Tree Automata for 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 finite tree automata that recognize the descendants of E, i.e. the terms issued from E by rewriting, according to innermost, outermost, leftmost, and innermost-leftmost strategies.

Key words: term rewriting, strategy, tree automaton.

⋆ This paper is the full and completed version of the conference paper (21) and the workshop paper (22).

Contents

1 Introduction

Finite tree automata have already been applied to many areas of computer science, and in particular to rewrite techniques (4). In comparison with more sophisticated refinements (1; 6; 18; 14; 13; 16), 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 has been investigated (3). Moreover, the related problem of expressing the set of descendants $R^*(E)$ of E with respect to a rewrite system R, has also been investigated (7; 17; 23; 5; 12; 15; 25; 24) [1] (and (2) for string rewriting). Unfortunately, it is undecidable whether a given rewrite system preserves regularity (also called recognizability) or not (11), and all previous papers define decidable subclasses. Except (15; 25; 24), they assume that the right-hand-sides (both sides when dealing with sets of equations) of rewrite rules are shallow [2], up to slight differences.

Réty's work (19) does not always preserve recognizability (E is not arbitrary), but allows rewrite rules forbidden by the other papers [3]. On the other hand, the possibility of computing a superset of the set of descendants, only assuming left-linearity, has been investigated in (9; 10).

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 $t_1 \stackrel{st}{\rightarrow}{}^* t_2 \iff t_2 \in R^*_{st}(\{t_1\})$, and st-joinability since $t_1 \stackrel{st}{\downarrow} t_2 \iff R^*_{st}(\{t_1\}) \cap R^*_{st}(\{t_2\}) \neq \emptyset$. More generally, it can help with the static analysis of rewrite programs, and by extension, of functional programs. For example, consider a functional language whose evaluation strategy is st, and consider a function f that sorts lists of elements. Let E be the set of all lists, we can check that the data st-descendants [4] of $f(E)$ are sorted lists indeed, by checking that the intersection with the set

[1] (15) 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 $f(s(x)) \rightarrow s(f(x))$, or when the left-hand-side is not linear.
[4] The data st-descendants can be computed as the intersection of the set of st-descendants and the set of data-terms.

of non-sorted lists [5] is empty. This paper is an extension of (19) 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 *constructor-based rewrite system* R, according to innermost, outermost, leftmost and innermost-leftmost strategies, assuming :

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

and, for outermost strategy :

4. There are no critical pairs between rules of R.

and, for leftmost strategy :

5. There are no permutative rewrite rules.

and, for leftmost and innermost-leftmost strategies :

6. Every rewrite rule is variable-preserving. However, by transforming R, Restriction 6 can be weakened into Restriction 6' : every rewrite rule is left-variable-preserving [6] .

The first three restrictions are necessary to obtain regular languages (see counter-examples in (19)). The others are necessary (according to the strategy) so that the automaton we build recognizes exactly the set of descendants.

Note that the counter-examples of (19) still hold if one of the four strategies we consider, is used. However, there are examples where the set of descendants is not regular, whereas it is regular if a strategy is used.

Example 1.1 Let $R = \{f(s(x)) \to s(f(x)), a \to a\}$, and $E = \{(f\, s)^*(a)\}$. Then $R^*(E)$ is not regular because $R^*(E) \cap s^*f^*(a) = s^n f^n(a)$, whereas $R^*_{in}(E) = E$ is regular (in means innermost).

[5] On a finite domain, the set of non-sorted lists is a regular language.
[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

The reader familiar with term rewriting and tree automata may skip this subsection.

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, $PosF(t)$ denotes 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 *TRS (term 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 $t_1, \ldots, t_n$ contain only constructors and variables. 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' is a *normal-form* of t if $t \to_R^* t'$ and t' is irreducible. $t \to_{[p]} t'$ 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.

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*. $\mathcal{A}$ is *deterministic* if whenever $t \to_\Delta^* q$ and $t \to_\Delta^* q'$ we have $q = q'$. A set E of ground terms is regular if there exists a finite automaton $\mathcal{A}$ s.t. $E = L(\mathcal{A})$. For a unary symbol

$s \in C$, s^* will denote arbitrarily many (possibly zero) occurrences of s.

2.2 Specific Notations

For a set of states Q, a *Q-substitution* σ is a substitution s.t. $\forall x \in dom(\sigma)$, $x\sigma \in Q$.

Given a term t, we define moreover :

Definition 2.1 *Let $p \in Pos(t)$. $Succ_t(p)$ are the nearest function positions below p :*

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

Definition 2.2 *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 $i < i'$.*

2.3 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 us to modify the behavior of $\mathcal{A}$ below position p without modifying the other positions, by replacing all transitions used below position p by those of another automaton $\mathcal{A}'$.

Definition 2.3 *Let us consider the derivation $t_0 \rightarrow_\Delta^* t_n$ (1).*
The sub-derivation $t_i \rightarrow_\Delta^* t_j$ of (1) composed of empty-transitions is length-max if :

$$\neg\,(t_{i-1} \rightarrow t_i \text{ via an } \epsilon\text{-transition}) \wedge \neg\,(t_j \rightarrow t_{j+1} \text{ via an } \epsilon\text{-transition})$$

Definition 2.4 *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 $t \rightarrow_\Delta^* q_f$ (1) where $q_f \in Q_f$, and for each sub-derivation $t[q_1]_{p'} \rightarrow_\Delta^* t[q_n]_{p'}$ of (1) composed of empty transition and length-max, we have*
  - *$q_n = q$ if $p' = p$,*
  - *$\forall i \in \{1 \ldots n\}$, $(q_i \neq q)$ otherwise.*

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

Lemma 2.5 $L(\mathcal{A}|_p) = \{t|_p \mid t \in L(\mathcal{A})\}$.

Proof: Let $\mathcal{A}$ be an automaton s.t. $\mathcal{A} = (C \cup F, Q, Q_f, \Delta)$ discriminates p into $q \in Q$.

- Let $t \in L(\mathcal{A})$. By Definition 2.4, there exists a derivation : $t \to_\Delta^* t[q]_p \to_\Delta^* q_f$ where $q_f \in Q_f$.
  Then, $t|_p \to_\Delta^* q$, and finally $t|_p \in L(\mathcal{A}_p)$. So, It is complete.
- Let $s \in L(\mathcal{A}|_p)$ then $s \to_\Delta^* q$. Since $L(\mathcal{A}) \neq \emptyset$, let $t' \in L(\mathcal{A})$.
  By Definition 2.4, there exists a derivation : $t' \to_\Delta^* t'[q]_p \to_\Delta^* q_f$.
  Thus, $t = t'[p \leftarrow s] \to_\Delta^* t'[q]_p \to_\Delta^* q_f$. Then $t \in L(\mathcal{A})$ and $t|_p = s$. So, it is correct.

Definition 2.6 *Let* $\mathcal{A} = (C \cup F, Q, Q_f, \Delta)$ *be an automaton that discriminates position* p *into state* q*, and let* $\mathcal{A}' = (C \cup F, Q', Q'_f, \Delta')$ *s.t.* $Q \cap Q' = \emptyset$ *and* $L(\mathcal{A}') \neq \emptyset$*. We define*

$$\mathcal{A}[\mathcal{A}']_p = (C \cup F,\ Q \cup Q',\ Q_f,\ \Delta'')$$

$$\text{where } \Delta'' = \Delta \setminus \{l \to q\} \cup \Delta' \cup \{q'_f \to q \mid q'_f \in Q'_f\}$$

Lemma 2.7 $L(\mathcal{A}[\mathcal{A}']_p) = \{t[t']_p \mid t \in L(\mathcal{A}),\ t' \in L(\mathcal{A}')\}$*, 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'*.*

Proof: By construction of $\mathcal{A}[\mathcal{A}']_p$ (see Definition 2.6), transition rules of Δ used above p are not changed and transition rules of Δ' are not changed either. Moreover $Q \cap Q' = \emptyset$, and the only transitions that mix states of Q and states of Q' are of the form $q'_f \to q$. Then the last sub-item of Definition 2.4 is satisfied. On the other hand, if $s \to^* q_f$ is a successful derivation of $\mathcal{A}[\mathcal{A}']_p$, it is of the form $s \to_{\Delta'}^* s[q'_f]_p \to s[q]_p \to_\Delta^* q_f$. Then the last-but-one sub-item of Definition 2.4 is satisfied. Therefore $\mathcal{A}[\mathcal{A}']_p$ still discriminates p into q.

- Let $t \in L(\mathcal{A})$ and $t' \in L(\mathcal{A}')$. By Definition 2.4, there exists a derivation : $t \to_\Delta^* t[q]_p \to_\Delta^* q_f$ where $q_f \in Q_f$.
  Moreover, since $t' \in L(\mathcal{A}')$, $t' \to_{\Delta'}^* q'_f$ where $q'_f \in Q'_f$.
  Then, $t[t']_p \to_{\Delta'}^* t[q'_f]_p$.
  According to Definition 2.6, $q'_f \to q \in \Delta''$. Then, $t[q'_f]_p \to t[q]_p \to_\Delta^* q_f$.
  Finally, $t[t']_p \in L(\mathcal{A}[\mathcal{A}']_p)$. So, it is complete.
- Let $s \in L(\mathcal{A}[\mathcal{A}']_p)$. Since $\mathcal{A}[\mathcal{A}']_p$ discriminates p into q, by Definition 2.4, there exists : $s \to^* s[q]_p \to^* q_f$ where $q_f \in Q_f$.
  - According to Definition 2.6, $q'_f \to q \in \Delta''$ and the only transitions

reaching q are of the form $q'_f \to q$. Then, $s \to^* s[q'_f]_p \to s[q]_p \to^* q_f$. Moreover, there are no transition rules that reduce states of Q into q'_f. Then, more precisely, $s \to^*_{\Delta'} s[q'_f]_p \to s[q]_p \to^* q_f$. And so, $s|_p \to^*_{\Delta'} q'_f$ i.e. $s|_p \in L(\mathcal{A}')$.

- On the other hand, suppose that $s[q]_p \to^* q_f$ uses transition rule(s) from Δ'. Then, it would have to eliminate states of Q' by means of transition $q'_f \to q$. According to Definition 2.4, $\neg\,(s[q]_p \to^* q_f$ via ϵ -transition at position $p)$, thus applying this transition would be at position different of p. Then, $s[q]_p \to^* . \to^*_{\Delta'} s'[q'_f]_{p'} \to s'[q]_{p'}$ with $p' \neq p$. Now, this is impossible because $\mathcal{A}[\mathcal{A}']_p$ discriminates p into q. Then, $s[q]_p \to^*_\Delta q_f$.
  Since $\mathcal{A}$ discriminates p into q, there exists t' s.t. $t' \to^*_\Delta q$, Let us write $t = s[t']_p$, then $t \in L(\mathcal{A})$. And then we have, $s = t[s|_p]_p$ which is of the wanted form. So, it is correct.

Lemma 2.8 *Let $\mathcal{A}, B$ 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, $t \to^*_{\Delta_{\mathcal{A}\cap\mathcal{B}}} t[(q'_\mathcal{A}, q'_\mathcal{B})]_{p'} \to^* (qf_\mathcal{A}, qf_\mathcal{B})$ (1)
  for each sub-derivation of (1) $t[(q'_{1\mathcal{A}}, q'_{1\mathcal{B}})]_{p'} \to^*_{\Delta_{\mathcal{A}\cap\mathcal{B}}} t[(q'_{n\mathcal{A}}, q'_{n\mathcal{B}})]_{p'}$ verifying Definition 2.3:
  - if $p' = p$ then from discrimination of $\mathcal{A}$ and $\mathcal{B}$, $q'_{n\mathcal{A}} = q_\mathcal{A}$ and $q'_{n\mathcal{B}} = q_\mathcal{B}$
  - if $p' \neq p$ then from discrimination of $\mathcal{A}$ and $\mathcal{B}$, $\forall i \in \{1, \ldots, n\}$, $(q'_{i\mathcal{A}} \neq q_\mathcal{A})$ and $(q'_{i\mathcal{B}} \neq q_\mathcal{B})$.

2.4 Particular Automata

2.4.1 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 2.9 *We define the automaton $\mathcal{A}_{data}$ that recognizes the set of data-terms T(C):*

$$\mathcal{A}_{data} = (C, Q_{data}, Q_{data_f}, \Delta_{data})$$

$$\textit{where} \left| \begin{array}{l} Q_{data} = Q_{data_f} = \{q_{data}\} \textit{ and} \\ \Delta_{data} = \{c(q_{data}, \ldots, q_{data}) \to q_{data} \mid c \in C\}. \end{array} \right.$$

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

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

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

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

Note that $\mathcal{A}_{t\theta}$ discriminates each position p $\in \overline{Pos}(t)$ into q^p. On the other hand, $\mathcal{A}_{t\theta}$ is not deterministic, whenever 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 $t|_p\theta \to^*_{[\Delta_{t\theta}]} q_{data}$.

2.4.2 Irreducible ground Terms at Position p

Let us now define an automaton that recognizes the terms irreducible at positions $\geq p$.

