Regular Sets of Descendants by some Rewrite Strategies

Pierre Réty   Julie Vuotto

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

Abstract. For a constructor-based rewrite system R, a regular set of ground terms E, and assuming some additional restrictions, we build a finite tree automaton that recognizes the descendants of E, i.e. the terms issued from E by rewriting, according to innermost, innermost-leftmost, and outermost strategies.

1 Introduction

Tree automata have already been applied to many areas of computer science, and in particular to rewrite techniques [2]. In comparison with more sophisticated refinements [4, 10, 9], finite tree automata are obviously less expressive, 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 (reachability, program testing), the problem of expressing by a finite tree automaton the transitive closure of a regular set E of ground terms with respect to a set of equations, as well as the related problem of expressing the set of descendants $R^*(E)$ of E with respect to a rewrite system R, have already been investigated [1, 5, 14, 3, 11, 15][1]. Except [11, 15], all those papers assume that the right-hand-sides (both sides when dealing with sets of equations) of rewrite rules are shallow[2], up to slight differences. On the other hand, P. Réty's work [12] does not always preserve recognizability (E is not arbitrary), but allows rewrite rules forbidden by the other papers[3].

Reduction strategies in rewriting and programming have drawn an increasing attention within the last years, and matter both from a theoretical point of view, if the computation result is not unique, and from a practical point of view, for termination and efficiency. For a strategy st, expressing by a finite tree automaton the st-descendants $R^*_{st}(E)$ of E, can help to study st: in particular it allows to decide st-reachability since $t_1 \xrightarrow{st}{}^* t_2 \iff t_2 \in R^*_{st}(\{t_1\})$, and st-joinability since $t_1 \overset{st}{\downarrow} t_2 \iff R^*_{st}(\{t_1\}) \cap R^*_{st}(\{t_2\}) \neq \emptyset$. More generally,

[1] [11] computes sets of normalizable terms, which amounts to compute sets of descendants by orienting the rewrite rules in the opposite direction.

[2] Shallow means that every variable appears at depth at most one.

[3] Like $f(s(x)) \to s(f(x))$.

it can help with the static analysis of rewrite programs, and by extension, of functional programs.

This paper is an extension of [12] that takes some strategies into account. As far as we know, the problem of expressing sets of descendants according to some strategies had not been addressed yet. We build finite tree automata that can express the sets of descendants of E with respect to a constructed-based rewrite system, according to innermost, innermost-leftmost, outermost strategies, assuming :

1. E is the set of ground constructor-instances (also called data-instances) of a given linear term.
2. Every rewrite rule is linear (both sides).
3. In right-hand-sides, there are no nested defined-functions, and arguments of defined-functions are either variables or ground terms.

For innermost-leftmost strategy, we temporarily assume in addition that right-hand-sides contain at most one defined-function. For outermost strategy, we also assume this extra restriction, and moreover that R has no critical pairs.

It is shown in [12] that if any restriction among 1, 2, 3 is not satisfied, then the set of descendants is not regular (even if a strategy among innermost, innermost-leftmost, outermost, is respected). About restriction 2, only non-right-linearity causes non-regularity. However, to deal with strategies, we need the regularity of the set of irreducible terms, hence left-linearity.

The paper is organized as follows. Section 2 introduces preliminaries notions. The reader used to term rewriting and tree automata may skip Subsection 2.1, but not the following ones which present more specific notions. Section 3 (resp. 4, 5) gives the computation of innermost (resp. innermost-leftmost, outermost) descendants. Missing proofs are available in the full version. See Réty's web page.

2 Preliminaries

2.1 Usual Notions : Term Rewriting and 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 s, t. A *data-term* is a *ground* term (i.e. without variables) that contains only constructors. T_C is the set of data-terms, $T_{C \cup F}$ is the set of ground-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 defined-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 $t|_p$, 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 $p, p' \in PosF(t)$ s.t. $p > p'$. 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 $l \to r$. 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 $l = f(t_1, \dots, t_n)$ where $f \in F$ and $t_1, \dots, t_n$ do not contain any functions. The rewrite relation $\to_R$ is defined as follows: $t \to_R t'$ if there exist $p \in Pos(t)$, a rule $l \to r \in R$, and a substitution θ s.t. $t|_p = l\theta$ and $t' = t[r\theta]_p$ (also denoted by $t \to_{[p,l\to r,\theta]} t'$). $\to_R^*$ denotes the reflexive-transitive closure of $\to_R$. t is *irreducible* if $\neg(\exists t' \mid t \to_R t')$. $t \to_{[p]} t'$ is innermost (resp. leftmost, outermost) if $\forall v > p$ (resp. $\forall v$ occurring strictly on the left of p, $\forall v < p$) $t|_v$ is irreducible. The *narrowing* relation $\rightsquigarrow_R$ is defined as follows: $t \rightsquigarrow_R t'$ if there exist $p \in \overline{Pos}(t)$, a rule $l \to r \in R$, and a substitution θ s.t. $t|_p\theta = l\theta$ and $t' = (t\theta)[r\theta]_p$ (also denoted by $t \rightsquigarrow_{[p,l\to r,\theta]} t'$).

A (bottom-up) finite tree *automaton* is a quadruple $\mathcal{A} = (C \cup F, Q, Q_f, \Delta)$ where $Q_f \subseteq Q$ are sets of states and Δ is a set of *transitions* of the form $c(q_1, \dots, q_n) \to q$ where $c \in C \cup F$ and $q_1, \dots, q_n, q \in Q$, or of the form $q_1 \to q$ (*empty transition*). 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. A derivation $t \to_\Delta^* q$ where $q \in Q_f$ is called a *successful run* on t. 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. $\forall x \in dom(\sigma)$, $x\sigma \in Q$. A set E of ground terms is regular if there exists a finite automaton $\mathcal{A}$ s.t. $E = L(\mathcal{A})$.

2.2 Nesting Automata

Intuitively, the automaton $\mathcal{A}$ discriminates position p into state q means that along every successful run on $t \in L(\mathcal{A})$, $t|_p$ (and only this subterm) is recognized into q. This property allows to modify the behavior of $\mathcal{A}$ below position p without modifying the other positions, by replacing $\mathcal{A}|_p$ by another automaton $\mathcal{A}'$. See [12] for missing proofs.

Definition 2.1. *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*
  - $q' \to_\Delta^* q$ *(i.e. by empty transitions) if* $p' = p$,
  - *not* $(q' \to_\Delta^* q)$ *otherwise.*

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

Lemma 2.2. $L(\mathcal{A}|_p) = \{t|_p \mid t \in L(\mathcal{A})\}$.

