URL: http://www.elsevier.nl/locate/entcs/volume70.html 20 pages

Regular Sets of Descendants by Leftmost Strategy

Pierre Réty [1,2] Julie Vuotto [1]

LIFO
Université d'Orléans
B.P. 6759, 45067 Orléans cedex 2, France

Keywords: term rewriting, strategy, tree automaton.

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 leftmost strategy.

1 Introduction

Because of potential applications to automated deduction and program validation (reachability, program testing), the problem of expressing some infinite sets of terms in a finite way, is essential. By means of tree automata, expressing 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 [1,2,3,4,7,8] [3]. Except [4,8], all those papers assume that the right-hand-sides (both sides when dealing with sets of equations) of rewrite rules are shallow [4], up to slight differences. On the other hand, P. Réty's work [5] does not always preserve recognizability (E is not arbitrary), but allows rewrite rules forbidden by the other papers [5].

Reduction strategies in rewriting and programming have drawn an increasing attention within the last years, and matter both from a theoretical point

[1] Email: {rety,vuotto}@lifo.univ-orleans.fr
[2] http://www.univ-orleans.fr/SCIENCES/LIFO/Members/rety/
[3] [4] computes sets of normalizable terms, which amounts to compute sets of descendants by orienting the rewrite rules in the opposite direction.
[4] Shallow means that every variable appears at depth at most one.
[5] Like $f(s(x)) \to s(f(x))$.

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 $t_1 \ {}^{st}\!\!\rightarrow^* t_2 \iff t_2 \in R^*_{st}(\{t_1\})$, 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, it can help with the static analysis of rewrite programs, and by extension, of functional programs.

In this paper, we compute leftmost descendants, whereas in [6] we compute innermost, innermost-leftmost and outermost descendants. Studying a lazy evaluation, like leftmost-outermost, is difficult, and for the moment we consider the leftmost case and the outermost case separately. Here, we build a finite automaton that can express the sets of leftmost descendants of E with respect to a constructor-based rewrite system, assuming:

(i) E is the set of ground constructor-instances (also called data-instances) of a given linear term t (i.e. $E = \{t\theta\}$).

(ii) Every rewrite rule is linear (both sides).

(iii) In right-hand-sides, there are no nested defined-functions, and arguments of defined-functions are either variables or ground terms.

(iv) Permutative rewrite rules are forbidden.

(v) Every rewrite rule is variable-preserving. By transforming R, Restriction v can be weakened into restriction v' : every rewrite rule is left-variable-preserving [6] .

It is shown in [5] that if any restriction among 1, 2, 3 is not satisfied, then the set of descendants (even if a leftmost strategy is respected) is not regular.

Compared to the innermost-leftmost descendants of [6], rhs's may contain several defined-functions, thanks to the new result given in Section 3. On the other hand, we assume additional Restrictions iv, v, because computing leftmost descendants is more difficult than computing only innermost-leftmost ones.

The paper is organized as follows. Section 3 gives the computation of the language $R^{*\lhd}_p(\{t\theta\})$ (i.e. terms obtained after rewriting eventually at position p of t plus possibly at functions positions issued from rhs's, according to a leftmost strategy). Leftmost descendants are computed in section 4 (we apply the previous language for every p position of t, by checking out before that sub-terms occurring on the left of p be normalized by intersection with an automaton that recognizes irreducible terms). Now, let us see, in the following section preliminaries notions. Subsection 2.1 (usual notions) may be skipped by the reader. Missing proofs are in the appendix.

[6] if $x \in Var(l) \cap Var(r)$ and $y \in Var(l) - Var(r)$, then x occurs in l on the left of y.

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, and $T_{C \cup F}$ the set of ground terms that contains constructors and functions. 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. $PosVar(t)$ is the set of variable positions of 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 $l = f(t_1, \ldots, t_n)$ 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 $p \in Pos(t)$, 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 *leftmost* if $\forall v$ occurring strictly on the left of p, $t|_v$ is irreducible.

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 $c(q_1, \ldots, q_n) \to q$ where $c \in C \cup F$ and $q_1, \ldots, q_n, q \in Q$, or of the form $q_1 \to q$ (*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*. 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 and Discrimination

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 [5] for proofs of Lemmas 2.2 and 2.4, and [6] for the proof of

Lemma 2.5.

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 $t \rightarrow_\Delta^* t[q']_{p'} \rightarrow_\Delta^* q_f$ where $q_f \in Q_f$, we have
  - $q' \rightarrow_\Delta^* q$ (i.e. by empty transitions) if $p' = p$,
  - $q' \neq q$ otherwise.

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

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

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

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

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* $(q_\mathcal{A}, q_\mathcal{B})$.

2.3 Positions and Antecedents

Given a term t, we define :

Definition 2.6 Let $p \in Pos(t)$. $Succ(p)$ are the nearest function positions below p :

$$Succ(p) = \{p' \in PosF(t) \mid p' > p \wedge \forall q \in Pos(t)\,(p < q < p' \Rightarrow q \notin PosF(t))\}$$

Definition 2.7 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 $i, i' \in \mathbb{N}$ and $i < i'$.

Definition 2.8 Let $t \rightarrow_{[q, l \rightarrow r]} t'$ be a rewrite step.
Let $v' \in Pos(t')$. v is an antecedent of v' in t (denoted by $ant(v', t)$) through this step if:

(i) $v \in Pos(t)$

(ii) - $v' \lhd q$ *and* $v = v'$ or,

- $\exists p' \in PosVar(r)$ *with* $r|_{p'} = x$ *s.t.* $v' = q.p'.w$ *and* $v = q.p''.w$ *where* p'' is a position of x in l.

Remark : Since lhs's are linear, the antecedent (if exists) is unique.

Definition 2.9 Let us consider the following derivation:

$$t_0 \rightarrow_{[p_0, l_0 \rightarrow r_0]} t_1 \rightarrow \ldots t_n \rightarrow_{[p_n, l_n \rightarrow r_n]} t_{n+1} \qquad (1)$$

$v_0 \in Pos(t_0)$ is an antecedent of $v_{n+1} \in Pos(t_{n+1})$ through this derivation if $\exists v_1 \in pos(t_1), \ldots, v_n \in Pos(t_n)$ s.t. $\forall i \in \{0, ..., n\}$, v_i is an antecedent of v_{i+1} through step $t_i \rightarrow t_{i+1}$.

2.4 Descendants and irreducible ground terms

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. $R^*_{left}(E)$ denotes the set of descendants of E, according to a leftmost strategy.

