Visibly Pushdown Languages and Term Rewriting

Jacques Chabin Pierre Réty

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

Abstract. To combine tree languages with term rewriting, we introduce a new class of tree languages, that both extends regular languages and restricts context-free languages, and that is closed under intersection (unlike context-free languages). To do it, we combine the concept of visibly pushdown language, with top-down pushdown tree automata, and we get the visibly pushdown tree automata. Then, we use them to express the sets of descendants for a sub-class of growing term rewrite systems, and thanks to closure under intersection, we get that joinability and (restricted) unifiability are decidable.

Keywords : tree languages, term rewriting.

1 Introduction

Tree languages allow to express, manipulate, and decide properties over infinite sets of first-order terms. This is why tree languages have been combined with several other domains, like term rewriting [3, 9], logic programming [10], concurrency [12, 5] (process algebra), protocol verification [4],...

We want to combine tree languages with term rewriting. However, to get something interesting, it is desirable that the considered tree languages have both a big expressivity and good properties. Since these two requirements are contradictory, one will look for a middle-way in between. Regular tree languages have good properties, but have a poor expressivity. Context-free tree languages are much more expressive, but have fewer good properties, in particular they are not closed under intersection. Some other expressive tree languages have the same disadvantages [14, 5, 11], or do not extend regular languages [7].

In this paper, we introduce a new class of tree languages, that both extends regular languages and restricts context-free languages, and that is closed under intersection. To do it, we combine the concept of visibly pushdown language, already used for string languages [2], with the top-down pushdown tree automata of I. Guessarian [6]. We call these new languages *Visibly Pushdown Tree Languages (VPTL)*, and show that they are closed under union and intersection. As a sub-class of context-free languages, membership and emptiness are decidable. This work is presented in Section 2.

Next, we define the *Visibly Context-free Term Rewrite Systems (VCF-TRS)* as a sub-class of growing term rewrite systems. In [8] it is proved that the set of predecessors of a given regular tree language by a linear (weakened to left-linear in [13]) growing rewrite system, is also regular. Consequently reachability is decidable. However, nothing has been proved about descendants (successors). In this paper, we prove that the descendants of a given VPTL (which may be regular) by a linear VCF-TRS, is also a VPTL. Since VPTL's are closed under intersection, joinability is decidable in linear VCF-TRS's, whereas it is undecidable in growing rewrite systems. As a consequence, it is decidable whether two linear terms with disjoint variables are unifiable modulo a confluent linear VCF-TRS. Thanks to expressivity and closure under intersection, other applications of VPTL's should arise. This work is presented in Section 3.

2 Visibly Pushdown Tree Languages

A *pushdown alphabet* is a finite set $\Sigma = \Sigma_c \cup \Sigma_r \cup \Sigma_l$ where $\Sigma_c, \Sigma_r, \Sigma_l$ are disjoint. Σ_c is the set of *call symbols*, Σ_r is the set of *return symbols*, Σ_l is the set of *local symbols*. Intuitively, the automaton pushes onto the stack when it reads a call symbol, it pops the stack when it reads a return symbol, and it leaves the stack unchanged when it reads a local symbol.

Each symbol $f \in \Sigma$ has a unique arity, denoted by $ar(f)$. We consider a set of variables Var. The notions of first-order term, position, substitution, term rewriting, are defined as usual. Let us just precise some notations. T_Σ denotes the set of ground terms over Σ. For a term t, $Var(t)$ is the set of variables of t, $Pos(t)$ is the set of positions of t. For $p \in Pos(t)$, $t(p)$ is the symbol of $\Sigma \cup Var$ occurring at position p in t, and $t|_p$ is the subterm of t at position p. $PosVar(t) = \{p \in Pos(t) \mid t(p) \in Var\}$, $PosNonVar(t) = \{p \in Pos(t) \mid t(p) \notin Var\}$. Note that if $p \in PosNonVar(t)$, $t|_p = f(t_1, \dots, t_n)$, and $i \in \{1, \dots, n\}$, then $p.i$ is the position of t_i in t. For $p, p' \in Pos(t)$, $p < p'$ means that p occurs in t strictly above p'.

The automata defined below work top-down. States are binary symbols, whose first argument is a term in $T_{\Sigma \cup Var}$ and the second argument is a stack. Stacks are terms over the signature Π, which contains only unary symbols and one constant.

Definition 1. *A* Visibly Pushdown Tree Automaton (VPTA) *is a tuple $A = (\Sigma, Q, q_0, \Pi, z_0, \Delta)$ where Σ is a pushdown alphabet, Q is a finite set of states of arity 2, $q_0 \in Q$ is the initial state, Π is a finite set of stack symbols of arities in $\{0, 1\}$ which contains only one constant z_0 (z_0 is the empty stack), Δ is a set of rewrite rules, called* transitions*, of the form ($q, q_1, \dots, q_n \in Q$, $f \in \Sigma$, $e, e_1, \dots, e_n \in \Pi$, $x_1, \dots, x_n$ and z are variables):*

- $q(f(x_1, \dots, x_n), z) \rightarrow f(q_1(x_1, e_1(z)), \dots, q_n(x_n, e_n(z)))$ *if* $f \in \Sigma_c$
- $q(f(x_1, \dots, x_n), e(z)) \rightarrow f(q_1(x_1, z), \dots, q_n(x_n, z))$ *if* $f \in \Sigma_r$
- $q(f(x_1, \dots, x_n), z_0) \rightarrow f(q_1(x_1, z_0), \dots, q_n(x_n, z_0))$ *if* $f \in \Sigma_r$
- $q(f(x_1, \dots, x_n), z) \rightarrow f(q_1(x_1, z), \dots, q_n(x_n, z))$ *if* $f \in \Sigma_l$

- $q(x,\, z) \;\to\; q_1(x,\, z)$ *(empty transition)*

f may be a constant, whereas $e, e_1, \ldots, e_n$ are always unary.
The language $L(A)$ recognized by A is $L(A) = \{t \in T_\Sigma \mid q_0(t, z_0) \to^*_\Delta t\}$. A set of terms that can be recognized be a VPTA is called Visibly Pushdown Tree Language (VPTL).

Note that the automaton can check the stack top only if it reads a return symbol.