Definition 2.3. *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*

$$\begin{aligned}\mathcal{A}[\mathcal{A}']_p = (C \cup F,\ Q \cup Q',\ Q_f,\ & \Delta \backslash \{l \to r \mid l \to r \in \Delta \wedge r = q\} \\ & \cup \Delta' \cup \{q'_f \to q \mid q'_f \in Q'_f\})\end{aligned}$$

Lemma 2.4. *$L(\mathcal{A}[\mathcal{A}']_p) = \{t[t']_p \mid t \in L(\mathcal{A}), t' \in L(\mathcal{A}')\}$, and $\mathcal{A}[\mathcal{A}']_p$ still discriminates p into q. Moreover, 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'.*

Lemma 2.5. *Let $\mathcal{A}, \mathcal{B}$ be automata, and let $\mathcal{A} \cap \mathcal{B}$ be the classical automaton used to recognize intersection, whose states are pairs of states of $\mathcal{A}$ and $\mathcal{B}$.*
If $\mathcal{A}$ discriminates p into $q_{\mathcal{A}}$, $\mathcal{B}$ discriminates p into $q_{\mathcal{B}}$, and $L(\mathcal{A}) \cap L(\mathcal{B}) \neq \emptyset$, then $\mathcal{A} \cap \mathcal{B}$ discriminates p into $(q_{\mathcal{A}}, q_{\mathcal{B}})$.

Proof. Let $t \in L(\mathcal{A} \cap \mathcal{B})$.

- since $t \in L(\mathcal{A}),\ p \in Pos(t)$
- for any successful run on t,
  $t \to^*_{\Delta_{\mathcal{A}\cap\mathcal{B}}} t[(q'_{\mathcal{A}}, q'_{\mathcal{B}})]_{p'} \to^* (qf_{\mathcal{A}}, qf_{\mathcal{B}})$
  - if $p' = p$ then from discrimination of $\mathcal{A}$ and $\mathcal{B}$, $q'_{\mathcal{A}} \to^*_{\Delta} q_{\mathcal{A}}$ and $q'_{\mathcal{B}} \to^*_{\Delta} q_{\mathcal{B}}$
  - if $p' \neq p$ then from discrimination of $\mathcal{A}$ and $\mathcal{B}$, $not(q'_{\mathcal{A}} \to^*_{\Delta} q_{\mathcal{A}})$ and $not(q'_{\mathcal{B}} \to^*_{\Delta} q_{\mathcal{B}})$.

2.3 Particular Automata

Let us define the initial automaton, i.e. the automaton that recognizes the data-instances of a given linear term t.

Definition 2.6. *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 $\Delta_{data} = \{c(q_{data}, \dots, q_{data}) \to q_{data} \mid c \in C\}$.

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

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

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, 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 $t|_p\theta \to^*_{[\Delta_{t\theta}]} q_{data}$.

Let us now define an automaton that recognizes the terms irreducible at position p.

Definition 2.7. *Let $IRR_p(R) = \{s \in T_{C\cup F} \mid p \in Pos(s)$ and $s|_p$ is irreducible$\}$.*

To prove the regularity of $IRR_p(R)$, we need some more definitions.

Definition 2.8. *Let $RED(R)$ be the language of reducible terms:*

$$RED(R) = \{s \mid \exists p' \in Pos(s) \quad s \rightarrow_{[p', l\rightarrow r, \sigma]} s'\}$$

Lemma 2.9. *[6] If R is left-linear, $RED(R)$ is a regular language.*

Lemma 2.10. *$IRR_\epsilon(R) = \overline{RED(R)}$. Therefore, $IRR_\epsilon(R)$ is a regular language.*

Thanks to an automaton that recognizes $IRR_\epsilon(R)$, we can now build an automaton that recognizes $IRR_p(R)$.

Theorem 2.11. *Let t be a term, and $p \in \overline{Pos}(t)$. $IRR_p(R)$ is a regular language and is recognized by an automaton that discriminates every position $p' \in \overline{Pos}(t)$ s.t. $p' \not> p$.*

Proof. Let $\mathcal{A}_\epsilon = (\mathcal{C} \cup \mathcal{F}, Q_\epsilon, Q_{\epsilon f}, \Delta_\epsilon)$ be an automaton that recognizes $IRR_\epsilon(R)$.
Let $p = p_1. \dots .p_k$ with $p_1, \dots, p_k \in \mathbb{N} - \{0\}$
and $\forall i \; p_i \leq Max_{f \in F\cup C}(ar(f))$
We define $\mathcal{A}_{irr}$ as follows :
$\mathcal{A}_{irr} = (C \cup F, Q_{irr}, Q_{irrf}, \Delta_{irr})$ where

$$Q_{irr} = \{q_{any},\ q_{rec}\} \cup_{v<p} \{q^v\} \cup_{v\in\overline{Pos}(t)\setminus\{v' \mid v' \leq p\}} \{q^v_{any}\} \cup Q_\epsilon$$

$$Q_{irrf} = \{q^\epsilon\} \text{ and}$$

$$\Delta_{irr} = \{s(S_1, \dots, S_n) \rightarrow q^j \mid s \in F \cup C, ar(s) \geq p_{long(j)+1}$$

$$q^j \in Q_{irr},\ S_i = \left| \begin{array}{ll} q^{j.i} & \text{if } j.i < p \\ q_{rec} & \text{if j.i=p} \\ q^{j.i}_{any} & \text{otherwise} \end{array} \right. \quad \} \, if \, p \neq \epsilon$$

$$\cup \{q_f \rightarrow q_{rec} \mid q_f \ \in \ Q_{\epsilon f}\} \, if \, p \neq \epsilon$$
$$\cup \{q_f \rightarrow q^\epsilon \mid q_f \ \in \ Q_{\epsilon f}\} \, if \, p = \epsilon$$
$$\cup \{s(S_1, \dots, S_n) \rightarrow q^j_{any} \mid s \in F \cup C,\ q^j_{any} \in Q_{irr},$$
$$S_i = \left| \begin{array}{ll} q^{j.i}_{any} & \text{if } j.i \in \overline{Pos}(t) \\ q_{any} & \text{otherwise} \end{array} \right. \}$$
$$\cup \{s(q_{any}, \dots, q_{any}) \rightarrow q_{any} \mid s \in F \cup C\}$$
$$\cup \Delta_\epsilon$$

$\mathcal{A}_{irr}$ recognizes $IRR_p(R)$ indeed, because:
$t|_p$ reducible i.e. $\exists\, u$ position s.t $u \geq p$ and $t \rightarrow_{[u]} t'$.

