HAL open science

Rigid Tree Automata and Applications

Florent Jacquemard, Francis Klay, Camille Vacher

To cite this version:

Florent Jacquemard, Francis Klay, Camille Vacher. Rigid Tree Automata and Applications. Information and Computation, 2011, 209 (3), pp.486-512. 10.1016/j.ic.2010.11.015 . inria-00578820

HAL Id: inria-00578820

https://inria.hal.science/inria-00578820v1

Submitted on 23 Mar 2011

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Rigid Tree Automata and Applications☆

Florent Jacquemard[a,b], Francis Klay[c], Camille Vacher[d,b]

[a]*INRIA Saclay - Île de France*
[b]*LSV (CNRS/ENS Cachan)*
[c]*FT/RD/MAPS/AMS/SLE*
[d]*FT/RD*

Abstract

We introduce the class of Rigid Tree Automata (RTA), an extension of standard bottom-up automata on ranked trees with distinguished states called rigid. Rigid states define a restriction on the computation of RTA on trees: RTA can test for equality in subtrees reaching the same rigid state. RTA are able to perform local and global tests of equality between subtrees, non-linear tree pattern matching, and some inequality and disequality tests as well. Properties like determinism, pumping lemma, Boolean closure, and several decision problems are studied in detail. In particular, the emptiness problem is shown decidable in linear time for RTA whereas membership of a given tree to the language of a given RTA is NP-complete. Our main result is the decidability of whether a given tree belongs to the rewrite closure of an RTA language under a restricted family of term rewriting systems, whereas this closure is not an RTA language. This result, one of the first on rewrite closure of languages of tree automata with constraints, is enabling the extension of model checking procedures based on finite tree automata techniques, in particular for the verification of communicating processes with several local non rewritable memories, like security protocols. Finally, a comparison of RTA with several classes of tree automata with local and global equality tests, with dag automata and Horn clause formalisms is also provided.

Keywords: Tree Automata, Symbol Constraints, Term Rewriting, Verification

Introduction

Tree automata (TA) are finite representations of infinite sets of terms. In automated theorem proving, they allow to cut infinite computation branches

☆This work has been partly supported by the FET-Open grant agreement FOX no. FP7-ICT-23359, the INRIA ARC 2010 project ACCESS, and the ANR Sesur 07 project AVOTÉ.

Email addresses: `florent.jacquemard@inria.fr` (Florent Jacquemard), `francis.klay@orange-ftgroup.com` (Francis Klay), `vacher@lsv.ens-cachan.fr` (Camille Vacher)

by reduction to TA decision problems. In system and software verification, TA can be used to represent infinite sets of states of a system or a program (in the latter case, a term can represent the program itself), messages exchanged in a communication protocol, XML documents... In these settings, the closure properties of TA languages permit incremental constructions and verification problems can be reduced to TA problems decidable in polynomial time like emptiness (is the language recognized by a given TA empty) and membership (is a given term t recognized by a given TA).

Despite these nice properties, a big limitation of TA is their inability to test equalities between subterms during their computation: TA are able to detect linear patterns like $\mathsf{fst}(\mathsf{pair}(x_1, x_2))$ but not a pattern like $\mathsf{pair}(x, x)$. Several extensions of TA have been proposed to overcome this problem, by addition of equality and disequality tests in TA transition rules (the classes [1, 2] have a decidable emptiness problem), or an auxiliary memory containing a tree and memory comparison [3]. Pushdown tree automata [4, 5] also permit such tests. However, they are all limited to local tests, at a bounded distance from the current position.

In this paper, we define the *rigid tree automata* (RTA) by the distinction of some states as *rigid*, and the condition that the subterms recognized in one rigid state during a computation are all equal. With such a formalism, it is possible to check local and global equality tests between subterms, and also the subterm relation or restricted disequalities. In Sections 2 to 6 we study issues like pattern matching, pumping lemmas, compare expressiveness with related classes of automata, determinism, closure of recognized languages under Boolean operations, and decision problems for RTA. RTA are a particular case of the more general class Tree Automata with General Equality and Disequality constraints (TAGED [6], see Section 3.1). The study of the class RTA alone is motivated by the complexity results and specific applications to verification mentioned below. But our most original contribution is the study of the rewrite closure of RTA languages in Section 7.

Term rewriting systems (TRS) is a general formalism for the symbolic evaluation of terms by replacement of some patterns by others, following rewrite rules. Combining tree automata and term rewriting techniques has been very successful in verification, see e.g. [7, 8]. In this context, term rewriting systems (TRS) can describe the transitions of a system, the evaluation of a program [7], the specification of operators used to build protocol messages [9] or also transformation of documents. If a TA $\mathcal{A}$ is used to finitely represent an infinite set $L(\mathcal{A})$ of states of a system, the rewrite closure $\mathcal{R}^*\big(L(\mathcal{A})\big)$ of the language $L(\mathcal{A})$ using $\mathcal{R}$ represents the set of states reachable from states described by $\mathcal{A}$. When $\mathcal{R}^*\big(L(\mathcal{A})\big)$ is again a TA language, the verification of a safety property amounts to checking for the existence of an error state in $\mathcal{R}^*\big(L(\mathcal{A})\big)$ (either a given term t or a term in a given regular language). This technique, sometimes referred as regular tree model checking, has driven a lot of attention to the rewrite closure of tree automata languages. However, there has been very few studies of this issue for constrained TA (see e.g. [10]). The reason is the difficulty to capture

the behaviour of constraints after the application of rewrite rules.

In Section 7, we show that it is decidable whether a given term t belongs to the rewrite closure of a given RTA language for a restricted class of linear TRS called invisibly pushdown, whereas this closure is generally not an RTA language. Linear and invisibly pushdown TRS can typically specify cryptographic operators like $\mathsf{decrypt}(\mathsf{crypt}(x, \mathsf{pk}(A)), \mathsf{sk}(A)) \to x$.

Using RTA instead of TA in a regular tree model checking procedure permits to handle processes with local and global memories taking their values in infinite domains and which can be written only once. We illustrate this idea in Section 8 with the description of a potential application of RTA to the verification of security protocols.

1. Preliminaries

A *signature* Σ is a finite set of function symbols with arity. We write Σ_m for the subset of function symbols of Σ of arity m. Given an infinite set $\mathcal{X}$ of variables, the set of terms built over Σ and $\mathcal{X}$ is denoted $\mathcal{T}(\Sigma, \mathcal{X})$, and the subset of ground terms (terms without variables) is denoted $\mathcal{T}(\Sigma)$. The set of variables occurring in a term $t \in \mathcal{T}(\Sigma, \mathcal{X})$ is denoted $vars(t)$. A term $t \in \mathcal{T}(\Sigma, \mathcal{X})$ is called *linear* if every variable of $vars(t)$ occurs at most once in t. A *substitution* σ is a mapping from a finite subset of $\mathcal{X}$ into $\mathcal{T}(\Sigma, \mathcal{X})$. The application of a substitution σ to a term t is the homomorphic extension of σ to $\mathcal{T}(\Sigma, \mathcal{X})$.

A term t can be seen as a function from its set of *positions* $\mathcal{P}os(t)$ into function symbols or variables of $\Sigma \cup \mathcal{X}$. The positions of $\mathcal{P}os(t)$ are sequences of positive integers (ε, the empty sequence, is the root position). Positions are compared wrt the prefix ordering: $p_1 < p_2$ iff there exists $p \neq \varepsilon$ such that $p_2 = p_1 \cdot p$ (where $p_1 \cdot p$ denotes the concatenation of p_1 and p). In this case, p is denoted $p_2 - p_1$. The subterm of t at position p is denoted $t|_p$, and the replacement in t of the subterm at position p by u is denoted $t[u]_p$. The *depth* $d(t)$ of t is the length of its longest position. A *n-context* is a linear term of $\mathcal{T}(\Sigma, \{x_1, \ldots, x_n\})$. The application of a *n-context* C to n terms $t_1, \ldots, t_n$, denoted by $C[t_1, \ldots, t_n]$, is defined as the application to C of the substitution $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$.

1.0.1. Term Rewriting.

A *term rewrite system* (TRS) over a signature Σ is a finite set of rewrite rules $\ell \to r$, where $\ell \in \mathcal{T}(\Sigma, \mathcal{X})$ (it is called the left-hand side (*lhs*) of the rule) and $r \in \mathcal{T}(\Sigma, vars(\ell))$ (it is called right-hand-side (*rhs*)). A term $t \in \mathcal{T}(\Sigma, \mathcal{X})$ rewrites to $s \in \mathcal{T}(\Sigma, \mathcal{X})$ by a TRS $\mathcal{R}$ (denoted $t \xrightarrow[\mathcal{R}]{} s$) if there is a rewrite rule $\ell \to r \in \mathcal{R}$, a position $p \in \mathcal{P}os(t)$ and a substitution σ such that $t|_p = \sigma(\ell)$ and $s = t[\sigma(r)]_p$. In this case, t is called *reducible*. An irreducible term is also called an *$\mathcal{R}$-normal-form*. The transitive and reflexive closure of $\xrightarrow[\mathcal{R}]{}$ is denoted $\xrightarrow[\mathcal{R}]{*}$. Given $L \subseteq \mathcal{T}(\Sigma, \mathcal{X})$, we denote $R^*(L) = \{t \mid \exists s \in L, s \xrightarrow[R]{*} t\}$. A TRS $\mathcal{R}$ is called *linear* if all the terms in its rules are linear and *collapsing* if every rhs of rules of $\mathcal{R}$ is a variable.