Example 1. Let

$$E = \left\{ \begin{array}{ccc} & c^n & \\ & \mid & \\ & f & \\ / & & \backslash \\ g^n & & s \\ \mid & & \mid \\ a & & h^n \\ & & \mid \\ & & a \end{array} \;\middle|\; n \in \mathbb{N} \right\}$$

where $c^n = c(c(\cdots c(\ldots)))$ in which c occurs n times. E is a VPTL recognized by the automaton $A = (\Sigma, Q, q_0, \Pi, z_0, \Delta)$ where $\Sigma_c = \{c\}$, $\Sigma_r = \{g, h, a\}$, $\Sigma_l = \{f, s\}$, $Q = \{q_0, q_1, q_2, q_3\}$, $\Pi = \{e\}$, and $\Delta = \{$

$$\begin{gathered} q_0(c(x), z) \to c(q_0(x, e(z))) \\ q_0(f(x, y),\, z) \to f(q_1(x, z),\, q_2(y, z)) \\ q_1(g(x), e(z)) \to g(q_1(x), z) \\ q_1(a, z_0) \to a \\ q_2(s(x), z) \to s(q_3(x, z)) \\ q_3(h(x), e(z)) \to h(q_3(x), z) \\ q_3(a, z_0) \to a \; \} \end{gathered}$$

Note that a checks that the current stack is the empty stack z_0, which ensures that there are exactly n occurrences of g and n occurrences of h.

Example 2. Let

$$E' = \left\{ \begin{array}{ccc} & c^n & \\ & \mid & \\ & f & \\ / & & \backslash \\ g^n & & s^i \\ \mid & & \mid \\ a & & h^j \\ & & \mid \\ & & a \end{array} \;\middle|\; n, i, j \in \mathbb{N} \wedge i + j = n \right\} \quad \text{and} \quad E'' = \left\{ \begin{array}{ccc} & c^n & \\ & \mid & \\ & f & \\ / & & \backslash \\ g^n & & s^n \\ \mid & & \mid \\ a & & h^n \\ & & \mid \\ & & a \end{array} \;\middle|\; n \in \mathbb{N} \right\}$$

E' is a VPTL if s is a return symbol. E'' is not a VPTL : c should be call and s, h should be return to be able to count n, but reading one c necessarily pushes exactly one symbol onto the stack, and reading one s or one h pops exactly one symbol from the stack.
On the other hand, a language like $\{f(g^n(a), h^n(a)) \mid n \in \mathbb{N}\}$ is not a VPTL, since to check that the number of g and h are the same, we need to pop a stack

of height n in both branches, and no symbol (it was c in Languages E and E') enables to build such a stack.

Theorem 2.

- *If L is a regular tree language over Σ, then L is also a visibly pushdown tree language over any partition $\Sigma_c \cup \Sigma_r \cup \Sigma_l$ of Σ.*
- *Every visibly pushdown tree language is a context-free language.*

Proof. 1) Since L is regular, it can be recognized by a top-down automaton without a stack $A = (\Sigma, Q, q_0, \Delta)$. Let $\Pi = \{e, z_0\}$, and Q' be the set composed of the states of Q considered as symbols of arity 2. Let Δ' be the set of transitions built as follows : for each non-empty transition
$q(f(x_1, \ldots, x_n)) \to f(q_1(x_1), \ldots, q_n(x_n))$ of Δ, we add into Δ' :

- $q(f(x_1, \ldots, x_n),\, z) \;\to\; f(q_1(x_1, e(z)), \ldots, q_n(x_n, e(z)))$ if $f \in \Sigma_c$
- $q(f(x_1, \ldots, x_n),\, e(z)) \;\to\; f(q_1(x_1, z), \ldots, q_n(x_n, z))$ if $f \in \Sigma_r$
- $q(f(x_1, \ldots, x_n),\, z_0) \;\to\; f(q_1(x_1, z_0), \ldots, q_n(x_n, z_0))$ if $f \in \Sigma_r$
- $q(f(x_1, \ldots, x_n),\, z) \;\to\; f(q_1(x_1, z), \ldots, q_n(x_n, z))$ if $f \in \Sigma_l$

and for each empty transition $q(x) \to q_1(x)$ of Δ, we add into Δ' :

- $q(x,\, z) \;\to\; q_1(x,\, z)$

The automaton $A' = (\Sigma, Q', q_0, \Pi, z_0, \Delta')$ is a VPTA which also recognizes L.

2) VPTA's are particular cases of the top-down pushdown automata of [6], which recognize exactly context-free tree languages.

Although VPTL's are more expressive than regular tree languages, they still have good properties.

Theorem 3. *Visibly Pushdown Tree Languages are closed under union and intersection. Emptiness and membership are decidable.*

Unsurprisingly, the number of states (resp. transitions) of the union automaton is in the order of the sum of the number of states (resp. transitions) of the initial automata. The number of states (resp. transitions) of the intersection automaton is in the order of the product of the number of states (resp. transitions) of the initial automata if they have no empty transitions.

Proof. Let L_1 and L_2 be VPTL's over the same pushdown alphabet Σ. Let $A_1 = (\Sigma, Q_1, q_0^1, \Pi_1, z_0^1, \Delta_1)$ and $A_2 = (\Sigma, Q_2, q_0^2, \Pi_2, z_0^2, \Delta_2)$ be automata that recognize L_1 and L_2 respectively. We assume that Q_1 and Q_2 are disjoint, and Π_1 and Π_2 are disjoint.

1) $L_1 \cup L_2$ is recognized by $A = (\Sigma, Q, q_0, \Pi, z_0, \Delta)$ where $Q = Q_1 \cup Q_2 \cup \{q_0\}$, $\Pi = \Pi_1 \cup \Pi_2 \cup \{z_0\}$ and $\Delta = \Delta_1 \cup \Delta_2 \cup \{q_0(x, z) \to q_0^1(x, z),\, q_0(x, z) \to q_0^2(x, z)\}$.

