UNIVERSITE D'ORLEANS
Faculté des Sciences
LIFO

Laboratoire d'Informatique Fondamentale d'Orléans
4, rue Léonard de Vinci, BP 6759
F-45067 Orléans Cedex 2
FRANCE

Rapport de Recherche

www : http://www.univ-orleans.fr/SCIENCES/LIFO/

Context-Free Tree languages for Descendants

Pierre Réty and Julie Vuotto
Université d'Orléans, LIFO

Rapport N^o **2004-04**

Context-Free Tree languages for Descendants

Pierre Réty Julie Vuotto

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

Abstract

The preservation of regular tree languages through rewriting, has already been studied. In this paper, we study the preservation of context-free tree languages through rewriting, for constructor-based term rewrite systems. We give positive and negative results. Positive results are effective since we give algorithms to build context-free grammars.

1 Introduction

The descendants of a set of terms E by a rewrite system R are the terms obtained by rewriting the elements of E with the rules of R. Computing descendants may be used for checking rewriting properties, like reachability, joinability,..., that may help to analyse rewrite programs. It may also help to check safety properties for cryptographic[1] protocols [5] : the set of descendants (denoted by $R^*(E)$) expresses all the possible messages running in the net, and let $\mathcal{L}_I$ be the set of illegal messages. Intuitively, R simulates the protocol steps and the intruder actions; and illegal messages simulate intrusions. The protocol is safe (in a certain sense) iff $R^*(E) \cap \mathcal{L}_I = \emptyset$. Protocol verification has also been investigated using other techniques : [1, 8, 7, 11].

In general, the set of descendants $R^*(E)$ is infinite. An easy way to express and handle such infinite sets is to use finite tree automata, i.e. regular tree languages [3, 10, 17, 16, 12]. However, it is undecidable whether a given rewrite system preserves regularity (also called recognizability) or not [6], and all those papers define subclasses. On the other hand, the possibility of computing a regular superset of the (possibly non-regular) set of descendants, assuming weaker restrictions, has been investigated in [4, 5]. Computing only descendants obtained by rewriting respecting some strategies has also been studied [14, 13].

In [12], certain restrictions are assumed in order to make $R^*(E)$ regular. R is assumed to be constructor-based, as well as :

1. Right-hand-sides of rewrite rules are linear.
2. No nested defined-functions in rhs's.
3. Innermost[2] function calls in rhs's are shallow subterms[3].

[1]For more information on cryptographic protocol, the reader can refer to [15].

[2]Innermost is useless when Restriction 2 is satisfied. Without "innermost", Restriction 3 would imply Restriction 2, and we want them to be independent.

[3]I.e. $\forall l \to r \in R$, $\forall p \in Pos(r)$, ($r(p)$ is an innermost defined-function in $r \implies r|_p = f(r_1, \ldots, r_n)$) where $r_1, \ldots, r_n$ are variables or ground constructor-terms.

4. E is a particular regular language: E is the set of constructor-instances of a fixed linear term t (or more generally t is instantiated by arbitrary regular languages of constructor-terms).

Moreover, some counter-examples show that if anyone among the above four restrictions is not satisfied, then $R^*(E)$ is not regular. Consequently, the following question arises: does $R^*(E)$ belong to the previous class in Chomsky's hierarchy, i.e. is it a context-free tree language? If it is, this can still be used for protocol verification because $R^*(E) \cap \mathcal{L}_I = \emptyset$ is decidable, provided $\mathcal{L}_I$ is still regular. This paper attempts to answer this question.

Before giving the results, some more notions need to be introduced. We say that a tree language is *NT-bounded* if it is generated by a NT-bounded grammar, which implies that the number of nested non-terminals in trees generated by the grammar is bounded. Note that regular languages are NT-bounded. A context-free tree language is said *top-context-free*[4] if it is generated by a top-context-free grammar, i.e. whose production right-hand-sides contain only one non-terminal, located on top. Note that top-context-free languages are context-free and NT-bounded.

We also define some other restrictions:

5. Left-hand-sides of rewrite rules are linear.
6. Recursive rewrite rules are not consuming, i.e. their left-hand-sides are of the form $f(x_1, \ldots, x_n)$, where $x_1, \ldots, x_n$ are variables.
7. E is the set of instances of a fixed linear term t, by arbitrary NT-bounded context-free languages of constructor-terms.
8. E is the set of instances of a fixed **non-linear** term t that contains only one defined-function occurring on top, by instantiating the linear variables of t by arbitrary NT-bounded context-free languages of constructor-terms, and by instantiating the non-linear variables of t by arbitrary top-context-free languages of constructor-terms.

Still assuming that R is constructor-based, the results are:

- For all $i \in \{1, 2, 3\}$, if restrictions 1, 2, 3, 4 are satisfied except Restriction i then $R^*(E)$ is not context-free in the general case.
- If Restrictions 1, 2, 5, 6 and (7 or 8) are satisfied, then $R^*(E)$ is context-free and NT-bounded.
- If Restrictions 1, 2, 6, 7 are assumed (i.e. Restriction 5 is not assumed anymore), and even if Restriction 3 is assumed in addition, then $R^*(E)$ is not context-free in the general case.
- If Restrictions 1, 2, 5 and (7 or 8) are assumed (i.e. Restriction 6 is not assumed anymore), and even if Restriction 3 is assumed in addition, then $R^*(E)$ is not context-free in the general case.

The positive results are obtained by building a context-free grammar that generates $R^*(E)$.

[4] Initially introduced by A. Arnold and M. Dauchet [2], and called *co-regular*. It is called *top-context-free* in [9].

2 Preliminaries

2.1 Terms and Positions

We denote respectively by $\mathcal{C}$, $\mathcal{F}$, $\mathcal{X}$ the sets of *constructors, defined-functions* and *variables*. For $s \in \mathcal{C} \cup \mathcal{F}$, $ar(s)$ denotes the arity of s. In the following, we write $f \in \mathcal{F}^{\backslash n}$ for $f \in \mathcal{F}$ and $ar(f) = n$. A term is *linear* if no variable occurs more than once. A *ground term* is a term that does not contain variables. $T(\Sigma, \mathcal{X})$ is the set of terms defined on the signature $\Sigma = \mathcal{F} \cup \mathcal{C}$, T_Σ is the set of ground terms, and $T_\mathcal{C}$ is the set of ground *constructor-terms* (terms that contain only constructors). Let t, t' be terms. We denote by $Var(t)$ the set of variables of t, by $Pos(t)$ the set of positions of t, by $PosF(t)$ the set of positions of defined-functions of t, and by $\overline{Pos}(t)$ the set of positions of non-variable symbols of t. A position p is a list of integers whose length is denoted by $|p|$. For positions p, p', $p \geq p'$ means that p is located below p', i.e. $p = p'.v$ for some position v, whereas $p \| p'$ means that p and p' are incomparable, i.e. $\neg(p \geq p') \wedge \neg(p' \geq p)$. Let $p \in \overline{Pos}(t)$, $t|_p$ is the subterm of t at position p, and $t(p)$ is the top symbol of $t|_p$. $t[t']_p = t[p \leftarrow t']$ is the term obtained from t by replacing the subterm at position p by t'. A term $C \in T(\Sigma, \mathcal{X})$ s.t. $Var(C) = \{x_1, \ldots, x_n\}$ is called a *context*, and $C[t_1, \ldots, t_n]$ denotes the term obtained from C by replacing each x_i by the term t_i.

2.2 Rewrite Systems and Descendants

A *term rewrite system (TRS)* is a pair (Σ, R) where R is a finite set of rewrite rules $l \to r$ where $l, r \in T(\Sigma, \mathcal{X})$ and $Var(r) \subseteq Var(l)$. In the following of the paper we write only R for a rewrite system. *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 $f \in \mathcal{F}$ and $t_1, \ldots, t_n$ contain only constructors and variables. The rewrite relation $\to_R$ is defined as follows: $t \to_R t'$ (or for short, $t \to t'$) if there exists $p \in \overline{Pos}(t)$, a rule $l \to r$ and a substitution σ s.t. $t|_p = l\sigma$ and $t' = t[r\sigma]_p$. $\to_R^*$ denotes the reflexive-transitive closure of $\to_R$. t' is a *descendant* of t if $t \to_R^* t'$. If E is a set of terms, $R^*(E)$ denotes the set of descendants of elements of E. t is *irreducible* if $\neg(\exists t' \mid t \to_R t')$. The set of irreducible ground terms is denoted by $IRR(R)$. t' is a *normal-form* of t if $t \to_R^* t'$ and t' is irreducible. The set of normal-forms of elements of E is denoted by $R^!(E)$. Thus, $R^!(E) = R^*(E) \cap IRR(R)$.

