_1, s_2) \otimes g(r_1, r_2) &= fg(s_1 \otimes r_1, s_2 \otimes r_2) \\
f(s_1, s_2) \otimes \epsilon_a &= f\epsilon_a(s_1 \otimes \bot, s_2 \otimes \bot) \\
\epsilon_a \otimes f(s_1, s_2) &= \epsilon_a f(\bot \otimes s_1, \bot \otimes s_2) \\
f(s_1, s_2) \otimes \bot &= f\bot(s_1 \otimes \bot, s_2 \otimes \bot) \\
\bot \otimes f(s_1, s_2) &= \bot f(\bot \otimes s_1, \bot \otimes s_2)
\end{aligned}
$$

Lemma 7. *The set of trees* $\{\tau(w) \otimes \tau(v) |\ w, v \in A^*\}$ *is recognizable by a tree automaton.*

Proof: We just need to take the product of two copies of the automaton in Lemma 6. □

Theorem 1. *The set of trees* $\{\tau(w) \otimes \tau(v) |\ \ w \succ_{rpo} v\}$ *is recognizable by a tree automaton.*

Proof: let $Q = \{q^a_+, q^a_-, q^a_>, q^a_<, q^a_= \,|\, a \in A\}$ be the set of states. The set of accepting states is $Q_f = \{q^m_+, q^m_>\}$ where m is the maximum element of the alphabet. The meaning of q^a_+ (resp. q^a_-) is that on the right branch we have encountered more (resp. less) a's in $\tau(w)$ than in $\tau(v)$. The meaning of $q^a_>$ (resp. $q^a_<$, resp. $q^a_=$) is that the number of a's is the same but some a-free subword tips the balance in favour of w (resp. v, resp. equality).

The transitions of the bottom-up tree automaton include, for all $a, b \in A$ with $a = b'$:

$aa(\_, q^a_+) \rightarrow q^a_+$	$aa(q^b_>, q^a_=) \rightarrow q^a_>$
$aa(\_, q^a_-) \rightarrow q^a_-$	$aa(q^b_>, q^a_<) \rightarrow q^a_>$
$a\epsilon_a(\_, \_) \rightarrow q^a_+$	$aa(q^b_>, q^a_>) \rightarrow q^a_>$
$\epsilon_a a(\_, \_) \rightarrow q^a_-$	$aa(q^b_+, q^a_=) \rightarrow q^a_>$
$\bot a(\_, \_) \rightarrow q^a_-$	$aa(q^b_+, q^a_<) \rightarrow q^a_>$
$a\bot(\_, \_) \rightarrow q^a_+$	$aa(q^b_+, q^a_>) \rightarrow q^a_>$
$\epsilon_a\epsilon_a \rightarrow q^a_=$	$aa(q^b_-, q^a_=) \rightarrow q^a_<$
$\bot\epsilon_a \rightarrow q^a_-$	$aa(q^b_-, q^a_<) \rightarrow q^a_<$
$\epsilon_a\bot \rightarrow q^a_+$	$aa(q^b_-, q^a_>) \rightarrow q^a_<$
$aa(q^b_=, q^a_=) \rightarrow q^a_=$	$aa(q^b_<, q^a_=) \rightarrow q^a_<$
$aa(q^b_=, q^a_>) \rightarrow q^a_>$	$aa(q^b_<, q^a_<) \rightarrow q^a_<$
$aa(q^b_=, q^a_<) \rightarrow q^a_<$	$aa(q^b_<, q^a_>) \rightarrow q^a_<$

In order to complete the automaton we may add a failure state and transitions for detecting trees that do not stand for a couple of words. We call $\mathcal{A}$ the automaton we get finally.

We now show that if $w \succ_{rpo} u$ then $\tau(w) \otimes \tau(u)$ is accepted by $\mathcal{A}$. For this we show the more general result:

Claim: *if a is the maximal letter in w then* $\tau_a(w) \otimes \tau_a(u) \rightarrow_\Delta q \in \{q^a_+, q^a_>\}$.

We proceed by induction on a (w.r.t. $\succ^A$). The base case when $w \in \{o\}^*$ is left to the reader. Assume that the result is true for all $c \prec a$. Consider now the words w, u and $a \in A$ such that $a = max(w, A)$ and $w \succ_{rpo} u$.