Definition 2.10 *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 2.11 *Let $RED(R)$ be the language of reducible terms:*

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

Lemma 2.12 *(8) If R is left-linear, $RED(R)$ is a regular language. (An automaton that recognizes $RED(R)$ is given in Appendix A).*

Lemma 2.13 *$IRR_\epsilon(R) = \overline{RED(R)}$. 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 2.14 *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: Let $\mathcal{A}_{irr\epsilon} = (\mathcal{C} \cup \mathcal{F}, Q_\epsilon, Q_{\epsilon f}, \Delta_\epsilon)$ be an automaton that recognizes $IRR_\epsilon(R)$.
Let $p = p_1.\ldots.p_k$ with $p_1, \ldots, p_k \in \mathbb{N} - \{0\}$
and $\forall i\ p_i \leq Max_{f \in F \cup C}(ar(f))$
The length of position p is $length(p) = k$.
We define $\mathcal{A}_{irr}$ as follows :
$\mathcal{A}_{irr} = (C \cup F, Q_{irr}, Q_{irrf}, \Delta_{irr})$ where
$Q_{irr} = \{q_{any},\ q_{rec}\} \cup_{v<p} \{q^v\} \cup_{v \in \overline{Pos}(t) \setminus \{v' | v' \leq p\}} \{q^v_{any}\} \cup Q_\epsilon$
$Q_{irrf} = \{q^\epsilon\}$ and

$$\Delta_{irr} = \{s(S_1, \ldots, S_n) \to q^j \mid s \in F \cup C, ar(s) \geq p_{length(j)+1}$$
$$q^j \in Q_{irr},\ S_i = \left| \begin{array}{l} q^{j.i} \quad \text{if } j.i < p \\ q_{rec} \quad \text{if j.i=p} \\ q^{j.i}_{any} \ \text{otherwise} \end{array} \right\} if\ p \neq \epsilon$$
$$\cup \{q_f \to q_{rec} \mid q_f \in Q_{\epsilon f}\}\ if\ p \neq \epsilon$$
$$\cup \{q_f \to q^\epsilon \mid q_f \in Q_{\epsilon f}\}\ if\ p = \epsilon$$
$$\cup \{s(S_1, \ldots, S_n) \to q^j_{any} \mid s \in F \cup C,\ q^j_{any} \in Q_{irr},$$
$$S_i = \left| \begin{array}{l} q^{j.i}_{any} \text{ if } j.i \in \overline{Pos}(t) \\ q_{any} \text{ otherwise} \end{array} \right\}$$
$$\cup \{s(q_{any}, \ldots, q_{any}) \to q_{any} \mid s \in F \cup C\}$$
$$\cup \Delta_\epsilon$$

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

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

We have written $ar(s) \geq p_{length(j)+1}$ to ensure that $p \in Pos(t)$. For example, if $p = 1.2.1$ and $s(\ldots) \to q^1$, 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 $p' < p$).

2.5 Descendants

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

Definition 2.15 *$t \rightarrow^{+}_{[p,rhs's]} t'$ 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 2.16 *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 2.17 Let $R = \{f(x) \rightarrow s(x),\ g(x) \rightarrow s(h(x)),\ h(x) \rightarrow p(f(x))\}$ where $F = \{f, g, h\}$ and $C = \{s, a\}$. The symbols(s) that are eligible for rewriting, are underlined :

$$R^*_1(f(\underline{h}(g(a)))) = f(\underline{h}(g(a))) \cup f(p(\underline{f}(g(a)))) \cup f(p(s(g(a))))$$

An insight into the algorithm underlying the following result is given in Subsection 3.1 as an example, and a formal description is in Appendix B. The resultant automaton is different from the starting one only at positions below p, and in the general case, is built by nesting automata.

Theorem 2.18 *(19) Let R be a rewrite system satisfying Restrictions 1 to 3. 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 state q', then so is $R^*_p(E)$.*

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

3.1 Example

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

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

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

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

3.2 Algorithm

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, $R^*_{in,p}(L)$ is the set of innermost descendants of L over the TRS R, reducing positions below (or equal to) p, i.e.*

$$R^*_{in,p}(L) = \{s' \mid \exists s \in L,\ s \to^*_{[u_1,\ldots,u_n]} s' \text{ by an innermost strategy}, \forall i\ (u_i \geq p)\}$$

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

Lemma 3.2 *Let R be a constructor-based TRS satisfying Restrictions 1 to 3, 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,

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

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

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

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 $Succ_t(p) = \emptyset$, 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 2.18.
- Let $Succ_t(p) = \{p_1, \ldots, p_n\}$. Let us define:

$$R^*_{in,>p}(L) =$$

$$\{s' \mid \exists s \in L,\ s \to^*_{[u_1,\ldots,u_n]} s' \text{ by an innermost strat., } \forall i\ (u_i > p)\}$$

  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:

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

  Now, note that $\forall i, j \in \{1 \ldots n\}$, $(i \neq j \implies p_i || p_j)$. 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 2.14, $IRR_{p_i}(R)$ is recognized by an automaton that discriminates every position $p' \in PosF(t)$ s.t. $p' \not> p_i$, then necessarily p. By lemma 2.8, $\cap_{p_i \in Succ_t(p)} IRR_{p_i}(R)$ is recognized by an automaton that discriminates p.

  Therefore $R^*_{in,p_1}(\ldots(R^*_{in,p_n}(L))\ldots) \cap_{p_i \in Succ_t(p)} IRR_{p_i}(R)$ is recognized by an automaton that discriminates p, and from Theorem 2.18, so is $R^*_p[R^*_{in,p_1}(\ldots(R^*_{in,p_n}(L))\ldots) \cap_{p_i \in Succ_t(p)} IRR_{p_i}(R)]$. 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 constructor-based TRS satisfying the restrictions 1 to 3, and E be the data-instances of a given linear term t.*

$$R^*_{in}(E) = \begin{cases} R^*_{in,\epsilon}(E) & \text{if } t(\epsilon) \in F \\ R^*_{in,p_1}(\ldots(R^*_{in,p_n}(E)\ldots)) & \text{otherwise} \\ \text{with } Succ_t(\epsilon) = \{p_1,\ldots,p_n\} & \end{cases}$$

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 $t(\epsilon) \notin F$, $\forall i,j \in \{1\ldots n\}$, $(i \neq j \implies p_i || p_j)$, 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 2.9 and from applying Lemma 3.2 (several times in the second case).

Example 3.4 Let E be the data-instances of $t = f(g(x),\ h(g(y)))$ and

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

$$\text{where } F = \{f,\ g,\ h\} \text{ and } C = \{s,\ a\}$$

$*$ 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 $R^*_{in,2}(E)$.
$Succ_t(2) = \{2.1\}$
So, $R^*_{in,2}(E) = R^*_2[R^*_{in,2.1}(E) \cap IRR_{2.1}(R)] \cup R^*_{in,2.1}(E)$
where $R^*_{in,2.1}(E) = E \cup \{f(g(*), h(*))\}$.
$R^*_{in,2}(E) = R^*_2[f(g(*),\ h(*))] \cup R^*_{in,2.1}(E)$
$= \{f(g(*),\ h(*))\} \cup \{f(g(*),\ s(*))\} \cup R^*_{in,2.1}(E)$ (denoted by E1).
Now, we can compute $R^*_{in,1}(R^*_{in,2}(E))$.
$Succ_t(1) = \emptyset$.
So, $R^*_{in,1}(E1) = R^*_1(E1)$
$= E1 \cup \{f(*, h(*))\} \cup \{f(*, s(*))\} \cup \{f(*, h(g(*)))\}$ (denoted by E2).
$R^*_{in,\epsilon}(E) = R^*_{\epsilon}[E2 \cap IRR_1(R) \cap IRR_2(R)] \cup E2$
$= R^*_{\epsilon}[\{f(*, s(*))\}] \cup E2$
$= \{f(*, s(*))\} \cup \{s(*)\} \cup E2$
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 Outermost Descendants : $R^*_{Out}(E)$

This section is structured as follows : the method is introduced and explained thanks to a detailed example, which also informally gives the notions used by the algorithm. Then the algorithm is given (Subsection 4.2) as well as smaller examples. In particular, Counter-example 4.4 shows why the TRS must not have critical pairs. The notions are formally defined (which is very technical) after the algorithm, since the reader do not need to read technical details to understand it. The proofs are given in last.

As shown in the following example, the principle of the method is paradoxically to rewrite in a innermost way, to compute outermost descendants.

4.1 Example

Warning : To help the reader, a picture is given at the end of the example.

Let $E = \{g(f(s(a)))\}$ and $R = \{g(x) \to h(x),\ h(p(x)) \to g(x),\ f(s(x)) \to p(f(x))\}$, where $f, g, h \in F$ and $a, p, s \in C$. Obviously, the outermost descendants of E over R are : $E \cup \{h(f(s(a))),\ h(p(f(a))),\ g(f(a)),\ h(f(a))\}$.

Let i be a positive integer. In a term t, we say that a defined-function position p is at level i if there are i defined-function positions (including p), along the branch going from the root of t to position p. For example, if $t = s(f(p(g(a))))$, f occurs at level 1, and g occurs at level 2 (s, p are constructors).
$OutF_i(t)$ will denote the set of defined-function positions of t, at level i. $OutF_1(t)$ is sometimes abbreviated into $OutF(t)$.
In the following, we underline terms that are outermost descendants of E over R.

Paradoxically, to compute $R^*_{out}(E)$, we first rewrite at level 2. More precisely, we compute $E' = g[R^*_{out}(f(s(a)))]$. In this example, $f(s(a))$ has no nested defined-functions, then $E' = g[R^*(f(s(a)))] = \{\underline{g(f(s(a)))},\ g(p(f(a)))\}$.
Secondly, we rewrite E' at level 1. We obtain [7] :

$$E'' = E' \cup \{\underline{h(f(s(a)))},\ \underline{h(p(f(a)))},\ \underline{g(f(a))},\ \underline{h(f(a))}\}$$

Let us notice that $g(p(f(a)))$ is not an outermost descendant. We have to get rid of it. To do it, we mark symbols of E with M (and not rewrite rules)

[7] Here, it turns out that $E'' = R^*(E)$. It is not true if E has more than two nested defined-functions.

to locate in descendants, symbols coming from E. Therefore, the symbols that are not labeled with M come from rhs's of rewrite rules. Thus, let $E = \{g^M(f^M(s^M(a^M)))\}$. Then

$$E'' = \{\underline{g^M(f^M(s^M(a^M)))},\ g^M(p(f(a^M))),$$
$$\underline{h(f^M(s^M(a^M)))},\ \underline{h(p(f(a^M)))},\ \underline{g(f(a^M))},\ \underline{h(f(a^M))}\}$$

And we keep only terms satisfying the following condition (denoted by (I)) : $\forall p \in OutF_1(t)$

- $\neg\,(t \rightarrow_p)$,
- or all symbols of t occurring strictly below p are labeled with M.

Let t be the term of E, and $t'' \in E''$. Then t'' is a descendant of t : there exists a non-necessarily outermost derivation $t \rightarrow^* t''$ (2). Since R is linear, we can change the order of rewrite steps of (2), in the hope of getting a outermost derivation. We have shown (see correction proof) that if t'' satisfies (I) and R has no critical pairs, then we can get a outermost derivation from t to t'' in this way. Consequently t'' is an outermost descendant.

In the following, the set of terms satisfying (I) is denoted by $IRR'_{OutF}(R)$. Then, we keep only $E''' = E'' \cap IRR'_{OutF}(R) =$

$$\{\underline{g^M(f^M(s^M(a^M)))},\ \underline{h(f^M(s^M(a^M)))},\ \underline{h(f(a^M))}\}$$

Unfortunately, the outermost descendants $\underline{h(p(f(a^M)))}$ and $\underline{g(f(a^M))}$ are missing. Moreover, if we rewrite elements of E''' at level 1, we do not obtain them.

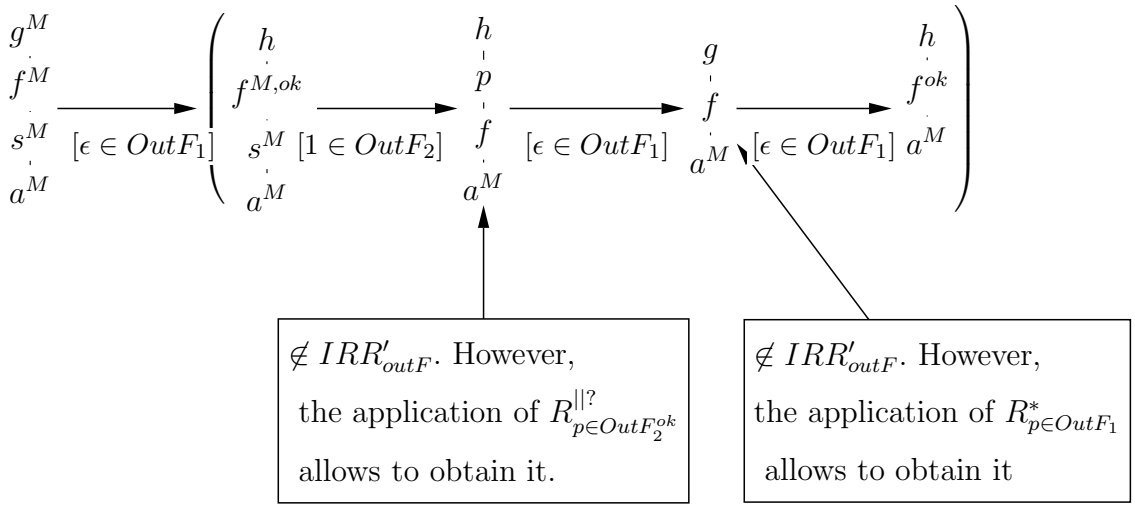
This is why we introduce the label ok, to indicate which defined-functions at level 2 can be reduced in one step, respecting the outermost strategy, i.e. the positions $q \in OutF_2(t)$ such that $\neg\,(\exists p \in OutF_1(t),\ (p < q \wedge t \rightarrow_p))$. Thus

$$E''' = \{\underline{g^M(f^M(s^M(a^M)))},\ \underline{h(f^{M,ok}(s^M(a^M)))},\ \underline{h(f^{ok}(a^M))}\}$$

Now, we rewrite positions labeled with ok in one step, and we obtain the missing term $\underline{h(p(f(a^M)))}$.
In the following, this one-step rewriting of ok-positions is denoted by $R^{\|?}_{p \in OutF_2^{ok}}$.

