Sequentiality, Second Order Monadic Logic and Tree Automata

H. Comon
CNRS and LRI
Bat. 490, Univ. Paris-Sud
91405 ORSAY cedex, France
E-mail: comon@lri.fr

Abstract

Given a term rewriting system R and a normalizable term t, a redex is needed *if in any reduction sequence of t to a normal form, this redex will be contracted. Roughly, R is* sequential *if there is an optimal reduction strategy in which only needed redexes are contracted. More generally, G. Huet and J.-J. Lévy define in [8] the sequentiality of a predicate P on partially evaluated terms. We show here that the sequentiality of P is definable in SkS, the second-order monadic logic with k successors, provided P is definable in SkS. We derive several known and new consequences of this remark: 1–*strong sequentiality*, as defined in [8], of a left linear (possibly overlapping) rewrite system is decidable, 2–* NV-sequentiality*, as defined in [15] is decidable, even in the case of overlapping rewrite systems 3–* sequentiality *of any linear shallow rewrite system is decidable.*

Then we describe a direct construction of a tree automaton recognizing the set of terms that do have needed redexes, which, again, yields immediate consequences: 1– Strong sequentiality of possibly overlapping linear rewrite systems is decidable in EXPTIME, 2– For strongly sequential rewrite systems, needed redexes can be read directly on the automaton.

1 Introduction

Besides confluence, there are two important issues concerning non-terminating computations in term rewriting theory. One is to find a *normalizing reduction strategy*, which has been investigated in, e.g., [14, 10, 1]. The other is to find an *optimal reduction strategy*, for which only *needed redexes* are contracted. This question was first investigated by Huet and Lévy in 1978 [8]. They call *sequential* a rewrite system for which there exists such an optimal strategy. We focus here on the latter issue.

A typical example is the "parallel or", whose definition contains the two rules $\top \vee x \rightarrow \top$ and $x \vee \top \rightarrow \top$. Given an expression $e_1 \vee e_2$, which of e_1 and e_2 should be evaluated first? If e_1 is tried first, its evaluation may be unnecessary because e_2 evaluates to $\top$, and the whole expression can be reduced to $\top$. Hence, this strategy is not optimal. Evaluating e_2 first is not optimal either: there is no optimal (sequential) reduction strategy for the "parallel or".

Given a term rewriting system R, can we decide whether R is *sequential* ? In case it is, is it possible to compute (and compile) an optimal strategy? These questions have been addressed in several papers, starting with [8]. Unfortunately, the sequentiality of R is in general undecidable. In their landmark paper, Huet and Lévy introduce a sufficient criterion: *strong sequentiality*, and show that this property is decidable for *orthogonal term rewriting systems*, in which left hand sides do not overlap nor contain repeated occurrences of a same variable. The original proof is quite intricate. J.-W. Klop and A. Middeldorp [13] give a simpler proof to the price of an increased complexity. The case of linear, possibly overlapping rewrite systems was considered first by Toyama [22] and later shown decidable by Jouannaud and Sadfi [9]. M. Oyamaguchi defines *NV-sequentiality* a property intermediate between sequentiality and strong sequentiality, which is also decidable for orthogonal rewrite systems [15].

In this paper we use another quite simple approach, though less elementary: we show that the sequentiality of P is definable in SkS (resp. WSkS), the second-order monadic logic with k successors, provided that P is definable in SkS (resp. WSkS). It allows to easily derive all aforementioned decidability results. Relying on automata theory, the decidability of strong sequentiality (resp. NV-sequentiality) of possibly overlapping left linear rewrite systems becomes straightforward. This sheds new light on which properties of rewrite systems are indeed necessary in proving (NV-, strong-) sequentiality. Then, it becomes possible to derive new decidability results, for example NV-sequentiality for overlapping left linear rewrite systems or sequentiality of shallow linear rewrite systems. We may also add a *sort discipline* to the rewrite systems without loosing decidability.

This method has however several drawbacks: first, the complexity of SkS is non-ELEMENTARY, which is far too complex in general for "effective" methods. Second, for non-left linear rewrite system, the reducibility predicate is not expressible in SkS. Hence we cannot derive that strong sequentiality of R is expressible in SkS in such a case. Last, but not least, even if we know that the formula expressing the sequentiality of R is valid, how do we effectively find needed redexes in a term?

In order to answer these questions, we construct directly a tree automaton which accepts all terms that have a needed redex. By the well-known correspondence between WSkS and finite tree automata (see e.g. the survey [21]), we know in advance that such an automaton exists. Here, we show that it can be constructed in exponential time (for k fixed). This has several consequences. First, deciding strong sequentiality of any left linear rewrite system is in EXPTIME, since it reduces to an emptiness problem for tree automata, which can be decided in polynomial time. Then, the automaton which accepts all terms that have a needed redex yields directly the algorithm for searching needed redexes in a term.

There are still many issues to be investigated with profit in this framework: is strong sequentiality decidable for any (possibly non-linear) rewrite systems? Though automata with constraints [2, 3] cannot be used directly, we might consider some tree automata inspired by these definitions. What is the exact complexity of all decision questions in this area? We have only shown and EXPTIME inclusion. However there is no evidence that this is the best we can do. Also, what happens in the case of orthogonal rewrite systems? The automata should have a particular form, from which it might be possible to deduce more efficient procedures. Finally, we do not show how to *compile* an optimal reduction strategy, avoiding any backtrack in the input term, as done in [8]. Again, this should be possible from the tree automaton. Finally, other (sequential) reduction strategies as in [1] should also be investigated within this framework.

The paper is organized as follows: section 2 gives the definitions of an index and sequentiality and we recall the necessary background on SkS and tree automata. In section 3, we show how to express the sequentiality of a predicate P in SkS and apply this result to rewrite systems in section 4. In section 5 we construct directly the automaton accepting all terms that have an index (using the characterization of [13]) and derive extensions as well as complexity results. We also explain how an index search can be read on the automaton. Most of the proofs are included here. This should be a witness of simplicity.