Let t be $\tau_a(w) \otimes \tau_a(u)$. By Lemma 4 there are three ways to get $w \succ_{rpo} u$:

If $a = max(w, A) \succ d = max(u, A)$ then $\tau_a(t) = aa(t_1, a\epsilon_a(t_2, t_3))$ for some terms t_1, t_2, t_3. It can be checked that $t \rightarrow_\Delta q^a_+$.

The case when $a = max(w, A) = max(u, A)$ and $mul(w, A) > mul(u, A)$ is similar.

Assume now that $w = w_0aw_1aw_2 \ldots aw_kaw_k$, $u = u_0au_1au_2 \ldots au_kau_k$ and a is larger than every symbol in w_i, u_j where $i, j = 0, \ldots, k$. Let b be the predecessor of a: $b' = a$. By Lemma 4, $w \succ_{rpo} u$ iff there exists j such that $w_j \succ_{rpo} u_j$ and $l > j$ implies $w_l = u_l$. Let us denote $\tau_b(w_j) \otimes \tau_b(u_j)$ by s_j.

Then $t = aa(s_k, aa(s_k, aa(\dots, aa(s_j, \dots, aa(s_0, \epsilon_a)\dots))$. Running the automaton on t one can get, after some steps, $aa(s_k, \dots aa(q', q)\dots)$ for some $q \in \{q^a_=, q^a_<, q^a_>\}$. By the induction hypothesis (since $b \prec a$) $s_j \rightarrow^*_\Delta q'$. where q' is either q^b_+ or $q^b_>$. Hence applying one more rule of the automaton we also have: $aa(q', q) \rightarrow_\Delta q^a_>$. Since s_l codes a couple of identical words, we have $s_l \rightarrow^*_\Delta q^b_=$, for $l > j$. Applying several times the rule $aa(q^b_=, q^a_>) \rightarrow q^a_>$ allows one to conclude. By symmetry we can show that $\tau(w) \otimes \tau(u) \rightarrow_\Delta q \in \{q^m_-, q^m_<\}$ whenever $w \prec_{rpo} u$.

For the converse one needs to show that whenever $\tau(w) \otimes \tau(u) \rightarrow_\Delta q \in Q_f$ we have $w \succ_{rpo} u$. Since the ordering $\succ_{rpo}$ is total either $w \prec_{rpo} u$ or $w \succ_{rpo} u$. In the first case $q \in \{q^m_-, q^m_<\}$ and in the second, $q \in \{q^m_+, q^m_>\} = Q_f$. Now since the automaton $\mathcal{A}$ is deterministic if $q \in Q_f$ then necessarily $w \succ_{rpo} u$. □

The first-order theory we will show decidable has both inequality interpreted as $\succ_{rpo}$ and equality interpreted as identity. When $a \in A$ we shall reduce the satisfiability of a formula $au \succ_{rpo} v$ (resp. $u \succ_{rpo} av$) to the satisfiability of $w \succ_{rpo} v \wedge w = au$ (resp.$u \succ_{rpo} w \wedge w = au$). This motivates the next theorem.

Note that the set $\{av \otimes v | v \in A^*\}$ is recognizable by a word automaton. However we have now adopted a tree representation for the words to handle inequalities and we cannot mix different representations of the same word. Hence we have to prove:

Theorem 2. *The set of trees* $\{\tau(av) \otimes \tau(v) \mid v \in A^*\}$ *(where* $a \in A$*) is recognizable by a tree automaton.*

Proof: Note that adding a letter a to a word w amounts to replacing a leaf by a node of the tree representation of w. There is a unique position where this node can be inserted: it is at the position of the rightmost occurrence of ϵ_a. Let $Q = \{q_0, q_1, q_2\}$ be the set of states. The state q_1 recognizes the trees on F^2 obtained as the product of two identical trees. The accepting state is q_0. The transitions of the bottom-up tree automaton include, for all $b \in A$:

$$\boxed{\begin{array}{r} \epsilon_a \bot \rightarrow q_0 \\ \epsilon_b \epsilon_b \rightarrow q_1 \\ bb(q_1, q_1) \rightarrow q_1 \\ a\epsilon_a(q_1, q_0) \rightarrow q_0 \\ bb(q_1, q_0) \rightarrow q_0 \end{array}}$$

□

4 RPO theory

An *RPO formula* is a first-order formula constructed from terms on the unary signature A and the binary predicate symbols "$\succ$" and "$=$". The formula is interpreted in $T(A)$, with $\succ$ as $\succ_{rpo}$ and $=$ as identity. We denote by $\phi(x_1, \dots, x_n)$ an RPO formula with free variables $x_1, \dots, x_n$.

