Context-Free Tree languages for Descendants

Pierre Réty and Julie Vuotto

Lifo - Université d'Orléans,
B.P. 6759, 45067 Orléans cedex 2, France
{rety,vuotto}@lifo.univ-orleans.fr

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,... 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 [3,10,18,17,12]. 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].
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 :

[1] For more information on cryptographic protocol, the reader can refer to [16].

[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.

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 means 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[5] 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 $R^*(E)$.

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 $f \in \mathcal{F}^{\backslash n}$

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

[5] $l \to r$ is recursive means that : $l \to_R^* \alpha$ where there exists a position p of α s.t. $\alpha(p) = l(\epsilon)$.

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, $t \to_{[p, l \to r]} t'$) 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 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. 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 1. *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 3. 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 $f \in \mathcal{F}^{\setminus n}$ and $x_1, \ldots, x_n$ 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 $t \in T(\Sigma \cup \mathcal{N} \cup \{x_1, \ldots, x_n\})$[6]. 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 1.a below generates the regular tree language $\{f(s^*(a))\}$ while Grammar 1.b generates the context-free tree language $\{f(s^n(a), s^n(b))\}$.

Example 1.

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

Fig. 1. Properties

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

Definition 4. A context-free tree grammar is said *NT-bounded* if the transitions whose rhs's contain nested non-terminals, are not recursive.

Lemma 2. *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 3. *[2] Let $L = \{f(t, t) \mid t \in L', f \in \Sigma^{\setminus 2}\}$. L is context-free iff L' is top-context-free. And in this case, L is also top-context-free.*

Lemma 4. *[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.*

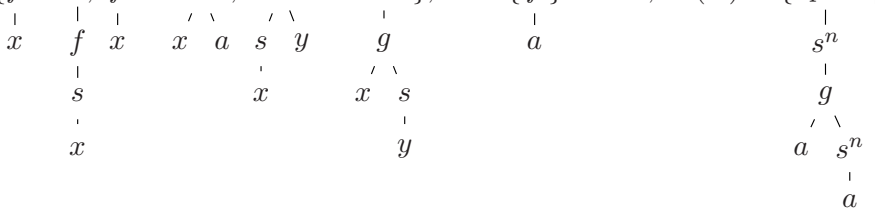
[6] Note that t is not necessarily linear.

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 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 $E = \{f(s) \,|\, s \in T_{\mathcal{C}}\}$. Then, $R^!(E) = \{c(s,s) \mid s \in T_{\mathcal{C}}\}$.
From Lemmas 3 and 4, we can conclude that $R^!(E)$ is not context-free.

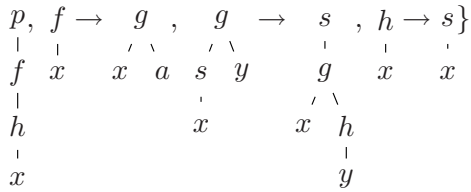
Counter-example 2 Restrictions 1 to 4 are satisfied, except Restriction 3.
$R = \{f \to p,\ f \to g(x,a),\ g(s(x),y) \to s(g(x,s(y)))\}$, $E = \{f\}$. Then, $R^!(E) = \{p^n(a) \mid n \in \mathbb{N}\}$.



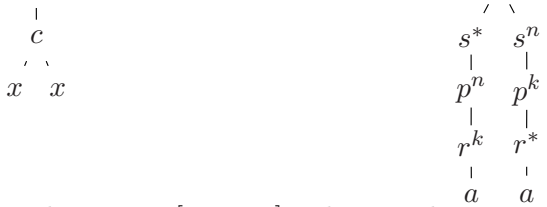
It is not context-free because generating the right-hand branch $p^n(s^n(g(s^n(a))))$ needs to count n three times.