2) $L_1 \cap L_2$ is recognized by $A = (\Sigma, Q, q_0, \Pi, z_0, \Delta)$ where $Q = Q_1 \times Q_2$, $q_0 = (q_0^1, q_0^2)$, $\Pi = \Pi_1 \times \Pi_2$, $z_0 = (z_0^1, z_0^2)$, and Δ is the following set of transitions:

- $(q^1, q^2)(f(x_1, \ldots, x_n), z) \rightarrow$
$$f((q_1^1, q_1^2)(x_1, (e_1^1, e_1^2)(z)), \ldots, (q_n^1, q_n^2)(x_n, (e_n^1, e_n^2)(z)))$$
if $f \in \Sigma_c$, $q^1(f(x_1, \ldots, x_n), z) \rightarrow f(q_1^1(x_1, e_1^1(z)), \ldots, q_n^1(x_n, e_n^1(z))) \in \Delta_1$ and $q^2(f(x_1, \ldots, x_n), z) \rightarrow f(q_1^2(x_1, e_1^2(z)), \ldots, q_n^2(x_n, e_n^2(z))) \in \Delta_2$.
- $(q^1, q^2)(f(x_1, \ldots, x_n), (e^1, e^2)(z)) \rightarrow f((q_1^1, q_1^2)(x_1, z), \ldots, (q_n^1, q_n^2)(x_n, z))$ if $f \in \Sigma_r$, $q^1(f(x_1, \ldots, x_n), e^1(z)) \rightarrow f(q_1^1(x_1, z), \ldots, q_n^1(x_n, z)) \in \Delta_1$ and $q^2(f(x_1, \ldots, x_n), e^2(z)) \rightarrow f(q_1^2(x_1, z), \ldots, q_n^2(x_n, z)) \in \Delta_2$.
- $(q^1, q^2)(f(x_1, \ldots, x_n), z_0) \rightarrow f((q_1^1, q_1^2)(x_1, z_0), \ldots, (q_n^1, q_n^2)(x_n, z_0))$ if $f \in \Sigma_r$, $q^1(f(x_1, \ldots, x_n), z_0^1) \rightarrow f(q_1^1(x_1, z_0^1), \ldots, q_n^1(x_n, z_0^1)) \in \Delta_1$ and $q^2(f(x_1, \ldots, x_n), z_0^2) \rightarrow f(q_1^2(x_1, z_0^2), \ldots, q_n^2(x_n, z_0^2)) \in \Delta_2$.
- $(q^1, q^2)(f(x_1, \ldots, x_n), z) \rightarrow f((q_1^1, q_1^2)(x_1, z), \ldots, (q_n^1, q_n^2)(x_n, z))$ if $f \in \Sigma_l$, $q^1(f(x_1, \ldots, x_n), z) \rightarrow f(q_1^1(x_1, z), \ldots, q_n^1(x_n, z)) \in \Delta_1$ and $q^2(f(x_1, \ldots, x_n), z) \rightarrow f(q_1^2(x_1, z), \ldots, q_n^2(x_n, z)) \in \Delta_2$.
- $(q^1, q)(x, z) \rightarrow (q_1^1, q)(x, z)$ if $q \in Q_2$ and $q^1(x, z) \rightarrow q_1^1(x, z) \in \Delta_1$.
- $(q, q^2)(x, z) \rightarrow (q, q_1^2)(x, z)$ if $q \in Q_1$ and $q^2(x, z) \rightarrow q_1^2(x, z) \in \Delta_2$.

Note that we crucially use the fact that A_1 and A_2 being VPTA's over the same pushdown alphabet, they synchronize on the push and pop operations on the stack.

3) VPTA's are particular Guessarian's pushdown tree automata [6], which have been proved to be equivalent to context-free languages (i.e. generated by context-free grammars). Therefore emptiness and membership are decidable.

Remark 4. : VPTL's are different from languages accepted by NP-NTA's [1], even if they have the same closure and decidability properties. Indeed, NP-NTA's deal with binary nested trees (sort of DAG's), which are necessarily infinite (all branches are infinite). Moreover, statuses call, return, local, are not attached to symbols in Σ, they are attached to nodes, i.e. positions.

3 Visibly Context-free Term Rewrite Systems

Definition 5. *A rewrite system R is* growing *[8] if for every $l \rightarrow r \in R$, each variable occurring both in l and in r occurs at depth 0 or 1 in l.*
A rewrite system R is context-free *if for every $l \rightarrow r \in R$, $l = f(x_1, \ldots, x_n)$ where $x_1, \ldots, x_n$ are distinct variables.*
A symbol is reducible if it occurs at the top of at least one rewrite rule. So, the set of reducible symbols is $RED(R) = \{f \in \Sigma \mid \exists l \rightarrow r \in R, l(\epsilon) = f\}$.

Note that context-free is a restriction stronger than growing. However, even if R is context free, joinability is undecidable, and in general the set of descendants $R^*(\{t\})$ is not a VPTL (otherwise emptiness of VPTL's would be undecidable). This is why the stronger restriction *visibly context-free* is introduced in the following.

Proposition 6. *Joinability in context-free rewrite systems is undecidable.*

Proof. Let $(u_1, v_1), \ldots, (u_m, v_m)$ be an instance of PCP (Post Correspondence Problem). Let $R = \{f(x) \to u_i(f(i(x))) \mid i \in \{1, \ldots, m\}\} \cup \{g(x) \to v_i(g(i(x))) \mid i \in \{1, \ldots, m\}\} \cup \{f(x) \to s(x),\ g(x) \to s(x)\}$.
R is context-free. However
$\exists t \mid f(a) \to_R^* t \ {}_R^*\!\leftarrow g(a) \iff R^*(f(a)) \cap R^*(g(a)) \neq \emptyset \iff$
$\exists i_1, \ldots, i_k,\ j_1, \ldots, j_{k'} \mid u_{i_1} \ldots u_{i_k} s\, i_k \ldots i_1 = v_{j_1} \ldots v_{j_{k'}} s\, j_{k'} \ldots j_1 \iff$
$k = k' \wedge i_1 = j_1, \ldots i_k = j_k \wedge u_{i_1} \ldots u_{i_k} = v_{i_1} \ldots v_{i_k} \iff$ this PCP instance has at least one solution.

Given a pushdown alphabet Σ and a term t, if we try to reduce $q(t, z_0)$ with transitions of a VPTA over Σ, for each $p \in Pos(t)$ the height of the current stack obtained when reducing at position p depends only on call and return symbols within t, and does not depend on q.

Definition 7. *We define the weight $w(f)$ of $f \in \Sigma \cup Var$ by:*
- $w(f) = 1$ *if* $f \in \Sigma_c$,
- $w(f) = -1$ *if* $f \in \Sigma_r$,
- $w(f) = 0$ *if* $f \in \Sigma_l$ *or* f *is a variable.*

The height of stack is the mapping $h : Pos(t) \to Z$ defined inductively by:
- $h(\epsilon) = w(t(\epsilon))$,
- $\forall p \in PosNonVar(t), \forall i \in \{1, \ldots, ar(t(p))\},\ h(p.i) = h(p) + w(t(p.i))$.

