

A New Decidability Technique for Ground Term Rewriting Systems with Applications

RAKESH VERMA and ARA HAYRAPETYAN
University of Houston

Programming language interpreters, proving equations (e.g. $x^3 = x$ implies the ring is Abelian), abstract data types, program transformation and optimization, and even computation itself (e.g., turing machine) can all be specified by a set of rules, called a rewrite system. Two fundamental properties of a rewrite system are the confluence or Church–Rosser property and the unique normalization property. In this article, we develop a standard form for ground rewrite systems and the concept of standard rewriting. These concepts are then used to: prove a pumping lemma for them, and to derive a new and direct decidability technique for decision problems of ground rewrite systems. To illustrate the usefulness of these concepts, we apply them to prove: (i) polynomial size bounds for witnesses to violations of unique normalization and confluence for ground rewrite systems containing unary symbols and constants, and (ii) polynomial height bounds for witnesses to violations of unique normalization and confluence for arbitrary ground systems. Apart from the fact that our technique is direct in contrast to previous decidability results for both problems, which were indirectly obtained using tree automata techniques, this approach also yields tighter bounds for rewrite systems with unary symbols than the ones that can be derived with the indirect approach. Finally, as part of our results, we give a polynomial-time algorithm for checking whether a rewrite system has the unique normalization property for all subterms in the rules of the system.

Categories and Subject Descriptors: F.4.2 [Mathematical Logic and Formal Languages]: Grammars and Other Rewriting Systems—*Decision problems*

General Terms: Algorithms, Languages, Theory

Additional Key Words and Phrases: Confluence, decision procedures, pumping lemma, rewriting, standard forms, unique normalization

A substantial part of this research was done while A. Hayrapetyan was a student at the University of Houston.

The research of R. Verma was supported in part by the National Science Foundation (NSF) grants CCR-9732186 and CCR-0306475.

The research of A. Hayrapetyan was supported in part by NSF grant CCR-9732186.

The confluence results of this article were presented at LICS 2001 (short paper) and at the BISFAI 2001 Conference in June 2001.

Authors' present addresses: R. Verma, Computer Science Department, University of Houston, 4800 Calhoun Road, Houston, TX 77204-3010; email: rmverma@cs.uh.edu; A. Hayrapetyan, Computer Science Department, Cornell University, Ithaca, NY 14853-7501; email: ara@cornell.edu.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or direct commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 1515 Broadway, New York, NY 10036 USA, fax: +1 (212) 869-0481, or permissions@acm.org.

© 2005 ACM 1529-3785/05/0100-0102 \$5.00

1. INTRODUCTION

Programming language interpreters, proving equations (e.g., $x^3 = x$ implies the ring is Abelian), abstract data types, program transformation and optimization, and even computation itself (e.g., Turing machine) can all be specified by a set of rules, called a rewrite system. Two fundamental properties of a rewrite system are the confluence or Church-Rosser [Church and Rosser 1936] property and the unique normalization property. Informally, confluence states that if an expression a can be reduced (in zero or more steps) to two different expressions b and c , then there is a common expression d to which b and c can be reduced in zero or more steps. Unique normalization states that no expression can be reduced to more than one irreducible expression. Confluence implies uniqueness of normal (“irreducible”) forms and helps to “determinize” their search by avoiding backtracking. In general, both confluence and unique normalization are well-known to be undecidable; however, they are known to be decidable for the subclass of variable-free (“ground”) systems. Ground systems include as a subclass the tree automata model, which has important computer science applications. The following examples from Dauchet et al. [1990] may give some idea of the confluence problem: the system $\{a \rightarrow f(a, b), f(a, b) \rightarrow f(b, a)\}$ is not confluent, whereas the system $\{g(f(a)) \rightarrow f(g(f(a))), g(f(a)) \rightarrow f(f(a)), f(f(a)) \rightarrow f(a)\}$ is confluent.

In this article, we develop a standard form for ground rewrite systems and the concept of standard rewriting. The concepts are then used to: prove a pumping lemma for them, and to derive a new and direct decidability technique for decision problems of ground rewrite systems. We then apply these concepts and technique to the confluence and unique normalization properties of ground rewrite systems and prove the following results: (i) polynomial size bounds for witnesses to violations of confluence and unique normalization for ground rewrite systems containing unary symbols and constants, and (ii) polynomial height bounds for witnesses to violations of unique normalization and confluence for arbitrary ground systems. Finally, as part of the results, we give the first (to the best of our knowledge) polynomial time algorithm for checking whether a rewrite system has the unique normalization property for all subterms in the system. We also construct a hierarchy of (not necessarily ground) systems which enables us to simulate any rewrite system with a system containing only 1 binary symbol and constants. Parts of this hierarchy may be “folklore”, it is included in the appendix for completeness.

The outline of the proofs is as follows: We design a standard form for ground rewrite systems and define standard reduction sequences, which are special reduction sequences with useful properties. We believe that this form and such sequences will have potential applications beyond this article. In particular, these sequences give a very easy to implement algorithm for reductions starting from left-hand sides of rules. We then prove *polynomial height* bounds for witnesses to nonconfluence and to nonunique normalization. These bounds are of independent interest and lead to more efficient algorithms for unary systems than those using tree automata techniques, viz., membership in PSPACE.

There are two existing works on the decidability of confluence for ground rewrite systems. Simultaneously and independently Oyamaguchi [1987] and Dauchet et al. [1990] proved the decidability using tree-automata and the ground tree transducer models respectively. The latter model was introduced specifically to solve this problem, but later found other applications. No explicit polynomial bounds are proved in these papers on the size or height of witnesses and no upper bounds are specified. One infers that the complexity bounds from these papers are in EXPTIME [Seidl 1990] for general and PSPACE for unary signatures, but the precise upper bounds still require tedious calculations and are likely to be high. In contrast, our proofs are direct and significantly shorter, and we get sharper upper bounds and complexities for unary systems. For unique normalization, decidability follows from the work of Dauchet and Tison [1990] but again no precise bounds are known. After this work was completed, we learned of polynomial-time algorithms for the confluence problem [Comon et al. 2001; Tiwari 2002] of ground systems. However, these papers are focussed on the confluence problem, whereas this article has a broader scope. Even though it is likely that efficient algorithms can be designed for the specific problems considered here, we think that the tools and techniques here are significant since they are applicable to more than one problem.¹ As another illustration of our technique, we mention that termination of ground rewrite systems is easily shown decidable using the techniques presented here, and, of course, a polynomial-time algorithm for this problem is known from Plaisted [1993].

The rest of the article is organized as follows. In Section 2, we present the basic definitions. In Section 3, we collect some simplifying observations. In Section 4, we present the standard form and standard rewrite sequences and prove two useful lemmas. In Section 5, we present a polynomial-time algorithm for a special case of the unique normalization problem, which is used in Section 6 for proving polynomial bounds for unique normalization. In Section 7, we present the bounds for confluence and Section 8 concludes the article.

2. PRELIMINARIES

We assume some familiarity with basic notions of rewriting (see Dershowitz and Plaisted [2001] and Klop [1992]). Let V be a countable set of *variables* and Σ be a countable set of *function symbols* with $\Sigma \cap V = \emptyset$. \mathcal{T} is the set of all terms of a first-order language constructed from V and Σ . It is convenient to think of terms as ordered rooted trees. The *height* of a term t , denoted $ht(t)$, is the height of the corresponding tree and its *size*, denoted $size(t)$, is the number of nodes in the corresponding tree. If $s \in \mathcal{T}$, then $Var(s)$ denotes the *set* of variables in s . A term s is *ground* if $Var(s) = \emptyset$. A *substitution* is a map from the set of variables to the set of terms such that it is the identity on all except a finite set of variables. We extend substitutions to the set of all terms in the usual way and make no distinction between a substitution and its extension. We use

¹In fact, subsequent to this work, a polynomial time algorithm for unique normalization was achieved by Verma [2002b], again using techniques specific for this problem.

