Tree automata for rewrite strategies

Pierre Réty*, Julie Vuotto

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

Received 3 January 2003; accepted 14 December 2004
Available online 2 March 2005

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.

© 2005 Elsevier Ltd. All rights reserved.

Keywords: Term rewriting; Strategy; Tree automaton

1. Introduction

Finite tree automata have already been applied to many areas of computer science, and in particular to rewrite techniques (Comon et al., 1999). In comparison with more sophisticated refinements (Bogaert and Tison, 1992; Dauchet et al., 1995; Raoult, 1997; Hermann and Galbavý, 1997; Gouranton et al., 2001; Limet et al., 2001), 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 (Comon, 1995). 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 (Dauchet and Tison, 1990; Löding, 2002; Salomaa, 1988; Coquidé et al., 1991; Gyenizse and Vagvolgyi, 1998; Jacquemard, 1996; Takai et al., 2000; Seki et al., 2002) (and Caucal (2001) for string rewriting). Unfortunately, it is undecidable whether a given rewrite system preserves regularity (also called recognizability) or not (Gilleron, 1991), and all previous papers define decidable subclasses. Except Jacquemard (1996), Takai et al. (2000) and Seki et al. (2002), they assume that the right-hand sides (both sides when dealing with sets of equations) of rewrite rules are shallow, up to slight differences.

Réty’s work (Réty, 1999) does not always preserve recognizability ($E$ is not arbitrary), but allows rewrite rules forbidden by the other papers. On the other hand, the possibility of computing a superset of the set of descendants, only assuming left-linearity, has been investigated in Genet (1998) and Genet and Klay (2000).

Reduction strategies in rewriting and programming have drawn increasing attention within the last few 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 means of a finite tree automaton the $st$-descendants $R_{st}^*(E)$ of $E$ can help in studying $st$: in particular it allows one to decide $st$-reachability since $t_1 \xrightarrow{st} t_2 \iff t_2 \in R_{st}^*([t_1])$, and $st$-joinability since $t_1 \downarrow t_2 \iff R_{st}^*([t_1]) \cap R_{st}^*([t_2]) \neq \emptyset$. 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 of $f(E)$ are indeed sorted lists, by checking that the intersection with the set of non-sorted lists is empty. This paper is an extension of Réty (1999) that takes some strategies into account. It is an extended and improved version of our conference/workshop papers Réty and Vuotto (2002a,b).

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. Jacquemard (1996) computes sets of normalizable terms, which amounts to computing 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.
5. On a finite domain, the set of non-sorted lists is a regular language.
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 for obtaining regular languages (see counter-examples in Réty (1999)). 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 Réty (1999) 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 \( R = \{ f(s(x)) \rightarrow s(f(x)), a \rightarrow a \} \), 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).

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 \( t \), \( \text{Var}(t) \) is the set of variables appearing in \( t \), \( \text{Pos}(t) \) is the set of **positions** of \( t \), \( \text{Pos}_{t}(t) \) is the set of non-variable positions of \( t \), \( \text{Pos}_{F}(t) \) denotes the set of defined-function positions of \( t \). \( t \) is **linear** if each variable of \( t \) appears only once in \( t \). For \( p \in \text{Pos}(t) \), \( t|_p \) is the subterm of \( t \) at position \( p \), \( t(p) \) is the top symbol of \( t|_p \), and \( t[p']_p \) denotes the subterm replacement. For positions \( p, p', p \geq p' \) means that \( p \) is located below \( p' \), i.e. \( p = p' \lor v \) for some position \( v \), whereas \( p \parallel p' \) means that \( p \) and \( p' \) are incomparable, i.e. \( \neg(p \geq p') \land \neg(p' \geq p) \). The term \( t \) contains **nested functions** if there exist \( p, p' \in \text{Pos}_{F}(t) \) s.t. \( p > p' \). The domain \( \text{dom}(\theta) \) of a substitution \( \theta \) is the set of variables \( x \) s.t. \( x \theta \neq x \).

\(^6\) if \( x \in \text{Var}(l) \cap \text{Var}(r) \) and \( y \in \text{Var}(l) - \text{Var}(r) \), then \( x \) occurs in \( l \) on the left of \( y \).
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 \( \theta \) 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 \( \forall v \succ p \) (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 \( A = (C \cup F, Q, Q_f, \Delta) \) where \( Q_f \subseteq Q \) are sets of states and \( \Delta \) 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 \( \to_A \)) is the rewrite relation induced by \( \Delta \). A ground term \( t \) is recognized by \( A \) into \( q \) if \( t \to_\Delta^* q \). \( L(A) \) is the set of terms recognized by \( A \) into any states of \( Q_f \). A derivation \( t \to_\Delta^* q \) where \( q \in Q_f \) is called a successful run on \( t \). The states of \( Q_f \) are called final states, \( A \) 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 \( \tilde{A} \) s.t. \( E = L(A) \). 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 \( \sigma \) is a substitution s.t. \( \forall x \in dom(\sigma), x\sigma \in Q \).

Given a term \( t \), we make the following definition moreover:

**Definition 2.1.** Let \( p \in Pos(t) \). \( Succ_1(p) \) are the nearest function positions below \( p \):

\[
Succ_1(p) = \{ p' \in PosF(t) \mid p' > p \land \forall q \in Pos(t) \ (p < q < p' \implies q \notin PosF(t)) \}.
\]

**Definition 2.2.** Let \( p, p' \in Pos(t) \), \( p \prec p' \) 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 \( A \) discriminates position \( p \) into state \( q \) means that along every successful run on \( t \in L(A), t|_p \) (and only this subterm) is recognized into \( q \). This property allows us to modify the behavior of \( A \) below position \( p \) without modifying the other positions, by replacing all transitions used below position \( p \) by those of another automaton \( A' \).

**Definition 2.3.** Let us consider the derivation \( t_0 \to_\Delta^* t_0 \) (1).

The sub-derivation \( t_i \to_\Delta^* t_j \) of (1) composed of empty transitions is length-max if:

\[
\neg (t_{i-1} \to t_i \text{ via an } \epsilon \text{-transition}) \land \neg (t_j \to t_{j+1} \text{ via an } \epsilon \text{-transition}).
\]

**Definition 2.4.** The automaton \( A = (C \cup F, Q, Q_f, \Delta) \) discriminates the position \( p \) into the state \( q \) if

- \( L(A) \neq \emptyset \),
- \( \forall t \in L(A), \ p \in Pos(t) \), 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 $A[p] = (C \cup F, Q, \{q\}, \Delta)$.

**Lemma 2.5.** $L(A[p]) = \{ t[p] \mid t \in L(A) \}$.

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

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

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

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

where $\Delta'' = \Delta \setminus \{ l \rightarrow q \}$

$$\cup \Delta' \cup \{ q_f \rightarrow q \mid q_f \in Q'_f \}.$$

**Lemma 2.7.** $L(A[A']_p) = \{ t[t']_p \mid t \in L(A), t' \in L(A') \}$, and $A[A']_p$ still discriminates $p$ into $q$. Moreover, if $A$ discriminates another position $p' s.t. p' \not\ni p$, into the state $q'$, then $A[A']_p$ still discriminates $p'$ into $q'$.

**Proof.** By construction of $A[A']_p$ (see Definition 2.6), transition rules of $\Delta$ used above $p$ are not changed and transition rules of $\Delta'$ 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 $q_f \rightarrow q$.

Then the last sub-item of Definition 2.4 is satisfied. On the other hand, if $s \rightarrow^* q_f$ is a successful derivation of $A[A']_p$, it is of the form $s \rightarrow^* \Delta, s[q_f]_p \rightarrow s[q]_p \rightarrow^* \Delta q_f$. Then the last-but-one sub-item of Definition 2.4 is satisfied. Therefore $A[A']_p$ still discriminates $p$ into $q$.

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

---

\[ P. \text{Réty, J. Vuotto / Journal of Symbolic Computation 40 (2005) 749–794 } 753 \]
According to Definition 2.6, \( q'_f \rightarrow q \in \Delta' \) and the only transitions reaching \( q \)
are of the form \( q'_f \rightarrow q \). Then, \( s \rightarrow^* s[q'_f]_p \rightarrow s[q]_p \rightarrow^* q_f \). Moreover,
there are no transition rules that reduce states of \( Q \) into \( q'_f \). Then, more precisely,
\( s \rightarrow^*_\Delta s[q'_f]_p \rightarrow s[q]_p \rightarrow^* q_f \). And so, \( s|_p \rightarrow^*_\Delta q'_f \), i.e. \( s|_p \in L(A') \).

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

**Lemma 2.8.** Let \( A, B \) be automata, and let \( A \cap B \) be the classical automaton used to recognize intersection, whose states are pairs of states of \( A \) and \( B \).
If \( A \) discriminates \( p \) into \( q_A \), \( B \) discriminates \( p \) into \( q_B \), and \( L(A) \cap L(B) \neq \emptyset \), then \( A \cap B \) discriminates \( p \) into \( (q_A, q_B) \).

**Proof.** Let \( t \in L(A \cap B) \);

- since \( t \in L(A) \), \( p \in Pos(t) \)
- for any successful run on \( t, t \rightarrow^*_\Delta q_f \)

\[
t[(q'_A, q'_B)]_p \rightarrow^* (q_f A, q_f B) \quad (1)
\]
for each sub-derivation of (1) \( t[q'_A, q'_B]_p \rightarrow^*_\Delta q_f \)

\[
t[(q'_n, q'_n B)]_p \rightarrow^* (q_f A, q_f B) \quad (1)
\]

verifying Definition 2.3:

- if \( p' = p \) then from discrimination of \( A \) and \( B \), \( q'_n A = q_A \) and \( q'_n B = q_B \);
- if \( p' \neq p \) then from discrimination of \( A \) and \( B \), \( \forall i \in \{1, \ldots, n\}, (q'_i A \neq q_A) \) and \( (q'_i B \neq q_B) \). \( \square \)

### 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-illustrations of a given linear term \( t \).

**Definition 2.9.** We define the automaton \( A_{\text{data}} \) that recognizes the set of data-terms \( T(C) \):

\[
A_{\text{data}} = (C, Q_{\text{data}}, Q_{\text{data}^f}, \Delta_{\text{data}})
\]

where

\[
Q_{\text{data}} = Q_{\text{data}^f} = \{q_{\text{data}}\}
\]

\[
\Delta_{\text{data}} = \{c(q_{\text{data}}, \ldots, q_{\text{data}}) \rightarrow q_{\text{data}} \mid c \in C\}.
\]
Given a linear term $t$, we define the automaton $A_{\theta}$ that recognizes the set of data-
instances of $t$: $A_{\theta} = (C \cup F, Q_{\theta}, Q_{\theta \rightarrow}, \Delta_{\theta})$ where

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

$$Q_{\theta \rightarrow} = \{q^i\} \text{ (if } t \text{ is a variable)}$$

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

Note that $A_{\theta}$ discriminates each position $p \in Pos(t)$ into $q^p$. On the other hand, $A_{\theta}$
is not deterministic, whenever there is $p \in Pos(t)$ s.t. $t|_p$ is a constructor-term. Indeed for
any data-instance $t|_{p,\theta}, t|_{p,\theta} \rightarrow^{*}_{\Delta_{\theta}} q^p$ and $t|_{p,\theta} \rightarrow^{*}_{\Delta_{\theta}} q_{\text{data}}$.

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 $IRR_p(R) = \{s \in T_{C \cup F} \mid p \in Pos(s) \text{ and } s|_{p} \text{ is irreducible}\}$.

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:

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

**Lemma 2.12** *(Gallier and Book, 1985).* 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.** $IRR_e(R) = \overline{RED(R)}$. Therefore, $IRR_e(R)$ is a regular language.

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

**Theorem 2.14.** Let $t$ be a term, and $p \in Pos(t)$. $IRR_p(R)$ is a regular language and is
recognized by an automaton that discriminates every position $p' \in Pos(t)$ s.t. $p' \neq p$.

**Proof.** Let $A_{\text{irre}} = (C \cup F, Q_{\epsilon}, Q_{\text{irf}}, \Delta_{\epsilon})$ be an automaton that recognizes $IRR_e(R)$.
Let $p = p_1, \ldots, p_k$ with $p_1, \ldots, p_k \in \mathbb{N} - \{0\}$
and $\forall i \ p_i \leq \text{Max}_{f \in F \cup C} (\text{ar}(f))$.

The length of position $p$ is $\text{length}(p) = k$.

We define $A_{\text{irr}}$ as follows:

$$A_{\text{irr}} = (C \cup F, Q_{\text{irr}}, Q_{\text{irf}}, \Delta_{\text{irr}})$$

where

$$Q_{\text{irr}} = \{q_{\text{any}}, q_{\text{rec}}\} \cup_{i < p} \{q^i\} \cup_{v \in Pos(t)} \{q^v\} \cup Q_{\epsilon}$$

$$Q_{\text{irf}} = \{q^i\}$$

and
\[\Delta_{irr} = \{ s(S_1, \ldots, S_n) \rightarrow q^j \mid s \in F \cup C, ar(s) \geq \text{length}(j) + 1 \}
\]

\[q^j \in Q_{irr}, S_i = \begin{cases} q^{i,i} & \text{if } j.i < p \\ q^{i,j} & \text{if } j.i = p \text{ } \text{if } p \neq \epsilon & \text{if } j.i = p \text{ } \text{if } p \neq \epsilon & \text{otherwise} \end{cases} \]

\[\cup \{ q_f \rightarrow q_{\text{rec}} \mid q_f \in Q_{\text{left}} \} \text{ if } p \neq \epsilon \]

\[\cup \{ q_f \rightarrow q^f \mid q_f \in Q_{\text{left}} \} \text{ if } p = \epsilon \]

\[\cup \{ s(S_1, \ldots, S_n) \rightarrow q^j_{\text{any}} \mid s \in F \cup C, q^j_{\text{any}} \in Q_{irr} \}
\]

\[S_i = \begin{cases} q^j_{\text{any}} & \text{if } j.i \in \text{Pos}(t) \\ q^j_{\text{any}} & \text{otherwise} \end{cases} \]

\[\cup \{ s(q_{\text{any}}, \ldots, q_{\text{any}}) \rightarrow q_{\text{any}} \mid s \in F \cup C \}
\]

\[\cup \Delta_L. \]

\[A_{irr} \text{ recognizes } IRR^p_p(R) \text{ indeed, because:} \]

\[t|_p \text{ reducible, i.e. } \exists u \text{ position s.t. } u \geq p \text{ and } t \rightarrow [a] t'. \]

- \(q^u_{\text{any}}\) recognizes any terms.
- \(q^w_{\text{any}}\) recognizes \(t|_w f\) or \(w < p\).