Definition 8. *The linear context-free rewrite system R is* visibly *over the pushdown alphabet $\Sigma = \Sigma_c \cup \Sigma_r \cup \Sigma_l$ if*
- $RED(R) \subseteq \Sigma_l$
- $\forall l \to r \in R$, $Var(l) = Var(r)$ *and* $\forall p \in PosNonVar(r)$:
  - $h(p) \geq 0$
  - $\forall i \in \{1, \ldots, ar(t(p))\}(t(p.i) \in Var \Longrightarrow (h(p) = 0 \wedge \forall p' < p, h(p') > 0))$

VCF-TRS *means Visibly Context-Free Term Rewrite System.*

Consequently, if $l \to r \in R$, then $l(\epsilon) \in \Sigma_l$, and
- r is a variable
- or r is shallow[1] and $r(\epsilon) \in \Sigma_l$
- or $r(\epsilon) \in \Sigma_c$ and every position just above a variable is the corresponding return (it pops from the stack the symbol pushed by $r(\epsilon)$).

Example 3. $R = \{f(x) \to f(f(x))$ is not visibly context-free, because $f \in RED(R) \subseteq \Sigma_l$ and for Positions $p = 1$, $p' = \epsilon$ in the right-hand-side, p is just above a variable whereas $p' < p \wedge \neg(h(p') > 0)$. Intuitively, in the right-hand-side, to distinguish the language generated by reducing the inner f from the one generated by the outer f, we need the stack. But the stack is not used

[1] Every variable occurs at depth 1.

since every symbol is local.
Now let $c \in \Sigma_c$ and $s \in \Sigma_r$. $R' = \{f(x) \to c(f(f(s(x))))\}$ is visibly context-free.

$$R'' = \{\ f(x, y) \to c(f(s(x), a), f(b, u(y)))\ \}$$

is visibly-context free with $\Sigma_c = \{c\}$, $\Sigma_r = \{s, a, b, u\}$, $\Sigma_l = \{f\}$.
The TRS given in Example 4 (page 9) is visibly context-free.

Definition 9. *Given a rewrite system R and a set of ground terms E, the set of* descendants *of E by R is $R^*(E) = \{t' \in T_\Sigma \mid \exists t \in E,\ t \to_R^* t'\}$.*

Theorem 10. *Let R be a linear visibly context-free rewrite system and E be a visibly pushdown tree language over the same pushdown alphabet Σ. Then $R^*(E)$ is a also a visibly pushdown tree language over Σ.*

Theorem 10 comes from the algorithm presented in the following section, which is proved to be terminating, complete, and sound.

Corollary 11. *Reachability and joinability are decidable for linear visibly context-free rewrite systems.*

Consequently, and according to the proof of Proposition 6, the PCP instances such that R is visibly context-free form a decidable sub-class of PCP.

Theorem 12. *It is decidable whether two linear terms with disjoint variables are unifiable modulo a confluent linear VCF-TRS.*

Proof. Let S be the set of all ground substitutions.
Since t and t' have disjoint variables and R is confluent, t and t' are R-unifiable iff there exists $\theta, \theta' \in S$ s.t. $\theta t \to_R^* u\ {}_R^*\!\leftarrow \theta' t'$, which is equivalent to

$$R^*(\{\sigma t \mid \sigma \in S\}) \cap R^*(\{\sigma' t' \mid \sigma' \in S\}) \neq \emptyset$$

Since t and t' are linear, $\{\sigma t \mid \sigma \in S\}$ and $\{\sigma' t' \mid \sigma' \in S\}$ are regular languages, and from Theorem 2 are also VPTL's. From Theorem 10, their descendants are also VPTL's. Then we can compute their intersection and check the emptiness.

3.1 Algorithm for computing descendants

The algorithm is illustrated by Example 4 (see below).

The algorithm starts with an automaton A_E that recognizes E, and achieves overlap steps. An overlap step overlaps a rewrite rule of R with a transition of Δ , and adds new states and new transitions into the automaton. If a transition existed already, it is not added.

Let $l \to r \in R$. Since R is supposed to be linear visible context-free, l has the form $l = f(x_1, \ldots, x_n)$ where $f \in \Sigma_l$ (because f is reducible). If Δ contains a transition like $q(f(x_1, \ldots, x_n),\, z) \rightarrow f(q_1(x_1, z), \ldots, q_n(x_n, z))$, a term of the form $C[\sigma l]$ may be recognized by the automaton. In order to recognize $C[\sigma r]$ as well, the algorithms adds states and transitions so that $q(r,\, z) \rightarrow^*_\Delta \theta r$ where $\theta = (x_1/q_1(x_1, z), \ldots, x_n/q_n(x_n, z))$. To do it, we introduce states q^p for each $p \in PosNonVar(r)\backslash\{\epsilon\}$. If $l \rightarrow r$ causes several overlap steps, we re-use the same states q^p (otherwise, the algorithm would not terminate). Thanks to the stack and the restrictions on R, it does not make a confusion, i.e. the algorithm is sound. However, if r contains call and return symbols, we use new stack symbols for each overlap step (otherwise the algorithm would not be sound).

To each transition is associated a unique identifier (i, p) where $i \in \mathbb{N}$ and p is a position. Intuitively, transition (i, p) has been created by the i^{th} overlap step, and allows to recognize the symbol $r(p)$ for some $l \rightarrow r \in R$. By convention, the transitions of the initial automaton A_E have identifiers of the form $(0, p)$ (and in this case, p's are chosen arbitrarily).
We will write $(i, p) \oplus (l \rightarrow r)$ to denote the overlap step between the transition (i, p) and the rewrite rule $l \rightarrow r$.

The overlap step $(x, u) \oplus (l_j \rightarrow r_j)$. Let A be the current automaton, and suppose that i_{max} overlap steps have already been done. Let $i = i_{max} + 1$.
Suppose there are $l_j \rightarrow r_j \in R$ and a transition in Δ of the form $q(l_j, z) \rightarrow \theta l_j$ identified by (x, u).