- q_{any} recognizes any terms.
- q^w recognize $t|_w$ for $w < p$.

We have written $ar(s) \geq p_{long(j)+1}$ to ensure that $p \in Pos(t)$. For example, if $p = 1.2.1$ and $s(\dots) \rightarrow q^1$, then s should have an arity ≥ 2.
Obviously, $\mathcal{A}_{irr}$ discriminates p into q_{rec} (into q^ϵ if $p = \epsilon$), and each $p' \in \overline{Pos}(t)$ s.t. $p' \not\geq p$ into $q^{p'}_{any}$ ($q^{p'}$ if $p' < p$).

2.4 Descendants

t' is a *descendant* of t if $t \rightarrow_R^* t'$. t' is a *normal-form* of t if $t \rightarrow_R^* t'$ and t' is irreducible. If E is a set of ground terms, $R^*(E)$ denotes the set of descendants of elements of E. $IRR(R)$ denotes the set of irreducible ground terms. $R^*_{in}(E)$ (resp. $R^*_{ileft}(E)$, $R^*_{out}(E)$) denotes the set of descendants of E, according to an innermost (resp. innermost-leftmost, outermost) strategy.

Definition 2.12. *$t \rightarrow^+_{[p,rhs's]} t'$ means that t' is obtained by rewriting t at position p, plus possibly at positions coming from the rhs's.*
Formally, there exist some intermediate terms $t_1, \dots, t_n$ and some sets of positions $P(t), P(t_1), \dots, P(t_n)$ s.t.
$t = t_0 \rightarrow_{[p_0, l_0 \rightarrow r_0]} t_1 \rightarrow_{[p_1, l_1 \rightarrow r_1]} \dots \rightarrow_{[p_{n-1}, l_{n-1} \rightarrow r_{n-1}]} t_n \rightarrow_{[p_n, l_n \rightarrow r_n]} t_{n+1} = t'$
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 : $P(t_j)$ contains only function positions. Since there are no nested functions in rhs's, $p, p' \in P(t_j)$ implies $p \| p'$.

Definition 2.13. *Given a language E and a position p, we define $R^*_p(E)$ as follows*

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

Example 1. $R = \{f(x) \rightarrow s(x),\ g(x) \rightarrow s(h(x)),\ h(x) \rightarrow f(x)\}$
$R^*_1(f(h(g(a)))) = E \cup f(f(g(a))) \cup f(s(g(a))$

An insight into the algorithm underlying the following result will be given in the sequel by Example 2, and a formal description is in the full version. The resultant automata are different from the starting one only at positions below p, and in the general case, are built by nesting automata.

Theorem 2.14. *[12] Let R be a rewrite system satisfying the restrictions given in the introduction. If E is recognized by an automaton that discriminates position p into some state q, and possibly p' into q' for some $p' \in \overline{Pos}(t)$ s.t. $p' \not\geq p$, and some states q', then so is $R^*_p(E)$.*

2.5 Positions

Given a term t, we define:

Definition 2.15. *Let $p \in Pos(t)$. $Succ(p)$ are the nearest function positions below p:*
$Succ(p) = \{p' \in PosF(t) \mid p' > p \text{ and } \forall q \in Pos(t)\ (p < q < p' \Rightarrow q \notin PosF(t))\}$

Definition 2.16. *Let $p, p' \in Pos(t)$. $p \lhd p'$ means that p occurs strictly on the left of p', i.e. $p = u.i.v$, $p' = u.i'.v'$, where $i, i' \in I\!N$ and $i < i'$.*

3 Innermost Descendants : $R^*_{in}(E)$

Example 2. Let a, s be constructors and f be a function, s.t. a is a constant, and s, f are unary symbols. Let $t = f(s(f(s(y))))$ and $\mathcal{A}_{t\theta}$ be the automaton that recognizes the language $E = f(s(f(s(s^*(a)))))$ of the data-instances of t.
$\mathcal{A}_{t\theta}$ can be summarized by writing :

$$\overset{q^{\epsilon}}{f}\ (\overset{q^1}{s}\ (\overset{q^{1.1}}{f}\ (\overset{q^{1.1.1}}{s}\ (\overset{q_{data}}{s^*}\ (a)))))$$

which means that

$$\Delta_{t\theta} = \{a \to q_{data},\ s(q_{data}) \to q_{data},\ s(q_{data}) \to q^{1.1.1},\\ f(q^{1.1.1}) \to q^{1.1},\ s(q^{1.1}) \to q^1,\ f(q^1) \to q^{\epsilon}\}$$

where q^{ϵ} is the accepting state.
Consider now the rewrite system $R = \{f(s(x)) \to s(x)\}$.
Obviously, $R^*_{in}(E) = E \cup f(s(s(s^*(a)))) \cup s(s(s^*(a)))$.

We can make two remarks:

- When rewriting E, some instances of rhs's of rewrite rules are introduced by rewrite steps. So, to build an automaton that can recognize $R^*_{in}(E)$, 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.
- When the starting term has nested functions, according to the innermost strategy, we first have to rewrite innermost function positions.

Note that here, we can rewrite E at positions ϵ and 1.1. According to the previous remark, we start from position 1.1.

Now, we calculate $R^*_{1.1}(E)$.

$$(1) \qquad f(s(f(s(s^*(a))))) \to_{[1.1, x/s^*(a)]} f(s(s(s^*(a))))$$

The language that instantiates the rewrite rule variable x is $s^*(a)$ (recognized into q_{data}). Therefore, we encode the first version of the rhs: $\overset{d^{\epsilon}_{q_{data}}}{s}\ (\overset{q_{data}}{x})$ by adding state $d^{\epsilon}_{q_{data}}$ and the transition $s(q_{data}) \to d^{\epsilon}_{q_{data}}$.

We can simulate the rewrite step, by adding transitions again. This step is called saturation in the following. Consider (1) again. Since $f(s(x))$ is the rule lhs, and $f(s(q_{data})) \to^*_{\Delta_{t\theta}} q^{1.1}$, we add the transition $d^{\epsilon}_{q_{data}} \to q^{1.1}$ so that the instance of the rhs by q_{data} is also recognized into $q^{1.1}$, i.e. $s(s^*(a)) \to^* q^{1.1}$.
So, $R^*_{1.1}(E) = E \cup f(s(s(s^*(a))))$ is recognized by the automaton.