\square to represent a special constant called box and terms containing at least one box are called *contexts* and denoted by $C[\square, \dots, \square]$, where the C contains no occurrences of \square .

Definition 2.1 (Rule). A rule is a pair of terms $l \rightarrow r$, such that $l \notin V$ and $\text{Var}(r) \subseteq \text{Var}(l)$ (the variables in a rule are implicitly universally quantified). A rule $l \rightarrow r$ is ground if both l, r are ground. A (ground) *system* R is a finite set of (ground) rules. A rule $l \rightarrow r$ is *increasing* if $\text{size}(l) < \text{size}(r)$ and *decreasing* if $\text{size}(l) > \text{size}(r)$.

The *size* of a rule $l \rightarrow r$ is $\text{size}(l) + \text{size}(r)$. The *size* of a rewrite system, denoted by $|R|$, is the sum of the sizes of all the rules. The notion of a *path* or *occurrence* is used to refer to subterms in a term as follows.

Definition 2.2 (Path). A path is either the empty string λ that reaches the root or $o.i$ (o is a path and i an integer) which reaches the i th argument of the root of the subterm reached by o .

The (natural) ordering on occurrences is $o \leq q$ whenever $\exists p \ o.p = q$; if $p \neq \lambda$ also, then $o < q$. $t|_o$ refers to the subterm of t reached by o .

Definition 2.3 (Rewrite Relation). We say $s \rightarrow t$ (s reduces to t) if there is a rule $l \rightarrow r$ and a substitution σ such that $s = C[\sigma(l)]$ and $t = C[\sigma(r)]$. A reduction sequence $q : t_1 \rightarrow t_2 \rightarrow \dots \rightarrow t_n$ is a sequence of reduction steps, and its length is denoted by $|q|$.

Notation. When we want to emphasize the system R , we add the subscript R to \rightarrow to get \rightarrow_R . The reflexive-transitive closure of \rightarrow is denoted by \rightarrow^* . We use the term *root reduction* to indicate reduction of the entire term, denoted \xrightarrow{r} , and *nonroot reduction* to indicate reduction at a proper subterm, denoted \xrightarrow{nr} . $s \uparrow t$ denotes that there is a u such that $u \xrightarrow{*} s$ and $u \xrightarrow{*} t$.

Definition 2.4 (Confluence). We say that relation \rightarrow is *confluent* (or Church–Rosser, CR for short) for a term s if and only if $\forall t, u \ s \xrightarrow{*} t$ and $s \xrightarrow{*} u$ implies $\exists v$ such that $t \xrightarrow{*} v$ and $u \xrightarrow{*} v$. We say that \rightarrow is confluent for a set of terms T if it is confluent for every term in T . If $T = \mathcal{T}$, then we say \rightarrow is confluent. A system is confluent for T if its rewrite relation is confluent for T .

Definition 2.5 (Normal Form, UN^{\rightarrow}). Let R be a rewrite system. A term t is a *normal form* if there is no u such that $t \rightarrow u$. A term t *has a normal form* if there is a normal form u such that $t \xrightarrow{*} u$. R (or \rightarrow_R) is *uniquely normalizing* (is UN^{\rightarrow}) if for all s, t, u such that $s \xrightarrow{*} t$ and $s \xrightarrow{*} u$ and t, u are normal forms we have $t \equiv u$ (identical syntactically).

LEMMA 2.6 ([KLOP 1992]). *For every system R : $CR \Rightarrow UN^{\rightarrow}$, but not the reverse.*

Definition 2.7 (Reachability Problem). *Instance:* System, R , terms s, t . *Ques.:* Does $s \xrightarrow{*}_R t$?

Definition 2.8 (Joinability Problem). *Instance:* Rewrite System, R , terms s, t .

Ques.: Is there a term u such that $s \xrightarrow{*}_R u$ and $t \xrightarrow{*}_R u$? If there is such a u , we write $s \downarrow t$, otherwise, $s \not\downarrow t$.

Definition 2.9 (Confluence/ UN^{\rightarrow} Problem). *Instance:* System R . *Ques.:* Is R confluent/ UN^{\rightarrow} ?

Definition 2.10 (Witness of Nonconfluence). A pair (u, v) is called a *witness for nonconfluence* if $u \uparrow v$ and $u \not\downarrow v$.

Definition 2.11 (Witness of Nonunique Normalization). A pair (u, v) is called a *witness for nonunique normalization* if $u \uparrow v$ and u, v are distinct normal forms.

Whenever it is clear from the context, we will just say witness, that is, we drop the phrase “for nonconfluence” and “for nonunique normalization.”

3. OBSERVATIONS ON CONFLUENCE AND UNIQUE NORMALIZATION

(**) For the rest of this article, we will assume that all rewrite systems are ground unless explicitly stated otherwise.

The following results have been proved earlier (in Plaisted [1993] for reachability, and in Verma [2002a] for joinability) and will be extensively used here.

LEMMA 3.1. *Given a rewrite system R and (u, v) , the reachability and joinability problems can be solved in time polynomial in $\text{size}(u) + \text{size}(v) + |R|$.*

3.1 Simplifying the Confluence Problem

COROLLARY 3.2. *Given a pair (u, v) , whether it is a witness can be tested in time polynomial in $\text{size}(u) + \text{size}(v) + |R|$.*

PROOF. Note that if u and v are reachable from the same term via R , then $u \downarrow v$ in R^{-1} , where $l \rightarrow r \in R^{-1}$ if $r \rightarrow l \in R$. So we can check whether they are joinable in R^{-1} and then check that they are not joinable in R . From the above lemma, all this can be done in time polynomial in $\text{size}(u) + \text{size}(v) + |R|$. \square

We now show that checking the confluence of any (not necessarily ground) system can be restricted to derivations originating from left-hand side instances only. This result was proved earlier in Oyamaguchi [1987] for ground rewrite systems and claimed (“proof is similar”) for general systems. We give a shorter, clearer proof of it here to make the article as self-contained as possible.

LEMMA 3.3. *Any general (not necessarily ground) rewrite system is confluent (for all terms) iff it is confluent for all instances of all left-hand sides.*

PROOF. The only if direction is trivial. For the if direction, we proceed by contradiction. Assume that the rewrite system is confluent for all instances of all left-hand sides but it is not confluent for some term t . Let t be the smallest height term with this property. Now consider the sequences $t \xrightarrow{*}_R u$ and $t \xrightarrow{*}_R v$ such that $u \not\downarrow v$. Among the pairs (u, v) satisfying these conditions, choose the pair with the smallest sum of the lengths of the reduction sequences. If there are no root reduction steps in both sequences, then $t = f(t_1, \dots, t_n)$, $u = f(u_1, \dots, u_n)$

and $v = f(v_1, \dots, v_n)$, with $t_i \xrightarrow{*} u_i$ and $t_i \xrightarrow{*} v_i$. Now if $u_i \downarrow v_i$ for all i , then this contradicts $u \not\downarrow v$, therefore for some i , $u_i \not\downarrow v_i$. But, then t_i is a term of smaller height than t and satisfies the nonconfluence property, which contradicts the choice of t . Thus, our assumption that there is no root reduction step in both the sequences must be false. So let $t \xrightarrow{*} u$ be the sequence such that $t \xrightarrow{*} x \xrightarrow{r} y \xrightarrow{*} u$, where $x \xrightarrow{r} y$ is the first root step in that sequence. Then, since the subsequence $t \xrightarrow{*} x$ is strictly smaller than $t \xrightarrow{*} u$, we must have $x \downarrow v$ say at term z . But, then we are done since x is an instance of a left-hand side with reduction sequences to u and to z and u and z cannot be joinable since otherwise $u \downarrow v$ also. \square

Remark. This lemma implies that to check for the nonconfluence of any arbitrary *ground* system we need only to look for witnesses whose terms are reachable from a left-hand side.