In order to know which stack symbols should be popped by return symbols, we first define a mapping $stack : Pos(r_j) \rightarrow T_{\{e_i^k | k \in \{1,\ldots,armax(\Sigma)\}\} \cup \{z_0\}}$, in an inductive way ($armax(\Sigma)$ is the maximum arity of symbols in Σ). Intuitively, $stack(p)$ is the stack obtained when starting with the initial stack z_0 and reading r_j until position p (just before reading the symbol that occurs at position p). For a stack s, $head(s)$ denotes the top symbol of s, and $tail(s)$ is the stack obtained by popping the head of s.

$$stack(\epsilon) = z_0$$
$$stack(p.k) = \left| \begin{array}{l} stack(p) \;\; \text{if } r_j(p) \in \Sigma_l \\ e_i^k(stack(p)) \;\; \text{if } r_j(p) \in \Sigma_c \\ tail(stack(p)) \;\; \text{if } r_j(p) \in \Sigma_r \end{array} \right.$$

In the transitions below, each symbol of the form q_j^y is a state that encodes position y of r_j, except for border positions of r_j, i.e. when $y \in \{\epsilon\} \cup PosVar(r_j)$. In this case we use the states coming from transition (x, u), hence:

- $q_j^\epsilon = q$
- $\forall p.k \in PosVar(r_j)$, $q_j^{p.k} = (\theta(r_j(p.k)))(\epsilon)$.

For each $p \in PosNonVar(r_j)$, we add into Δ the following transition (except if it is already in Δ) identified by (i, p):

- if $r_j(p) \in \Sigma_l$: $q_j^p(r_j(p)(x_1, \ldots, x_n), z) \rightarrow r_j(p)(q_j^{p.1}(x_1, z), \ldots, q_j^{p.n}(x_n, z))$
- if $r_j(p) \in \Sigma_c$:
$$q_j^p(r_j(p)(x_1, \ldots, x_n), z) \rightarrow r_j(p)(q_j^{p.1}(x_1, e_i^1(z)), \ldots, q_j^{p.n}(x_n, e_i^n(z)))$$

- if $r_j(p) \in \Sigma_r$ (let $e_i^k = head(stack(p))$) :
$$q_j^p(r_j(p)(x_1, \ldots, x_n), e_i^k(z)) \rightarrow r_j(p)(q_j^{p.1}(x_1, z), \ldots, q_j^{p.n}(x_n, z))$$
- however, if $r_j \in Var$, we add : $q(r_j, z) \rightarrow \theta r_j$ (identified by (i, ϵ)).

Note that states do not depend on the current overlap step i, whereas stack symbols do. States depend only on the rewrite rule number j.

Definition 13. *Let Q_{sat} denote the set of states, and Δ_{sat} denote the set of transitions, saturated by overlap steps (all overlap steps have been achieved).*

Example 4. Let $\Sigma_c = \{c, u\}$, $\Sigma_r = \{g, s, w\}$, $\Sigma_l = \{f, h, a\}$.

$$R = \left\{ \begin{array}{c} f \\ / \ \backslash \\ x \quad y \end{array} \xrightarrow{1} \begin{array}{c} c \\ | \\ f \\ / \ \backslash \\ \begin{array}{cc} g & h \\ | & | \\ x & s \\ & | \\ & y \end{array} \end{array}, \ \begin{array}{c} h \\ | \\ x \end{array} \xrightarrow{2} \begin{array}{c} u \\ | \\ h \\ | \\ w \\ | \\ x \end{array} \right\} \quad E = \left\{ \begin{array}{c} f \\ / \ \backslash \\ a \quad a \end{array} \right\} \text{ then } R^*(E) = \left\{ \begin{array}{c} c^n \\ | \\ f \\ / \quad \backslash \\ \begin{array}{cc} g^n & (u^i h\, w^i s)^n \\ | & | \\ a & a \end{array} \end{array} \ \middle| \ n \in \mathbb{N} \right\}$$

c^n means $c(c(\cdots(c$ where c occurs n times, and

$\begin{array}{c}(u^i h\, w^i s)^n \\ | \\ a\end{array}$ means $u^{i_1}(h(w^{i_1}(s(\cdots(u^{i_n}(h(w^{i_n}(s(a))))))))$ for any $i_1, \ldots, i_n \in \mathbb{N}$.

We start with the following transitions which recognize E:
$$(0, \epsilon): \ q_0^\epsilon(f(x, y), z) \rightarrow f(q_0^1(x, z), \ q_0^1(y, z))$$
$$(0, 1): \ q_0^1(a, z) \rightarrow a$$

The algorithm computes the overlaps steps and adds new transitions. Note that once the first overlap step for one particular rewrite rule has been processed, the other overlap steps for this rule add fewer transitions since some of the generated transitions already exist.

- Overlap step $(0, \epsilon) \oplus rule1$
$$(1, \epsilon): \ q_0^\epsilon(c(x), z) \rightarrow c(q_1^1(x, e_1^1(z)))$$
$$(1, 1): \ q_1^1(f(x, y), z) \rightarrow f(q_1^{1.1}(x, z), \ q_1^{1.2}(y, z))$$
$$(1, 1.1): \ q_1^{1.1}(g(x), e_1^1(z)) \rightarrow g(q_0^1(x, z))$$
$$(1, 1.2): \ q_1^{1.2}(h(x), z) \rightarrow h(q_1^{1.2.1}(x, z))$$
$$(1, 1.2.1): \ q_1^{1.2.1}(s(x), e_1^1(z)) \rightarrow s(q_0^1(x, z))$$

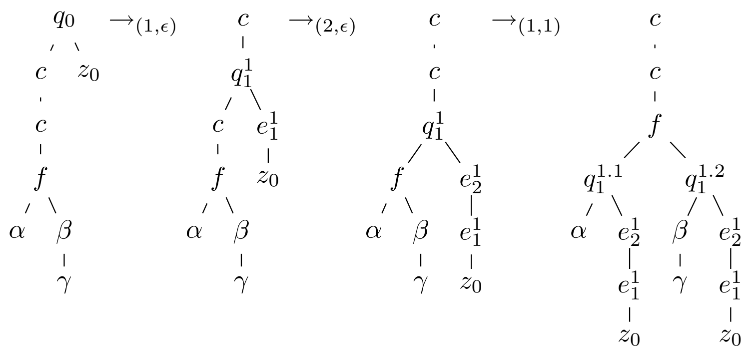
- Overlap step $(1, 1) \oplus rule1$
$$(2, \epsilon): \ q_1^1(c(x), z) \rightarrow c(q_1^1(x, e_2^1(z)))$$
$$(2, 1.1): \ q_1^{1.1}(g(x), e_2^1(z)) \rightarrow g(q_1^{1.1}(x, z))$$
$$(2, 1.2.1): \ q_1^{1.2.1}(s(x), e_2^1(z)) \rightarrow s(q_1^{1.2}(x, z))$$