2.3 Recursive and Consuming Rewrite Rules

Definition 2.1 *Let R be a TRS. Let f, $g \in \mathcal{F}$. The relation $>$ on $\mathcal{F}$ is defined as follows:*

$$f > g \iff \exists l \to r \in R,\ l(\epsilon) = f \wedge \exists p \in PosF(r),\ r(p) = g$$

Definition 2.2 *let R be a TRS and $l \to r \in R$. $l \to r$ is* recursive *in R if*

$$\exists g \in \mathcal{F}\ (g \in F(r)\ \wedge\ g >^* l(\epsilon))$$

where $>^*$ is the reflexive-transitive closure of $>$, and $F(r)$ are the defined-functions appearing in r.

Lemma 2.3 *Let R be a TRS and $l \to r,\ l_1 \to r_1, \ldots,\ l_i \to r_i \in R$. Let t be a term s.t. $PosF(t) = \{\epsilon\}$.*

$$\textit{If } t \to_{l \to r} t_1 \to_{l_1 \to r_1} t_2 \ldots \to_{l_n \to r_n} t_{n+1} \textit{ and } l \to r \textit{ is not recursive, then}$$
$$\forall i,\ l_i \to r_i \neq l \to r \textit{ (i.e } l \to r \textit{ can be used only once).}$$

Proof. *$l \to r$ being non-recursive, by applying rules on functions of r, we can never generate the symbol $l(\epsilon)$. So, $\forall i,\ \forall p_i \in posF(t_i),\ t_i(p_i) \neq l(\epsilon)$.*

Definition 2.4 *Let R be a TRS and $l \to r \in R$.*
$l \to r$ is not consuming *if l is of the form $f(x_1, \ldots, x_n)$, where $x_1, \ldots, x_n$ are variables.*

2.4 Tree Grammars

A *tree grammar* $\mathcal{G}$ is a quadruple $(\mathcal{N}, \Sigma, S, P)$ where $\mathcal{N}$ is a finite set of non-terminals, Σ a signature, $S \in \mathcal{N}$ the axiom, and P a finite set of production rules. In the regular case, the production rules are of the form: $A \to t$ where $A \in \mathcal{N}$ and $t \in T(\Sigma \cup \mathcal{N})$ and in the context-free case: $A(x_1, \ldots, x_n) \to t$ where A is a n-ary non-terminal, $x_1, \ldots, x_n$ are distinct variables and $t \in T(\Sigma \cup \mathcal{N} \cup \{x_1, \ldots, x_n\})$[5]. Note that there are only 0-ary non-terminals in the regular case. The language $\mathcal{L}(\mathcal{G})$ generated by $\mathcal{G}$ is the set of terms derivable from S. A set of terms E is a regular (resp. context-free) language if there is a regular (resp. context-free) grammar $\mathcal{G}$ such that $E = \mathcal{L}(\mathcal{G})$. Grammar 2.5.a below generates the regular tree language $\{f(s^*(a))\}$ while Grammar 2.5.b generates the context-free tree language $\{f(s^n(a), s^n(b))\}$.

Example 2.5

a.
$$S \to f(B)$$
$$B \to s(B) \mid a$$

b.
$$S \to F(a, b)$$
$$F(x, y) \to F(s(x), s(y)) \mid f(x, y)$$

Figure 1 recalls known properties.

	L_1	L_2	
$L_1 \cup L_2$	reg.	reg.	reg.
	c-free	c-free	c-free
$L_1 \cap L_2$	reg.	reg.	reg.
	reg.	c-free	c-free
	c-free	c-free	~~c-free~~
$\overline{L_1}$	reg.		reg.
$t \in L_1$	reg. or c-free		decidable
$L_1 = \emptyset$	reg. or c-free		decidable

Figure 1: Properties

In a grammar, the notion of recursive production is defined in the same way as for rewrite rules.

Definition 2.6 *A context-free tree grammar is said* NT-bounded *if the transitions whose rhs's contain nested non-terminals, are not recursive.*

[5]Note that t is not necessarily linear.

Lemma 2.7 *Let $\mathcal{G}$ be a NT-bounded context-free tree grammar. The number of nested non-terminals in trees generated by $\mathcal{G}$ is bounded.*

Recall that top-context-free languages are defined in the introduction.

Lemma 2.8 *[2] Let $L = \{f(t,t) \mid t \in L'\}$. L is context-free iff L' is top-context-free. And in this case, L is also top-context-free.*

Lemma 2.9 *[9] Let Σ be a signature. T_Σ is top-context-free iff Σ contains only symbols of arity at most one or no constant symbol.*

3 Negative Results

The negative results come from the counter-examples below, and the following remark: if R is left-linear, the set of irreducible terms $IRR(R)$ is regular. Therefore, if $R^*(E)$ is context-free, then $R^!(E) = R^*(E) \cap IRR(R)$ should also be context-free. More generally, if $R^*(E)$ is context-free and T is regular, then $R^*(E) \cap T$ should also be context-free.

Counter-example 3.1 Restrictions 1 to 4 are satisfied, except Restriction 1.
Let $C = \{c^{\backslash 2},\ a^{\backslash 0}\}$ and $T_{\mathcal{C}}$ be the set of terms on $\mathcal{C}$.
Let $R = \{f(x) \to c(x,x)\}$ and $E = \{f(s) \mid s \in T_{\mathcal{C}}\}$. Then, $R^!(E) = \{c(s,s) \mid s \in T_{\mathcal{C}}\}$.
From Lemmas 2.8 and 2.9, we can conclude that $R^!(E)$ is not context-free.

Counter-example 3.2 Restrictions 1 to 4 are satisfied, except Restriction 3.

$$R = \{f(x) \to p(f(s(x))),\ f(x) \to g(x,a),\ g(s(x),y) \to s(g(x,s(y)))\},\ E = \{f(a)\}.\ \text{Then, } R^!(E) = \{p^n(s^n(g(a,s^n(a)))) \mid n \in \mathbb{N}\}.$$

It is not context-free because generating the right-hand branch $p^n(s^n(g(s^n(a))))$ needs to count n three times.

Remark : If Restrictions 1 to 4 are satisfied, except Restriction 2, we can simulate the previous counter-example by using the same starting language and the following TRS:

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

Counter-example 3.3 Restrictions 1, 2, 3, 6, 7 are assumed (Restriction 5 is not assumed). Let $R = \{ f(c(x,x)) \rightarrow x \}$ and $t = f(y)$. Let $L = \{ c(s^*(p^n(r^k(a))), s^n(p^k(r^*(a)))) \mid n, k \in \mathbb{N} \}$.

Let us consider $E = t[y \leftarrow L]$. The tree language L is recognized by the following NT-bounded context-free grammar:

$S \rightarrow F(a, a)$

$F(x, y) \rightarrow F(x, r(y)) \mid F_1(x, y) \qquad F_1(x, y) \rightarrow F_1(r(x), p(y)) \mid F_2(x, y)$

$F_2(x, y) \rightarrow F_2(p(x), s(y)) \mid F_3(x, y) \quad F_3(x, y) \rightarrow F_3(s(x), y) \mid c(x, y)$.

Let $T_{\mathcal{C}}$ be the set of terms on $\mathcal{C} = \{c, s, p, r, a\}$. Obviously, $T_{\mathcal{C}}$ is regular.
Hence, $R^*(E)$ is not context-free since $R^*(E) \cap T_C = \{s^n(p^n(r^n(a))) \mid n \in \mathbb{N}\}$ is not context-free.

Counter-example 3.4 Restrictions 1, 2, 3, 5 and 7 are assumed (Restriction 6 is not assumed). Let:

$$R = \{ i(c(x,y,z)) \xrightarrow{r_0} f(x,y,z),\ f(s(x),y,z) \xrightarrow{r_1} s(f(x,y,z)),\ f(a,y,z) \xrightarrow{r'_1} g(a,y,z),$$
$$g(x,p(y),z) \xrightarrow{r_2} p(g(x,y,z)),\ g(x,a,z) \xrightarrow{r'_2} h(x,a,z),\ h(x,y,r(z)) \xrightarrow{r_3} r(h(x,y,z)) \}$$

Let us consider $E = \{ i(c(s^n(a), p^n(a), r^n(a))) \mid n \in \mathbb{N} \}$. Obviously, $R^!(E) = \{ s^n(p^n(r^n(h(a,a,a)))) \mid n \in \mathbb{N} \}$ is not context-free. Hence, $R^*(E)$ is not context-free.

