Regular Sets of Descendants for Constructor-based Rewrite Systems

Pierre Réty

LIFO - Université d'Orléans
B.P. 6759, 45067 Orléans cedex 2, France
e-mail : rety@lifo.univ-orleans.fr
http://www.univ-orleans.fr/SCIENCES/LIFO/Members/rety

Keywords : term rewriting, tree automata.

Abstract. Starting from the regular tree language E of ground constructor-instances of any linear term, we build a finite tree automaton that recognizes the set of descendants $R^*(E)$ of E for a constructor-based term rewrite system whose right-hand-sides fulfill the following three restrictions : linearity, no nested function symbols, function arguments are variables or ground terms. Note that left-linearity is not assumed. We next present several applications.

1 Introduction

Tree automata have already been applied to many areas of computer science, and in particular to rewriting techniques [2]. In comparison with more sophisticated refinements, finite tree automata are obviously less powerful, but have plenty of good properties and lead to much simpler algorithms from a practical point of view.

Because of potential applications to automated deduction and program validation, the problem of expressing by a finite tree automaton the transitive closure of a regular set E of ground terms with respect to an equational system, as well as the related problem of expressing the set of descendants of E with respect to a rewrite system, have already been investigated $[1, 5, 13, 4, 9]$[1]. All those papers assume that the right-hand-sides (both sides when dealing with equational systems) of rewrite rules are shallow, up to slight differences. Shallow means that every variable appears at depth at most one.

On the other hand, the possibility of approximating the set of descendants by means of a finite tree automaton, only assuming left-linearity, has been investigated in [7].

Our work is located in between : it adapts the former papers to constructor-based rewrite systems where right-hand-sides are not necessarily shallow, without making an approximation. Instead, we assume that function calls in right-

[1] [9] computes sets of normalizable terms, which amounts to compute sets of descendants by orienting the rewrite rules in the opposite sense.

hand-sides are shallow subterms (Restriction 1). However, to get regular sets of descendants, some additional restrictions are needed (Restrictions 2 and 3):

1. For each rule $l \to r$ and each function symbol position p in r, $r|_p = f(r_1, \ldots, r_n)$ where for all i, r_i is a variable or a ground term[2].
2. The right-hand-sides are linear and do not contain nested function symbols.
3. E is the set of the ground constructor-instances (also called data-instances) of a given linear term.

Fortunately, there is no need to start from any regular set E for applications. If any among the above restrictions is not satisfied, the set of descendants $R^*(E)$ is not regular in general. If it were, the set of normal forms $R^!(E)$ of E would be regular as well, provided that R is left-linear, because $R^!(E) = R^*(E) \cap IRR(R)$ and the set of irreducible ground terms $IRR(R)$ is regular in this case. The following array shows that $R^!(E)$ is not regular.

Unsatisfied Restriction	Rewrite System	E	$R^!(E)$
Linearity in rhs's	$f(x) \to c(x,x)$	$\{f(t)\}$	$\{c(t,t)\}$
Function calls in rhs's are shallow subterms	$f(s(x),y) \to s(f(x,s(y)))$	$f(s^*(0),0)$	$s^n(f(0,s^n(0)))$
No nested function symbols in rhs's	$f(s(x),y) \to s(f(x,g(y)))$ $g(x) \to s(x)$	$f(s^*(0),0)$	$s^n(f(0,s^n(0)))$
$E = \{t\theta\}$	$f(s(x)) \to s(f(x))$	$(fs)^*(0)$	$s^n(f^n(0))$

The construction of the automaton is presented in Section 3. It is necessary to nest automata, as defined in Subsection 2.2. Applications to reachability through rewrite steps, unification, program testing, sufficient completeness are outlined in Section 4.

2 Preliminaries

2.1 Term rewriting and finite tree automata

Surveys can be found in [6] about term rewriting, and in [2, 8] about tree automata.

Let C be a finite set of *constructors* and F be a finite set of *defined function symbols* (*functions* in a shortened form). For $c \in C \cup F$, $ar(c)$ is the arity of c. *Terms* are denoted by letters t, u. A *data-term* is a *ground* term (i.e. without variables) that contains only constructors. $T(C)$ is the set of data-terms. For a term t, $Var(t)$ is the set of variables appearing in t, $Pos(t)$ is the set of *positions* of t, $\overline{Pos}(t)$ is the set of non-variable positions of t, $PosF(t)$ is the set of function positions of t. t is *linear* if each variable of t appears only once in t. For $p \in Pos(t)$, $t|_p$ is the subterm of t at position p, $t(p)$ is the top symbol of

[2] It can be weakened into a more technical restriction, which allows non-shallow function calls: for each rewrite rule $l \to r$ and each function symbol position p in r, if $r|_p$ unifies with a left-hand-side l' (after variable renaming to avoid conflicts), then the mgu σ does not instantiate the variables of l', or only into ground subterms of r.

$t|_p$, and $t[t']_p$ denotes the subterm replacement. For positions p, p', $p \geq p'$ means that p is located below p', i.e. $p = p'.v$ for some position v, whereas $p||p'$ means that p and p' are incomparable, i.e. $\neg(p \geq p') \wedge \neg(p' \geq p)$. The term t contains *nested functions* if there exist $p, p' \in \overline{Pos}(t)$ s.t. $t(p) \in F$, $t(p') \in F$, and $p > p'$. The domain $dom(\theta)$ of a substitution θ is the set of variables x s.t. $x\theta \neq x$.

A *rewrite rule* is an oriented pair of terms, written $l \to r$. We always assume that $Var(r) \subseteq Var(l)$[3]. A *rewrite system* R is a finite set of rewrite rules. *lhs* stands for left-hand-side, *rhs* for right-hand-side. R is *constructor-based* if every lhs l of R is of the form $l = f(t_1, \ldots, t_n)$ where $f \in F$ and $t_1, \ldots, t_n$ do not contain any functions. The rewrite relation $\to_R$ is defined as follows: $t \to_R t'$ if there exist $p \in \overline{Pos}(t)$, a rule $l \to r \in R$, and a substitution θ s.t. $t|_p = l\theta$ and $t' = t[r\theta]_p$. $\to_R^*$ denotes the transitive closure of $\to_R$. t' is a *descendant* of t if $t \to_R^* t'$. t' is a *normal-form* of t if $t \to_R^* t'$ and t' is irreducible. If E is a set of ground terms, $R^*(E)$ denotes the set of descendants of elements of E, and $R^!(E)$ denotes the set of normal-forms. $IRR(R)$ denotes the set of irreducible ground terms. Thus $R^!(E) = R^*(E) \cap IRR(R)$. R is *weakly normalizing* if every term has at least one normal-form.