1.0.2. Tree automata.

Following definitions and notations of [11], we consider tree automata which compute bottom-up (from leaves to root) on (finite) ground terms in $\mathcal{T}(\Sigma)$. At each stage of computation on a tree t, a tree automaton reads the function symbol f at the current position p in t and updates its current state, according to f and the respective states reached at the positions immediately under p in t.

Definition 1. A *tree automaton* (TA) $\mathcal{A}$ on a signature Σ is a tuple $\langle Q, F, \Delta\rangle$ where Q is a finite set of nullary state symbols, disjoint from Σ, $F \subseteq Q$ is the subset of final states and Δ is a set of transition rules of the form: $f(q_1, \ldots, q_n) \to q$ where $n \geq 0$, $f \in \Sigma_n$, and $q_1, \ldots, q_n, q \in Q$.

A *run* of the TA $\mathcal{A}$ on a term $t \in \mathcal{T}(\Sigma)$ is a relabelling of t with states of Q compatible with Δ. More formally, it is a function $r : \mathcal{P}os(t) \to Q$ such that for all $p \in \mathcal{P}os(t)$ with $t(p) = f \in \Sigma_n$ $(n \geq 0)$, $f\big(r(p.1), \ldots, r(p.n)\big) \to r(p) \in \Delta$. A run r is called *successful* if $r(\varepsilon) \in F$. We will sometimes use term-like notation for runs. For instance, a run $\{\varepsilon \mapsto q, 1 \mapsto q_1, 2 \mapsto q_2\}$ will be denoted $q(q_1, q_2)$.

The *language* $L(\mathcal{A}, q)$ of a TA $\mathcal{A}$ in state q is the set of ground terms for which there exists a run r of $\mathcal{A}$ such that $r(\varepsilon) = q$. The language $L(\mathcal{A})$ of $\mathcal{A}$ is $\bigcup_{q\in F} L(\mathcal{A}, q)$, and a set of ground terms is called *regular* if it is the language of a TA. The *size* of $\mathcal{A}$, denoted $|\mathcal{A}|$, is the number of symbols in Δ.

A TA $\mathcal{A} = \langle Q, F, \Delta\rangle$ on Σ is *deterministic* (DTA), resp. *complete*, if for every $f \in \Sigma_n$, and every $q_1, \ldots, q_n \in Q$, there exists at most, resp. at least, one rule $f(q_1, \ldots, q_n) \to q \in \Delta$. In the deterministic (resp. complete) cases, given a tree t, there is at most (resp. at least) one run r of $\mathcal{A}$ on t.

2. RTA: Definition and Examples

We now introduce the class of rigid tree automata studied in this paper and show their expressiveness with some examples and first properties.

2.1. Definition and First Examples

Definition 2. A *rigid tree automaton* (RTA) $\mathcal{A}$ on a signature Σ is a tuple $\langle Q, R, F, \Delta\rangle$ where $\langle Q, F, \Delta\rangle$ is a tree automaton denoted $ta(\mathcal{A})$ and $R \subseteq Q$ is the subset of *rigid* states.

A *run* of the RTA $\mathcal{A}$ on a term $t \in \mathcal{T}(\Sigma)$ is a run r of the underlying TA $ta(\mathcal{A})$ on t with the additional condition (called *rigidity condition*) stating that: for all positions $p_1, p_2 \in \mathcal{P}os(t)$, if $r(p_1) = r(p_2) \in R$ then $t|_{p_1} = t|_{p_2}$.

The languages $L(\mathcal{A}, q)$ and $L(\mathcal{A})$ of RTA are defined the same way as for TA. Note that with these definitions, every regular language is an RTA language. We shall sometimes write below TA and RTA for the classes of TA and RTA languages. The size of an RTA $\mathcal{A}$, denoted $|\mathcal{A}|$, is the size $|ta(\mathcal{A})|$ of its underlying TA.

Example 1. Let $\Sigma = \{a{:}0, b{:}0, f{:}2\}$. The set $\{f(t,t) \mid t \in \mathcal{T}(\Sigma)\}$ is recognized by the following RTA on Σ

$$\mathcal{A} = \big\langle \{q, q_r, q_\mathsf{f}\}, \{q_r\}, \{q_\mathsf{f}\}, \{a \to q|q_r, b \to q|q_r, f(q,q) \to q|q_r, f(q_r,q_r) \to q_\mathsf{f}\}\big\rangle,$$

where $a \to q|q_r$ is an abbreviation for $a \to q$, $a \to q_r$.
An example of successful run of $\mathcal{A}$ on $f(f(a,b), f(a,b))$ is $q_\mathsf{f}\big(q_r(q,q), q_r(q,q)\big)$.$\diamond$

Note that the above RTA language is not regular; this can be shown using a classical *pumping* argument.

2.2. Pattern Matching

TA are able to characterize languages of terms which embed a given pattern. However, they are limited to linear patterns for this purpose. For instance, as recalled above, the set of terms embedding the pattern $f(x,x)$ is not a regular term language. The RTA permit to generalize this pattern matching ability to arbitrary patterns.

Example 2. Let us extend the RTA of Example 1 with the transitions rules $f(q, q_\mathsf{f}) \to q_\mathsf{f}$, $f(q_\mathsf{f}, q) \to q_\mathsf{f}$ ensuring the propagation of the final state q_f up to the root. The RTA obtained recognizes the set of terms of $\mathcal{T}(\Sigma)$ containing the pattern $f(x,x)$. $\diamond$

The principle of the construction of Examples 1 and 2 can be generalized into the following result.

Proposition 1. *For all terms $t \in \mathcal{T}(\Sigma, \mathcal{X})$, there exists an RTA of size linear in the size of t and constructed in linear time which recognizes the terms of $\mathcal{T}(\Sigma)$ having a ground instance of t as a subterm.*

Proof. The proposition is obvious when t is a variable. Let us assume that t is not a variable and let us associate one state q_s to every strict subterm s of t (including variables). The RTA for Proposition 1 has for set of states $Q = \{q_s \mid s \text{ strict subterm of } t\} \cup \{q, q_\mathsf{f}\}$, the subset R of rigid states is the set of states of Q of the form q_x such that x is a variable occurring at least twice in t, the subset of final states is $F = \{q_\mathsf{f}\}$, and its transition set is

$$\begin{array}{lll}
\Delta & = & \{f(q,\ldots,q) \to q \mid f \in \Sigma_n, n \geq 0\} \\
& \cup & \{f(q,\ldots,q) \to q_x \mid f \in \Sigma_n, n \geq 0, x \in \mathit{vars}(t)\} \\
& \cup & \{f(q_{s_1},\ldots,q_{s_n}) \to q_{f(s_1,\ldots s_n)} \mid f \in \Sigma_n, f(s_1,\ldots s_n) \text{ strict subterm of } t\} \\
& \cup & \{f(q_{s_1},\ldots,q_{s_n}) \to q_\mathsf{f} \mid f(s_1,\ldots s_n) = t\} \\
& \cup & \{f(q_1,\ldots,q_n) \to q_\mathsf{f} \mid f \in \Sigma_n, \exists i \leq n, q_i = q_\mathsf{f}\}.
\end{array}$$

The transitions in the first four lines ensure the recognition of the pattern t into the final state q_f and the transitions in the last line ensure the propagation of q_f. The choice of rigid states ensure that the non linearities in t are respected.$\square$

2.3. Other Examples: Disequalities, Inequalities, Global Equalities

In the above examples, the equalities tested by the RTA's are local, relatively to a position (e.g. the RTA tests that the subterms at the left and right child of some position p are equal). However, it is also possible to test equalities between subterms at arbitrary positions in a term.

Example 3. Let $\Sigma = \{a{:}0, g{:}1, f{:}2\}$. The set of terms $t \in \mathcal{T}(\Sigma)$ such that $s_1 = s_2$ for every two subterms $g(s_1)$, $g(s_2)$ of t is recognizable by the following RTA: $\mathcal{A} = \big\langle \{q, q_r\}, \{q_r\}, \{q, q_r\}, \big\{a \to q, g(q') \to q_r, f(q', q') \to q \mid q' \in \{q, q_r\}\big\}\big\rangle$. $\diamond$

RTA are not limited to testing equalities. Using rigid states also permits to test some disequality and inequality as well, like the subterm relation.