A *flat formula* is an RPO formula whose atoms are of type: $x = y, x = ay, x = a, x \succ y, a \succ x$ or $x \succ a$, where x, y are variables. Hence terms in flat formulas are restricted to types x, ax, a, where $a \in A, x \in X$.

Using transformations that preserve solutions we can reduce the decision of RPO formulas to the decision of flat formulas. These transformations include the following abstraction rules, where $a, b \in A$:

$$au \succ_{rpo} v \quad \vdash \quad \exists u'(u' \succ_{rpo} v \wedge u' = au) \tag{1}$$
$$u \succ_{rpo} av \quad \vdash \quad \exists v'(u \succ_{rpo} v' \wedge v' = av) \tag{2}$$
$$abu = v \quad \vdash \quad \exists u'(au' = v \wedge u' = bu) \tag{3}$$
$$u = abv \quad \vdash \quad \exists v'(av' = v \wedge v' = bv) \tag{4}$$

These rules are completed by the decomposition rules derived from the definition of $\succ_{rpo}$, and the decomposition and clash rules for the predicate $=$ in $T(A)$.

Theorem 3. *Given an RPO formula $\phi(x_1, \ldots, x_n)$ there exists an automaton that recognizes*

$$\{u_1 \otimes \ldots \otimes u_n \mid \ u_1, \ldots, u_n \text{ is a solution of } \phi\}.$$

Proof: We can assume the formula ϕ is flat. The proof will be by induction on the structure of ϕ. The technique is classical ([4]). Let $\mathcal{U}$ be an automaton that recognizes all terms. We first remark that given two automata $\mathcal{A}_1, \mathcal{A}_2$ for the solutions of $\phi_1(\overline{xz})$ and $\phi_2(\overline{zy})$, where $\overline{x}, \overline{y}, \overline{z}$ are disjoint sets of variables, the set of solutions of $\phi_1(\overline{xz}) \wedge \phi_2(\overline{zy})$ is the intersection of the regular languages recognized by $\mathcal{A}_1 \otimes (\otimes_{|y|}\mathcal{U})$ and $(\otimes_{|x|}\mathcal{U}) \otimes \mathcal{A}_2$.

Base case: assume that $\phi(x_1, \ldots, x_n)$ is atomic. Its satisfiability is equivalent to the conjunction of flat atomic formulas obtained by replacing (repeatedly) every strict maximal subterm t in every atom by a fresh variable x_t. Automata recognizing the solutions of formula of type $x \succ y$ (resp. $x = ay$) can be constructed thanks to Theorem 1 (resp. 2). For the other cases ($x = y, x = a, a \succ x, x \succ a$) automata are easily obtained too.

Step case: i) Suppose $\phi(x_1, \ldots, x_n) = \neg\psi(x_1, \ldots, x_n)$. By taking the intersection of the complement automaton for $\psi(x_1, \ldots, x_n)$ with $\otimes_n \ \mathcal{C}$ (product of n copies of $\mathcal{C}$) we get an automaton for ϕ. ii) If $\phi(x_1, \ldots, x_n) = \exists x_1 \ \psi(x_1, \ldots, x_n)$. then by projection (i.e. forgetting first component) one get an automaton for ϕ. □

We can now conclude with the main result which is a direct consequence of the previous theorem:

Theorem 4. *The first-order theory of rpo is decidable when the signature is built from a finite set of constants and unary symbols.*

5 On normal forms and ordered rewriting

5.1 The recursive path ordering on terms

We assume that F is a finite set of function symbols given with their arity. $T(F, V)$ is the set of finite terms built on F and an alphabet V of (first-order) *variable symbols.* $\equiv$ denotes syntactic equality of terms. $T(F)$ is the set of terms which do not contain any variables. A multiset over a set X is a function M from X to the natural numbers. Any ordering $>$ on X can be extended to an ordering $\gg$ on finite multisets over X as follows: $M \gg N$ if a) $M \neq N$ and b) whenever $N(x) > M(x)$ then $M(y) > N(y)$ for some $y > x$. Note that if $>$ is well-founded so is $\gg$.