A (bottom-up) finite tree *automaton* is a quadruple $\mathcal{A} = (C \cup F, Q, Q_f, \Delta)$ where $Q_f \subseteq Q$ and Δ is a set of *transitions* of the form $c(q_1, \ldots, q_n) \to q$ where $c \in C \cup F$ and $q_1, \ldots, q_n, q \in Q$, or of the form $q_1 \to q$. Sets of *states* are denoted by letters Q, S, D, and states by q, s, d. $\to_\Delta$ (also denoted $\to_\mathcal{A}$) is the rewrite relation induced by Δ. A ground term t is *recognized* by $\mathcal{A}$ into q if $t \to_\Delta^* q$. $L(\mathcal{A})$ is the set of terms recognized by $\mathcal{A}$ into any states of Q_f. The states of Q_f are called *final states*. $\mathcal{A}$ is *deterministic* if whenever $t \to_\Delta^* q$ and $t \to_\Delta^* q'$ we have $q = q'$. A *Q-substitution* σ is a substitution s.t. $\forall x \in dom(\sigma)$, $x\sigma \in Q$.

2.2 Nesting automata

Definition 1. *The automaton $\mathcal{A} = (C \cup F, Q, Q_f, \Delta)$* discriminates *the position p into the state q if*

- *$L(\mathcal{A}) \neq \emptyset$,*
- *and $\forall t \in L(\mathcal{A})$, $p \in Pos(t)$,*
- *and for each successful derivation $t \to_\Delta^* t[q']_{p'} \to_\Delta^* q_f$ where $q_f \in Q_f$, we have*
  - *$q' = q$ if $p' = p$,*
  - *$q' \neq q$ otherwise.*

In this case we define the automaton $\mathcal{A}|_p = (C \cup F, Q, \{q\}, \Delta)$.

Lemma 2. *$L(\mathcal{A}|_p) = \{t|_p \mid t \in L(\mathcal{A})\}$.*

Proof. Let $t \in L(\mathcal{A})$. There is a successful derivation $t \to_\Delta^* C[q]_p \to_\Delta^* q_f$. Then $t|_p \to_\Delta^* q$. Consequently $t|_p \in L(\mathcal{A}|_p)$.

[3] Left-hand-sides are allowed to be variables.

Conversely, let $s \in L(\mathcal{A}|_p)$ and $t \in L(\mathcal{A})$. Since $s \to^*_\Delta q$ and $t \to^*_\Delta C[q]_p \to^*_\Delta q_f$, necessarily $t[s]_p \to^*_\Delta C[q]_p \to^*_\Delta q_f$, hence $t[s]_p \in L(\mathcal{A})$. Consequently $s = (t[s]_p)|_p \in \{t|_p \mid t \in L(\mathcal{A})\}$.

Definition 3. *Let $\mathcal{A} = (C \cup F, Q, Q_f, \Delta)$ be an automaton that discriminates the position p into the state q, and let $\mathcal{A}' = (C \cup F, Q', Q'_f, \Delta')$ s.t. $Q \cap Q' = \emptyset$. We define*

$$\mathcal{A}[\mathcal{A}']_p = (C \cup F,\ Q \cup Q',\ Q_f,\ \Delta \backslash \{c(q_1, \ldots, q_n) \to q \mid c \in C \cup F,\ q_1, \ldots, q_n \in Q\} \\ \cup \Delta' \cup \{q'_f \to q \mid q'_f \in Q'_f\})$$

Lemma 4. *$L(\mathcal{A}[\mathcal{A}']_p) = \{t[t']_p \mid t \in L(\mathcal{A}),\ t' \in L(\mathcal{A}')\}$.*

Proof. Let $t \in L(\mathcal{A})$ and $t' \in L(\mathcal{A}')$.
There exist the derivations $t \to^*_\Delta t[q]_p \to^*_\Delta q_f$ and $t' \to^*_{\Delta'} q'_f$. Then

$$t[t']_p \to^*_{\Delta'} t[q'_f]_p \to t[q]_p \to^*_\Delta q_f$$

Therefore $t[t']_p \in L(\mathcal{A}[\mathcal{A}']_p)$.

Conversely, let $s \in L(\mathcal{A}[\mathcal{A}']_p)$.
There exists the derivation $s \to^* s[q']_p \to^* q_f$.

• Within the part $s[q']_p \to^* q_f$, if a transition of Δ' is used, then we get some states of Q', which must necessarily disappear in the rest of the derivation because $q_f \notin Q'$. The only way for eliminating the states of Q' consists in applying the transition $q'_f \to q$. Then $s[q']_p \to^* C[q]_{p'}$, where $p' \neq p$. However, this is impossible because $\mathcal{A}$ discriminates p into q. Therefore $s[q']_p \to^*_\Delta q_f$ and necessarily $q' \in Q$.

• Without loss of generality, we can assume that every state of Q is reachable: there exists a ground term u s.t. $u \to^*_\Delta q'$. Thus $s[u]_p \to^*_\Delta s[q']_p \to^*_\Delta q_f$, then $s[u]_p \in L(\mathcal{A})$. Since $\mathcal{A}$ discriminates p into q, necessarily $q' = q$. Therefore

$$s \to^* s[q'_f]_p \to s[q]_p$$

• Within the part $s \to^* s[q'_f]_p$, if a transition of $\Delta \cup \{q'_f \to q\}$ is used, a state of Q is introduced, which is impossible because q'_f could not be reached. Therefore $s \to^*_{\Delta'} s[q'_f]_p$, hence $s|_p \in L(\mathcal{A}')$.

• Finally, $s = (s[u]_p)[s|_p]_p \in \{t[t']_p \mid t \in L(\mathcal{A}),\ t' \in L(\mathcal{A}')\}$.