2 Basic Definitions

T is the set of terms built over a fixed alphabet $\mathcal{F}$ of function symbols. Each $f \in \mathcal{F}$ comes with its *arity* $a(f)$, a nonnegative integer. Terms may also be viewed as labeled trees, i.e. mappings from a finite prefix-closed subset of words of positive integers (the *positions* in the tree) into $\mathcal{F}$, in such a way that the successors of a position p are exactly the strings $p \cdot i$ for $1 \leq i \leq a(f)$ when p is labeled with f. We will use the notations of [6]: $t|_p$ is the *subterm* at position p, $t[u]_p$ is the term obtained by replacing $t|_p$ with u. $\mathcal{F}$ is assumed to be finite. [1]

$\mathcal{T}_\Omega$ is the set of terms obtained by augmenting the set $\mathcal{F}$ of function symbols with a new constant Ω (which stands intuitively for "unevaluated terms"). We assume that terms in $\mathcal{T}_\Omega$ always contain at least one occurrence of Ω. Such a set of terms is classically considered as the set of terms which are partially evaluated, i.e. terms in T which are "cut" on some branches.

[1] Note that if one wants to consider terms with possibly infinitely many "variables" (actually constants, but we use the standard terminology), it is always possible to represent the variables $x_0, x_1, \ldots, x_n, \ldots$ using an additional constant x and an additional unary function symbol s; they will be respectively represented by $x, s(x), \ldots, s(s(\ldots(s(x))\ldots)), \ldots$. In such a case, T is a regular subset of the set of all terms, but this does not cause any additional problem, as we will see later in the general case of typed terms. The status of variables in T (or $\mathcal{T}_\Omega$) is different from the status of the variables in the rewrite system: the former are actually considered as constants along the evaluation process, while the latter may be instanciated since several distinct instances of the rules can be used.

Definition 2.1 *Let $t, u \in \mathcal{T}_\Omega$, $t \sqsubseteq u$ iff u can be obtained from t by replacing some occurrences of Ω in t with terms in $\mathcal{T}_\Omega$.*

$s \sqsubseteq t$ intuitively means that "t is more evaluated than s".

Definition 2.2 (index,[8]) *Let P be a predicate on $\mathcal{T}_\Omega \cup T$. Let $t \in \mathcal{T}_\Omega$ and $p \in Pos(t)$. p is an* index *of P in t iff $t|_p = \Omega$ and*

$$\forall u \in \mathcal{T}_\Omega \cup T, \ (t \sqsubseteq u \wedge P(u) = \mathit{true}) \Rightarrow u|_p \neq \Omega$$

The set of indexes of a term $t \in \mathcal{T}_\Omega$ (which were also called *needed redexes* in the introduction) is written $\mathrm{Index}(t)$. Intuitively, p is an index for P in t if, for all successful evaluations of t (the predicate P becomes true), the term at position p has been evaluated.

Definition 2.3 (sequentiality,[8]) *A predicate P on $\mathcal{T}_\Omega \cup T$ is* sequential *if*

$$\forall t \in \mathcal{T}_\Omega \cup T, (\exists u \in \mathcal{T}_\Omega \cup T, P(u) = \mathit{true} \wedge t \sqsubseteq u) \\ \Rightarrow (P(t) = \mathit{true} \vee \exists p \in Pos(t), p \in \mathit{Index}(t))$$

Intuitively, P is sequential if, for every partially evaluated term t such that P is false and P becomes true for some further evaluation of t, then there is an index of P in t.

$\mathcal{X}$ is an infinite set of constant function symbols called *variables* and the set of terms build on $\mathcal{F} \cup \mathcal{X}$ is traditionally written $\mathcal{T}(\mathcal{F}, \mathcal{X})$. For any $s \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, $\mathrm{Var}(s)$ is the set of variables occurring in s. Substitutions are mappings from $\mathcal{X}$ into $\mathcal{T}(\mathcal{F}, \mathcal{X})$, which are extended into endomorphisms of $\mathcal{T}(\mathcal{F}, \mathcal{X})$. We use the postfix notation for substitution applications.

A *term rewriting system* is a (finite) set of pairs of terms in $\mathcal{T}(\mathcal{F}, \mathcal{X})$, each pair (s, t) is written $s \rightarrow t$ (we do not require $\mathrm{Var}(t) \subseteq \mathrm{Var}(s)$). A term t *rewrites to* s through a rewrite system R, which is written $t \xrightarrow[R]{} s$ if there is a position p in t, a substitution σ and a rule $l \rightarrow r \in R$ such that $t|_p = l\sigma$ and $s = t[r\sigma]_p$.

2.1 The logic (W)SkS and tree automata

Missing definitions can be found in [21]. *Terms* of SkS are formed out of individual variables $(x, y, z, \ldots)$, the empty string Λ and right concatenation with $1, \ldots, k$. Atomic formulas are equations between terms, inequations $w < w'$ between terms or expressions "$w \in X$" where w is a term and X is a (second-order) variable. Formulas are built from atomic formulas using the logical connectives $\wedge, \vee, \Rightarrow, \neg, \ldots$ and the quantifiers $\exists, \forall$ of both individual and second-order variables. Individual variables are interpreted as elements of $\{1, \ldots, k\}^*$ and second-order variables as subsets of $\{1, \ldots, k\}^*$. Equality is the string equality and inequality is the strict prefix ordering. In the *weak* second-order monadic logic WSkS, second-order variables only range over *finite* sets. Finite union an finite interesection, as well as inclusion and equality of sets are definable in (W)SkS in an obvious way. Hence we may use these additional connectives in the following.

The most remarkable result is the decidability of SkS (a result due to M. O. Rabin, see e.g. [17, 21] for comprehensive surveys). The main idea of the proof is to associate each formula ϕ whose free variables are $X_1, \ldots, X_n$ with a (Rabin) tree automaton which accepts the set of n-uples of trees (or sets of strings) that satisfy the formula. Then decidability follows from closure and decidability properties of the corresponding class of tree languages. We only use here the weak case, in which only finite state tree automata are used.