Let $E'''' = E''' \cup \{\underline{h(p(f(a^M)))}\}$. Unfortunately, $\underline{g(f(a^M))}$ is still missing. Now, we need to rewrite terms of E'''' at level 1 (which respects the outermost strategy) to get $R^*_{out}(E)$ exactly. We get $E'''' \cup \{\underline{g(f(a^M))}\} = R^*_{out}(E)$.

The picture below gives the outermost derivation.



Remark : In the general case, the scheme between parenthesis may occur several times (for example if the starting language is $g(f(s^*(a)))$).

4.2 Algorithm

Theorem 4.1 *Let R be a constructor-based TRS that satisfies restrictions 1 to 3 and 4. Let E be the data-instances of a given linear term t, and E^M be the set obtained from E by labeling every symbol with M. Let L be a language s.t. $L = E^M|_p$ for some $p \in Pos(t)$.*

If L contains only constructor-terms then $R^*_{out}(L) = L$
else if $L = \overset{c^M}{\underset{L_1 \quad L_n}{\cdot}}$ *where c^M is a constructor, then*

$$R^*_{out}(L) = \overset{c^M}{\underset{R^*_{out}(L_1) \quad R^*_{out}(L_n)}{\cdot}}$$

else $L = \overset{f^M}{\underset{L_1 \quad L_n}{\cdot}}$ *where $f^M \in F^M$ then*

$$R^*_{out}(L) = R^*_{p \in OutF_1}(R^{\|?}_{p \in OutF_2^{ok}}[\ (R^*_\epsilon(\overset{f^M}{\underset{R^*_{out}(L_1) \quad R^*_{out}(L_n)}{\cdot}}))^{ok} \cap\ IRR'_{OutF}(R)\ ])$$

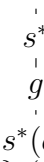
If $L = E^M|_p$ for $p \in PosVar(t)$ then L contains only constructor-terms. Consequently, recursivity of R^*_{out} terminates, and we can build and automaton

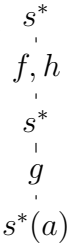
that recognizes $R^*_{out}(E^M)$ in this way. To get $R^*_{out}(E)$ we just have to remove labels.

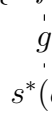
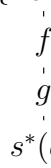
Remark : According to the definition of IRR'_{OutF}, if $L = f(L_1, \ldots, L_n)$ where $f \in F$ and $L_1, \ldots, L_n$ contain only constructors, then $R^*_{out}(L) = R^*_\epsilon(L)$.

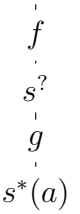
Notation : In the examples below, f, g as a node of a tree means that we can have either f or g at this position.

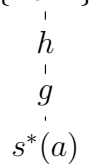
Example 4.2 M-labels are not necessary in this example. For simplicity, they are omitted. Let $R = \{f(s(x)) \to h(x),\ h(x) \to s(f(x)),\ g(x) \to s(g(x))\}$ and $E = \{f(g(s^*(a)))\}$.
The outermost descendants of E are: $\{s^*(f(s^?(g(s^*(a))))),\ s^*(h(g(s^*(a))))\}$.

step 1: $f(R^*_{out}(g(s^*(a)))) = \{$  $\}$ (denoted by L_1).

step 2: $R^*_\epsilon(L_1) = L_1 \cup \{$  $\}$ (denoted by L_2).

step 3: after intersection we obtain $\{$  $\} \cup \{$  $\}$ (denoted by L_3).

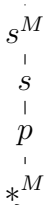
step 4: by applying $R^{\|}_{p \in OutF_2}$ on L_3, we obtain $L_3 \cup \{$  $\}$ (denoted by L_4).

step 5: by applying $R^*_{p \in outF_1}$ on L_4, we finally obtain $L_4 \cup \{$  $\}$.

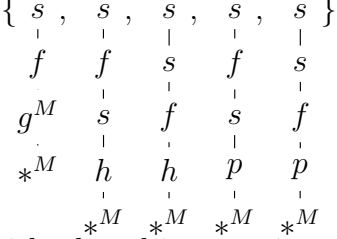
Example 4.3 In this example we see the usefulness of labels.
Let $R = \{f(s(x)) \to s(f(x)),\ g(x) \to s(h(x)),\ h(x) \to p(x)\}$ and let $E = \{f(s(g(s^*(a))))\}$.

The outermost descendants of E are:
$\{f(s(g(s^*(a)))),\ s(f(g(s^*(a)))),\ s(f(s(h(s^*(a))))),\ s(s(f(h(s^*(a))))),$
$s(s(f(p(s^*(a))))))\}$.
Below, $*^M$ will denote $s^{M*}(a^M)$.
Recall that the algorithm deals with innermost function at first.

step 1: $f^M(s^M(R^*_{out}(g^M(*^M)))) = \{f^M,\ f^M,\ f^M\}$ (denoted by L_1).



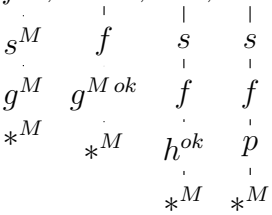
step 2: $R^*_\epsilon(L_1) = L_1 \cup \{$



$\}$

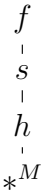
step 3: after labeling with ok and intersection, we obtain

$\{f^M,$



$\}$ (denoted by L_3).

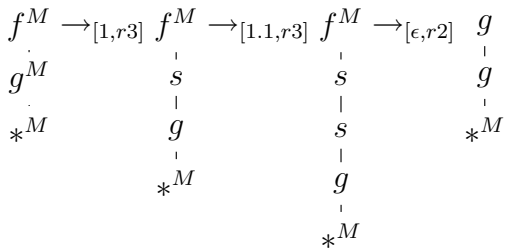
step 4: by applying $R^{\|}_{p\in OutF_2}$ on L_3, we obtain $L_3 \cup \{$



$\}$ (denoted by L_4)

step 5: $R^*_{p\in outF_1}$ applied on L_4 gives nothing else.

Counter-example 4.4 This counter-example shows that if R contains critical pairs (i.e. restriction 4 is not satisfied) the algorithm is not correct: it generates some non-outermost descendants.
Let $R = \{f(s(x)) \xrightarrow{r1} f(x),\ f(s(s(x))) \xrightarrow{r2} g(x),\ g(s(x)) \xrightarrow{r3} s(g(x))\}$ and consider the following derivation (as in the previous example, $*^M$ will denote $s^{M*}(a^M)$):

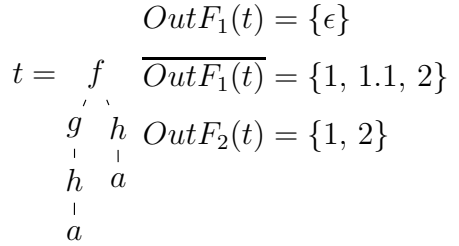


This derivation is not outermost, and we cannot change the order of rewrite steps. Actually, even using r_1, $g(g(*^M))$ cannot be reached by an outermost derivation: it is not an outermost descendant. However, $g(g(*^M))$ belongs to $IRR'_{OutF}(R)$, and then will be returned by the algorithm.

4.3 Formal Definitions

Definition 4.5 *Given a term t, $OutF_1(t)$ are the outermost elements of $PosF(t)$, and $OutF_2(t)$ are the outermost elements of $PosF(t) - OutF_1(t)$.*
$\overline{OutF_1(t)} = \{p \in PosF(t) \,|\, \neg\,(\exists\, q \in PosF(t)\ s.t.\ q < p)\}$
$\overline{OutF_1(t)} = PosF(t) - OutF_1(t)$
$OutF_2(t) = \{p \in PosF(t) \,|\, p \in \overline{OutF_1(t)} \wedge \neg\,(\exists\, q \in \overline{OutF_1(t)}\ s.t.\ q < p)\}$

Example 4.6 Let $F = \{f,\ g,\ h\}$



Remark : To each function f in F, we associate a new defined function f^{ok}.
Let F^{ok} be the set of functions labeled with ok.
Label "ok" will be used for locating some positions of $OutF_2$ that can be rewritten (in one step) respecting the outermost strategy.
Let $OutF_2^{ok}(t) = \{p \in Pos(t) \,|\, t(p) \in F^{ok}\}$

Definition 4.7 *Let us define an automaton $\mathcal{A}_{F-\epsilon} = (C \cup F \cup F^{ok},\ Q_{F-\epsilon},\ Q^f_{F-\epsilon},\ \Delta_{F-\epsilon})$ that describes the language of term t on $\Sigma = C \cup F \cup F^{ok}$ where at most one defined-function symbol of level 2 (i.e. $\in OutF_2(t)$) is labeled with ok. Formally:*
$L(\mathcal{A}_{F-\epsilon}) = \{t \,|\, t(\epsilon) \in PosF(t) \wedge \forall p \in Pos(t)\,((t(p) \in F^{ok}) \Rightarrow (p \in OutF_2(t) \wedge \forall q \neq p,\ t(q) \notin F^{ok}))\}$.
Let $Q_{F-\epsilon} = \{q_{any},\ q_{any-ok},\ q^{\epsilon}_{F-\epsilon}\}$ and $Q^f_{F-\epsilon} = \{q^{\epsilon}_{F-\epsilon}\}$.

$\Delta_{F-\epsilon}$ *is the following set of transitions :*

$$\begin{aligned}
&\{f(q_{any},\ldots,q_{any}) \to q^{\epsilon}_{F-\epsilon} \mid f \in F\}\\
\cup\ &\{s(q_{any},\ldots,q_{any}) \to q_{any} \mid s \in C \cup F\}\\
\cup\ &f(q_{any},\ldots,q_{any},q_{any-ok},q_{any},\ldots,q_{any}) \to q^{\epsilon}_{F-\epsilon} \mid f \in F\}\\
\cup\ &\{s(q_{any},\ldots,q_{any},q_{any-ok},q_{any},\ldots,q_{any}) \to q_{any_ok} \mid s \in C\}\\
\cup\ &\{f^{ok}(q_{any},\ldots,q_{any}) \to q_{any-ok} \mid f^{ok} \in F^{ok}\}
\end{aligned}$$

Definition 4.8 *Let $\mathcal{A}$ be an automaton. $\mathcal{A}^{ok}$ is the automaton obtained from $\mathcal{A}$ by adding label "ok" randomly on the defined-functions. This can be effected by the following transformation of the set of transitions Δ of $\mathcal{A}$:*
$\forall f \in F,\ \forall f(q_1,\ldots,q_n) \to q \in \Delta$, add $f^{ok}(q_1,\ldots,q_n) \to q$.
If L is the language recognized by $\mathcal{A}$, we denote by L^{ok} the language recognized by $\mathcal{A}^{ok}$.

Definition 4.9 *Consider again the automaton $\mathcal{A}_{irr\epsilon}$ that recognizes the set of terms that are not reducible at position ϵ, on the signature of R. Let $\mathcal{A}_{irr\_ok} = (\mathcal{A}_{irr\epsilon})^{ok}$.*

$IRR'_{OutF}(R)$ is the language of terms t s.t. each position $p \in OutF_1(t)$ satisfies : either $t \not\to_p$ or $t|_p$ comes from the starting language E. In the first case, we can reduce (in one step) one arbitrary $p' \in OutF_2(t)$ s.t. $p' > p$, respecting the outermost strategy. We locate such p' by labelling the symbol of t occurring at p' with *ok*. Moreover, in both cases, if $t \in IRR'_{OutF}(R)$ is a descendant of E obtained by a non-outermost derivation, then t can also be obtained from E by an outermost derivation (see correctness proof). In the second case, to check that $t|_p$ comes from E, we introduce another label M. If a symbol is labelled with M, this will mean that it comes from E.

Definition 4.10 *Let C^M (resp. F^M) be the set of constructors (resp. defined-functions) labelled with M.*
Let $\mathcal{A}_{irrF-\epsilon}$ be the automaton obtained by intersection between A_{irr_ok} and $\mathcal{A}_{F-\epsilon}$.
Let $\mathcal{A}'_{irrF-\epsilon} = (C \cup C^M \cup F \cup F^M \cup F^{ok},\ Q'_{irrF-\epsilon},\ Q'^{f}_{irrF-\epsilon},\ \Delta'_{irrF-\epsilon})$ be the automaton obtained by modifying $\mathcal{A}_{irrF-\epsilon}$ in order to add some labels M randomly on the symbols of recognized terms.