Remark : If Restrictions 1, 2, 3, 5 and 8 are assumed, we add the rule $j(x, y) \rightarrow i(x)$ to the previous counter-example and consider

$$E = \{ j(x,x)\ \theta \mid \theta : x \leftarrow \{ c(s^n(a), p^n(a), r^n(a)) \mid n \in \mathbb{N} \} \}$$

4 Positive Results

4.1 Definitions

Definition 4.1 *Let t be a fixed linear term, and let $Var(t) = \{x_1, \ldots, x_k\}$. Let $L_1, \ldots, L_k$ be context-free tree languages generated by the grammars $\mathcal{G}_{L_1}, \ldots, \mathcal{G}_{L_k}$.*

Let $p_1, \ldots, p_n \in \overline{Pos}(t)$ *s.t.* $\forall i, j \ (i \neq j \Rightarrow p_i || p_j)$. *Let* $L'_1, \ldots, L'_n$ *be context-free tree languages generated by the grammars* $\mathcal{G}_{L'_1}, \ \ldots, \ \mathcal{G}_{L'_n}$.

The language of instances of t*, whose sub-terms at positions* $p_1, \ldots, p_n$ *have been replaced, is defined as follows:*

$$E[p_1 \leftarrow L'_1] \ldots [p_n \leftarrow L'_n] = \{(t\sigma)[p_1 \leftarrow s_1] \ldots [p_n \leftarrow s_n] \,|\, \forall i, \ x_i\sigma \in L_i \ \wedge \ \forall j, \ s_j \in L'_j\}$$

By renaming, if necessary, the non-terminals, we can suppose that the set of non-terminals of $\mathcal{G}_{L_1}, \ldots, \mathcal{G}_{L_k}, \mathcal{G}_{L'_1}, \ \ldots, \ \mathcal{G}_{L'_n}$ *are disjoint.*
We denote by $S_{\mathcal{G}_{L_i}}$ *(resp.* $P_{\mathcal{G}_{L_i}}$*) the axiom (resp. the set of productions) of* $\mathcal{G}_{L_i}$.
The context-free tree grammar $\mathcal{G}_{E[p_1 \leftarrow L'_1] \ldots [p_n \leftarrow L'_n]}$ *that generates* $E[p_1 \leftarrow L'_1] \ldots [p_n \leftarrow L'_n]$*, has the axiom* $S_{\mathcal{G}_{E[p_1 \leftarrow L'_1] \ldots [p_n \leftarrow L'_n]}} = T_\epsilon$*, and the set of productions:*

$$\begin{aligned} P_{\mathcal{G}_{E[p_1 \leftarrow L'_1] \ldots [p_n \leftarrow L'_n]}} = \ & P_{\mathcal{G}_{L_1}} \cup \ldots \cup P_{\mathcal{G}_{L_k}} \cup P_{\mathcal{G}_{L'_1}} \cup \ldots \cup P_{\mathcal{G}_{L'_n}} \\ & \{T_q \rightarrow t(q)(T_{q.1}, \ldots, T_{q.ar(t(q))}) \,|\, q \in \overline{Pos}(t) \ \wedge \ \forall i, \ \neg(q \geq p_i)\} \\ & \{T_{p_i} \rightarrow S_{\mathcal{G}_{L'_i}} \,|\, i \in \{1, \ldots, n\}\} \\ & \{T_q \rightarrow S_{\mathcal{G}_{L_j}} \,|\, q \in Pos(t) \ \wedge \ t(q) = x_j \ \wedge \ \forall i, \neg(q \geq p_i)\} \end{aligned}$$

Remark :

- The particular case $n = 0$ defines the language E of the instances of t by elements of $L_1, \ldots, L_k$.
- If $\mathcal{G}_{L_1}, \ldots, \mathcal{G}_{L_k}, \mathcal{G}_{L'_1}, \ \ldots, \ \mathcal{G}_{L'_n}$ are NT-bounded, $\mathcal{G}_{E[p_1 \leftarrow L'_1] \ldots [p_n \leftarrow L'_n]}$ is also NT-bounded.

Definition 4.2 *Let* $p \in \overline{Pos}(t)$*, then we define* $E|_p = \{s|_p \,|\, s \in E\}$.
We obtain a grammar that generates $E|_p$ *by replacing the axiom of* $\mathcal{G}_E$ *by* T_p.

Now, we consider the descendants obtained by rewriting below or at position p ($R^*_{\geq p}$), and the descendants obtained by rewriting starting from position p (R^*_p).

Definition 4.3 *Let* L *be a language and* p *be a position. We define:*

$$R^*_{\geq p}(L) = \{s' \,|\, \exists s \in L, \ s \rightarrow^*_{[u_1, \ldots, \ u_n]} s' \ \wedge \ \forall i, \ u_i \geq p\}.$$

Definition 4.4 $t \rightarrow^+_{[p, rhs's]} t'$ *means that* t' *is obtained by rewriting* t *at position* p*, plus possibly at positions coming from the rhs's.*
Formally, there exist some intermediate terms $t_1, \ldots, t_n$ *and some sets of positions* $P(t), P(t_1), \ldots, P(t_n)$ *s.t.*

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

where

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

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

Definition 4.5 *Given a language L and a position p, we define $R_p^*(L)$ as follows*

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

Example 4.6 Let $R = \{f(x) \rightarrow s(x),\ g(x) \rightarrow s(h(x)),\ h(x) \rightarrow p(f(x))\}$ where $F = \{f,\ g,\ h\}$ and $C = \{s,\ a\}$. The symbols(s) that are eligible for rewriting, are underlined :

$$R_1^*(\{f(\underline{h}(g(a)))\}) = \{f(\underline{h}(g(a)))\} \cup \{f(p(\underline{f}(g(a))))\} \cup \{f(p(s(g(a))))\}$$

$Succ_t(p)$ are the nearest defined-function positions of t located below p. Formally :

Definition 4.7 *Let $p \in Pos(t)$.*

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

4.2 Computing $R^*(E)$

The following theorem and lemma give a recursive algorithm (using $R^*_{\geq p}$) to build a NT-bounded context-free grammar that generates $R^*(E)$, from NT-bounded context-free grammars that generate R^*_p.
The proofs are given in appendix Section A.

Theorem 4.8 *Let t be a linear term, and E be the language of instances of t.*

$$R^*(E) = \begin{cases} R^*_{\geq\epsilon}(E) & \textit{if } t(\epsilon) \in \mathcal{F} \\ E[p_1 \leftarrow R^*_{\geq p_1}(E)|_{p_1}] \ldots [p_n \leftarrow R^*_{\geq p_n}(E)|_{p_n}] & \textit{otherwise} \\ & \textit{with } Succ_t(\epsilon) = \{p_1, \ldots,\ p_n\}. \end{cases}$$

Lemma 4.9 *Let t be a linear term, and E be the language of instances of t. Let $p \in PosF(t)$.*

$$R^*_{\geq p}(E) = \begin{cases} R^*_p(E[p_1 \leftarrow R^*_{\geq p_1}(E)|_{p_1}] \ldots [p_n \leftarrow R^*_{\geq p_n}(E)|_{p_n}]) & \textit{if } Succ_t(p) = \{p_1, \ldots,\ p_n\} \\ R^*_p(E) & \textit{if } Succ_t(p) = \emptyset \end{cases}$$

The following lemma gives an algorithm to build a NT-bounded context-free grammar that generates R^*_p, from a NT-bounded context-free grammar that generates R^*_ϵ.

Lemma 4.10 *Let $L'_1, \ldots, L'_n$ be context-free tree languages.*

- *If $Succ_t(p) = \{p_1, \ldots,\ p_n\} \neq \emptyset$, then*
  $R^*_p(E[p_1 \leftarrow L'_1] \ldots [p_n \leftarrow L'_n]) = E[p \leftarrow R^*_\epsilon(E|_p[p_1 - p \leftarrow L'_1] \ldots [p_n - p \leftarrow L'_n])]$
- *If $Succ_t(p) = \emptyset$, then $R^*_p(E) = E[p \leftarrow R^*_\epsilon(E|_p)]$*

Remark : If t contains only one defined-function, occurring at the root, we have $R^*(E) = R^*_\epsilon(E)$. Moreover, we show in Section 4.3 that we can compute $R^*_\epsilon(E)$ provided E is a NT-bounded context-free language. Consequently, the second positive result (assuming t is not linear) comes from the following lemma.

Lemma 4.11 *Let t be a term s.t. :*

- *The linear variables of t are instantiated by NT-bounded context-free tree languages, and*
- *the non-linear ones are instantiated by top-context-free tree languages.*

Then E is a NT-bounded context-free tree language.

4.3 Computing R^*_ϵ

4.3.1 The Maximal Depth of consumable symbols : $Depth(R)$

Thanks to Restriction 6, in a term t s.t. $PosF(t) = \{\epsilon\}$, constructors occurring below a certain depth are not used when rewriting t in several steps : they are not consumable by rewriting.