A *finite (bottom-up) tree automaton* consists of a ranked alphabet $\mathcal{F}$, a finite set of states Q, a subset Q_f of final states and a set of transition rules of the form $f(q_1, \ldots, q_n) \rightarrow q$ where $f \in \mathcal{F}$, $n = a(f)$ and $q_1, \ldots, q_n, q \in Q$ or $q \rightarrow q'$ where $q, q' \in Q$ (the latter transition rules are called *ϵ-transitions*). A tree automaton *accepts* t if t can be rewritten to a final state using the transition rules (see e.g. [7] for more details). A *run* of the automaton on a tree t is a mapping ρ from the positions of t into Q such that $\rho(p) = q, \rho(p \cdot 1) = q_1, \ldots \rho(p \cdot n) = q_n$ and $t(p) = f$ only if there is a transition rule $f(q_1, \ldots, q_n) \rightarrow q'$ and a sequence of ϵ-transitions from q' to q. A run ρ is *successful* if $\rho(\Lambda)$ is a final state. t is accepted by A iff there is a successful run of A on t.

3 Relationship between Sequentiality and Recognizability

Let k be the maximal arity of a function symbol in $\mathcal{F}$ and n be the cardinal of $\mathcal{F}$. A term t is represented in WSkS using $n + 2$ set variables X, X_Ω and $X_f, f \in \mathcal{F}$ (which will be written $\vec{X}$ in the following). X will be the set of positions of t and X_Ω and each X_f will be the sets of positions that are labeled with the corresponding function symbol. We express in WSkS that some $n + 2$-uple of finite sets of words are indeed encoding a term, which can be achieved using the formula

$$\mathrm{Term}(\vec{X}) \stackrel{\mathrm{def}}{=}$$
$$X = X_\Omega \cup \bigcup_{i=1}^{n} X_{f_i}$$
$$\wedge \bigwedge_{i \neq j} (X_{f_i} \cap X_{f_j} = \emptyset \wedge X_\Omega \cap X_{f_i} = \emptyset)$$
$$\wedge\ \forall x \in X, \forall y < x, y \in X$$
$$\wedge \bigwedge_{f \in \mathcal{F} \cup \{\Omega\}} \forall x \in X_f, \bigwedge_{l=1}^{a(f)} x \cdot l \in X \bigwedge_{l=a(f)+1}^{k} x \cdot l \notin X$$

In this setting, it is quite easy to express the sequentiality of P in (W)SkS as shown by the following lemmas.

Lemma 3.1 $\sqsubseteq$ *is definable in WSkS.*

Proof: Assume that t, u are represented by $\vec{X}$ and $\vec{Y}$ respectively. Then $t \sqsubseteq u$ iff

$$X \subseteq Y \wedge \bigwedge_{f \in \mathcal{F}, f \neq \Omega} X_f \subseteq Y_f \qquad \square$$

Lemma 3.2 *Let P be a predicate on $\mathcal{T}_\Omega \cup T$ which is definable in (W)SkS. Then the set of terms in $\mathcal{T}_\Omega$ which have an index w.r.t. P is definable in (W)SkS.*

Proof: Let $\phi(\vec{X})$ be the definition of P in (W)SkS. Then the set of terms which have an index is defined by translating definition 2.2:

$$\mathrm{Index}(\vec{X}) \stackrel{\mathrm{def}}{=} \mathrm{Term}(\vec{X}) \wedge \exists x \in X.\ x \in X_\Omega \wedge \forall \vec{Y}.$$
$$(\mathrm{Term}(\vec{Y}) \wedge \vec{X} \sqsubseteq \vec{Y} \wedge \phi(\vec{Y})) \Rightarrow x \in Y_\Omega \qquad \square$$

Theorem 3.3 *If P is definable in (W)SkS, then the sequentiality of P is decidable.*

Proof: Using the previous lemma and assuming that P is defined by $\phi(\vec{X})$, P is sequential iff the following formula holds:

$$\forall \vec{X}, (\mathrm{Term}(\vec{X}) \wedge \exists \vec{Y}, \mathrm{Term}(\vec{Y}) \wedge \phi(\vec{Y}) \wedge \vec{X} \sqsubseteq \vec{Y})$$
$$\Rightarrow (\phi(\vec{X}) \vee \mathrm{Index}(\vec{X}))$$

which is a translation of definition 2.3. Then we conclude using Rabin's theorem [16, 17]. $\square$

4 Application to term rewriting systems

In this section, we show how to apply theorem 3.3 to various sequentiality results for term rewriting. We assume here the reader familiar with term rewriting (see e.g. [6] for missing definitions). We will say in particular that a term t is *linear* if each variable occurs at most once in t (A term is *shallow* if it is a variable or if all its variables occur at depth 1. A rewrite system R is *left linear* (resp. *linear*, resp. *shallow*) if all its left hand sides are linear (resp. all left and right hand sides are linear, resp. all left and right hand sides are shallow).

4.1 Strong sequentiality of left linear term rewriting systems

Let N be the predicate symbol on $\mathcal{T}_\Omega \cup T$ which holds true iff t has a normal form belonging to T.

Definition 4.1 ([8]) *A term rewriting system is* sequential *if the predicate N is sequential.*

This captures the intuitive notion sketched in introduction: when R is sequential, then there is an optimal reduction strategy. Since sequentiality of R is undecidable in general, a sufficient condition for sequentiality (called strong sequentiality) has been introduced in [8].

Let Red be the predicate symbol on $\mathcal{T}_\Omega \cup T$ which holds true iff t is reducible.