Definition 2.10 $t \rightarrow^+_{[p, rhs's]} t'$ 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 $t_1, \ldots, t_n$ and some sets of positions $P(t), P(t_1), \ldots, P(t_n)$ s.t.

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

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 2.11 Given a language E and a position p, we define $R_p^{*\lhd}(E)$ as follows

$$R_p^{*\lhd}(E) = E \cup \{t' \mid \exists t \in E, t \rightarrow^+_{[p, rhs's]} t' \text{ by } leftmost\ rewriting\}$$

Example 2.12 $R = \{f(x) \rightarrow s(x),\ g(x,y) \rightarrow c(h(x), f(y)),\ h(x) \rightarrow f(x)\}$
$R_1^{*\lhd}(\{f(g(a,b))\}) = \{f(g(a,b))\} \cup \{f(c(h(a), f(b)))\} \cup \{f(c(f(a), f(b)))\} \cup \{f(c(s(a), f(b)))\} \cup \{f(c(s(a), s(b)))\}$.

Definition 2.13 Let $IRR_p(R) = \{s \in T_{C \cup F} \mid p \in Pos(s) \wedge s|_p \text{ is irreducible}\}$

Theorem 2.14 *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. $p' \not> p$.*

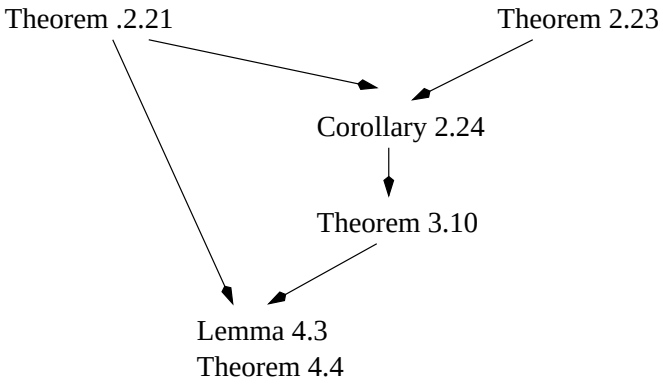
Proof. See [6].

Let $\mathcal{A}_{irr}$ be an automaton that recognizes $IRR_e(R)$. In the following, we will call q^{irr} the accepting state of $\mathcal{A}_{irr}$. □

2.5 Commutation of Rewriting and Left-Basic Derivation

In a leftmost strategy, before rewriting at some position p, the subterms occurring on the left of p must be rewritten into normal forms, by leftmost derivations too. However, in order to avoid a loop when building the automaton, we normalize these subterms by derivations without strategy, and we show that the normal forms obtained by leftmost derivations and by arbitrary derivations are the same. For this, we show that an arbitrary derivation can always be transformed into a left-basic derivation, and that a left-basic derivation leading to a normal form is leftmost.

The following figure shows the links between lemmas and theorems. Theorem 4.4 corresponds to the final result.



By the two following Lemmas, we see how to commute a derivation (in two steps, next in several steps).

Lemma 2.15 *Let R be a linear TRS and s, t, u be terms. If*

$$s \to_{[p,g\to d]} t \to_{[q,l\to r]} u \qquad (1)$$

is a derivation in two steps s.t. q admits an antecedant in s denoted by p_0. Then, (1) can be commuted into:

$$s \to_{[p_0,l\to r]} t' \to_{[p,g\to d]} u' \qquad (2)$$

Now, let us suppose restrictions v and iv. Then, if (1) is leftmost, (2) is leftmost.

Remark : *R being linear, $g \to d$ is linear and so d is linear, consequently $u' = u$, i.e, the last term of derivation is preserved by commutation [9]. Moreover, in (2), p does not have an antecedant in s.*

Lemma 2.16 *Let R be a linear TRS. If*

$$t_0 \rightarrow_{[p_0,l_0\rightarrow r_0]} t_1 \rightarrow \ldots t_{n-1} \rightarrow_{[pn-1,l_{n-1}\rightarrow r_{n-1}]} t_n \rightarrow_{[p_n,l_n\rightarrow r_n]} t_{n+1} \qquad (1)$$

is a derivation s.t. p_n admits an antecedant q_{n-1} in t_0. Then, (1) can be commuted in:

$$t_0 \rightarrow_{[q_{n-1},l_n\rightarrow r_n]} t'_1 \rightarrow_{[p_0,l_0\rightarrow r_0]} \ldots t'_n \rightarrow_{[p_{n-1},l_{n-1}\rightarrow r_{n-1}]} t'_{n+1} \qquad (2)$$

with $t'_{n+1} = t_{n+1}$ and p_0 has no antecedant in t_0.
Now, let us suppose restrictions v and iv. Then, if (1) is leftmost, (2) is leftmost too.

Now, let us define the notion of left-basic derivation.

Definition 2.17 Let us consider the following derivation:

$$t_0 \rightarrow_{[p_0,l_0\rightarrow r_0]} t_1 \rightarrow \ldots t_n \rightarrow_{[p_n,l_n\rightarrow r_n]} t_{n+1} \qquad (1)$$

This derivation is said to be left-basic if: it exists sets of positions $B(t_0)$, $B(t_1), \ldots, B(t_n)$ for the terms $t_0, t_1, \ldots, t_n$ s.t.

- $B(t_0) = PosF(t_0)$
- $\forall j,\ p_j \in B(t_j)$
- $\forall j,\ B(t_{j+1}) = \{p' \mid p' \leq p_j\} \cup \{p_j.v \mid v \in PosF(r_j)\} \cup \{p' \mid p_j \lhd p'\}$

Note that leftmost-innermost implies left-basic. The converse is false in the general case.

Counterexample 2.18
Let $R = \{f(x) \rightarrow s(f(x)),\ h(x,y) \rightarrow c(f(x), s(g(y))),\ g(x) \rightarrow x\}$ and $E = h(s^*(a), s^*(a))$. Consider the following derivation:

$$E \rightarrow_{[\epsilon]} c(f(s^*(a)), s(g(s^*(a)))) \rightarrow_{[2.1]} c(f(s^*(a)), s(s^*(a)))$$

This derivation is left-basic but is not leftmost-innermost because of the rewrite step at position 2.1 since position 1 is not normalized.

Lemma 2.19 *Let us consider the following derivation:*

$$t_0 \rightarrow_{[p_0,l_0\rightarrow r_0]} t_1 \rightarrow \ldots t_n \rightarrow_{[p_n,l_n\rightarrow r_n]} t_{n+1} \qquad (1)$$

then, (1) is left-basic if and only if $\forall i \in \{1, \ldots, n\}$, p_i has no antecedent in t_{i-1}.