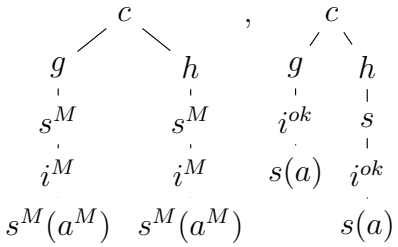
Definition 4.11 *Let us define the language :*
$IRR'_{OutF}(R) = \{t \mid \forall p \in OutF_1(t)(t \not\to_p \vee \forall u > p, t(u) \in C^M \cup F^M) \wedge \forall v \in Pos(t)\ ((v > p \wedge t(v) \in F^{ok}) \Rightarrow (v \in OutF_2(t) \wedge t \not\to_p \wedge \neg(\exists w \in PosF(t), w > p \wedge w \neq v \wedge t(w) \in F^{ok})))\}$,
which is recognized by the automaton :

$$\mathcal{A}_{IRR'_{OutF}} = (C \cup C^M \cup F \cup F^M \cup F^{ok},\ Q_{IRR'_{OutF}},\ Q^{f}_{IRR'_{OutF}},\ \Delta_{IRR'_{OutF}})$$

where $Q_{IRR'_{OutF}} = Q'_{irrF-\epsilon} \cup \{q_{out}, q_{out_M}, q_M, q^{\epsilon}_{irr'_{outF}}\}$ *and* $Q^f_{IRR'_{OutF}} = \{q^{\epsilon}_{irr'_{outF}}\}$, *and where* $\Delta_{IRR'_{OutF}}$ *is the following set of transitions:*

$$
\begin{aligned}
&\Delta'_{irrF-\epsilon}\\
\cup\ &\{q'^{\epsilon}_{irrF-\epsilon} \to q^{\epsilon}_{irr'_{outF}} \mid q'^{\epsilon}_{irrF-\epsilon} \in Q'^{f}_{irrF-\epsilon}\}\\
\cup\ &\{s(X_1, \ldots, X_n) \to q^{\epsilon}_{irr'_{outF}} \mid s \in C \cup C^M,\ \forall i\ X_i = q_{out} \cup q_{out_M}\}\\
\cup\ &\{s(X_1, \ldots, X_n) \to q_{out} \mid s \in C \cup C^M,\ \forall i\ X_i = q_{out} \cup q_{out_M}\}\\
\cup\ &\{q'^{\epsilon}_{irrF-\epsilon} \to q_{out} \mid q'^{\epsilon}_{irrF-\epsilon} \in Q'^{f}_{irrF-\epsilon}\}\\
\cup\ &\{f(q_M, \ldots, q_M) \to q_{out_M} \mid f \in F \cup F^M\}\\
\cup\ &\{s(q_M, \ldots, q_M) \to q_M \mid s \in C^M \cup F^M\}\\
\cup\ &\{q_{out_M} \to q^{\epsilon}_{irr'_{outF}}\}
\end{aligned}
$$

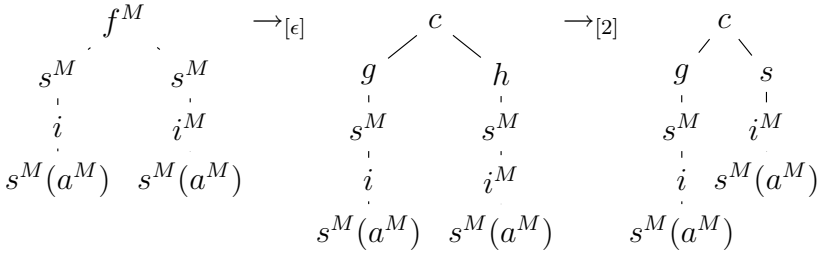
Example 4.12 Let $R = \{f(x, y) \to c(g(x),\ h(y)),\ g(s(x)) \to s(g(x)),\ h(s(s(x))) \to s(h(x)),\ i(s(x)) \to s(i(x))\}$.
The following terms are in $IRR'_{OutF}(R)$:



Definition 4.13 $R^{\|?}_{p \in OutF_2^{ok}}(t)$ *means parallel rewrite in at most one step at function positions identified by an ok label. By construction, these positions are in* $OutF_2(t)$.
Formally, $R^{\|?}_{p \in OutF_2^{ok}(t)} = \{t' \mid t \to^*_{[p_1,\ldots,p_n]} t' \wedge p_1, \ldots, p_n \in OutF_2^{ok}(t)\}$.
It consists in doing single saturation process on the TRS R^{ok} *where* $R^{ok} = \{l' \to r \mid l \to r \in R \wedge l' = l[\epsilon \leftarrow l(\epsilon)^{ok}]\}$.

Definition 4.14 *(labeled-term rewriting)*
Let s *be a term that may contain symbols labeled with* M. $s \to_{[p, l \to r, \sigma]} s'$ *if there exists a term* l^M *obtained from* l *by labeling some positions with* M, *s.t.* $s|_p = l^M\sigma$ *and* $s' = s[p \leftarrow r\sigma]$. *Note that* r *does not contain label* M *(because it does not come from the starting language* E*). However, if* r *is a variable (collapsing rewrite rule), the label of top symbol of* $r\sigma$ *has to be removed, in order to remember that a rewrite step has been done at position* p, *i.e. with respect to the starting language,* $s'|_p$ *has been modified.*

Example 4.15 Let $R = \{f(x,y) \to c(g(x), h(y)),\ \ h(x) \to x\}$.



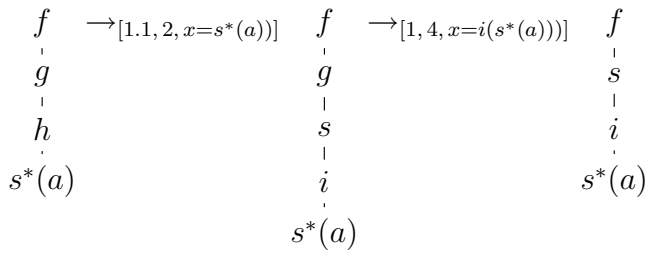
4.4 Correctness Proof

Recall that we consider only constructor-based TRS's. The following lemmas show that $(R^*_\epsilon(f^M\,(R^*_{out}(L_1), \ldots, R^*_{out}(L_n)))^{ok}\ \cap\ IRR'_{outF}(R)$ is correct. By construction, ok locates positions of $OutF_2$ that can be reduced, respecting the outermost strategy, and moreover rewriting $p \in OutF_1$ is necessarily outermost.

Definition 4.16 *A derivation $s_0 \to_{p_0} s_1 \ldots s_n \to_{p_n} s_{n+1}$ is outermost at level 2 if:*

$$\forall i,\ \neg\,(\exists q \in \overline{OutF_1(s_i)}\ s.t.\ q < p_i\ \ \wedge\ \ s_i \to_q).$$

Example 4.17 Let $R = \{f(x) \xrightarrow{1} s(f(x)),\ \ h(x) \xrightarrow{2} s(i(x)),\ \ i(x) \xrightarrow{3} s(x),\ \ g(s(x)) \xrightarrow{4} s(x)\}$ and $t = f(g(h(x)))$. Let E be the set of data-instances of t, $E = \{f(g(h(s^*(a))))\}$.



Notation : Let us denote by τ a fictitious position s.t.

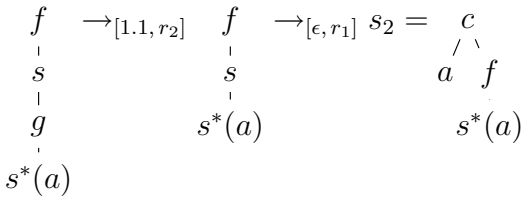
$$s \to_\tau t \text{ means that } s = t.$$

Definition 4.18 *Let $s_0 \to_{p_0, l_0 \to r_0} s_1 \to_{p_1, l_1 \to r_1} s_2$.*
p_0 admits a residue q into s_2, which is denoted by $res(p_0, s_2)$, if:

- *$p_0 = p_1.v.w$ where $v = occ(x, l_1)$*
- *and if $occ(x, r_1)$ exists then $v' = occ(x, r_1)$ and $q = p_1.v'.w$*
  otherwise, $q = \tau$.

Remark : Due to the linearity of TRS, the residue q is unique, and $p_0 \geq p_1 \wedge (q \geq p_1 \, or \, q = \tau)$.

Example 4.19 $R = \{f(s(x)) \xrightarrow{r_1} c(a, f(x)),\ g(x) \xrightarrow{r_2} x\}$.



$res(1.1, s_2) = 2.1$.

Definition 4.20 *A derivation in two steps $s_0 \rightarrow_{p_0, l_0 \rightarrow r_0} s_1 \rightarrow_{p_1, l_1 \rightarrow r_1} s_2$ is without residue if $p_0 || p_1$ or p_0 does not admit a residue into s_2.*

Lemma 4.21 *Let us consider the following derivation:*

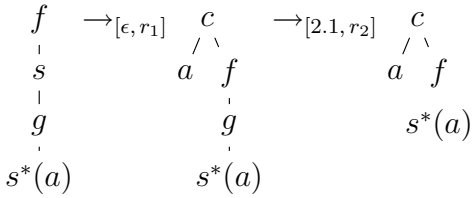
$$s_0 \rightarrow_{p_0, l_0 \rightarrow r_0} s_1 \rightarrow_{p_1, l_1 \rightarrow r_1} s_2 \quad (1)$$

where p_0 admits a residue q into s_2. Then, (1) can be commuted into

$$s_0 \rightarrow_{p_1, l_1 \rightarrow r_1} s'_1 \rightarrow_{q, l_0 \rightarrow r_0} s_2 \quad (2)$$

which is a derivation without residues. Moreover, if (1) is outermost at level 2 then so is (2).

Example 4.22 Let us take the same TRS and derivation as example 4.19. Then, this derivation can be commuted into:



Proof: The only non-trivial point is that (1) outermost at level 2 implies the same property for (2). p_0 admits a residue q into s_2 means that:

- (i) $p_0 = p_1.v.w$ where $v = occ(x, l_1)$,
- (ii) and either $occ(x, r_1)$ exists and $q = p_1.v'.w$ where $v' = occ(x, r_1)$, either it doesn't exist and $q = \tau$.

By (i), $p_1 < p_0$. Since (1) can be commuted into (2), we see that $s_0 \rightarrow_{p_1}$. Then, according to the fact that (1) is outermost at level 2, we deduce that $p_1 \in OutF_1(s_0)$.
The case $q = \tau$ is so trivial.
Now, let us see for $q = p_1.v'.w$.

- either, $\forall u$ s.t. $v' < u < w$, $p_1.u$ is not a function position. So, if $PosF(r_1)$ is empty, $q \in OutF_1(s'_1)$, otherwise, since there is not nested-functions in rhs's, $q \in OutF_2(s'_1)$.
- or, there are function(s) position(s) between $p_1.v'$ and $p_1.w$.
  So, $q \in \overline{OutF_1(s'_1)}$. Obviously, these function(s) position(s) are irreducible because of the fact that (1) is outermost at level (2). Then, q is the first function position in level 2 that can be rewritten in s'_1.

Definition 4.23 *A derivation $s_0 \rightarrow^* s_n$ is said to be without residues if any sub-derivation in two steps is without residue. A 1-step derivation is conventionally without residue.*

Definition 4.24 *Let us consider the following derivation:*

$$t_0 \rightarrow_{[p_0, l_0 \rightarrow r_0]} t_1 \rightarrow t_2 \rightarrow \ldots \rightarrow_{p_{n-1}} t_n.$$

p_0 admits a residue q_n into t_n if:
$res(p_0, t_2) = q_2 \ \wedge \ \forall i \in \{3, \ldots, n\}, \ \exists q_i \in Pos(t_i), \ q_i = res(q_{i-1}, t_i)$

Lemma 4.25 *Let us consider the following derivation:*

$$s_0 \rightarrow_{p_0} s_1 \rightarrow_{p_1} \ldots s_n \rightarrow_{p_n} s_{n+1} \quad (1)$$

where p_0 admits a residue q_{n+1} into s_{n+1} and $s_1 \rightarrow^* s_{n+1}$ is a derivation without residues. (1) can be commuted into:

$$s_0 \rightarrow_{p_1} s'_1 \rightarrow_{p_2} s'_2 \ldots \rightarrow_{p_n} s'_n \rightarrow_{q_{n+1}} s_{n+1} \quad (2)$$

where (2) is without residues. Moreover, (1) outermost at level 2 implies that (2) is an outermost derivation at level 2 too.

Proof: The proof follows from Lemma 4.21. Let us remark that all residues founded up to the previous last are $\neq \tau$ at the aim that q_{n+1} exists.
$s_0 \rightarrow_{p_0} s_1 \rightarrow_{p_1} s_2$ is an outermost derivation at level 2 s.t. $res(p_0, s_2) = q_2$ and can be commuted into $s_0 \rightarrow_{p_1} s_1 \rightarrow_{q_2} s_2$ that is outermost at level 2 and without residues.
$s'_1 \rightarrow_{q_2} s_2 \rightarrow_{p_2} s_3$ is an outermost derivation at level 2 s.t. $res(q_2, s_3) = q_3$ and can be commuted into $s'_1 \rightarrow_{p_2} s'_2 \rightarrow_{q_3} s_2$ that is outermost at level 2 and without residues.
....
By induction, $s'_{n-1} \rightarrow_{q_n} s_n \rightarrow_{p_n} s_{n+1}$ is an outermost derivation at level 2 s.t. $res(q_n, s_{n+1}) = q_{n+1}$ and can be commuted into $s'_{n-1} \rightarrow_{p_n} s'_n \rightarrow_{q_{n+1}} s_{n+1}$ that is outermost at level 2 and without residues.
So, we finally obtain the expected property.

Lemma 4.26 *A derivation with residue(s) can always be transformed into a derivation without residues by commutation. Moreover, if the initial derivation is outermost at level 2 then this property is preserved.*

Proof: Let $s_0 \to^* s_{n+1}$ be a derivation with residue(s). Let us consider the biggest i s.t. $s_i \to^* s_{n+1}$ is with residue and $s_{i+1} \to^* s_{n+1}$ is without residues. Now, let us take $s_i \to^* s_{n+1}$ and consider the biggest j s.t. p_i admits a residue ${p'}_j$ into s_j. Then, $s_i \to^* s_{n+1}$ satisfies the assumptions of Lemma 4.25, therefore it can be commuted into a derivation without residues. In this way, we get a derivation $s_i \to^* s_{n+1}$ without residues. Moreover, the outermost at level 2 property is preserved.
Now, we apply Lemma 4.25 again if necessary.

Lemma 4.27 *Let R be a TRS without critical pairs. Let $s_0 \to^* s_n$ be a derivation without residues, outermost at level 2 and s.t. $\forall p \in OutF_1(s_n)$, $s_n \not\to_p$. Then, $s_0 \to^* s_n$ is outermost.*

The following counter-example shows why we need to forbid critical pairs.