3.2 Observations on Unique Normalization

COROLLARY 3.4. *Given a pair (u, v) , whether it is a witness can be tested in time polynomial in $\text{size}(u) + \text{size}(v) + |R|$.*

PROOF. Checking whether u and v are distinct normal forms can be done in time polynomial in $\text{size}(u) + \text{size}(v) + |R|$. Now, if there is an s such that $s \xrightarrow{*} u$ and $s \xrightarrow{*} v$, then $u \downarrow v$ using R^{-1} . The rest follows from Lemma 3.1. \square

Unfortunately, there is no lemma for the unique normalization property that corresponds to the lemma for confluence above, viz., Lemma 3.3. In other words, for unique normalization it does not suffice to consider sequences beginning with only instances of left-hand sides. To see this, consider the following ground system.

Example. Let $R = \{f(b) \rightarrow b, b \rightarrow a, a \rightarrow a, f(a) \rightarrow c\}$. Then, R has the unique normalizing property for all left-hand sides. But, $f(f(b)) \xrightarrow{*} c$ and $f(f(b)) \xrightarrow{*} f(c)$ so R is not uniquely normalizing.

Notation. For convenience, we use the following abbreviations for describing the classes of systems: b for binary, u for unary, c for constants, M for many. For example, a system of type 1bMc contains 1 binary symbol and 1 or more constants.

In the Appendix, we show how to build a hierarchy of systems of increasing expressive power. First, we show that *any* general rewrite system (not necessarily ground) can be transformed in polynomial time into a system containing only binary symbols and constants. We then show that the latter type of systems can be converted in polynomial time into systems containing only 1 binary symbol and constants. Thus, any general rewrite system can be converted in polynomial time into a system containing only one binary symbol and constants. Finally, we show that any system containing only unary symbols and constants can be converted in polynomial time into a system containing only unary symbols and a single constant. Together, these results imply a hierarchy of systems.

4. STANDARD FORMS, STANDARD SEQUENCES AND PUMPING LEMMA

For any system R , let $subterm(R)$ denote the set of all subterms of terms appearing in R . We now define a standard form for rewrite systems.

Definition 4.1 (Standard Form). Given a system R , we define its standard system R^* as follows: for every pair $x, y \in subterm(R)$ (where x and y are not necessarily distinct) we add the rule $x \rightarrow y$ to R^* , iff $x \xrightarrow{*}_R y$.

Note that the standard form of a rewrite system can be constructed in polynomial time since the reachability problem can be solved in time that is a polynomial function of $|R|$ for all the pairs from $subterm(R)$. Note also that we are changing the rewrite system itself and working with the new system as opposed to just solving the reachability problem, which is part of the goal in Plaisted [1993]. Hence, this definition of standard form should not be confused with the reachability algorithm for a ground rewrite system as in Plaisted [1993].

LEMMA 4.2. R is confluent iff R^* is confluent.

PROOF. Straightforward since $u \xrightarrow{*}_R v$ iff $u \xrightarrow{*}_{R^*} v$. \square

Definition 4.3 (lsubterm/rsbterm). Let $t = f(u, v)$ be any term. We will denote u by $lsubterm(t)$ and v by $rsbterm(t)$.

Definition 4.4. A reduction sequence $q : x \xrightarrow{*}_R y$ is said to be *distinguished* or *standard* if no later reduction step is applied at an occurrence that is a prefix of the occurrence at which a previous step was applied.

Remark. We can also make these sequences unique, if necessary, by insisting that the reductions are carried out in some specific order, for example, leftmost-outermost, or left-to-right breadth-first, etc.

LEMMA 4.5. Let R be a given rewrite system of type 1bMc. Then, for every reduction sequence $q : x \xrightarrow{*}_R y$ starting from $x \in subterm(R)$ there is a distinguished reduction sequence $p : x \xrightarrow{*}_{R^*} y$, such that $|p| = size(y)$.

PROOF. We will apply mathematical induction on $size(y)$ to prove this lemma.

Basis: If $size(y) = 1$, then we know that y is a reachable constant, thus $y \in subterm(R)$. Since we are starting from a left-hand side, we know that $x \in subterm(R)$. Now, by the construction of R^* , $x \rightarrow y \in R^*$. From which it follows that the reduction $x \rightarrow y$ can be carried out in precisely one step in R^* .

Inductive Step: Assume that the lemma holds for all terms t , such that $size(t) < size(y)$. We will prove that this proposition also holds for the term y . There are two cases.

Case 1. There is a root rewrite in the reduction sequence $x \xrightarrow{*}_R y$. In this case, let z be the result after applying the last root rewrite in the reduction sequence. Then $x \xrightarrow{*}_R t \xrightarrow{r} z \xrightarrow{*}_R y$, and $z \in subterm(R)$ (because z is a right-hand side). We know that $x \rightarrow z \in R^*$ (because $x, z \in subterm(R)$), and thus $x \xrightarrow{*}_R z$ can be implemented in one step in R^* . Since z was the result of the last root rewrite, we can conclude that, in the sequence $z \xrightarrow{*}_R y$, there has been no root rewrite applied.

Thus, only the two subterms of z were modified to reach the corresponding subterms of y . But since the subterms of y have size less than that of y , and since the subterms of z are also in $\text{subterm}(R)$, we can claim from the inductive hypothesis, that the left subterm of z can reach the left subterm of y in $\text{size}(\text{lsubterm}(y))$ steps in R^* , and similarly the right subterm of z can reach the right subterm of y in $\text{size}(\text{rsubterm}(y))$ steps in R^* . Thus, the number of steps to reach y from x in R^* will be $1 + \text{size}(\text{lsubterm}(y)) + \text{size}(\text{rsubterm}(y)) = \text{size}(y)$ (the 1 comes from the rule $x \rightarrow z$).

Case 2. There is no root rewrite in the reduction sequence $x \xrightarrow{*} y$. In this case, we add an additional reduction rule $x \rightarrow x$ in the beginning of the sequence in R^* (note that the construction of R^* has introduced such an identity rule), and since $x \xrightarrow{*} y$ contains no root rewrite, we apply the argument of the previous case to achieve the desired reduction sequence in R^* (z becomes x , and the 1 in the final count comes from the identity rule). \square

Remark. Note that as a special case this lemma is easily proved for systems of type Mu.

THEOREM 4.6 (PUMPING LEMMA). *Let R be any ground rewrite system. Let y be any term such that $\text{ht}(y) > |R^*|$ and there exists $x \in \text{subterm}(R)$ such that $x \xrightarrow{*} y$. Then, there exist: (i) a rule $l \rightarrow r \in R^*$, (ii) contexts C, D with D non-empty such that $y = C[D[z]]$ for some z , and (iii) sequences $x \xrightarrow{*} C[l]$, and $l \xrightarrow{*} D[l] \xrightarrow{*} D[z]$. Hence, for every $n \geq 0$, $x \xrightarrow{*} C[D^{(n)}[z]]$.*

PROOF. Straightforward from Lemma 4.5. \square

Remarks. The pumping lemma will be mostly used as a “cutting lemma”, that is, the case $n = 0$, where a part of a context is removed and the associated subsequence is deleted. Thus, we will use the phrases: “cutting” and “deleting the subsequence”, etc. We will flag some of the places where it is used by the words pumping lemma in parenthesis. In some cases, its use will be implicit. The pumping lemma may also be proved by first constructing a flat system for a ground system by renaming subterms by constants and then using tree-automata techniques [Comon et al. 1999]; but this does not immediately yield any of our results, and we are not aware of any previous work that has used this lemma as the basis for decidability of the two properties we consider here.

Standard sequences allow us to perform reductions in a special way, that is, we first apply a root reduction to the term, then root reductions to its immediate subterms, etc. (some of which may be identity reductions). This property will be exploited in the following proofs.

Unless stated otherwise, in the rest of this article, we assume that all the rewrite systems will first be converted to standard systems, and all reduction sequences starting from a term in $\text{subterm}(R)$ will be transformed to standard reduction sequences.