Given a finite total precedence $\succ^F$ on functions,

$$s = f(s_1, \ldots, s_m) \succ^F_{rpo} g(t_1, \ldots, t_n) = t$$

if and only if one of the following holds:

1. $f \succ^F g$ and $s \succ^F_{rpo} t_i$ for all $1 \leq i \leq n$
2. $f = g$, $m = n$ and $\{s_1, \ldots, s_m\} \ggcurly^F_{rpo} \{t_1, \ldots, t_m\}$
3. There exists a j, $1 \leq j \leq m$, such that $s_j \succ^F_{rpo} t$ or $s_j \sim_{rpo} t$.

where $\sim_{rpo}$ is defined as *permutatively equal,* or, in other words, the terms are treated as *unordered* trees. The set $t/\!\sim_{rpo}$ is the equivalence class of t for $\sim_{rpo}$. The multiset extension of $\succ_{rpo}$ used above, namely $\ggcurly_{rpo}$, is defined in terms of this equivalence.

5.2 Ordered rewriting

An *ordered* term rewrite system (TRS) is a pair $(E, \succ)$, where E is a set of equations, and $\succ$ is an ordering on terms. The *ordered rewriting relation* defined by $(E, \succ)$ is the smallest monotonic binary relation $\rightarrow_{E,\succ}$ on terms such that $s\sigma \rightarrow_{E,\succ} t\sigma$ whenever $s = t \in E$ and $s\sigma \succ t\sigma$.

Let us recall that a term t is *ground reducible* w.r.t. a rewrite system $\mathcal{R}$ iff all instances $t\sigma \in T(F)$ of t are reducible by $\mathcal{R}$. This definition extends to ordered rewriting, replacing $\mathcal{R}$ with $\rightarrow_{E,>}$ when E is a finite set of (unconstrained) equations.

Ground reducibility is decidable for arbitrary finite term rewriting systems [19]. It is undecidable for finite sets of *equations* [9].

We show it is decidable in the special case where all symbols occcurring in the set of equations E have arity 0 or 1, (in that case we say that the equations are *unary*) and the ordering is $\succ_{rpo}$ with a total precedence. In fact, we can state a slightly more general result. An equation is said to be *semiground* if at least one member is a ground term.

Theorem 5. *Given a term t, and system of equations E such each element of E is either unary or semi-ground it is decidable whether t is ground reducible by $\rightarrow_{E,\succ_{rpo}}$.*

Given set of terms $G_1, \ldots, G_n$ and a function symbol f of arity n we denote by $f(G_1, \ldots, G_n)$ the set of terms $\{f(g_1, \ldots, g_n) \mid g_1 \in G_1, \ldots, g_n \in G_n\}$.

Lemma 8. *$\{u \in T(F) \mid s(u) \succ_{rpo} t(u)\}$, for unary terms s, t is either empty or $T(F)$.*

Proof: Applying the rpo definition we have that $s(x) \succ_{rpo} t(x)$ is either equivalent to $v(x) \succ_{rpo} x$ or $x \succ_{rpo} v(x)$ for some term v. $\square$

Lemma 9. *$\{w \in T(F) \mid \exists v : \ s(w) \succ_{rpo} t(v)\}$ is either equal to $T(F)$ or to $\{x \in T(F) \mid x \succ_{rpo} u\}$, where u is a ground term.*

Proof: $s(x) \succ_{rpo} t(y)$ for some y iff $s(x) \succ_{rpo} t(\perp)$ where $\perp$ is the minimal constant in the signature. By decomposition we get the result. $\square$

Lemma 10. *$\{u \in T(F) \mid u \succ_{rpo} t\}$ where t is a ground term, is recognizable by a tree automaton.*

Proof: It is by simple induction on t w.r.t. $\succ_{rpo}$. If t is $\perp$ this is trivial. Otherwise assume by the induction hypothesis that for all $t' \prec_{rpo} t$, $L_{t'} = \{u \in T(F) \mid u \succ_{rpo} t'\}$ is a regular tree language.