Example 4.12 Let $R = \{\, h(s(x)) \overset{ru_1}{\to} x,\ g(p(x)) \overset{ru_2}{\to} h(x)\,\}$.

When rewriting $g(p(s(a)))$ by ru_2 and ru_1, p, s are consumed, whereas a is not consumed.

$Depth(R)$ as defined below is actually the maximal depth of consumable symbols (see the completeness proof in Appendix D).

Definition 4.13 *Let R be a TRS satisfying restriction 6, and $l \to r \in R$. We define $Depth(l) = Max(\{|p| \mid p \in \overline{pos}(l)\})$ and*

$$Depth(R) = \sum_{l \to r \in R} Depth(l)$$

Remark : Only consuming rules matter when computing $Depth(R)$, because if $l \to r$ is not consuming, then $Depth(l) = 0$.

4.3.2 Explanations and Example

Let R be a constructor-based TRS satisfying Restrictions 1, 2, 5, 6, and let E satisfying Restrictions 7 or 8 be the starting tree language, generated by the NT-bounded context-free grammar $\mathcal{G}_E = (\mathcal{N}_{\mathcal{G}_E}, \Sigma, S_{\mathcal{G}_E}, P_{\mathcal{G}_E})$. Let $p \in PosF(t)$ and $L = E|_p$. The algorithm for building a context-free grammar $\mathcal{G}_{R^*_\epsilon L} = (\mathcal{N}_{\mathcal{G}_{R^*_\epsilon L}}, \Sigma, S_{\mathcal{G}_{R^*_\epsilon L}}, P_{\mathcal{G}_{R^*_\epsilon L}})$ generating $R^*_\epsilon(L)$ starts with the productions of $\mathcal{G}_L$, and adds new productions with 0-ary non-terminals of the form A_t, where t is a tree that may contain non-terminals of $\mathcal{G}_L$, s.t. $t(\epsilon) \in \mathcal{F}$ and $\mathcal{L}(A_t) = R^*(\{t' \mid t \to^*_{\mathcal{G}_L} t'\})$. The algorithm also uses n-ary non-terminals of the form A_t, and in this case t contains in addition constants $\perp_1, \dots, \perp_n$, and productions are added s.t. ($t_1, \dots, t_n$ are arbitrary trees that may contain non-terminals of $\mathcal{G}_L$) $\mathcal{L}(A_t(t_1, \dots, t_n)) = R^*(\{t' \mid t[\dots, \perp_i \leftarrow t_i, \dots] \to^*_{\mathcal{G}_L} t'\})$.

The role of A_t's is as follows. Since $\mathcal{G}_{R^*_\epsilon L}$ has to be context-free, we cannot create a production like $A_t(t_1, \dots, t_n) \to t'$ to simulate a rewrite step issued from the term $t[\dots, \perp_i \leftarrow t_i, \dots]$ (only $A_t(x_1, \dots, x_n) \to t'$ is allowed). However, $Depth(R)$ gives the depth of symbols that may be needed when rewriting. Then, when we deal with A_t, we first focus on the depth of symbols in t. When a symbol is deeper than $Depth(R)$, we get rid of the corresponding sub-term by moving it into arguments of A_t. To define the depth of a symbol, we must not take into account the non-terminals that occur above it, since a non-terminal is replaced by nothing if it is derived by a collapsing production.

Example 4.14 Let $R = \{g(x, y) \to c(h(x), f(y)),\ f(x) \to s(f(s(x))),\ h(s(x)) \to a\}$. Then, $Depth(R) = 1$. Let $E = \{g(s^*(a), a)\}$ generated by the set of production rules : $P_{\mathcal{G}_E} = \{S \to g(B, a),\ B \to a \,|\, s(B)\}$.

Here, E contains only one defined-function. So, $R^*_\epsilon(E) = R^*(E)$. Then, in the following, we use $R^*(E)$ instead of $R^*_\epsilon(E)$.

Let us denote by $S_{\mathcal{G}_{R^*_\epsilon L}}$ the axiom of $\mathcal{G}_{R^*_\epsilon L}$: we add $S_{\mathcal{G}_{R^*_\epsilon L}} \to A_{g(B,a)}$ since $S_{\mathcal{G}_L}$ is the axiom of the starting grammar and $S_{\mathcal{G}_L} \to g(B,a)$. No symbol is deeper than 1 in $g(B,a)$, then we try to rewrite $g(B,a)$ and derive B (it is achieved by the sub-program *Rewrite*). We add the productions :
$A_{g(B,a)} \to c(A_{h(B)}, A_{f(a)}) \mid A_{g(a,a)} \mid A_{g(s(B),a)}$.

Four new non-terminals have appeared. We have to deal with them. As previously, we try to rewrite and derive $h(B)$, $f(a)$ and $g(a,a)$. We add the productions :
$A_{h(B)} \to A_{h(s(B))} \mid A_{h(a)}$, $A_{f(a)} \to s(A_{f(s(a))})$ and $A_{g(a,a)} \to c(A_{h(a)}, A_{f(a)})$.
Let us remark that we do not rewrite and derive $(g(s(B),a))$. Indeed, it contains some symbol (here B) that are deeper than $Depth(R)$. We get rid of it by adding the production (*Move*) : $A_{g(s(B),a)} \to A_{g(s(\perp),a)}(B)$.

Four new non-terminals have appeared. $h(a)$ cannot be rewritten nor derived.
$g(s(\perp),a)$ can be rewritten. We add the production :
$A_{g(s(\perp),a)}(x) \to c(A_{h(s(\perp))}(x), A_{f(a)})$.
We have to deal with $A_{h(s(B))}$ and $A_{f(s(a))}$. We add the productions (*Move*) :
$A_{h(s(B))} \to A_{h(s(\perp))}(B)$ and $A_{f(s(a))} \to A_{f(s(\perp))}(a)$.

Two new non-terminals have appeared. $h(s(\perp))$ and $f(s(\perp))$ can be rewritten. We add the productions : $A_{h(s(\perp))}(x) \to a$ and $A_{f(s(\perp))}(x) \to sA_{f(s(s(\perp)))}(x)$.

One new non-terminal have appeared. We have to deal with $A_{f(s(s(\perp)))}$ We add the production (*Move*) : $A_{f(s(s(\perp)))}(x) \to A_{f(s(\perp))}(s(x))$.

Moreover, for each non-terminal A_t where $t = C[\perp_1, \ldots, \perp_n]$, we add the production $A_t(x_1, \ldots, x_n) \to C[x_1, \ldots, x_n]$ (MoveAll).

No more new non-terminals. The algorithm stops. The reader can checks that the resulting grammar really generates :
$\{g(s^*(a),a),\ c(h(s^*(a)),\ s^n(f(s^n(a)))),\ c(a, s^n(f(s^n(a))))\}$

To begin, we need some notations and some sub-programs that we use inside the main algorithm.

4.3.3 Algorithm : Sub-programs

- *Depth*(R) is an integer $\in \mathbb{N}$ that gives the maximal depth of symbol useful for rewriting over R as defined in Section 4.3.1.
- *Realized* is the set of non-terminals whose associated productions have already been created.
- *NT* is the list of non-terminals that are not in *Realized* (l::b means that we add b at the end of the list l).
- *Order* is a procedure that takes as parameters a non-terminal and its arguments $A_t(arg_1, \ldots, arg_n)$. It renames all the $\perp$ of t s.t. $\forall i,\ j\ (\perp_i \lhd \perp_j \Rightarrow i < j)$[6], permutes the corresponding arguments, and then deletes the extra arguments.

[6] $\perp_i \lhd \perp_j$ means that $\perp_i$ occurs on the left of $\perp_j$

Example 4.15 Let us consider $A_{f(\perp_1,\perp_5,\perp_4)}(x_1, x_2, x_3, x_4, B(x_2, x_3))$. Then, *Order* applied on it provides $A_{f(\perp_1,\perp_2,\perp_3)}(x_1, B(x_2, x_3), x_4)$.

- *Linearize* is a procedure that takes as parameters a non-terminal and its arguments $A_t(arg_1, \ldots, arg_n)$. It makes t linear by using $\perp_{n+1}, \ldots,$ and duplicates the corresponding arguments (addition at the tail of list).

Example 4.16 Let us consider $A_{f(B(\perp_1,\perp_1),\perp_2)}(x_1, x_2)$. Then, *Linearize* applied on it provides $A_{f(B(\perp_1,\perp_3),\perp_2)}(x_1, x_2, x_1)$.

- *Rewrite* is a procedure that takes as parameters a non-terminal A_s, a TRS R, the set of production rules $P_{\mathcal{G}_{R^*_\epsilon L}}$ in progress, the set *Realized*, and the list NT. This procedure rewrites s by R, and also derives the non-terminals of s. It updates the set of production rules, the list NT and finally, the set *Realized*.