5. COMPUTING NORMAL FORMS

In this section, we present a polynomial-time algorithm to determine whether R has the uniquely normalizing property for elements in $\text{subterm}(R)$. In this

algorithm, we will represent the terms by Directed Acyclic Graphs (DAGs), because as the following example shows, we cannot represent the terms explicitly since then the algorithm will run in exponential time.

Example. $a_1 \rightarrow f(a_2, a_2), a_2 \rightarrow f(a_3, a_3), \dots, a_{n-1} \rightarrow f(a_n, a_n)$. In this example, the (tree) normal form of a_1 is exponentially large in the size of the system, and thus cannot be constructed in polynomial time.

Now we present an algorithm for determining for a given rewrite system R , which elements of $subterm(R)$ reach normal forms, and also the normal forms that they reach. If a term reaches more than 1 normal form, this will also be detected.

Before continuing, we modify the rewrite system R by adding all rules of the form $r \rightarrow s$ iff $r, s \in subterm(R)$ and $r \rightarrow^+ s$. We call this new system R^+ (transitive closure). Note that this construction can be done in polynomial time in R and doesn't change the normal form reachability property of the rewrite system, because a normal form in R would be a normal form in R^+ and vice versa. From now on, in this section, we will be working with transitive closures only, unless otherwise specified.

With each term $t \in subterm(R)$, we associate two flags: $t.in$, which is true iff t can reach a normal form that belongs to $subterm(R)$, and $t.out$, which is true iff t can reach a normal form not in $subterm(R)$. Also with each term $t \in subterm(R)$, we associate a linked list $t.nforms$ of pointers to the DAGs of the terms t_1, t_2, \dots , which are the normal forms that this term reaches. Note that if during the computation of the algorithm, any of these lists contains more than one element, then the rewrite system doesn't satisfy the uniqueness of normal forms property.

The intuition behind this algorithm is first to set the *in* flags (which can be done by exhaustive search), and then the *out* flags. The idea behind setting of the outside flags is based on the following three observations.

- (1) If term t reaches a term s that reaches an outside normal form, then t also reaches an outside normal form.
- (2) If for a term $t = f(u, v)$ either of its subterms reaches an outside normal form while the other subterm reaches an arbitrary normal form, then t also reaches an outside normal form.
- (3) The only other way for $t = f(u, v)$ to reach a normal form would be if its subterms reach inside normal forms u', v' respectively, such that $f(u', v')$ is an outside normal form.

ALGORITHM 1 (UNIQUELY NORMALIZING FOR SUBTERMS(R)).

```

for all  $t \in subterm(R)$ 
  if  $t$  is a normal form then
     $t.in = true$ 
    create a DAG for  $t$  and add a pointer to it to  $t.nforms$ 
  for all  $t \in subterm(R)$ 
    for all  $s \in subterm(R)$  such that  $t \rightarrow s$  and  $s$  is a normal form
       $t.in = true$ 
      add a pointer to the DAG of  $s$  to  $t.nforms$ 
     $t.out = false$ 

```

```

comment At this step all the in flags will be assigned their correct values
comment The rest of the algorithm will correct the values of out flags
comment The main loop.
until no normal forms were added or any nform list has  $\geq 2$  elements do
  comment The first case
  for all  $t, s \in \text{subterm}(R)$ 
    if  $t \rightarrow s$  and  $s.out = true$  then
       $t.out = true$ 
      add the pointer  $s.nforms$  to  $t.nforms$  if not already there
      continue to the next iteration of the until loop
    comment The second case
    for all  $t, s, u \in \text{subterm}(R)$ 
      if  $t == f(s, u)$  and  $s.out == true$  then
        if  $u.in == true$  or  $u.out == true$  then
           $t.out = true$ 
          create a DAG for  $f(s', u')$  and add a pointer to it to  $t.nforms$ 
          if not already there
          where  $s' \in s.nforms$  and  $u' \in u.nforms$ 
          continue to the next iteration of the until loop
        else if  $t == f(s, u)$  and  $u.out == true$  then
          if  $s.in == true$  or  $s.out == true$  then
             $t.out = true$ 
            create a DAG for  $f(s', u')$  and add a pointer to it to  $t.nforms$ 
            if not already there
            where  $u' \in u.nforms$  and  $s' \in s.nforms$ 
            continue to the next iteration of the until loop
          comment The third case
          for all  $t, s, u \in \text{subterm}(R)$ 
            if  $s.in == true$  and  $u.in == true$ 
              let  $s' \in s.nforms, u' \in u.nforms$ 
              if  $t == f(s, u)$  and  $f(s', u')$  is an outside normal form then
                 $t.out = true$ 
                create a DAG for  $f(s', u')$  and add a pointer to it to  $t.nforms$ 
                if not already there
                continue to the next iteration of the until loop
            end until
  end until

```

It is not hard to see that the inside flags will be set correctly after the execution of the first part of the algorithm.

We now prove the correctness of Algorithm 1.

THEOREM 5.1. *If Algorithm 1 exited because it found more than one element in any of the lists, then the system doesn't satisfy the unique normalization property. Otherwise, the algorithm will find for each term in $\text{subterm}(R)$ the normal form associated with it (if any).*

PROOF. First, we show the soundness of the above algorithm. As mentioned above, it is trivial to see that the *in* flags will be correctly set. For the *out* flags, we observe that initially (after setting them to *false*) the result is sound. Then we show that each of the above cases (1 through 3) preserves soundness. As a result, we will have that the final result is sound.

(1) Trivial. $t \rightarrow s$ means that any term reachable from s is also reachable from t .

- (2) Since for all elements $t \in \text{subterm}(R)$, $\text{subterm}(t)$ is a subset of $\text{subterm}(R)$, we can conclude that, if a term has a subterm not belonging to the set $\text{subterm}(R)$, then the term itself will not belong to the set $\text{subterm}(R)$. And because both of its subterms are normal forms, it will also be a normal form.
- (3) Again trivial, because there is a check being done if $f(u', v')$ is an outside normal form.

Now we show that if there exists a $t \in \text{subterm}(R)$ that is a witness of nonuniqueness of normal forms, the algorithm will find one. To show this, we prove that if t reaches any normal form s , then if this algorithm is executed long enough, it will find this normal form. We will prove this claim only for outside normal forms, since inside normal forms are trivial. The proof is by induction on the length of the shortest reachability path from t to s . For equal path lengths, we will perform induction on the size of t .

Basis: If the length is one and the size of t is smallest (ties are broken arbitrarily), that is $t \rightarrow s$ and also since s is an outside normal form, we can conclude that $t \rightarrow s$ was not a root step. Thus, t is of form $f(u, v)$ and this step reduces one of the subterms (assume u) to a normal form (denote by u'), and v must itself be a normal form (obviously it will be an inside normal form). That is, s must be of form $f(u', v)$.

Next, we note that u' cannot be an outside normal form, since then there would be a smaller t which reduces to an outside normal form in one step (because $\text{size}(u) < \text{size}(t)$). Thus, both u' and v are inside normal forms. Thus, the third case of our algorithm will find the normal form s associated with t .

Inductive Step: Assume now that $q : t \rightarrow^* s$ is the shortest reduction sequence leading from t to a normal form s . Let's assume that the first reduction performed in q was $t \rightarrow t'$. We note that if q contains a root reduction it must only appear at the beginning and must be only 1 reduction, because our system was transitively closed. If the first step is indeed a root step, then because of the inductive hypothesis the algorithm will find the normal form s for t' (because it is a shorter sequence). And due to the first case of the algorithm the normal form for t will also be found. If $t \rightarrow t'$ is not a root step, then as discussed above there can be no root rewrite in our system. So t is of form $f(u, v)$ where u reaches a normal form u' and v reaches a normal form v' , and s will be of form $f(u', v')$. Both of these normal forms will be found because $u \rightarrow^* u'$ and $v \rightarrow^* v'$ have at most the same lengths as $t \rightarrow^* s$ and have smaller sizes.