Now, rewriting terms of $R^*_{1.1}(E)$ at position ϵ is allowed only if position 1.1 is normalized. Consider $E' = R^*_{1.1}(E) \cap IRR_{1.1}(R)$ where $IRR_{1.1}(R)$ is the ground terms irreducible at position 1.1, over the TRS R. Thus $E' = f(s(s(s^*(a))))$, and let us calculate $R^*_{\epsilon}(E')$.

Let $\mathcal{A}' = (\mathcal{C} \cup \mathcal{F}, Q', \{q'^{\epsilon}\}, \Delta')$ an automaton that recognizes the language E' where $\Delta' = \{a \to q_{data},\ s(q_{data}) \to q_{data},\ s(q_{data}) \to q'^{1.1},\ s(q'^{1.1}) \to$

$q'^1,\ f(q'^1) \rightarrow q'^\epsilon\}$ where q'^ϵ is the accepting state.

$$(2) \qquad f(s(s(s^*(a)))) \rightarrow_{[x/s(s^*(a))]} s(s(s^*(a)))$$

The language that instantiates x is $s(s^*(a))$ (recognized into ${q'}^{1.1}$). Therefore, we encode a second version of the rhs : $\overset{d^\epsilon_{q'^{1.1}}}{s}(\overset{q'^{1.1}}{x})$ by adding state $d^\epsilon_{q'^{1.1}}$ and the transition $s({q'}^{1.1}) \rightarrow d^\epsilon_{q'^{1.1}}$.
By saturation, since $f(s(x))$ is the rule lhs and $f(s({q'}^{1.1})) \rightarrow {q'}^\epsilon$, we add the transition $d^\epsilon_{q'^{1.1}} \rightarrow {q'}^\epsilon$ so that $s(s(s^*(a))) \rightarrow^* q'^\epsilon$.
So, $R^*_\epsilon(E') = E' \cup s(s(s^*(a)))$ is recognized by the automaton.

E' contains only terms normalized at position 1.1, which is not required by the innermost strategy when no rewrite step is applied at position ϵ.
Therefore, $R^*_{in}(E) = R^*_\epsilon(E') \cup R^*_{1.1}(E) = R^*_\epsilon(R^*_{1.1}(E) \cap IRR_{1.1}(R)) \cup R^*_{1.1}(E)$.

Remark : In the previous example, the starting term has nested functions. When it is not the case, every rewrite step is innermost, because rhs's have no nested functions either.

In general t may have more than two function positions. To generalize, we need the following notion.

Definition 3.1. *Given a language L and a position p, $R^*_{in,p}(L)$ are the innermost descendants of L over the TRS R, reducing positions below (or equal to) p, i.e.*
$R^*_{in,p}(L) = \{s' \mid \exists s \in L,\ s \rightarrow^*_{[u_1,\ldots,u_n]} s' \text{ by an innermost strategy}, \forall i\ (u_i \geq p)\}$

For a language L, let $L|_p = \{s|_p \mid s \in L,\ p \in Pos(s)\}$.

Lemma 3.2. *Let R be a rewrite system satisfying the restrictions given in the introduction, and E be the data-instances of a given linear term t.
Let $p \in PosF(t)$, and L be a language s.t. $L|_p = E|_p$, and that is recognized by an automaton $\mathcal{A}$ that discriminates every position $p' \in PosF(t) \mid p' \geq p$. Then,*

$$R^*_{in,p}(L) = R^*_p(L)\ if\ Succ(p) = \emptyset$$

Otherwise, let $Succ(p) = \{p_1, \ldots, p_n\}$, and in this case

$$R^*_{in,p}(L) = \left| \begin{array}{l} R^*_p[R^*_{in,p_1}(\ldots(R^*_{in,p_n}(L))\ldots) \cap_{p_i \in Succ(p)} IRR_{p_i}(R)] \\ \ \cup\ R^*_{in,p_1}(\ldots(R^*_{in,p_n}(L))\ldots) \end{array} \right.$$

and $R^*_{in,p}(L)$ is recognized by an automaton $\mathcal{A}'$ s.t. if $p' \in \overline{Pos}(t)$, $p' \not> p$, and $\mathcal{A}$ discriminates p' into q', then $\mathcal{A}'$ also discriminates p' into q'.

Proof. By noetherian induction on $(PosF(t), >)$.

- If $Succ(p) = \emptyset$, then $\forall s \in L, \forall p' \in Pos(s)$, $(p' > p \implies s(p') \in C)$. And since rhs's have no nested functions, $R^*_p(L) = R^*_{in,p}(L)$.
  We get $\mathcal{A}'$ by Theorem 2.14.
- Let $Succ(p) = \{p_1, \ldots, p_n\}$. Let us define :
  $R^*_{in,>p}(L) = \{s' \mid \exists s \in L,\ s \rightarrow^*_{[u_1,\ldots,u_n]} s'$ by an innermost strat., $\forall i\ (u_i > p)\}$

Let $s \in L$. Either a rewrite step is applied at position p, and the strategy is innermost only if we first normalize s below position p by an innermost derivation, or no rewrite step is applied at position p. And since no defined-function occurs along any branches between p and p_i:

$$R^*_{in,p}(L) = R^*_p[R^*_{in,>p}(L) \cap_{p_i \in Succ(p)} IRR_{p_i}(R)] \cup\ R^*_{in,>p}(L)$$

Now, note that $\forall i, j \in [1..n]$, $(i \neq j \implies p_i || p_j)$. Moreover rewrite steps at incomparable positions can be commuted. Then obviously:

$$R^*_{in,>p}(L) = R^*_{in,p_1}(\ldots(R^*_{in,p_n}(L)\ldots))$$

L is recognized by an automaton $\mathcal{A}$ that discriminates every $p' \in PosF(t)$ s.t. $p' \geq p$. For each i, $p_i > p$, then $\mathcal{A}$ discriminates every $p' \in PosF(t)$ s.t. $p' \geq p_i$. By induction hypothesis, $R^*_{in,p_n}(L)$ is recognized by an automaton $\mathcal{A}'_n$ that still discriminates p and every position p' s.t. $p' \geq p_i$, $i = 1, \ldots, n-1$, $R^*_{in,p_{n-1}}(R^*_{in,p_n}(L))$ is recognized by an automaton $\mathcal{A}'_{n-1}$ that still discriminates p and every position p' s.t. $p' \geq p_i$, $i = 1, \ldots, n-2$, $R^*_{in,p_1}(\ldots(R^*_{in,p_n}(L)\ldots))$ is recognized by an automaton $\mathcal{A}'_1$ that still discriminates p.