Example 4.17 Let $R = \{\ f(x,y,z) \rightarrow c(g(y,z),x)\ \}$ and let $A_{f(\perp_1,\perp_2,\perp_3)}(x_1, x_2, x_3)$.
If we consider $s = f(\perp_1, \perp_2, \perp_3)$, $s \rightarrow_R c(g(\perp_2, \perp_3), \perp_1)$.
So, we obtain in a first time: $c(A_{g(\perp_2,\perp_3)}(x_1, x_2, x_3), x_1)$ (in the algorithm below, it is before applying *Order*).
Then, we add to the set of production rules:
$A_{f(\perp_1,\perp_2,\perp_3)}(x_1, x_2, x_3) \rightarrow c(A_{g(\perp_1,\perp_2)}(x_2, x_3), x_1)$

Example 4.18 In this example, we consider only the derivation step.
Let $A_{f(B)}$ be a non-terminal and $B \rightarrow s(B) \,|\, a \in P_{\mathcal{G}_L}$.
Then, we add to the set of production rules: $A_{f(B)} \rightarrow A_{f(s(B))} \,|\, A_{f(a)}$.

Rewrite$(A_s, R, P_{\mathcal{G}_{R^*_\epsilon L}}, Realized, NT)$
    `/******Rewriting******/`
    $\forall l \rightarrow r \in R$ `s.t.` $s \rightarrow_{[\epsilon, l \rightarrow r]}$ `Do`
        $n := ar(A_s)$;
        `Let` Θ `s.t.` $s = l\theta$;
        $t := r\Theta$;
        $\forall p$ `s.t.` $t|_p = \perp_i \ \wedge\ \neg(\exists q \in PosF(t)$ `s.t.` $q < p)$ `Do`
            $t := t[p \leftarrow x_i]$;

$posF := PosF(t)$;
While $posF \neq \emptyset$ Do
take $p \in posF$; $posF := posF \backslash \{p\}$;
$t := t[p \leftarrow Order(A_{t|_p}(x_1, \ldots, x_n))]$;
If $t(p) \notin Realized$ then $NT := NT :: t(p)$;
$P_{\mathcal{G}_{R_\epsilon^* L}} := P_{\mathcal{G}_{R_\epsilon^* L}} \cup A_s(x_1, \ldots, x_n) \to t$;
/******Derivation******/
$\forall p \in pos(s)$ s.t. $s(p)$ is a non-terminal Do
$n := ar(A_s); m := ar(s(p))$;
$\forall s(p)(x_1, \ldots, x_m) \to C[y_1, \ldots, y_k]$ Where C is a context $\in T(\Sigma, \mathcal{X})$
Do
Let $s|_p = s(p)(\alpha_1, \ldots, \alpha_m)$;
$t := C[\beta_1, \ldots, \beta_k]$ where $y_i = x_j \Rightarrow \beta_i = \alpha_j$ with $i \in \{1, \ldots, k\}$
and $j \in \{1, \ldots, m\}$;
$t' := Order(Linearize(A_t(x_1, \ldots, x_n)))$;
$P_{\mathcal{G}_{R_\epsilon^* L}} := P_{\mathcal{G}_{R_\epsilon^* L}} \cup A_s(x_1, \ldots, x_n) \to t'$;
If $t'(\epsilon) \notin Realized$ then $NT := NT :: t'(\epsilon)$;
$Realized := Realized \cup A_s$;

- Being given a term s and $p \in Pos(s)$, $Depth_{\overline{NT}}(p, s)$ returns the depth of $s(p)$ without taking non-terminals into account.

Example 4.19

Let $s =$

```
 f
 |
 B
/ \
s  a
|
D
```

then, $Depth_{\overline{NT}}(1.1, s) = Depth_{\overline{NT}}(1.2, s) = 1$,
$Depth_{\overline{NT}}(1.1.1, s) = 2$

$$Depth_{\overline{NT}}(p, s) = |p| - card(\{q \in Pos(s) \mid q < p \ \wedge\ s(q) \text{ is a non terminal}\})$$

- $Move$ is a procedure that takes as parameters a non-terminal A_s, the depth of R, the set of production rules $P_{\mathcal{G}_{R_\epsilon^* L}}$ in progress, the set *Realized*, the list NT and a boolean *notmove* (updated by the procedure when moves are made). This procedure consists in moving through the parameters of A_s (or as new parameters) all symbols that are strictly deeper than the depth of R. We so compare the depth of each symbol without taking into account non-terminals (denoted by $Depth_{\overline{NT}}$) with $Depth(R)$.

Example 4.20 Let us suppose that $Depth(R) = 1$ and let us consider A

```
     f
    / \
   B   c
   |   |
   s   ⊥3
   |
   D
  / \
 ⊥1  ⊥2
```

.

Then, we obtain (before applying *Order*) :

$$A \begin{array}{c} f \\ / \quad \backslash \\ \begin{array}{cc} B & c \\ | & | \\ s & \perp_3 \\ | & \\ D & \\ \begin{array}{cc} / & \backslash \\ \perp_1 & \perp_2 \end{array} & \end{array} \end{array} (x_1, x_2, x_3) \rightarrow A \begin{array}{c} f \\ / \quad \backslash \\ \begin{array}{cc} B & c \\ | & | \\ s & \perp_3 \\ | & \\ \perp_4 & \end{array} \end{array} (x_1, x_2, x_3, D(x_1, x_2))$$

Finally, we add to the set of production rules :

$$A \begin{array}{c} f \\ / \quad \backslash \\ \begin{array}{cc} B & c \\ | & | \\ s & \perp_3 \\ | & \\ D & \\ \begin{array}{cc} / & \backslash \\ \perp_1 & \perp_2 \end{array} & \end{array} \end{array} (x_1, x_2, x_3) \rightarrow A \begin{array}{c} f \\ / \quad \backslash \\ \begin{array}{cc} B & c \\ | & | \\ s & \perp_2 \\ | & \\ \perp_1 & \end{array} \end{array} (D(x_1, x_2), x_3)$$

$Move(A_s, depth_R, P_{\mathcal{G}_{R_\epsilon^* L}}, Realized, NT, notmove)$

```
new_s := s;  j := ar(A_s);  arg := (x_1, ..., x_j);
pos := Pos(s)\{p ∈ Pos(s) | ∃i ∈ {1...j}  s.t.  s|_p =⊥_i};
While pos ≠ ∅ Do
   Begin
      take p from pos;  pos := pos\p;
      if Depth_{NT-bar}(p, new_s) > depth_R then
         begin
            notmove := false;
            new_s := new_s[p ←⊥_{j+1}];
            arg := (x_1, ..., x_j, s|_p);
            j := j + 1;
         end;
   End;
t := Order(A_{new_s} arg);
P_{G_{R*_ε L}} := P_{G_{R*_ε L}} ∪ A_s(x_1, ..., x_j) → t;
If t(ε) ∉ Realized then NT := NT :: t(ε);
Realized := Realized ∪ A_s;
```

- *MoveAll* is a procedure that takes in parameters a non-terminal A_s, the set of production rules $P_{\mathcal{G}_{R_\epsilon^* L}}$ in progress, and the set *Realized*. It consists in transforming A_s to a term without non-terminals. It can be obtained by replacing each $\perp_i$ ($\imath \in \mathbb{N}$) appearing in s by its associated x_i.

Example 4.21 let $A \begin{array}{c} f \\ / \quad \backslash \\ \begin{array}{cc} s & \perp_2 \\ \begin{array}{cc} / & \backslash \\ \perp_1 & a \end{array} & \end{array} \end{array} (x_1, x_2)$ be a non-terminal.

Then, we add to the set of production rules : $A_{f(s(\perp_1, a), \perp_2)}(x_1, x_2) \to f(s(x_1, a), x_2)$.