Finally, if any of u' or v' is an outside normal form, then the second step of the algorithm will find the normal form for t ; otherwise, the third step of the algorithm will find the normal form for t .

From the above, we can conclude that if the algorithm exited because of the second condition of the main loop, then there exists a term that has two normal forms, otherwise the algorithm will find at most one normal form for each term. \square

The following theorem will prove that this algorithm runs in polynomial time.

THEOREM 5.2. *Algorithm 1 runs in time polynomial in $|R|$.*

PROOF. Indeed, first we will show that the main loop will be executed at most $\text{subterm}(R) + 1$ times. This is easy to see, because each iteration will add precisely one normal form to one of the $nforms$ lists. And thus because of the pigeonhole principle, this loop will not be executed more than $\text{subterm}(R) + 1$ times.

Next it is easy to observe that the operation in the for loops for each of the three cases take polynomial time, since adding a DAG node can be done in constant time, checking for its existence (for sharing purposes) can be done in polynomial time if the DAG's are properly stored (in a hash table for instance). Also adding pointers from one list of normal forms to another is constant, since each list can consist of no more than 1 normal form (otherwise, the algorithm would have terminated earlier). Checking whether $f(s', u')$ is an outside normal form can also be done in polynomial time since in this case s' and u' are inside normal forms and therefore $O(|R|)$. Finally, the number of iterations of each for all loop is also polynomial. \square

Thus, by running the above algorithm we will know if any of the elements in $\text{subterm}(R)$ reach to different normal forms.

6. POLYNOMIAL BOUNDS AND COMPLEXITY OF UN^{\rightarrow}

6.1 Polynomial Height Bound for Arbitrary Ground Systems

In this section, we assume that R is UN^{\rightarrow} on $\text{subterm}(R)$. This can be ensured by first running Algorithm 1 and checking the condition which terminates the main loop. Because of the hierarchy, we may assume that R is a ground system of type 1bMc. Let \mathcal{B} denote $\max\{ht(n) \mid n \text{ is a normal form of some term in } \text{subterm}(R)\}$. We may assume that the set of heights is nonempty since otherwise R has the UN^{\rightarrow} property trivially. Note that \mathcal{B} is well defined because R is UN^{\rightarrow} on $\text{subterm}(R)$. In the previous section, we showed that \mathcal{B} is a polynomial function of $|R|$. We now prove a polynomial height bound for witnesses to non-unique normalization for arbitrary systems, which, with some extra work, can be used to derive an EXPTIME algorithm for arbitrary systems. The details of this derivation are omitted since such an algorithm may be derived using previous methods also.

THEOREM 6.1. *If R is not UN^{\rightarrow} , then there exists a witness (E, F) , such that there are DAG's G_1 and G_2 representing E and F respectively, with both heights $ht(G_1), ht(G_2)$ at most $M + N + 2\mathcal{B}$, where M is the number of rules in R^* and N is the height of the tree representing the highest left-hand side in R^* .*

PROOF. In the following \mathcal{S} will denote $(R^{-1})^*$ and $\mathcal{N} = \max(N, ht(E))$. We may assume, that R is not UN^{\rightarrow} , but R is UN^{\rightarrow} on $\text{subterm}(R)$. This can be ensured by first running Algorithm 1 and checking the condition which terminates the main loop. If the algorithm terminates without finding a violation of uniqueness of normal forms, then the normal forms (whenever they exist) of all left-hand sides have been computed and the compact DAG's representing

their normal forms are also of size \mathcal{B} . If the algorithm terminates by detecting a violation, then R is not UN^\rightarrow and the algorithm has found the desired witness. Since R is not UN^\rightarrow , by definition, there are distinct normal forms E and F and a term A such that $p : A \xrightarrow{*}_R E$ and $q : A \xrightarrow{*}_R F$. Let A be a smallest size term with this property. Since A is the smallest term with this property, either p or q must have a root reduction. Otherwise, we can get a smaller term also violating the UN^\rightarrow property, by picking one of the proper subterms of A . We have two cases:

Case 1. There are root reductions in both p and q . Then, there are left-hand sides B and D on p and q respectively to which these root reductions are applied. Then, by Algorithm 1, we have that $size(E), size(F) \leq \mathcal{B}$.

Case 2. Only one of p or q has a root reduction. Without loss of generality, let it be p . Then, we have $q : A \xrightarrow{nr^*}_R F$ and $p : A \xrightarrow{nr^*}_R B \xrightarrow{r}_R D \xrightarrow{*}_R E$. Let $O = \{o_1, o_2, \dots, o_n\}$ be the set of minimal occurrences to which reductions are applied in $q : A \xrightarrow{nr^*}_R F$. Then, we have that $A = C[A_1, A_2, \dots, A_n]$ and $F = C[F_1, F_2, \dots, F_n]$ for some context C and sequences $q_i : A_i \xrightarrow{*}_R F_i$. Consider the sequence $B \xrightarrow{*}_{R^{-1}} A$. Since B is a left-hand side of S , this implies a standard sequence $p_1 : B \xrightarrow{*}_S A$. Let $P = C[P_1, P_2, \dots, P_n]$ denote the term on p_1 that is obtained after exactly $|size(C)|$ many reductions constructing C have been applied, so that $P_i \xrightarrow{*}_S A_i$. Note that this can be done by rearranging (if necessary) the rewrites of a standard sequence in breadth-first fashion.

There cannot be occurrences $o < o'$ in C such that the same rule was applied at o and o' in the sequence p_1 and $A|_{o'}$ has at least \mathcal{N} descendant vertices on at least one path in C , since then we may cut the reduction sequence p_1 (pumping lemma) so that we eliminate this rule repetition getting a smaller term than A that also yields a witness. This would contradict the choice of A . The reason for insisting on at least \mathcal{N} descendants is to ensure that after these cuts the resulting term, $\mathcal{P} = C[P_1, P_2, \dots, P_n]$, yields a term $\mathcal{A} = C[A_1, A_2, \dots, A_n]$ (the context \mathcal{C} is obtained from the context C by replacing the “subterm” $C|_o$ with $C|_{o'}$), which in turn leads to a normal form $\mathcal{F} = C[F_1, F_2, \dots, F_n]$ through a sequence \mathcal{Q} corresponding to normal form F and sequence q . This is because any term in which all subterms of up to \mathcal{N} descendants are normal forms must itself be a normal form, since R is ground and \mathcal{N} is the height of the highest left-hand side of R^* , so also R . The reason for insisting on $ht(E)$ descendants is to ensure that \mathcal{F} remains distinct from E (since $ht(\mathcal{F}) > ht(E)$). We can now assume that there is no rule repetition along any path of C except possibly in a tail of the path whose size is bounded by \mathcal{N} . Then, there exists \mathcal{C} such that $ht(\mathcal{C}) \leq \mathcal{N} + \mathcal{M}$. Thus, there is a normal form \mathcal{F} such that $ht(\mathcal{F}) \leq \mathcal{N} + \mathcal{M} + \mathcal{B} \leq \mathcal{M} + \mathcal{N} + 2\mathcal{B}$ and we are done. \square

6.2 UN^\rightarrow is in P for 1uMc systems

THEOREM 6.2. *UN^\rightarrow is in P for ground systems of type 1uMc.*

PROOF. Height and size are the same for terms of such systems. Since there is only 1 unary symbol, there are only polynomially many pairs of terms with the height bound of Theorem 6.1. \square

6.3 UN^{\rightarrow} is in co-NP for MuMc Systems

THEOREM 6.3. *UN^{\rightarrow} is in co-NP for ground systems of type MuMc.*

PROOF. Height and size are the same for terms of such systems. Because there is more than one unary symbol, there are exponentially many pairs of terms of *polynomial* size with the bound given in Theorem 6.1. Therefore, an NP algorithm can be designed for detecting violations of the UN^{\rightarrow} property. \square

7. POLYNOMIAL BOUNDS FOR CONFLUENCE

7.1 Polynomial Size Bound for 1uMc Systems

THEOREM 7.1. *For a standard system R^* of type 1uMc, if R^* is not confluent, then there is a witness (u, v) such that $\text{size}(u), \text{size}(v) \leq a$, where $a = (M^2 + 1) + M + N + 1 = O(|R|^2)$, M is the number of rules in the system, and N is the size of the largest term in R^* .*

PROOF. Let us assume that we have a non-confluent system, but every witness (u, v) from a left-hand side violates the size restriction of the theorem. We arrange all the witnesses in accordance with the following rules.