By Theorem 2.11, $IRR_{p_i}(R)$ is recognized by an automaton that discriminates every position $p' \in PosF(t)$ s.t. $p' \not> p_i$, then necessarily p. By lemma 2.5, $\cap_{p_i \in Succ(p)} IRR_{p_i}(R)$ is recognized by an automaton that discriminates p.

Therefore $R^*_{in,p_1}(\ldots(R^*_{in,p_n}(L))\ldots) \cap_{p_i \in Succ(p)} IRR_{p_i}(R)$ is recognized by an automaton that discriminates p, and from Theorem 2.14, so is $R^*_p[R^*_{in,p_1}(\ldots(R^*_{in,p_n}(L))\ldots) \cap_{p_i \in Succ(p)} IRR_{p_i}(R)]$. Moreover discrimination of positions $p' \not> p$ is preserved. Finally, by union, we obtain an automaton that discriminates p and preserves the discrimination of positions $p' \not> p$.

Theorem 3.3. *Let R be a rewrite system satisfying the restrictions given in the introduction, and E be the data-instances of a given linear term t.*

$$R^*_{in}(E) = \left| \begin{array}{ll} R^*_{in,\epsilon}(E) & \textit{if } t(\epsilon) \in F \\ \\ R^*_{in,p_1}(\ldots(R^*_{in,p_n}(E)\ldots)) & \textit{otherwise} \\ \textit{with } Succ(\epsilon) = \{p_1, \ldots, p_n\} & \end{array} \right.$$

and $R^*_{in}(E)$ is effectively recognized by an automaton.

Proof. We have two cases:

- If $t(\epsilon) \in F$, obviously $R^*_{in}(E) = R^*_{in,\epsilon}(E)$.
- If $t(\epsilon) \notin F$, $\forall i, j \in [1..n]$, $(i \neq j \implies p_i || p_j)$, and rewrite steps at incomparable positions can be commuted. Then $R^*_{in}(E) = R^*_{in,p_1}(\ldots(R^*_{in,p_n}(E)\ldots))$.

The automaton comes from Definition 2.6 and from applying Lemma 3.2 (several times in the second case).

Example 3. Let E the data-instances of $t = f(g(x),\ h(g(y)))$ and

$$R = \{f(x, y) \rightarrow y,\ h(x) \rightarrow s(x),\ g(x) \rightarrow x\}$$

$*$ will symbolize the data-terms that instantiate t.
$t(\epsilon) \in \mathcal{F}$, we so calculate $R^*_{in,\epsilon}(E)$ where $E = f(g(*),\ h(g(*)))$.
$R^*_{in,\epsilon}(E) = R^*_{\epsilon}[R^*_{in,1}(R^*_{in,2}(E)) \cap IRR_1(R) \cap IRR_2(R)] \cup R^*_{in,1}(R^*_{in,2}(E))$.

We have to compute $R^*_{in,2}(E)$.
$Succ(2) = \{2.1\}$
So, $R^*_{in,2}(E) = R^*_2[R^*_{in,2.1}(E) \cap IRR_{2.1}(R)] \cup R^*_{in,2.1}(E)$
where $R^*_{in,2.1}(E) = E \cup f(g(*), h(*))$.

$$
\begin{aligned}
R^*_{in,2}(E) &= R^*_2[f(g(*),\ h(*))] \cup R^*_{in,2.1}(E)\\
&= f(g(*),\ h(*)) \cup f(g(*),\ s(*)) \cup R^*_{in,2.1}(E) \text{ (denoted by E1).}
\end{aligned}
$$

Now, we can compute $R^*_{in,1}(R^*_{in,2}(E))$.
$Succ(1) = \emptyset$.

$$
\begin{aligned}
\text{So, } R^*_{in,1}(E1) &= R^*_1(E1)\\
&= E1 \cup f(*, h(*)) \cup f(*, s(*)) \cup f(*, h(g(*))) \text{ (denoted by E2).}
\end{aligned}
$$

$$
\begin{aligned}
R^*_{in,\epsilon}(E) &= R^*_{\epsilon}[E2 \cap IRR_1(R) \cap IRR_2(R)] \cup E2\\
&= R^*_{\epsilon}[f(*, s(*))] \cup E2\\
&= f(*, s(*)) \cup s(*) \cup E2
\end{aligned}
$$

Finally, we obtain $R^*_{in}(E) = E \cup f(g(*), h(*)) \cup f(g(*), s(*)) \cup f(*, h(*)) \cup f(*, s(*)) \cup f(*, h(g(*))) \cup s(*)$.

4 Innermost-Leftmost Descendants: $R^*_{ileft}(E)$

Definition 4.1. *Given a language L and a position p, let us define:*

$$
R^*_{ileft,p}(L) = \{t' \mid \exists t \in L,\ t \rightarrow^*_{[u_1,\ldots,u_n]} t' \text{ by an innermost}-\text{leftmost strategy, and } \forall i\ (u_i \geq p)\}
$$

Lemma 4.2. *Let R be a rewrite system satisfying the restrictions given in the introduction, and E be the data-instances of a given linear term t.*
Let $p \in PosF(t)$ and L be a language s.t. $L|_p = E|_p$, and that is recognized by an automaton $\mathcal{A}$ that discriminates every position $p' \in PosF(t) \mid p' \geq p$. Then,

$$
R^*_{ileft,p}(L) = R^*_p(L) \text{ if } Succ(p) = \emptyset
$$

Otherwise, let $Succ(p) = \{p_1, \ldots, p_n\}$ s.t. $p_1 \lhd \ldots \lhd p_n$, and in this case

$$
R^*_{ileft,p}(L) = \left|
\begin{array}{l}
R^*_p[R^*_{ileft,p_n}(\ldots(R^*_{ileft,p_1}(L) \cap IRR_{p_1}(R))\ldots) \cap IRR_{p_n}(R)]\\
\quad \cup R^*_{ileft,p_1}(L) \cup R^*_{ileft,p_2}(R^*_{ileft,p_1}(L) \cap IRR_{p_1}(R)) \cup \ldots\\
\qquad\qquad \ldots \cup\ R^*_{ileft,p_n}(\ldots(R^*_{in,p_1}(L) \cap IRR_{p_1}(R))\ldots)
\end{array}
\right.
$$

and $R^*_{ileft,p}(L)$ is recognized by an automaton $\mathcal{A}'$ s.t. if $p' \in \overline{Pos}(t)$, $p' \not\geq p$, and $\mathcal{A}$ discriminates p' into q', then $\mathcal{A}'$ also discriminates p' into q'.