Definition 4.2 ([8]) *A term rewriting system is* strongly sequential *if the predicate Red is sequential.*

Lemma 4.3 *When R is left linear, Red is recognized by a finite bottom-up tree automaton.*

Proof: For each non-variable strict subterm u of a left hand side of a rule, consider a state q_u. In addition, we have a state q_r (the final state, or the state in which we know that the term is reducible) and the state $q_\top$ which accepts all terms. Then, to each $u = f(u_1, \ldots, u_n)$, we associate the production rule $f(q_{u_1}, \ldots, q_{u_n}) \to q_u$ where q_{u_i} is understood as $q_\top$ when u_i is a variable. To each left hand side of a rule $l = f(t_1, \ldots, t_n)$, we associate the rule $f(q_{t_1}, \ldots, q_{t_n}) \to q_r$ and the states q_r are propagated: we have the rules $f(q_\top, \ldots, q_\top, q_r, q_\top, \ldots, q_\top) \to q_r$ for all function symbols f. Finally, if not already present, we add the rules $f(q_\top, \ldots, q_\top) \to q_\top$. $\square$

This yields as an easy consequence the following (known from [9]) decidability result:

Corollary 4.4 *The strong sequentiality of left linear (possibly overlapping) rewrite systems is decidable.*

Proof: This is a consequence of lemma 4.3 and theorem 3.3 since recognizable sets of terms are definable in WSkS [20]. □

Note that this doesn't work for non-left linear rewrite systems because then Red is not definable by a finite bottom-up tree automaton: we need disequality tests. Actually, adding the corresponding tests to the logic (W)SkS yields an undecidable logic.

Another consequence of lemma 4.3 is the recognizability of the set of irreducible terms in $T(\mathcal{F})$: this is a consequence of the closure property of recognizable tree languages by complement. Let us show however an explicit construction of such an automaton, since we will re-use it for further analysis in the following.

Given two terms $s, t \in T(\mathcal{F}, \mathcal{X})$, we write $s \downarrow t$ for a most general unifier (when it exists) of s and a renaming t' of t such that t' and s do not share variables. Given a left linear rewrite system $\mathcal{R}$, let $S(\mathcal{R})$ be the set of strict subterms of the left hand sides of $\mathcal{R}$, up to similarity, which we close under $\downarrow$. (This may yield an exponential number of states: one for each set of subterms of the lhs of $\mathcal{R}$). With each term t in $S(\mathcal{R})$, which is not an instance of a lhs of $\mathcal{R}$, we associate a state q_t. (we write q_x the state associated with all variables. We assume that q_x is in the set of states). Let $Q = \{q_t \mid t \in S(\mathcal{R})\} \cup \{q_r\}$ and Q_f be all states but q_r. Intuitively, all reducible terms will be accepted in q_r. The terms accepted in a state q_t will be all instances of t that are not instances of any $q_{t\sigma}$. More precisely, we consider the following set of production rules:

$$
\begin{array}{lll}
f(q_{t_1}, \dots, q_{t_n}) & \rightarrow & q_t \\
 & & \text{If } f(t_1, \dots, t_n) \text{ is an instance of } t \\
 & & \text{and not an instance of some } t\sigma \\
f(q_{t_1}, \dots, q_{t_n}) & \rightarrow & q_r \\
 & & \text{If } f(t_1, \dots, t_n) \text{ is an instance} \\
 & & \text{of some lhs of } R \\
f(q_1, \dots, q_n) & \rightarrow & q_r \\
 & & \text{If } q_r \in \{q_1, \dots, q_n\}
\end{array}
$$

Let us call $\mathcal{A}_{NF(\mathcal{R})}$ the above constructed automaton.

Lemma 4.5 *$\mathcal{A}_{NF(\mathcal{R})}$ accepts the set of irreducible terms in $T(\mathcal{F})$. This automaton is deterministic and completely specified.*

Example 4.6

$$
R = \left\{
\begin{array}{rcl}
h(x) & \rightarrow & g(f(a,a)) \\
f(x,a) & \rightarrow & a \\
f(g(a),x) & \rightarrow & f(x,g(a))
\end{array}
\right.
$$

The automaton $\mathcal{A}_{NF(R)}$ consists of the production rules (assuming for simplicity that there are no additional function symbols besides h, f, g, a)

$$
\begin{array}{rclrcl}
a & \rightarrow & q_a & g(q_a) & \rightarrow & q_{g(a)} \\
f(q_a, q_a) & \rightarrow & q_r & h(q_a) & \rightarrow & q_r \\
f(q_a, q_{g(a)}) & \rightarrow & q_x & f(q_{g(a)}, q_a) & \rightarrow & q_r \\
f(q_{g(a)}, q_{g(a)}) & \rightarrow & q_r & h(q_{g(a)}) & \rightarrow & q_r \\
g(q_{g(a)}) & \rightarrow & q_x & f(q_x, q_a) & \rightarrow & q_r \\
f(q_x, q_{g(a)}) & \rightarrow & q_x & f(q_x, q_x) & \rightarrow & q_x \\
f(q_a, q_x) & \rightarrow & q_x & f(q_{g(a)}, q_x) & \rightarrow & q_r \\
h(q_x) & \rightarrow & q_r & g(q_x) & \rightarrow & q_x \\
f(q_r, q) & \rightarrow & q_r & f(q, q_r) & \rightarrow & q_r \\
g(q_r) & \rightarrow & q_r & h(q_r) & \rightarrow & q_r
\end{array}
$$

where q stands for any state. Final states will be $q_a, q_{g(a)}, q_x$.

These constructions can be simplified for a particular class of rewrite system:

Lemma 4.7 *Assume that for any two strict subterms s, t of some left hand side(s) of R, if s and t are unifiable, then either s is an instance of t or t is an instance of s. Then Red and $NF(R)$ are accepted by deterministic bottom-up tree automata with $O(|R|)$ states.*

This is a direct consequence of the construction of $\mathcal{A}_{NF(R)}$: the set of states is $O(|R|)$ in this case.

4.2 Sequentiality of shallow linear rewrite systems

Lemma 4.8 *If R is a shallow linear rewrite system, then N is recognized by a finite tree automaton.*

Proof:(sketch) We start with the automaton $\mathcal{A}_{NF(\mathcal{R})}$ which accepts the set of irreducible terms in $T(\mathcal{F})$ (thanks to lemma 4.5). Then we complete it in such a way that for every ground term t occurring in a right side of R there is a state q_t which accepts t (and only this term), and a state $q_\top$ which accepts all terms in $\mathcal{T}_\Omega \cup T$. (This differs from q_x in which all terms of T are accepted).