$MoveAll(A_s, P_{\mathcal{G}_{R_\epsilon^* L}}, Realized)$
  $k := ar(A_s)$;
  Let $C \in T(\Sigma)$ s.t. $s = C[\perp_1 \dots \perp_k]$;
  $P_{\mathcal{G}_{R_\epsilon^* L}} := P_{\mathcal{G}_{R_\epsilon^* L}} \cup A_s(x_1, \dots, x_k) \to C[x_1 \dots x_k]$;
  $Realized := Realized \cup A_s$;

4.3.4 Main Algorithm

$Realized := \mathcal{N}_{\mathcal{G}_L}$;
$NT := \emptyset$;
$P_{\mathcal{G}_{R_\epsilon^* L}} := P_{\mathcal{G}_L}$;
Let $S_{\mathcal{G}_L}$ be the axiom of $\mathcal{G}_L$.
$\forall S_{\mathcal{G}_L} \to f(B_1, \dots, B_n) \in P_{\mathcal{G}_L}$ where $B_{i_{i\in 1..n}} \in \mathcal{N}_{\mathcal{G}_L}$ and $f \in \Sigma$ Do
  $P_{\mathcal{G}_{R_\epsilon^* L}} := P_{\mathcal{G}_{R_\epsilon^* L}} \cup S_{\mathcal{G}_{R_\epsilon^* L}} \to A_{f(B_1, \dots, B_n)}$
  $Realized := Realized \cup S_{\mathcal{G}_{R_\epsilon^* L}}$;
  $NT := NT :: A_{f(B_1, \dots, B_n)}$
While $NT \neq \emptyset$ Do
  Begin
    $A_s := head(NT)$;
    $NT := tail(NT)$;
    $Move(A_s, Depth(R), P_{\mathcal{G}_{R_\epsilon^* L}}, Realized, NT, notmove)$;
    If $notmove$ then $Rewrite(A_s, R, P_{\mathcal{G}_{R_\epsilon^* L}}, Realized, NT, notmove)$;
   $MoveAll(A_s, P_{\mathcal{G}_{R_\epsilon^* L}}, Realized)$;
  End.
  $\mathcal{N}_{\mathcal{G}_{R_\epsilon^* L}} := Realized$;

Theorem 4.22 *If $\mathcal{G}_L$ is NT-bounded (see Definition 2.6), then so is $\mathcal{G}_{R_\epsilon^* L}$.*

Theorem 4.23 *The algorithm is correct and complete and terminates.*

Proof. *The proofs of Theorem 4.22 and Theorem 4.23 are respectively given in the appendix Sections C and D.*

5 Conclusion

The preservation of regular tree languages through rewriting needs strong restrictions. Surprisingly, it is the same for context-free languages. Now, the following question arises : are there known classes of tree languages that are preserved through rewriting, assuming only weak restrictions ?

References

[1] M. Abadi. Security protocols and specifications. In *Second International Conference, FOSSACS'99*, volume 1578 of *LNCS*, pages 1–13. Springer-Verlag, March 1999.

[2] A. Arnold and M. Dauchet. Un théorème de duplications pour les forêts algébriques. In *Journal of Computer and System Sciences*, pages 13:223–244, 1976.

[3] M. Dauchet and S. Tison. The theory of ground rewrite systems is decidable. In *Fifth Annual IEEE Symposium on Logic Computer Science*, pages 242–248. IEEE Computer Society Press, Philadelphia, Pennsylvania, 1990.

[4] T. Genet. Decidable approximations of sets of descendants and sets of normal forms. In *Proceedings of 9th Conference RTA, Tsukuba (Japan)*, volume 1379 of *Lecture Notes in Computer Science*, pages 151–165. Springer-Verlag, 1998.

[5] T. Genet and F. Klay. Rewriting for cryptographic protocol verification. In *Proceedings 17th International CADE, Pittsburgh (Pen., USA)*, volume 1831 of *LNAI*. Springer-Verlag, 2000. (extended version in Technical Report RR-3921, Inria 2000).

[6] R. Gilleron. Decision problems for term rewrite systems and recognizable tree languages. In *STACS*, volume 480 of *LNCS*, pages 148–159. Springer-Verlag, 1991.

[7] J. Goubault-Larrecq. A method for automatic cryptographic protocol verification. In Beverly Sanders Dominique Méry Beverly Sanders, editor, editor, *Fifth International Workshop, FMPPTA 2000*, volume 1800 of *LNCS*. Springer-Verlag, 2000.

[8] J. Goubault-Larrecq. Vérification de protocoles cryptographiques : la logique à la rescousse! In editor J.Goubault-Larrecq, editor, *1st international workshop on communications security on the internet, SECI'02, Tunis, Tunisia*, pages 119–152, INRIA, 2002, Sept.

[9] D. Hofbauer, M. Huber, and G. Kucherov. Some results on top-context-free tree languages. In *Proceedings of the 19th International Colloquium on Trees in Algebra and Programming*, volume 787 of *Lectures Notes in Computer Science*, pages 157–171. Springer-Verlag, 1994.

[10] F. Jacquemard. Decidable approximations of term rewrite systems. In editor H. Ganzinger, editor, *Proceedings 7th Conference RTA, New Brunswick (USA)*, volume 1103 of *Lecture Notes in Computer Science*, pages 362–376. Springer-Verlag, 1996.

[11] D. Monniaux. Abstracting Cryptographic Protocols with Tree Automata. In *6th International Static Analysis Symposium*, volume 1694 of *LNCS*. Springer-Verlag, 1999. (appeared in Science of Computiner Programming, 47(2-3):177-202, 2003).

[12] P. Réty. Regular Sets of Descendants for Constructor-based Rewrite Systems. In *Proceedings of the 6th international conference LPAR*, number 1705 in Lecture Notes in Artificial Intelligence (LNAI), Tbilisi, Republic of Georgia, 1999. Springer Verlag.

[13] P. Réty and J. Vuotto. Regular Sets of Descendants by Leftmost Strategy. In *Second International WRS, Copenhagen (Denmark)*, 2002. (appeared in ENTCS, 70(6), 2002).

[14] P. Réty and J. Vuotto. Regular Sets of Descendants by some Rewrite Strategies. In *Proceedings of 13th Conference RTA, Copenhagen (Denmark)*, LNCS. Springer, 2002.

[15] B. Schneier. *Applied Cryptography : Protocols, algorithms, and source code in C.* John Wiley and sons edition, 1996.

[16] H. Seki, T. Takai, F. Youhei, and Y. Kaji. Layered Transducing Term Rewriting System and its Recognizability Preserving Property. In *13th International Conference RTA*, volume 2378 of *Lectures Notes in computer Science*. Springer-Verlag, 2002.

[17] T. Takai, Y. Kaji, and H. Seki. Right-linear finite path overlapping term rewriting systems effectively preserve recognizability. In L. Bachmair, editor, *Proceedings 11th Conference RTA, Norwich (UK)*, volume 1833 of *LNCS*, pages 246–260. Springer-Verlag, 2000.

Appendix

A Computing $R^*(E)$: Proofs

A.1 Proof of Theorem 4.8

- If $t(\epsilon) \in \mathcal{F}$ then obviously $R^*(E) = R^*_{\geq\epsilon}(E)$.
- Otherwise, let $Succ_t(\epsilon) = \{p_1, \ldots, p_n\}$. Let us remark that $\forall i,\ j \in \{1, \ldots, n\}$, $(i \neq j \Rightarrow p_i || p_j)$. Moreover, rewrite steps at incomparable positions can be commuted. Then, $R^*(E) = E[p_1 \leftarrow R^*_{\geq p_1}(E)] \ldots [p_n \leftarrow R^*_{\geq p_n}(E)]$.

A.2 Proof of Lemma 4.9

- If $Succ_t(p) = \emptyset$, then obviously $R^*_{\geq p}(E) = R^*_p(E)$.
- Otherwise, let $Succ_t(p) = \{p_1, \ldots, p_n\}$.
  By induction on the number k of nested defined-functions in $t|_p$.
  Let us define: $R^*_{>p}(L) = \{s' \mid \exists s \in L,\ s \rightarrow^*_{[u_1,\ldots,u_n]} s',\ \forall i(u_i > p)\}$.
  Note that $\forall i,\ j \in \{1, \ldots, n\}$, $(i \neq j \Rightarrow p_i || p_j)$. Moreover, rewrite steps at incomparable positions can be commuted. Then, obviously, and according to the induction hypothesis ($E|_{p_i}$ contains at most $k-1$ nested defined-functions).
  $R^*_{>p}(E) = E[p_1 \leftarrow R^*_{\geq p_1}(E)] \ldots [p_n \leftarrow R^*_{\geq p_n}(E)]$.
  Finally, $R^*_{\geq p}(E) = R^*_p(E[p_1 \leftarrow R^*_{\geq p_1}(E)] \ldots [p_n \leftarrow R^*_{\geq p_n}(E)])$.

A.3 Proof of Lemma 4.11

Let us consider $C \in T(\Sigma, \mathcal{X})$. Let us take $Var(C) = \{x_1, \ldots, x_n\}$. Let $\{x_{i_1}, \ldots, x_{i_k}\}$ be the non-linear variables of C.
Let us consider $\forall j \in \{1, \ldots, n\} \backslash \{i_1, \ldots, i_k\}$, $L = C[x_j \leftarrow L_j]$ where L_j is a NT-bounded context-free tree language. Obviously, L is NT-bounded context-free.
Let $L'_1, \ldots, L'_k$ be top-context-free tree languages.
Let us consider the c-substitution operation denoted by $\diamond_c$ in [9]. $L \diamond_c L'$ consists in substituting all occurrences of c in a term of L by the same term of L'.
Let $L'' = ((\ldots((L \diamond_{x_{i_1}} L'_1) \diamond_{x_{i_2}} L'_2) \ldots) \diamond_{x_{i_k}} L'_k)$. According to [9, Lemma 24], L'' is context-free. The fact that L'' is NT-bounded is trivial.