Lemma 2.20 *Let us consider the following left-basic derivation:*

$$t_0 \rightarrow_{[p_0,l_0\rightarrow r_0]} t_1 \rightarrow \ldots t_n \qquad (1)$$

followed by the non-left-basic step:

$$t_n \rightarrow_{[p_n,l_n\rightarrow r_n]} t_{n+1}$$

Let us remark that p_n admits an antecedent in t_{n-1}. Let $j \leq n$ be the smallest integer s.t. p_n admits an antecedent in t_j.

Then, the following derivation obtained by commutation:

$$t_0 \rightarrow_{[p_0,l_0\rightarrow r_0]} t_1 \rightarrow \ldots t_j \rightarrow_{[q,l_n\rightarrow r_n]} t'_{j+1} \rightarrow_{[p_j,l_j\rightarrow r_j]} t'_{j+2} \rightarrow \ldots \ldots \rightarrow t'_n \rightarrow_{[p_{n-1},l_{n-1}\rightarrow r_{n-1}]} t_{n+1} \quad (2)$$

is left-basic.

Theorem 2.21 *Let R be a linear TRS. If $t_0 \rightarrow^* t_n$ (1) then, $t_0 \rightarrow^* t_n$ (2) by a left-basic derivation. Now, let us suppose restrictions v and iv. Then, if (1) is leftmost, (2) is leftmost too.*

Lemma 2.22 *Let R be a given constructor-based TRS and let us assume restriction v. Let us consider the following left-basic derivation:*

$$t_0 \rightarrow_{[p_0,l_0\rightarrow r_0]} t_1 \rightarrow \ldots t_n \rightarrow_{[p_n,l_n\rightarrow r_n]} t_{n+1} \quad (1)$$

and let $j \in \{0,\ldots,n\}$. If $\exists\ p' \in \overline{Pos}(t_j) - B(t_j)$ s.t. $t_j(p') \in \mathcal{F}$, then $\forall k > j$, $\exists p'_k \in \overline{Pos}(t_k) - B(t_k)$ s.t. $t_k|_{p'_k} = t_j|_{p'}$.

Theorem 2.23 *Let R be a given constructor-based TRS and let us assume restriction v. Let us consider the following left-basic derivation:*

$$t_0 \rightarrow^* t_{n+1} \ s.t.\ t_{n+1} = t_o\downarrow \quad (1)$$

Then, (1) is leftmost.

Corollary 2.24 *Let R be a given constructor-based TRS and let us assume restrictions ii and v. The normal forms of a given term t obtained by a leftmost rewrite strategy are the same as those obtained without rewrite strategy.*

Proof. Obviously, it follows from Theorem 2.21 and of Theorem 2.23. □

2.6 Starting Automaton

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

Definition 2.25 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 $\Delta_{data} = \{c(q_{data},\ldots,q_{data}) \rightarrow q_{data} \mid c \in C\}$.

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

$$Q_{t\theta} = \{q^p \mid p \in \overline{Pos}(t)\} \cup \{q_{data}\}$$

$$Q_{t\theta_f} = \{q^\epsilon\}\ (q_{data} \text{ if t is a variable})$$

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

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 $t|_p\theta$, $t|_p\theta \rightarrow^*_{[\Delta_{t\theta}]} q^p$ and $t|_p\theta \rightarrow^*_{[\Delta_{t\theta}]} q_{data}$.

3 Recognizing $R_p^{*\lhd}(E)$

Notions introduced in this section are illustrated by example 3.11.

It may occur that the matches used in rewrite steps instantiate the variables by languages not recognized into states of $\mathcal{A}_E$. I.e the instances are not always (sub)terms of E. Let us see the following example:

Example 3.1 Let $R = \{g(x) \stackrel{r1}{\rightarrow} h(x,b),\ h(x,y) \stackrel{r2}{\rightarrow} c(x,y)\}$ and E be the set of data-instances of $t = g(a)$.

$\mathcal{A}_{t\theta}$ can be summarized by writing : $\overset{q^\epsilon}{g}\ (\overset{q^1}{a})$ (which means that $g(a) \rightarrow_{\Delta_{t\theta}} g(q^1) \rightarrow_{\Delta_{t\theta}} q^\epsilon$). The rewrite steps issued from E are $g(a) \rightarrow_{[\epsilon,r1,x/a]} h(a,b) \rightarrow_{[\epsilon,r2,x/a\ y/b]} c(a,b)$. Unfortunately, $Q_{t\theta} = \{q^\epsilon,\ q^1\}$ and the language recognized into q^ϵ (resp. q^1) is $g(a)$ (resp. a).Thus, we do not have any states that can recognize $\{b\}$. This comes from the fact that $\{b\}$ is provided by the rhs $r1$. Therefore, we need to encode $\{b\}$ by additional states.
So, we give the following definition.

Definition 3.2 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.

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

Now, we define how to encode a version of each instantiated rhs.
Let $\mathcal{A} = (C \cup F, Q, Q_f, \Delta)$ an automaton that discriminates the position p into the state q, s.t. $Q \cap Q_{arg} = \emptyset$ (notation used in the continuation of this section). Let $Q' = Q \cup Q_{arg}$. We use states of the form d^p_σ where σ is a Q'-substitution, because rhs's may contain variables.

Definition 3.3 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

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

Thus, $r_i\sigma$ and only it, is recognized into the state $d^{i,\epsilon}_{\sigma}$.

Definition 3.4 Let $D^{sat} and \Delta^{sat}_d$ the set of states and transitions obtained as in Definition 3.3 by replacing each state $d^{i,p}_{\sigma} \in D$ by $d^{i,p}_{sat,\sigma}$.

The goal of these two similar encodings of rhs's is to recognize the instances of rhs's thanks to (D, Δ_d), and theirs descendants thanks to $(D^{sat}, \Delta^{sat}_d)$ and Δ^{sat} generated by the saturation process defined below:

Definition 3.5 (*saturation*)
Let Δ^{sat} be the set of transitions added in the following way: whenever there are $l_i \to r_i \in R$, a $(Q \cup Q_{arg})$-substitution σ s.t. $dom(\sigma) = Var(l_i) \cup Var(r_i)$ and $l_i\sigma \to^*_{\Delta_{t\theta}\cup\Delta_{arg}\cup\Delta^{sat}_d} q'$ where $q' \in \{q\} \cup D^{sat}$, add the transition $d^{i,\epsilon}_{sat,\sigma|_{Var(r_i)}} \to q'$.