Then, we build an automaton $\mathcal{A}_1$ which accepts all terms in $T \cup \mathcal{T}_\Omega$ that have a normal form in T. For, we use the same set of states as $\mathcal{A}$ (and the same set of final states). We start with the set P of production rules of $\mathcal{A}$ and saturate P using the following set of inference rules:

$$\frac{f(t_1,\ldots,t_n) \underset{R}{\rightarrow} g(u_1,\ldots,u_m) \quad g(q_1,\ldots,q_m) \underset{P}{\rightarrow} q}{f(q'_1,\ldots,q'_n) \underset{P}{\rightarrow} q}$$

If

- when u_i is a ground term, then u_i is accepted in state q_i (by the current automaton)
- $q'_i = q_j$ whenever $t_i = u_j$
- $q'_i = q_{t_i}$ when $t_i \in T$
- $q'_i = q_\top$ when t_i is a variable not occurring in the right side of the rule

and
$$\frac{f(t_1,\ldots,t_n) \underset{R}{\rightarrow} x \quad q \in Q}{f(q_1,\ldots,q_n) \underset{P}{\rightarrow} q}$$

If

- x is a variable
- $q_i = q$ whenever $t_i = x$
- $q_i = q_{t_i}$ whenever t_i is a ground term.
- $q_i = q_\top$ when t_i is a variable distinct from x.

This terminates since the set of states being fixed, the number of possible inferred production rules is bounded. It yields a finite bottom-up tree automaton accepting all terms in $T \cup \mathcal{T}_\Omega$ that have a normal form in T. $\square$

Let us show an example of the automaton accepting all terms that can be reduced to a normal form.

Example 4.9 We use the example 4.6. In what follows, for sake of simplicity, we will not consider the rules which involve a state q_r (they do not play any role).

We add first the rules:

$$\begin{array}{rcl} f(q_a,q_a) & \rightarrow & q_{f(a,a)} \\ a & \rightarrow & q_\top \\ h(q_\top) & \rightarrow & q_\top \\ \Omega & \rightarrow & q_\top \end{array} \qquad \begin{array}{rcl} g(q_{f(a,a)}) & \rightarrow & q_{g(f(a,a))} \\ g(q_\top) & \rightarrow & q_\top \\ f(q_\top,q_\top) & \rightarrow & q_\top \end{array}$$

Now, we start the saturation process, yielding to the new rules (in order of computation, and excluding the rules yielding $q_\top$ which are irrelevant):

$$\begin{array}{rcl} h(q_\top) & \rightarrow & q_{g(f(a,a))} \\ f(q_{g(a)},q_x) & \rightarrow & q_x \\ h(q_\top) & \rightarrow & q_{g(a)} \end{array} \qquad \begin{array}{rcl} f(q_\top,q_a) & \rightarrow & q_a \\ f(q_{g(a)},q_a) & \rightarrow & q_x \end{array}$$

This last rule is obtained after noticing that $f(a,a) \xrightarrow{*} q_a$, hence from $h(x) \rightarrow g(f(a,a)) \in R$ and $g(q_a) \rightarrow q_{g(a)}$, we can deduce $h(q_\top) \rightarrow q_{g(a)}$.

Let us consider two examples of computations using the resulting automaton:

$$h(\Omega) \rightarrow h(q_\top) \rightarrow q_{g(a)}$$

hence $h(\Omega)$ is accepted by the automaton.

$$\begin{array}{rcl} f(g(a),\Omega) & \rightarrow & f(g(q_a),\Omega) \\ & \rightarrow & f(g(q_a),q_\top) \\ & \rightarrow & f(q_{g(a)},q_\top) \end{array}$$

the reduction cannot be continued any longer (except by going to $q_\top$). Moreover, there is no other computation sequence: $f(g(a),\Omega)$ is not accepted.

As a consequence of theorem 3.3 we have the new decidability result:

Corollary 4.10 *The sequentiality of shallow linear rewrite systems is decidable.*

4.3 NV-sequentiality of linear rewrite systems

Oyamaguchi has shown in [15] that sequentiality is decidable for linear rewrite systems such that the left and right hand sides do not share variables. A linear rewrite system R is then called *NV-sequential* if, when renaming the variables of the right hand sides in such a way that they do not share any more variables with the left hand sides, then we get a sequential rewrite system R_V. It turns out that NV-sequentiality is again definable in WSkS (without the orthogonality assumption of [15]).

Lemma 4.11 *For any linear rewrite system R, N_{R_V} is definable in WSkS.*

Proof:(sketch) We only have to prove that N_{R_V} is recognized by a finite tree automaton. We start with the automaton of lemma 4.5 and add a state q_t for each subterm t of a right hand side of R_V, up to literal similarity. (If this state is not already present). We also add production rules in such a way that the set of terms accepted in such a state q_t is the set of instances (in T) of t. In addition, we consider states q'_t, for each strict subterm t of a left hand side of R_V, up to litteral similarity. We add the production rules in such a way that the terms accepted in state q'_t are the instances in $\mathcal{T}_\Omega \cup T$ of t. Let us call again $\mathcal{A}$ the resulting automaton; its final states are the final states of $\mathcal{A}_{NF(R_V)}$. Of course $\mathcal{A}$ is now non-deterministic.