- Overlap step $(1, 1.2) \oplus rule2$
$$(3, \epsilon): \ q_1^{1.2}(u(x), z) \rightarrow u(q_2^1(x, e_3^1(z)))$$
$$(3, 1): \ q_2^1(h(x), z) \rightarrow h(q_2^{1.1}(x, z))$$
$$(3, 1.1): \ q_2^{1.1}(w(x), e_3^1(z)) \rightarrow w(q_1^{1.2.1}(x, z))$$

- Overlap step $(3,1) \oplus rule2$

$$(4,\epsilon): \ q_2^1(u(x),z) \to u(q_2^1(x, e_4^1(z)))$$
$$(4,1.1): \ q_2^{1.1}(w(x), e_4^1(z)) \to w(q_2^{1.1}(x,z))$$

For readability, we omit parentheses for unary symbols. Consider the term $ccf(gga,\ uhwshsa)$, also denoted $ccf(\alpha,\ \beta\gamma)$ where $\alpha = gga$, $\beta = uhws$, and $\gamma = hsa$. This term is in $R^*(E)$, and is recognized in the following way:



Now, let us replace α, β, γ by their values:
$q_1^{1.1}(\alpha,\ e_2^1 e_1^1(z_0)) = q_1^{1.1}(gg(a),\ e_2^1 e_1^1(z_0)) \to_{(2,1.1)} g(q_1^{1.1}(g(a),\ e_1^1(z_0))) \to_{(1,1.1)} gg(q_0^1(a, z_0)) \to_{(0,1)} gg(a) = \alpha$
On the other hand,
$q_1^{1.2}(\beta\gamma,\ e_2^1 e_1^1(z_0)) = q_1^{1.2}(uhws\gamma,\ e_2^1 e_1^1(z_0)) \to_{(3,\epsilon)} u(q_2^1(hws\gamma,\ e_3^1 e_2^1 e_1^1(z_0))) \to_{(3,1)} uh(q_2^{1.1}(ws\gamma,\ e_3^1 e_2^1 e_1^1(z_0))) \to_{(3,1.1)} uhw(q_1^{1.2.1}(s\gamma,\ e_2^1 e_1^1(z_0))) \to_{(2,1.2.1)} uhws(q_1^{1.2}(\gamma,\ e_1^1(z_0))) = \beta(q_1^{1.2}(hs(a),\ e_1^1(z_0))) \to_{(1,1.2)} \beta h(q_1^{1.2.1}(s(a),\ e_1^1(z_0))) \to_{(1,1.2.1)} \beta hs(q_0^1(a,\ z_0)) \to_{(0,1)} \beta hsa = \beta\gamma$

Note that although a is local (then does not check the stack top), the stack is necessarily empty when a is recognized, which ensures that there are as many c occurrences as g occurrences. This is due to the fact that the transitions deal with the outermost c and the innermost g as particular cases.

A consequence of Example 4 is:

Proposition 14. *Regular languages are not closed under rewriting by linear VCF-TRS.*

3.2 Termination

Although each overlap step uses new stack symbols, the algorithm terminates. It comes from the fact that when computing an overlap step $(x,u) \oplus (l_j \to r_j)$, transition (x,u) necessarily deals with a reducible symbol of Σ, which is local.

Proposition 15. *Q_{sat} and Δ_{sat} are finite.*

Proof. Let $A_E = (\Sigma, Q_E, q_0, \Pi_E, z_0, \Delta_E)$ be the initial automaton, i.e. an automaton that recognizes the initial language E. ¿From the algorithm

$$Q_{sat} = Q_E \cup \{q_j^p \mid l_j \to r_j \in R,\ p \in PosNonVar(r_j) \backslash \{\epsilon\}\}$$

Then Q_{sat} is finite.

In the following, for any finite set S, $|S|$ will denote the number of elements of S. Let $f \in RED(R)$. Since $RED(R) \subseteq \Sigma_l$, every transition in Δ_{sat} that uses f is of the form :

$$s(f(x_1, \ldots, x_n), z) \to f(s_1(x_1, z), \ldots, s_n(x_n, z))$$

where $s, s_1, \ldots, s_n \in Q_{sat}$. Then the number of transitions in Δ_{sat} that uses a reducible symbol is less than or equal to $x = |RED(R)| \times |Q_{sat}|^{armax(\Sigma)+1}$ where $armax(\Sigma)$ is the maximum arity of symbols in Σ.
Then the number of overlap steps is less than or equal to $y = x \times |R|$. Therefore

$$|\Delta_{sat}| \leq |\Delta_E| + y \times Max(\{|PosNonVar(r_j)| \mid l_j \to r_j \in R\} \cup \{1\})$$

Note that $\cdots \cup \{1\}$ comes from the fact that in an overlap step, a collapsing rule generates one new empty transition, whereas the number of positions of its rhs is 0.

3.3 Completeness

Lemma 16. *Let $l \to r \in R$, $q \in Q_{sat}$, s be a stack, and σ be a substitution. If $q(l, s) \to_{\Delta_{sat}} \sigma l$, then $q(r, s) \to^*_{\Delta_{sat}} \sigma r$.*

Proof. According to the form of transitions, σ is necessarily of the form $\sigma = (x_1/q_1(x_1, s), \ldots, x_n/q_n(x_n, s))$.
By hypothesis $q(l, s) \to_{(i,j)} \sigma l$. Then Transition $(i, j) = (q(l, z) \to \theta l)$ where z is a variable and $\theta = (x_1/q_1(x_1, z), \ldots, x_n/q_n(x_n, z))$.
The overlap step $(i, j) \oplus (l \to r)$ has been computed within Δ_{sat}, then $q(r, z) \to^*_{\Delta_{sat}} \theta r$. By instantiating z by s, we get $q(r, s) \to^*_{\Delta_{sat}} \sigma r$.