Notation : $\Delta^{sat*}_d = \Delta^{sat}_d \cup \Delta^{sat}$.
Remark : Let us note $\mathcal{B}' = (C \cup F, Q \cup Q_{arg} \cup D^{sat}, \{q\}, \Delta \cup \Delta_{arg} \cup \Delta^{sat*}_d)$. Then, $L(\mathcal{B}') = R^*_{\epsilon}(L(\mathcal{A}|_p))$ i.e the same as $R^{*\lhd}_{\epsilon}(L(\mathcal{A}|_p))$ except that the rewrite steps are not necessarily leftmost (see [5] for more details and explanations, here only *sat* differs).

We create another rhs's encoding. So, it permits us to have descendants of instances of rhs's obtained by a leftmost strategy. For example, consider the rhs $c(f(x), g(y))$. We check that instances of $f(x)$ are reduced to their normal forms, by any strategy (thanks to Corollary 2.24), before reducing instances of $g(y)$ by a leftmost strategy.

Definition 3.6 For this, we consider the set of states:

$$D_{spec} = D \cup D^{sat} \cup D^{sat} \times Q_{irr}$$

and the following set of transitions where ($d^{\cdots}_{\ldots}q^{irr}$ denotes the pair $(d^{\cdots}_{\ldots}, q^{irr})$):

$$\Delta_{spec} = \bigcup_{l_i \to r_i \in R} \bigcup_{p \in Pos(r_i)/(PosF(r_i) \cup Arg(r_i))} \bigcup_{k \in \{1,\ldots,ar(r_i(p))\}}$$
$$\{r_i(p)(X_1, \ldots, X_n) \to d^{i,p}_{sat,\sigma_1 \cup \ldots \cup \sigma_n} \mid \forall j,\ \sigma_j \text{ is any } Q'\text{-substitution}$$
$$\text{s.t. } dom(\sigma_j) = Var(r_i|_{p.j}),$$
$$\text{where } \forall j,\ X_j = \left| \begin{array}{l|l} x\sigma_j, q^{irr} \text{ if } j < k & r_i(p.j) \text{ is any variable } x \\ x\sigma_j \text{ otherwise} & \end{array} \right.$$
$$\left| \begin{array}{l} d^{i,p.j}_{sat,\sigma_j} \text{ if } j = k \\ d^{i,p.j}_{sat,\sigma_j} q^{irr} \text{ if } j < k \\ d^{i,p.j}_{\sigma_j} \text{ otherwise} \end{array} \right. \}$$
$$\cup \{r_i(p)(X_1, \ldots, X_n) \to d^{i,p}_{sat,\sigma} \mid l_i \to r_i \in R,\ p \in PosF(r_i),$$
$$\sigma \text{ is any } Q'\text{-substitution s.t. } dom(\sigma) = Var(r_i|_p)$$
$$\text{where } \forall j,\ X_j = \left| \begin{array}{l} x\sigma \mid r_i(p.j) \text{ is any variable } x \\ q^{i,p.j} \in Q_{arg} \text{ otherwise} \end{array} \right. \}$$
$$\cup \{x\sigma \to d^{i,\epsilon}_{sat,\sigma} \mid l_i \to r_i \in R,\ r_i \text{ is any variable } x,$$
$$\sigma \text{ is any } Q'\text{-substitution s.t. } dom(\sigma) = \{x\}\}$$

Thus, $r_i\sigma$ is also recognized into the state $d^{i,\epsilon}_{sat,\sigma}$.

Now, we define an automaton that recognizes $R_p^{*\triangleleft}(L(\mathcal{A}))$.

Definition 3.7 We define $\mathcal{B}^{\triangleleft}$ an automaton s.t.:

$$\mathcal{B}^{\triangleleft} = (C \cup F, Q^{\triangleleft}, Q_f^{\triangleleft}, \Delta^{\triangleleft})$$

where $Q^{\triangleleft} = Q \cup D_{spec} \cup Q_{arg} \cup (Q \cup Q_{arg}) \times Q_{irr}$ and $Q_f = \{q\}$ and
$\Delta^{\triangleleft} = \Delta_d \cup \Delta_d^{sat*} \otimes \Delta_{irr} \cup \Delta_{spec} \cup \Delta^{sat} \cup \Delta_{arg}$.
$\Delta_d^{sat*} \otimes \Delta_{irr}$ is obtained by running the automaton intersection algorithm on transition sets Δ_d^{sat*} and Δ_{irr}. Thus it encodes normal-forms of instances of rhs's.

Lemma 3.8 $L(\mathcal{B}^{\triangleleft}) = R_{\epsilon}^{*\triangleleft}(L(\mathcal{A}|_p))$.

Proof. The proof follows from Corollary 2.24 and [5]. □

Corollary 3.9 $L(\mathcal{A}[\mathcal{B}^{\triangleleft}]_p) = R_p^{*\triangleleft}(L(\mathcal{A}))$.

Theorem 3.10 *Let R be a rewrite system satisfying restrictions 1,2,3,and 5, and E be the set of data-instances of a given linear term t. If E is recognized by an automaton that discriminates position p into the 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 $R_p^{*\triangleleft}(E)$.*

Example 3.11 Let $R = \{f(x) \stackrel{r1}{\to} s(x),\ h(x,y) \stackrel{r2}{\to} c(f(x),g(y)),\ g(x) \stackrel{r3}{\to} s(x)\}$ and $t = h(x,y)$. We consider only instances of t by constructors a, s, i.e. $E = \{h(s^*(a), s^*(a))\}$.
We are looking for an automaton that recognized $R^{*\triangleleft}_{\epsilon}(E)$. For sake of simplicity we denote by σ any substitution.
The only leftmost derivation is: $E \to_{[\epsilon,r2,\sigma]} c(f(s^*(a)), g(s^*(a))) \to_{[1,r1,\sigma]} c(s(s^*(a)), g(s^*(a))) \to_{[2,r3,\sigma]} c(s(s^*(a)), s(s^*(a)))$. In the following, we give only sets of transitions.