We have written \(ar(s) \geq \text{length}(j) + 1\) to ensure that \(p \in \text{Pos}(t)\). For example, if \(p = 2.1\) and \(s(\ldots) \rightarrow q^1\), then \(s\) should have an arity \(\geq 2\).

Obviously, \(A_{irr}\) discriminates \(p\) into \(q_{\text{rec}}\) (into \(q^f\) if \(p = \epsilon\)), and each \(p' \in \text{Pos}(t)\) s.t. \(p' \not\in p\) into \(q_{\text{any}}\) (\(q^p\) if \(p' < p\)). \(\square\)

2.5. Descendants

\(t'\) is a descendant of \(t\) if \(t \rightarrow [p] t'\). If \(E\) is a set of ground terms, \(R^s(E)\) denotes the set of descendants of elements of \(E\). \(R^*_{\text{cons}}(E)\) (resp. \(R^*_{\text{left}}(E), R^*_\text{left}(E)\)) denotes the set of descendants of \(E\), according to an innermost (resp. outermost, leftmost, innermost-leftmost) strategy.

**Definition 2.15.** \(t \rightarrow [p, \text{rhs's}]^{+} t'\) 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.

\[t = t_0 \rightarrow [p_0, \text{left}_0] t_1 \rightarrow [p_1, \text{left}_1] t_2 \rightarrow [p_2, \text{left}_2] t_3 \rightarrow [p_3, \text{left}_3] \cdots \rightarrow [p_{n-1}, \text{left}_{n-1}] t_n \rightarrow [p_n, \text{left}_n] t_n+1 = t' \]

where

- \(p_0 = p\) and \(P(t) = \{ p \}\).
- \(\forall j, p_j \in P(t_j)\).
- \(\forall j, P(t_{j+1}) = P(t_j) \cup \{ p' \mid p' \geq p_j \} \cup \{ p, w \mid w \in \text{Pos}(r_{j+1}) \}.\)

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

**Definition 2.16.** Given a language \(E\) and a position \(p\), we define \(R^s_p(E)\) as follows:

\[R^s_p(E) = E \cup \{ t' \mid \exists t \in E, t \rightarrow [p, \text{rhs's}]^{+} t' \}.\]
Example 2.17. Let $R = \{ f(x) \to s(x), \ g(x) \to s(h(x)), \ h(x) \to p(f(x)) \}$ where $F = \{ f, g, h \}$ and $C = \{ s, a \}$. The symbol(s) that are eligible for rewriting, are underlined:

$$R^*_p(f(h(g(a)))) = f(h(g(a))) \cup f(p(f(g(a)))) \cup f(p(s(g(a)))) .$$

An insight into the algorithm underlying the following result is given in Section 3.1 as an example, and a formal description appears 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 (Réty, 1999). Let $R$ be a rewrite system satisfying restrictions 1–3. If $E$ is recognized by an automaton that discriminates position $p$ into some state $q$, and possibly $p'$ into $q'$ for some $p' \in \text{Pos}(t)$ s.t. $p' \not\geq p$ and some state $q'$, then so is $R^*_p(E)$.

3. Innermost descendants: $R^*_\text{in}(E)$

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 $A_{q^\theta}$ be the automaton that recognizes the language $E = \{ f(s(f(s^*(a)))) \}$ of the data-instances of $t$.

$A_{q^\theta}$ can be summarized by writing

$$q' \quad q^1 \quad q_{\text{data}}^1 \quad q_{\text{data}}^{1.1} \quad q^1_{\text{data}}$$

$$f(\quad s(\quad f(\quad s(\quad s^*\quad(a))))))$$

which means that

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

where $q^\epsilon$ is the accepting state.

Consider now the rewrite system $R = \{ f(s(x)) \to s(x) \}$.

Obviously, $R^*_\text{in}(E) = E \cup f(s(s(s^*(a)))) \cup s(s(s^*(a)))$.

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 $R^*_\text{in}(E)$, 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 $\epsilon$ and 1.1. According to the previous remark, we start from position 1.1.

Now, we calculate $R^*_{1,1}(E)$.

(1) \[ f(s(f(s(s^*(a))))) \rightarrow_{[1.1, x/s^*(a)]} f(s(s^*(a))). \]
The language that instantiates the rewrite rule variable \( x \) is \( s^* (a) \) (recognized into \( q_{\text{data}} \)). Therefore, we encode the first version of the rhs: \( d^*_{\text{data}} q_{\text{data}} \), by adding state \( d^*_{\text{data}} \) and the transition \( s (q_{\text{data}}) \rightarrow d^*_{\text{data}} \).

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 \( f (s(q_{\text{data}})) \rightarrow^* q_{1,1} \), we add the transition \( d^*_{\text{data}} q_{\text{data}} \rightarrow q_{1,1} \) so that the instance of the rhs by \( q_{\text{data}} \) is also recognized into \( q_{1,1} \), i.e. \( s (s^* (a)) \rightarrow^* q_{1,1} \). So, \( R^*_{1,1} (E) = E \cup f (s(s^* (a))) \) is recognized by the automaton.

Now, rewriting terms of \( R^*_{1,1} (E) \) at position \( \epsilon \) is allowed only if position 1.1 is normalized. Consider \( E' = R^*_{1,1} (E) \cap \text{IRR}_{1,1} (R) \) where \( \text{IRR}_{1,1} (R) \) is the ground term irreducible at position 1.1, over the TRS \( R \). Thus \( E' = f (s(s^* (a))) \), and let us calculate \( R^*_{1,1} (E') \).

Let \( A' = (C \cup F, Q', \{ q^* \}, \Delta') \) be an automaton that recognizes the language \( E' \) where, \( \Delta' = \{ a \rightarrow q_{\text{data}}, s(q_{\text{data}}) \rightarrow q_{\text{data}}, s(q_{\text{data}}) \rightarrow q_{1,1}, s(q_{1,1}) \rightarrow q'_{1,1}, f(q'_{1,1}) \rightarrow q^* \} \) where \( q^* \) is the accepting state.

\[
(2) \quad f (s(s^* (a))) \rightarrow_{\{ [s(s^* (a))] \}} s (s^* (a)).
\]

The language that instantiates \( x \) is \( s^* (a) \) (recognized into \( q_{1,1}^* \)). Therefore, we encode the second version of the rhs: \( d^*_{q_{1,1}} q_{1,1} \), by adding state \( d^*_{q_{1,1}} \) and the transition \( s(q_{1,1}) \rightarrow d^*_{q_{1,1}} \). By saturation, since \( f (s(x)) \) is the rule lhs and \( f (s(q_{1,1})) \rightarrow q^* \), we add the transition \( d^*_{q_{1,1}} q_{1,1} \rightarrow q^* \) so that \( s(s^* (a)) \rightarrow^* q^* \).

So, \( R^*_{1,1} (E') \) 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 \( \epsilon \). Therefore, \( R^*_{s_{\text{in}}} (E) = R^*_{s_{\text{in}}} (E') \cup R^*_{s_{\text{in}}} (E) = R^*_{s_{\text{in}}} (R^*_{1,1} (E) \cap \text{IRR}_{1,1} (R)) \cup R^*_{s_{\text{in}}} (E) \).

**Remark.** In the previous example, the starting term has nested functions. When this 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 \( p \), \( R^*_{s_{\text{in}}} (L) \) is the set of innermost descendants of \( L \) over the TRS \( R \), reducing positions below (or equal to) \( p \), i.e.

\[
R^*_{s_{\text{in}}} (L) = \{ s' \mid \exists s \in L, s \rightarrow^* \sum_{u_i, \ldots, n_i} s' \} \text{ by an innermost strategy, } \forall i \ (u_i \geq p) \}.
\]

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

**Lemma 3.2.** Let \( R \) be a constructor-based TRS satisfying restrictions 1–3, and \( E \) be the data-instances of a given linear term \( t \).
Let \( p \in \text{Pos}(t) \), and \( L \) be a language s.t. \( L|_{p} = E|_{p} \), and that is recognized by an automaton \( A \) that discriminates every position \( p' \in \text{Pos}(t) \mid p' \geq p \). Then,

\[
R^*_p(L) = R^*_p(L) \text{ if } \text{Succ}(p) = \emptyset.
\]

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

\[
R^*_p(L) = \left[ R^*_p[R^*_{p_1}(\ldots(R^*_{p_n}(L))\ldots) \cap \text{IRR}_{p_i}(R)] \cup R^*_p(\ldots(R^*_{p_n}(L))\ldots) \right]
\]

and \( R^*_p(L) \) is recognized by an automaton \( \mathcal{A}' \) s.t. if \( p' \in \text{Pos}(t) \), \( p' \neq p \), and \( \mathcal{A} \) discriminates \( p' \) into \( q' \); then \( \mathcal{A}' \) also discriminates \( p' \) into \( q' \).

**Proof.** By noetherian induction on \( (\text{Pos}(t), >) \).

- If \( \text{Succ}(p) = \emptyset \), then \( s \in L \), \( \forall p' \in \text{Pos}(s) \), \( (p' > p \implies s(p') \in C) \). And since rhs’s have no nested functions, \( R^*_p(L) = R^*_{\text{in},p}(L) \).

  We get \( \mathcal{A}' \) by Theorem 2.18.

- Let \( \text{Succ}(p) = \{p_1, \ldots, p_n\} \). Let us define

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

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 \( p_i \),

\[
R^*_{\text{in}, p}(L) = R^*_{\text{in}, p_1}(\ldots(R^*_{\text{in}, p_n}(L))\ldots).
\]

Now, note that \( \forall i, j \in \{1 \ldots n\}, (i \neq j \implies p_i \parallel p_j) \). Moreover rewrite steps at incomparable positions can be commuted. Then obviously,

\[
R^*_{\text{in}, p}(L) = R^*_{\text{in}, p_1}(\ldots(R^*_{\text{in}, p_n}(L))\ldots).
\]

\( L \) is recognized by an automaton \( \mathcal{A} \) that discriminates every \( p' \in \text{Pos}(t) \) s.t. \( p' \geq p \).

For each \( i \), \( p_i > p \), then \( \mathcal{A} \) discriminates every \( p' \in \text{Pos}(t) \) s.t. \( p' \geq p_i \).

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

By Theorem 2.14, \( \text{IRR}_{p_i}(R) \) is recognized by an automaton that discriminates every position \( p' \in \text{Pos}(t) \) s.t. \( p' \neq p_i \), thus necessarily \( p \). By Lemma 2.8, \( \cap_{p_i \in \text{Succ}(p)} \text{IRR}_{p_i}(R) \) is recognized by an automaton that discriminates \( p \).

Therefore \( R^*_{\text{in}, p_1}(\ldots(R^*_{\text{in}, p_n}(L))\ldots) \cap_{p_i \in \text{Succ}(p)} \text{IRR}_{p_i}(R) \) is recognized by an automaton that discriminates \( p \), and from Theorem 2.18, so is \( R^*_p(R^*_{\text{in}, p_1}(\ldots(R^*_{\text{in}, p_n}(L))\ldots) \cap_{p_i \in \text{Succ}(p)} \text{IRR}_{p_i}(R)) \). Moreover discrimination of positions \( p' \neq p \) is preserved. Finally, by union, we obtain an automaton that discriminates \( p \) and preserves the discrimination of positions \( p' \neq p \). \( \square \)
Theorem 3.3. Let $R$ be a constructor-based TRS satisfying the restrictions 1–3, and $E$ be the data-instances of a given linear term $t$:

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

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

Proof. We have two cases:

- If $t(e) \in F$, obviously $R^*_{in}(E) = R^*_{in,e}(E)$.
- If $t(e) \notin F$, $\forall i, j \in [1 \ldots n], (i \neq j \implies p_i \parallel p_j)$, 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). \qed

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

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

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

* will symbolize the data-terms that instantiate $t$.

$t(e) \in F$, so we calculate $R^*_{in,e}(E)$ where $E = \{f(g(*), h(g(*)))\}$.

$$R^*_{in,e}(E) = R^*_{in}([R^*_{in,2}(E)] \cap \text{IRR}_1(R) \cap \text{IRR}_2(R) \cap R^*_{in,1}(R^*_{in,2}(E)).$$

We have to compute $R^*_{in,2}(E)$.

$\text{Succ}_1(2) = \{2\}$

So, $R^*_{in,2}(E) = R^*_{in,2}(E) \subseteq (R^*_{in,2}(E)] \cap \text{IRR}_2(E) \cap R^*_{in,2}(E)$

where $R^*_{in,2}(E) = E \cup \{f(g(*), h(*))\}$.

$$R^*_{in,2}(E) = R^*_{in,2}([f(g(*), h(*))]) \cup R^*_{in,2}(E)$$

$$= \{f(g(*), h(*))\} \cup \{f(g(*), s(*))\} \cup R^*_{in,2}(E)$$

(Nota denoted by $E1$).

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

$\text{Succ}_1(1) = \emptyset$.

So, $R^*_{in,1}(E1) = R^*_{in,1}(E1)$

$$= E1 \cup \{f(s, h(*))\} \cup \{f(s, s(*))\} \cup \{f(s, h(g(*)))\}$$

(Nota denoted by $E2$).

$$R^*_{in,1}(E) = R^*_{in,1}[E2 \cap \text{IRR}_2(R) \cap \text{IRR}_2(R)] \cup E2$$

$$= R^*_{in,1}[[f(s, s(*))]] \cup E2$$

$$= \{f(s, s(*))\} \cup \{s(*)) \cup E2.$$

Finally, we obtain $R^*_{in}(E) = E1 \cup f(g(*), h(*)) \cup f(g(*), s(*)) \cup f(s, h(*)) \cup f(s, s(*)) \cup f(s, h(g(*))) \cup s(*))$.

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 in the algorithm. Then the
algorithm is given (Section 4.2) as well as smaller examples. In particular, Counterexample 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 does not need to read technical details to understand it. The proofs are given last.

As shown in the following example, the principle of the method is paradoxically to rewrite in an 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 \[ R = \{ g(x) \rightarrow h(x), \; h(p(x)) \rightarrow g(x), \; f(s(x)) \rightarrow p(f(x)) \} \], 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).

\( \text{Out}_F(t) \) will denote the set of defined-function positions of \( t \), at level \( i \). \( \text{Out}_F(t) \) is sometimes abbreviated to \( \text{Out}_F(t) \).

In the following, we underline terms that are outermost descendants of \( E \) over \( R \).