- (1) Witnesses with smaller size difference come before witnesses with larger difference.
- (2) Witnesses with equal size difference are arranged according to the sum of the sizes with smaller sums first.

Now let's pick a smallest witness according to this ordering (ties are broken arbitrarily) and denote it by (A, B) . We will derive a contradiction assuming that this witness violates the size restriction of the theorem. Let l be the left-hand side that reaches A and B and let the two standard sequences to A and B be $p : l \rightarrow A_1 \rightarrow A_2 \rightarrow \dots \rightarrow A$ and $q : l \rightarrow B_1 \rightarrow B_2 \rightarrow \dots \rightarrow B$. There are 2 cases:

Case 1. Both A and B have size greater than a . In this case the standard sequences p, q have length more than a . Note that after the i^{th} reduction step in both p, q we will never again change the top i symbols.

Because both A and B have size greater than M^2 there are two indices j, k such that the same pair of rules is applied to get (A_j, B_j) and (A_k, B_k) (by the pigeonhole principle, because there are only M^2 pairs). Without loss of generality, assume that $j < k$. We know, that the number of symbols by which A_k differs from A_j is the same as that for B_k and B_j . Since p and q are standard sequences and since we do not change the top i symbols after reduction i in a standard sequence, therefore we can apply all the rules applied to A_k in p to A_j and similarly for B_j (pumping lemma). Let's assume that the new sequences yield the terms A' and B' . Because A' and B' are smaller by exactly the same number of symbols than respectively A and B , from the non-confluence of the chosen witness would follow the non-confluence of the pair (A', B') (recall that we have just 1 unary symbol), which has the same difference in size and smaller sum of sizes, thus it precedes our chosen witness in the ordering and is itself a witness. This contradicts the minimality of the chosen witness.

Case 2. If one of the terms (assume A) has size greater than α and the other (B) has size at most that, then we have 2 subcases.

Subcase 1. The $size(B) > M^2$. The reasoning here is the analogous to that of Case 1.

Subcase 2. The $size(B) \leq M^2$, meaning that $size(A) - size(B) > N + M$. Because (A, B) is the witness with the smallest size difference and N is the size of the largest term in the rewrite system, we conclude that no (size) increasing rules can be applied to B and similarly no decreasing rules can be applied to A . But, on the other hand because their size difference is more than number of rules M , we can conclude that two of the terms in the first $M + 1$ terms among the terms $A_{size(B)+1}, A_{size(B)+2}, \dots, A$ will have the same rule application at the end. Let's denote these terms by A_n and A_m ($n > m$). Note that since p is a standard sequence the rule application in A_n is "lower down" than in A_m and hence $ht(A_m) < ht(A_n)$. Let A' be the term obtained by applying on A_m the same sequence of rules that was applied on A_n to reach A (this is possible, since we recall that p is a standard reduction sequence, pumping lemma). Then, the terms A and A' only differ in that $size(A') < size(A)$ (this follows from the fact that A and A' end in the same rule application, and that each rule application in the standard reduction sequence fixes exactly 1 symbol). A' still ends in the same constant as A . Note also that $size(A') > size(B)$ (since $size(A) - size(A') \leq M$), and since it was impossible to decrease the size of A , it is similarly impossible to decrease the size of A' , or increase the size of B , thus (A', B) is also a nonconfluence witness, but it has a less size difference than (A, B) . Again a contradiction. \square

THEOREM 7.2. *There is an $O(|R|^9)$ -time algorithm for confluence of ground $1uMc$ systems.*

PROOF. Since the system has only one unary symbol and constants, there are only number-of-constants many terms of each size, which is at most $|R|$. Hence, the quadratic-size bound (in $|R|$) on the terms in the witness implies that we only have to check $O(|R|^6)$ many pairs of terms to find a witness if one exists. Checking joinability takes $O(|R|^3)$ time for each pair. Actually, a more efficient algorithm is possible that does all the joinability checks together. \square

7.2 Polynomial Size Bound for MuMc Systems

From the hierarchy constructed in Section A.1, it follows that the confluence problem for $MuMc$ systems can be polynomially reduced to the confluence problem for systems of type $Mu1c$. Now, we prove a polynomial size bound for witnesses to non-confluence of MuMc systems, which can be used to show that confluence problem is in co-NP for such systems.

Definition 7.3. Let P be the set of pairs of elements in $Subterm(R^*)$ that are joinable. To each joinable pair, we assign a number, which is the minimal size of the term through which they join. Then, $k(R^*)$ will be the maximum of all this numbers, where the joinable pair is picked from P .

LEMMA 7.4. *$k(R^*)$ is polynomial in $|R|$ for R of type $Mu1c$.*

PROOF. Straightforward from the polynomial time algorithm for joinability [Verma 2002a], since it gives a polynomial height bound on a term at which two terms join. For systems of type $Mu1c$, height and size are the same. Note that $|R^*|$ is at most quadratic in $|R|$. \square

Notation. Let $k = \max(2, k(R^*))$.

THEOREM 7.5. *For a standard system R^* containing only unary symbols and one constant, if R^* is not confluent, then there is a witness (u, v) , such that $\text{size}(v) \leq \text{size}(u) \leq b$, where $b = k(M^2 + 1) + M + N + 1$, M is the number of rules in R^* , and N is the size of the largest term in R^* .*

PROOF. In the Appendix. \square

Remark. Clearly, this bound also applies to $1uMc$ systems, but the one proved earlier is sharper.

COROLLARY 7.6. *Confluence for $MuMc$ systems is in $co-NP$.*

7.3 Polynomial Height Bound for Arbitrary Ground Systems

From the hierarchy constructed in Section A.1, it follows that the confluence problem for arbitrary ground systems can be polynomially reduced to the confluence problem for systems of type $1b$. Now, we prove a polynomial height bound for witnesses that can, with some additional work, be used to derive an (deterministic) EXPTIME algorithm. Details of this derivation are omitted since such an algorithm is already available through tree automata techniques.

The following theorem proves a polynomial bound on the height of the witnesses for nonconfluence.

THEOREM 7.7. *For a standard system R^* containing only 1 binary symbol and constants, if R^* is not confluent, then there is a witness (u, v) , such that $ht(v) \leq ht(u) \leq ((M^2 + 1) + (N + 1)) + (M + 1) + 2(N + 1)$, where M is the number of rules in R^* , and N is the height of the highest term in R^* .*

PROOF. In the Appendix. \square

8. CONCLUSIONS

In this article, we have developed a standard form for ground rewrite systems and standard rewrite sequences and using these concepts proved a pumping lemma for them and derived a new decidability technique for decision problems of ground rewrite systems. We then applied these concepts and technique to prove: (i) polynomial size bounds for witnesses to violations of confluence and unique normalization for ground rewrite systems containing unary symbols and constants, and (ii) polynomial height bounds for witnesses to violations of confluence and unique normalization for arbitrary ground systems. It seems likely that the standard form and such sequences may have other potential applications beyond this paper. Our proofs are direct and significantly shorter than earlier work, and we get sharper upper bounds and complexities for unary systems than are obtainable using tree automata techniques. Finally, as part

of the results, we gave the first (to the best of our knowledge) polynomial-time algorithm for checking whether a rewrite system has the unique normalization property for all subterms in the system.

An interesting direction for future research is to study conditions under which the framework presented here can be “lifted” to nonground classes of rewrite systems.