Example 4. Let $\Sigma = \{a : 0, b : 0, f : 2, < : 2\}$. The set of terms $<(s, t)$ such that $s, t \in \mathcal{T}\big(\Sigma \setminus \{<\}\big)$ and s is a strict subterm of t is recognized by the following RTA on Σ, $\langle \{q, q_r, q', q_{\mathsf{f}}\}, \{q_r\}, \{q_{\mathsf{f}}\}, \Delta\rangle$, with

$$\Delta = \left\{ \begin{array}{rcl@{\qquad}rcl@{\qquad}rcl} a & \to & q|q_r, & b & \to & q|q_r, & & & \\ f(q,q) & \to & q|q_r, & f(q,q_r) & \to & q', & f(q_r,q) & \to & q', \\ f(q,q') & \to & q', & f(q',q) & \to & q', & <(q_r,q') & \to & q_{\mathsf{f}} \end{array} \right\}.$$

For instance, a successful run on $<\big(a, f(a, b)\big)$ is $q_{\mathsf{f}}\big(q_r, q'(q_r, q)\big)$. The idea is that in a successful run, the rigid state q_r identifies (by a non-deterministic choice) the subterm s on the left side of $<$, and, on the right side t of $<$, the state q' is reached immediately above q_r and propagated up to the root, in order to ensure that t is a strict superterm of s. $\diamond$

The RTA can also test disequalities between subterms built only with unary and constant symbols.

Example 5. Let $\Sigma = \{c : 0, a : 1, b : 1, \neq : 2\}$. The set of terms of $\mathcal{T}(\Sigma)$ of the form $\neq(s, t)$, where $s, t \in \mathcal{T}\big(\Sigma \setminus \{\neq\}\big)$ and s is distinct from t is recognized by the following RTA on Σ, $\langle \{q, q_r, q_a, q_b, q_{\mathsf{f}}\}, \{q_r\}, \{q_{\mathsf{f}}\}, \Delta\rangle$, with

$$\begin{array}{rcl} \Delta & = & \{c \to q|q_r, a(q) \to q|q_r, b(q) \to q|q_r, a(q_r) \to q_a, b(q_r) \to q_b\} \\ & \cup & \big\{a(q_x) \to q_x, b(q_x) \to q_x \mid q_x \in \{q_a, q_b\}\big\} \\ & \cup & \big\{\neq(q_1, q_2) \to q_{\mathsf{f}} \mid q_1, q_2 \in \{q_a, q_b, q_r\}, q_1 \neq q_2\big\}. \end{array}$$

A successful run on $\neq\big(a(a(c)), b(a(c))\big)$ is $q_{\mathsf{f}}\big(q_a(q_r(q)), q_b(q_r(q))\big)$. The rigid state q_r will be placed at the position of the largest common postfix of s and t and q_a or q_b are used to memorize the letters immediately above this position, in order to check that s and t differ when reaching the top symbol $\neq$ in $\neq(s, t)$. $\diamond$

The construction of Example 4 cannot be generalized to the characterization of a maximal subterm amongst some subterms. This is shown in the following counter example, using a pumping argument.

Example 6. Let $\Sigma = \{0 : 0, g : 1, h : 2\}$, and let $L_{\max}$ be the set of terms of the form $H\big[g^m(0), g^{n_1}(0), \ldots, g^{n_k}(0)\big]$ where k is an arbitrary positive integer, $m \geq n_1, \ldots, n_k$, H is an $k + 1$-context made of the symbol h only, and g^n represents n nested symbols g.

Fact 1. *$L_{\max}$ is not an RTA language.*

Proof. Assume that $L_{\max}$ is recognized by an RTA $\mathcal{A}$ with n states and d rigid states. We can assume wlog that $d < n$. Let $t \in L_{\max}$ be of the form $H[t_0, \ldots, t_{d+1}]$ where for each $0 \leq i \leq d+1$, $t_i = g^{(d+2-i)(n+1)}(0)$. Let r be a run of $\mathcal{A}$ on t. We show, by a pumping argument, that for one $i \geq 1$, we can increase as much as we want the number of g's in t_i, while keeping the term recognized by $\mathcal{A}$ (a contradiction).

First, note that the t_i's are pairwise distinct. It follows that there are no rigid states in r at the positions of the symbols h in t, except rigid states which occur only once in r (such rigid states are not affected by a modification of some t_i). Second, a rigid state of $\mathcal{A}$ cannot occur twice in some t_i. By a pigeonhole principle, it follows that there exists some $i > 0$ such that the $n+1$ smaller (wrt prefix ordering) positions of t_i are not labelled by a rigid state in r. Hence, there exists one non-rigid state of $\mathcal{A}$ labelling two of these $n+1$ positions. Let k be the distance between these two positions. For all $j \geq 0$, we can build from r a successful run of $\mathcal{A}$ on $t'_j := H[t_0, \ldots, t_{i-1}, g^{jk}(t_i), t_{i+1}, \ldots, t_{d+1}]$. But for a j sufficiently large, $t'_j \notin L_{\max}$, a contradiction. $\square$ $\diamond$

2.4. Pumping Lemma

Following some ideas developed in the proof of Fact 1 above, we propose a weak form, adapted to RTA, of the pumping (or iteration) lemma for TA. Pumping on runs of RTA is not as easy as for standard TA. Indeed, we must take care of the position of rigid states in order to preserve recognizability. For this reason, the transformation of a subterm must be performed in several branches in parallel (instead of one single branch for TA) in order to preserve the rigidity condition. Moreover, we cannot repeat a term containing a rigid state, because the same rigid state cannot label two different positions on the same branch.

Lemma 1. *For all RTA $\mathcal{A} = \langle Q, R, F, \Delta \rangle$, for all terms $t \in L(\mathcal{A})$ such that $d(t) > (|Q|+1)|R|$, there exist a context C, two 1-contexts C' and D, with D non-trivial (non-variable), and a term u such that $t = C\big[C'[D[u]], \ldots, C'[D[u]]\big]$ and for all $n \geq 0$, $C\big[C'[D^n[u]], \ldots, C'[D^n[u]]\big] \in L(\mathcal{A})$.*

Proof. Let $t \in L(\mathcal{A})$ be such that $d(t) > (|Q|+1)|R|$, let r be a successful run of $\mathcal{A}$ on t, and let p be a position in $\mathcal{P}os(t)$ of length at least $(|Q|+1)|R|$.

With the rigidity condition in the definition of successful runs, a rigid state can occur at most once on a path of r. Hence, there exist two positions $p_0 < p'_0 < p$ such that $|p'_0| - |p_0| > |Q|$ and no rigid state occurs between p_0 and p'_0 in r. By a pigeon-hole principle, there exist two positions p_1, p_2 with $p_0 \leq p_1 < p_2 \leq p'_0$ labeled with the same state of $Q \setminus R$ in r. We let $u := t|_{p_2}$ and $D = (t|_{p_1})[x_1]_{p_2 - p_1}$. This situation is depicted in Figure 1.

In order to preserve the property of being a run while iterating D, we need to take care of rigid states above p_0 in r (rigid states below p'_0 and below D are not affected by iteration of D). Let π_1 be the maximal position of a rigid state in r smaller than p_0 wrt the prefix ordering. Let $q_r = r(\pi_1)$ and let $\pi_2, \ldots, \pi_k$

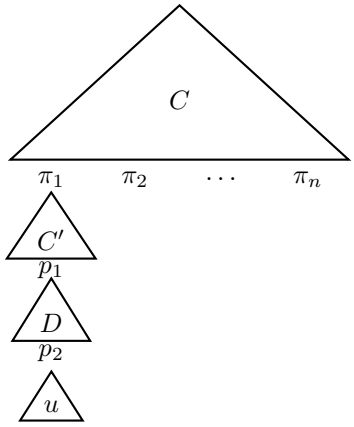

Figure 1: Pumping lemma

be the other positions of q_r in r. Note that by definition of r being a run, the positions $\pi_1, \ldots, \pi_k$ are pairwise incomparable wrt the prefix ordering. We let $C = t[x_1]_{\pi_1} \ldots [x_k]_{\pi_k}$ and $C' = (t|_{\pi_1})[x_1]_{p_1 - \pi_1}$ ($x_1, \ldots, x_k$ are distinct variables). Since $r(p_1) = r(p_2)$ and there are no rigid states between p_1 and p_2, we can construct a run on every $C'[D^n[u]]$. Moreover, $t|_{\pi_i} = t|_{\pi_j}$ for all $i, j \in \{1..k\}$, hence we may assume wlog that the subruns $r|_{\pi_i}$ are equal for all $i \in \{1..k\}$. It follows that we can perform the same operation as in $C'[D^n[u]]$ under each $r|_{\pi_i}$, and that $C\big[C'[D^n[u]], \ldots, C'[D^n[u]]\big] \in L(\mathcal{A})$. □

As usual, such a lemma can be used to show that a language is not in RTA.

Example 7. As a consequence of the above pumping lemma, we can show that the set $\mathcal{B}$ of balanced binary trees built over the signature $\{a : 0, f : 2\}$ is not an RTA language. Assume indeed that it is recognized by an RTA $\mathcal{A} = \langle Q, R, F, \Delta \rangle$ and let $t \in L(\mathcal{A})$ such that $d(t) > (|Q|+1)|R|$ and C, C', D, u be as in Lemma 1. By hypothesis, $C'[D[u]]$ is balanced, but for any $n > 1$, $C'[D^n[u]]$ is not balanced since C' and D are not trivial. It contradicts the fact that $C\big[C'[D^n[u]], \ldots, C'[D^n[u]]\big] \in L(\mathcal{A})$ by Lemma 1. ◇

3. Related Classes of Tree Automata

We shall present below some other classes of automata strictly more expressive than the standard TA, and compare their expressiveness to RTA. The decidability and complexity results presented in Section 6 and summarized in Table 1 also offer a base of comparison.