We saturate $\mathcal{A}$ with the following inference rules:

$$\frac{g(l_1,\ldots,l_n) \xrightarrow[R_V]{} f(r_1,\ldots,r_m) \quad f(q_1^*,\ldots,q_m^*) \xrightarrow[P]{} q^*}{g(q'_{l_1},\ldots,q'_{l_n}) \xrightarrow[P]{} q^*}$$

If, for every i there is an instance of r_i which is accepted in state q_i^* and q_i^* is either q_i or q'_i (Note that this condition is decidable as the set of instances of r_i is accepted by a finite tree automaton and by decision properties for tree automata).

$$\frac{g(l_1,\ldots,l_n) \xrightarrow[R_V]{} x \quad q^* \in Q}{g(q'_{l_1},\ldots,q'_{l_n}) \xrightarrow[P]{} q^*}$$

If x is a variable and q^* is either q or q'.

The saturation process does terminate since no new state is added. the resulting automaton accepts the terms of $T \cup \mathcal{T}_\Omega$ that have a normal form in T. □

Example 4.12 Let us consider the very simple example:

$$R = \begin{cases} h(h(x)) & \to & f(g(x)) \\ g(x) & \to & h(a) \\ f(a) & \to & f(a) \end{cases}$$

In the system R_V, the variable on the right side of the first rule is replaced with another variable y.

The automaton $\mathcal{A}_{NF(R)}$ contains, besides the rules yielding q_r, the following productions:

$$\begin{array}{rcl@{\qquad}rcl} a & \to & q_a & h(q_a) & \to & q_{h(x)} \\ f(q_{h(x)}) & \to & q_x & h(q_x) & \to & q_x \\ f(q_x) & \to & q_x & & & \end{array}$$

Final states will be $q_a, q_{h(a)}$ and q_x.

We add the following production rules:

$$\begin{array}{rcl@{\qquad}rcl} h(q_a) & \to & q_{h(a)} & f(q_a) & \to & q_{f(a)} \\ a & \to & q_\top & h(q_\top) & \to & q_\top \\ f(q_\top) & \to & q_\top & g(q_\top) & \to & q_\top \\ g(q_\top) & \to & q_{g(x)} & \Omega & \to & q'_x \\ a & \to & q'_x & h(q'_x) & \to & q'_x \\ g(q'_x) & \to & q'_x & f(q'_x) & \to & q'_x \\ h(q'_x) & \to & q'_{h(x)} & a & \to & q'_a \end{array}$$

We arrive at the saturation process which produces the following rules (we exclude the rules yielding $q_\top$):

$$\begin{array}{rcl@{\qquad}rcl} h(q'_{h(x)}) & \to & q'_x & g(q'_x) & \to & q_{h(x)} \\ g(q'_x) & \to & q_{h(a)} & g(q'_x) & \to & q'_{h(x)} \\ h(q'_{h(x)}) & \to & q_x & & & \end{array}$$

The last rule is obtained as follows: there is an instance of $g(y)$ which is accepted in state $q_{h(x)}$ (thanks to the rule $g(q'_x) \to q_{h(x)}$). Hence, from the rules $h(h(x)) \to f(g(y))$ and $f(q_{h(x)}) \to q_x$, we deduce $h(q'_{h(x)}) \to q_x$.

Corollary 4.13 *NV-sequentiality is decidable for left linear (possibly overlapping) rewrite systems.*

Proof: This is a consequence of theorem 3.3 and lemma 4.11. □

Note that this gives a much simpler proof than in [15], and in a more general case. We could also prove this result using ideas similar to [5].

4.4 Sorted systems

All above results can be extended to *order-sorted rewrite systems*. In such systems, variables are restricted to range over some regular sets of trees[2]. In particular, we find again some decidability results of [12] as well as their extension to arbitrary left-linear rewrite systems.

5 Direct construction of the automaton

For reasons which have already been explained, we construct here directly the automaton accepting all terms that have an index w.r.t Red. We only consider left linear rewrite systems.

For the direct construction of automata, it is more convenient to use the formalism of [13]. Of course, we could construct an automaton using the correspondence between automata and logic, but this construction would be too complex.

5.1 Another characterization of indexes

Let $t \in \mathcal{T}(\mathcal{F} \cup \{\Omega\}, \mathcal{X})$ and $u \in T(\mathcal{F}, X)$. We say that t is *compatible* with u if there is some instance v of u such that $t \sqsubseteq v$. Let R be a rewrite system. Then $\xrightarrow[R_\Omega]{}$ is the relation on $\mathcal{T}_\Omega$ defined by $t \xrightarrow[R_\Omega]{} u$ iff there is a $p \in Pos(t)$ such that $t|_p \neq \Omega$ and $t|_p$ is compatible with some left hand side of a rule and $u = t[\Omega]_p$. $\xrightarrow[R_\Omega]{}$ is a convergent reduction relation. The normal form of a term t w.r.t to $\xrightarrow[R_\Omega]{*}$ is written $t\downarrow_{R_\Omega}$.

[2] Order-sorted signatures, which include subsort declarations and overloading are exactly tree automata, as noticed e.g. in [4]

Theorem 5.1 ([13]) *Let $x \in \mathcal{X}$. The position p such that $t|_p = \Omega$ is an index of t (w.r.t. Red) iff $(t[x]_p \downarrow_{R_\Omega})|_p = x$.*

5.2 Construction of the automaton $\mathcal{A}_{noind}$

We construct first an automaton accepting the set of terms which that can be reduced to Ω by R_Ω.

Lemma 5.2 *For every linear rewrite system, there is an automaton $\mathcal{A}_{R_\Omega}$ with $O(|R|)$ states which recognizes the set of terms $t \in \mathcal{T}_\Omega$ that can be reduced to Ω using R_Ω.*

Note that $\mathcal{A}_{R_\Omega}$ is in general non-deterministic and that determinizing it may require exponentially many states. For example, if $g(f(x,a))$ and $h(f(x,b))$ are two left members of R, from $f(q_x, q_\Omega)$ we can reach the two states $q_{f(x,a)}$ and $q_{f(x,b)}$ and we cannot commit to any of them before knowing what is the symbol above. One way to prevent this situation is to add, for every subterm of a left hand side of a rule a state for each term obtained by replacing some subterms of t with Ω. But this step is exponential. It is not, in principle, better than determinizing $\mathcal{A}_{R_\Omega}$. Following this, it is possible to compute an automaton $\mathcal{A}_{ind}$ which accepts all terms that have an index. The automaton would be non-deterministic and contain exponetially many states. We will show its construction later on. However, for the decision problem, we need to decide wether all irreducible terms in $\mathcal{T}_\Omega$ are accepted by $\mathcal{A}_{ind}$. This question can only be decided in exponential time w.r.t. the number of states of the automaton (automata inclusion is EXPTIME-complete when the right member of the inclusion can be non-deterministic [18]). This would yield a doubly exponential test. It is however possible to reduce the complexity to a single exponential, computing directly an automaton for the complement of $\mathcal{A}_{ind}$ and deciding its inclusion in the set of reducible terms. That is what we are doing now.