B Computing $R^*(E)$: An Example

In this example, we use $\perp$ instead of $\perp_1$ since there is only one variable (and so there is not ambiguity)
Let $R = \{f(x) \rightarrow s(f(s(x))),\ g(x) \rightarrow p(g(p(x)))\}$ and $E = g(f(a))$.

- We have first to deal with $L = E|_1 = \{f(a)\}$. The grammar that generates it is given by: $S_{\mathcal{G}_L} \rightarrow f(B)$, $B \rightarrow a$.
  Then, the set of production rules obtained after applying the algorithm is:
  $S_{\mathcal{G}_{R^*_\epsilon L}} \rightarrow A_{f(B)}$, $\quad A_{f(B)} \rightarrow A_{f(\perp)}(B) \mid f(B)$, $\quad A_{f(\perp)}(x) \rightarrow s(A_{f(s(\perp))}(x) \mid f(x)$,
  $A_{f(s(\perp))}(x) \rightarrow A_{\perp}(s(x)) \mid f(s(x))$

- Now, we have to deal with $L' = \{g(R^*_\epsilon(f(a)))\}$. The grammar that generates it, is given by: $S_{\mathcal{G}_{L'}} \to g(S_{\mathcal{G}_{R^*_\epsilon L}})$. Then, the set of production rules obtained after applying the algorithm is:
  $S_{\mathcal{G}_{R^*_\epsilon L'}} \to A_{g(S_{\mathcal{G}_{R^*_\epsilon L}})}$, $\quad A_{g(S_{\mathcal{G}_{R^*_\epsilon L}})} \to A_{g(\perp)}(S_{\mathcal{G}_{R^*_\epsilon L}}) \,|\, g(S_{\mathcal{G}_{R^*_\epsilon L}})$,
  $A_{g(\perp)}(x) \to p(A_{g(p(\perp))})(x) \,|\, g(x)$, $\; A_{g(p(\perp))}(x) \to A_{g(\perp)}(p(x)) \,|\, g(p(x))$.

Obviously, the reader can easily check that $\{p^k(g(p^k(s^n f(s^n(a)))))\}$ is well generated.

C Proof of Theorem 4.22

Let Σ be the signature and $\mathcal{N}_{\mathcal{G}_{R^*_\epsilon L}}$ (resp. $\mathcal{N}_{\mathcal{G}_L}$) be the set of non-terminals of $\mathcal{G}_{R^*_\epsilon L}$ (resp. $\mathcal{G}_L$).

Let us suppose that $\mathcal{G}_{R^*\epsilon L}$ is not NT-bounded and let us show that $\mathcal{G}_L$ neither. According to Definition 2.6,

$$\exists A_{\alpha_1}(x_1, \dots, x_n) \to A_{\alpha_2}(y_1, \dots, y_k) \in P_{\mathcal{G}_{R^*_\epsilon L}}$$

$$Where\ \exists J \subseteq \{1, \dots, n\} \text{ s.t. } \left|\begin{array}{l} \forall j \in J,\ y_j = \beta_j[x_a, \dots, x_b] \\ \forall i \in \{1, \dots, n\} \backslash J,\ y_j = x_j \end{array}\right.$$

$$\text{with } \left|\begin{array}{l} \{x_a, \dots, x_b\} \subseteq \{x_1, \dots, x_n\} \text{ and} \\ \beta_j \text{ is a context on } \Sigma \cup \mathcal{N}_{\mathcal{G}_{R^*_\epsilon L}} \\ \text{ that contains at least one non-terminal} \end{array}\right.$$

$$\text{s.t. } A_{\alpha_1} \to^*_{\mathcal{G}_{R^*_\epsilon L}} A_{\alpha_1}.$$

Remark :

- Indeed, it is well $A_{\alpha_1} \to^*_{\mathcal{G}_{R^*_\epsilon L}} A_{\alpha_1}$, because let us take B s.t. $B \in \beta_j$ for $j \in J$. Then, $\neg(B \to^*_{\mathcal{G}_{R^*_\epsilon L}} A_{\alpha_1})$ because $B \in P_{\mathcal{G}_L}$

- Another kind of rules can have nested non-terminals: $A_\alpha(x_1, \dots, x_n) \to \alpha[\perp_1 \leftarrow x_1, \dots, \perp_n \to x_n]$ where α contains nested non-terminals. However, this kind of rules are not recursive because every non-terminal of α belong to $P_{\mathcal{G}_L}$.

According to the algorithm of construction of $\mathcal{G}_{R^*_\epsilon L}$, these rules can only be obtained by applying $Move$. Then, for $A_{\alpha_1} \to^*_{\mathcal{G}_{R^*_\epsilon L}} A_{\alpha_1}$, it can only be effected by derivation of α_2.

Remark : Indeed, the rewriting does not product non-terminals.

$\forall j \in J$, there exists at least on non-terminal above $\perp_j$. Because of that, during the derivation of α_2, we can generate on the same branch deleted non-terminals.

$$\begin{array}{ll} A_{\alpha_1} \to^*_{\mathcal{G}_{R^*_\epsilon L}} A_{\alpha_1} & \Rightarrow \alpha_2 \to_{\mathcal{G}_L} \alpha_1 \\ & \Rightarrow \exists C_j(x_{a'}, \dots, x_{b'}) \to \dots C_j \dots B_j \dots \\ & \Rightarrow \mathcal{G}_L \text{ is not NT-bounded.} \end{array}$$

D Correctness, Completeness and termination Proofs

Let us recall that R is a constructor-based TRS that satisfies Restrictions 1, 2, 5, 6. Let E be the starting language that satisfies restriction 7 or restriction 8.

D.1 Correctness (Theorem 4.23)

Remark : Let L, $\mathcal{G}_L$ and $\mathcal{G}_{R^*_\epsilon L}$ as given in Section 4.3. Let $A_t^{\backslash n} \in \mathcal{N}_{\mathcal{G}_{R^*\epsilon L}}$. By construction, all non-terminals that appear in t are non-terminals of $\mathcal{G}_L$.
Below, we have just write A_t for $A_t(x_1, \ldots, x_n)$ and so t for $t[\perp_1 \leftarrow x_1, \ldots, \perp_n \leftarrow x_n]$. It is just for more clarity, but this does not change the principle of the proof.
Let us consider the following derivation :

$$S_{\mathcal{G}_{R^*_\epsilon L}} \rightarrow^*_{\mathcal{G}_{R^*_\epsilon L}} A_t \rightarrow t \rightarrow^!_{\mathcal{G}_L} t'$$

Let us show that $t' \in R^*_\epsilon(L)$.

- First, if we take $* = 0$. Then, obviously we have $S \rightarrow^!_{\mathcal{G}_L} t'$ and $t' \in R^*_\epsilon(L)$.
- Let us suppose that it is true for $* = n$ and let us take $* = n + 1$.

$$S_{\mathcal{G}_{R^*_\epsilon L}} \rightarrow^n_{\mathcal{G}_{R^*_\epsilon L}} A_{t_1} \rightarrow_{\mathcal{G}_{R^*_\epsilon L}} A_t \rightarrow t \rightarrow^!_{\mathcal{G}_L} t'$$

  By construction, we have three kind of treatment.

  - *MoveAll* returns to consider $* = n$, and we would have $A_{t_1} \rightarrow t_1$. Then, by induction hypothesis, t_1 derives itself (by $\mathcal{G}_L$) to a term of $R^*_\epsilon(L)$.
  - A step that consists to use *Move* if some symbols of t_1 are deeper than $Depth(R)$. I.e. we remove symbols that are not consumable. Since $A_t \rightarrow t$ consists in doing *MoveAll*, this step returns to apply *MoveAll* to A_{t_1}. Then, we obtain a term that belongs to $R^*_\epsilon(L)$.
  - A step that consists to use the *Rewrite* procedure if t_1 contains only symbols that are consumable (i.e. we do not apply *Move*). Let us recall that in *Rewrite*, we derive A_{t_1} and we rewrite if it is possible. So if we forget A, derivation step consists in : $t_1 \rightarrow_{\mathcal{G}_L} t \rightarrow^!_{\mathcal{G}_L} t'$ and rewriting step consists in : $t_1 \rightarrow_R \rightarrow^!_{\mathcal{G}_L} t'$. It is obvious that $t' \in R^*_\epsilon(L)$.