As seen in the above proof, the states of Q' concern only the positions located below p. Therefore:

Corollary 5. *If $\mathcal{A}$ discriminates another position p' s.t. $p' \not\geq p$, into the state q', then $\mathcal{A}[\mathcal{A}']_p$ still discriminates p' into q'.*

3 An automaton that recognizes $R^*(E)$

Definition 6. *We define the automaton $\mathcal{A}_{data}$ that recognizes the set of data-terms $T(C)$:*
$\mathcal{A}_{data} = (C, Q_{data}, Q_{data_f}, \Delta_{data})$ where $Q_{data} = Q_{data_f} = \{q_{data}\}$ and $\Delta_{data} = \{c(q_{data}, \ldots, q_{data}) \to q_{data} \mid c \in C\}$.

Given a linear term t, we define the automaton $\mathcal{A}_{t\theta}$ that recognizes the data-instances of t : $\mathcal{A}_{t\theta} = (C \cup F, Q_{t\theta}, Q_{t\theta_f}, \Delta_{t\theta})$ *where*

$$Q_{t\theta} = \{q^p \mid p \in \overline{Pos}(t)\} \cup \{q_{data}\}$$
$$Q_{t\theta_f} = \{q^\epsilon\} \text{ (} q_{data} \textit{ if } t \textit{ is a variable)}$$
$$\Delta_{t\theta} = \left\{ t(p)(s_1, \ldots, s_n) \to q^p \mid p \in \overline{Pos}(t),\ s_i = \left| \begin{array}{l} q_{data} \textit{ if } t|_{p.i} \textit{ is a variable} \\ q^{p.i} \textit{ otherwise} \end{array} \right. \right\}$$
$$\cup \Delta_{data}$$

Note that $\mathcal{A}_{t\theta}$ discriminates each position $p \in \overline{Pos}(t)$ into q^p. On the other hand, $\mathcal{A}_{t\theta}$ is not deterministic[4] as soon as there is $p \in \overline{Pos}(t)$ s.t. $t|_p$ is a constructor-term. Indeed for any data-instance $t|_p\theta$, $t|_p\theta \to^*_{[\Delta_{t\theta}]} q^p$ and $t|_p\theta \to^*_{[\Delta_{t\theta}]} q_{data}$.

Example 1. Let a, s be constructors and f be a function, s.t. a is a constant and s, f are unary symbols. Consider the term $t = f(s(s(y)))$ as well as the automaton $\mathcal{A}_{t\theta}$ that recognizes the language $E = f(s(s(s^*(a))))$ of the data-instances of t. $\mathcal{A}_{t\theta}$ can be summarized by writing $\overset{q^\epsilon}{f}(\overset{q^1}{s}(\overset{q^{1.1}}{s}(\overset{q_{data}}{s^*(a)})))$, which means that $s^*(a) \to^*_{[\Delta_{t\theta}]} q_{data}$, $s(s^*(a)) \to^*_{[\Delta_{t\theta}]} q^{1.1}$, $s(s(s^*(a))) \to^*_{[\Delta_{t\theta}]} q^1$, $f(s(s(s^*(a)))) \to^*_{[\Delta_{t\theta}]} q^\epsilon$. Consider now the rewrite system $R = \{f(s(x)) \to s(f(x))\}$. Obviously $R^*(E) = E \cup s(f(s(s^*(a)))) \cup s(s(s^*(f(s^*(a)))))$.

When rewriting E, some instances of rhs's of rewrite rules are introduced by rewrite steps. So, to build an automaton that can recognize $R^*(E)$, we need to recognize the instances of rhs's into some states, without making any confusion between the various potential instances of the same rhs. Indeed consider the first two rewrite steps issued from E :

$$f(s(s(s^*(a)))) \to_{[x/s(s^*(a))]} \mathcal{L}' = s(f(s(s^*(a)))) \to_{[x/s^*(a)]} s(s(f(s^*(a))))$$

The language that instantiates x along the first step is $s(s^*(a))$ (recognized into $q^{1.1}$), whereas it is $s^*(a)$ (recognized into q_{data}) along the second step. Therefore we encode two versions of the rhs : $\overset{d^\epsilon_{q^{1.1}}}{s}(\overset{d^1_{q^{1.1}}}{f}(\overset{q^{1.1}}{x}))$ and $\overset{d^\epsilon_{q_{data}}}{s}(\overset{d^1_{q_{data}}}{f}(\overset{q_{data}}{x}))$, by adding the states $d^\epsilon_{q^{1.1}}$, $d^1_{q^{1.1}}$, $d^\epsilon_{q_{data}}$, $d^1_{q_{data}}$ and the transitions

$$f(q^{1.1}) \to d^1_{q^{1.1}},\ \ s(d^1_{q^{1.1}}) \to d^\epsilon_{q^{1.1}},\ \ f(q_{data}) \to d^1_{q_{data}},\ \ s(d^1_{q_{data}}) \to d^\epsilon_{q_{data}}$$

Thus the language recognized into $d^\epsilon_{q^{1.1}}$ (resp. $d^\epsilon_{q^{data}}$) is exactly the rhs instantiated by $\overset{q^{1.1}}{s}(\overset{q_{data}}{s^*(a)})$ (resp. $\overset{q_{data}}{s^*(a)}$). In other words $\mathcal{L}' = s(f(s(s^*(a)))) \to^* d^\epsilon_{q^{1.1}}$ (resp. $s(f(s^*(a))) \to^* d^\epsilon_{q_{data}}$). More generally we encode a version of the rhs for each state of $Q_{t\theta}$.

Now we can simulate rewrite steps on languages, by adding transitions again. This step is called saturation in the following. For example, consider again the first rewrite step issued from E :

[4] A direct construction of a deterministic automaton is given in [3]. However, it does not discriminate the positions of $\overline{Pos}(t)$.

$$\overset{q^\epsilon}{f}\,(\overset{q^1}{s}\,(\overset{q^{1.1}}{s}\,(\overset{q_{data}}{s^*(a)}))) \rightarrow_{[x/s(s^*(a))]} \mathcal{L}' = s(f(s(s^*(a))))$$

Since $f(s(x))$ is the rule lhs, and $f(s(q^{1.1})) \rightarrow^*_{\Delta_{t\theta}} q^\epsilon$, we add the transition $d^\epsilon_{q^{1.1}} \rightarrow q^\epsilon$. Thus $\mathcal{L}' = s(f(s(s^*(a)))) \rightarrow^* d^\epsilon_{q^{1.1}} \rightarrow q^\epsilon$ which is the final state. So $\mathcal{L}'$ is recognized by the automaton. More generally, whenever $f(s(q)) \rightarrow^* q'$ for $q \in Q_{t\theta}$, we add $d^\epsilon_q \rightarrow q'$.