Lemma 5.3 *For every left linear rewrite system R, it is possible to compute in $O(2^{|R|})$ time an automaton A_{noind} which accepts the terms $t \in \mathcal{T}_\Omega$ that do not have an index.*

5.3 Complexity issues

As a consequence of lemma 5.3, we get a complexity result:

Theorem 5.4 *Strong sequentiality is in EXPTIME when R is left linear (possibly overlapping).*

Proof: R is strongly sequential iff the set of terms that do not have an index is contained in the set of reducible terms. Or, equivalently, R is strongly sequential if there is no term in $\mathcal{T}_\Omega$ which is accepted by both $\mathcal{A}_{noind}$ and $\mathcal{A}_{NF(R)}$. Both automata can be computed in exponential time thanks to lemmas 5.3 and 4.5. Intersection can be done in quadratic time and the emptiness decision is again polynomial. □

The complex construction of lemma 5.3 can only be avoided when any two strict subterms of left hand sides are comparable w.r.t. $\sqsubseteq$ whenever they are headed with the same function symbol. In such a situation, $\mathcal{A}_{R_\Omega}$ can be made deterministic without adding any state (this is quite straightforward) and $\mathcal{A}_{noind}$ can be computed in polynomial time. We have also seen that, in such a case, the automaton $\mathcal{A}_{NF(R)}$ can be computed in polynomial time, hence:

Corollary 5.5 *For rewrite systems R such that any two strict subterms of a left hand side of a rule which have the same top symbol are comparable w.r.t. $\sqsubseteq$, strong sequentiality is decidable in polynomial time.*

In the case of non-left linear rewrite systems, the construction does not work, even if we use automata with constraints [2, 3] instead of bottom-up tree automata. (This was a mistake in a former version of this paper). For example, consider the rewrite system with only one rule whose left side is $f(x,x)$. Then

$$\begin{aligned} & f(f(g^k(f(a,a)),b), f(g^k(b),b)) \\ & \qquad \xrightarrow[R_\Omega]{} f(f(g^k(\Omega),b), f(g^k(b),b)) \\ & \qquad \xrightarrow[R_\Omega]{} \Omega \end{aligned}$$

However, the replacement for Ω in $f(g^k(\Omega),b)$ is known only when we reach the root of the term, i.e. arbitrary "far" from the first reduction. It is not possible to keep such an information in the finite memory of an automaton with constraints.

5.4 Construction of $\mathcal{A}_{ind}$

Now, instead of constructing $\mathcal{A}_{noind}$, let us construct $\mathcal{A}_{ind}$. This will show how to find indexes in a term. We start from a deterministic (completely specified) version of $\mathcal{A}_{R_\Omega}$ and complete it as follows. Each state q of $\mathcal{A}_{R_\Omega}$ is duplicated: we add the state $q^\bullet$ which will intuitively mean that we found an index below. Then we add the following rules:

- $\Omega \to q_\bullet^\bullet$ (this is a guess of an index position; we will express that it has to be applied once in each successful run)

- For each rule $f(q_1, \dots q_i, \dots, q_n) \to q$ we add the rules $f(q_1, \dots, q_i^\bullet, \dots, q_n) \to q^\bullet$

Final states are those which are marked with a $\bullet$.

Example 5.6 Consider a rewrite system whose left hand sides are $f(a, x), f(f(x, y), a)$ (with 3 function symbols, f, a, b).

The states of $\mathcal{A}_{ind}$ are $\{q_a, q_{f(x,y)}, q_\Omega, q_\bullet, q^\bullet_{f(x,y)}, q^\bullet_\bullet\}$ (states $q_a^\bullet$ and $q_b^\bullet$ have been removed since they are useless). The rules are:

$$
\begin{array}{rcl@{\qquad}rcl}
f(q_\bullet, q_\bullet) & \to & q_{f(x,y)} & \Omega & \to & q^\bullet_\bullet \\
f(q_1, q^\bullet_\bullet) & \to & q^\bullet_{f(x,y)} & \Omega & \to & q_\Omega \\
f(q_{f(x,y)}, q_{f(x,y)}) & \to & q_{f(x,y)} & f(q_a, q_3) & \to & q_\Omega \\
f(q_\bullet, q_{f(x,y)}) & \to & q_{f(x,y)} & a & \to & q_a \\
f(q_1, q^\bullet_{f(x,y)}) & \to & q^\bullet_{f(x,y)} & b & \to & q_\bullet \\
f(q_{f(x,y)}, q_a) & \to & q_\Omega & f(q^\bullet_\bullet, q_2) & \to & q^\bullet_{f(x,y)} \\
f(q^\bullet_{f(x,y)}, q_a) & \to & q_\Omega & f(q_\bullet, q_\Omega) & \to & q_{f(x,y)} \\
f(q^\bullet_{f(x,y)}, q_1) & \to & q^\bullet_{f(x,y)} & f(q_\bullet, q_a) & \to & q_{f(x,y)} \\
f(q_{f(x,y)}, q_\bullet) & \to & q_{f(x,y)} & f(q_\Omega, q_3) & \to & q_\Omega
\end{array}
$$

where q_1 is any state in $\{q_\bullet, q_{f(x,y)}\}$, q_2 is any state in $\{q_\bullet, q_{f(x,y)}, q_a, q_\Omega\}$ and q_3 is any state in $\{q_a, q_\Omega, q_\bullet, q_{f(x,y)}, q^\bullet_\bullet, q^\bullet_{f(x,y)}\}$. The final states are $q^\bullet_\bullet$ and $q^\bullet_{f(x,y)}$.