$\Delta_{t\theta} = \{a \to q_{data},\ s(q_{data}) \to q_{data},\ h(q_{data}, q_{data}) \to q^{\epsilon}\}$ where final state is q^{ϵ}. Let us define $\mathcal{B}^{\triangleleft}$.
$\Delta_d = \{c(d^{r2,1}_{\sigma}, d^{r2,2}_{\sigma}) \to d^{r2,\epsilon}_{\sigma},\ f(q_{data}) \to d^{r2,1}_{\sigma},\ g(q_{data}) \to d^{r2,2}_{\sigma},\ s(q_{data}) \to d^{r1,\epsilon}_{\sigma},\ s(q_{data}) \to d^{r3,\epsilon}_{\sigma}\}$.
$\Delta_{spec} = \{c(d^{r2,1}_{sat,\sigma}, d^{r2,2}_{\sigma}) \to d^{r2,\epsilon}_{sat,\sigma}, c(d^{r2,1}_{sat,\sigma} q^{irr}, d^{r2,2}_{sat,\sigma}) \to d^{r2,\epsilon}_{sat,\sigma},\ f(q_{data}) \to d^{r2,1}_{sat,\sigma}, g(q_{data}) \to d^{r2,2}_{sat,\sigma},\ s(q_{data}) \to d^{r1,\epsilon}_{sat,\sigma},\ s(q_{data}) \to d^{r3,\epsilon}_{sat,\sigma}\}$.
$\Delta^{sat}_d = \{c(d^{r2,1}_{sat,\sigma}, d^{r2,2}_{sat,\sigma}) \to d^{r2,\epsilon}_{sat,\sigma},\ f(q_{data}) \to d^{r2,1}_{sat,\sigma},\ g(q_{data}) \to d^{r2,2}_{sat,\sigma},\ s(q_{data}) \to d^{r1,\epsilon}_{sat,\sigma},\ s(q_{data}) \to d^{r3,\epsilon}_{sat,\sigma}\}$.
$\Delta^{sat} = \{d^{r2,\epsilon}_{sat,\sigma} \to q^{\epsilon},\ d^{r1,\epsilon}_{sat,\sigma} \to d^{r2,1}_{sat,\sigma},\ d^{r3,\epsilon}_{sat,\sigma} \to d^{r2,2}_{sat,\sigma}\}$.
For the following set, we give only what we will use: $\Delta^{sat*}_d \otimes \Delta_{irr} \supseteq \{s(q_{data}) \to d^{r3,\epsilon}_{sat,\sigma} q^{irr},$
$s(q_{data}) \to d^{r1,\epsilon}_{sat,\sigma} q^{irr},\ d^{r1,\epsilon}_{sat,\sigma} q^{irr} \to d^{r2,1}_{sat,\sigma} q^{irr}\}$.

Leftmost descendants are recognized indeed. In particular, the non-leftmost descendants $c(f(s^*(a)), s(s^*(a)))$ are not recognized because $s(s^*(a))$ is recognized into the state $d^{r2,2}_{sat,\sigma}$. This state appears in a lhs only in transition $c(d^{r2,1}_{sat,\sigma} q^{irr}, d^{r2,2}_{sat,\sigma}) \to d^{r2,\epsilon}_{sat,\sigma}$ of Δ_{spec} and in a transition of Δ^{sat}_d (but transitions of Δ^{sat}_d do not belong to the final set of transitions, see previous definition). And using $c(d^{r2,1}_{sat,\sigma} q^{irr}, d^{r2,2}_{sat,\sigma}) \to d^{r2,\epsilon}_{sat,\sigma}$ requires that the first argument is normalized, which does not hold for $f(s^*(a))$.

4 An automaton that recognizes $R^*_{left}(E)$

We look for an automaton that recognizes the sets of descendants according to a leftmost strategy.

Recall that R is assumed to satisfy Restriction 5' (left-variable preserving). We can transform it so that Restriction 5 (variable preserving) is satisfied, in the following way. A new binary constructor eat is introduced, and $eat(t, t')$ intuitively means that we want to keep t as the result, and to get rid of t'. Because of the leftmost strategy, the term to be kept has to be the left argument of eat. By introducing eat into the rhs, we can transform a rule which is not variable-preserving into a variable-preserving one. Then, by introducing more rewrite rules, we extend the existing defined-functions to take the new constructor eat into account. The method is explained by the following

example, and the algorithms are given in the appendix B.1.

Example 4.1 Let $R = \{f(s(s(x)), y) \to x\}$.
After running algorithm, we obtain the following TRS:
$R = \{f(s(s(x)), y) \to eat(x, y),\ f(eat(s(s(x)), x_1), y) \to eat(x, eat(x_1, y)),$
$f(s(eat(s(x), x_{1.1})), y) \to eat(x, eat(x_{1.1}, y)),\ f(eat(s(eat(s(x), x_{1.1.1})), x_1), y)$
$\to eat(x, eat(x_{1.1.1}, eat(x_1, y)))\}$.

Definition 4.2

$R^*_{left,p}(L) = \{s' \mid \exists s \in L,\ s \to^*_{[u_1,\ldots,u_n]} s'$ by a leftmost strat. with $\forall i,\ u_i \geq p\}$

Lemma 4.3 *Let R be a rewrite system satisfying restrictions 1 to 5, and E be the set of 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 that discriminates every position $p' \in PosF(t) \mid p' \geq p$. Then,

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

Otherwise, let $Succ(p) = \{p_1, \ldots, p_n\}$ with $p_1 \lhd \ldots \lhd p_n$

$$R^*_{left,p}(L) = \left| \begin{array}{l} R^{*\lhd}_p[R^*_{left,p_1}(L)] \\ \cup\ R^{*\lhd}_p[R^*_{left,p_2}((R^*_{left,p_1}(L) \cap IRR_{p_1}(R)))] \\ \cup \ldots \cup\ R^{*\lhd}_p[R^*_{left,p_n}(\ldots(R^*_{left,p_1}(L) \cap IRR_{p_1}(R)) \ldots \cap IRR_{p_{n-1}}(R))] \end{array} \right.$$

and $R^*_{left,p}(L)$ 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. See appendix B. □

Theorem 4.4 *Let R be a rewrite system satisfying restrictions 1 to 5, and E be the set of data-instances of a given linear term t.*

$$R^*_{left}(E) = R^*_{left,\epsilon}(E) \quad if\ t(\epsilon) \in F$$

Otherwise, let $Succ(p) = \{p_1, \ldots, p_n\}$ with $p_1 \lhd \ldots \lhd p_n$

$$R^*_{left}(E) = \left| \begin{array}{l} R^*_{left,p_1}(E) \\ \cup\ R^*_{left,p_2}((R^*_{left,p_1}(L) \cap IRR_{p_1}(R))) \\ \cup \ldots \cup R^*_{left,p_n}(\ldots(R^*_{left,p_1}(L) \cap IRR_{p_1}(R)) \ldots \cap IRR_{p_{n-1}}(R)) \end{array} \right.$$

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

Proof. See appendix B. □

To remove eat, we replace each transition of the form $eat(q, q') \to q''$ by $q \to q''$.