In the previous example, the matches used in rewrite steps always instantiate the variable by languages recognized into states of $\mathcal{A}_{t\theta}$, i.e. the instances are (sub)terms of E. This is not the case in the following example.

Example 2. Let E be the data-instances of $t = f(z)$ and $R = \{f(x) \overset{r_1}{\rightarrow} g(s(a)),\ g(y) \overset{r_2}{\rightarrow} s(y)\}$. The rewrite steps issued from E are $\overset{q^\epsilon}{f}\,(\overset{q_{data}}{s^*(a)}) \rightarrow_{[r_1]} g(s(a)) \rightarrow_{[r_2, y/s(a)]} s(s(a))$. Unfortunately $Q_{t\theta} = \{q^\epsilon, q_{data}\}$ and the language recognized into q^ϵ (resp. q_{data}) is $f(s^*(a))$ (resp. $s^*(a)$). Thus we do not have any states that can exactly recognize the instance of the second rewrite step $\{s(a)\}$. This comes from the fact that $s(a)$ does not come from E, but from the rhs of r_1. Therefore we need to encode $s(a)$ by additional states.

In rhs's, function calls are assumed to be shallow subterms, and nested functions are not allowed. Therefore (see Lemma 17) the matches used in rewrite steps instantiate variables by either subterms of E, or (sub)-arguments of function calls in rhs's (which are data-terms). So, adding to $\mathcal{A}_{t\theta}$ states and transitions to encode all the function call arguments is enough.

Definition 7. *The non-variable arguments of functions in rhs's are encoded by the set of states Q_{arg} and the set of transitions Δ_{arg} as defined below:*

$$Q_{arg} = \{q^{i,p} \mid\ l_i \rightarrow r_i \in R,\ p \in Arg(r_i)\}$$
$$\Delta_{arg} = \{r_i(p)(q^{i,p.1}, \ldots, q^{i,p.n}) \rightarrow q^{i,p} \mid\ q^{i,p} \in Q_{arg}\}$$

where $Arg(r_i)$ are the non-variable argument positions in r_i, i.e.

$$Arg(r_i) = \{p \in \overline{Pos}(r_i) \mid\ \exists p_{fct} \in PosF(r_i),\ p > p_{fct}\}$$

Actually, for each state of $Q' = Q_{t\theta} \cup Q_{arg}$, we have to encode a version of each rhs. In general, unlike the previous examples, rhs's may contain several variables. This is why we use states of the form d^p_σ where σ is a Q'-substitution, instead of d^p_q where q is a single state of Q', to encode rhs's. Note that function arguments in rhs's are not encoded by new states in the following definition, but by those of Q_{arg}.

Definition 8. *The rhs's of rewrite rules are encoded by the sets of states Q_{arg} and*

$$D = \{d^{i,p}_\sigma \mid\ l_i \rightarrow r_i \in R,\ p \in Pos(r_i) \backslash Arg(r_i),\ \sigma \text{ is a } Q'\text{-substitution s.t. } dom(\sigma) = Var(r_i|_p)\}$$

and the set of transitions

$$\begin{array}{rl}\Delta_d = \{r_i(p)(X_1,\ldots,X_n) \to d^{i,p}_{\sigma_1\cup\ldots\cup\sigma_n} \mid & l_i \to r_i \in R, \\ & p \in Pos(r_i)\backslash Arg(r_i),\ r_i(p) \in C \\ & \forall j,\ \sigma_j \textit{ is any } Q'\textit{-substitution s.t. } dom(\sigma_j) = Var(r_i|_{p.j}), \\ & \textit{where } \forall j,\ X_j = \left| \begin{array}{l} x\sigma_j \mid\ r_i(p.j) \textit{ is any variable } x \\ d^{i,p.j}_{\sigma_j} \textit{ otherwise} \end{array} \right. \} \\ \cup \{r_i(p)(X_1,\ldots,X_n) \to d^{i,p}_{\sigma} \mid & l_i \to r_i \in R,\ p \in PosF(r_i), \\ & \sigma \textit{ is any } Q'\textit{-substitution s.t. } dom(\sigma) = Var(r_i|_p) \\ & \textit{where } \forall j,\ X_j = \left| \begin{array}{l} x\sigma \mid\ r_i(p.j) \textit{ is any variable } x \\ q^{i,p.j} \in Q_{arg} \textit{ otherwise} \end{array} \right. \} \\ \cup \{x\sigma \to d^{i,\epsilon}_{\sigma} \mid & l_i \to r_i \in R,\ r_i \textit{ is any variable } x, \\ & \sigma \textit{ is any } Q'\textit{-substitution s.t. } dom(\sigma) = \{x\}\} \end{array}$$

Thus, the ground term t is recognized into the state $d^{i,\epsilon}_{\sigma}$ iff $t = r_i\sigma$.

Let us explain now what happens when adding the transitions that simulate rewrite steps, if some lhs's are not linear.

Example 3. Let E be the data-instances of $t = f(s(x), y)$ and $R = \{f(x,x) \xrightarrow{r_1} x\}$. Thus $E = \overset{q^\epsilon}{f}(\overset{q^1}{s}(\overset{q_{data}}{s^*(a)}), \overset{q_{data}}{s^*(a)})$. Obviously $R^*(E) = E \cup \{s(s^*(a))\}$. If we try to add transitions to simulate rewrite steps as in Example 1, we have to look for the states $q \in Q_{t\theta}$ s.t. $f(q,q) \to^*_{\Delta_{t\theta}} q^\epsilon$. Unfortunately, no state of $Q_{t\theta} = \{q^\epsilon, q^1, q_{data}\}$ works.

Even so, the terms in the non-regular subset of E: $f(\overset{q^1}{s}(s^n(a)), \overset{q_{data}}{s^{n+1}}(a))$, are reducible. However they instantiate the left x of the lhs by q_1 and the right x by q_{data}, i.e. by two different states. So $s^{n+1}(a) \to^*_{\Delta_{t\theta}} q^1$ and $s^{n+1}(a) \to^*_{\Delta_{t\theta}} q_{data}$. In other words $\mathcal{A}_{t\theta}$ is not deterministic. If it were, the common instances $s^{n+1}(a)$ of the two occurrences of x would be recognized into the same state, so it would work.