Let $t = f(t_1, \ldots, t_n)$. Assume w.l.o.g. that $t_1 \succeq_{rpo} t_2, \ldots, \succeq_{rpo} t_n$. Then $x \succ_{rpo} f(t_1, \ldots, t_n)$ iff either

1. $x = g(x_1, \ldots, x_m)$ and $f \succ g$ and for some i: $x_i \succ_{rpo} t$.
2. $x = g(x_1, \ldots, x_m)$ and $f \prec g$ and for all j: $x \succ_{rpo} t_j$.
3. $x = f(x_1, \ldots, x_n)$ and $\{x_1, \ldots, x_n\} \succ\!\!\succ_{rpo} \{t_1, \ldots, t_n\}$

The language L_t is regular since it can be defined as a component of the least solution of the following system of equations:

$$
\begin{aligned}
L_t &= L_1 \cup L_2 \cup L_3 \\
L_1 &= \bigcup_{g \prec f,\ 1 \leq j \leq m} g(h_1, \ldots, h_i, \ldots, h_m) \ \text{ where } \ h_i = \begin{cases} L_t & \text{if } i = j, \\ T(F) & \text{otherwise} \end{cases} \\
L_2 &= \bigcup_{f \prec g} \left(g(T(F), \ldots, T(F)) \cap \bigcap_{1 \leq i \leq n} L_{t_i} \right) \\
L_3 &= \bigcup_{1 \leq j \leq n,\ \sigma \in \mathcal{S}_n} f(l^j_{\sigma(1)}, \ldots, l^j_{\sigma(n)}) \ \text{ where } \ l^j_k = \begin{cases} t_k/\!\sim_{rpo} & \text{if } k < j, \\ L_{t_j} & \text{if } k = j, \\ T(F) & \text{if } k > j. \end{cases}
\end{aligned}
$$

and where $\mathcal{S}_n$ is the permutation group of $\{1, \ldots, n\}$. Note that L_2 and L_3 are regular by induction hypothesis since they are obtained by composition of languages of type L_u with $u \prec_{rpo} t$. $\square$

Since $\{u \in T(F) \mid t \succ_{rpo} u\}$ is the complement in $T(F)$ of the language $\{u \in T(F) \mid u \succ_{rpo} t\} \cup \{t\}$ which is regular we also have:

Lemma 11. *$\{u \in T(F) \mid t \succ_{rpo} u\}$ where t is a ground term, is recognizable by a tree automaton.*

Now given a set of equations E satisfying the hypothesis of Theorem 5 it can be decomposed as a set of orientable unary equations E_1 union a set E_2 of non-orientable unary ones union E_3 a set of semiground equations.

Hence by Lemma 8 and Lemma 9 the set of ground and reducible terms for E is the set of ground terms that have a subterm in the set:

$$\{l(t) \mid l(x) = r(x) \in E_1, l(x) \succ_{rpo} r(x), t \in T(F)\} \cup \{l(t) \mid l(x) = r(y) \in E_2, l(t) \succ_{rpo} r(\bot), t \in T(F)\}$$
$$\cup \{s \mid s = t \in E_3, s \succ_{rpo} t, t \in T(F)\} \cup \{s \mid s = t \in E_3, s \succ_{rpo} t, s \in T(F)\} \quad (5)$$

By Lemma 10 and Lemma 11 this set is a regular tree language. We denote it by RED_E.

Given a term t of $T(F, X)$ it can be decided whether it is ground reducible by the ordered rewrite system E since in this special case it amounts to checking whether t belongs to a regular tree language. This may be obtained as a consequence of stronger results [2]. We give here a simple direct proof.

Let $\mathcal{A} = (Q, Q_f, \Delta)$ be the automaton for RED_E. We may assume that all states are reachable. And $Q_f = \{q_f\}$. Assume that t has m variables $x_1, \dots, x_m$ with possibly repeated occurrences. Let $t(q_1, \dots, q_m)$ be the term (on an extended signature $F \cup Q$) obtained by replacing every variable x_i with a state q_i. We can compute the result of applying the automaton to it. Let $SUC = \{\langle q_1, \dots, q_m\rangle \mid t(q_1, \dots, q_m) \rightarrow^*_{\Delta} q_f\}$. Now it suffices to show that for every m-tuples of ground terms $\langle t_1, \dots, t_m\rangle$ we have

$$\langle t_1, \dots, t_m\rangle \rightarrow^*_{\Delta^m} \langle q_1, \dots, q_m\rangle \in SUC$$

This is a reachability problem on the product automaton Δ^m and therefore is decidable.

Remark: The result also holds for lexicographic path orderings.

Conclusion

Using a non-standard coding of words as trees and tree automata techniques we have been able to show decidability of the theory of total unary rpo. A question that remains open is whether the existential theory of rpo with *partial* precedence is decidable.

Acknowledgements: We thank Florent Jacquemard and Pierre Lescanne for their comments.