Theorem 4.3. *Let E be the data-instances of a linear term t and let R a TRS satisfying the restrictions given in the introduction.*

$$
R^*_{ileft}(E) = \left|
\begin{array}{ll}
R^*_{ileft,\epsilon}(E) & \text{if } \epsilon \in PosF(t)\\
 & \\
R^*_{ileft,p_1}(E) \cup \ldots \cup & \\
R^*_{ileft,p_n}(\ldots(R^*_{ileft,p_1}(E) \cap IRR_{p_1})\ldots) & \text{otherwise}\\
\text{with } Succ(\epsilon) = \{p_1, \ldots, p_n\} \text{ s.t. } p_1 \lhd \ldots \lhd p_n &
\end{array}
\right.
$$

and $R^*_{ileft}(E)$ is effectively recognized by an automaton.

Proof. We have two cases:

- If $\epsilon \in PosF(t)$, obviously $R^*_{ileft}(E) = R^*_{ileft,\epsilon}(E)$.
- If $\epsilon \notin PosF(t)$, $\forall i,\ j\ s.t.\ p_i \lhd p_j$, innermost-leftmost descendants at position p_j can be computed after to have normalized those at position p_i. Then obviously, $R^*_{ileft}(E) = R^*_{ileft,p_1}(E) \cup \ldots \cup R^*_{ileft,p_n}(\ldots(R^*_{ileft,p_1}(E) \cap IRR_{p_1})\ldots)$

The automaton comes from Definition 2.6 and from applying Lemma 4.2.

Example 4. Let E the data-instances of $t = f(g(x),\ h(y))$ and

$$R = \{f(x,y) \to s(f(x,y)),\ h(x) \to s(x),\ g(x) \to x\}.$$

$*$ will symbolize the data-terms that instantiate t.
$t(\epsilon) \in \mathcal{F}$, we so calculate $R^*_{ileft,\epsilon}(E)$ where $E = f(g(*),\ h(*))$.

$$R^*_{ileft,\epsilon}(E) = R^*_\epsilon[R^*_{ileft,2}(R^*_{ileft,1}(E) \cap IRR_1(R)) \cap IRR_2(R)]$$
$$\cup R^*_{ileft,1}(E) \cup R^*_{ileft,2}(R^*_{ileft,1}(E) \cap IRR_1(R)).$$

We have to compute $R^*_{ileft,1}(E)$.
So, $R^*_{ileft,1}(E) = R^*_1(E)$
$= E \cup f(*,h(*))$
Now, we can compute $R^*_{ileft,2}(R^*_{ileft,1}(E) \cap IRR_1(R))$.
$R^*_{ileft,2}(f(*,h(*))) = f(*,s(*)) \cup f(*,h(*))$
So, $R^*_\epsilon(f(*,s(*))) = s^*(f(*,s(*)))$
Finally, we obtain $R^*_{ileft}(E) = s^*(f(*,s(*))) \cup E \cup f(*,h(*))$.

Remark : The current computation of R^*_p does not take any strategies into account. Since rhs's do not have nested defined-functions, R^*_p is automatically innermost, but not leftmost. This is why we assume in addition that rhs's have at most one defined-function. However it is possible to modify the computation of R^*_p, to take the leftmost strategy into account (see [13]). Then, Theorem 4.3 still holds even if right-hand-sides contain several defined-functions.

5 Outermost Descendants: $R^*_{out}(E)$

Example 5. Consider two constructors s, a, and let

$$R = \{f(s(x),\ s(y)) \to s(f(x,\ y)),\quad g(s(x)) \to s(g(x))\}$$

Let $t = f(g(x),\ g(y))$, hence the starting language is $E = f(g(s^*(a)),\ g(s^*(a)))$. Obviously the descendants of E are $R^*(E) = s^*(f(s^*(g(s^*(a))),\ s^*(g(s^*(a)))))$, whereas the outermost descendants are:

$$R^*_{out}(E) = s^*(f(s^*(g(s^*(a))),\ s^?(g(s^*(a))))) \cup s^*(f(s^?(g(s^*(a))),\ s^*(g(s^*(a)))))$$

where $s^?$ means zero or one occurrence of s.

Surprisingly, to compute $R^*_{out}(E)$, we first reduce innermost positions 1 and 2 by computing $R^*_2(R^*_1(E)) = f(s^*(g(s^*(a))),\ s^*(g(s^*(a))))$. Now we reduce f

(position ϵ in E), however we keep only the descendants s.t. f cannot be reduced any more. Let $R^!_\epsilon$ denote this local normalization. We get :
$R^!_\epsilon[R^*_2(R^*_1(E))] = s^*(f(s^*(g(s^*(a))),\, g(s^*(a)))) \cup s^*(f(g(s^*(a)),\, s^*(g(s^*(a)))))$
which is a subset of $R^*_{out}(E)$.

Some outermost descendants are missing because the normalization of f is too harsh. To define the right one, we introduce an outermost abstract rewriting, that selects exactly the outermost descendants among the elements of $R^*(E)$. It is composed of an outermost narrowing step, a subterm decomposition, and an approximation. Subterm decomposition and approximation are just for making abstract rewriting terminate. The following definitions are illustrated by Example 6.

Notations : $\rightsquigarrow$ denotes the narrowing relation. Given a TRS R, $t \rightsquigarrow_{[p]}$ means that there exists a narrowing step at position p issued from t. Because of the restrictions, every considered term is linear : variable names do not matter, and for readability, we will often replace variables by the anonymous variable $\perp$.