In this example $Q_{arg} = \emptyset$. If it is not, we have to start from the automaton $(C \cup F, Q_{t\theta} \cup Q_{arg}, \{q^\epsilon\}, \Delta_{t\theta} \cup \Delta_{arg})$, which is not necessarily deterministic even if $\mathcal{A}_{t\theta}$ is. Therefore this automaton must be determinized before starting.

In all previous examples, t contained only one function. If there are several ones, and in particular, nested ones, we consider each function on its own and work incrementally.

Example 4. Let E be the data-instances of $t = f(g(z))$ and

$$R = \{f(x) \xrightarrow{r_1} x,\ g(x) \xrightarrow{r_2} s(h(x)),\ h(s(x)) \xrightarrow{r_3} x\}$$

Then

$$f(g(s^*(a))) \to_{[r_1]} g(s^*(a)) \to_{[r_2]} s(h(s^*(a))) \to^*_{[r_3]} s(s^*(a))$$

This derivation can be commutated so that the innermost function g is first reduced, as well as the rhs's coming from the reduction of g:

$$f(g(s^*(a))) \to_{[r_2]} f(s(h(s^*(a)))) \to_{[r_3]} f(s(s^*(a))) \to_{[r_1]} s(s^*(a))$$

Thanks to right-linearity, commuting rewrite derivations leaves the final term unchanged. This is why, we can write (roughly) $R^*(E) = R^*_{[r_1]}(f(R^*_{[r_2,r_3]}(g(s^*(a)))))$.

If t is for example $t = f(f(x))$, we must not make confusion between both occurrences of f in t. This is the goal of the following definitions.

Definition 9. *$t \rightarrow^{+}_{[p,rhs's]} t'$ means that t' is obtained by reducing t at position p, plus possibly at positions coming from the rhs's.*
Formally, there exist some intermediate terms $t_1, \ldots, t_n$ and some sets of positions $P(t), P(t_1), \ldots, P(t_n)$ s.t.

$$t = t_0 \rightarrow_{[p_0, l_0 \rightarrow r_0]} t_1 \rightarrow_{[p_1, l_1 \rightarrow r_1]} \ldots \rightarrow_{[p_{n-1}, l_{n-1} \rightarrow r_{n-1}]} t_n \rightarrow_{[p_n, l_n \rightarrow r_n]} t'$$

where

- $p_0 = p$ *and* $P(t) = \{p\}$,
- $\forall j,\ p_j \in P(t_j)$,
- $\forall j,\ P(t_{j+1}) = P(t_j) \backslash \{p' \mid p' \geq p_j\} \cup \{p_j.w \mid w \in PosF(r_j)\}$.

Remark : $P(t_j)$ only contains function positions. Since there are no nested functions in rhs's, $p, p' \in P(t_j)$ implies $p \| p'$.

Definition 10. *Given a language E and a position p, let*

$$R^*_p(E) = E \cup \{t' \mid \exists t \in E,\ t \rightarrow^{+}_{[p,rhs's]} t'\}$$

Example 5. Consider again Example 4. Then
$R^*_1(f(g(s^*(a)))) = f(g(s^*(a))) \cup f(s(h(s^*(a)))) \cup f(s(s^*(a)))$.

Lemma 11. *Let E be the set of data-instances of a linear term t, and $\{p_1, \ldots, p_n\}$ be the function positions of t sorted in an innermost way (i.e. $i < j \Longrightarrow p_j \not\geq p_i$). Then*

$$R^*(E) = R^*_{p_n}(\ldots(R^*_{p_1}(E))\ldots)$$

Proof. Obviously $R^*_{p_n}(\ldots(R^*_{p_1}(E))\ldots) \subseteq R^*(E)$.

Conversely, since R is right-linear, each rewrite derivation $t \rightarrow^* t'$ can be commuted into $t \rightarrow_{[p'_1]} \rightarrow \ldots \rightarrow_{[p'_k]} t'$ s.t. $i < j \Longrightarrow p'_j \not\geq p'_i$ (see [12, page 74]). Rewrite steps at incomparable positions can also be commuted. Therefore we can get a derivation of the form

$$t = t_1 \rightarrow^*_{[p_1,rhs's]} t_2 \rightarrow^* \ldots \rightarrow^* t_n \rightarrow^*_{[p_n,rhs's]} t_{n+1} = t'$$

where $t_i \rightarrow^*_{[p_i,rhs's]} t_{i+1}$ means $t_{i+1} = t_i$ or $t_i \rightarrow^{+}_{[p_i,rhs's]} t_{i+1}$.

From an automaton $\mathcal{A}$, we are now able to define an automaton that recognizes $R^*_p(L(\mathcal{A}))$.
Notation : $\mathcal{A}_{det}$ denotes the automaton obtained by determinizing $\mathcal{A}$.

Definition 12. *Let $\mathcal{A} = (C \cup F, Q, Q_f, \Delta)$ be an automaton that discriminates the position p into the state q, and s.t. $Q \cap Q_{arg} = \emptyset$. We define :*

$$\begin{aligned} \mathcal{A}' &= (C \cup F, Q', Q'_f, \Delta') = (C \cup F,\ Q \cup Q_{arg},\ \{q\},\ \Delta \cup \Delta_{arg})_{det} \\ \mathcal{A}'' &= (C \cup F, Q'', Q''_f, \Delta'') \\ &= (C \cup F,\ Q' \cup D \cup \{s\},\ \{s\},\ \Delta' \cup \Delta_d \cup \{q'_f \rightarrow s \mid q'_f \in Q'_f\}) \end{aligned}$$

Roughly speaking, $\mathcal{A}''$ is $\mathcal{A}'$ to which the encoding of rhs's has been added. In general $\mathcal{A}''$ is not deterministic although $\mathcal{A}'$ is (for example if two rhs's are identical). Determinizing $\mathcal{A}'$ is useless if every rewrite rule is left-linear. Note that $L(\mathcal{A}') = L(A|_p)$ and $\mathcal{A}|_p$ discriminates the position ϵ into q. However, due to the determinization, $\mathcal{A}'$ does not necessarily discriminate ϵ. This is why an additional state s has been added into $\mathcal{A}''$. Thus, $L(\mathcal{A}'') = L(\mathcal{A}') = L(\mathcal{A}|_p)$ and $\mathcal{A}''$ discriminates the position ϵ into s. This property is necessary in the saturation process defined below, to ensure that the first rewrite step is performed at position ϵ on the terms recognized by $\mathcal{A}''$, i.e. at position p on the terms recognized by $\mathcal{A}$.