For example, $f(\Omega, \Omega)$ is accepted since $f(q^\bullet_\bullet, q_\Omega) \to q^\bullet_{f(x,y)}$. But $f(f(\Omega, \Omega), \Omega)$ is not accepted. Moreover, this term being irreducible, the system is not strongly sequential.

Lemma 5.7 *The automaton $\mathcal{A}_{ind}$ accepts all terms in $\mathcal{T}_\Omega$ that have an index.*

Now, if we come back to the problem of finding an index in a term, we can use $\mathcal{A}_{ind}$: all successful runs on t contain a $\bullet$-marked path from the root to an index. Consider for example the term $f(f(\Omega, \Omega), f(\Omega, \Omega))$, the only successful run is $q^\bullet_{f(x,y)}(q^\bullet_{f(x,y)}(q^\bullet_\bullet, q_\Omega), q_{f(x,y)}(q_\Omega, q_\Omega))$, which shows the path 11, which is the only index.

There is still some work to do w.r.t. reduction strategies. Indeed, so far, we have to apply the automaton on the whole term after each reduction step. Huet and Lévy, on the other hand, give a deterministic algorithm which never visits twice a node in the input term. To do something similar, we would have first to consider our automaton top-down (instead of bottom-up as we did through all the paper). Such an automaton is in general non-deterministic (and cannot be determinized). In order to avoid backtracking on the input term we would have to keep a stack of choice points and derive simultaneously the possible runs on the branch which is explored. This requires some additional implementation machinery, which is out of the scope of this paper.

6 Further applications

We believe that the tree automata approach can be used successfully to other works on reduction strategies. For example the *strong root stability* of [11] is also expressible in WSkS for left linear rewrite systems. The use of automata should also be investigated in parallel reduction strategies, such as in [19]: a run of the automaton not only gives an index position, but all index positions.

References

[1] S. Antoy and A. Middeldorp. A sequential reduction strategy. In *Proc. Algebraic and Logic Programming*, volume 850 of *LNCS*, pages 168–185. Springer-Verlag, 1994.

[2] B. Bogaert and S. Tison. Equality and disequality constraints on brother terms in tree automata. In A. Finkel, editor, *Proc. 9th Symp. on Theoretical Aspects of Computer Science*, Paris, 1992. Springer-Verlag.

[3] A.-C. Caron, J.-L. Coquidé, and M. Dauchet. Encompassment properties and automata with constraints. In C. Kirchner, editor, *Proc. 5th. Int. Conf. on Rewriting Techniques and Applications*, Lecture Notes in Computer Science, vol. 690, Montreal, Canada, 1993. Springer-Verlag.

[4] H. Comon and C. Delor. Equational formulae with membership constraints. *Information and Computation*, 112(2):167–216, Aug. 1994.

[5] M. Dauchet and S. Tison. The theory of ground rewrite systems is decidable. In *Proc. 5th IEEE Symp. Logic in Computer Science, Philadelphia*, 1990.

[6] N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 243–309. North-Holland, 1990.

[7] M. Gécseg and M. Steinby. *Tree Automata.* Akademia Kiadó, Budapest, 1984.

[8] G. Huet and J.-J. Lévy. Computations in orthogonal rewriting systems II. In J.-L. Lassez and G. Plotkin, editors, *Computational Logic: Essays in Honor of Alan Robinson*, pages 415–443. MIT Press, 1991. This paper was written in 1979.

[9] J.-P. Jouannaud and W. Sadfi. Strong sequentiality of left-linear overlapping rewrite systems. In *Workshop on Conditional Term Rewriting systems*, Jerusalem, July 1994.

[10] J. Kennaway. Sequential evaluation strategies for parallel-or and related reduction systems. *Annals of Pure and Applied Logic*, 43:31–56, 1989.

[11] R. Kennaway. A conflict between call-by-need computation and parallelism. In N. Dershowitz, editor, *Workshop on Conditional Term Rewriting Systems*, Jerusalem, 1994.

[12] D. Kesner. La définition de fonctions par cas à l'aide de motifs dans des langages applicatifs. Thèse de Doctorat, Université de Paris-Sud, France, Décembre 1993.

[13] J. W. Klop and A. Middeldorp. Sequentiality in orthogonal term rewriting systems. *Journal of Symbolic Computation*, 12:161–195, 1991.

[14] M. J. O'Donnell. Computing in systems described by equations. In *Lecture Notes in Computer Science*, volume 58. Springer, Berlin, West Germany, 1977.

[15] M. Oyamaguchi. NV-sequentiality: a decidable condition for call-by-need computations in term rewriting systems. *SIAM J. Comput.*, 22(1):114–135, 1993.

[16] M. Rabin. Decidability of second-order theories and automata on infinite trees. *Trans. Amer. Math. Soc.*, 141:1–35, 1969.

[17] M. Rabin. Decidable theories. In J. Barwise, editor, *Handbook of Mathematical Logic*, pages 595–629. North-Holland, 1977.

[18] H. Seidl. Deciding equivalence of finite tree automata. *Siam Journal of Computing*, 19(3):424–437, 1990.

[19] R. Sekar and I. Ramakrishnan. Programming in equational logic : beyond strong sequentiality. In *Proc. 5th IEEE Symp. Logic in Computer Science, Philadelphia*, 1990.

[20] J. Thatcher and J. Wright. Generalized finite automata with an application to a decision problem of second-order logic. *Math. Systems Theory*, 2:57–82, 1968.

[21] W. Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 134–191. Elsevier, 1990.

[22] Y. Toyama. Strong sequentiality of left linear overlapping term rewriting systems. In *Proc. 7th IEEE Symp. on Logic in Computer Science*, Santa Cruz, CA, 1992.