Counter-example 4.28 Let $R = \{f(s(x)) \stackrel{r1}{\to} f(x),\ f(s(s(x))) \stackrel{r2}{\to} g(x),\ g(s(x)) \stackrel{r3}{\to} s(g(x))\}$ and let us see the following derivation:

$$\begin{array}{ccccccc} f & \to_{[1,r3]} & f & \to_{[1.1,r3]} & f & \to_{[\epsilon,r2]} & g \\ | & & | & & | & & | \\ g & & s & & s & & g \\ | & & | & & | & & | \\ s^*(a) & & g & & s & & s^*(a) \\ & & | & & | & & \\ & & s^*(a) & & g & & \\ & & & & | & & \\ & & & & s^*(a) & & \end{array}$$

This derivation is outermost at level 2 and does not have any residues. Moreover, the last term is irreducible at outermost function position ϵ. However, this derivation is not outermost. Let us remark that the TRS has critical pairs.

Proof: (of Lemma 4.27). Let us suppose that $s_0 \to^* s_n$ is not outermost. Let us consider $s_i \to_{p_i} s_{i+1}$ the last non-outermost step of the derivation. So, $s_{i+1} \to^* s_n$ is outermost. By hypothesis, $s_i \to_{p_i} s_{i+1}$ is not outermost but is outermost at level 2. Then, $\exists p \in OutF_1(s_i)$ s.t. $s_i \to_{[p,l\to r]}$ with $p < p_i$. Because of the constructor discipline, $p_i = p.q.w$ where $q = occ(x,l)$, and $s_{i+1} \to_{[p,l\to r]}$.
Since $s_{i+1} \to_{[p_{i+1},l_{i+1}\to r_{i+1}]} s_{i+2}$ is outermost, we have $p_{i+1}||p$ or $p_{i+1} < p$ or $p_{i+1} = p$:

- $p_{i+1} < p$ is impossible because $p \in OutF_1(s_{i+1})$.
- $p_{i+1} = p$ means that we have $p_i = p.q.w = p_{i+1}.q.w$ where $q = occ(x,l)$. Since the rewrite system is without critical pairs, $l_{i+1} = l$ then $q = occ(x, l_{i+1})$. So, p_i admits a residue into s_{i+2}, but by hypothesis, the derivation $s_0 \to^* s_n$ is without residues.
- $p_{i+1}||p$ is possible. We prove in the same way that $p_{i+2}||p ..., p_{n-1}||p$.

Then, $s_n|_p = s_{i+1}|_p$, so $s_n \rightarrow_p$. This is impossible since according to the hypothesis $\forall p \in OutF_1(s_n),\ s_n \not\rightarrow_p$.

Lemma 4.29 *Let R be a non-collapsing TRS without critical pairs. Let $s_0 \rightarrow^* s_n$ be a derivation without residues, outermost at level 2 and s.t. $\forall p \in OutF_1(s_n)$, either $s_n \not\rightarrow_p$, or $\forall q > p,\ s_n(q)$ is M-labeled . Then, $s_0 \rightarrow^* s_n$ is outermost.*

Proof: For the case $s_n \not\rightarrow_p$, it is enough to use Lemma 4.27.
Otherwise, like in the proof of Lemma 4.27, we suppose that $s_0 \rightarrow^* s_n$ is not outermost and we consider $s_i \rightarrow_{p_i} s_{i+1}$ the last non-outermost step of the derivation. So, $s_{i+1} \rightarrow^* s_n$ is outermost. And, we can show that $\exists p \in OutF_1(s_i)$ s.t. $s_i \rightarrow_{[p,l\rightarrow r]}$ with $p < p_i$ and $s_{i+1} \rightarrow_{[p,l\rightarrow r]}$. We have also $s_n|_p = s_{i+1}|_p$.
By definition, the labeled symbols come from the starting language and the non-labeled ones come from a rhs. $s_i \rightarrow_{p_i}$ means (since there is no rhs equal to a variable) that $\exists q > p$ s.t. $s_i(q)$ non-labeled and so, $\neg(\forall q > p,\ s_n(q)$ M-labeled).

4.5 Completeness Proof

Recall that we consider only constructor-based TRS's.
Let us consider the property P on terms, defined by:

$$P(t) = (\forall p \in OutF_1(t),\ t \not\rightarrow_p \ \vee\ \forall u > p,\ t(u) \in C^M \cup F^M)$$

Note that $\neg P(t) = (\exists p \in OutF_1(t),\ t \rightarrow_p \ \wedge\ \exists u > p, t(u)$ not labeled). Let $s_0 \in T_{C^M \cup F^M}$ s.t. $s_0(\epsilon) \in F^M$, and let us consider the outermost derivation $s_0 \rightarrow^* s_n$. Let us take the biggest i s.t. $P(s_i)$, i.e. $s_0 \rightarrow^* s_i \rightarrow^*_{[p_i,\ldots,p_{n-1}]} s_n$ and $\forall j > i$, $\neg P(s_j)$. According to Lemma 4.39, we can suppose that $\forall j > i,\ p_j \in OutF_1(s_j)$.
According to Corollary 4.37 applied to $s_0 \rightarrow^* s_i$ (let us suppose that $s_0(\epsilon) = f^M$), we obtain that $s_i \in R^*_\epsilon(f^M(R^*_{out}(s_0|_1), \ldots, R^*_{out}(s_0|_k)))$. Moreover, $P(s_i)$, then we have $s_i \in IRR'_{outF}(R)$.
According to Lemma 4.44 applied on $s_i \rightarrow^* s_n$, it exists a derivation of the form $s_i \rightarrow^{\|}_{p\in OutF_2} \rightarrow^*_{p\in OutF_1} s_n$.
Then, $s_n \in R^*_{out}(s_0)$.

The following is for proving Corollary 4.37 and Lemmas 4.39 and 4.44.

Definition 4.30 *Let us consider the following derivation:*

$$s_0 \rightarrow_{[p_0,l_0\rightarrow r_0]} s_1 \rightarrow_{[p_1,l_1\rightarrow r_1]} s_2 \ (1)$$

We say that p_1 has an antecedent$^{\|}$ q_1 into s_0 (denoted by $q_1 = ant^{\|}(p_1, s_0)$), if:

- *either* $p_1 \| p_0$ *and so,* $q_1 = p_1$
- *or,* $p_1 = p_0.q.w$ *where* $q = occ(x, r_0)$ *and* $q_1 = p_0.q'.w$ *where* $q' = occ(x, l_0)$.

Remark : Do not confuse antecedent$^{\|}$ (defined above) with antecedent (defined in section 5. The only difference is : $p_1 \| p_0$ (in antecedent$^{\|}$) is replaced by $p_1 \lhd p_0$ (in antecedent).

Lemma 4.31 *Let us consider the following derivation :*

$$s_0 \rightarrow_{p_0 \in OutF_1(s_0), l_0 \rightarrow r_0} s_1 \rightarrow_{p_1, l_1 \rightarrow r_1} s_2 \ (1)$$

where p_1 *admits an antecedent*$^{\|}$ q_1 *into* s_0. *Then, (1) can be commuted into :*

$$s_0 \rightarrow_{q_1, l_1 \rightarrow r_1} s'_1 \rightarrow_{p_0, l_0 \rightarrow r_0} s_2 \ (2)$$

Moreover, if (1) is outermost then (2) is outermost at level 2.

Proof : Obviously, in (2), $p \in OutF_1(s'_1)$. Let us show the second property (the first is trivial). Let suppose that (1) is outermost and p_1 admits an antecedent$^{\|}$ q_1 into s_0 then,

- either $p_1 \| p_0$ and so, $q_1 = p_1$, then it is trivial.
- or, $p_1 = p_0.q.w$ where $q = occ(x, r_0)$ and $q_1 = p_0.q'.w$ where $q' = occ(x, l_0)$. In (1), $p_1 \in PosF(s_1)$ s.t. $\neg(\exists q \in PosF(s_1)\ q < p_1 \ \wedge \ s_1 \rightarrow_q)$ because (1) is outermost. l_0 does not contain nested defined-functions (because of the constructor discipline) then, $\neg(\exists u\ p_0 < u < p_0.q';\ u \in PosF(s_0))$. Moreover, $\neg(\exists v \in PosF(s_0)\ p_0.q' \leq v < p_0.q'.w;\ s_0 \rightarrow_v)$, otherwise $\exists v' \in PosF(s_1)\ p_0.q \leq v' < p_0.q.w;\ s_0 \rightarrow_{v'}$ that contradicts the fact that (1) is outermost.

So, (2) is outermost at level 2.

Lemma 4.32 *Suppose* $s_0(\epsilon) \in F \cup F^M$.
Let us consider the following derivation :

$$s_0 \rightarrow^*_{[\epsilon, rhs's]} s_n \rightarrow_{p_n} s_{n+1} \ (1)$$

s.t. p_n *admits an antecedent*$^{\|}$ q *into* s_0. *Then (1) can be commuted into :*

$$s_0 \rightarrow_{p_n} s'_n \rightarrow^*_{[\epsilon, rhs's]} s_{n+1} \ (2)$$

Moreover, if (1) is outermost then (2) is outermost at level 2.

Proof : The proof comes from Lemma 4.31. By induction on the length of $s_0 \rightarrow^*_{[\epsilon, rhs's]} s_n$.

- if $length = 0$, (2)=(1) and the result is trivial.
- otherwise, suppose $(1) = (s_0 \rightarrow_\epsilon s_1 \rightarrow_{p_1} s_2 \rightarrow \ldots \rightarrow_{p_{n-1}} s_n \rightarrow_{p_n} s_{n+1})$.

By applying Lemma 4.31 on the last two steps, we get
$s_0 \rightarrow_\epsilon s_1 \rightarrow_{p_1} s_2 \rightarrow \ldots s_{n-1} \rightarrow_{q_{n-1}} s'_n \rightarrow_{p_n} s_{n+1}$ (1') where $q_{n-1} = ant^{\|}(p_n, s_{n-1})$.
If (1) is outermost then so is (1').
We get (2) by applying the induction hypothesis on $s_0 \rightarrow^*_{s_{n-1}}$ in (1').

Remark : If p_n admits an antecedent$^{\|}$ into s_{n-1}, then p_n admits an antecedent$^{\|}$ into s_0.

Lemma 4.33 *Suppose $s_0(\epsilon) \in F \cup F^M$.*
Let us consider the following derivation :

$$s_0 \rightarrow^*_{[\epsilon, rhs's]} s_n \rightarrow_{p_n} s_{n+1} \ (1)$$

s.t. p_n does not admit an antecedent$^{\|}$ into s_{n-1}. Then, (1) is a derivation of the form :

$$s_0 \rightarrow^*_{[\epsilon, rhs's]} s_{n+1} \ (2)$$

Proof : Let us suppose that $[\epsilon, rhs's] = [\epsilon, p_1, \ldots p_{n-1}]$, and let $l_i \rightarrow r_i$ be the rewrite rule used in the step $s_i \rightarrow s_{i+1}$. p_n does not admit an antecedent$^{\|}$ into s_{n-1}. Then, $p_n < p_{n-1}$ or $p_n = p_{n-1}.q$ where $q \in PosF(p_{n-1})$. By construction, $p_{n-1} \in OutF_1(s_n)$, therefore the case $p_n < p_{n-1}$ is impossible. Thus necessarily, $p_n = p_{n-1}.q$, which shows that (1) is of the form $s_0 \rightarrow^*_{[\epsilon, rhs's]} s_{n+1}$.

Lemma 4.34 *Let us consider the following derivation :*

$$s_0 \rightarrow^*_{[\epsilon, rhs's]} s_n \rightarrow^* s_k \ (1)$$

Then (1) can be commuted into :

$$s_0 \rightarrow^*_{\neq\epsilon} s'_i \rightarrow^*_{[\epsilon, rhs's]} s_k \ (2)$$

Moreover, if (1) is outermost then (2) is outermost at level 2.

Proof : By induction on the length of the derivation $s_n \rightarrow^*_{p_n} s_k$. Let us suppose that (1) is outermost at level 2.

- if $length = 0$ then it is proved.
- else
  - if $s_n \rightarrow_{p_n} s_{n+1}$ is s.t. p_n admits an antecedent$^{\|}$ into s_{n-1}, and so into s_0 according to the previous remark, then we apply Lemma 4.32. We obtain the following derivation that is outermost at level 2 :

    $$s_0 \rightarrow_{\neq\epsilon} s'_1 \rightarrow^*_{[\epsilon, rhs's]} s_{n+1} \rightarrow^* s_k \ (1')$$

  - else, according to Lemma 4.33, (1) is of the form :

    $$s_0 \rightarrow^*_{[\epsilon, rhs's]} s_{n+1} \rightarrow^* s_k \ (1')$$

The length of the end of the derivation (i.e. $s_{n+1} \to^* s_k$) has decreased, we can then use the induction-hypothesis.

Example 4.35 $R = \{f(s(x)) \to s(f(x)),\ g(s(x)) \to s(g(x))\}$

$$\begin{array}{ccccc} f & \to_\epsilon & s & \to_{1.1} & s & \to_1 & s \\ | & & | & & | & & | \\ s & & f & & f & & s \\ | & & | & & | & & | \\ g & & g & & s & & f \\ | & & | & & | & & | \\ s & & s & & g & & g \\ | & & | & & | & & | \\ a & & a & & a & & a \end{array} \quad \text{can be commuted into} \quad \begin{array}{ccccc} f & \to_{1.1} & f & \to_\epsilon & s & \to_1 & s \\ | & & | & & | & & | \\ s & & s & & f & & s \\ | & & | & & | & & | \\ g & & s & & s & & f \\ | & & | & & | & & | \\ s & & g & & g & & g \\ | & & | & & | & & | \\ a & & a & & a & & a \end{array}$$

Lemma 4.36 *Let s_0 be a term s.t. $s_0(\epsilon) \in F \cup F^M$.*
If $s_0 \to^* s'$ is outermost, then this derivation can be commuted into $s_0 \to^*_{p \neq \epsilon} s'' \to^*_{[\epsilon, rhs's]} s'$, which is outermost at level 2.