Lemma 17. *Let $t \in T_\Sigma$ and $l \to r \in R$. Recall that q_0 is the initial state and z_0 is the initial stack.
If $q_0(t, z_0) \to^*_{\Delta_{sat}} t$ and $t \to_{[p,l\to r,\gamma]} t'$, then $q_0(t', z_0) \to^*_{\Delta_{sat}} t'$.*

Proof. ¿From the hypotheses, $t|_p = \gamma l$ and

$$q_0(t, z_0) \to^*_{\Delta_{sat}} t[p \leftarrow q(\gamma l, s)] \to_{\Delta_{sat}} t[p \leftarrow \theta l] \to^*_{\Delta_{sat}} t[p \leftarrow \gamma l] = t$$

where $\theta = (x_1/q_1(\gamma x_1, s), \ldots, x_n/q_n(\gamma x_n, s))$.
Then $q(l, s) \to_{\Delta_{sat}} \sigma l$ where $\sigma = (x_1/q_1(x_1, s), \ldots, x_n/q_n(x_n, s))$. ¿From the previous lemma, $q(r, s) \to^*_{\Delta_{sat}} \sigma r$, then $\gamma q(r, s) = q(\gamma r, s) \to^*_{\Delta_{sat}} \gamma\sigma r = \theta r$.
Therefore

$$q_0(t', z_0) = q_0(t[p \leftarrow \gamma r], z_0) \to^*_{\Delta_{sat}} t[p \leftarrow q(\gamma r, s)] \to^*_{\Delta_{sat}} t[p \leftarrow \theta r] \to^*_{\Delta_{sat}} t[p \leftarrow \gamma r] = t'$$

Proposition 18. *If t is recognized by the saturated automaton, and $t \to^*_R t'$, then t' is also recognized by the saturated automaton.*

Proof. From the previous lemma, and by induction on the length of $t \to^*_R t'$.

3.4 Soundness

Proving soundness is more difficult, and needs to take into account all the restrictions.

Definition 19. *A transition in Δ_{sat} identified by (i,p) is a* base-transition *if $i \neq 0$ and $p = \epsilon$.*
In other words, a base-transition is a transition created by an overlap step and that recognizes the top symbol $r(\epsilon)$ for some $l \to r \in R$.

Definition 20. *If the transition identified by (i',p') has been added into Δ_{sat} by the overlap step $(i,p) \oplus l \to r$, we write $parent(i',p') = (i,p) \oplus l \to r$.*

The following lemma means that if a base transition is used, then an instance of the right-hand-side of some rewrite rule is recognized.

Lemma 21. *If $q_0(t,z_0) \to^*_{\Delta_{sat}} \to_{[p,(x',y')]} t' \to^*_{\Delta_{sat}} t$ such that (x',y') is a base-transition (let $parent(x',y') = (x,y) \oplus l_k \to r_k$) and no base-transition is used in $t' \to^*_{\Delta_{sat}} t$, then there is a substitution σ such that $t|_p = \sigma(r_k)$.*

Proof. - If r_k is a variable, necessarily $\exists \sigma \mid t|_p = \sigma(r_k)$.

- if r_k is of the form $f(x_1,\ldots,x_n)$, rewriting by (x',y') generates $f(\ldots)$, then $t|_p(\epsilon) = f$. Consequently t_p is an instance of r_k.
- Otherwise, rewriting by (x',y') generates $r_k(\epsilon)(q_k^1(\ldots),\ldots,q_k^n(\ldots))$. According to the form of the transitions added by the algorithm, rewriting (in one step) any state q_j^u generates $r_j(u)$, except if a base-transition is used. Since no base-transition is used within $t' \to^*_{\Delta_{sat}} t$, then each $q_k^i(\ldots)$ generates an instance of $r_k|_i$, therefore the sub-derivation $\to_{[p,(x',y')]} t' \to^*_{\Delta_{sat}} t$ generates an instance of r_k. Thus $t|_p$ is an instance of r_k.

The following lemma means (loosely) that if the right-hand-side of some rewrite rule is recognized using a base-transition (which is a transition added by the algorithm), then the corresponding left-hand-side is also recognized. The idea is: if a right-hand-side is not recognized by the initial automaton, but is recognized using some transitions added by the algorithm, then it should be a descendant of a term (the corresponding lhs) recognized by older transitions.

Lemma 22. *Let $l \to r \in R$, and we write $Var(r) = \{x_1,\ldots,x_n\}$. Let s be a stack, $q, q_1,\ldots,q_n \in Q_{sat}$, $\sigma = (x_1/q_1(x_1,s),\ldots,x_n/q_n(x_n,s))$, and (i,ϵ) be a base-transition in Δ_{sat} such that $parent(i,\epsilon) = (j,u) \oplus l \to r$.*
If $q(r,s) \to_{[(i,\epsilon)]} t' \to^*_{\Delta_{sat}} \sigma r$ and no base-transition is used within $t' \to^*_{\Delta_{sat}} \sigma r$, then $q(l,s) \to_{[(j,u)]} \sigma l$ and $j < i$.

Proof. Since R is visibly context-free:

- If r is shallow or is a variable, then $r(\epsilon) \in Var \cup \Sigma_l$. So, Transition (i,ϵ) does not modify the stack, and variables of r occurs at depth at most 1. Therefore $t' = \sigma(r(\epsilon)(r|_1,\ldots,r|_k))$. ¿From the algorithm, $(j,u) = (q(l,z) \to \theta l)$ where $\theta = (x_1/q_1(x_1,z),\ldots,x_n/q_n(x_n,z))$, and $j < i$. By instantiating z by s we get $q(l,s) \to_{[(j,u)]} \sigma l$.

- Otherwise, $r(\epsilon) \in \Sigma_c$ and for every $p \in PosNonVar(r)$ located just above a variable, $r(p)$ is the corresponding return. Since each overlap step uses new stack symbols, rewrite steps within $t' \rightarrow^*_{\Delta_{sat}} \sigma r$ at such positions p use transitions also created by the overlap step $(j, u) \oplus l \rightarrow r$ (i.e. transitions identified by $(i, \ldots)$). ¿From the algorithm, $(j, u) = (q(l, z) \rightarrow \theta l)$ where $\theta = (x_1/q_1(x_1, z), \ldots, x_n/q_n(x_n, z))$, and $j < i$. By instantiating z by s we get $q(l, s) \rightarrow_{[(j,u)]} \sigma l$.

Definition 23. *Let t be a ground term. A* successful derivation *for t is a derivation D that allows to recognizes t, i.e. of the form*

$$D = \; q_0(t, z_0) \rightarrow_{[(i_1,p_1)]} \cdots \rightarrow_{[(i_k,p_k)]} t$$

The composed length of D is $|D| = i_1 + \cdots + i_k$.
The term recognized by D is $term(D) = t$.

Terms recognized by successful derivations are descendants indeed.