Example 4.5 Let $R = \{f(x) \to s(x),\ h(x, y) \to c(f(x), g(y)),\ g(x) \to s(x)\}$ and $E = \{h(g(s^*(a)), f(s^*(a)))\}$. For clarity, we denote $s^*(a)$ by $*$.
$R^*_{left}(E) = R^*_{left,\epsilon}(E) = R^{*\lhd}_\epsilon(R^*_{left,1}(E)) \cup R^{*\lhd}_\epsilon(R^*_{left,2}(R^*_{left,1}(E) \cap IRR_1(R))$.

- $R^*_{left,1}(E) = R_1^{*\lhd}(E)$
  $= E \cup \{h(s(*), f(*))\}$ denoted by L_1.
- $R^*_{left,2}(R^*_{left,1}(E) \cap IRR_1(R)) = R^*_{left,2}(\{h(s(*), f(*))\}$
  $= \{h(s(*), f(*)\} \cup \{h(s(*), s(*)\}$ denoted by L_2

Finally, we obtain leftmost descendants which are :
$L_1 \cup L_2 \cup \{c(f(g(*)), g(f(*)))\} \cup \{c(s(g(*)), g(f(*)))\} \cup \{c(s(s(*)), g(f(*)))\}$
$\cup \{c(s(s(*)), s(f(*)))\} \cup \{c(s(s(*)), s(s(*)))\} \cup \{c(f(s(*)), g(s(*)))\} \cup$
$\{c(s(s(*)), g(s(*)))\} \cup \{c(f(s(*)), g(f(*)))\}$.

5 Conclusion

When computing descendants, taking the leftmost strategy into account is possible, keeping the same class of tree languages (the regular ones) and assuming additional restrictions (Restrictions iv and v'). Restriction iv (no permutative rules) is the most annoying, but it cannot be weakened easily, because our method computes left-basic derivations, instead of computing leftmost derivations directly. However, we think that without this restriction, the set of leftmost descendants is still regular.

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] 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.

[3] 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.

[4] 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.

[5] P. Réty. Regular Sets of Descendants for Constructor-based Rewrite Systems. In *Proceedings of the 6th international conference on Logic for Programming and Automated Reasoning (LPAR), Tbilisi (Republic of Georgia)*, Lecture Notes in Artificial Intelligence. Springer-Verlag, 1999.

[6] P. Réty and J. Vuotto. Regular sets of descendants by some rewrite strategies. In *Proceedings of 13th Conference on Rewriting Techniques and Applications, Copenhagen (Denmark),* LNCS. Springer-Verlag, 2002.

[7] K. Salomaa. Deterministic Tree Pushdown Automata and Monadic Tree Rewriting Systems. *The Journal of Computer and System Sciences,* 37:367-394, 1988.

[8] 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.

[9] P. Réty. Méthodes d'unification par surreduction. Thèse de doctorat de l'université de Nancy 1.

Appendix

A Proofs on Rewriting Commutation and Left-basic Derivation

A.1 Proof of Lemma 2.15

The fact that p has no antecedent it's obvious because of Definition 2.8. Either p occurs on the right of p_0, either p occurs on top of p_0.
Now, let us show that (1) $leftmot \Rightarrow$ (2) $leftmost$.
Let (1) leftmost. Let us classify $Var(r)$ from left to right. if $|Var(r)| = n$ then $Var(r) = \{x_1, \ldots, x_n\}$. Since (1) is leftmost, let us denote by $x_i = x$ the first variable instantiated by a term containing a defined-function. Then, $\forall j \in [1..i[,\ \sigma(x_j)$ do not contain function.
Let us suppose that p_0 are not a leftmost position in s. This is possible only if $\exists$ a function that occurs on the left of x in l instantiated by a term containing a function. Now it happens that it is not possible because of prohibition of permutative rules (see restriction iv) and because of variable preserving TRS (see restriction v). Then, if we classify $Var(l)$ from left to right, we obtain the same order as $Var(r)$. Hence, (2) is leftmost.

A.2 Proof of Lemma 2.16

This proof follows from Lemma 2.15.
Let $ant(p_n, t_{n-1}) = q_0 \ \Rightarrow t_{n-1} \rightarrow_{[p_{n-1}, l_{n-1} \rightarrow r_{n-1}]} t_n \rightarrow_{[p_n, l_n \rightarrow r_n]} t_{n+1}$ commutes itself into $t_{n-1} \rightarrow_{[q_0, l_n \rightarrow r_n]} t'_n \rightarrow_{[p_{n-1}, l_{n-1} \rightarrow r_{n-1}]} t_{n+1}$. And p_{n-1} has not antecedent in t_{n-1}, and it is leftmost.
Let $ant(q_0, t_{n-2}) = q_1 \ \Rightarrow t_{n-2} \rightarrow_{[p_{n-2}, l_{n-2} \rightarrow r_{n-2}]} t_{n-1} \rightarrow_{[q_0, l_n \rightarrow r_n]} t'_n$ commutes itself into $t_{n-2} \rightarrow_{[q_1, l_n \rightarrow r_n]} t'_n \rightarrow_{[p_{n-2}, l_{n-2} \rightarrow r_{n-2}]} t'_n$. And p_{n-2} has not antecedent in t_{n-2}, and it is leftmost.
...
By induction, let $ant(q_{n-2}, t_0) = q_{n-1} \ \Rightarrow t_0 \rightarrow_{[p_0, l_0 \rightarrow r_0]} t_1 \rightarrow_{[q_{n-2}, l_n \rightarrow r_n]} t'_2$ commutes itself into $t_0 \rightarrow_{[q_{n-1}, l_n \rightarrow r_n]} t'_1 \rightarrow_{[p_0, l_0 \rightarrow r_0]} t'_2$. And p_0 has not antecedant in t_0 and it is leftmost.
Hence, we obtain (2).

A.3 Proof of Lemma 2.19

Let (1) be a left-basic derivation.
Let us suppose that $\exists i \in \{1 \ldots n\}$ s.t. p_i has an antecedent in t_{i-1}. By Definition 2.17, $p_i \in B(t_i)$ where $\forall j,\ B(t_i) = \{p' \mid p' \leq p_{i-1}\} \cup \{p_{i-1}.v \mid v \in PosF(r_{i-1})\} \cup \{p' \mid p_{i-1} \lhd p'\}$. Let $ant(p_i, t_{i-1}) = q$.
By Definition 2.8,

(i) $q \in Pos(t_{i-1})$

(ii) - $p_i \lhd p_{i-1}$ *and* $p_i = q$ or, (A)
- $\exists p' \in PosVar(r_{i-1})$ *with* $r_{i-1}|_{p'} = x$ *s.t.* $p_i = p_{i-1}.p'.w$ *and* $q = p_{i-1}.p''.w$ *where* p'' is an occurrence of variable x in l_{i-1}. (B)

It happens that (B) is impossible because of Definition 2.17; p_i would occur in forbidden position. (A) is impossible too, $not(p_i \lhd p_{i-1})$ else the derivation is not left-basic.
Hence, $\forall i \in \{1 \dots n\}$, p_i has not an antecedent in t_{i-1}.