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

Secondly, we rewrite \( E' \) at level 1. We obtain\(^7\)

\[
E'' = E' \cup \{ h(f(s(a))), \; h(p(f(a))), \; g(f(a)), \; h(f(a)) \}.
\]

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

\[
E'' = \{ g^M(f^M(s^M(a^M))), \; g^M(p(f(a^M))), \; h(f^M(s^M(a^M))), \; h(p(f(a^M))), \; g(f(a^M)), \; h(f(a^M)) \}.
\]

And we keep only terms satisfying the following condition (denoted by (I)): \( \forall p \in \text{Out}_F(t) \)

- \( \neg (t \rightarrow p) \), 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 an outermost derivation. We have shown (see the

\(^7\) Here, it turns out that \( E'' = R^*(E) \). It is not true if \( E \) has more than two nested defined-functions.
correction proof) that if \( t'' \) satisfies (I) and \( R \) has no critical pairs, then we can get an 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 \( \text{IRR}_{\text{outF}}'(R) \). Then, we keep only \( E'' = E'' \cap \text{IRR}_{\text{outF}}'(R) = \{ g^M(f^M(s^M(a^M))), h(f^M(s^M(a^M))), h(f(a^M)) \} \).

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

This is why we introduce the label \( \text{ok} \), to indicate which defined-functions at level 2 can be reduced in one step, respecting the outermost strategy, i.e. the positions \( q \in \text{OutF}_2(t) \) such that \( \neg (\exists p \in \text{OutF}_1(t), (p < q \land t \rightarrow p)) \). Thus

\[
E'' = \{ g^M(f^M(s^M(a^M))), h(f^M,\text{ok})(s^M(a^M)), h(f,\text{ok})(a^M) \}.
\]

Now, we rewrite positions labeled with \( \text{ok} \) in one step, and we obtain the missing term \( h(p(f(a^M))) \).

In the following, this one-step rewriting of \( \text{ok} \)-positions is denoted by \( R_{p \in \text{OutF}_2^1}^{\mid\mid} \).

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

The picture below gives the outermost derivation.

Remark. In the general case, the scheme in parentheses 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 1–3 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. \( L = E^M \mid_p \) for some \( p \in \text{Pos}(t) \).
If $L$ contains only constructor-terms then $R^\ast_{\text{out}}(L) = L$
else if $L = c^M \mid L_1 \ldots \mid L_n$ where $c^M$ is a constructor, then

\[
R^\ast_{\text{out}}(L) = c^M \mid R^\ast_{\text{out}}(L_1) \mid \ldots \mid R^\ast_{\text{out}}(L_n)
\]

or else $L = f^M \mid L_1 \ldots \mid L_n$ where $f^M \in F^M$ and then

\[
R^\ast_{\text{out}}(L) = f^M \mid R^\ast_{\text{out}}(L_1) \mid \ldots \mid R^\ast_{\text{out}}(L_n)
\]

If $L = E^M \mid_p$ for $p \in \text{PosVar}(t)$ then $L$ contains only constructor-terms. Consequently, recursivity of $R^\ast_{\text{out}}$ terminates, and we can build an automaton that recognizes $R^\ast_{\text{out}}(E^M)$ in this way. To get $R^\ast_{\text{out}}(E)$ we just have to remove labels.

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

**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 $R = \{ f(s(x)) \rightarrow h(x), h(x) \rightarrow s(f(x)), g(x) \rightarrow s(g(x)) \}$ and $E = \{ f(g(s^*(a))) \}$. The outermost descendants of $E$ are $\{ s^*(f(s^*(g(s^*(a))))), s^*(h(s^*(g(s^*(a)))))) \}$.

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

\[\begin{array}{c}
| f \\
| s^* \\
| g \\
| s^*(a)
\end{array}\]

Step 2: $R^\ast_{\epsilon}(L_1) = L_1 \cup \{ s^* \}$ (denoted by $L_2$).

\[\begin{array}{c}
| f, h \\
| s^* \\
| g \\
| s^*(a)
\end{array}\]
Step 3: after intersection we obtain \( \{ f \} \cup \{ s^* \} \) (denoted by \( L_3 \)).

\[
\begin{array}{c}
g \quad f \\
\text{s}^*(a) \quad g
\end{array}
\]

Step 4: by applying \( R^\parallel_{p \in \text{Out}F_2} \) on \( L_3 \), we obtain \( L_3 \cup \{ s^* \} \) (denoted by \( L_4 \)).

\[
\begin{array}{c}
f \\
\text{s}^* \quad g
\end{array}
\]

Step 5: by applying \( R^*_{p \in \text{out}F_1} \) on \( L_4 \), we finally obtain \( L_4 \cup \{ s^* \} \).

\[
\begin{array}{c}
h \\
g \\
\text{s}^*(a)
\end{array}
\]

Example 4.3. In this example we see the usefulness of labels.

Let \( R = \{ f(s(x)) \rightarrow s(f(x)), g(x) \rightarrow s(h(x)), h(x) \rightarrow p(x) \} \) and let \( E = \{ f(s(g(s^*(a)))) \} \).

The outermost descendants of \( E \) are:

\[
\{ f(s(g(s^*(a)))) \}, s(f(g(s^*(a)))) \}, s(f(h(s^*(a)))) \}, s(s(f(p(s^*(a)))) \}
\]

Below, \( *^M \) will denote \( s^M(s^*(a^M)) \).

Recall that the algorithm deals with innermost function at first.

Step 1: \( f^M(s^M(R^\parallel_{out}(g^M(s^*(a^M)))) = \{ f^M, f^M, f^M \} \) (denoted by \( L_1 \)).

\[
\begin{array}{cccc}
s^M & s^M & s^M \\
g^M & s & s \\
*^M & h & p \\
\end{array}
\]

Step 2: \( R^*_E(L_1) = L_1 \cup \{ s, s, s, s \} \)

\[
\begin{array}{cccc}
f & f & s & s \\
s^M & s & s & s \\
\text{s}^* \quad h & h & p & p \\
\end{array}
\]
Step 3: after labeling with \(ok\) and intersection, we obtain
\[
\{ f^M, s^M, s^M, s^M \} \text{ (denoted by } L_3). 
\]

Step 4: by applying \(R^\parallel_{p \in \OutF_2}\) on \(L_3\), we obtain \(L_3 \cup \{ s^M \} \) (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 \(R = \{ f(s(x)) \rightarrow f(x), \ f(s(s(x))) \rightarrow g(x), \ g(s(x)) \rightarrow s(g(x)) \}\) and consider the following derivation (as in the previous example, \(*^M\) will denote \(s^M(a^M)\)):

\[
f^M \rightarrow [1, r] \ f^M \rightarrow [1, 1, r] \ f^M \rightarrow [\varepsilon, r] \ g^M \\
g^M \rightarrow s^M \ \ s^M \ \ s^M \ \ g^M \\
*^M \ \ g^M \ \ s^M \ \ *^M \\
*^M \ \ g^M \\
*^M 
\]

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 \(\text{IRR}\_\OutF(R)\), 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 \(\text{Pos}(t)\), and \(\OutF_2(t)\) are the outermost elements of \(\text{Pos}(t) - \OutF_1(t)\).

\[
\OutF_1(t) = \{ p \in \text{Pos}(t) \mid \neg (\exists q \in \text{Pos}(t) \text{ s.t. } q < p) \}
\]

\[
\OutF_2(t) = \{ p \in \text{Pos}(t) \mid p \in \OutF_1(t) \land \neg (\exists q \in \OutF_1(t) \text{ s.t. } q < p) \}
\]
Example 4.6. Let $F = \{ f, g, h \}$:

\[ \text{Out}_1(t) = \{ \epsilon \} \]
\[ t = f \quad \text{Out}_1(t) = \{ 1, 1.1, 2 \} \]
\[ g \quad \text{Out}_2(t) = \{ 1, 2 \} \]
\[ h \]
\[ \overset{a}{f} \]
\[ \overset{s}{g} \]
\[ \overset{h}{a} \]
\[ \overset{a}{f} \]

Remark. With each function $f$ in $F$, we associate a new defined-function $f^{ok}$. Let $F^{ok}$ be the set of functions labeled with $ok$.

The label “ok” will be used for locating some positions of $\text{Out}_2$ that can be rewritten (in one step) respecting the outermost strategy.

Let $\text{Out}_2^{ok}(t) = \{ p \in \text{Pos}(t) | t(p) \in F^{ok} \}$.

Definition 4.7. Let us define an automaton $A_{F - \epsilon} = (C \cup F \cup F^{ok}, Q_{F - \epsilon}, Q^f_{F - \epsilon}, \Delta_{F - \epsilon})$ 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. $\in \text{Out}_2$) is labeled with $ok$. Formally:

\[ L(A_{F - \epsilon}) = \{ t | (t(\epsilon) \in \text{Pos}(t) \land \forall p \in \text{Pos}(t) ((t(p) \in F^{ok}) \Rightarrow (p \in \text{Out}_2^{ok} \land \forall q \neq p, t(q) \notin F^{ok})) \} \]

Let $Q_{F - \epsilon} = \{ q_{any}, q_{any-ok}, q^{\epsilon}_{F - \epsilon} \}$ and $Q^f_{F - \epsilon} = \{ q^f_{\epsilon} \}$.

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

\[ \{ f(q_{any}, \ldots, q_{any}) \rightarrow q^f_{\epsilon} | f \in F \} \]
\[ \cup \{ s(q_{any}, \ldots, q_{any}) \rightarrow q_{any} | s \in C \cup F \} \]
\[ \cup \{ f(q_{any}, \ldots, q_{any}, q_{any-ok}, q_{any}, \ldots, q_{any}) \rightarrow q^f_{\epsilon} | f \in F \} \]
\[ \cup \{ s(q_{any}, \ldots, q_{any}, q_{any-ok}, q_{any}, \ldots, q_{any}) \rightarrow q_{any-ok} | s \in C \} \]
\[ \cup \{ f^{ok}(q_{any}, \ldots, q_{any}) \rightarrow q_{any-ok} | f^{ok} \in F^{ok} \} \].

Definition 4.8. Let $A$ be an automaton. $A^{ok}$ is the automaton obtained from $A$ by adding the label “ok” randomly on the defined-functions. This can be effected by the following transformation of the set of transitions $\Delta$ of $A$:

\[ \forall f \in F, \forall f(q_1, \ldots, q_n) \rightarrow q \in \Delta, \text{add } f^{ok}(q_1, \ldots, q_n) \rightarrow q. \]

If $L$ is the language recognized by $A$, we denote by $L^{ok}$ the language recognized by $A^{ok}$.

Definition 4.9. Consider again the automaton $A_{irre}$ that recognizes the set of terms that are not reducible at position $\epsilon$, on the signature of $R$. Let $A_{irre-ok} = (A_{irre})^{ok}$.

$IRR_{Out}(R)$ is the language of terms $t$ s.t. each position $p \in \text{Out}_1(t)$ satisfies: either $t \not\rightarrow p$ or $t|_p$ comes from the starting language $E$. In the first case, we can reduce (in one step) one arbitrary $p' \in \text{Out}_2(t)$ s.t. $p' > p$, respecting the outermost strategy. We locate such $p'$ by labeling the symbol of $t$ occurring at $p'$ with $ok$. Moreover, in both cases, if $t \in IRR_{Out}(R)$ is a descendant of $E$ obtained by a non-outermost derivation, then $t$ can also be obtained from $E$ by an outermost derivation (see the correctness proof). In the
second case, to check that \( t \models p \) 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 \( A_{\text{irr}F-\epsilon} \) be the automaton obtained by intersection between \( A_{\text{irr},ok} \) and \( A_{F-\epsilon} \).

Let \( A'_{\text{irr}F-\epsilon} = (C \cup CM \cup F \cup FM \cup F^ok, Q_{\text{irr}F-\epsilon}, Q_{\text{irr}F-\epsilon}^f, \Delta_{\text{irr}F-\epsilon}) \) be the automaton obtained by modifying \( A_{\text{irr}F-\epsilon} \) in order to add some labels \( M \) randomly on the symbols of recognized terms.

**Definition 4.11.** Let us define the language:

\[
IRR_{OutF}'(R) = \{ t \mid \forall p \in OutF_1(t) \land t \rightarrow_p v \land \forall u \land t(t) \in C^M \land F^M \land \forall v \in Pos(t) ((v > p \land t(v) \in F^ok) \Rightarrow (v \in OutF_2(t) \land t \rightarrow_p w \land \exists w \in Pos_F(t), w > p \land w \neq v \land t(w) \in F^ok) \})
\]

which is recognized by the automaton

\[
A_{IRR_{OutF}'} = (C \cup CM \cup F \cup FM \cup F^ok, Q_{IRR_{OutF}}', Q_{IRR_{OutF}}^f, \Delta_{IRR_{OutF}}')
\]

where \( Q_{IRR_{OutF}}' = Q_{\text{irr}F-\epsilon}^f \cup \{ q_{\text{out}}, q_{\text{out}}, q, q_{\text{out}} \} \) and \( Q_{IRR_{OutF}}^f = \{ q_{\text{irr}F-\epsilon}^f \} \), and where \( \Delta_{IRR_{OutF}}' \) is the following set of transitions:

\[
\Delta_{\text{irr}F-\epsilon}'
\]

\[
\bigcup \{ q_{\text{irr}F-\epsilon} \rightarrow q_{\text{irr}F-\epsilon}^f \mid q_{\text{irr}F-\epsilon} \in Q_{\text{irr}F-\epsilon}' \}
\]

\[
\bigcup \{ q_{\text{out}} \rightarrow q_{\text{out}}^f \mid q_{\text{out}} \in Q_{\text{irr}F-\epsilon}' \}
\]

\[
\bigcup \{ f(q_M, \ldots, q_M) \rightarrow q_{\text{out}} \mid f \in F \cup F^M \}
\]

\[
\bigcup \{ q_{\text{out}} \rightarrow q_{\text{irr}F-\epsilon}^f \mid q_{\text{out}} \in Q_{\text{irr}F-\epsilon}' \}
\]

**Example 4.12.** Let \( R = \{ f(x, y) \rightarrow c(g(x), h(y)), g(s(x)) \rightarrow s(g(x)), h(s(s(x))) \rightarrow s(h(x)), i(s(x)) \rightarrow s(i(x)) \} \).