Proposition 24. *For every successful derivation D in the saturated automaton, $term(D) \in R^*(E)$.*

Proof. By strong induction on $|D|$.
If D uses only transitions of the initial automaton A_E (that recognizes E), the property trivially holds. Otherwise D uses at least one transition created by an overlap step. Consider the first step of D that uses a transition (say (i_l, p_l)) created by an overlap step :

$$D = \; q_0(t, z_0) \rightarrow^*_{\Delta_E} t_1 \rightarrow_{[(i_l,p_l)]} \rightarrow^*_{\Delta_{sat}} t$$

t_1 does not contains states added by the algorithm, i.e. states of the form q^p_j. But the transitions created by an overlap step that are not base-transitions are of the form $q^p_j(\ldots) \rightarrow \ldots$. Therefore (i_l, p_l) is necessarily a base-transition.

Thus D contains at least one base-transition. Now consider the last step of D that uses a base transition (say (i, p')) :

$$D = \; q_0(t, z_0) \rightarrow^*_{\Delta_{sat}} t'' \rightarrow_{[p,(i,p')]} t' \rightarrow^*_{\Delta_{sat}} t$$

No base-transition is used within $t' \rightarrow^*_{\Delta_{sat}} t$. Then we have the derivation :

$$E = \; t''|_p = q(t|_p, s) \rightarrow_{[\epsilon,(i,p')]} u \rightarrow^*_{\Delta_{sat}} t|_p$$

and no base-transition is used within $u \rightarrow^*_{\Delta_{sat}} t|_p$.
Let $parent(i, p') = (j, p'') \oplus l_k \rightarrow r_k$. Then $j < i$. ¿From Lemma 21, $\exists \sigma \mid t|_p = \sigma r_k$, then

$$E = \; q(\sigma r_k, s) \rightarrow_{[\epsilon,(i,p')]} u \rightarrow^*_{\Delta_{sat}} \sigma r_k$$

¿From the form of transitions, and since r_k is linear, we can deduce from E :

$$q(r_k, s) \rightarrow_{[\epsilon,(i,p')]} u \rightarrow^*_{\Delta_{sat}} \theta r_k \quad \text{and} \quad \sigma\theta r_k \rightarrow^*_{\Delta_{sat}} \sigma r_k$$

where $Var(r_k) = \{x_1, \ldots, x_n\}$ and $\theta = (x_1/q_1(x_1, s), \ldots, x_n/q_n(x_n, s))$, because the stack is not modified since $\forall p \in PosVar(r_k)$, $w(p) = 0$.
¿From Lemma 22, $q(l_k, s) \rightarrow_{[(j,p'')]} \theta l_k$. Then we have the derivation :

$$G = \; q(\sigma l_k, s) \rightarrow_{[(j,p'')]} \sigma\theta l_k \rightarrow^*_{\Delta_{sat}} \sigma l_k$$

$|G| < |E|$ since $j < i$.
Consider D again. $t''|_p = q(t|_p, s)$ and σl_k is a ground term without states. Therefore we have the derivation :

$$H = q_0(t[p \leftarrow \sigma l_k], z_0) \rightarrow^*_{\Delta_{sat}} t''[p \leftarrow q(\sigma l_k, s)] \rightarrow^*_{\Delta_{sat}} t[p \leftarrow \sigma l_k]$$

H is a successful derivation and $|H| < |D|$. By induction hypothesis $t[p \leftarrow \sigma l_k] \in R^*(E)$. Since $t[p \leftarrow \sigma l_k] \rightarrow_R t[p \leftarrow \sigma r_k] = t$, we have $t \in R^*(E)$.

4 Conclusion

A question arises : are visibly pushdown tree languages closed under complementation ? This property holds for visibly pushdown string languages [2]. When dealing with trees, it seems to hold as well. However, it is difficult to prove it, since our automata work top-down, and top-down automata cannot be determinized, unlike bottom-up automata. Moreover, the stack is duplicated by n-ary symbols, which is a major difference with respect to strings.

On the other hand, the algorithm for computing descendants could be extended to allow reducible symbols that are call or return.

Thanks to expressivity and properties, we hope that visibly pushdown tree languages, as well as visibly context-free rewrite systems, could be used in other domains as decision procedures.

References

1. R. Alur, S. Chaudhuri, and P. Madhusudan. Languages of Nested Trees. In *Proc. 18th conference on Computer-Aided Verification (CAV'06)*, LNCS. Springer, 2006.
2. R. Alur and P. Madhusudan. Visibly Pushdown Languages. In *Proc. 36th ACM Symposium on Theory of Computing (STOC'04)*, pages 202–211. ACM Press, 2004.
3. H. Comon, M. Dauchet, R. Gilleron, D. Lugiez, S. Tison, and M. Tommasi. *Tree Automata Techniques and Applications* (TATA). http://www.grappa.univ-lille3.fr/tata.
4. 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).
5. V. Gouranton, P. Réty, and H. Seidl. Synchronized Tree Languages Revisited and New Applications. In *Proceedings of 6th Conference on Foundations of Software Science and Computation Structures, Genova (Italy)*, LNCS. Springer, 2001.
6. Irène Guessarian. Pushdown Tree Automata. *Mathematical Systems Theory*, 4(16):237–263, 1983.
7. M. Hermann and R. Galbavý. Unification of Infinite Sets of Terms Schematized by Primal Grammars. *Theoretical Computer Science*, 176, 1997.
8. 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.
9. S. Limet and P. Réty. A New Result about the Decidability of the Existential One-Step Rewriting Theory. In *Proceedings of 10th Conference on Rewriting Techniques and Applications, Trento (Italy)*, number 1631 in LNCS. Springer, 1999.

10. S. Limet and G. Salzer. Proving properties of term rewrite systems via logic programs. In *proceedings of RTA 2004*, volume 3091 of *LNCS*, pages 170–184. Springer Verlag, 2004.
11. Sébastien Limet and Gernot Salzer. Manipulating tree tuple languages by transforming logic programs. In Ingo Dahn and Laurent Vigneron, editors, *Electronic Notes in Theoretical Computer Science*, volume 86. Elsevier, 2003.
12. D. Lugiez and P. Schnoebelen. The Regular Viewpoint on PA-Processes. *Theoretical Computer Science*, 2000.
13. T. Nagaya and Y. Toyama. Decidability for Left-Linear Growing Term Rewriting Systems. In *Proceedings of 10th Conference on Rewriting Techniques and Applications (RTA'99), Trento (Italy)*, number 1631 in LNCS. Springer, 1999.
14. J.C. Raoult. Rational Tree Relations. *Bulletin of the Belgian Mathematical Society Simon Stevin*, 4:149–176, 1997.