Let $\forall i \in \{1 \dots n\}$, p_i has not an antecedent in t_{i-1}.
$\neg(p_i \lhd p_{i-1})$ else we find an antecedent, and $p_i \neq p_{i-1}.p'.w$ where $p' \in PosVar(r_{i-1})$ $(r_{i-1}|_{p'} = x)$ because else we find $q = p_{i-1}.p''.w$ where p'' is an occurrence of x in p_{i-1}.
Hence, it is left-basic.

A.4 Proof of Lemma 2.20

Let $ant(p_n, t_{n-1}) = q_0$. Derivation can be commuted and we obtain:
$t_0 \to t_1 \to \dots t_j \to t_{n-1} \to_{[q_0]} t'_n \to_{[p_{n-1}]} t_{n+1}$ where p_{n-1} has not antecedent in t_{n-1} according to Lemma 2.15.
By induction, let $j < n$ the smaller integer s.t. p_n admits an antecedent in t_j. Let denote by q this antecedent. By Lemma 2.16, we can commute and we obtain:
$t_0 \to \dots t_j \to_{[q, l_n \to r_n]} t'_{j+1} \to_{[p_j]} \dots \to t'_n \to_{[p_{n-1}]} t_{n+1}$ (4) and p_j has not antecedent in t_j.
q has not an antecedent in t_{j-1} since according to the remark $j < n$ is the smaller integer s.t. p_n admits an antecedent in t_j. Moreover, since (1) is left-basic, $t_0 \to \dots t_j$ is left-basic. And since $\forall i \in \{j+1 \dots n-1\}$ p_i has not antecedent in t_i for the derivation $t_j \to_{[q]} t'_{j+1} \to_{[p_j]} \dots \to t'_n \to_{[p_{n-1}]} t_{n+1}$ (3) then (3) is left-basic according to Lemma 2.19. And, since q has not antecedent in t_{j-1} then (4) is left-basic.

A.5 Proof of Lemma 2.21

Let $i \in \{1 \dots n\}$ s.t. $t_0 \to \dots t_i$ be a left-basic derivation and $t_0 \to \dots t_i \to t_{i+1}$ (1') be a non-left-basic.
By running of Lemma 2.20, (1') can be commuted in $t_0 \to \dots t_j \to_{[q]} t'_{j+1}$ $\to \dots t'_i \to t_{i+1}$ (2') with $ant(p_i, t_j) = q$ and (2') is left-basic. And by Lemma 2.16, (2') is leftmost if (1') is leftmost.
We use Lemma 2.20, many times are necessary, and so $t_0 \to^* t_n$ by a left-basic derivation. We proceed in the same way with Lemma 2.16, and (2) is leftmost if (1) is leftmost.

A.6 Proof of Lemma 2.22

Let $t_0 \rightarrow_{[p_0,l_0\rightarrow r_0]} t_1 \rightarrow \ldots t_n \rightarrow_{[p_n,l_n\rightarrow r_n]} t_{n+1}$ (1) be a left-basic derivation, and let $j \in \{0,\ldots,n\}$.
Let $p' \in \overline{Pos}(t_j) - B(t_j)$ s.t. $t_j(p') \in \mathcal{F}$. Let us take $k = j+1$. By absurd, let us suppose that $\neg(\exists p'_k \in \overline{Pos}(t_k) - B(t_k)$ s.t. $t_k|_{p'_k} = t_j|_{p'})$. I.e. $\forall p'_k \in \overline{Pos}(t_k) - B(t_k)$ s.t. $t_k|_{p'_k} \neq t_j|_{p'}$. Then, we have to remove $t_j|_{p'}$ during the rewriting step.

- if $p' \lhd p_j$ then this is impossible.
- if p' below p_j, $p' \in PosVar(l_j)$, seeing the constructor-based system, then the rewrite rule $l_j \rightarrow r_j$ should have to eliminate variables. It happens that we have a restrictions removing this model of rules.

By induction, we obtain the result for $k = j+2,\ldots,n$.

A.7 Proof of Theorem 2.23

By absurd, let us suppose that $\exists j, p'$ s.t. $t_j \rightarrow_{[p']} u$ with $p' \lhd p_j$. According to Lemma 2.22, $\nexists k > j$ s.t. $p_k = p'$. Then, $p' \notin B(t_{j+1})$ and $t_{j+1} \in \mathcal{F}$. According to Lemma 2.22, $\exists {p'}_{n+1} \in Pos(t_{n+1})$ s.t. $t_{n+1}|_{{p'}_{n+1}} = t_j|p'$. That contradicts the fact as t_{n+1} be normalized.

B Proofs on Computation of Leftmost Descendants

B.1 Transforming R to satisfy restriction v

Construct_r$(l \rightarrow r)$
  EraseVar $:= Var(l) - Var(r)$ *where* $|Var(l)| = n$ *and* $|Var(r)| = m$.
  $r := eat(r, x_{m+1})$
  For $i := m+2$ *to* n *do* $r := r[2 \leftarrow eat(r|_2, x_i)]$

In the following function, $Shift(stack, p)$ raises all positions which are under p in stack by a depth of 1. For example, if $p = 2$ and, 2.1, 2.2 and 1 are in stack, after running this function, 2.1.1, 2.1.2 and 1 are in stack.
$Shift(stack, p)$
  $\forall q \in stack$ *s.t.* $\exists w$ *and* $q = p.w$ *replace* q *by* $p.1.w$

Build_l$(l, stack, R')$
  $p := head(stack)$; $stack := pop(stack)$;
  if not empty$(stack)$ *then* $build(l, stack, R')$
  $l' := l[p \leftarrow eat(l|_p, x_p)]$; $r' := Construct\_r(l' \rightarrow r)$; *Add* $l' \rightarrow r'$ *to* R'
  $stack := Shift(stack, p)$; *if not empty*$(stack)$ *then* $Build(l', stack, R')$

$Transfom(R)$
  init : $R' = \emptyset$

$\forall l \to r \in R,\ if\ Var(l) = Var(r)\ then\ add\ l \to r\ to\ R'$
$if Var(l) \neq Var(r)\ then\ r' := Construct\_r(l \to r);$
$add\ l \to r'\ to\ R'$
let $C(l)$ stack that contains all constructor positions of l.
$if\ |C(l)| \neq \emptyset\ then\ Build(l, C(l), R')$
$R := R'$

Example B.1 Let us take again the TRS of example 4.1.