The following terms are in \( IRR_{OutF}'(R) \):

\[
\begin{array}{|c|c|}
\hline
\rightarrow & \rightarrow \\
\hline
s^M & s^M \\
\hline
\_ & \_ \\
\hline
s^M & s^M \\
\hline
\_ & \_ \\
\hline
i^M & i^M \\
\hline
\_ & \_ \\
\hline
s^M(a^M) & s^M(a^M) \\
\hline
\_ & s(a) \\
\hline
\end{array}
\]

Definition 4.13. $R^2_{p\in Out F^ok_2}(t)$ means parallel rewrite in at most one step at function positions identified by an $ok$ label. By construction, these positions are in $Out F_2(t)$.

Formally, $R^2_{p\in Out F^ok_2}(t) = \{ t' | t \rightarrow^*_{p_1,\ldots,p_n} t' \wedge p_1,\ldots,p_n \in Out F^ok_2 \}$.

This consists on carrying out a single saturation process on the TRS $R^ok$ where $R^ok = \{ l' \rightarrow r | l \rightarrow R \wedge l' = I[\epsilon \leftarrow l(\epsilon)^ok] \}$.

Definition 4.14 (Labeled-term Rewriting). Let $s$ be a term that may contain symbols labeled with $M$. $s \rightarrow^{[p, i \rightarrow r, \sigma]} s'$ if there exists a term $l M$ obtained from $l$ by labeling some positions with $M$, s.t. $s\big|_p = l M \sigma$ 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 the 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, $s'$ has been modified.

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

\[
\begin{array}{cccc}
  s^M & \rightarrow^* & c & \rightarrow^2 & c \\
  s^M \quad s^M & g & h & g & s \\
  i & i^M & s^M & s^M & i^M \\
  s^M(a^M) & s^M(a^M) & i & i^M & s^M(a^M)
\end{array}
\]

4.4. Correctness proof

Recall that we consider only constructor-based TRS’s. The following lemmas show that $(R^*_n(f^M(R^out_1(L_1)), \ldots, R^out_1(L_n)))^{ok} \cap IR^\prime_{out}(R)$ is correct. By construction, $ok$ locates positions of $Out F_2$ that can be reduced, respecting the outermost strategy, and moreover rewriting $p \in Out F_1$ is necessarily outermost.

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

$\forall i, \neg (\exists q \in Out F_1(s_i) \ \mathrm{s.t.} \ q < p_i \wedge s_i \rightarrow q)$. 

Example 4.17. Let $R = \{ f(x) \rightarrow s(f(x)), h(x) \rightarrow s(i(x)), i(x) \rightarrow s(x), g(s(x)) \rightarrow s(x) \}$ and $r = f(g(h(s(a))))$. Let $E$ be the set of data-instances of $r$, $E = \{ f(g(h(s^*(a)))) \}$.
Notation. Let us denote by τ a fictitious position s.t.
\[ s \rightarrow \tau \] means that \( s = \tau \).

Definition 4.18. Let \( s_0 \rightarrow_{p_0,l_0} s_1 \rightarrow_{p_1,l_1} s_2 \). 
\( p_0 \) admits a residue \( q \) into \( s_2 \), which is denoted by \( \text{res}(p_0, s_2) \), if:
- \( p_0 = p_1.v.w \) where \( v = \text{occ}(x, l_1) \)
- and if \( \text{occ}(x, r_1) \) exists then \( v' = \text{occ}(x, r_1) \) and \( q = p_1.v'.w \)
and otherwise, \( q = \tau \).

Remark. Due to the linearity of TRS, the residue \( q \) is unique, and \( p_0 \geq p_1 \land (q \geq p_1 \lor q = \tau) \).

Example 4.19. \( R = \{ f(s(x)) \xrightarrow{\tau} c(a, f(x)), g(x) \xrightarrow{\tau} x \} \):

\[
\begin{array}{c}
| \hspace{1cm} f \\
| f \rightarrow_{[1.1, r_2]} \hspace{1cm} c \\
| s \rightarrow_{[\varepsilon, r_1]} \hspace{1cm} f \\
| s \rightarrow \hspace{1cm} a \\
| g \rightarrow \hspace{1cm} s^\tau(a) \\
| g \rightarrow \hspace{1cm} s^\tau(a) \\
\end{array}
\]

\( \text{res}(1.1, s_2) = 2.1 \).

Definition 4.20. A derivation in two steps \( s_0 \rightarrow_{p_0,l_0} s_1 \rightarrow_{p_1,l_1} s_2 \) 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:
\[
\begin{array}{c}
\begin{array}{c}
| \hspace{1cm} s_0 \\
| s_0 \rightarrow_{p_0,l_0} s_1 \\
| s_1 \rightarrow_{p_1,l_1} s_2 \hspace{1cm} (1) \\
\end{array}
\end{array}
\]
where \( p_0 \) admits a residue \( q \) into \( s_2 \). Then, (1) can be commuted into
\[
\begin{array}{c}
\begin{array}{c}
| \hspace{1cm} s_0 \\
| s_0 \rightarrow_{p_1,l_1} s_1 \\
| s_1 \rightarrow_{q,l_0} s_2 \hspace{1cm} (2) \\
\end{array}
\end{array}
\]
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
\[
\begin{array}{c}
\begin{array}{c}
| \hspace{1cm} f \\
| f \rightarrow_{[\varepsilon, r_1]} \hspace{1cm} c \\
| s \rightarrow_{[2.1, r_2]} \hspace{1cm} c \\
| s \rightarrow \hspace{1cm} a \\
| g \rightarrow \hspace{1cm} g \\
| g \rightarrow \hspace{1cm} s^\tau(a) \\
| s^\tau(a) \rightarrow \hspace{1cm} s^\tau(a) \\
\end{array}
\end{array}
\]

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 \( v = \text{occ}(x, l_1) \), and
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 \( p_1 \in \text{OutF}_1(s_0) \).

The case \( q = \tau \) is trivial.

By induction, ...

\begin{equation}
\text{Lemma 4.25.}\quad \text{Let us consider the following derivation:}
\end{equation}

\[ s_0 \rightarrow p_n s_n \rightarrow s_n + 1 \]

where \( p_n \) 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

\[ s_0 \rightarrow p_1 s_1 \rightarrow p_2 s_2 \rightarrow p_n s_n \rightarrow q_{n+1} s_{n+1} \]

(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 found up to the previous case are \( \not= \tau \), on the assumption that \( q_{n+1} \) exists.

\( s_0 \rightarrow p_n s_1 \rightarrow p_1 s_2 \) is an outermost derivation at level 2 s.t. \( \text{res}(p_0, s_2) = q_2 \) and can be commuted into \( s_0 \rightarrow p_1 s_1 \rightarrow q_2 s_2 \) which 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. \( \text{res}(q_2, s_3) = q_3 \) and can be commuted into \( s_1 \rightarrow p_2 s_2 \rightarrow q_3 s_2 \) which is outermost at level 2 and without residues.

\[ \ldots \]

By induction, \( s_{n-1} \rightarrow q_n \) is an outermost derivation at level 2 s.t. \( \text{res}(q_n, s_{n+1}) = q_{n+1} \) and can be commuted into \( s_{n-1} \rightarrow p_n s'_n \rightarrow q_{n+1} s_{n+1} \) which is outermost at level 2 without residues.

So, we finally obtain the expected property. \( \square \)
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 \( s_0 \rightarrow^* s_{n+1} \) be a derivation with residue(s). Let us consider the biggest \( i \) s.t. \( s_i \rightarrow^* s_{n+1} \) is with residue and \( s_{i+1} \rightarrow^* s_{n+1} \) is without residues. Now, let us take \( s_i \rightarrow^* s_{n+1} \) and consider the biggest \( j \) s.t. \( p_i \) admits a residue \( p_j \) into \( s_i \). Then, \( s_i \rightarrow^* s_{n+1} \) satisfies the assumptions of Lemma 4.25; therefore it can be commuted into a derivation without residues. In this way, we get a derivation \( s_i \rightarrow^* s_{n+1} \) 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 \( s_0 \rightarrow^* s_n \) be a derivation without residues, outermost at level 2, and s.t. \( \forall p \in \text{OutF}_1(s_n), s_n \not \rightarrow p \). Then, \( s_0 \rightarrow^* s_n \) is outermost.

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

Counter-example 4.28. Let \( R = \{ f(s(s(x))) \rightarrow g(x); f(s(s(s(x)))) \rightarrow g(x); f(s(s(s(s(x))))) \rightarrow g(x) \} \) and let us look at the following derivation:

\[
\begin{array}{c}
\text{f} \rightarrow_{[1, r3]} \text{f} \rightarrow_{[1, r3]} \text{f} \rightarrow_{[\epsilon, r2]} \text{g} \\
\text{g} \rightarrow s \rightarrow s \rightarrow \text{g} \\
\text{g} \rightarrow s \rightarrow s \rightarrow \text{g} \\
\text{s}^\ast(a) \rightarrow \text{g} \rightarrow s \rightarrow \text{s}^\ast(a) \\
\text{s}^\ast(a) \rightarrow \text{s}^\ast(a) \\
\text{s}^\ast(a)
\end{array}
\]

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

Proof. (of Lemma 4.27). Let us suppose that \( s_0 \rightarrow^* s_n \) is not outermost. Let us 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. By hypothesis, \( s_j \rightarrow p_i s_{i+1} \) is not outermost but is outermost at level 2. Then, \( \exists p \in \text{OutF}_1(s_i) \) s.t. \( s_j \rightarrow_{[p, j \rightarrow r]} p < p_i \). Because of the constructor discipline, \( p_i = p . q . w \) where \( q = \text{occ}(x, l) \), and \( s_{i+1} \rightarrow_{[p, j \rightarrow r]} \).

Since \( s_{i+1} \rightarrow_{[p_i, p_{i+1} \rightarrow r_i]} s_{i+2} \) is outermost, we have \( p_i \| p \) or \( p_{i+1} = p \):

- \( p_{i+1} \not< p \) is impossible because \( p \in \text{OutF}_1(s_{i+1}) \).
- \( p_{i+1} = p \) means that we have \( p_i = p . q . w \) where \( q = \text{occ}(x, l) \). Since the rewrite system is without critical pairs, \( l_{i+1} = l \) and then \( q = \text{occ}(x, l_{i+1}) \). So, \( p_i \) admits a residue into \( s_{i+2} \), but by hypothesis, the derivation \( s_0 \rightarrow^* s_n \) is without residues.
- \( p_{i+1} \| p \) is possible. We prove in the same way that we proved \( p_{i+2} \| p \ldots , p_{n-1} \| p. \)
Then, \( s_n|_p = s_{i+1}|_p \), so \( s_n \rightarrow p \). This is impossible since according to the hypothesis \( \forall p \in \text{OutF}_1(s_n), s_n \not\rightarrow p \). □

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. \( \forall p \in \text{OutF}_1(s_n) \), either \( s_n \not\rightarrow p \), or \( \exists q > p, s_n(q) \) is M-labeled. Then, \( s_0 \rightarrow^* s_n \) is outermost.

Proof. For the case \( s_0 \not\rightarrow p \), 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 \( \exists p \in \text{OutF}_1(s_i) \) s.t. \( s_i \rightarrow [p_i, l \rightarrow r_i] \) 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 \( \exists q > p, s_i(q) \) is non-labeled and, so, \( \neg(\forall q > p, s_n(q) \text{ M-labeled}) \). □

4.5. Completeness proof

Recall that we consider only constructor-based TRS’s.

Let us consider the property \( P \) on terms, defined by

\[ P(t) = (\forall p \in \text{OutF}_1(t), t \not\rightarrow p \lor \forall u > p, t(u) \in C^M \cup F^M). \]

Note that \( \neg P(t) = (\exists p \in \text{OutF}_1(t), t \rightarrow p \land \exists u > p, t(u) \text{ not labeled}) \). 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. \( s_0 \rightarrow^* s_i \rightarrow^* [p_i, \ldots, p_{i-1}] s_n \) and \( \forall j > i, \neg P(s_j) \). According to Lemma 4.39, we can suppose that \( \forall j > i, p_j \in \text{OutF}_1(s_j) \).

According to Corollary 4.37 applied to \( s_0 \rightarrow^* s_i \) (let us suppose that \( s_0(\epsilon) = f^M \)), we obtain that \( s_i \in R^*_f(f^M(R^*_\text{out}(s_0|_1), \ldots, R^*_\text{out}(s_0|_k))) \). Moreover, \( P(s_i) \); then we have \( s_i \in \text{IRR}_{\text{out}}(R) \).

According to Lemma 4.44 applied on \( s_i \rightarrow^* s_n \), there exists a derivation of the form

\[ s_i \rightarrow^* |_{p_{\text{out}}F_2} \rightarrow^* |_{\text{out}}F_1 \ s_n. \]

Then, \( s_n \in R^*_{\text{out}}(s_0) \).

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\(^\dagger\) \( q_1 \) into \( s_0 \) (denoted by \( q_1 = \text{ant}^\dagger(p_1, s_0) \)), if:

- either \( p_1 \parallel p_0 \) and, so, \( q_1 = p_1 \);
- or \( p_1 = p_0.q.w \) where \( q = \text{occ}(x, r_0) \) and \( q_1 = p_0.q'.w \) where \( q' = \text{occ}(x, l_0) \).

Remark. Do not confuse antecedent\(^\parallel\) (defined above) with antecedent (defined in Section 5). The only difference is: \( p_1 \parallel p_0 \) (in antecedent\(^\parallel\)) is replaced by \( p_1 \not\parallel p_0 \) (in antecedent).
Lemma 4.31. Let us consider the following derivation:

\[ s_0 \rightarrow p_0 \in OutF(s_0) \rightarrow t_0 \rightarrow s_1 \rightarrow p_1.l_1 \rightarrow t_1 \rightarrow s_2 \quad (1) \]

where \( p_1 \) admits an antecedent \( \parallel q_1 \) into \( s_0 \). Then, (1) can be commuted into

\[ s_0 \rightarrow q_1.l_1 \rightarrow t_1 \rightarrow s' \rightarrow p_0.l_0 \rightarrow t_0 \rightarrow s_2 \quad (2) \]

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

**Proof.** Obviously, in (2), \( p \in OutF(s'1) \). Let us show the second property (the first is trivial). Let us suppose that (1) is outermost and \( p_1 \) admits an antecedent \( \parallel q_1 \) into \( s_0 \); then,

- either \( p_1 \parallel p_0 \) and, so, \( q_1 = p_1 \); then it is trivial;
- or \( p_1 = p_0.q.w \) where \( q = \text{occ}(x, r_0) \) and \( q_1 = p_0.q'.w \) where \( q' = \text{occ}(x, l_0) \).