Proof: Let us consider the outermost derivation $s_0 \to^* s'$ where $s_0(\epsilon) \in F \cup F^M$. Let us take the smallest i s.t. $s_0 \to^* s_i \to_\epsilon s_{i+1} \to^* s'$. By Lemma 4.34 applied on $s_i \to^* s'$, we obtain $s_0 \to^*_{p \neq \epsilon} s'' \to^*_{[\epsilon, rhs's]} s'$, which is outermost at level 2.

Corollary 4.37 *Let $s_0 = f(s_1, \ldots, s_n)$, and $s_0 \to^* s'$ be an outermost derivation. Then,*

$$s' \in R^*_\epsilon(\ \overset{f}{\underset{R^*_{out}(s_1)}{\swarrow}} \quad \overset{}{\underset{R^*_{out}(s_n)}{\searrow}}\ )$$

Remark : Recall that the property P is defined at the beginning of Section 4.5.

Lemma 4.38 *Let us suppose that $s_0(\epsilon) \in F \cup F^M$, and that there are not critical pairs.*
Let $s_0 \to_{p_0} s_1$ be an outermost derivation s.t. $P(s_0)$ and $\neg P(s_1)$. Then, $s_0 \not\to_\epsilon$

Proof: If $s_0 \to_{[\epsilon, l \to r, \sigma]} s'$ then $\forall u > \epsilon,\ s_0(u) \in C^M \cup F^M$ then $\forall x,\ x\sigma \in T_{C^M \cup F^M}$, then

$$\forall q \in OutF_1(s'),\ \forall u > q,\ s'(u) \in C^M \cup F^M \quad (P'(s')).$$

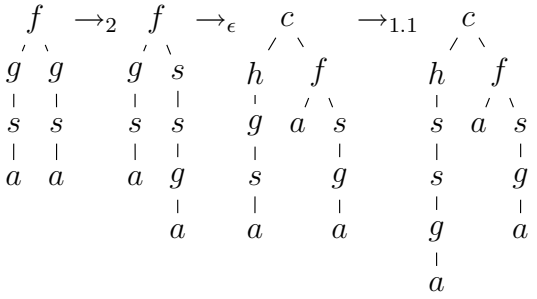
Since $s_0 \to_{p_1} s_1$ is outermost and $s_0 \to_\epsilon s'$, we have $p_0 = \epsilon$. And since there are not critical pairs, $s' = s_1$. Then, we have $\neg P(s')$ (because $\neg P(s_1)$) and $P'(s')$, and it is impossible.

Lemma 4.39 *Let us suppose that $s_0(\epsilon) \in F \cup F^M$ and there are not critical pairs. Let $s_0 \to_{p_1} s_1 \to^* s_n$ (1) be an outermost derivation s.t. $P(s_0)$ and $\forall i \in \{1, \ldots, n\}\ \neg P(s_i)$.*
If it exists $i \in \{1, \ldots, n-1\}$ s.t. $s_i \to_{p_i} s_{i+1}$ satisfies $p_i \notin OutF_1(s_i)$, then p_i admits an antecedent$^{\|}$ q into s_0 and we can commute (1) into :

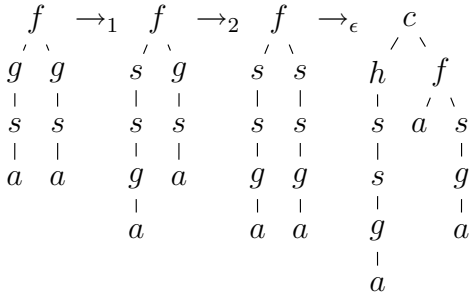
$s_0 \rightarrow_q s'_1 \rightarrow_{p_0} \ldots s'_i \rightarrow s_{i+1} \rightarrow^* s_n$ (2) *s.t. (2) is outermost and* $P(s'_1)$ *and* $\forall j \in \{2, \ldots, i\}\ \neg P(s'_i)$.

Proof: Let us consider the smallest i s,t, $p_i \notin OutF_1(s_i)$. $s_i \rightarrow_{p_i} s_{i+1}$ being outermost, so outermost at level 2, it is obvious that $s_0 \rightarrow_q s'_1$ is outermost at level 2. On the other hand, according to Lemma 4.38, $s_0 \not\rightarrow$, then $s_0 \rightarrow_q s_1$ is outermost.

Example 4.40 Let $R = \{f(x, s(y)) \rightarrow c(h(x), f(a, y)),\ g(s(x)) \rightarrow s(s(g(x)))\}$.



$ant^{\|}(1.1, s_0) = 1$, then it can be commuted into



Lemma 4.41 *Let us suppose that there are no critical pairs and let us consider the following outermost derivation:*

$$s_0 \rightarrow_{p_0} s_1 \rightarrow^*_{[p_1,\ldots,p_{n-1}]} s_n \ (1)$$

s.t. $s_0(\epsilon) \in F \cup F^M$, $P(s_0)$ *and* $\forall k \in \{1, \ldots, n\}$, $\neg P(s_k)$ *and suppose that* $\forall k \in \{1, \ldots, n-1\}$ $p_k \in OutF_1(s_k)$ *Then, (1) is of the form* $s_0 \rightarrow^?_{p \in OutF_2} s_1 \rightarrow^*_{p \in OutF_1} s_n$.

Proof: We have $P(s_0)$ so,

- According to Lemma 4.38, $s_0 \not\rightarrow_\epsilon$: since $\neg P(s_1)$ and since there are not nested functions in lhs's then $p_0 \in OutF_2(s_0)$. So, (1) is of the form $s_0 \rightarrow^?_{p \in OutF_2} s_1 \rightarrow^*_{p \in OutF_1} s_n$.

Definition 4.42 $\rightarrow^*_{[\geq p]}$ *denotes a derivation of the form* $\rightarrow_{p_1} \ldots \rightarrow_{p_n}$ *s.t.* $\forall i,\ p_i \geq p$.

Lemma 4.43 *Let us consider the following outermost derivation:*

$$s_0 \to_{q_0 \in OutF_1(s_0)} s_1 \to \ldots \to_{q_{n-1} \in OutF_1(s_{n-1})} s_n \ (1)$$

s.t. $OutF_1(s_0) = \{p_1, \ldots, p_k\}$. *Then, (1) can be commuted into:*

$$s_0 \to^*_{[\geq p_1]} s_1 \ldots \to^*_{[\geq p_k]} s_n \ (2)$$

s.t. (2) is outermost.

Proof: It is trivial.

Lemma 4.44 *Let us suppose that there are not critical pairs and let us consider the following outermost derivation:*

$$s_0 \to s_1 \to^* s_n \ (1)$$

where $P(s_0)$ *and* $\forall k \in \{1, \ldots, n\}\ \neg P(s_k)$. *Then, (1) can be commuted into:*

$$s_0 \to^{\|}_{p \in OutF_2} s' \to^*_{p \in OutF_1} s_n \ (2)$$

Remark : If $|OutF_1(s_0)| = 1$ then (2)=(1).

Proof: Let $OutF_1(s_0) = \{p_1, \ldots, p_k\}$. By Lemma 4.43, (1) can be commuted into

$$s_0 \to^*_{[\geq p_1]} s_i \ldots s_j \to^*_{[\geq p_k]} s_n \ (1)$$

By Lemma 4.41 applied on each sub-derivation, we obtain that each sub-derivation is of the form $s_0 \to^?_{p \in OutF_2} \to^*_{p \in OutF_1}, \ldots$
To obtain (2), we have to transfer every position of $OutF_2$ in the beginning of the derivation (it changes nothing because of incomparability). And, finally, we obtain $s_0 \to^{\|}_{p \in OutF_2} s' \to^*_{p \in OutF_1} s_n$.

5 Leftmost Descendants: $R^*_{left}(E)$

5.1 Algorithm

Recall that R is assumed to satisfy Restriction 6' (left-variable preserving). We can transform it so that Restriction 6 (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 an algorithm is given in Appendix D.

Example 5.1 Let $R = \{f(s(s(x)), y) \to x\}$.
After running the algorithm, we obtain the following TRS:
$R = \{f(s(s(x)), y) \to eat(x, y),\ f(eat(s(s(x)), x_1), y) \to eat(x, eat(x_1, y)),$
$f(s(eat(s(x), x_{1.1})), y) \to eat(x, eat(x_{1.1}, y)),$
$f(eat(s(eat(s(x), x_{1.1.1})), x_1), y) \to eat(x, eat(x_{1.1.1}, eat(x_1, y)))\}$.

$R_p^*(E)$ does not take the leftmost strategy into account. We will use $R_p^{*\lhd}(E)$ instead.

Definition 5.2 *Given a language E and a position p, we define $R_p^{*\lhd}(E)$ as follows*

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

Example 5.3 $R = \{f(x) \to s(x),\ g(x, y) \to c(h(x), f(y)),\ h(x) \to f(x)\}$
$R_1^{*\lhd}(\{f(g(a, b))\}) = \{f(g(a, b))\} \cup \{f(c(h(a), f(b)))\} \cup \{f(c(f(a), f(b)))\} \cup$
$\{f(c(s(a), f(b)))\} \cup \{f(c(s(a), s(b)))\}$

Theorem 5.4 *Let R be a rewrite system satisfying restrictions 1,2, 3, and 6, 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 $R_p^{*\lhd}(E)$.*

Proof: Building an automaton and proving its correctness is not easy. See Subsection 5.2.

Definition 5.5 *Given a language L and a position p, we define:*

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

Lemma 5.6 *Let R be a constructor-based TRS satisfying restrictions 1 to 3, 5 and 6, 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,

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

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

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

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 :

- If $Succ_t(p) = \emptyset$, then $\forall s \in L$, $\forall p' \in Pos(s)$, $(p' > p \implies s(p') \in C)$. p is the only 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 5.4.
- Let $Succ_t(p) = \{p_1, \ldots, p_n\}$ 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. $R_p^{*\lhd}(L_1) \cup \ldots \cup R_p^{*\lhd}(L_n) = R^*_{left,p}(L)$.
  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 5.4, $R_p^{*\lhd}(L_1)$ 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 2.8, $L_{j-1} \cap IRR_{p_{j-1}}(R)$ 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 Theorem 5.4, $R_p^{*\lhd}(L_j)$ 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 $p' \not> p$.

Theorem 5.7 *Let R be a constructor-based TRS satisfying restrictions 1 to 3, 5 and 6, and E be the set of data-instances of a given linear term t.*

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

Otherwise, let $Succ_t(p) = \{p_1, \ldots, p_n\}$ 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: We have two cases:

- If $\epsilon \in \ PosF(t)$, obviously $R^*_{left}(E) = R^*_{left,\epsilon}(E)$.
- If $\epsilon \notin \ PosF(t)$, and $Succ_t(p) = \{p_1, \ldots, p_n\}$ *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,$R^*_{left}(E) = R^*_{left,p_1}(E) \cup$ $\ldots$
  $\cup\ R^*_{left,p_{n-1}}(\ldots(R^*_{left,p_1}(L) \cap IRR_1(R)))\ldots) \cup \ldots$
  $\cup\, R^*_{left,p_n}(\ldots(R^*_{left,p_1}(L) \cap IRR_1(R)))\ldots)$

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

Example 5.8 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 $*$.
$R^*_{left}(E) = R^*_{left,\epsilon}(E) = R^{*\lhd}_{\epsilon}(R^*_{left,1}(E)) \cup R^{*\lhd}_{\epsilon}(R^*_{left,2}(R^*_{left,1}(E) \cap IRR_1(R))$.

- $R^*_{left,1}(E) = R^{*\lhd}_1(E)$
  $= E \cup \{h(s(*), f(*))\}$ 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

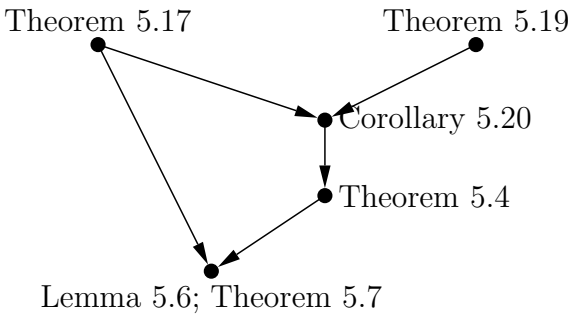
Finally, we obtain leftmost descendants which are:
$L_1 \cup L_2 \cup \{c(f(g(*)), g(f(*)))\} \cup \{c(s(g(*)), g(f(*)))\} \cup \{c(s(s(*)), g(f(*)))\}$
$\cup\ \{c(s(s(*)), s(f(*)))\} \cup \{c(s(s(*)), s(s(*)))\} \cup \{c(f(s(*)), g(s(*)))\} \cup$
$\{c(s(s(*)), g(s(*)))\} \cup \{c(f(s(*)), g(f(*)))\}$.

Missing proofs are given in Appendix C.

5.2.1 Commutation of Rewriting and Left-Basic Derivations

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 (see Definition 5.13), and that a left-basic derivation leading to a normal form is leftmost.

The following figure shows the links between lemmas and theorems. Theorem 5.7 corresponds to the final result.



By the two following Lemmas, we see how to commute a derivation (in two steps, next in several steps). Commutation is based on the notion of antecedent.