Definition 13. *(*saturation*)*
Let $\mathcal{B}$ be the automaton obtained from $\mathcal{A}''$ by adding transitions in the following way : whenever there are $l_i \to r_i \in R$, a Q'-substitution σ s.t. $dom(\sigma) = Var(l_i)$ and $l_i\sigma \to^*_{\Delta''} q$ where $q \in \{s\} \cup D$, add the transition $d^{i,\epsilon}_{\sigma|_{Var(r_i)}} \to q$.

q may be in D in order to simulate the rewrite steps issued from the rhs's coming from previous rewrite steps. The saturation process necessarily terminates because it does not add any new states, and the number of transition rules is bounded in a finite automaton.

Lemma 14. $L(\mathcal{B}) = R^*_\epsilon(L(\mathcal{A}|_p))$.

Proof. See Subsection 3.1.

Now, the main result is a straightforward consequence of lemmas 14 and 4.

Corollary 15. $L(\mathcal{A}[\mathcal{B}]_p) = R^*_p(L(\mathcal{A}))$.

Remark : From Corollary 5 $\mathcal{A}[\mathcal{B}]_p$ preserves the discrimination of every position $p' \not\geq p$. Thus the construction of $\mathcal{B}$ from $\mathcal{A}$ can be used incrementally to compute $R^*(E) = R^*_{p_n}(\ldots(R^*_{p_1}(E))\ldots)$ as mentioned in Lemma 11.

Lemma 16. *In the worst case, the number of states of the final automaton that recognizes $R^*(E) = R^*_{p_n}(\ldots(R^*_{p_1}(E))\ldots)$ is a tower of exponentials of height n, in the number of positions of t.*

Proof. For simplicity, let us suppose that $card(Q_{arg}) = 0$. Thus

$$\sum_{l_i \to r_i \in R} card(Var(r_i)).card(Q') \leq card(D) \leq \sum_{l_i \to r_i \in R} |r_i|.card(Var(r_i)).card(Q')$$

where $|r_i| = card(Pos(r_i))$. Therefore for a given rewrite system, $card(D)$ is of the same order as $card(Q')$, i.e. $2^{card(Q)}$ because of the determinization. So $card(Q_\mathcal{B}) = card(Q'') = card(Q') + card(D) + 1$ is of the same order as $card(Q')$, i.e. $2^{card(Q)}$.

The result comes from the fact that building $\mathcal{B}$ from $\mathcal{A}$ is performed n times incrementally.

3.1 Proof of lemma 14

This proof breaks down into some lemmas.

We first need some properties about the matches used along a rewrite derivation.

Lemma 17. *Let $\mathcal{A} = (C \cup F, Q, Q_f, \Delta)$ be an automaton s.t. $Q \cap Q_{arg} = \emptyset$. For all $t \in L(\mathcal{A})$ and $p \in \overline{Pos}(t)$, if $t = t_0 \to^{+}_{[p,rhs's]} t_n$ then for each step $t_i \to_{[p_i, l_i \to r_i, \theta_i]} t_{i+1}$ within the derivation we have*

$$\forall x \in Var(l_i),\ x\theta_i \to^{*}_{\Delta \cup \Delta_{arg}} q \in Q \cup Q_{arg}$$

Proof. By induction on the length of the derivation.

If $n = 1$, $\forall x \in Var(l_0)$, $x\theta_0 = t|_v$ for some $v \in Pos(t)$. Therefore $x\theta_0 \to^{*}_{\Delta} q \in Q$.

Induction step. We know that $p_n \in P(t_n)$. Then there exists $j < n$ s.t. $p_n = p_j.w,\ w \in PosF(r_j)$, and $r_j|_w = f(u_1, \ldots, u_k)$. Therefore for each $x \in Var(l_n)$:

- $x\theta_n = u_l|_v$ where u_l is a non-variable function argument. Then, by construction of Δ_{arg}, $x\theta_n \to^{*}_{\Delta_{arg}} q \in Q_{arg}$.
- Or $x\theta_n = (y\theta_j)|_v$ where $y \in Var(r_j)$. By induction hypothesis, $y\theta_j \to^{*}_{\Delta \cup \Delta_{arg}} q \in Q \cup Q_{arg}$. Therefore $x\theta_n \to^{*}_{\Delta \cup \Delta_{arg}} q \in Q \cup Q_{arg}$.

Recall that $L(\mathcal{A}|_p) = L(\mathcal{A}')$. To prove completeness, i.e. $L(\mathcal{B}) \supseteq R^{*}_{\epsilon}(L(\mathcal{A}'))$, we have to prove a more precise property, to be able to make the induction step.

Lemma 18. *Let $t \in L(\mathcal{A}')$ and assume $t \to^{+}_{[\epsilon,rhs's]} t'$. Let us write $P(t') = \{p_1, \ldots, p_k\}$.*
Then $\forall p \in P(t')$, $t'|_p \to^{*}_{\Delta''} q_p \in \{s\} \cup D$ and $t'[q_{p_1}]_{p_1} \ldots [q_{p_k}]_{p_k} \to^{*}_{\mathcal{B}} s$.

Proof. By induction on the length of the derivation. Consider the last step $t_n \to_{[v, l \to r, \theta]} t'$. From lemma 17, $x\theta \to^{*}_{\Delta \cup \Delta_{arg}} q \in Q \cup Q_{arg}$. Thus, after determinizing, there exists one and only one $s_x \in Q'$ s.t. $x\theta \to^{*}_{\Delta'} s_x$.

Now $v \in P(t_n)$. By induction hypothesis $l\theta = t_n|_v \to^{*}_{\Delta''} q_n \in \{s\} \cup D$. Let σ be the Q'-substitution defined by $dom(\sigma) = Var(l)$ and $\forall x \in Var(l)$, $x\sigma = s_x$. Then $l\sigma \to^{*}_{\Delta''} q_n \in \{s\} \cup D$. From the construction of $\mathcal{B}$, $d^{\epsilon}_{\sigma|_{Var(r)}} \to q_n \in \Delta_{\mathcal{B}}$. Let $p' \in P(t')$.

- If $p' = v.w$ then $t'|_{p'} = (r\theta)|_w \to^{*}_{\Delta'} (r\sigma)|_w \to^{*}_{\Delta''} d^{w}_{\sigma} \in D$.
- Otherwise $p' \| v$. Then $t'|_{p'} = t_n|_{p'}$ and $p' \in P(t_n)$. By induction hypothesis $t_n|_{p'} \to^{*}_{\Delta''} q' \in \{s\} \cup D$.