In (1), \( p_1 \in PosF(s_1) \) s.t. \( \neg(\exists q \in PosF(s_1) q < p_1 \land s_1 \rightarrow q) \) because (1) is outermost. \( l_0 \) does not contain nested defined-functions (because of the constructor discipline); then, \( \neg(\exists u \in p_0 < u < p_0.q'; \ u \in PosF(s_0)) \). Moreover, \( \neg(\exists v \in PosF(s_0) p_0.q' \leq v < p_0.q'.w; \ s_0 \rightarrow v) \); otherwise \( v' \in PosF(s_1) p_0.q \leq v' < p_0.q.w; \ s_0 \rightarrow v' \) which contradicts the fact that (1) is outermost.

So, (2) is outermost at level 2. \( \square \)

Lemma 4.32. Suppose \( s_0(\epsilon) \in F \cup F^M \).

Let us consider the following derivation:

\[ s_0 \rightarrow \ast_{[\epsilon, r h s's]} s_n \rightarrow p_n \rightarrow s_{n+1} \quad (1) \]

s.t. \( p_n \) admits an antecedent \( \parallel q \) into \( s_0 \). Then (1) can be commuted into

\[ s_0 \rightarrow p_n \rightarrow s'_{n} \rightarrow \ast_{[\epsilon, r h s's]} s_{n+1} \quad (2) \]

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 \( s_0 \rightarrow \ast_{[\epsilon, r h s's]} s_n \):

- if \( \text{length} = 0 \), (2) = (1) and the result is trivial.
- otherwise, suppose \( (1) \equiv (s_0 \rightarrow \epsilon \ s_1 \rightarrow p_1 \ s_2 \rightarrow \ldots \rightarrow p_{n-1} \ s_n \rightarrow p_n \ s_{n+1}) \); by applying Lemma 4.31 on the last two steps, we get

\[ s_0 \rightarrow \epsilon \ s_1 \rightarrow p_1 \ s_2 \rightarrow \ldots \rightarrow s_{n-1} \rightarrow q_{n-1} \rightarrow s'_{n} \rightarrow p_n \ s_{n+1} \quad (1') \]

where \( q_{n-1} = \text{ant}^{1}(p_n, s_{n-1}) \). If (1) is outermost then so is \( (1') \).

We get (2) by applying the induction hypothesis on \( s_0 \rightarrow \ast_{[\epsilon, r h s's]} \) in \( (1') \). \( \square \)

**Remark.** If \( p_n \) admits an antecedent \( \parallel q \) into \( s_{n-1} \), then \( p_n \) admits an antecedent \( \parallel q \) into \( s_0 \).

Lemma 4.33. Suppose \( s_0(\epsilon) \in F \cup F^M \).

Let us consider the following derivation:

\[ s_0 \rightarrow \ast_{[\epsilon, r h s's]} s_n \rightarrow p_n \rightarrow s_{n+1} \quad (1) \]
s.t. $p_n$ does not admit an antecedent\# into $s_{n-1}$. Then, (1) is a derivation of the form

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

**Proof.** Let us suppose that $[\epsilon,\text{rhs's}] = [\epsilon, p_1, \ldots, p_{n-1}]$, and let $l_i \rightarrow r_j$ 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 $q \in \text{PosF}(p_{n-1})$. By construction, $p_{n-1} \in \text{OutF}_1(s_n)$; 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 $s_0 \rightarrow^*_{[\epsilon,\text{rhs's}]} s_{n+1}$. □

**Lemma 4.34.** Let us consider the following derivation:

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

Then (1) can be commuted into

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

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, and we obtain the following derivation that is outermost at level 2:
    $$s_0 \rightarrow^*_{\neq \epsilon} s'_i \rightarrow^*_{[\epsilon,\text{rhs's}]} s_{n+1} \rightarrow^* s_k \ (1')$$
  - else, according to Lemma 4.33, (1) is of the form
    $$s_0 \rightarrow^*_{[\epsilon,\text{rhs's}]} s_{n+1} \rightarrow^* s_k \ (1')$$

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

**Example 4.35.** $R = \{ f(s(x)) \rightarrow s(f(x)), \ g(s(x)) \rightarrow s(g(x)) \}$

$$
\begin{array}{cccccccccc}
& & & & & & & & & \\
& & & & f \rightarrow_{\epsilon} & s \rightarrow_{1.1} & s \rightarrow_{1} & s \\
& & & & | & | & | & | \\
& & & & s & f & s & s \\
& & & & | & | & | & | \\
& & & & g & g & s & f \\
& & & & | & | & | & | \\
& & & & s & s & s & f \\
& & & & | & | & | & | \\
& & & & a & a & a & a \\
\end{array}
$$

**Lemma 4.36.** Let $s_0$ be a term s.t. $s_0(\epsilon) \in F \cup F^M$. If $s_0 \rightarrow^* s'$ is outermost, then this derivation can be commuted into $s_0 \rightarrow^{p \neq \epsilon}_{\neq \epsilon} s'' \rightarrow^*_{[\epsilon,\text{rhs's}]} s'$, which is outermost at level 2.
Proof. Let us consider the outermost derivation \( s_0 \rightarrow^* s' \) where \( s_0(e) \in F \cup F^M \). Let us take the smallest \( i \) s.t. \( s_0 \rightarrow^* s_i \rightarrow^e s_{i+1} \rightarrow^* s' \). By Lemma 4.34 applied on \( s_i \rightarrow^* s' \), we obtain \( s_0 \not\rightarrow^*_{p \neq e} s'' \rightarrow^*_{[e, \text{rhs}]} s' \), which is outermost at level 2. \( \square \)

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

\[
s' \in R^*_r(s) \setminus R^*_r(s_1) \setminus R^*_r(s_n)
\]

Remark. Recall that the property \( P \) is defined at the beginning of Section 4.5.

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

Proof. If \( s_0 \rightarrow_{[e, l \rightarrow r, \sigma]} s' \) then \( \forall u > \epsilon, s_0(u) \in C^M \cup F^M \); then \( \forall x, x \sigma \in T_{C^M \cup F^M}, \forall q \in \text{OutF}_1(s') \), \( \forall u > q, s'(u) \in C^M \cup F^M \) (\( P'(s') \)).

Since \( s_0 \rightarrow p_1 s_1 \) is outermost and \( s_0 \not\rightarrow^* \epsilon s' \), we have \( p_0 = \epsilon \). And since there are no critical pairs, \( s' = s_1 \). Then, we have \( \neg P(s') \) (because \( \neg P(s_1) \)) and \( P'(s') \), and this is impossible. \( \square \)

Lemma 4.39. Let us suppose that \( s_0(e) \in F \cup F^M \) and there are no critical pairs. Let \( s_0 \rightarrow p_1 s_1 \rightarrow^* s_n (1) \) be an outermost derivation s.t. \( P(s_0) \) and \( \forall \epsilon \in \{1, \ldots, n\} \neg P(s_\epsilon) \).

If there exists \( i \in \{1, \ldots, n-1\} \) s.t. \( s_i \rightarrow_{p_i} s_{i+1} \) satisfies \( p_i \not\in \text{OutF}_1(s_i) \), then \( p_i \) admits an antecedent \( \hat{q} \) into \( s_0 \) and we can commute (1) into:

\[
s_0 \rightarrow q s'_1 \rightarrow p_0 \rightarrow \ldots s'_i \rightarrow s_{i+1} \rightarrow^* s_n (2) \text{ s.t. (2) is outermost and } P(s'_1) \text{ and } \forall f \in \{2, \ldots, i\} \neg P(s'f).
\]

Proof. Let us consider the smallest \( i \) s.t. \( p_i \not\in \text{OutF}_1(s_i) \). \( s_i \rightarrow_{p_i} s_{i+1} \) being outermost, and 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, \( s_0 \not\rightarrow^* \epsilon \), thus \( s_0 \rightarrow q s_1 \) is outermost. \( \square \)

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

\[
\begin{array}{ccccccc}
  f & \rightarrow & 2 & f & \rightarrow & \epsilon & c \\
  g & \rightarrow & g & s & h & f & h \\
  s & s & s & s & g & a & s \\
  a & a & a & g & s & g \\
  a & a & a & g & a \\
  a
\end{array}
\]
ant\(\parallel\) (1, 1, \(s_0\)) = 1, then it can be commuted into
\[
\begin{array}{cccccccccc}
f & \rightarrow & f & \rightarrow & 2 & f & \rightarrow & 3 & c \\
\backslash / & \backslash / & \backslash / & \backslash / & \backslash / & \backslash / & \backslash / & \backslash / & \backslash /
\end{array}
\]
\[
\begin{array}{cccccccccc}
g & g & s & g & s & s & h & f \\
\backslash / & \backslash / & \backslash / & \backslash / & \backslash / & \backslash / & \backslash / & \backslash /
\end{array}
\]
\[
\begin{array}{cccccccccc}
s & s & s & s & s & s & a & s \\
\backslash / & \backslash / & \backslash / & \backslash / & \backslash / & \backslash / & \backslash / & \backslash /
\end{array}
\]
\[
\begin{array}{cccccccccc}
a & a & g & a & g & g & s & g \\
\backslash / & \backslash / & \backslash / & \backslash / & \backslash / & \backslash / & \backslash / & \backslash /
\end{array}
\]
\[
\begin{array}{cccccccccc}
a & a & a & g & a \\
\backslash / & \backslash / & \backslash / & \backslash / & \backslash / & \backslash / & \backslash / & \backslash /
\end{array}
\]
\[
\begin{array}{cccccccccc}
a
\end{array}
\]

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 \(s_0 \rightarrow_{p\in OutF_2} p_1 \rightarrow_{p\in OutF_1} s_n\).

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 no nested functions in lhs’ s, then \(p_0 \in OutF_2(s_0)\). So, (1) is of the form \(s_0 \rightarrow_{p\in OutF_2} p_1 \rightarrow_{p\in OutF_1} s_n\). □

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

Lemma 4.43. Let us consider the following outermost derivation:
\[
s_0 \rightarrow q_0 \in OutF_1(s_0) \ s_1 \rightarrow \ldots \rightarrow q_{n-1} \in OutF_1(s_{n-1}) \ s_n \ (1)
\]
s.t. \(OutF_1(s_0) = \{p_1, \ldots, p_k\}\). Then, (1) can be commuted into
\[
s_0 \rightarrow_{[\geq p_1]} s_1 \rightarrow_{[\geq p_k]} s_n \ (2)
\]
s.t. (2) is outermost.

Proof. It is trivial. □

Lemma 4.44. Let us suppose that there are no critical pairs and let us consider the following outermost derivation:
\[
s_0 \rightarrow s_1 \rightarrow_{\epsilon} s_n \ (1)
\]
where \(P(s_0)\) and \(\forall k \in \{1, \ldots, n\}\) \(\neg P(s_k)\). Then, (1) can be commuted into
\[
s_0 \parallel \rightarrow_{p\in OutF_2} p^0 \rightarrow_{p\in OutF_1} s_n \ (2)
\]
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
\[
s_0 \rightarrow_{[\geq p_1]} s_1 \ldots s_j \rightarrow_{[\geq p_k]} s_n \ (1)
\]
By Lemma 4.41 applied on each sub-derivation, we obtain that each sub-derivation is of the form \( s_0 \overset{3}{\longleftarrow}_{p \in \text{OutF}_2} \overset{*}{\longleftarrow}_{p \in \text{OutF}_1} \cdot \cdot \cdot \).
To obtain (2), we have to transfer every position of OutF_2 at the beginning of the derivation (this changes nothing because of incomparability). And, finally, we obtain \( s_0 \overset{\|}{\rightarrow}_{p \in \text{OutF}_2} s' \overset{*}{\rightarrow}_{p \in \text{OutF}_1} s_n \). □

5. Leftmost descendants: \( R^*_{\text{left}}(E) \)

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 \( \text{eat} \) is introduced, and \( \text{eat}(t, t') \) 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 \( \text{eat} \). By introducing \( \text{eat} \) 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 \( \text{eat} \) into account. The method is explained by the following example, and an algorithm is given in Appendix D.

Example 5.1. Let \( R = \{ f(s(s(x)), y) \rightarrow x \} \).
After running the algorithm, we obtain the following TRS:
\[
R = \{ f(s(s(x)), y) \rightarrow \text{eat}(x, y), f(\text{eat}(s(s(x)), x_1), y) \rightarrow \text{eat}(x, \text{eat}(x_1, y)),
\]
\[
f(s(\text{eat}(s(s(x), x_{1_1})), y) \rightarrow \text{eat}(x, \text{eat}(x_{1_1}, y)),
\]
\[
f(\text{eat}(s(\text{eat}(s(s(x), x_{1_1_1})), x_1), y) \rightarrow \text{eat}(x, \text{eat}(x_{1_1_1}, \text{eat}(x_1, y))))\}.
\]

\( R^*_{\text{left}}(E) \) does not take the leftmost strategy into account. We will use \( R^*_{\text{left}}(E) \) instead.

Definition 5.2. Given a language \( E \) and a position \( p \), we define \( R^*_{p}(E) \) as follows:
\[
R^*_{p}(E) = E \cup \{ t' \mid \exists t \in E, t \overset{+}{\rightarrow}_{[p, \text{rhs}]} t' \text{ by leftmost rewriting} \}.
\]

Example 5.3. \( R = \{ f(x) \rightarrow s(x), g(x, y) \rightarrow c(h(x), f(y)), h(x) \rightarrow f(x) \} \)
\[
R^*_{\text{left}}(\{ f(g(a, b)) \}) = \{ f(g(a, b)) \} \cup \{ f(c(h(a), f(b))) \} \cup \{ f(c(f(a), f(b))) \} \cup \{ f(c(s(a), f(b))) \} \cup \{ f(c(s(a), s(b))) \}.
\]

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 positions \( p \) into the state \( q \), and possibly \( p' \) into \( q' \) for some \( p' \in \text{Pos}(t) \) s.t. \( p' \nsubseteq p \), and some states \( q' \), then so is \( R^*_{p}(E) \).