Definition 5.9 *Let $t \to_{[q,l\to r]} t'$ be a rewrite step, and let $v' \in Pos(t')$.*
$v \in Pos(t)$ is an antecedent of v' in t (denoted by $ant(v', t)$) through this step if:

- *$v' \lhd q$ 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 5.10 *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 $t_i \rightarrow t_{i+1}$.

Lemma 5.11 *Let R be a linear TRS and s, t, u be terms. If*

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

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

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

Now, let us suppose restrictions 6 and 5. Then, if (1) is leftmost, (2) is leftmost.

Remark : R being linear, $g \rightarrow d$ is linear and so d is linear, consequently $u' = u$, i.e, the last term of derivation is preserved by commutation (20). Moreover, in (2), p does not have an antecedent in s.

Lemma 5.12 *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 antecedent 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 antecedent in t_0.
Now, let us suppose restrictions 6 and 5. Then, if (1) is leftmost, (2) is leftmost too.

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

Definition 5.13 *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 there exist 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.

Counter-example 5.14
Let $R = \{f(x) \rightarrow s(f(x)),\ h(x,y) \rightarrow c(f(x), s(g(y))),\ g(x) \rightarrow x\}$ and let $E = \{h(s^*(a), s^*(a))\}$. 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 5.15 *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 5.16 *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:

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

is left-basic.

Theorem 5.17 *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 6 and 5. Then, if (1) is leftmost, (2) is leftmost too.*

Lemma 5.18 *Let* R *be a given constructor-based TRS and let us assume restriction 6. Let us consider the following left-basic derivation:*

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

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 5.19 *Let* R *be a given constructor-based TRS and let us assume restriction 6. Let us consider the following left-basic derivation:*

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

Then, (1) is leftmost.

Corollary 5.20 *Let R be a given constructor-based TRS and let us assume restrictions 2 and 6. 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 comes from Theorems 5.17 and 5.19.

5.2.2 The Automaton

To build an automaton that recognizes $R_p^{*\lhd}(E)$, we modify the method used for recognizing $R_p^*(E)$ (in Appendix B) in a non-trivial way. Notions introduced in this subsection are illustrated by Example 5.27.

Definition 5.21 *Let D^{sat} and Δ_d^{sat} be the set of states and transitions obtained as in Definition B.3 by replacing each state $d_\sigma^{i,p} \in D$ by $d_{sat,\sigma}^{i,p}$.*

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

Definition 5.22 *(*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 $l_i\sigma \to^*_{\Delta_{t\theta} \cup \Delta_{arg} \cup \Delta_d^{sat}} q'$ where $q' \in \{q\} \cup D^{sat}$, then add the transition $d^{i,\epsilon}_{sat,\sigma|_{Var(r_i)}} \to q'$.

Notation : $\Delta_d^{sat*} = \Delta_d^{sat} \cup \Delta^{sat}$.

Remark : Let us note $\mathcal{B}' = (C \cup F, Q \cup Q_{arg} \cup D^{sat}, \{q\}, \Delta \cup \Delta_{arg} \cup \Delta_d^{sat*})$. Then, $L(\mathcal{B}') = R_\epsilon^*(L(\mathcal{A}|_p))$ i.e the same as $R_\epsilon^{*\lhd}(L(\mathcal{A}|_p))$ except that the rewrite steps are not necessarily leftmost (see (19) 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 5.20), before reducing instances of $g(y)$ by a leftmost strategy.

Definition 5.23 *Let us recall that the construction of $\mathcal{A}_{irr}$ (and Q_{irr}) is given in the proof of Theorem 2.14. 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 $(d^{\cdots}_{\cdots} q^{irr}$ denotes the pair $(d^{\cdots}_{\cdots}, q^{irr}))$:

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

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

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

Definition 5.24 *We define $\mathcal{B}^\triangleleft$ an automaton s.t.:*

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

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

Lemma 5.25 $L(\mathcal{B}^\triangleleft) = R^{*\triangleleft}_\epsilon(L(\mathcal{A}|_p))$.

Proof: The proof comes from Corollary 5.20 and (19).

Corollary 5.26 $L(\mathcal{A}[\mathcal{B}^\triangleleft]_p) = R^{*\triangleleft}_p(L(\mathcal{A}))$.

Example 5.27 Let $R = \{f(x) \xrightarrow{r1} s(x),\ h(x,y) \xrightarrow{r2} c(f(x), g(y)),\ g(x) \xrightarrow{r3} s(x)\}$ 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 $R_\epsilon^{*\lhd}(E)$. 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 $\mathcal{B}^\lhd$.
$\Delta_d = \{c(d_\sigma^{r2,1}, d_\sigma^{r2,2}) \to d_\sigma^{r2,\epsilon},\ f(q_{data}) \to d_\sigma^{r2,1},\ g(q_{data}) \to d_\sigma^{r2,2},\ s(q_{data}) \to d_\sigma^{r1,\epsilon},\ s(q_{data}) \to d_\sigma^{r3,\epsilon}\}$.
$\Delta_{spec} = \{c(d_{sat,\sigma}^{r2,1}, d_\sigma^{r2,2}) \to d_{sat,\sigma}^{r2,\epsilon}, c(d_{sat,\sigma}^{r2,1} q^{irr}, d_{sat,\sigma}^{r2,2}) \to d_{sat,\sigma}^{r2,\epsilon},\ f(q_{data}) \to d_{sat,\sigma}^{r2,1}, g(q_{data}) \to d_{sat,\sigma}^{r2,2},\ s(q_{data}) \to d_{sat,\sigma}^{r1,\epsilon},\ s(q_{data}) \to d_{sat,\sigma}^{r3,\epsilon}\}$.
$\Delta_d^{sat} = \{c(d_{sat,\sigma}^{r2,1}, d_{sat,\sigma}^{r2,2}) \to d_{sat,\sigma}^{r2,\epsilon},\ f(q_{data}) \to d_{sat,\sigma}^{r2,1},\ g(q_{data}) \to d_{sat,\sigma}^{r2,2},\ s(q_{data}) \to d_{sat,\sigma}^{r1,\epsilon},\ s(q_{data}) \to d_{sat,\sigma}^{r3,\epsilon}\}$.
$\Delta^{sat} = \{d_{sat,\sigma}^{r2,\epsilon} \to q^\epsilon,\ d_{sat,\sigma}^{r1,\epsilon} \to d_{sat,\sigma}^{r2,1},\ d_{sat,\sigma}^{r3,\epsilon} \to d_{sat,\sigma}^{r2,2}\}$.
For the following set, we give only what we will use: $\Delta_d^{sat*} \otimes \Delta_{irr} \supseteq \{s(q_{data}) \to d_{sat,\sigma}^{r3,\epsilon} q^{irr},$
$s(q_{data}) \to d_{sat,\sigma}^{r1,\epsilon} q^{irr},\ d_{sat,\sigma}^{r1,\epsilon} q^{irr} \to d_{sat,\sigma}^{r2,1} q^{irr}\}$.

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 $d_{sat,\sigma}^{r2,2}$. This state appears in a lhs only in transition $c(d_{sat,\sigma}^{r2,1} q^{irr}, d_{sat,\sigma}^{r2,2}) \to d_{sat,\sigma}^{r2,\epsilon}$ of Δ_{spec} and in a transition of Δ_d^{sat} (but transitions of Δ_d^{sat} do not belong to the final set of transitions, see previous definition). And using $c(d_{sat,\sigma}^{r2,1} q^{irr}, d_{sat,\sigma}^{r2,2}) \to d_{sat,\sigma}^{r2,\epsilon}$ requires that the first argument is normalized, which does not hold for $f(s^*(a))$.

6 Innermost-Leftmost Descendants : $R_{ileft}^*(E)$

In this section, we will use $R_p^{*\lhd}$ again to take leftmost strategy into account. Recall that restriction 6 can be weakened into restriction 6' by transforming TRS R using a new constructor *eat*. See Section 5 for details.

Definition 6.1 *Given a language L and a position p, let us define :*
$R_{ileft,p}^*(L) = \{t' \mid \exists t \in L,\ t \to_{[u_1,\ldots,u_n]}^* t'$ *by an innermost−leftmost strategy, and* $\forall i\ (u_i \geq p)\}$

Lemma 6.2 *Let R be a constructor-based TRS satisfying the restrictions 1 to 3 and 6, 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,

$$R^*_{ileft,p}(L) = R^{*\lhd}_p(L) \; if \; Succ_t(p) = \emptyset$$

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

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

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

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

- If $Succ_t(p) = \emptyset$, then $\forall s \in L$, $\forall p' \in Pos(s)$, $(p' > p \implies s(p') \in C)$. And since p is a leftmost position and rhs's have no nested defined functions, $R^{*\lhd}_p(L) = R^*_{ileft,p}(L)$.
  We get $\mathcal{A}'$ by Theorem 2.18.
- Let $Succ_t(p) = \{p_1, \ldots, p_n\}$, *s.t.* $p_1 \lhd \ldots \lhd p_n$.
  Let $s \in L$. Either no rewrite step is applied at position p, either a rewrite step is applied at position p and the strategy is innermost-leftmost only if we first normalize s below p by innermost-leftmost derivation.
  To compute innermost-leftmost descendants at position p_n, since $p_{n-1} \lhd p_n$, we first calculate those at position p_{n-1} and normalize it,..., to compute innermost-leftmost descendants at position p_2, since $p_1 \lhd p_2$, we first calculate those at position p_1 and normalize it. So, we search

  $$B = R^*_{ileft,p_n}(\ldots(R^*_{ileft,p_1}(L) \cap IRR_{p_1}(R))\ldots)$$

  $\forall p'$ position s.t. p' occurs strictly on the left of p_n, $\forall s' \in B$, s' is normalized in p'. Then, obviously,
  $R^*_{ileft,p}(L) = R^*_p[R^*_{ileft,p_n}(\ldots(R^*_{ileft,p_1}(L) \cap IRR_{p_1}(R))\ldots) \cap IRR_{p_n}(R)]$
  $\cup R^*_{ileft,p_1}(L) \cup \ldots \cup \; R^*_{ileft,p_n}(\ldots(R^*_{ileft,p_1}(L) \cap IRR_{p_1}(R))\ldots)$
  L is recognized by an automaton $\mathcal{A}$ that discriminates every $p' \in PosF(t)$ s.t. $p' \geq p$ and so, since $\forall i \in \{1, \ldots, n\}$ $p_i > p$, every p' s.t. $p' \geq p_i$. By induction hypothesis, $R^*_{ileft,p_1}(L)$ is recognized by an automaton $\mathcal{A}_1$ that still discriminates p and every position p' s.t. $p' \not> p_1$ and so every p' s.t. $p' \geq p_i$, $i = 2, \ldots, n$. 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* $IRR_{P_1}(R), \ldots$). So, by Lemma 2.8, $R^*_{ileft,p_1}(L) \cap IRR_{p_1}(R)$ is recognized by an automaton that discriminates every position $p' \in PosF(t)$ *s.t.* $p' \not> p_1, \ldots,$ $R^*_{ileft,p_n}(\ldots)$ is recognized by an automaton $\mathcal{A}_n$ that still discriminates p and every position p' s.t. $p' \not> p_n$ (i.e positions that are discriminated before computing $R^*_{ileft,p_n}(\ldots)$ except those below p_n). By Lemma 2.8, $R^*_{ileft,p_n}(L) \cap IRR_{p_n}(R)$is recognized by an automaton that discriminates every position $p' \in PosF(t)$ *s.t.* $p' \not> p_n$ and in

particular $p' \not> p$ and by Theorem 2.18, so is $R^*_p[R^*_{ileft,p_n}(\ldots(R^*_{ileft,p_1}(L) \cap IRR_{p_1}(R))\ldots) \cap IRR_{p_n}(R)]$. Finally, by union, we effectively obtain an automaton that still discriminates position $p' \not> p$.

Theorem 6.3 *Let E be the data-instances of a linear term t and let R a constructor-based TRS satisfying the restrictions 1 to 3 and 6.*

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

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 2.9 and by applying Lemma 6.2.

Example 6.4 Let E be the set of 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 $R^*_{ileft}(E) = s^*(f(*,s(*))) \cup E \cup f(*,h(*))$.

7 Conclusion

Let us make the following remarks.

- Expressing the descendants with strategies, is (much) more difficult than expressing them without a strategy.
- Restrictions 4, 5, 6 are necessary for our algorithms. But we do not know whether they are really necessary to get regular languages. This is an open question.
- R_p^* does not respect the leftmost strategy, except if every rewrite-rule rhs contains at most one defined-function. In this case we can compute leftmost and innermost-leftmost descendants by replacing $R_p^{*\triangleleft}$ with R_p^*, and consequently, we do not have to assume Restriction 6.

To study lazy evaluation, it would be interesting to express the descendants for the leftmost-outermost strategy. It seems easier to introduce "leftmost" inside our computation of outermost descendants, rather than the opposite.

If the (strong) restrictions we need cannot be satisfied in some practical cases, two new research directions are then possible :

- either using approximations (generating a super-set of the descendants),
- or using a more-expressive class of tree languages.