References

1. R. Caferra and N. Peltier. Disinference rules, model building and abduction. *Logic at work: Essays dedicated to the memory of Helena Rasiowa* (Part 5: Logic in Computer Science, Chap. 20). Physica-Verlag, 1998.
2. A-C. Caron, J-L. Coquide, and M. Dauchet. Encompassment properties and automata with constraints. In C. Kirchner, editor, *Proceedings 5th Conference on Rewriting Techniques and Applications, Montreal (Canada)*, volume 690 of *Lecture Notes in Computer Science*, pages 328–342. Springer-Verlag, 1993.
3. H. Comon. Solving inequations in term algebras. In *Proc. 5th IEEE Symposium on Logic in Computer Science (LICS),* Philadelphia, June 1990.
4. H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, M. Tommasi. *Tree Automata Techniques and Applications.* http://www.grappa.univ-lille3.fr/tata/
5. H. Comon, P. Narendran, R. Nieuwenhuis, and M. Rusinowitch. Decision problems in ordered rewriting. In Proc. 13th IEEE Symp. Logic in Computer Science (LICS'98), Indianapolis, IN, USA, June 1998, pages 276-286, 1998.
6. H. Comon and R. Treinen. The first-order theory of lexicographic path orderings is undecidable. *Theoretical Computer Science* 176, April 1997.

7. N. Dershowitz. Orderings for term-rewriting systems. *Theoretical Computer Science* 17(3): 279–301. 1982.
8. J.-P. Jouannaud and M. Okada. Satisfiability of systems of ordinal notations with the subterm ordering is decidable. In *18th International Colloquium on Automata, Languages and Programming (ICALP),* volume 510 of *Lecture Notes in Computer Science*, pages 455-468, Madrid, Spain, July 1991. Springer-Verlag.
9. D. Kapur, P. Narendran, D. Rosenkrantz and H. Zhang. Sufficient Completeness, Ground-Reducibility and Their Complexity. *Acta Informatica* 28 (1991) 311-350.
10. D. Kapur, P. Narendran and G. Sivakumar. A path ordering for proving termination of term rewriting systems. In H. Ehrig (ed.), *10th CAAP,* volume 185 of *Lecture Notes in Computer Science*, pages 173-187, Berlin, March 1985.
11. D. Kapur and H. Zhang. An Overview of Rewrite Rule Laboratory (RRL), J. of Computer and Mathematics with Applications, 29, 2, 1995, 91-114.
12. C. Kirchner, H. Kirchner, and M. Rusinowitch. Deduction with symbolic constraints. *Revue Franaise d'Intelligence Artificielle*, 4(3):9–52, 1990. Special issue on automatic deduction.
13. P. Lescanne. Some properties of the Decomposition Ordering, a simplification ordering to prove termination of rewriting systems. RAIRO, 16(4):331–347, 1982.
14. U. Martin and E. Scott. The order types of termination orderings on monadic terms, strings and multisets. J. Symbolic Logic 62 (1997) 624–635.
15. P. Narendran, M. Rusinowitch and R. Verma. RPO constraint solving is in NP. In: Computer Science Logic (CSL 98), Brno, Czech Republic. August 1998. 12p. LNCS 1584, Springer-Verlag, 1999.
16. W. McCune and R. Padmanabhan. Automated Deduction in Equational Logic and Cubic Curves, Springer-Verlag LNCS 1095 (1996)
17. R. Nieuwenhuis. Simple LPO constraint solving methods. *Inf. Process. Lett.* 47(2):65–69, Aug. 1993.
18. R. Nieuwenhuis and A. Rubio. Theorem proving with ordering constrained clauses. In D. Kapur, editor, Proceedings of 11th Conf. on Automated Deduction, Saratoga Springs, NY, 1992, volume 607 of *Lecture Notes in Artificial Intelligence,* pages 477–491, June 1992, Springer-Verlag.
19. D. Plaisted. Semantic confluence tests and completion methods. *Information and Control*, 65:182–215, 1985.
20. J. Steinbach. Extensions and comparison of simplification orderings. Proceedings of 3rd RTA, Chapel Hill, NC, volume 355 of *Lecture Notes in Computer Science,* pages 434-448, 1989, Springer-Verlag.
21. C. Weidenbach. SPASS: Combining Superposition, Sorts and Splitting in A. Robinson and A. Voronkov, editors, Handbook of Automated Reasoning, Elsevier, 1999. To appear.