Proof. Building an automaton and proving its correctness is not easy. See Section 5.2. □

Definition 5.5. Given a language \( L \) and a position \( p \), we define
\[
R^*_{\text{left}, L}(p) = \{ s' \mid \exists s \in L, s \overset{*}{\rightarrow}_{[u_1, \ldots, u_n]} s' \text{ by a leftmost strat. with } \forall i, u_i \geq p \}.
\]

Lemma 5.6. Let \( R \) be a constructor-based TRS satisfying restrictions 1–3, 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,

\[
R^*_\text{left,}p(L) = R^*_p(L) \text{ if } \text{Succ}_t(p) = \emptyset.
\]

Otherwise, let \( \text{Succ}_t(p) = \{p_1, \ldots, p_n\} \) with \( p_1 \prec \ldots \prec p_n \):

\[
R^*_\text{left,}p(L) = \left[ R^*_p[R^*_\text{left,}p_1(L)] \right]
\]

\[
\cup \left[ R^*_p[R^*_\text{left,}p_2((R^*_\text{left,}p_1(L) \cap \text{IRR}_{p_1}(R)))] \right]
\]

\[
\cup \ldots \cup R^*_p[R^*_\text{left,}p_n((R^*_\text{left,}p_1(L) \cap \text{IRR}_{p_1}(R)) \ldots \cap \text{IRR}_{p_n-1}(R))]
\]

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

**Proof.**

- If \( \text{Succ}_t(p) = \emptyset \), 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 positions that are incomparable (as opposed to nested). \( R^*_p(L) \) compute leftmost. Therefore, \( R^*_p(L) = R^*_\text{left,}p(L) \).

  We get \( A' \) by Theorem 5.4.

- Let \( \text{Succ}_t(p) = \{p_1, \ldots, p_n\} \) with \( p_1 \prec \ldots \prec p_n \). Let \( s \in L, \forall i \in \{1, \ldots, n\}, s \) can be rewritten leftmost at position \( p_i \), but descendants at position \( p_{i-1} \) must have computed and normalized at this position before, since \( p_{i-1} \prec p_i \). Let us denote as \( L_1 \) the set of terms obtained from \( s \) after leftmost rewriting at position \( p_1 \), and as \( L_n \) the set of terms obtained from \( s \) after leftmost rewriting at position \( p_n \). We have:

  \[
  L_1 = R^*_\text{left,}p_1(L)
  \]

  \[
  L_2 = R^*_\text{left,}p_2(R^*_\text{left,}p_1(L) \cap \text{IRR}_{p_1}(R))
  \]

  \[
  \ldots
  \]

  \[
  L_n = R^*_\text{left,}p_n(L_{n-1} \cap \text{IRR}_{p_{n-1}}(R)).
  \]

**Remark.** Those descendants are obtained by left-basic rewriting.

\( L_1, \ldots, L_n \) can be possibly rewritten at position \( p \). \( R^*_p(L_1) \cup \ldots \cup R^*_p(L_n) = R^*_\text{left,}p(L) \).

\( L \) is recognized by an automaton \( A \) that discriminates every position \( p' \in PosF(t) \) s.t. \( p' \prec p \) and so, since \( \forall i \in \{1, \ldots, n\}, p_i > p \), every position \( p' \prec p \).

By the induction hypothesis, \( L_1 \) is recognized by an automaton \( A_1 \) that discriminates every position \( p' \in PosF(t) \) s.t. \( p' \prec p_1 \) and so, in particular, every position \( p' \geq p_i \), \( \forall i \in \{2, \ldots, n\} \), and every position \( p' \ngeq p \). By Theorem 5.4, \( R^*_p(L_1) \) is recognized by an automaton that discriminates positions \( p' \nless p \).

By Theorem 2.14, \( \text{IRR}_{p_j}(R) \) is recognized by an automaton that discriminates every position \( p' \in PosF(t) \) s.t. \( p' \ngeq p_j \) (i.e. positions that are discriminated before computing \( R^*_\text{left,}p_{j-1}(\ldots) \) minus positions that are below \( p_{j-1} \)). By Lemma 2.8, \( L_{j-1} \cap \text{IRR}_{p_{j-1}}(R) \) is recognized by an automaton that discriminates every position \( p' \nless p_{j-1} \),
so in particular, $p' \geq p_j$. $L_j$ is recognized by an automaton that discriminates every position $p' \neq p_j$ (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 $p' \neq p$. By Theorem 5.4, $R^*_{p^s}(L_j)$ is recognized by an automaton that discriminates every position $p' \neq p$.

Finally, by union, we obtain an automaton that discriminates $p$ and preserves discrimination of positions $p' \neq p$. □

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

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

Otherwise, let $Succ_c(p) = \{p_1, \ldots, p_n\}$ with $p_1 < \ldots < p_n$;

$$R^*_{left}(E) = \begin{cases} R^*_{left,p_1}(E) \\ \cup R^*_{left,p_2}((R^*_{left,p_1}(E) \cap IRR_{p_1}(R))) \\ \cup \ldots \cup R^*_{left,p_n}((R^*_{left,p_1}(E) \cap IRR_{p_1}(R)) \ldots \cap IRR_{p_{n-1}}(R)) \end{cases}$$

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

**Proof.** We have two cases:

- If $e \in PosF(t)$, obviously $R^*_{left}(E) = R^*_{left,e}(E)$.
- If $e \not\in PosF(t)$, and $Succ_c(p) = \{p_1, \ldots, p_n\}$ with $p_1 < \ldots < p_n$, $\forall i, j$ s.t. $p_i < p_j$, leftmost descendants at position $p_j$ can be computed after we have normalized those at position $p_i$. Then obviously, $R^*_{left}(E) = R^*_{left,p_1}(E) \cup \ldots \cup R^*_{left,p_{n-1}}((R^*_{left,p_1}(E) \cap IRR_1(R)) \ldots \cup \ldots \cup R^*_{left,p_n}((R^*_{left,p_1}(E) \cap IRR_1(R)) \ldots))$. □

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

**Example 5.8.** Let $R = \{f(x) \rightarrow s(x), h(x, y) \rightarrow c(f(x), g(y)), g(x) \rightarrow s(x)\}$ and $E = \{h(g(s^*(a)), f(s^*(a)))\}$. For clarity, we denote $s^*(a)$ by $*$;

- $R^*_{left,1}(E) = R^*_{left,e}(E) = R^*_{left,1}(E) \cup R^*_{*}(R^*_{left,1}(E) \cap IRR_1(R))$.

- $R^*_{left,1}(E) = R^*_{left,1}(E)$, $R^*_{1}(E)$ 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:

$L_1 \cup L_2 \cup \{c(f(g(s(*))), g(f(*))))\} \cap \{c(s(s(s(*)), g(f(*)))\} \cup \{c(s(s(s(*)), s(s(*)))\} \cup \{c(f(s(*)), g(s(*)))\} \cup \{c(s(s(*)), g(s(*)))\} \cup \{c(f(s(*)), g(f(*)))\}$

5.2. Recognizing $R^*_{p^s}(E)$

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.

![Diagram showing links between lemmas and theorems]

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 \( t \xrightarrow{\{q,l \rightarrow r\}} t' \) be a rewrite step, and let \( v' \in \text{Pos}(t') \).
\( v \in \text{Pos}(t) \) is an antecedent of \( v' \) in \( t \) (denoted by \( \text{ant}(v', t) \)) through this step if:

1. \( v' = q \) and \( v = v' \), or
2. \( \exists p' \in \text{PosVar}(r) \) with \( r|_{p'} = x \) s.t. \( v' = q \cdot p'.w \) and \( v = q \cdot p''.w \) where \( p'' \) is a position of \( x \) in \( l \).

**Remark.** Since lhs’s are linear, the antecedent (if it exists) is unique.

**Definition 5.10.** Let us consider the following derivation:

\[
I_0 \Rightarrow [p_0,l_0 \rightarrow r_0] \quad I_1 \Rightarrow \ldots \quad I_n \Rightarrow [p_n,l_n \rightarrow r_n] \quad I_{n+1}
\]

\( v_0 \in \text{Pos}(I_0) \) is an antecedent of \( v_{n+1} \in \text{Pos}(I_{n+1}) \) through this derivation if \( \exists v_1 \in \text{pos}(t_1), \ldots, v_n \in \text{Pos}(t_n) \) s.t. \( \forall i \in \{0, \ldots, n\}, \ v_i \) is an antecedent of \( v_{i+1} \) through step \( I_i \Rightarrow I_{i+1} \).

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

\[
s \Rightarrow [p,g \rightarrow d] \quad t \Rightarrow [q,l \rightarrow r] \quad u
\]
is a derivation in two steps s.t. \( q \) admits an antecedent in \( s \) denoted by \( p_0 \).

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 the derivation is preserved by commutation (Réty, 1988). Moreover, in (2), \( p \) does not have an antecedent in \( s \).

**Lemma 5.12.** Let \( R \) be a linear TRS. If

\[
t_0 \rightarrow_{[p_0, l \rightarrow r]} t_1 \rightarrow \ldots t_{n-1} \rightarrow_{[p_{n-1}, l_{n-1} \rightarrow r_{n-1}]} t_n \rightarrow_{[p_n, l_n \rightarrow r_n]} t_{n+1}
\]

is a derivation s.t. \( p_n \) admits an antecedent \( q_n \) in \( t_0 \), then, (1) can be commuted into

\[
t_0 \rightarrow_{[q_{n-1}, l_n \rightarrow r_n]} t'_1 \rightarrow \ldots t'_{n-1} \rightarrow_{[p_{n-1}, l_{n-1} \rightarrow r_{n-1}]} t'_n \rightarrow_{[p_n, l_n \rightarrow r_n]} t'_{n+1}
\]

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:

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

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.

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

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:

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

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:

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

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:

\[
t_0 \rightarrow_{[p_0, l_0 \rightarrow r_0]} t_1 \rightarrow \ldots t_n
\]
followed by the non-left-basic step

\[ I_n \rightarrow_{[p_n, I_n \rightarrow r_n]} I_{n+1} \]

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:

\[
I_0 \rightarrow_{[p_0, I_0 \rightarrow r_0]} I_1 \rightarrow \cdots I_j \rightarrow_{[q_j, I_j \rightarrow r_j]} I_{j+1} \rightarrow \cdots \\
\cdots \rightarrow I_n \rightarrow_{[p_n-1, I_{n-1} \rightarrow r_{n-1}]} I_{n+1} \tag{2}
\]

is left-basic.

**Theorem 5.17.** Let \( R \) be a linear TRS. If \( I_0 \rightarrow^* I_n \) (1), then \( I_0 \rightarrow^* I_n \) (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:

\[
I_0 \rightarrow_{[p_0, I_0 \rightarrow r_0]} I_1 \rightarrow \cdots I_n \rightarrow_{[p_n, I_n \rightarrow r_n]} I_{n+1} \tag{1}
\]

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

**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:

\[
I_0 \rightarrow^* I_{n+1} \quad \text{s.t.} \quad I_{n+1} = I_0 \downarrow \quad (1)
\]

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 a rewrite strategy.

**Proof.** Obviously, it comes from Theorems 5.17 and 5.19. □

5.2.2. The automaton

To build an automaton that recognizes \( R_P^* \) (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 \( \Delta_{sat}^* \) be the set of states and transitions obtained as in Definition B.3 by replacing each state \( d_{i,p}^{l,h} \in D \) by \( d_{i,p}^{l,h} \).

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

**Definition 5.22** (Saturation). Let \( \Delta_{sat}^* \) be the set of transitions added in the following way: whenever there are \( l_i \rightarrow r_i \in R \), a \((Q \cup Q_{arg})\)-substitution \( \sigma \) s.t. \( \text{dom}(\sigma) = \text{Var}(l_i) \cup \text{Var}(r_i) \).
Var(r), and \( l_i \sigma \rightarrow^{x} \Delta_{\text{arg}} \cup \Delta_{\text{sat}} \) \( q' \) where \( q' \in \{ q \} \cup D^{\text{sat}} \), then add the transition \( d_{\text{sat}, \sigma |_{l_i(p)}}^{i, \epsilon} \rightarrow q' \).

**Notation.** \( \Delta_{d}^{\text{sat}} = \Delta_{d}^{\text{sat}} \cup \Delta_{d}^{\text{sat}} \).

**Remark.** Let us note \( B' = (C \cup F, Q \cup Q_{\text{arg}} \cup D^{\text{sat}}, \{ q \}, \Delta \cup \Delta_{\text{arg}} \cup \Delta_{d}^{\text{sat}}) \); i.e. the same as \( R_{d}^{\times}(L(A | p)) \) except that the rewrite steps are not necessarily leftmost (see Réty (1999) for more details and explanations; here only \( \text{sat} \) differs).

We create another rhs’s encoding. 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 \( A_{\text{irr}} \) (and \( Q_{\text{irr}} \)) is given in the proof of Theorem 2.14. We consider the set of states

\[
D_{\text{spec}} = D \cup D^{\text{sat}} \cup D^{\text{sat}} \times Q_{\text{irr}}
\]

and the following set of transitions where \( (d_{\text{...}}, q_{\text{irr}}) \) denotes the pair \( (d_{\text{...}}, q_{\text{irr}}) \):

\[
\Delta_{\text{spec}} = \bigcup_{l_i \rightarrow r_i \in R} \bigcup_{p \in \text{Pos}(r_i) \cup \text{Arg}(r_i)} \bigcup_{k \in \{ 1, \ldots, \text{ar}(r_i(p)) \}} \{ r_i(p)(X_1, \ldots, X_n) \rightarrow d_{\text{sat}, \sigma |_{l_i \rightarrow r_i}}^{i, p, j} | \forall j, \sigma_j \text{ is any Q'-substitution s.t. } \text{dom}(\sigma_j) = \text{Var}(r_i | p, j), \}
\]

\[
\begin{align*}
&x \sigma_j, q_{\text{irr}} \text{ if } j < k \\
&x \sigma_j \text{ otherwise}
\end{align*}
\]

where \( \forall j, X_j = \{ d_{\text{sat}, \sigma |_{l_i \rightarrow r_i}}^{i, p, j} \text{ if } j = k \\
\quad d_{\text{sat}, \sigma |_{l_i \rightarrow r_i}}^{i, p, j} \text{ if } j < k \\
\quad d_{\text{sat}, \sigma |_{l_i \rightarrow r_i}}^{i, p, j} \text{ otherwise}
\}
\]

\[
\bigcup \{ r_i(p)(X_1, \ldots, X_n) \rightarrow d_{\text{sat}, \sigma |_{l_i \rightarrow r_i}}^{i, p, j} | l_i \rightarrow r_i \in R, p \in \text{Pos}(r_i), \sigma \text{ is any Q'-substitution s.t. } \text{dom}(\sigma) = \text{Var}(r_i | p) \}
\]

where \( \forall j, X_j = \{ x | r_i(p, j) \text{ is any variable } x \\
\quad q^{i, p, j} \in \text{Qarg} \text{ otherwise}
\}
\]

\[
\bigcup \{ x \sigma \rightarrow d_{\text{sat}, \sigma |_{l_i \rightarrow r_i}}^{i, \epsilon} | l_i \rightarrow r_i \in R, r_i \text{ is any variable } x, \sigma \text{ is any Q'-substitution s.t. } \text{dom}(\sigma) = \{ x \} \}
\]

Thus, \( r_i \sigma \) is also recognized into the state \( d_{\text{sat}, \sigma}^{i, \epsilon} \).
Now, we define an automaton that recognizes $R_p^{α}(L(A))$.