3.1. TAGED

Tree Automata with General Equality and Disequality constraints [6] were introduced in the context of spatial logics for XML querying [12]. They are defined, like RTA, by an underlying TA, but instead of having simply a set of

rigid state for testing equality, they have two binary relations on states: $R_=$ for testing equalities and $R_\neq$ for disequalities. More precisely, a run r of a TAGED on a term t is a run of the underlying TA on t with the additional condition that for all $p_1, p_2 \in \mathcal{P}os(t)$, if $\langle r(p_1), r(p_2)\rangle \in R_=$ then $t|_{p_1} = t|_{p_2}$ and if $\langle r(p_1), r(p_2)\rangle \in R_\neq$ then $t|_{p_1} \neq t|_{p_2}$.

TAGED are strictly more general than RTA. The emptiness problem has been shown decidable for the class of negative TAGED (such that $R_= = \emptyset$) [6], and a subclass of TAGED where the number of disequalities tested on every path is bounded [12]. More recently, emptiness was shown decidable for a class of TA with constraints strictly larger than TAGED [13].

The fragment of positive TAGED (with $R_\neq = \emptyset$, denoted TAGED+) has the same expressiveness as RTA. This is shown in [6] where a construction is proposed for transforming any TAGED+ into an RTA (i.e. a TAGED with a reflexive relation $R_=$) recognizing the same language, at the price of an exponential blowup. The transformation of [6] was originally proposed in order to show the decidability of emptiness for TAGED+. This result is reused in the proof of Theorem 1 below.

Proposition 2 ([6]). *For all TAGED+ $\mathcal{A}$, there exists an RTA $\mathcal{A}'$ of size exponential in the size of $\mathcal{A}$, constructed in exponential time, and such that $L(\mathcal{A}') = L(\mathcal{A})$.*

The emptiness problem is EXPTIME-complete for TAGED+, and PTIME for RTA (see Section 6). To our knowledge, the rewrite closure of TAGED has not been studied so far.

3.2. TA with Equality Constraints

TA with equality constraints (TAC) are TA whose transitions can perform local equality and disequality tests on the subterms of the term in input (see e.g. [1, 2]). More precisely, a TAC $\langle Q, F, \Delta\rangle$ is defined by a finite set of states Q, a subset of final states F and a set Δ of transitions of the form $f(q_1, \ldots, q_n) \xrightarrow{c} q$ where $f \in \Sigma_n$, $q_1, \ldots, q_n, q \in Q$, and c is a conjunction of constraints of the form $\pi = \pi'$ or $\pi \neq \pi'$ where π and π' are positions (sequences of positive integers). A run of a TAC on a term t is a function $r : \mathcal{P}os(t) \to Q$ such that for all $p \in \mathcal{P}os(t)$ with $t(p) = f \in \Sigma_n$ $(n \geq 0)$, there exists a transition $f\big(r(p \cdot 1), \ldots, r(p \cdot n)\big) \xrightarrow{c} r(p) \in \Delta$ such that for all constraints $\pi = \pi'$ (resp. $\pi \neq \pi'$) in c, we have $p \cdot \pi, p \cdot \pi' \in \mathcal{P}os(t)$ and $t|_{p\cdot\pi} = t|_{p\cdot\pi'}$ (resp. $t|_{p\cdot\pi} \neq t|_{p\cdot\pi'}$). A TAC is called positive if all its transitions contain only equalities. The class of positive TAC is called TAC+. Note that the RTA language of Examples 1 and 2 are recognizable by TAC+:

$$\mathcal{A} = \big\langle \{q, q_\mathsf{f}\}, \{q_\mathsf{f}\}, \{a \to q, b \to q, f(q, q) \to q, f(q, q) \xrightarrow{1=2} q_\mathsf{f}\}\big\rangle$$

for Example 1, and the same extended with the transitions $f(q, q_\mathsf{f}) \to q_\mathsf{f}$, $f(q_\mathsf{f}, q) \to q_\mathsf{f}$ for Example 2.

The emptiness problem is undecidable in general [14] for TAC+. Two decidable subclasses of TAC have been identified: tree automata with equality

and disequality tests between brother positions [1] (BTTA) and Reduction Automata [2] (RA); the complexity of emptiness is at least EXPTIME for these subclasses.

The equality tests of TAC are performed locally, but can involve an unbounded number of subterms (yielding undecidability of emptiness). In contrast, the equality tests of RTA can be global, but can involve only a bounded number of subterms (the bound is the number of rigid states). Hence it is not surprising that the languages of these two classes of automata are incomparable.

Proposition 3. *The classes of languages of TAC+ and RTA are orthogonal.*

Proof. The RTA languages of Examples 3 and 4 are not recognizable by a TAC. The language $\mathcal{B}$ of Example 7 is not recognizable by an RTA but it is recognizable by the TAC+: $\mathcal{A} = \langle \{q\}, \{q\}, \{a \to q, f(q,q) \xrightarrow{1=2} q\} \rangle$. □

3.3. DAG Automata

DAG automata (DA) [15] are defined as TA computing on the representation of terms as directed acyclic graphs (DAG) with maximal sharing. In some sense, RTA are the dual of DA. Indeed, in the runs of DA, a unique state is associated to equal subtrees (which are rooted by the same node in the DAG representations) whereas for RTA, a unique subtree is associated to every occurrence of the same rigid state. However, for DA, the state condition for equal subtrees must be enforced for every state (since DAG-representation with maximal sharing are considered), whereas the "dual" rigidity condition of RTA must only be enforced for rigid states. Consequently, the languages defined are incomparable.

Proposition 4. *The classes of languages of DA and RTA are orthogonal.*

Proof. On the one hand, one can observe that the language of the DAG representations of the terms of the RTA language of Example 1, $L = \{f(t,t) \mid t \in \mathcal{T}(\Sigma)\}$, is not recognizable by a DA. Assume by contradiction that it is recognized by a DA $\mathcal{A}$ with n states and let t be a term of depth strictly greater than n. By hypothesis, there exists a run r of $\mathcal{A}$ on the DAG representation of the term $f(t,t)$. Since there is a path in t of length strictly greater than n, there exists two different nodes v_1 and v_2 in the DAG representation of t that are labelled with the same state q in the run r. Let t_1 and t_2 be the subterms of t whose DAG representations are the subgraph rooted respectively in v_1 and v_2, and let t' be the term obtained form t by replacement of every occurrence of t_1 by t_2. Since both DAG representations of t_1 and t_2 are recognized by $\mathcal{A}$ in the same state q, any term containing some occurrences of t_1 recognized by $\mathcal{A}$, is still recognized by A if you replace any number of occurrences of t_1 by t_2. Then, the DAG representation of $f(t,t')$ is recognized by $\mathcal{A}$. Since $t \neq t'$, this is a contradiction and we conclude that there does not exist a DA recognizing the language of DAG representations of the terms of the RTA language L.

On the other hand, the language of DAG-representations of the terms of $L_{\max}$ of Example 6 (which is not recognizable by an RTA), is recognized by the

following DA (we admit that for the terms $t = H[t_0, \ldots, t_n]$ of $L_{\max}$, the context H can be empty. In this case, t is reduced to t_0):

$$\mathcal{A} = \langle \{q, q'\}, \{q'\}, \{0 \to q, s(q) \to q | q', h(q, q) \to q, h(q', q) \to q'\} \rangle$$

With this DA, a term $t = H[t_0, \ldots, t_n]$ is accepted in q' iff t_0 is accepted in q' and every t_i, for $i > 0$ is accepted in q. Moreover, since t is put in DAG form with maximal sharing for the computation of $\mathcal{A}$, and q' can only be reached through q, every t_i, for $i > 0$ is a strict subterm of t_0, meaning that $t \in L_{\max}$.□

Note also that the emptiness problem is PTIME for RTA and NP-complete for DA [15]. Moreover, deterministic DA coincide with DTA, and, as we show in Section 5.1, it is not the case for DRTA. Actually, DA and RTA are defined for different purposes: DA are proposed for computing on compressed trees, and not for checking equalities like RTA.

3.4. TA1M

Like pushdown tree automata [4], TA with one memory (TA1M) [3, 16] are TA extended in order to carry an unbounded amount of information along the states in computations. But instead of a stack, a TA1M stores this information in a memory with a tree structure. More precisely, this memory contains a ground term over a memory signature Γ. The memory is updated during the bottom-up computations. The general form of the transitions of TA1M is

$$f\big(q_1(m_1), \ldots, q_n(m_n)\big) \to q(m)$$

where $f \in \Sigma_n$, $q_1, \ldots, q_n, q$ are states with an argument carrying the memories $m_1, \ldots, m_n, m \in \mathcal{T}(\Gamma, \mathcal{X})$. The new current memory m is built from the memories $m_1, \ldots, m_n$ which have been reached at the positions immediately below the current position of computation. For instance, in the following *push* transition, the new current memory m is built by pushing a symbol $h \in \Gamma_n$ at the top of memories $m_1, \ldots, m_n$ (which are variables $x_1, \ldots, x_n$ in this case):

$$f\big(q_1(x_1), \ldots, q_n(x_n)\big) \to q\big(h(x_1, \ldots, x_n)\big). \qquad \text{(push)}$$

In a *pop* transition, the new current memory is a subterm of one of the memories reached so far:

$$f\big(q_1(x_1), \ldots, q_i(h(y_1, \ldots, y_k)), \ldots, q_n(x_n)\big) \to q(y_j) \qquad \text{(pop)}$$

The top symbol h of m_i is also read in the above pop transition.
In an *internal* transition, the new current memory is one of the memories reached:

$$f\big(q_1(x_1), \ldots, q_n(x_n)\big) \to q(x_i) \qquad \text{(internal)}$$

with $1 \leq i \leq n$.