Remark 1. 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 :

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

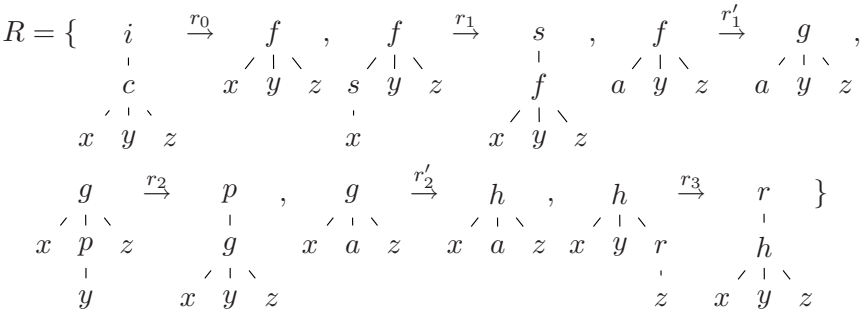


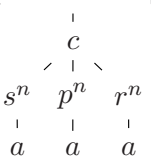
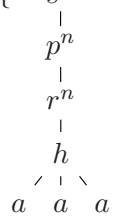
Counter-example 3 Restrictions 1, 2, 3, 6, 7 are assumed (Restriction 5 is not assumed).
Let $R = \{f(c(x,x)) \to x\}$ and $t = f(y)$. Let $L = \{c(s^*(p^n(r^k(a))), s^n(p^k(r^*(a)))) \mid n,k \in \mathbb{N}\}$.



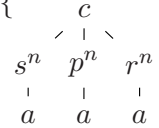
Let us consider $E = t[y \leftarrow L]$. The tree language L is recognized by the following NT-bounded context-free grammar :
$S \to F(a,a)$
$F(x,y) \to F(x,r(y)) \,|\, F_1(x,y) \qquad F_1(x,y) \to F_1(r(x),p(y)) \,|\, F_2(x,y)$
$F_2(x,y) \to F_2(p(x),s(y)) \,|\, F_3(x,y)$ $F_3(x,y) \to F_3(s(x),y) \,|\, c(x,y)$.
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 4 Restrictions 1, 2, 3, 5 and 7 are assumed (Restriction 6 is not assumed). Let :



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

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

$$E = \{ \; j \quad \theta \mid \theta : x \leftarrow \{ \quad \text{  } \quad \mid n \in \mathbb{N}\}\}$$

4 Positive Results

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 $\mathcal{G}_E = (\mathcal{N}_{\mathcal{G}_E}, \Sigma, S_{\mathcal{G}_E}, P_{\mathcal{G}_E})$.

In this section, we give some definitions needed to compute $R^*(E)$. Recall that we want to construct a tree grammar that generates $R^*(E)$. We proceed step by step : Suppose that $\{p_1, \ldots, p_n\}$ are outermost defined-functrions positions of E. Being incomparable positions, they can be treated independantly. Finding $R^*(E)$ consists in computing descendants of elements of E at positions $\geq p_i$, $\forall i \in \{1 \ldots n\}$ (so, we introduce $R^*_{\geq p_i}(E)$). We use the notation $\geq p_i$ in the case of there is at least one defined-function position below p_i. If it is not the case, it is sufficient to compute $R^*_{p_i}(E)$ (terms obtained by rewriting E at position p_i in any steps, i.e. at position p_i plus possibly at defined-functions positions issued of rewriting steps) since there is no nested defined-functions in rhs's.

4.1 Definitions

We first need to introduce how to construct a grammar that generates E in which we substitute some sub-languages $L_i = E|_{p_i}$ by other languages L'_i.

Definition 5. 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 3.
- 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 6. 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 7. 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 8. $t \rightarrow^+_{[p, rhs's]} t'$ 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.

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

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 4. $P(t_j)$ 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 9. 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 2. 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 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 10. 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 [15].

Theorem 1. *Let t be a linear term, and E be the language of instances of t.*

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

Lemma 5. *Let t be a linear term, and E be the language of instances of t. Let $p \in PosF(t)$.*

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

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 6. *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 5. 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 7. *Let t be a term. Let E be the set of instances of t 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

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 3. Let $R = \{\, h(s(x)) \stackrel{ru_1}{\to} x,\ g(p(x)) \stackrel{ru_2}{\to} h(x) \,\}$.

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 [15]).

Definition 11. 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 6. Only consuming rules matter when computing $Depth(R)$, because if $l \to r$ is not consuming, then $Depth(l) = 0$.

Explanations and Example

Recall that 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 $\bot_1, \ldots, \bot_n$, and productions are added s.t. ($t_1, \ldots, t_n$ are arbitrary trees that may contain non-terminals of $\mathcal{G}_L$) $\mathcal{L}(A_t(t_1, \ldots, t_n)) = R^*(\{t' \mid t[\ldots, \bot_i \leftarrow t_i, \ldots] \to^*_{\mathcal{G}_L} t'\})$.