**Definition 5.24.** We define $B^α$, an automaton s.t.

$$B^α = (C ∪ F, Q^α, Q^γ, Δ^α)$$

where $Q^α = Q ∪ D_{spec} ∪ Q_{arg} ∪ (Q ∪ Q_{arg}) × Q_{irr}$ and $Q_f = \{q\}$ and

$Δ^α = Δ_q ∪ Δ_{spec} ∩ Δ_{irr} ∪ Δ_{spec} ∪ Δ_{sat} ∪ Δ_{arg}$.

$Δ_{irr}$ is obtained by running the automaton intersection algorithm on transition sets $Δ_{d}^{irr}$ and $Δ_{d}^{spec}$. Thus it encodes normal-forms of instances of rhs’s.

**Lemma 5.25.** $L(B^α) = R_p^{α}(L(A)|_f)$.

**Proof.** The proof comes from Corollary 5.20 and Réty (1999).

**Corollary 5.26.** $L(A[B^α]|_f) = R_p^{α}(L(A))$.

**Example 5.27.** Let $R = \{f(x) \rightarrow \{s(x), h(x, y) \rightarrow c(f(x), g(y)), g(x) \rightarrow s(x)\}\},$ and $t = h(x, y)$. We consider only instances of $t$ by constructors $a, s$, i.e. $E = \{h(s^a(a), s^a(a))\}$.

We are looking for an automaton that recognized $R_p^{α}(E)$. For the sake of simplicity we denote by $σ$ any substitution.

The only leftmost derivation is $E \rightarrow_{[1, \alpha]} c(f(s^a(a)), g(s^a(a))) \rightarrow_{[1, \alpha]} c(s(s^a(a)), s(s^a(a)))$. In the following, we give only sets of transitions.

$Δ_0 = \{a \rightarrow q_{data}, s(q_{data}) \rightarrow q_{data}, h(q_{data}, q_{data}) \rightarrow q^e\}$ where the final state is $q^e$.

Let us define $B^α$.

$Δ^α = \{c(d^{2,1}_q, d^{2,2}_q) \rightarrow d^{2,2}_q, f(q_{data}) \rightarrow d^{2,1}_q, g(q_{data}) \rightarrow d^{2,2}_q, s(q_{data}) \rightarrow d^{3,1}_q, s(q_{data}) \rightarrow d^{3,2}_q\}$.

$Δ_{spec} = \{c(d^{2,1}_q, d^{2,2}_q) \rightarrow d^{2,2}_q, c(d^{2,1}_q, d^{2,2}_q) \rightarrow d^{2,2}_q, f(q_{data}) \rightarrow d^{2,1}_q, g(q_{data}) \rightarrow d^{2,2}_q, s(q_{data}) \rightarrow d^{3,1}_q, s(q_{data}) \rightarrow d^{3,2}_q\}$.

$Δ^α_{d} = \{c(d^{2,1}_q, q_{data}) \rightarrow d^{2,2}_q, f(q_{data}) \rightarrow d^{2,1}_q, g(q_{data}) \rightarrow d^{2,2}_q, s(q_{data}) \rightarrow d^{3,1}_q, q_{data} \rightarrow d^{3,2}_q\}$.

$Δ^α_{sat} = \{d^{3,1}_q \rightarrow \alpha, d^{3,1}_q \rightarrow \alpha, d^{3,1}_q \rightarrow \alpha, d^{3,1}_q \rightarrow \alpha\}$.

For the following set, we give only what we will use: $Δ^α_{d} = \{s(q_{data}) \rightarrow d^{3,1}_q, q_{data} \rightarrow d^{3,1}_q, q_{data} \rightarrow d^{3,1}_q, q_{data} \rightarrow d^{3,1}_q\}$.

Leftmost descendants are indeed recognized. In particular, the non-leftmost descendants $c(f(s^a(a)), s(s^a(a)))$ are not recognized because $s(s^a(a))$ is recognized into the state $d^{2,2}_q$. This state appears in a lhs only in transition $c(d^{2,1}_q, q_{data}) \rightarrow d^{2,2}_q, \alpha$ of $Δ_{spec}$ and in a transition of $Δ^α_{d}$ (but transitions of $Δ^α_{sat}$ do not belong to the final set of transitions; see the previous definition). And using $c(d^{2,1}_q, q_{data}) \rightarrow d^{2,2}_q$ requires that the first argument is normalized, which does not hold for $f(s^a(a))$. 
6. Innermost-leftmost descendants: $R^*_\text{left}(E)$

In this section, we will use $R^*_p$ 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 $e\text{at}$. See Section 5 for details.

**Definition 6.1.** Given a language $L$ and a position $p$, let us define

$$R^*_\text{left,p}(L) = \{ t' \mid \exists t \in L, \ t \rightarrow^*_{[u_1, \ldots, u_n]} t' \}$$

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 1–3 and 6, and $E$ be the data-instances of a given linear term $t$.

Let $p \in \text{Pos}(t)$ and $L$ be a language s.t. $L|_p = E|_p$, and that is recognized by an automaton $A$ that discriminates every position $p' \in \text{Pos}(t) \mid p' \geq p$. Then,

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

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

$$R^*_\text{left,p}(L) = R^*_p \left[ R^*_{\text{left,p}_1} \left( \ldots \left( R^*_p \left( \ldots \left( R^*_{\text{left,p}_n} \left( L \cap \text{IRR}_p(R) \right) \right) \right) \right) \cap \text{IRR}_p(R) \right) \right]$$

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

**Proof.** By noetherian induction on $(\text{Pos}(t), \succ)$.

- If $\text{Succ}_t(p) = \emptyset$, then $\forall s \in L, \forall p' \in \text{Pos}(s), \ (p' \succ p \implies s(p') \in \mathcal{C})$. And since $p$ is a leftmost position and rhs's have no nested defined-functions, $R^*_p(L) = R^*_\text{left,p}(L)$.

We get $A'$ by Theorem 2.18.

- Let $\text{Succ}_t(p) = \{ p_1, \ldots, p_n \}$, s.t. $p_1 < \ldots < p_n$.

Let $s \in L$. Either no rewrite step is applied at position $p$, or 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 $p_n - 1 \prec p_n$, we first calculate those at position $p_n - 1$ and normalize, \ldots, to compute innermost-leftmost descendants at position $p_2$; since $p_1 \prec p_2$, we first calculate those at position $p_1$ and normalize. So, we search

$$B = R^*_\text{left,p}_n \left( \ldots \left( R^*_p \left( \ldots \left( R^*_p \left( L \cap \text{IRR}_p(R) \right) \right) \right) \right) \right)$$

$\forall p'$ positions s.t. $p'$ occurs strictly on the left of $p_n$; $\forall s' \in B$, $s'$ is normalized in $p'$. Then, obviously,

$$R^*_\text{left,p}(L) = R^*_p \left[ R^*_p \left( \ldots \left( R^*_p \left( \ldots \left( R^*_p \left( L \cap \text{IRR}_p(R) \right) \right) \right) \right) \right]$$

$\cap \text{IRR}_p(R) \cup R^*_{\text{left,p}_1} \cup \ldots \cup R^*_p \cap \text{IRR}_p(R) \ldots).$

$L$ is recognized by an automaton $A$ that discriminates every $p' \in \text{Pos}(t)$ 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 the induction hypothesis, $R^*_{\text{left},p_i}(L)$ is recognized by an automaton $A_1$ that still discriminates $p$ and every position $p''$ s.t. $p'' \neq p_1$ 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 $p' \in \text{PosF}(t)$ s.t. $p' \neq p_i$ ($p' \neq p_1$ for $IRR_{p_1}(R)$, …). So, by Lemma 2.8, $R^*_{\text{left},p_1}(L) \cap IRR_{p_1}(R)$ is recognized by an automaton that discriminates every position $p' \in \text{PosF}(t)$ s.t. $p' \neq p_1, \ldots, R^*_{\text{left},p_n}(\ldots)$ is recognized by an automaton $A_n$ that still discriminates $p$ and every position $p'^* \neq p_n$ (i.e. positions that are discriminated before computing $R^*_{\text{left},p_n}(\ldots)$ except those below $p_n$). By Lemma 2.8, $R^*_{\text{left},p_n}(L) \cap IRR_{p_n}(R)$ is recognized by an automaton that discriminates every position $p' \in \text{PosF}(t)$ s.t. $p' \neq p_n$ and in particular $p' \neq p$ and by Theorem 2.18, and so is $R^*_{E}([R^*_{\text{left},p_n}(\ldots) \cap R^*_{\text{left},p_1}(L) \cap IRR_{p_1}(R) \ldots) \cap IRR_{p_n}(R)]$. Finally, by union, we effectively obtain an automaton that still discriminates position $p' \neq p$. ☐

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

$$R^*_{\text{left}}(E) = \begin{cases} R^*_{\text{left},e}(E) & \text{if } e \in \text{PosF}(t) \\ R^*_{\text{left},p_1}(E) \cup \ldots \cup R^*_{\text{left},p_n}(\ldots (R^*_{\text{left},p_1}(E) \cap IRR_{p_1}) \ldots) & \text{otherwise} \\ \text{with } \text{Succ}_t(e) = \{p_1, \ldots, p_n\} \text{ s.t. } p_1 < \ldots < p_n \end{cases}$$

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

**Proof.** We have two cases:

- If $e \in \text{PosF}(t)$, obviously $R^*_{\text{left}}(E) = R^*_{\text{left},e}(E)$.
- If $e \notin \text{PosF}(t)$, then all innermost-leftmost descendants at position $p_j$ can be computed, so by normalization, $R^*_{\text{left}}(E) = R^*_{\text{left},p_1}(E) \cup \ldots \cup R^*_{\text{left},p_n}(\ldots (R^*_{\text{left},p_1}(E) \cap IRR_{p_1}) \ldots)$.

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

$*$ will symbolize the data-terms that instantiate $t$.

$t(e) \in \mathcal{F}$, so we calculate $R^*_{\text{left}}(E)$ where $E = \{ f(g(*), h(*)) \};$

$$R^*_{\text{left},e}(E) = R^*_{\text{left},2}(R^*_{\text{left},1}(E) \cap IRR_{1}(R) \cap IRR_{2}(R)) \cup R^*_{\text{left},1}(E) \cup R^*_{\text{left},2}(R^*_{\text{left},1}(E) \cap IRR_{1}(R)).$$

We have to compute $R^*_{\text{left},1}(E)$. So, $R^*_{\text{left},1}(E) = R^*_1(E) = E \cup f(*, h(*))$.

Now, we can compute $R^*_{\text{left},2}(R^*_{\text{left},1}(E) \cap IRR_{1}(R))$.

$$R^*_{\text{left},2}(f(*, h(*)) = f(*, s(*)) \cup f(*, h(*))$$
So, $R^*_e(f(*, s(*))) = s^*(f(*, s(*)))$

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

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 for getting 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 $R^*_\text{left}$ 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 $A_{\text{red}}$ as follows:

$$A_{\text{red}} = (C \cup F, Q_{\text{red}}, Q_{\text{redf}}, \Delta_{\text{red}})$$

where

$$Q_{\text{red}} = \{q_{\text{any}}, q_{\text{rec}}\} \cup_{w \in \text{Pos}(l_i)} \{q_{w} \}$$

$Q_{\text{redf}} = \{q_{\text{rec}}\}$ and

$$\Delta_{\text{red}} = \{l_i(p')(S_1, \ldots, S_n) \rightarrow q_{\text{rec}}^{i,p'}, l_i \rightarrow r_i \in R, p' \in \text{pos}(l_i) \}$$

$$S_j = \begin{cases} q_{\text{rec}}^{i,p',j} & \text{if } l_i(p', j) \neq \text{variable} \\ q_{\text{any}} & \text{otherwise} \end{cases}$$

$A_{\text{red}}$ does indeed recognizes $RED(R)$, because $t|_e$ reducible, i.e. $\exists$ $u$ position s.t. $u \geq \epsilon$ and $t \rightarrow_{[u,l \rightarrow r]} t'$.

– $q_{\text{any}}$ recognizes any terms (subterms of $t$ at positions incomparable with $u$, as well as instances of variables of $l$).

– $q_{\text{rec}}^{i,p',j}$ recognizes $l\sigma$ (subterms of $t$ at position $u$).

– $q_{\text{rec}}$ recognizes $C[l\sigma]$ (subterms of $t$ at positions $v$ s.t. $\epsilon \leq v \leq u$).
Example A.1. Let $\mathcal{F} \cup C = \{ f^1, g^1 \}$ and

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

The automaton that recognizes $\text{RED}(R)$ contains

$$f(q_1^1) \rightarrow q_1^1, s(q_{\text{any}}) \rightarrow q_1^{1, e}, g(q_{\text{any}}, q_{\text{any}}) \rightarrow q_1^{2, e}, q_1^{1, e} \rightarrow q_{\text{rec}},$$

$$q_1^{2, e} \rightarrow q_{\text{rec}}, f(q_{\text{any}}) \rightarrow q_{\text{any}}, s(q_{\text{any}}) \rightarrow q_{\text{any}}, g(q_{\text{any}}, q_{\text{any}}) \rightarrow q_{\text{any}},$$

$$f(q_{\text{rec}}) \rightarrow q_{\text{rec}}, s(q_{\text{rec}}) \rightarrow q_{\text{rec}}, g(q_{\text{rec}}, q_{\text{any}}) \rightarrow q_{\text{rec}}, g(q_{\text{rec}}, q_{\text{any}}) \rightarrow q_{\text{rec}}.$$

For example, we have $g(s(a), f(s(a)))$ and $f(g(s(a), s(a))) \in L(A_{\text{red}})$.