Moreover, TA1M can perform equality tests on the memory contents, with transitions like

$$f\big(q_1(x_1), \ldots, q_n(x_n)\big) \xrightarrow{x_i = x_j} q(x_k) \qquad \text{(internal}_=\text{)}$$

where $1 \leq i, j, k \leq n$. This ability makes possible the simulation of some tests of the TA with constraints (by storing some subterms in memory and comparing them), with a limitation to local tests (like for TAC).

Proposition 5. *The classes of languages of TA1M and RTA are orthogonal.*

Proof. Let $\Sigma = \{a : 0, g : 1, f : 2\}$. The language

$$L = \{f(g^n(f(s,t)), f(s, g^n(f(s,t)))) \mid s, t \in \mathcal{T}(\Sigma), n \geq 1\}$$

is recognized by the following RTA

$$\mathcal{A} = \left\langle \{q, q_s, q_1, q_r, q_2, q_\mathsf{f}\}, \{q_s, q_r\}, \{q_\mathsf{f}\}, \left\{ \begin{array}{l} a \to q|q_s, g(q) \to q|q_s, f(q,q) \to q|q_s, \\ f(q_s, q) \to q_1, g(q_1) \to q_1|q_r, \\ f(q_s, q_r) \to q_2, f(q_r, q_2) \to q_\mathsf{f} \end{array} \right\} \right\rangle$$

The language L cannot be recognized by a TA1M. The reason is that, in order to recognize a term of the language, the automata needs to test equalities between i) the two subterms $g^n(f(s,t))$ and ii) the two subterms s in the right side of every term of L. For i, it would need to store $g^n(f(s,t))$ in its memory but for ii, it would also need to have s stored at the same time, which is not possible, since the memory can only store one term at a time.

Let us now consider $\Sigma' = \{a : 0, g : 1, h : 1, f : 2\}$ and the language $L' = \{f(g^n(a), h^n(a)) \mid n \geq 0\}$. This language is recognized by the TA1M with the following 4 push transitions and 1 test transition (q_f is the only final state):

$$\begin{array}{l} a \to q_g(a)|q_h(a),\ g(q_g(x_1)) \to q_g(g(x_1)),\ h(q_h(x_1)) \to q_h(g(x_1)), \\ f(q_g(x_1), q_h(x_2)) \xrightarrow{x_1 = x_2} q_\mathsf{f}(x_1) \end{array}$$

Note that the above transitions can only push the symbol g, whenever g or h is read. The states q_g and q_h permit to differentiate between $g^n(a)$ and $h^n(a)$.

As a consequence of the pumping Lemma 1, the language L' is not recognized by an RTA. □

3.5. Automatic Clauses with Rigid Variables

In this section, we show that the languages of rigid tree automata can alternatively be defined as finite set of Horn clauses with rigid variables [17]. This formalism was used in several related works [18, 19]. These papers do not mention the name of tree automata, but they are targeted at the same application as the one presented in Section 8: the static analysis of security protocols.

Following [20], it is a common approach to represent tree automata by Horn clause sets. A tree automata transition $f(q_1, \ldots, q_n) \to q$ can indeed be encoded into the following first order Horn clause (variables are implicitly universally quantified)

$$q_1(y_1), \ldots, q_n(y_n) \Rightarrow q\big(f(y_1, \ldots, y_n)\big) \qquad \text{(reg)}$$

where $y_1, \ldots, y_n$ are distinct variables and $q_1, \ldots, q_n, q$ are unary predicate symbols. Let us call *regular clauses* the Horn clauses of the above form. Given a

finite set $\mathcal{C}$ of regular clauses (an *automaton* in these settings) and a predicate q (a *state*), the language of $\mathcal{C}$ in q, denoted by $L(\mathcal{C}, q)$, is the set of terms $t \in \mathcal{T}(\Sigma)$ such that $q(t)$ is a logical consequence of $\mathcal{C}$ ($q(t)$ is in the smallest Herbrand model of $\mathcal{C}$). This definition corresponds exactly to the language of the TA whose transitions are encoded by the clauses of $\mathcal{C}$.

One advantage of this presentation of tree automata by Horn clause sets is that it permits to use classical first-order theorem proving techniques in order to decide TA problems. For instance, if $\mathcal{C}$ is a finite set of regular clauses and $t \in \mathcal{T}(\Sigma)$, it holds that $t \in L(\mathcal{C}, q)$ iff $\mathcal{C} \cup \{q(t) \Rightarrow\}$ is inconsistent, and $L(\mathcal{C}, q) \neq \emptyset$ iff $\mathcal{C} \cup \{q(x) \Rightarrow\}$ is inconsistent. These sets can be finitely saturated by a resolution calculus with appropriate strategies [21], hence, the above decision problems can be solved using first order theorem provers.

This approach can also be suitable for studying RTA, by distinguishing, in regular clauses, some variables as so called *rigid variables* [17]. We use below uppercase letters $X, Y...$ for rigid variables and lowercase $x, y...$ for other variables, called *flexible* variables. Recently, in [18, 19], some models of Horn clauses with rigid variables (including regular clauses) have been studied in the context of the verification of security protocols. We recall the definitions and results of [19] in order to establish connections with RTA.

A set $\mathcal{C}$ of clauses with rigid variables $X_1, \ldots, X_n$ and flexible variables $y_1, \ldots, y_m$ is *satisfiable* if there exists a Σ-algebra $\mathfrak{A}$ such that for all valuation $\sigma : \{X_1, \ldots, X_n\} \to \mathfrak{A}$, there exists a model $\mathcal{S}$ with domain $\mathfrak{A}$ such that $\mathcal{S}, \sigma \models \forall y_1, \ldots, y_m\, \mathcal{C}$. It is equivalent to say that for all valuation $\sigma : \{X_1, \ldots, X_n\} \to \mathcal{T}(\Sigma)$, there exists an Herbrand model $\mathcal{L}$ such that $\mathcal{L} \models \forall y_1, \ldots, y_m\, \sigma(\mathcal{C})$.

This semantics permits to redefine the languages of RTA in term of models of regular clauses with rigid variables. Let us consider an RTA $\mathcal{A} = \langle Q, R, F, \Delta \rangle$ and let us associate a rigid variable X_q to each $q \in R$. We associate to $\mathcal{A}$ the set $\mathcal{C}$ of regular clauses with rigid variables

$$q_1(\alpha_1), \ldots, q_n(\alpha_n) \Rightarrow q\big(f(\alpha_1, \ldots, \alpha_n)\big) \qquad \text{(reg')}$$

such that $f(q_1, \ldots, q_n) \to q \in \Delta$ and for all $i \leq n$, $\alpha_i = X_{q_i}$ if $q_i \in R$ and α_i is a flexible variable y_i otherwise. Then, we have ($\overline{X} = \{X_q \mid q \in R\}$)

$$\bigcup_{\sigma : \overline{X} \to \mathcal{T}(\Sigma)} L\big(\sigma(\mathcal{C}), q\big) = L(\mathcal{A}, q).$$

In [19], a translation of clauses with rigid variables into first order clauses (without rigid variables) preserving satisfiability is proposed. In the case of the above regular clause with rigid variables (reg'), the translation returns

$$q_1(\overline{x}, \alpha'_1), \ldots, q_n(\overline{x}, \alpha'_n) \Rightarrow q\big(\overline{x}, f(\alpha'_1, \ldots, \alpha'_n)\big)$$

where $\overline{x} = (x_q)_{q \in R}$ is a sequence of $|R|$ flexible variables, one variable x_q for each rigid state $q \in R$ (hence one for each rigid variable X_q). Every variable α'_i, $i \leq n$, is either x_{q_i} if α_i is the rigid variable X_{q_i} (i.e. if $q_i \in R$) and α'_i is the (flexible) variable $\alpha_i = y_i$ otherwise. Such clauses can alternatively be seen as

transitions of tree automata extended with $|R|$ auxiliary registers storing terms of $\mathcal{T}(\Sigma)$. For such an automaton, the values are stored in the registers once and for all at the beginning of the computation (in the variables of $\overline{x}$) and during the application of a transition, the current subterm can be compared to the content of one register (in the case where $\alpha'_i = y_i$).

It is shown in [19] that binary resolution with an appropriate ordered strategy terminates on clauses of the above form as long as there is only one unary predicate; [19] consider also other kinds of clauses; some of them can be seen as a generalisation of RTA to two way and alternating rigid tree automata.