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 $A_t(t_1, \ldots, t_n) \to t'$ to simulate a rewrite step issued from the term $t[\ldots, \bot_i \leftarrow t_i, \ldots]$ (only $A_t(x_1, \ldots, x_n) \to t'$ 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. Let R be the TRS given by the following set of rewrite rules:

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

Then, $Depth(R) = 1$. Obviously, R statisfies restrictions 1, 2, 5, 6.
Let a term $t = g(x, a)$. Obviously, t is linear. And let

$$E = \{\ g(s^n(a), a) \mid n \in \mathbb{N}\} = \{t\sigma \mid x\sigma = \{s^n(a) \mid n \in \mathbb{N}\}\}$$

generated by the set of production rules:

$$P_{\mathcal{G}_E} = \{S \to g(B, a),\ B \to a \mid s(B)\}.$$

The language $\{s^n(a) \mid n \in \mathbb{N}\}$ that instanciates the linear term t is a context-free tree language. Then, E satisfies restrictions 7.

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. 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:

$$A_{h(B)} \to A_{h(s(B))} \mid A_{h(a)},\ A_{f(a)} \to s(A_{f(s(a))}) \text{ and } A_{g(a,a)} \to c(A_{h(a)}, A_{f(a)})$$

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:

$$A_{g(s(B),a)} \to A_{g(s(\bot),a)}(B)$$

Four new non-terminals have appeared. $h(a)$ cannot be rewritten nor derived. $g(s(\bot), a)$ can be rewritten. We add the production:

$$A_{g(s(\bot),a)}(x) \to c(A_{h(s(\bot))}(x), A_{f(a)})$$

We have to deal with $A_{h(s(B))}$ and $A_{f(s(a))}$.
We add the productions:

$$A_{h(s(B))} \to A_{h(s(\bot))}(B) \text{ and } A_{f(s(a))} \to A_{f(s(\bot))}(a)$$

Two new non-terminals have appeared. $h(s(\bot))$ and $f(s(\bot))$ can be rewritten. We add the productions :

$$A_{h(s(\bot))}(x) \to a \text{ and } A_{f(s(\bot))}(x) \to sA_{f(s(s(\bot)))}(x).$$

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

$$A_{f(s(s(\bot)))}(x) \to A_{f(s(\bot))}(s(x)).$$

Moreover, for each non-terminal A_t where $t = C[\bot_1, \ldots, \bot_n]$, we add the production

$$A_t(x_1, \ldots, x_n) \to C[x_1, \ldots, x_n].$$

No more new non-terminals. The algorithm stops. The reader can checks that the resulting grammar really generates :

$$\{ \; g(s^n(a), a, h(s^n(a)), s^p(f(s^p(a)))), \; c(a, s^p(f(s^p(a)))), \; c(a, s^p(a)) \; | \; n, p \in \mathbb{N} \}$$

5 Conclusion

In this paper, we have studied the behaviour of descendants of a set of terms over a rewriting system. Since obtaining regular descendants has already been investigated in the past, we have studied languages of higher level in the Chomsky hierarchy (so more expressive languages) : Context-free tree languages.

We have seen that the set of descendants of a language depends on the form of the rewriting rules and the starting language too. So, some restrictions are necessary.

We have shown by an example how to construct a grammar that generates the set of descendants of elements of a set E. The reader can find a general algorithm that permits to construct a grammar that generates the set $R^*(E)$ in the full version of the paper [15].

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 Computer 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. P. Réty and J. Vuotto. Context-free Tree Languages for Descendants. Research Report RR-LIFO-2004-04, LIFO, Laboratoire d'Informatique Fondamentale d'Orléans, Université d'Orléans, BP 6759, F-45067 Orléans Cedex 2, 2004.
http://www.univ-orleans.fr/SCIENCES/LIFO/prodsci/rapports/RR/RR2004/RR-2004-04.ps.gz
16. B. Schneier. *Applied Cryptography : Protocols, algorithms, and source code in C*. John Wiley and sons edition, 1996.
17. 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.
18. 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.