APPENDIX A

A.1 A Hierarchy of Systems

In this section, we show how to build a hierarchy of systems of increasing expressive power. First, we show that *any* general rewrite system (not necessarily ground) can be transformed in polynomial time into a system containing only binary symbols and constants. We then show that the latter type of systems can be converted in polynomial time into systems containing only 1 binary symbol and constants. Thus, any general rewrite system can be converted in polynomial time into a system containing only one binary symbol and constants. Finally, we show that any system containing only unary symbols and constants can be converted in polynomial time into a system containing only unary symbols and a single constant. Together these results imply a hierarchy of systems.² All of the transformations can be carried out in polynomial time. To prove that the transformed rewrite system is confluent (over an extended signature) if and only if the original system is confluent we will use the following theorem [Zantema 1994] whenever possible. It states that proving confluence for arbitrary terms over the signature is equivalent to proving confluence of the well-typed terms according to any many-sorted discipline which is compatible with the rewrite system under consideration. This is called *persistence* in Zantema [1994].

THEOREM A.1. *Confluence is a persistent property.*

Notation. For convenience, we use the following abbreviations for describing the classes of systems: b for binary, u for unary, c for constants, M for many. For example, a system of type 1bMc contains 1 binary symbol and 1 or more constants.

A.1.1 General Systems \rightarrow Mb Transformation. The transformation consists of the following steps. We denote by t' the transformed term t .

- (1) Each variable and each constant is mapped to itself.
- (2) For each unary symbol g , we add a new unique binary symbol g' and a new unique constant c_g to the new system. The term $g(t)$ for any t is mapped to the term $g'(t', c_g)$.
- (3) For each symbol h having arity $n \geq 2$, we add n new binary symbols H_1, \dots, H_{n-1}, h' and a new unique constant c_h to the new system. The term

²There is a technicality that needs to be addressed, viz., whether the signature (constants and function symbols) of terms is assumed to be fixed or not. If the signature is not fixed, then the new symbols must also be transformed as they are introduced.

$h(t_1, t_2, \dots, t_n)$ is mapped to the term

$$H_1(h'(t'_1, c_h), H_2(h'(t'_2, c_h), \dots, H_{n-1}(h'(t'_{n-1}, c_h), h'(t'_n, c_h)) \dots))$$

for any t_1, t_2, \dots, t_n .

Remark. The reader may wonder why we do not map the term $h(t_1, t_2, \dots, t_n)$ to the term $h'(t'_1, h'(t'_2, \dots, h'(t'_n, c_h)) \dots)$ instead of the one given above. The reason is that the confluence proof below is somewhat simpler with the given transformation.

THEOREM A.2

- (1) *The mapping F from the old terms to the new terms is injective.*
- (2) *Let R be any given rewrite system and let R' denote the transformed system obtained by applying this transformation to every term in each rule. Then, $s \rightarrow_R t$ via rule $l \rightarrow r$ at occurrence o iff $s' \rightarrow_{R'} t'$ via rule $l' \rightarrow r'$ at the occurrence corresponding to o .*
- (3) *R is confluent iff R' is confluent.*

PROOF

- (1) Suppose that $s'_1 = s'_2 = s'$ (say). We do a straightforward induction on $size(s')$ to show that $s_1 = s_2$.
- (2) Follows from (1) and the fact that no extra rules are introduced in R' .
- (3) Assume R is confluent, we need to prove R' confluent (over the new signature). The problem is that there are many terms over the new signature, which have no pre-image in the old signature (in other words, the mapping is not surjective). To circumvent this problem, we use the above Theorem A.1.

From Theorem A.1, it suffices to prove confluence of every well-typed term in accordance with the following many-sorted type declarations: every constant and every variable in the original signature is assigned type 2, $c_g : 1$ for the new constants c_g corresponding to unary functions symbols g in the original signature, $c_h : 1$ for the new constants c_h corresponding to symbols of arity more than 1 in the original signature, $g' : 2 \times 1 \rightarrow 2$, $h' : 2 \times 1 \rightarrow rank(h')$, $H_i : rank(h') \times rank(H_{i+1}) \rightarrow rank(H_i)$ for all $2 \leq i \leq n - 2$, $H_1 : rank(h') \times rank(H_2) \rightarrow 2$, and $H_{n-1} : rank(h') \times rank(h') \rightarrow rank(H_{n-1})$, where the $rank(h')$ ($rank(H_i)$) is equal to $2 + \text{position of } h'$ (position of H_i) in some linear ordering of *all* the binary symbols corresponding to symbols of arity 2 or more in the original signature. The reason for shifting the positions by 2 is to ensure that the $rank$ is at least 3.

Observe that these declarations are compatible with R' since for every rewrite rule $l' \rightarrow r'$ both terms l' and r' are well-typed and have the same type 2. Also, the mapping is surjective on the well-typed terms over the new signature and the confluence proof is now straightforward. In a confluence proof we do have to consider terms of the form $H_1(c_h, c_g)$, etc., which are ill-typed.

For the other direction, assume R' is confluent. We need to prove confluence of R . Suppose $s \uparrow t$, then $s' \uparrow t'$ for well-typed terms s' and t' . By confluence of R' there exists a most-general (i.e., a proof that does not instantiate variables

unnecessarily) joinability proof for s' and t' in which all the terms are well typed, and hence we can convert this into a joinability proof of s and t via R . \square

A.1.2 $Mb \rightarrow Ib$ Transformation. The transformation consists of the following steps. We denote by t' the transformed term t . We omit the proofs since they are similar to the ones given above. There is one significant difference in the confluence proof, however, we cannot use Theorem A.1 in this case. The reason is that the type of b cannot be defined so that the mapping becomes surjective on well-typed terms. Hence, we must resort to “ad-hoc reasoning” using case analysis (based on whether one or both terms in a rewrite step using R' over the new signature have a pre-image in the original signature) and the idea of most general proofs as above.

- (1) Each variable and each constant is mapped to itself.
- (2) We introduce a new universal binary symbol b .
- (3) For all binary symbols f in the old representation we introduce a new constant c_f . The term $f(s, t)$ is mapped to the term $b(s', b(t', c_f))$.

THEOREM A.3

- (1) *The mapping F from the old terms to the new terms is injective.*
- (2) *Let R be any given rewrite system and let R' denote the transformed system obtained by applying this transformation to every term in each rule. Then, $s \rightarrow_R t$ via rule $l \rightarrow r$ at occurrence o iff $s' \rightarrow_{R'} t'$ via rule $l' \rightarrow r'$ at the occurrence corresponding to o .*
- (3) *R is confluent iff R' is confluent.*

A.1.3 $MuMc \rightarrow Mu1c$ Transformation. The transformation consists of the following steps. We denote by t' the transformed term t .

- (1) Each variable is mapped to itself.
- (2) Introduce a new constant c .
- (3) For each constant d of the old system, we introduce a new unary symbol d' to the new system and the constant is encoded as $d'(c)$.
- (4) The term $f_1(f_2(\dots f_n(a)\dots))$ of the old system is translated to the term $f_1(f_2(\dots f_n(a'(c))\dots))$ in the new system.

THEOREM A.4

- (1) *The mapping F from the old terms to the new terms is injective.*
- (2) *Let R be any given rewrite system and let R' denote the transformed system obtained by applying this transformation to every term in each rule. Then, $s \rightarrow_R t$ via rule $l \rightarrow r$ at occurrence o iff $s' \rightarrow_{R'} t'$ via rule $l' \rightarrow r'$ at the occurrence corresponding to o .*
- (3) *R is confluent iff R' is confluent.*

The proofs are omitted since they are similar to the proofs of the first transformation above. For the confluence proof, we may use the many-sorted declarations: $c : 1$, where c is the new constant, $d' : 1 \rightarrow 2$ for each new unary symbol

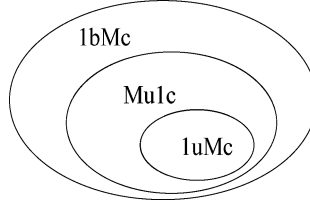


Fig. 1. The hierarchy.

d' , every variable is of type 2 and $f : 2 \rightarrow 2$, for every unary f in the original signature. It is easy to see that every rule $l' \rightarrow r' \in R'$ is well-typed, both l' and r' are of type 2.