Hence, in the result of [19], termination is limited to automata with one state. The resolution strategy of [19] does not terminate on automata with more than one state and a terminating resolution strategy for this case is not known. Some progress in this direction would enable the application of first order theorem proving techniques to decision problem for RTA. This could permit in particular to consider extensions of RTA with e.g. equational tests or language modulo equational theories, like what was done in [10] for standard tree automata using a Horn clauses approach and a paramodulation calculus. In particular, the latter extension (modulo equational theories) is related to the problem of Section 7.3, and in this context, first order theorem proving tools could provide an efficient alternative to the complicated decision algorithm described in Section 7.3.

4. Boolean Closure

We show below that the class of RTA languages is closed under union and intersection but not under complement.

4.1. Union and Intersection

Theorem 1. *Given two RTA $\mathcal{A}_1$ and $\mathcal{A}_2$, there exist two RTA of respective sizes $O(|\mathcal{A}_1|+|\mathcal{A}_2|)$ and $O(2^{|\mathcal{A}_1||\mathcal{A}_2|})$, constructed respectively in polynomial and exponential time, and recognizing respectively $L(\mathcal{A}_1)\cup L(\mathcal{A}_2)$ and $L(\mathcal{A}_1)\cap L(\mathcal{A}_2)$.*

Proof. Let $\mathcal{A}_i = \langle Q_i, R_i, F_i, \Delta_i\rangle$ with $i = 1, 2$. For $L(\mathcal{A}_1) \cup L(\mathcal{A}_2)$, we do a classical disjoint union of automata. Let us assume *wlog* that the state sets Q_1, Q_2 of $\mathcal{A}_1$ and $\mathcal{A}_2$ are disjoint. Like for the union of TA, the RTA $\mathcal{A}$ is obtained by disjoint union of the state sets, rigid state sets, final state sets and transitions sets.

For $L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$, it is easy to construct a TAGED+ $\mathcal{A}'$ (see Section 3.1) recognizing $L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$ by a Cartesian product operation like for standard TA. The state set of $\mathcal{A}'$ is $Q_1 \times Q_2$, its final state set $F_1 \times F_2$ and its set of transition rules is

$$\left\{ \begin{array}{ll} f\big(\langle q_{11}, q_{21}\rangle, \ldots, \langle q_{1n}, q_{2n}\rangle\big) \to \langle q_1, q_2\rangle & \mid \quad q_{i1} \ldots q_{in}, q_i \in Q_i, \\ & \mid \quad f(q_{i1}, \ldots, q_{in}) \to q_i \in \Delta_i, i = 1, 2 \end{array} \right\}.$$

Moreover, the equality relation of $\mathcal{A}'$ is

$$\begin{array}{rcl} R_= & = & \big\{\big\langle\langle q_{r_1}, q_2\rangle, \langle q_{r_1}, q'_2\rangle\big\rangle \mid q_{r_1} \in R_1, q_2, q'_2 \in Q_2\big\} \\ & \cup & \big\{\big\langle\langle q_1, q_{r_2}\rangle, \langle q'_1, q_{r_2}\rangle\big\rangle \mid q_1, q'_1 \in Q_1, q_{r_2} \in R_2\big\}. \end{array}$$

We can use Proposition 2 of [6] in order to transform this TAGED+ into an RTA recognizing the same language, at the price of an exponential blowup. Combining the two above steps results in an exponential construction for the intersection of RTA. □

The following lemma shows that the exponential time complexity for the construction of the intersection automaton in Theorem 1 is a lower bound, with a reduction of the EXPTIME-complete problem of the non-emptiness of the intersection of n TA.

Lemma 2. *Given n TA $\mathcal{A}_1, \ldots, \mathcal{A}_n$ on Σ, we can compute in polynomial time two RTA $\mathcal{A}_\times$ and $\mathcal{A}_r$, both of size $O\big(|\mathcal{A}_1| + \ldots + |\mathcal{A}_n|\big)$, and such that $L(\mathcal{A}_1) \cap \ldots \cap L(\mathcal{A}_n) = \emptyset$ iff $L(\mathcal{A}_\times) \cap L(\mathcal{A}_r) = \emptyset$.*

Proof. Let $\Sigma_d = \Sigma \uplus \{0:0, d:2\}$. Both the RTA constructed will compute on Σ_d. Let

$$\mathcal{A}_r = \big\langle \{q, q_r, q_\mathsf{f}\}, \{q_r\}, \{q_\mathsf{f}\}, \{0 \to q_\mathsf{f}, d(q_r, q_\mathsf{f}) \to q_\mathsf{f}\} \cup \{f(q, \ldots, q) \to q | q_r \mid f \in \Sigma\}\big\rangle.$$

It recognizes the set of right combs of the form $d(t, d(t, \ldots d(t, 0)))$ with $t \in \mathcal{T}(\Sigma)$. Let $\mathcal{A}_i = \langle Q_i, R_i, F_i, \Delta_i \rangle$ for all $1 \leq i \leq n$. We assume wlog that $Q_1, \ldots, Q_n$ are disjoint and that for each $i \leq n$, $F_i = \{q_i\}$.

Let $\mathcal{A}_\times = \big\langle \biguplus_{i=1}^n Q_i \uplus \{q_0, q'_1, \ldots, q'_n\}, \biguplus_{i=1}^n R_i, \{q'_1\}, \Delta_\times \big\rangle$, with

$$\Delta_\times = \biguplus_{i=1}^{n} \Delta_i \uplus \{0 \to q_0, d(q_n, q_0) \to q'_n\} \cup \biguplus_{i=1}^{n-1} d(q_i, q'_{i+1}) \to q'_i.$$

This RTA $\mathcal{A}_\times$ recognizes the set of right combs of the form $d(t_1, \ldots d(t_n, 0)))$ with $t_i \in L(\mathcal{A}_i)$ for all $i \leq n$. Hence $L(\mathcal{A}_\times) \cap L(\mathcal{A}_r)$ is exactly the set of right combs $d(t_1, \ldots d(t_n, 0)))$ such that $t_i \in L(\mathcal{A}_i)$ for all $i \leq n$ and $t_1 = \ldots = t_n$. Therefore, this intersection is empty iff $L(\mathcal{A}_1) \cap \ldots \cap L(\mathcal{A}_n)$ is empty as well. □

Note that the above construction also works (hence Lemma 2 also holds) for n given RTA. With Lemma 2, we have a polynomial time reduction into the non-emptiness of the intersection of two RTA of the problem of the intersection non-emptiness for n TA (given n TA $\mathcal{A}_1, \ldots, \mathcal{A}_n$, do we have $L(\mathcal{A}_1) \cap \ldots \cap L(\mathcal{A}_n) \neq \emptyset$?). The latter problem is known to be EXPTIME-complete [22]. Since by Theorem 1, the intersection of two RTA is an RTA, and the emptiness of RTA can be decided in linear time (Theorem 8 below), we conclude that EXPTIME is a lower bound for the construction of an RTA for the intersection. Moreover, in the above construction, $\mathcal{A}_\times$ is a TA if every $\mathcal{A}_i$ ($1 \leq i \leq n$) is a TA. Hence the intersection of an RTA with a TA also leads to an exponential construction.

4.2. Complement

Theorem 2. *The class of RTA languages is not closed under complement.*

Proof. We have seen in Example 7 that the set $\mathcal{B}$ of balanced binary trees over $\Sigma := \{a : 0, f : 2\}$ is not an RTA language. We show that its complement $\overline{B}$ in $\mathcal{T}(\Sigma)$ is an RTA language. The idea is similar to the construction for the subterm relation in Example 4: one rigid state q_r is used to choose non-deterministically a subterm, and it is checked that the sibling of q_r contains q_r at depth more than one (such subterms are characterised by the state q' below). More precisely, the RTA for $\overline{B}$ is $\langle \{q, q_r, q', q_\mathsf{f}\}, \{q_r\}, \{q_\mathsf{f}\}, \Delta \rangle$ with

$$\Delta = \left\{ \begin{array}{rcl@{\qquad}rcl@{\qquad}rcl} a & \to & q|q_r, & f(q,q) & \to & q|q_r, & f(q,q') & \to & q', \\ f(q,q_r) & \to & q', & f(q_r,q) & \to & q', & f(q',q) & \to & q', \\ f(q_r,q') & \to & q_\mathsf{f}, & f(q',q_r) & \to & q_\mathsf{f}, & & & \\ f(q_\mathsf{f},q) & \to & q_\mathsf{f}, & f(q,q_\mathsf{f}) & \to & q_\mathsf{f} & & & \end{array} \right\}.$$

The last two transition rules ensure the propagation of the final state q_f up to the root, like in Example 2. □

5. Deterministic and Visibly Rigid Tree Automata

Non-determinism is crucial for an RTA recognizing the terms of the form $f(t,t)$ like in Example 1. Indeed, in a bottom-up computation, such an automaton needs to guess both positions of the two occurrences of t under the symbol f, and put one rigid state at these positions.