D.2 Completeness (Theorem 4.23)

When rewriting (procedure <u>Rewrite</u>) by $l \rightarrow r \in R$, the algorithm creates for each function f issued of r a non-terminal A_t where $t(\epsilon) = f$.
Therefore, we need to define a new rewriting : rewriting with decomposition of the resultant term.

Definition D.1

$$t \overset{dec}{\rightarrow}_{[p,l\rightarrow r,\sigma]} s \ \mathit{if} \ (t \rightarrow_{[p,l\rightarrow r,\sigma]} t' \ \wedge \ \exists p' \in PosF(t'), \ s = t'|_{p'}.$$

Lemma D.2

$$\textit{If } t \stackrel{dec^*}{\to}\stackrel{dec}{\to}_{l_1\to r_1}\stackrel{dec^*}{\to}\stackrel{dec}{\to}_{l_2\to r_2}\stackrel{dec^*}{\to} t' \textit{ and } l_1 \to r_1,\ l_2 \to r_2 \textit{ are not recursive,}$$

$$\textit{then } l_1 \to r_1 \neq l_2 \to r_2.$$

Proof. *It comes from Lemma 2.3.*

We define $Unfold_n(t)$ that cuts the term t at depth n. It is for simulating the procedure $\underline{Move}$.

Definition D.3 *Let t be a term, $n \in I\!N$, $x_1, \dots, x_n$ new fresh variables. $Unfold_n(t)$ returns a pair (t', σ') defined by:*

$$\textit{let } p_1, \dots, p_k \in \overline{Pos}(t) \textit{s.t. } |p_1| = \dots = |p_k| = n \textit{ (exhaustive list)}$$

$$\textit{Then,}\quad \begin{array}{l} t' = t[p_1 \to x_1] \dots [p_k \to x_k] \\ \sigma' = (x_1/t|_{p_1}) \dots (x_k/t|_{p_k}) \end{array}$$

Remark : $t'\sigma' = t$.

Theorem D.4 *Let R be a TRS satisfying our restrictions (see Introduction) and $l_1 \to r_1, \dots, l_n \to r_n \in R$. Let $k = Depth(R) + 1$, and t_1 be a term s.t. $PosF(t_1) = \{\epsilon\}$.*

$$\textit{If } t_1 \stackrel{dec}{\to}_{l_1\to r_1} t_2 \stackrel{dec}{\to}_{l_2\to r_2} t_3 \dots \stackrel{dec}{\to}_{l_{n-1}\to r_{n-1}} t_n \stackrel{dec}{\to}_{l_n\to r_n} t_{n+1}$$

then,

$$t''_1 \stackrel{dec}{\to} t'_2 \textit{ where } (t''_1, \sigma'_1) = Unfold_k(t_1) \textit{ and } t'_2\sigma'_1 = t_2,$$

$$t''_2 \stackrel{dec}{\to} t'_3 \textit{ where } (t''_2, \sigma'_2) = Unfold_k(t'_2) \textit{ and } t'_3\sigma'_2\sigma'_1 = t_3,$$

$$\dots t''_n \stackrel{dec}{\to} t'_{n+1} \textit{ where } (t''_n, \sigma'_n) = Unfold_k(t'_n) \textit{ and } t'_{n+1}\sigma'_n \dots \sigma'_1 = t_{n+1},$$

$$(t''_{n+1}, \sigma'_{n+1}) = Unfold_k(t'_{n+1})$$

and

$$\forall i,\ \forall p \in \overline{Pos}(t_i)\quad (|p| \leq Depth(R) - \sum_{j=1,\dots,i-1} Depth(l_j) \Rightarrow p \in \overline{Pos}(t''_i))$$

Proof. *Thanks to the restrictions, we have $\forall i$, $PosF(t_i) = \{\epsilon\}$, then each rewrite step is done at position ϵ. Moreover $k \geq 1$, then $\forall i$, $\epsilon \in Pos(t''_i)$. Therefore $\forall i$, $posF(t''_i) = \{\epsilon\}$.*
Now, the proof is by induction on n:

- *if $n = 1$, it is trivial.*
- *otherwise, let us suppose that the property holds for n, and let us prove it for $n + 1$.*
  Let us consider $t_n \stackrel{dec}{\to}_{l_n\to r_n} t_{n+1}$.

- *if $l_n \rightarrow r_n$ is not consuming, then no symbol is useful to apply this rule. So, the fact to apply $Unfold_k$ on t'_n does not change the fact that we can apply this rule and find the result.*
  Moreover, $Depth(l_n) = 0$. Since

$$\forall i,\ \forall p \in \overline{Pos}(t_i)\ \ (|p| \leq Depth(R) - \sum_{j=1,\ldots,i-1} Depth(l_j) \Rightarrow p \in \overline{Pos}(t''_i))$$

  is true for $i = n$ then it is still true for $i = n + 1$.
- *Otherwise, since $PosF(t''_n) = \{\epsilon\}$, we have $t''_n(\epsilon) = t_n(\epsilon)$, and $t''_n\sigma'_n \ldots \sigma'_1 = t_n$, and $\forall p \in \overline{Pos}(t_n)$ $(|p| \leq Depth(R) - \sum_{j=1,\ldots,n-1} Depth(l_j) \Rightarrow p \in \overline{Pos}(t''_n))$.*
  Since according to restriction 6, and thanks to Lemma D.2,
  $Depth(l_n) + \sum_{j=1,\ldots,n-1} Depth(l_j) \leq Depth(R)$,
  i.e $Depth(l_n) \leq Depth(R) - \sum_{j=1,\ldots,n-1} Depth(l_j)$.
  Moreover, $\overline{Pos}(l_n) \subseteq \overline{Pos}(t_n)$ (since $t_n \rightarrow_{l_n \rightarrow r_n}$) and $\forall p \in \overline{pos}(l_n)$, $|p| \leq Depth(l_n)$ (according to Definition 4.13). Therefore, $\forall p \in \overline{Pos}(l_n)$, $p \in \overline{Pos}(t''_n)$.
  Then, $t''_n \overset{dec}{\rightarrow}_{l_n \rightarrow r_n} t'_{n+1}$ and $t'_{n+1}\sigma'_n \ldots \sigma'1 = t_{n+1}$ and
  $\forall p \in \overline{Pos}(t_{n+1})$ $(|p| \leq Depth(R) - \sum_{j=1,\ldots,n} Depth(l_j) \rightarrow p \in \overline{Pos}(t'_{n+1}))$
  which trivially also holds for t''_{n+1}.

D.3 Termination (Theorem 4.23)

Let us recall that the number of symbol of t located strictly above p, and that are not non-terminals is given by $Depth_{\overline{NT}}(p, t)$ (see in Section 4.3.3 and Example 4.19).
For proving termination, we use the following notations too: For a term t,

$$Depth_{\overline{NT}}(t) = Max(\{Depth_{\overline{NT}}(p, t) \mid p \in Pos(t)\ \wedge\ t(p) \neq \perp\}).$$

$Nested\_NT(t) = Max(\{(|p| - Depth_{\overline{NT}}(p, t)) + 1 \mid p \in Pos(t) \wedge t(p)$ is a non-terminal)
(i.e. $Nested\_NT(t)$ is the maximal number of nested non-terminals in t).

Lemma D.5 *Let $b, b' \in I\!N$ and T be a set of trees.*

$$(\forall t \in T,\ (Depth_{\overline{NT}}(t) \leq b\ \wedge\ Nested\_NT(t) \leq b')) \Rightarrow T \textit{ is finite.}$$

Proof. *Trivial.*

Now, we can prove termination.

Proof. *Let us denote by $\mathcal{N}_{\mathcal{G}_{R^*_\epsilon L}}$ (resp. $\mathcal{N}_{\mathcal{G}_L}$) the non-terminals of $\mathcal{G}_{R^*_\epsilon L}$ (resp. $\mathcal{G}_L$). Let us recall that our algorithm generates a (new) non-terminal by moving symbol(s) into arguments of the considered non-terminal, by rewriting over R or by derivation of non-terminals of $\mathcal{G}_L$.*
Let us recall that $\overset{dec}{\rightarrow}$ is defined in Definition D.1. Then, $\forall A_t \in \mathcal{N}_{\mathcal{G}_{R^*_\epsilon L}}$, $\exists A_{t'} \in \mathcal{N}_{\mathcal{G}_L}$ s.t. $(t' \overset{dec}{\rightarrow}_R t \vee t' \rightarrow_{\mathcal{G}E} t)\ \wedge\ Depth_{\overline{NT}}(t') \leq Depth(R)$. $\mathcal{G}E$ being NT-bounded, then by Theorem 4.22 it is still for $\mathcal{G}_{R^*_\epsilon L}$. Therefore, according to Lemma 2.7, $Nested\_NT(t)$ is bounded. Then, we can apply Lemma D.5 and that proves termination.