A.2 Proofs

THEOREM A.5. *For a standard system R^* containing only unary symbols and one constant, if R^* is not confluent, then there is a witness (u, v) , such that $size(v) \leq size(u) \leq b$, where $b = k(M^2 + 1) + M + N + 1$, M is the number of rules in R^* , and N is the size of the largest term in R^* .*

PROOF. Let us assume that the system R^* is nonconfluent. We arrange all the witnesses according to the following rules.

- (1) The pair (A, B) would be considered smaller than the pair (A', B') if the larger of A and B is smaller than the larger of A' and B' .
- (2) If the largest elements have equal lengths, then the pair with smaller sum of sizes would be considered smaller (this is the same as having small size for the smaller term in the pair).

Then, we pick the smallest witness according to this ordering, claiming that both of the terms in this witness will be of size less than or equal to the above bound. Let us assume this is not the case. We denote the witness by (A, B) , where A is the larger term. Let $l \in \text{subterm}(R^*)$ be such that $l \xrightarrow{*} A$ and $l \xrightarrow{*} B$. The following observations apply to the chosen witness:

(1) From our ordering it follows that we should never be able to decrease the sizes of A and B , because a decrease in A would lead a smaller witness according to the first rule (in case A and B have same size, we would have to use second rule). And a decrease in B would lead to a smaller witness according to the second rule.

(2) From the first observation, it follows that we would never be able to apply a rule beyond the last N elements in each of our 2 terms, since otherwise we would have a decrease in the size of the term. Thus, beyond the last N elements, nothing will change in both A and B .

There are 2 cases.

(1) $size(A), size(B) > b$. Then there are two subcases.

(a) A and B differ in the first $M^2 + 1$ positions. In this case, in the second $M^2 + 1$ positions, we would have a rule pair repetition. Delete the rule applications (pumping lemma) in between (the way it is done is exactly the same as that in $luMc$ case). Since we could never apply a rule beyond the last N elements

in both terms, we would have that this new pair is also a witness (they are reachable from l and they differ in first $M^2 + 1$ positions).

(b) A and B have the same prefix of length $M^2 + 1$. In this case, in the first $M^2 + 1$ position we would have a rule pair repetition, thus deleting that part of the rule applications (pumping lemma) would not change the witness property of the pair (they still would be nonjoinable and reachable from l).

(2) $size(A) > b$ and $size(B) \leq b$. Then, A can be decomposed into the following parts $A = XYZ$ where Z is the last N symbols, Y is the next $M + 1$ symbols, and X is the prefix containing the remaining, at least $k(M^2 + 1)$, symbols. Then, among Y we have a rule repetition (since it has size 1 more than the number of rules). Thus, deleting this part, that is, the rule applications in between the rule repetition, we would obtain a new term $A' = XY'Z$, which is joinable with B (otherwise, we would have a smaller witness). But since A' ends with the same N symbols as A does, and there was no possible decrease in A , then there would be no possible decrease in A' as well, meaning that in order for B to join A' , it has to reach a term $B' = XY'Z'$, where Z' is in $Subterm(R^*)$ (this can be assured by picking the most immediate Z' , that is, the first term in the standard reduction sequence from l to B' that contains the prefix XY') and Z and Z' are joinable. Let us assume that $Z \rightarrow L$ and $Z' \rightarrow L$ for the smallest size of L (which is at most $k(R^*)$ by definition of $k(R^*)$). Note, that since A could not be decreased in size, $size(L) \geq size(Z) = N$ and L cannot be decreased either. Then the terms A and $B'' = XY'L$ would not be joinable, since they are also reachable from l and if they were joinable so would A, B . Moreover, no decrease in size can occur in A , and B'' can not be changed beyond L (i.e., X cannot be changed).

Let us observe that the size of B'' is not more than $k(R^*) + Size(A) - 1$ because $Size(L)$ is less than $k(R^*)$.

Now, since the first $k(M^2 + 1)$ or more elements of both A and B'' are the same, and we can never reach these parts, we would have a rule pair repeated here for at least $(k(R^*) + 1)$ times. Thus deleting everything in between the first application of this pair and the last in A, B'' , we would acquire a witness, with both terms of sizes less than $size(A)$. This is a contradiction. \square

THEOREM A.6. *For a standard system R^* containing only 1 binary symbol and constants, if R^* is not confluent, then there is a witness (u, v) , such that $ht(v) \leq ht(u) \leq ((M^2 + 1) + (N + 1)) + (M + 1) + 2(N + 1)$, where M is the number of rules in R^* , and N is the height of the highest term in R^* .*

PROOF. Let us assume that the system R^* is nonconfluent. We arrange all the witnesses (reachable from a left-hand side) in accordance with the following rules.

- (1) Witnesses with smaller height difference come before those with larger height difference.
- (2) Witnesses with equal height difference are arranged in accordance with the sum of the sizes with smaller sums first.

The rest of the proof is omitted since it very similar to the proof of Theorem 6.1 and the previous proof. \square

REFERENCES

- CHURCH, A. AND ROSSER, J. 1936. Some properties of conversion. *Trans. AMS* 39, 472–482.
- COMON, H., DAUCHET, M., GILLERON, R., JACQUEMARD, F., LUGIEZ, D., TISON, S., AND TOMASSI, M. 1999. *Tree Automata Techniques and Applications*. <http://www.grappa.univ-lille3.fr/tata/>.
- COMON, H., GODOY, G., AND NIEUWENHUIS, R. 2001. Confluence of ground rewrite systems is in P. In *Proceedings of the IEEE Symposium on Foundations of Computer Science*. IEEE Computer Society Press, Los Alamitos, Calif.
- DAUCHET, M., HEUILLARD, T., LESCANNE, P., AND TISON, S. 1990. Decidability of the confluence of finite ground term rewrite systems. *Inf. Comput.* 88, 187–201 (Also in *Proc. IEEE Symp. on LICS* 1987).
- DAUCHET, M. AND TISON, S. 1990. The theory of ground rewrite systems is decidable. In *Proceedings of the IEEE Conference on Logic in Computer Science*. IEEE Computer Society Press, Los Alamitos, Calif. 242–248.
- DERSHOWITZ, N. AND PLAISTED, D. 2001. Rewriting. In *Handbook of Automated Reasoning*, J. A. Robinson and A. Voronkov, Eds. Vol. 1. Elsevier Science, Chapter 9, 535–610.
- KLOP, J. 1992. Rewrite systems. In *Handbook of Logic in Computer Science*. Oxford.
- OYAMAGUCHI, M. 1987. The church rosser property for ground term rewriting systems is decidable. *Theoret. Comput. Sci.* 49, 43–79.
- PLAISTED, D. 1993. Polynomial time termination and constraint satisfaction tests. In *Proceedings of the Conference on Rewriting Techniques & Applications*.
- SEIDL, H. 1990. Deciding equivalence of finite tree automata. *SIAM J. Comput.* 19, 424–437.
- TIWARI, A. 2002. Polynomial time algorithms for deciding confluence of certain classes of rewrite systems. In *Proceedings of the IEEE Conference on Logic in Computer Science*. IEEE Computer Society Press, Los Alamitos, Calif.
- VERMA, R. M. 2002a. Algorithms and reductions for rewriting problems. II. *Info. Proc. Lett.* 84, 4, 227–233.
- VERMA, R. M. 2002b. On the complexity of normal form problems for certain rewrite systems. Tech. rep., University of Houston.
- ZANTEMA, H. 1994. Termination of term rewriting: Interpretation and type elimination. *J. Symb. Comput.* 17, 23–50.

Received August 2002; revised June 2003; accepted June 2003