Assume that $p_1, \ldots, p_k$ are sorted s.t. $p_1, \ldots, p_j \geq v$ and $p_{j+1}, \ldots, p_k \| v$. Thus

$$t'[d^{w_1}_{\sigma}]_{p_1} \ldots [d^{w_j}_{\sigma}]_{p_j} \to^{*}_{\mathcal{A}''} t'[d^{\epsilon}_{\sigma}]_v \to_{\mathcal{B}} t'[q_n]_v = t_n[q_n]_v$$
$$t'[q_{j+1}]_{p_{j+1}} \ldots [q_k]_{p_k} = t_n[q_{j+1}]_{p_{j+1}} \ldots [q_k]_{p_k}$$

Therefore, from the induction hypothesis

$$t'[d^{w_1}_{\sigma}]_{p_1} \ldots [d^{w_j}_{\sigma}]_{p_j} [q_{j+1}]_{p_{j+1}} \ldots [q_k]_{p_k} = t_n[q_n]_v [q_{j+1}]_{p_{j+1}} \ldots [q_k]_{p_k} \to^{*}_{\mathcal{B}} s$$

To prove correctness, i.e. $l(\mathcal{B}) \subseteq R^*_\epsilon(L(\mathcal{A}'))$, we also have to prove a more precise property. We first need an additional definition.

Definition 19. *Let Δ_{sat} be the transitions added by the saturation process (so $\Delta_\mathcal{B} = \Delta_{sat} \cup \Delta''$). For $t \in L(\mathcal{B})$, let $\|t\| = Min(\{length_{sat}(t \to^*_\mathcal{B} s)\})$ where $length_{sat}(t \to^*_\mathcal{B} s)$ is the number of steps using a transition of Δ_{sat}.*
The derivation $t \to^*_\mathcal{B} s$ is said minimal if $length_{sat}(t \to^*_\mathcal{B} s) = \|t\|$.

The proof of the following lemma shows that $\|t\|$ is the length of the shortest rewrite derivation that can reach t from a term of $L(\mathcal{A}')$.

Lemma 20. *If $t \to^*_\mathcal{B} s$ is minimal,*
then there exists $u \in L(\mathcal{A}')$ s.t. $u \to^*_{[\epsilon, rhs's]} t$, and for each intermediate term $C[q]_p$ in this minimal derivation (thus $t \to^*_\mathcal{B} C[q]_p \to^*_\mathcal{B} s$) s.t. $t(p)$ is a function and $q \in \{s\} \cup D$, we have $p \in P(t)$.

Proof. By induction on $\|t\|$.

If $\|t\| = 0$, then no minimal derivation $t \to^* q_f \to s$ ($q_f \in Q'_f$) contains any steps in Δ_{sat}, or in $\Delta''\backslash\Delta'$. Therefore $t \in L(\mathcal{A}')$. If $t \to^* C[s]_p$ then $p = \epsilon$ and $P(t) = \{\epsilon\}$.

Induction step. Consider a minimal derivation

$$t \to^*_{\Delta''} t_1 \to_{[p, d^{i,\epsilon}_\sigma \to q \in \Delta_{sat}]} t_2 \to^*_\mathcal{B} s$$

Since $t_2 \to^*_\mathcal{B} s$ is minimal, then $\|t_2\| = length_{sat}(t_2 \to^*_\mathcal{B} s) < \|t\|$. But t_2 contains some states. By construction of Δ_{sat}, $t_2[l_i\sigma]_p \to^*_{\Delta''} t_2[q]_p = t_2$ and $\forall x \in Var(l_i)$, $x\sigma \in Q'$. From rhs encoding, $\forall x \in Var(r_i)$, $x\theta \to^*_{\Delta'} x\sigma$. Besides, for each $x \in Var(l_i)\backslash Var(r_i)$ we choose a reachable state for $x\sigma$. Thus we can extend θ s.t. $x\theta \to^*_{\Delta'} x\sigma$. Consequently $t_2[l_i\theta]_p \to^*_{\Delta''} t_2 \to^*_{\Delta_\mathcal{B}} s$ and it is minimal. Thus $\|t_2[l_i\theta]_p\| < \|t\|$ and it does not contain any states. By induction hypothesis there exists $u \in L(\mathcal{A}')$ s.t. $u \to^*_{[\epsilon, rhs's]} t_2[l_i\theta]_p$ and for each intermediate term $C[q']_{p'}$ in this minimal derivation (thus $t \to^*_\mathcal{B} C[q']_{p'} \to^*_\mathcal{B} s$) s.t. $t_2[l_i\theta]_p(p)$ is a function and $q' \in \{s\} \cup D$, we have $p' \in P(t_2[l_i\theta]_p)$.

Now $q \in \{s\} \cup D$ because $d^{i,\epsilon}_\sigma \to q \in \Delta_{sat}$ and $t_2[l_i\theta]_p(p) = (l_i\theta)(\epsilon)$ is a function. So $p \in P(t_2[l_i\theta]_p)$. Therefore $u \to^*_{[\epsilon, rhs's]} t_2[r_i\theta]_p = t$, and for each intermediate term $C[q']_{p'}$ in the minimal derivation $t \to^* t_2 \to^*_\mathcal{B} s$ s.t. $t(p')$ is a function and $q' \in \{s\} \cup D$:

- if $p' \not\geq p$, then $t(p') = t_2[l_i\theta]_p(p')$, so $p' \in P(t)$.
- Otherwise since $t_2[l_i\theta]_p \to t$ and from the definition of P, we get $p' \in P(t)$.

4 Applications

4.1 Reachability

Reachability between ground terms is obviously decidable since $t_1 \to^* t_2$ is equivalent to $t_2 \in R^*(\{t_1\})$.

4.2 Unification of linear equations

We assume that R is confluent. Thus the equation $t \doteq t'$ admits a solution σ iff $t\sigma$ and $t'\sigma$ rewrite into the same term, i.e. $R^*(L(\mathcal{A}_{t\theta})) \cap R^*(L(\mathcal{A}_{t'\theta})) \neq \emptyset$ provided $Var(t) \cap Var(t') = \emptyset$ and t, t' are linear.