Definition 5.1. *Let T_2 be the set of terms (with variables) s.t. $t(\epsilon) \in F$ and every branch of t contains at most two nested defined-functions. Given a TRS R, let us define on T_2 a binary relation $\approx$, to approximate terms. Let $t, t' \in T_2$, and for t let $Succ(\epsilon) = \{p_1, \dots, p_n\}$. For each $i \in \{1, \dots, n\}$, let$t_i = t[\perp]_{p_j, \forall j \neq i}$, and let $t_0 = t[\perp]_{p_j, \forall j}$. Then $t \approx t'$ iff*

1. $t' = t \,\wedge\, (1)$ *where* $(1) = (t \rightsquigarrow_{[\epsilon]} \,\vee Succ(\epsilon) = \emptyset)$
2. *or* $t' = t_0 \,\wedge\, \neg(1) \,\wedge\, t_0 \not\rightsquigarrow_{[\epsilon]}$
3. *or* $((t' = t_1 \,\wedge\, t_1 \not\rightsquigarrow_{[\epsilon]}) \vee \dots \vee (t' = t_n \,\wedge\, t_n \not\rightsquigarrow_{[\epsilon]})) \,\wedge\, \neg(1) \,\wedge\, t_0 \rightsquigarrow_{[\epsilon]}$

Remark : For a given t, several t' may exist in case 3, and there is at least one because $t_1 \rightsquigarrow_{[\epsilon]} \,\wedge \dots \wedge t_n \rightsquigarrow_{[\epsilon]}$ implies $t \rightsquigarrow_{[\epsilon]}$ (thanks to constructor discipline), i.e. condition (1) is true.
Comments : In case 1, t can be narrowed at position ϵ, or t does not contain nested defined-functions. Therefore a step $t \rightsquigarrow_{[\epsilon]}$ (if any) is outermost : we keep t and will try to narrow it at ϵ.
Case 2 means that t cannot be narrowed at ϵ, even if inner positions are first narrowed. Therefore, any narrowing step at an inner position is outermost. Note that a term obtained by narrowing steps at inner positions is an instance of t_0. Thus, the descendants of $t\theta$ that are instances of t_0 are outermost descendants : we keep t_0 instead of t.
Case 3 means that t cannot be narrowed at ϵ, but could be narrowed at ϵ if inner positions are first modified (narrowed). Suppose $t_i \not\rightsquigarrow_{[\epsilon]}$. Then every derivation $t \rightsquigarrow^*_{[p_{j_1}, \dots, p_{j_l}]} s$ s.t. $\forall j_k$, $p_{j_k} \neq p_i$ is outermost, s is an instance of t_i, and $s \not\rightsquigarrow_{[\epsilon]}$: we keep t_i instead of t, and will try to narrow it at p_i.

Definition 5.2. *(outermost abstract rewriting)*
Let $t, t' \in T_2$. $t \hookrightarrow_{[p, l \to r]} t'$ if there exists a term s s.t. $t \rightsquigarrow_{[p, l \to r, \theta]} s$, and
- $p = \epsilon \,\wedge\, \exists u \in PosF(r) \mid s|_u \approx t'$,

- *or* $t \not\rightsquigarrow_{[\epsilon]}$ *and* $s \approx t'$.

Example 6. Let $R = \{h(c(x,y)) \to c(h(x), i(y)),\ \ i(s(x)) \to p(i(x))\}$ where c, s, p are constructors.
Since $h(h(\perp)) \not\rightsquigarrow_{[\epsilon]}$ and $h(h(\perp)) \rightsquigarrow_{[1]} h(c(h(\perp), i(\perp))) \approx_{case1} itself$, then $h(h(\perp)) \hookrightarrow h(c(h(\perp), i(\perp)))$.
Since $h(c(h(\perp), i(\perp))) \rightsquigarrow_{[\epsilon]} c(h(h(\perp)), i(i(\perp)))$ and subterms $h(h(\perp)) \approx_{case3} itself$ ($n = 1$ then $t_1 = t$) and $i(i(\perp)) \approx_{case3} itself$, then
$h(c(h(\perp), i(\perp))) \hookrightarrow h(h(\perp))$ and $h(c(h(\perp), i(\perp))) \hookrightarrow i(i(\perp))$.
Since $i(i(\perp)) \not\rightsquigarrow_{[\epsilon]}$ and $i(i(\perp)) \rightsquigarrow_{[1]} i(p(i(\perp))) \approx_{case2} i(p(\perp))$, then $i(i(\perp)) \hookrightarrow i(p(\perp))$.
Note that $i(p(\perp))$ does not narrow.

Lemma 5.3. *Let* $t \in T_2$*,* σ *be a data-substitution, and* $t\sigma \to_{[p,l\to r]} t'$ *be a rewrite step. If* $p = \epsilon$ *let* $u \in PosF(r)$*, otherwise let* $u = \epsilon$*. Then:*

$$\begin{array}{ll} & t\sigma \to_{[p,l\to r]} t' \text{ is outermost} \\ \iff & \exists s, s' \in T_2 \mid t \approx s \ \wedge\ s \hookrightarrow^{?}_{[p,l\to r]} s' \ \wedge\ t'|_u \text{ is an instance of } s' \end{array}$$

The previous lemma does not extend to several steps if rhs's may contain several defined-functions (resp. if R has critical pairs), due to the decomposition step included in abstract rewriting, which may cause a confusion between different subterms (resp. between terms obtained by applying different rewrite rules).

Example 7.

$$R = \{i(x,y) \to c(h(x), h(y)),\quad h(x) \to x,\quad g(s(x)) \to s(g(x))\}$$

Let $t = i(s(g(x)),\ g(y))$. Then $t \approx t \hookrightarrow_{[\epsilon]} h(s(g(\perp)))$. The step

$$i(s(g(s^*(a))),\ g(s^*(a))) \to_{[\epsilon]} t' = c(h(s(g(s^*(a)))),\ h(g(s^*(a))))$$

is outermost. Now $t' \to_{[2.1]} t''$ is not outermost whereas $t''|_2 = h(s(g(s^*(a))))$ is an instance of $h(s(g(\perp)))$.

Definition 5.4. *Let* E *be the data-instances of a given linear term* t*, and* $p \in PosF(t)$*. Assume* $Succ(p) \neq \emptyset$*.*
Let us define $(t|_p)_{abs} \in T_2$ *by: if* $Succ(Succ(p)) = \emptyset$ *(i.e.* $t|_p \in T_2$*), let* $(t|_p)_{abs} = t|_p$*, otherwise* $(t|_p)_{abs} = (t[\perp]_{p_j,\forall j})|_p$ *where* $Succ(Succ(p)) = \{p_1, \ldots, p_n\}$*. Note that* $t|_p$ *is an instance of* $(t|_p)_{abs}$*.*
Let $R^*_{abs,p}(E)$ *be the set of descendants of* $\{s \in T_2 \mid (t|_p)_{abs} \approx s\}$ *by outermost abstract rewriting.*
Let $\mathcal{L}(R^*_{abs,p}(E))$ *be the set of ground instances of elements of* $R^*_{abs,p}(E)$*.*

Example 8. Consider Example 5 again, and let $p = \epsilon$. Then $t_{abs} = t$, and $t \approx t_1 = f(g(\perp), \perp)$, and $t \approx t_2 = f(\perp, g(\perp))$. Now:

$$t_1 = f(g(\perp),\ \perp) \hookrightarrow_{[1]} f(s(g(\perp)),\ \perp) \hookrightarrow_{[\epsilon]} f(g(\perp),\ \perp) = t_1$$
$$t_2 = f(\perp,\ g(\perp)) \hookrightarrow_{[2]} f(\perp,\ s(g(\perp))) \hookrightarrow_{[\epsilon]} f(\perp,\ g(\perp)) = t_2$$

Therefore

$$R^*_{abs,\epsilon}(E) = \{t_2,\ f(\perp,\ s(g(\perp))),\ t_1,\ f(s(g(\perp)),\ \perp)\}$$

Now, if $R^!_\epsilon$ reduces f and keeps the descendants whose subterms headed by f are instances of elements of $R^*_{abs,\epsilon}(E)$, then even the missing descendants (when $s^?$ is exactly s) are obtained, and $R^!_\epsilon[R^*_2(R^*_1(E))] = R^*_{out}(E)$.