Example 8. Let us come back to Example 1, where $\Sigma = \{a{:}0, b{:}0, f{:}2\}$ and the RTA $\mathcal{A}$ with transition set $\{a \to q|q_r, b \to q|q_r, f(q,q) \to q|q_r, f(q_r,q_r) \to q_\mathsf{f}\}$ recognizing $\{f(t,t) \mid t \in \mathcal{T}(\Sigma)\}$. Applying a classical subset construction to the transition set of $\mathcal{A}$ returns a deterministic set of transitions

$$\left\{ \begin{array}{c} a \to \{q,q_r\}, \quad b \to \{q,q_r\}, \quad f\big(\{q,q_r\},\{q,q_r\}\big) \to \{q,q_r,q_\mathsf{f}\}, \\ f\big(\{q,q_r,q_\mathsf{f}\},\{q,q_r,q_\mathsf{f}\}\big) \to \{q,q_r,q_\mathsf{f}\} \end{array} \right\}.$$

However, it is not possible to choose a subset of rigid states amongst the two states obtained, in order to recognize the above language. ◇

We show in this section that RTA can not be determinized in general, and propose a subclass for which a determinization is possible, though it is still not closed under complement.

5.1. Determinism and Completeness

Definition 3. A *deterministic rigid tree automaton* (DRTA) (resp. complete RTA) on a signature Σ is an RTA $\mathcal{A}$ whose underlying TA $ta(\mathcal{A})$ is deterministic (resp. complete).

Like standard TA, every RTA can be completed into a complete RTA, by the addition of a trash state.

Theorem 3. *For every RTA $\mathcal{A}$, a complete RTA $\mathcal{A}'$ of size polynomial in $|\mathcal{A}|$ and such that $L(\mathcal{A}') = L(\mathcal{A})$ can be constructed in PTIME from $\mathcal{A}$.*

Proof. Let $\mathcal{A} = \langle Q, R, F, \Delta \rangle$. We use the same construction as for standard TA, adding a state $q_\perp$ which is neither final nor rigid: $\mathcal{A}' = \langle Q \cup \{q_\perp\}, R, F, \Delta_\perp \rangle$ with

$$\begin{aligned}
&\Delta_\perp = \Delta \\
&\cup \big\{ f(q_1, \ldots, q_n) \to q_\perp \mid f \in \Sigma_n, q_1, \ldots, q_n \in Q, \forall q \in Q, f(q_1, \ldots, q_n) \to q \notin \Delta \big\} \\
&\cup \big\{ f(q_1, \ldots, q_n) \to q_\perp \mid f \in \Sigma_n, q_1, \ldots, q_n \in Q \cup \{q_\perp\}, \exists i \leq n, q_i = q_\perp \big\}.
\end{aligned}$$

□

However, unlike standard TA, it is not true in general that for a complete RTA $\mathcal{A}$, for every term t there exists at least one run of $\mathcal{A}$ on t. Indeed, a run of $ta(\mathcal{A})$ on t might not be a run of $\mathcal{A}$ on t because of the rigidity condition.

Example 9. The RTA $\mathcal{A} = \big\langle \{q, q_r\}, \{q_r\}, \{q\}, \{a \to q, g(q) \to q_r, g(q_r) \to q\} \big\rangle$ is deterministic and complete. The term $t = g(g(g(a)))$ is in $L\big(ta(\mathcal{A}), q_r\big)$, it is accepted with a unique (TA) run $r = q_r(q(q_r(q)))$. However, r is not a run of the RTA $\mathcal{A}$, because the two subterms at the positions of q_r are distinct. ◇

It is well-known that DTA are as expressive as TA, and that every TA can effectively be determinized, at the price of an exponential blowup. We show below that it is not the case for RTA: the class of DRTA languages is strictly included in the class of RTA languages.

Theorem 4. *DRTA $\subsetneq$ RTA.*

Proof. Let $\Sigma = \{a{:}0, f{:}2\}$. The language $L = \{f(t,t) \mid t \in T(\Sigma)\}$ is recognized by the RTA of Example 1, without the transitions rules for symbol b.

We show now that L is not recognized by a DRTA. Assume that there is a DRTA $\mathcal{A} = \langle Q, R, F, \Delta \rangle$ recognizing L. On any run r of $\mathcal{A}$, on any tree, each rigid state can only appear once on a path; otherwise it would not respect the rigid condition. Hence there is at most $|R|$ occurrences of rigid states on every path. Let t be a tree on which there exists a (unique) run r of $\mathcal{A}$, and let $p \in \mathcal{P}os(t)$ be a path from the root to a leaf which contains a maximal number of rigid states in r.

We build a tree t' such that there exists a position $p' \in \mathcal{P}os(t')$, $|p'| > |Q| - |R|$ and $t'|_{p'} = t$. Since $f(t', t')$ is recognized by $\mathcal{A}$, there exists a (unique) run r' on t'. Since $\mathcal{A}$ is deterministic, we know that $r'|_{p'} = r$. Hence there exists a path in r' from the position p' to a leaf that contains the maximal number of rigid states. So for each strict prefix p'_0 of p', $r'(p'_0) \in Q \setminus R$. Since $|p'| > |Q| - |R|$, there exists two strict prefixes p'_1, p'_2 of p', such that p'_1 is a strict prefix of p'_2 and $r'(p'_1) = r'(p'_2)$. Let t'' be the tree $t'[t'_{p'_2}]_{p'_1}$. This construction is illustrated in Figure 2.

Then $r'' = r'[r'_{p'_2}]_{p'_1}$ is a valid run of $\mathcal{A}$ on t'': no rigid states occur between the root and p'_1, and between p'_1 and p'_2, so a position p'_3 of an occurrence of a rigid state was either

- a position incomparable (wrt prefix ordering) with p'_1, which still exists with the same subtree and the same rigid states in t'',

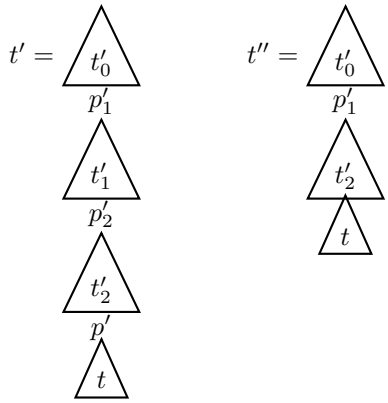

Figure 2: Proof that DRTA $\subsetneq$ RTA

- a position $p'_2 \cdot \pi$, $\pi \in \mathcal{P}os(t'|_{p'_2})$, and then the position $p'_1 \cdot \pi$ in t'' has the same subtree and the same rigid state,
- a position $p'_1 \cdot \pi$, $\pi \in \mathcal{P}os(t'|_{p'_1})$, where π is not a suffix of p'_2, and in this case, this occurrence of the rigid states disappears in t''.

Therefore, r'' satisfies the rigid condition on every rigid state of R. Since $r''(\varepsilon) = r'(\varepsilon)$, $\mathcal{A}$ recognizes the tree $f(t'', t')$ which is not in L. □

Moreover, the class of regular tree languages is strictly included into the class of DRTA languages.

Theorem 5. *TA $\subsetneq$ DRTA.*

Proof. The inclusion TA $\subset$ DRTA is immediate since DTA $\equiv$ TA and DTA are particular cases of DRTA.

Let $\Sigma = \{a : 0, g : 1, f : 2\}$. The language $\big\{f(g(t), g(t)) \mid t \in \mathcal{T}(\Sigma \setminus \{g\}\big\}$ is recognized by the DRTA

$$\mathcal{A} = \big\langle \{q, q_r, q_{\mathsf{f}}\}, \{q_r\}, \{q_{\mathsf{f}}\}, \big\{a \to q, f(q, q) \to q, g(q) \to q_r, f(q_r, q_r) \to q_{\mathsf{f}}\big\}\big\rangle.$$

But this language if not regular. □

5.2. Visibly Rigid Tree Automata

We propose here a class of restricted RTA which can be determinized. The definition of the restriction is inspired by the theory of visibly pushdown automata (VPA) [23]. VPA define a subset of context-free languages closed under intersection and complement. They were generalized to tree recognizers in [5, 16]. The idea in these works is that the signature Σ is partitioned into $\Sigma = \Sigma_c \uplus \Sigma_r \uplus \Sigma_\ell$ and the operation performed by the VPA on the stack depends on the current symbol in the input: if it is a *call* symbol of Σ_c, the VPA can only do a push, for a *return* symbol of Σ_r it can do a pop and it must leave

the stack untouched for a *local* symbol of Σ_ℓ. The transitions of a VPA follow this discipline. There exists a determinization for this subclass of pushdown automata, and it is closed under intersection and complement.

The RTA use no auxiliary stack but they permit the comparison between subterms based on the rigid states. Hence, a natural way for defining a condition similar to the one of visibly pushdown automata, and enabling determinization for some RTA, is to restrict the rigid states that can be reached according to the function symbol in the input. In that sense, the rigidity of the states is made visible by the input signature.

Definition 4. A visibly rigid tree automaton (VRTA) is an RTA $\mathcal{A} = \langle Q, R, F, \Delta\rangle$ on a signature Σ such that there exists a partial function ν from Σ to R such that for every transition $f(q_1, \ldots, q_n) \to q \in \Delta$, $q = \nu(f)$ if ν is defined on f and $q \in Q \setminus R$ otherwise.

Example 10. The RTA of Example 3 is visibly rigid, with a function ν defined only on g by $\nu(g) = q_r$. The DRTA in the above proof of Theorem 5 is also visibly rigid, with the same function.