$Transform(f(s(s(x)), y) \to x)$

$init : R' = \emptyset$

$r' = Construct_r(l \to r) = eat(x, y);\quad R' = \{f(s(s(x)), y) \to eat(x, y)\}$

$C(l) = \{1, 1.1\}$

$Build(l, C(l), R')$

- $p = 1; stack = \{1.1\}$
- $Build(l, \{1.1\}, R')$
  - $p = 1.1; stack = \emptyset$
  - $l' = l[1.1 \leftarrow eat(l|_{1.1}, x_{1.1})];\ r' = Construct\_r(l' \to r)$
  - $add\ f(s(eat(s(x), x_{1.1})), y) \to eat(x, eat(x_{1.1}, y))\ to\ R'.$
- $l' = l[1 \leftarrow eat(l|_1, x_1)];\ r' = Construct\_r(l' \to r)$
- $add\ f(eat(s(s(x)), x_1), y) \to eat(x, eat(x_1, y))\ to\ R'$
- $stack = Shift(stack, 1) = \{1.1.1\}$
- $Build(l', \{1.1.1\}, R')$
  - $p = 1.1.1; stack = \emptyset$
  - $l'' = l'[1.1.1 \leftarrow eat(l'|_{1.1.1}, x_{1.1.1})];\ r' = Construct\_r(l'' \to r)$
  - $add\ f(eat(s(eat(s(x), x_{1.1.1})), x_1), y) \to eat(x, eat(x_{1.1.1}, eat(x_1, y)))\ to\ R'$

$R = R'$

B.2 Proof of Lemma 4.3

- If $Succ(p) = \emptyset$, then $\forall s \in L, \forall p' \in Pos(s), (p' > p \implies s(p') \in C)$. p is the only one function position of $s|_p$. In rhs's, it may have several function position that are incomparable (in opposition with nested). $R_p^{*\lhd}(L)$ compute leftmostly. Therefore, $R_p^{*\lhd}(L) = R^*_{left,p}(L)$.
  We get $\mathcal{A}'$ by Theorem 3.10.
- Let $Succ(p) = \{p_1, \ldots, p_n\}$ with $p_1 \lhd \ldots \lhd p_n$. Let $s \in L$. $\forall\ i \in \{1, \ldots, n\}$, s can be rewritten leftmostly at position p_i, but descendants at position p_{i-1}

must have computed and normalized at this position before, since $p_{i-1} \lhd p_i$.
Let us note L_1 set of terms obtained from s after leftmost rewriting at position p_1,..., L_n set of terms obtained from s after leftmost rewriting at position p_n. We have:
$L_1 = R^*_{left,p_1}(L)$
$L_2 = R^*_{left,p_2}(R^*_{left,p_1}(L) \cap IRR_{p_1}(R))$
...
$L_n = R^*_{left,p_n}(L_{n-1} \cap IRR_{p_{n-1}}(R))$.
Remark : Those descendants are obtained by left-basic rewriting.
$L_1, \ldots, L_n$ can be possibly rewritten at position p. $R^{*\lhd}_p(L_1) \cup \ldots \cup R^{*\lhd}_p(L_n) = R^*_{left,p}(L)$.
L is recognized by an automaton $\mathcal{A}$ that discriminates every position $p' \in PosF(t)$ s.t. $p' \geq p$ and so, since $\forall i \in \{1, \ldots, n\}$, $p_i > p$, every position p' s.t. $p' \geq p_i$.
By induction hypothesis, L_1 is recognized by an automaton $\mathcal{A}_1$ that discriminates p and every position $p' \in PosF(t)$s.t. $p' \not> p_1$ and so, in particular, every position $p' \geq p_i$ $\forall i \in \{2, \ldots, n\}$ and every position $p' \not> p$. By Theorem 3.10, $R^{*\lhd}_p(L_1)$ is recognized by an automaton that discriminates positions $p' \not> p$.
By Theorem 2.14, $IRR_{p_i}(R)$ is recognized by an automaton that discriminates every position $p' \in PosF(t)$ s.t. $p' \not> p_i$ ($p' \not> p_1$ for $IRR_{p_1}(R), \ldots,$ $p' \not> p_n$ for $IRR_{p_n}(R)$). For $j \in \{1, \ldots, n\}$, let us suppose that L_{j-1} is recognized by an automaton that discriminates every position $p' \not> p_{j-1}$ (i.e. positions that are discriminated before computing $R^*_{left,p_{j-1}}(\ldots)$ minus positions that are below p_{j-1}). By Lemma 2.5, $L_{j-1} \cap IRR_{p_{j-1}}(R)$ is recognized by an automaton that discriminates every position $p' \not> p_{j-1}$, so in particular $p' \geq p_j$. L_j is recognized by an automaton that discriminates every position $p' \not> p_j$ (i.e. positions that are discriminated before computing $R^*_{left,p_j}(\ldots)$ minus positions that are below p_j) and in particular every position $p' \not> p$. By Lemma 3.10, $R^{*\lhd}_p(L_j)$ is recognized by an automaton that discriminates every position $p' \not> p$.
Finally, by union, we obtain an automaton that discriminates p and preserves discrimination of positions $p' \not> p$.

B.3 Proof of Theorem 4.4

We have two cases:

- If $\epsilon \in PosF(t)$, obviously $R^*_{left}(E) = R^*_{left,\epsilon}(E)$.
- If $\epsilon \notin PosF(t)$, and $Succ(p) = \{p_1, \ldots, p_n\}$ *with* $p_1 \lhd \ldots \lhd p_n$, $\forall\ i, j$ s.t. $p_i \lhd p_j$, leftmost descendants at position p_j can be computed after have normalized those at position p_i. Then obviously,$R^*_{left}(E) = R^*left, p_1(E) \cup \ldots$
$\cup\ R^*left, p_{n-1}(\ldots(R^*left, p_1(L) \cap IRR_1(R)))\ldots) \cup \ldots$
$\cup\, R^*left, p_n(\ldots(R^*left, p_1(L) \cap IRR_1(R)))\ldots)$