This extends the result of [10, 11] established thanks to TTSG's (Tree Tuple Synchronized Grammars), since left-linearity is not required any more. On the other hand, TTSG's can express unifiers, and they allow to weaken the linearity of the equation to be solved, as we are going to show in some further work.

4.3 Program testing

Let us see term rewrite systems as functional programs. The informal method to test a program usually consists in checking that for finitely many (and well chosen) data, the result is just the expected one. We suggest to do the same starting from an infinite language of data E, by providing an automaton that recognizes the language of expected results, and checking that it is equivalent to the automaton that recognizes $R^{data}(E) = R^*(E) \cap T(C)$.

Example 6. Consider the identity functions Id_2, (resp. Id_3) defined only for even natural integers (resp. integers multiple of three).

$$R = \{\ Id_2(s(s(x))) \to s(s(Id_2(x))),\ Id_2(0) \to 0,$$
$$Id_3(s(s(s(x)))) \to s(s(s(Id_3(x)))),\ Id_3(0) \to 0\}$$

Let E be the set of data-instances of $Id_2(Id_3(x))$. An automaton $\mathcal{A}_r$ that recognizes the expected results, i.e. the multiples of 6, can be easily built by the programmer. Then testing the program may consist in checking that $R^{data}(E) = L(\mathcal{A}_r)$.

4.4 Sufficient completeness

R is sufficiently complete if every ground term rewrites into a data-term.

If R is left-linear, a finite automaton that recognizes the set of reducible ground terms can be easily built, since reducible ground terms are the instances of the lhs's, nested in any context. By complementation we get an automaton that recognizes $IRR(R)$. Thus if in addition R is weakly normalizing, sufficient completeness is decidable thanks to the following lemma.

Lemma 21. *Assume that R is weakly normalizing. Then R is sufficiently complete iff $IRR(R) \subseteq T(C)$.*

Proof. $\Rightarrow$ is obvious.
$\Leftarrow$. Each term t admits a normal-form $t{\downarrow} \in T(C)$.

When R is not sufficiently complete, it might be interesting to check that the functions that are supposed to be completely defined, are indeed.

Notation: For each function f, let $E_f = \{f(t_1, \ldots, t_n) \mid t_1, \ldots, t_n \in T(C)\}$. E_f contains the data-instances of $f(x_1, \ldots, x_n)$.

Lemma 22. *[7] Assume that R is weakly normalizing. If $R^!(E_f) \subseteq T(C)$, then f is completely defined.*

Proof. $\forall t_1, \ldots, t_n \in T(C),\ f(t_1, \ldots, t_n) \rightarrow^* f(t_1, \ldots, t_n)\!\downarrow \in T(C)$.

5 Conclusion

We think that this work could be used in practice, because the final automaton often includes much fewer states than mentioned in Lemma 16. Indeed, determinization is useless whenever we reduce a function f s.t. every rewrite derivation issued from f uses only left-linear rules, which can be easily checked. Moreover if the encoding of rhs's is performed only when needed, we get a set of states much smaller than D in most cases.

The assumed restrictions about rhs's are not realistic from a programming point of view. We however hope that this work could give rise to approximations dealing with any rhs's, more precise than that of [7].

Another way to weaken the restrictions might consist in using more sophisticated tree languages, at the expense of more complicated algorithms.

Acknowledgments

I would like to thank Sébastien Limet for comments on this work, and Thomas Genet, Florent Jacquemard for helpful discussions.

References

1. H. Comon. Sequentiality, second order monadic logic and tree automata. In *Proc., Tenth Annual IEEE Symposium on Logic in Computer Science*, pages 508–517. IEEE Computer Society Press, 26–29 June 1995.
2. H. Comon, M. Dauchet, R. Gilleron, D. Lugiez, S. Tison, and M. Tommasi. *Tree Automata Techniques and Applications (TATA)*. http://l3ux02.univ-lille3.fr/tata.
3. Hubert Comon and Florent Jacquemard. Ground reducibility is exptime-complete. In *Proc. IEEE Symp. on Logic in Computer Science*, Varsaw, June 1997. IEEE Comp. Soc. Press.
4. J. Coquidé, M. Dauchet, R. Gilleron, and S. Vagvolgyi. Bottom-up Tree Pushdown Automata and Rewrite Systems. In R. V. Book, editor, *Proceedings 4th Conference on Rewriting Techniques and Applications, Como (Italy)*, volume 488 of *LNCS*, pages 287–298. Springer-Verlag, April 1991.
5. M. Dauchet and S. Tison. The theory of ground rewrite systems is decidable. In *Proc., Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 242–248, Philadelphia, Pennsylvania, 1990. IEEE Computer Society Press.
6. N. Dershowitz and J.-P. Jouannaud. Rewrite Systems. In J. Van Leuven, editor, *Handbook of Theoretical Computer Science*, chapter 6, pages 243–320. Elsevier Science Publishers, 1990.
7. T. Genet. Decidable Approximations of Sets of Descendants and Sets of Normal Forms. In *Proceedings of 9th Conference on Rewriting Techniques and Applications, Tsukuba (Japan)*, volume 1379 of *LNCS*, pages 151–165. Springer-Verlag, 1998.
8. R. Gilleron and S. Tison. Regular Tree Languages and Rewrite Systems. *Fundamenta Informaticae*, 24:157–175, 1995.
9. F. Jacquemard. Decidable Approximations of Term Rewrite Systems. In H. Ganzinger, editor, *Proceedings 7th Conference RTA, New Brunswick (USA)*, volume 1103 of *LNCS*, pages 362–376. Springer-Verlag, 1996.
10. S. Limet and P. Réty. E-Unification by Means of Tree Tuple Synchronized Grammars. In *Proceedings of 6th Colloquium on Trees in Algebra and Programming*, volume 1214 of *LNCS*, pages 429–440. Springer-Verlag, 1997.
11. S. Limet and P. Réty. E-Unification by Means of Tree Tuple Synchronized Grammars. *Discrete Mathematics and Theoritical Computer Science (http://dmtcs.loria.fr/)*, 1:69–98, 1997.
12. P. Réty. *Méthodes d'Unification par Surréduction.* Thèse de Doctorat d'Université, Université de Nancy I, March 1988. In french.
13. K. Salomaa. Deterministic Tree Pushdown Automata and Monadic Tree Rewriting Systems. *The Journal of Computer and System Sciences*, 37:367–394, 1988.