Appendix

A RED(R)

We define $\mathcal{A}_{red}$ as follows :

$\mathcal{A}_{red} = (\mathcal{C} \cup \mathcal{F}, Q_{red}, Q_{redf}, \Delta_{red})$ where
$Q_{red} = \{q_{any},\ q_{rec}\} \cup_{w \in Pos(l_i)} \{q_{rec}^{i,w}\}$
$Q_{redf} = \{q_{rec}\}$ and

$$\Delta_{red} = \{l_i(p')(S_1, \ldots, S_n) \to q_{rec}^{i,p'} \mid l_i \to r_i \in R,\ p' \in \overline{pos}(l_i) \quad S_j = \begin{cases} q_{rec}^{i,p'.j} \text{ if } l_i(p'.j) \neq \text{variable} \\ q_{any} \quad \text{otherwise} \end{cases} \}$$

$$\cup \{q_{rec}^{i,\epsilon} \to q_{rec}\}$$

$$\cup \{s(q_{any}, \ldots, q_{any}) \to q_{any} \mid s \in \mathcal{F} \cup \mathcal{C}\}$$

$$\cup \{s(q_{any}, \ldots, q_{any}, q_{rec}, q_{any}, \ldots, q_{any}) \to q_{rec} \mid s \in \mathcal{F} \cup \mathcal{C},\ ar(s) \geq 1\}$$

$\mathcal{A}_{red}$ recognizes $RED(R)$ indeed, because:
$t|_\epsilon$ reducible i.e. $\exists\ u$ position s.t $u \geq \epsilon$ and $t \to_{[u, l \to r]} t'$.

- q_{any} recognizes any terms (subterms of t at positions incomparable with u, as well as instances of variables of l).
- $q_{rec}^{i,\epsilon}$ recognizes $l\sigma$ (subterms of t at position u).
- q_{rec} recognizes $C[l\sigma]$ (subterms of t at positions v s.t. $\epsilon \leq v \leq u$).

Example A.1 Let $\mathcal{F} \cup \mathcal{C} = \{f^{\backslash 1}, g^{\backslash 2}, s^{\backslash 1}\}$ and

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

The automaton that recognizes $RED(R)$ contains:

$f(q_{rec}^{1,1}) \to q_{rec}^{1,\epsilon},\ s(q_{any}) \to q_{rec}^{1,1},\ g(q_{any}, q_{any}) \to q_{rec}^{2,\epsilon},\ q_{rec}^{1,\epsilon} \to q_{rec},$
$q_{rec}^{2,\epsilon} \to q_{rec},\ f(q_{any}) \to q_{any},\ s(q_{any}) \to q_{any},\ g(q_{any}, q_{any}) \to q_{any},$
$f(q_{rec}) \to q_{rec}, s(q_{rec}) \to q_{rec},\ g(q_{any}, q_{rec}) \to q_{rec},\ g(q_{rec}, q_{any}) \to q_{rec}$

For example, we have $g(s(a), f(s(a)))$ and $f(g(s(a), s(a))) \in L(\mathcal{A}_{red})$

B Recognizing $R_p^*(E)$

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 B.1 let E be the set of data-instances of $t = g(a)$ and let us consider the following TRS:

$$R = \{g(x) \xrightarrow{r1} h(x, b),\ h(x, y) \xrightarrow{r2} c(x, y)\}$$

. $\mathcal{A}_{t\theta}$ 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 B.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. $Q \cap Q_{arg} = \emptyset$. Let $Q' = Q \cup Q_{arg}$. We use states of the form d_σ^p where σ is a Q'-substitution, because rhs's may contain variables.

Definition B.3 *The rhs's of rewrite rules are encoded by the sets of states Q_{arg} and*

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

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 $d^{i,\epsilon}_{\sigma}$.

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

Definition B.4 *Let $\mathcal{A} = (C \cup F, Q, Q_f, \Delta)$ be an automaton that discriminates the position p into the state q, and s.t. $Q \cap Q_{arg} = \emptyset$. We define:* $\mathcal{A}'' = (C \cup F, Q'', Q''_f, \Delta'')$ *where* $Q'' = Q \cup Q_{arg} \cup D$, $Q''_f = \{q\}$, $\Delta'' = \Delta \cup \Delta_{arg} \cup \Delta_d$.

Note that $L(\mathcal{A}'') = L(\mathcal{A}|_p)$ and $\mathcal{A}''$ discriminates the position ϵ into q. This property is necessary in the saturation process defined below, to ensure that the first rewrite step is performed at position ϵ on the terms recognized by $\mathcal{A}''$, i.e. at position p on the terms recognized by $\mathcal{A}$.

Now, we can define the saturation process.

Definition B.5 *(*saturation*)*
Let $\mathcal{B}$ be the automaton obtained from $\mathcal{A}''$ by adding transitions in the following way: whenever there are $l_i \to r_i \in R$, a $(Q\cup Q_{arg})$-substitution σ s.t. $dom(\sigma) = Var(l_i) \cup Var(r_i)$ and $l_i\sigma \to^*_{\Delta''} q'$ where $q' \in \{q\} \cup D$ add the transition $d^{i,\epsilon}_{\sigma|_{Var(r_i)}} \to q'$.

Lemma B.6 $L(\mathcal{B}) = R^*_\epsilon(L(\mathcal{A}|_p))$

For proof, see (19).

Corollary B.7 $L(\mathcal{A}[\mathcal{B}]_p) = R_p^*(L(\mathcal{A}))$

Remark : From Lemma 2.7, if $\mathcal{A}$ discriminates p'$\not> p$ into q', then $\mathcal{A}[B]_p$ also discriminates p' into q'.

C Proofs on Rewriting Commutation and Left-basic Derivations

C.1 Proof of Lemma 5.11

The fact that p has no antecedent it's obvious because of Definition 5.9. Either p occurs on the right of p_0, either p occurs on top of p_0.
Now, let us show that (1) $leftmost \Rightarrow$ (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, $\forall j \in \{1 \ldots (i-1)\},\ \sigma(x_j)$ 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 5) and because of variable preserving TRS (see restriction 6). Then, if we classify $Var(l)$ from left to right, we obtain the same order as $Var(r)$. Hence, (2) is leftmost.

C.2 Proof of Lemma 5.12

This proof follows from Lemma 5.11.
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).

C.3 Proof of Lemma 5.15

Let (1) be a left-basic derivation.
Let us suppose that $\exists\, i \in \{1 \dots n\}$ s.t. p_i has an antecedent in t_{i-1}. By Definition 5.13, $p_i \in B(t_i)$ 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 5.9,

(1) $q \in Pos(t_{i-1})$
(2) - $p_i \lhd p_{i-1}$ *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 5.13; 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.

C.4 Proof of Lemma 5.16

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 5.11.
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 5.12, 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 5.15. And, since q has not antecedent in t_{j-1} then (4) is left-basic.

C.5 Proof of Lemma 5.17

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 5.16, (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 5.12, (2') is leftmost if (1') is leftmost.
We use Lemma 5.16, many times are necessary, and so $t_0 \to^* t_n$ by a left-basic derivation. We proceed in the same way with Lemma 5.12, and (2) is leftmost if (1) is leftmost.

C.6 Proof of Lemma 5.18

Let $t_0 \to_{[p_0, l_0 \to r_0]} t_1 \to \dots t_n \to_{[p_n, l_n \to r_n]} t_{n+1}$ (1) be a left-basic derivation, and let $j \in \{0, \dots, n\}$.
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 $l_j \to r_j$ should have to eliminate variables. It happens that we have a restrictions removing this model of rules.

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

C.7 Proof of Theorem 5.19

By absurd, let us suppose that $\exists j, p'$ s.t. $t_j \to_{[p']} u$ with $p' \lhd p_j$. According to Lemma 5.18, $\nexists k > j$ s.t. $p_k = p'$. Then, $p' \notin B(t_{j+1})$ and $t_{j+1} \in \mathcal{F}$. According to Lemma 5.18, $\exists {p'}_{n+1} \in Pos(t_{n+1})$ s.t. $t_{n+1}|_{{p'}_{n+1}} = t_j|p'$. That contradicts the fact as t_{n+1} be normalized.

D Transforming R to Satisfy Restriction 6

$Transfom(R)$

$init : R' = \emptyset$

$\forall l \to r \in R,\ if\ Var(l) = Var(r)\ then\ add\ l \to r\ to\ R'$

$else\ r' := Construct\_r(l \to r);$

$add\ l \to r'\ to\ R'$

$endif$

let $C(l)$ be a stack that contains all constructor pos. of l.

$if\ |C(l)| \neq \emptyset\ then\ Build(l \to r, C(l), R')\ endif$

$R := R'$

$Construct\_r(l \to r)$

$EraseVar := Var(l) - Var(r)\ where\ |Var(l)| = n\ and\ |Var(r)| = m.$

$r := eat(r, x_{m+1})$

$For\ i := m + 2\ to\ n\ do\ r := r[2 \leftarrow eat(r|_2, x_i)]$

$Build(l \to r, stack, R')$

$p := head(stack);\quad stack := pop(stack);$

$if\ not\ empty(stack)\ then\ Build(l \to r, stack, R')\ endif$

$l' := l[p \leftarrow eat(l|_p, x_p)];\quad r' := Construct\_r(l' \to r);\quad Add\ l' \to r'\ to\ R'$

$stack := Shift(stack, p);$

$if\ not\ empty(stack)\ then\ Build(l' \to r, stack, R')\ endif$

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.

$Shift(stack, p)$

$\forall q \in stack\ s.t.\ \exists w\ and\ q = p.w\ replace\ q\ by\ p.1.w$

Example D.1 Let us take again the TRS of example 5.1.

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

$init : R' = \emptyset$

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

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

$Build(l \to r, C(l), R')$

$p = 1; stack = \{1.1\}$

$Build(l \to r, \{1.1\}, R')$

$p = 1.1; stack = \emptyset$

$l' = l[1.1 \leftarrow eat(l|_{1.1}, x_{1.1})]; \; r' = Construct\_r(l' \to r)$

$add \; f(s(eat(s(x), x_{1.1})), y) \to eat(x, eat(x_{1.1}, y)) \; to \; R'.$

$l' = l[1 \leftarrow eat(l|_1, x_1)]; \; r' = Construct\_r(l' \to r)$

$add \; f(eat(s(s(x)), x_1), y) \to eat(x, eat(x_1, y)) \; to \; R'$

$stack = Shift(stack, 1) = \{1.1.1\}$

$Build(l' \to r, \{1.1.1\}, R')$

$p = 1.1.1; stack = \emptyset$

$l'' = l'[1.1.1 \leftarrow eat(l'|_{1.1.1}, x_{1.1.1})]; \; r' = Construct\_r(l'' \to r)$

$add \; f(eat(s(eat(s(x), x_{1.1.1})), x_1), y) \to eat(x, eat(x_{1.1.1}, eat(x_1, y))) \, to \, R'$

$R = R'$

References

[1] B. Bogaert and S. Tison. Equality and Disequality Constraints on Direct Subterms in Tree Automata. *Proceedings of 9th Annual Symposium on Theoretical Aspects of Computer Science*, volume 577 of *LNCS*. Springer-Verlag, 1992.

[2] D. Caucal. On Word Rewriting Systems Having a Rational Derivation. *Proceedings of 6th Conference on Foundations of Software Science and Computation Structures*, LNCS, Springer-Verlag, 2001.

[3] 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.

[4] H. Comon, M. Dauchet, R. Gilleron, D. Lugiez, S. Tison, and M. Tommasi. *Tree Automata Techniques and Applications* (TATA). http://www.grappa.univ-lille3.fr/tata.

[5] 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.

[6] M. Dauchet, A.C. Caron, and J.L. Coquidé. Reduction Properties and Automata with Constraints. In *Journal of Symbolic Computation, 20:215-233*. 1995.

[7] 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.

[8] J. H. Gallier and R. V. Book. Reductions in tree replacement systems. *Theoretical Computer Science,*37:123-150, 1985.

[9] 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.

[10] T. Genet and F. Klay. Rewriting for Cryptographic Protocol Verification In *Proceedings of CADE*, LNCS, Springer-Verlag, 2000.

[11] R. Gilleron. Decision Problems for Term Rewrite Systems and Recognizable Tree Languages. In *Procedings of STACS*, volume 480 of LNCS, pp. 148-159, Springer-Verlag 1991.

[12] P. Gyenizse and S. Vagvolgyi. Linear Generalized Semi-monadic Rewrite Systems Effectively Preserve Recognizability. *Theoretical Computer Science*, 194, pp 87-122, 1998.

[13] 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.

[14] M. Hermann and R. Galbavý. Unification of Infinite Sets of Terms Schematized by Primal Grammars. *Theoretical Computer Science*, 176,

1997.

[15] 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.

[16] S. Limet, P. Réty, and H. Seidl. Weakly Regular Relations and Applications. In *Proceedings of 12th Conference on Rewriting Techniques and Applications, Utrecht (The Netherlands)*, *LNCS*. Springer-Verlag, 2001.

[17] C. Löding. Model-Checking Infinite Systems Generated by Ground Tree Rewriting. *Proceedings of 7th Conference on Foundations of Software Science and Computation Structures*, LNCS, Springer-Verlag, 2002.

[18] J.C. Raoult. Rational Tree Relations. In *Bulletin of the Belgian Mathematical Society Simon Stevin*, 4:149-176, 1997.

[19] 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.

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

[21] P. Réty and J. Vuotto. Regular Sets of Descendants by Some Rewrite Strategies. In S. Tison, editor, *Proceedings 13th Conference RTA*, volume 2378 of LNCS, pages 129-143. Springer-Verlag, 2002.

[22] P. Réty and J. Vuotto. Regular Sets of Descendants by Leftmost Strategy. In *Proceedings of the second international workshop on Reduction Strategies in Rewriting and Programming (WRS)*, Copenhagen (Denmark). Technical report E1852-2002-02 of Technische Universitat Wien. 2002.

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

[24] H. Seki, T. Takai, F. Youhei and Y. Kaji. Layered Transducing Term Rewriting System and Its Recognizability Preserving Property. In S. Tison, editor, *Proceedings 13th Conf. RTA*, volume 2378 of LNCS, p. 98-113. Springer-Verlag, 2002.

[25] 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.