Lemma 5.5. *$R^*_{abs,p}(E)$ is finite. Then the set $\mathcal{L}(R^*_{abs,p}(E))$ of ground instances of elements of $R^*_{abs,p}(E)$ is effectively recognized by a tree automaton.*

Definition 5.6. *Let $p \in PosF(t)$ s.t. $Succ(p) \neq \emptyset$. $t \rightarrow^!_{[p,rhs's]} t'$ means that $t \rightarrow^*_{[p,rhs's]} t'$ (like in Definition 2.12, except that zero step is allowed), provided in addition*

$$\forall q \in P(t'),\ t'|_q \in \mathcal{L}(R^*_{abs,p}(E))$$

Given a language L, let $R^!_p(L) = \{t' \mid \exists t \in L, t \rightarrow^!_{[p,rhs's]} t'\}$.

Lemma 5.7. *If L is recognized by an automaton $\mathcal{A}$ that discriminates p, then $R^!_p(L)$ is recognized by an automaton $\mathcal{A}'$ s.t. if $p' \not> p$ and $\mathcal{A}$ discriminates p' into q', then $\mathcal{A}'$ also discriminates p' into q'.*

Definition 5.8.

$$t_0 \rightarrow_{[p_0,l_0\rightarrow r_0]} t_1 \rightarrow \ldots t_n \rightarrow_{[p_n,l_n\rightarrow r_n]} t_{n+1} = t'$$

is outermost under a position p if

$$\forall i,\ p_i \geq p\ \wedge\ \forall q_i\,(p < q_i < p_i \implies t_i \not\rightarrow_{[q_i]})$$

Given a language L, $R^*_{out,p}(L)$ are the descendants of L outermost under p, using the TRS R, i.e. $R^*_{out,p}(L) = \{t' \mid \exists s \in L,\ s \rightarrow^* t'$ by a derivation outermost under $p\}$.

Lemma 5.9. *Let R be a rewrite system satisfying the restrictions given in the introduction, and E be the data-instances of a given linear term t.*
Let $p \in PosF(t)$, and L be a language s.t. $L|_p = E|_p$, and that is recognized by an automaton $\mathcal{A}$ that discriminates every position $p' \in PosF(t) \mid p' \geq p$. Then,

$$R^*_{out,p}(L) = R^*_p(L)\ if\ Succ(p) = \emptyset$$

Otherwise, let $Succ(p) = \{p_1, \ldots, p_n\}$, and in this case

$$R^*_{out,p}(L) = R^!_p[R^*_{out,p_1}(\ldots(R^*_{out,p_n}(L))\ldots)]$$

and $R^*_{out,p}(L)$ is recognized by an automaton $\mathcal{A}'$ s.t. if $p' \in \overline{Pos}(t)$, $p' \not> p$, and $\mathcal{A}$ discriminates p' into q', then $\mathcal{A}'$ also discriminates p' into q'.

Theorem 5.10. *Let R be a rewrite system satisfying the restrictions given in the introduction, and E be the data-instances of a given linear term t.*

$$R^*_{out}(E) = \left| \begin{array}{ll} R^*_{out,\epsilon}(E) & if\ t(\epsilon) \in F \\ & \\ \begin{array}{l} R^*_{out,p_1}(\ldots(R^*_{out,p_n}(E)\ldots)) \\ with\ Succ(\epsilon) = \{p_1, \ldots, p_n\} \end{array} & otherwise \end{array} \right.$$

and $R^*_{out}(E)$ is effectively recognized by an automaton.

6 Conclusion

When computing descendants, taking some strategies into account is possible, keeping the same class of tree language (the regular ones) and assuming the same restrictions (plus left-linearity). Does it also hold for any other computations of descendants ?

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. 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.
4. M. Dauchet, A.C. Caron, and J.L. Coquidé. Reduction Properties and Automata with Constraints. In *Journal of Symbolic Computation, 20:215-233*. 1995.
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. J. H. Gallier and R. V. Book. Reductions in tree replacement systems. *Theoretical Computer Science,*37:123-150, 1985.
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. T. Genet, F. Klay. Rewriting for Cryptographic Protocol Verification. *Technical report, CNET-France Telecom, 1999.* http://www.loria.fr/genet/Publications/GenetKlay-RR99.ps.
9. V. Gouranton, P. Réty, and H. Seidl. Synchronized Tree Languages Revisited and New Applications. In *Proceedings of FoSSaCs*, volume 2030 of LNCS, Springer-Verlag, 2001.
10. M. Hermann and R. Galbavý. Unification of Infinite Sets of Terms Schematized by Primal Grammars. *Theoretical Computer Science*, 176, 1997.
11. 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.
12. P. Réty. Regular Sets of Descendants for Constructor-based Rewrite Systems. In *Proceedings of the 6th international conference on LPAR, Tbilisi (Republic of Georgia)*, Lecture Notes in Artificial Intelligence. Springer-Verlag, 1999.
13. P. Réty. J. Vuotto. Regular Sets of Descendants by Leftmost Strategy. Research Report RR-2002-08 LIFO, 2002.
14. K. Salomaa. Deterministic Tree Pushdown Automata and Monadic Tree Rewriting Systems. *The journal of Computer and System Sciences,* 37:367-394, 1988.
15. T. Takai, Y. Kaji, and H. Seki. Right-linear Finite Path Overlapping Term Rewriting Systems Effectively Preserve Recognizability. In L. Bachmair, editor, *Proceedings 11th Conference RTA, Norwich (UK),* volume 1833 of LNCS, pages 246-260. Springer-Verlag, 2000.