Appendix B. Recognizing $R^*_p(E)$

It may occur that the matches used in rewrite steps instantiate the variables by means of languages not recognized into states of $A_E$. That is, the instances are not always (sub)terms of $E$. Let us look at 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:

$$R = \{ g(x) \rightarrow h(x, b), h(x, y) \rightarrow c(x, y) \}.$$  

$A_{\theta}$ can be summarized by writing $\delta \rightarrow (a)$ (which means that $g(a) \rightarrow \Delta_{\theta} g(q^1) \rightarrow \Delta_{\theta} q^e$). The rewrite steps issued from $E$ are $g(a) \rightarrow_{[r_1.x/a]} h(a, b) \rightarrow_{[r_2.x/a]} c(a, b)$. Unfortunately, $Q_{\theta} = \{ q^e, q^1 \}$ and the language recognized into $q^e$ (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 $r_1$. 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 $Q_{\text{arg}}$ and the set of transitions $\Delta_{\text{arg}}$ as defined below:

$$Q_{\text{arg}} = \{ q^{i,p} | \ i \rightarrow r_i \in R, \ p \in \text{Arg}(r_i) \}$$

$$\Delta_{\text{arg}} = \{ r_i(p)q^{i,p,1}, \ldots, r_i(p)q^{i,p,n} \rightarrow q^{i,p} | \ q^{i,p} \in Q_{\text{arg}} \}$$

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

$$\text{Arg}(r_i) = \{ p \in \text{Pos}(r_i) | \ \exists p_{fct} \in \text{PosF}(r_i) \ p > p_{fct} \}.$$  

Now, we define how to encode a version of each instantiated rhs.

Let $A = (C \cup F, Q, Q_f, \Delta)$ be an automaton that discriminates the position $p$ into the state $q$, s.t. $Q \cap Q_{\text{arg}} = \emptyset$. Let $Q' = Q \cup Q_{\text{arg}}$. We use states of the form $d_\sigma^p$ where $\sigma$ 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

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

and the set of transitions

$$\Delta_d = \{r_i(p)(X_1, \ldots, X_n) \rightarrow d^{i,p}_{\sigma_1|\ldots|\sigma_n} \mid l_i \rightarrow r_i \in R, \ p \in Pos(r_i) \setminus Arg(r_i), \ r_i(p) \in C$$

$$\forall j, \ \sigma_j \text{ is any } Q'\text{-substitution s.t. } dom(\sigma_j) = Var(r_i|_{p,j}),$$

where $\forall j, \ X_j = \{x \in \sigma_j \mid r_i(p,j) \text{ is any variable } x
\}
d^{i,p,j}_{\sigma_j}$ otherwise

$$\cup \{r_i(p)(X_1, \ldots, X_n) \rightarrow d^{i,p}_{\sigma} \mid l_i \rightarrow r_i \in R, \ p \in PosF(r_i), \ \sigma \text{ is any } Q'\text{-substitution s.t. } dom(\sigma) = Var(r_i|_p)$$

where $\forall j, \ X_j = \{x \in \sigma_j \mid r_i(p,j) \text{ is any variable } x
\}
d^{i,p,j}_{\sigma_j}$ otherwise

$$\cup \{x \sigma \rightarrow d^{i,e}_{\sigma} \mid l_i \rightarrow r_i \in R, \ r_i \text{ is any variable } x,$$

$$\sigma \text{ is any } Q'\text{-substitution s.t. } dom(\sigma) = \{x\}\}$$

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

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

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

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

Now, we can define the saturation process.

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

Lemma B.6. $L(B) = R^*_p(L(A|_p))$

For the proof, see Réty (1999).

Corollary B.7. $L(A[B]_p) = R^*_p(L(A)).$
Remark. From Lemma 2.7, if \( A \) discriminates \( p' \not\equiv p \) into \( q' \), then \( A[B]_p \) also discriminates \( p' \) into \( q' \).

Appendix C. Proofs on rewriting commutation and left-basic derivations

C.1. Proof of Lemma 5.11

The fact that \( p \) has no antecedent is obvious because of Definition 5.9. Either \( p \) occurs on the right of \( p_0 \), or \( p \) occurs on top of \( p_0 \).

Now, let us show that (1) leftmost \( \Rightarrow \) (2) leftmost.

Let (1) leftmost. Let us classify \( \text{Var}(r) \) from left to right. If \( |\text{Var}(r)| = n \) then \( \text{Var}(r) = \{x_1, \ldots, x_n\} \). Since (1) is leftmost, let us denote by \( x_j = x \) 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 a function.

Let us suppose that \( p_0 \) is 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 \( \text{Var}(l) \) from left to right, we obtain the same order as \( \text{Var}(r) \). Hence, (2) is leftmost.

C.2. Proof of Lemma 5.12

This proof follows from Lemma 5.11.

Let \( \text{ant}(p_n, t_{n-1}) = q_0 \Rightarrow t_{n-1} \rightarrow_{[p_{n-1}, l_{n-1} \rightarrow r_{n-1}]} t_n \rightarrow_{[p_n, l_n \rightarrow r_n]} t_{n+1} \) commute itself into \( t_{n-1} \rightarrow_{[q_0, l_0 \rightarrow r_0]} t'_n \rightarrow_{[p_{n-1}, l_{n-1} \rightarrow r_{n-1}]} t_{n+1} \). And \( p_{n-1} \) has no antecedent in \( t_{n-1} \), and it is leftmost.

Let \( \text{ant}(q_0, t_{n-2}) = q_1 \Rightarrow t_{n-2} \rightarrow_{[p_{n-2}, l_{n-2} \rightarrow r_{n-2}]} t_{n-1} \rightarrow_{[q_0, l_0 \rightarrow r_0]} t'_n \) commute itself into \( t_{n-2} \rightarrow_{[q_1, l_1 \rightarrow r_1]} t'_n \rightarrow_{[p_{n-2}, l_{n-2} \rightarrow r_{n-2}]} t_{n+1} \). And \( p_{n-2} \) has no antecedent in \( t_{n-2} \), and it is leftmost.

\[ \vdots \]

By induction, let \( \text{ant}(q_{n-2}, t_0) = q_{n-1} \Rightarrow t_0 \rightarrow_{[p_0, l_0 \rightarrow r_0]} t_1 \rightarrow_{[q_{n-2}, l_n \rightarrow r_n]} t'_2 \) commute 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 no antecedent 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 \( \exists i \in \{1 \ldots n\} \) s.t. \( p_i \) has an antecedent in \( t_{i-1} \). By Definition 5.13, \( p_i \in B(t_i) \) where \( \forall j, B(t_i) = \{p' \mid p' \leq p_{i-1} \} \cup \{p_{i-1}, v \mid v \in \text{PosF}(r_{i-1})\} \cup \{p' \mid p_{i-1} < p'\} \). Let \( \text{ant}(p_i, t_{i-1}) = q \).

By Definition 5.9,

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

(2) \( (A) \) \( p_i < p_{i-1} \) and \( p_i = q \), or
We use Lemma 5.16, any number of times as necessary, and so according to Lemma 5.11.

By induction, let
\[ C.6. \text{Proof of Lemma } 5.18 \]

C.4. Proof of Lemma 5.16

Let \( \text{ant}(p_n, t_{n-1}) = q_0 \). Derivation can be commuted and we obtain
\[ t_0 \rightarrow t_1 \rightarrow \ldots t_j \rightarrow t_{n-1} \rightarrow_{[q_0]} t'_n \rightarrow_{[p_{n-1}]} t_{n+1} \]
where \( p_{n-1} \) has no antecedent in \( t_{n-1} \) according to Lemma 5.11.

By induction, let \( j \leq n \) be the smaller integer s.t. \( p_n \) admits an antecedent in \( t_j \). Let us denote this antecedent by \( q \). By Lemma 5.12, we can commute and we obtain
\[ t_0 \rightarrow \ldots t_j \rightarrow_{[q, t_n \rightarrow r_n]} t'_j \rightarrow_{[p_j]} \ldots \rightarrow t'_n \rightarrow_{[p_{n-1}]} t_{n+1} \]
and \( p_j \) has no antecedent in \( t_j \).

Hence, it is left-basic.

C.5. Proof of Lemma 5.17

Let \( i \in \{1 \ldots n\} \) s.t. \( t_0 \rightarrow \ldots t_i \) be a left-basic derivation and \( t_0 \rightarrow \ldots t_i \rightarrow t_{i+1} \) be a non-left-basic.

By running Lemma 5.16, \( '1' \) can be commuted in \( t_0 \rightarrow \ldots t_j \rightarrow_{[q]} t'_j \rightarrow \ldots t'_i \rightarrow t_{i+1} \) with \( \text{ant}(p_i, t_j) = q \) and \( '2' \) is left-basic. And by Lemma 5.12, \( '2' \) is leftmost if \( '1' \) is leftmost.

We use Lemma 5.16, as many times as are necessary, and so \( t_0 \rightarrow^* t_n \) 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 \( t_0 \rightarrow_{[p_{o, l_0 \rightarrow r_0}]} t_1 \rightarrow \ldots t_n \rightarrow_{[p_{o, l_n \rightarrow r_n}]} t_{n+1} \) (1) be a left-basic derivation, and let \( j \in \{0, \ldots, n\} \).

Let \( p' \in \text{Pos}(t_j) - B(t_j) \) s.t. \( t_j(p') \in F \). Let us take \( k = j + 1 \). For absurdity, let us suppose that
\[ \neg (\exists \ p' \in \text{Pos}(t_k) - B(t_k) \ s.t. \ t_k | p'_j = t_j | p' - that is, \ \forall p'_j \in \text{Pos}(t_k) - B(t_k) \ s.t. \ t_k | p'_j \neq t_j | p') \]
Then, we have to remove \( t_j | p' \) during the rewriting step.

- If \( p' < p_j \) then this is impossible.
– If $\rho'$ below $p_j$, $\rho' \in \text{PosVar}(l_j)$, looking at the constructor-based system, the rewrite rule $l_j \rightarrow r_j$ should have to eliminate variables. It happens that we have a restriction removing this model of rules.

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

C.7. Proof of Theorem 5.19

For absurdity, let us suppose that $\exists j, \rho'$ s.t. $t_j \rightarrow [\rho'] u$ with $\rho' \prec p_j$. According to Lemma 5.18, $\exists k > j$ s.t. $p_k = \rho'$. Then, $\rho' \notin B(t_{j+1})$ and $t_{j+1} \in \mathcal{F}$. According to Lemma 5.18, $\exists \rho'_{n+1} \in \text{Pos}(t_{n+1})$ s.t. $t_{n+1}|\rho'_{n+1} = t_j|\rho'$. That contradicts the fact that $t_{n+1}$ must be normalized.

Appendix D. Transforming $R$ to satisfy restriction 6

Transform($R$)

```latex
\begin{align*}
\text{init} & : R' = \emptyset \\
\forall l \rightarrow r \in R, \text{ if } \text{Var}(l) = \text{Var}(r) \text{ then add } l \rightarrow r \text{ to } R' & \\
\text{else } r' := \text{Construct}_r(l \rightarrow r); & \\
\text{add } l \rightarrow r' \text{ to } R' & \\
\text{endif} \\
\text{let } C(l) \text{ be a stack that contains all constructor pos. of } l. & \\
\text{if } |C(l)| \neq \emptyset \text{ then Build}(l \rightarrow r, C(l), R') & \\
R := R' & \\
\end{align*}
```

Construct$_r$(l $\rightarrow$ r)

```latex
\begin{align*}
\text{EraseVar} & := \text{Var}(l) - \text{Var}(r) \text{ where } |\text{Var}(l)| = n \text{ and } |\text{Var}(r)| = m. & \\
r & := \text{eat}(r, x_{m+1}) & \\
\text{For } i := m + 2 \text{ to } n \text{ do } r := r[2 \leftarrow \text{eat}(r|2, x_i)] & \\
\end{align*}
```

Build(l $\rightarrow$ r, stack, R’)

```latex
\begin{align*}
p & := \text{head}(\text{stack}); \text{ stack } := \text{pop}(\text{stack}); & \\
\text{if not empty}(\text{stack}) \text{ then Build}(l \rightarrow r, \text{stack}, R') & \\
l' & := l[p \leftarrow \text{eat}(l|p, x_p)]; \text{ r' } := \text{Construct}_r(l' \rightarrow r); \text{ Add } l' \rightarrow r' \text{ to } R' & \\
\text{stack } := \text{Shift}(\text{stack}, p); & \\
\text{if not empty}(\text{stack}) \text{ then Build}(l' \rightarrow r, \text{stack}, R') & \\
\end{align*}
```

In the following function, Shift(stack, p) raises all positions which are under $p$ in the stack by a depth of 1; for example, if $p = 2$ and, 2.1, 2.2 and 1 are in the stack, after running this function, 2.1.1, 2.1.2 and 1 are in the stack:

```latex
\text{Shift}(\text{stack}, p)
\begin{align*}
\forall q \in \text{stack} \text{ s.t. } \exists w \text{ and } q = p.w \text{ replace } q \text{ by } p.1.w
\end{align*}
```
Example D.1. Let us take again the TRS of Example 5.1.

\[\text{Transform} \left(f(s(s(x)), y) \rightarrow x\right)\]

\[\text{init} : R' = \emptyset\]

\[r' = \text{Construct}_r(l \rightarrow r) = \text{eat}(x, y); \quad R' = \{f(s(s(x)), y) \rightarrow \text{eat}(x, y)\}\]

\[C(l) = \{1, 1.1\}\]

\[\text{Build}(l \rightarrow r, C(l), R')\]

\[p = 1; \text{stack} = \{1.1\}\]

\[\text{Build}(l \rightarrow r, \{1.1\}, R')\]

\[p = 1.1; \text{stack} = \emptyset\]

\[l' = {[1.1 \leftarrow \text{eat}(l_{[1,1], x_{1.1}})}]; \quad r' = \text{Construct}_r(l' \rightarrow r)\]

\[\text{add } f(s(\text{eat}(s(s(x), x_{1.1})), y) \rightarrow \text{eat}(x, \text{eat}(x_{1.1}, y))) \text{ to } R'.\]

\[l'' = {[1 \leftarrow \text{eat}(l_{[1], x_{1})}]; \quad r'' = \text{Construct}_r(l'' \rightarrow r)\]

\[\text{add } f(\text{eat}(s(s(x), x_{1}), y) \rightarrow \text{eat}(x, \text{eat}(x_{1}, y))) \text{ to } R'.\]

\[\text{stack} = \text{Shift}(\text{stack}, 1) = \{1.1.1\}\]

\[\text{Build}(l'' \rightarrow r, \{1.1.1\}, R')\]

\[p = 1.1.1; \text{stack} = \emptyset\]

\[l'' = {[1.1.1 \leftarrow \text{eat}(l''_{[1,1.1], x_{1.1.1})}); \quad r'' = \text{Construct}_r(l'' \rightarrow r)\]

\[\text{add } f(\text{eat}(s(\text{eat}(s(s(x), x_{1.1.1})), x_{1}), y) \rightarrow \text{eat}(x, \text{eat}(x_{1.1.1}, \text{eat}(x_{1}, y)))) \text{ to } R'.\]

\[R = R'\]

References