Conversely, the RTA of Example 1 (recognizing the terms $f(t, t)$ with $t \in \mathcal{T}\big(\{a : 0, b : 0, f : 2\}\big)$ is not visibly rigid. Intuitively, some non-determinism is needed for the bottom-up recognition of this language (because t may contain the symbol f), and it is not compatible with the visibly rigid condition. Indeed, the above language is not regular, hence at least one rigid state is necessary for the definition of a RTA recognizing it. Defining rigid states for $\nu(a)$ and $\nu(b)$, is pointless (it can be simulated by standard tree automata). Hence,$\nu(f)$ must be defined in order to ensure the visibly rigid condition, but this would contradict the recognition a term such as e.g. $f(f(a, a), f(a, a))$. $\diamond$

With the visibly rigid condition, a determinization procedure can be applied to VRTA.

Theorem 6. *Given a VRTA $\mathcal{A}$ on Σ, a deterministic VRTA $\mathcal{A}'$ on Σ of size exponential in $|\mathcal{A}|$ and such that $L(\mathcal{A}') = L(\mathcal{A})$ can be constructed in exponential time.*

Proof. The RTA $\mathcal{A}'$ is obtained by a classical subset construction. Let $\mathcal{A} = \big\langle Q, R, F, \Delta\big\rangle$ and let $\mathcal{A}' = \big\langle 2^Q, 2^R, \{S \subseteq Q \mid S \cap F \neq \emptyset\}, \Delta'\big\rangle$ with

$$\Delta' = \left\{ \begin{array}{l} f(S_1, \ldots, S_n) \to S \mid S_1, \ldots S_n, S \subseteq Q, \\ \qquad S = \{q \in Q \mid \exists q_1 \in S_1, \ldots, \exists q_n \in S_n, f(q_1, \ldots, q_n) \to q \in \Delta \end{array} \right\}.$$

The RTA $\mathcal{A}'$ is deterministic. Moreover, because of the visibly rigid condition for $\mathcal{A}$, every state of $\mathcal{A}'$ occurring in Δ (i.e. every state of $\mathcal{A}'$ with a non empty language) is either a subset of $Q \setminus R$ (and it is not a rigid state of $\mathcal{A}'$) or is a singleton subset of R (and it is a rigid state of $\mathcal{A}'$). Hence, given a function ν associated to the VRTA $\mathcal{A}$ like in Definition 4, there exists a function ν' making $\mathcal{A}'$ a VRTA, defined by $\nu'(f) = \{q_r\}$ iff $\nu(f) = q_r \in R$.

We can show by induction on $t \in \mathcal{T}(\Sigma)$ that there exists a run r of $\mathcal{A}$ on t iff there exists a run r' of $\mathcal{A}'$ on t such that for all $p \in \mathcal{P}os(t)$, $r(p) \in r'(p)$. The part of the proof which is specific to (V)RTA concerns the rigidity condition, and uses the above observation about the states of $\mathcal{A}'$: all the rigid states in a run r' of $\mathcal{A}'$ are singleton subsets of R. Hence, for the *if* direction, given a run r' of $\mathcal{A}'$ on t, every relabeling $r : \mathcal{P}os(t) \to Q$ extracted from r' (i.e. such that $r(p) \in r'(p)$, $p \in \mathcal{P}os(t)$) satisfies the rigidity condition. Similarly, for the *only if* direction, a relabeling $r' : \mathcal{P}os(t) \to 2^Q$ embedding a given run r of $\mathcal{A}$ on t also satisfies the rigidity condition. It follows that $t \in L(A)$ iff $t \in L(\mathcal{A}')$. □

Being able to determinize VRTA is not enough however to ensure the closure of this subclass of RTA under complement. Intuitively, the reason is that for the (unique) run r of a deterministic VRTA to be successful, a conjunction of two conditions must be realized: the top state of r must be final and the rigidity condition has to be enforced. In comparison, for a TA, only the first condition is necessary, and in order to construct the complement of a deterministic and complete TA, an inversion of final and non final states is sufficient. But in order to characterize the complement of a VRTA language, the disjunction of the negation of the two above conditions is necessary, and VRTA are not expressive enough in order to characterize a term not satisfying a rigidity condition.

Theorem 7. *The class of VRTA languages is not closed under complement.*

Proof. Let us consider the language L_g of Example 3: the set of terms $t \in \mathcal{T}(\Sigma)$, with $\Sigma = \{a : 0, g : 1, f : 2\}$, such that $s_1 = s_2$ for every two subterms $g(s_1)$, $g(s_2)$ of t. L_g is recognized by the VRTA $\mathcal{A}$ given in Example 3 but its complement is not a language of VRTA.

Assume that the complement $\mathcal{T}(\Sigma) \setminus L_g$ of L_g is recognized by a VRTA $\mathcal{A}'$ and let ν' be the function associated to $\mathcal{A}'$ like in Definition 4. Since L_g is not regular, $L(\mathcal{A}')$ is neither regular, and hence $\mathcal{A}'$ has to contain at least one rigid state q_r such that $L(\mathcal{A}', q_r) \neq \emptyset$. Hence, there exists a function symbol $h \in \Sigma$ such that $\nu(h) = q_r$. It cannot be g, otherwise $\mathcal{A}'$ would not be able to recognize any term of the form $f(g(t_1), g(t_2))$ with $t_1 \neq t_2$ (such a term is in the complement of L_g). It cannot be f either, otherwise $\mathcal{A}'$ would not be able to recognize terms of the form $f(g(t_1), g(t_2))$ with $t_1 = f(t_3, t_4)$ and $t_1 \neq t_2$ (those terms are also all is in the complement of L_g). Hence h has to be a, but with rigid states bound to constant symbols, VRTA do not have more expressive power than standard TA. It follows that there does not exist any VRTA $\mathcal{A}'$ recognizing $\mathcal{T}(\Sigma) \setminus L_g$. □

It is not known whether or not, in general, the complement of a VRTA language is an RTA language.

6. Decision problems

We study in this section several decision problems for RTA: emptiness, membership, intersection non-emptiness, universality, inclusion, equivalence, and

	TA	**RTA**	**TAGED+**	**DA**
$\cup$	PTIME	PTIME	PTIME	PTIME
$\cap$	PTIME	EXPTIME	EXPTIME	not [24]
$\neg$	EXPTIME	not	not	not
emptiness	linear-time	linear-time	EXPTIME-complete	NP-complete
membership	PTIME	NP-complete	NP-complete	NP-complete
$\cap$-emptiness	EXPTIME-complete	EXPTIME-complete	EXPTIME-complete	
universality	EXPTIME-complete	undecidable	undecidable	undecidable
inclusion	EXPTIME-complete	undecidable	undecidable	undecidable
finiteness	PTIME	PTIME	EXPTIME	

Table 1: Summary of closure and decision results

finiteness. Table 1 provides a summary of closure and decision results and a comparison with other classes of extended TA mentioned in Section 3.

6.1. Emptiness

Emptiness is the problem of deciding, given an RTA $\mathcal{A}$ whether $L(\mathcal{A}) = \emptyset$. We show below that deciding emptiness for an RTA amounts to decide emptiness for the underlying TA.

Theorem 8. *The emptiness problem is decidable in linear time for RTA.*

Proof. Let $\mathcal{A} = \langle \Sigma, Q, R, F, \Delta \rangle$ and let $\mathit{rigid}(\mathcal{A}) = \langle \Sigma, Q, Q, F, \Delta \rangle$ be a copy of $\mathcal{A}$ where every state is rigid. We show that the emptiness of $L(\mathcal{A})$ and $L\big(\mathit{rigid}(\mathcal{A})\big)$ and $L\big(\mathit{ta}(\mathcal{A})\big)$ are equivalent. The latter problem (emptiness for standard TA) is known to be decidable in linear-time (see e.g. [11]) with an algorithm marking the inhabited states of $\mathit{ta}(\mathcal{A})$ and using an appropriate data structure for the transitions rules. The idea of the proof is that if $L(\mathit{ta}(\mathcal{A}))$ is not empty, then the classical "state marking" algorithm builds a witness which respects the rigidity condition for all states, and is therefore a witness for $L(\mathcal{A})$ non-emptiness.

In order to establish the above equivalence, we use a similar algorithm for $\mathcal{A}$ except that every inhabited state q is marked by a witness (minimal) term $t_q \in L(\mathit{rigid}(\mathcal{A}), q)$ and a run r_q of $\mathit{rigid}(\mathcal{A})$ on t_q. At the beginning, each t_q and r_q are undefined. Then we iterate the following transformation until it is applicable:

> if $q \in Q$, t_q is undefined, and there exists $f(q_1, \ldots, q_n) \to q \in \Delta$ such that $t_{q_1}, \ldots, t_{q_n}$ are all defined, then let $t_q := f(t_{q_1}, \ldots, t_{q_n})$ and $r_q := q(r_{q_1}, \ldots, r_{q_n})$.

The above step will be repeated at most $|Q|$ times, and using suitable data structures (see [11]) for the representation of transition rules ensures that it runs in linear time (note that the update of t_q and r_q can be performed in constant times at each step). For all $q \in Q$, the following facts are equivalent:

i. t_q is defined,

ii. $L(rigid(\mathcal{A}), q) \neq \emptyset$,

iii. $L(\mathcal{A}, q) \neq \emptyset$.

iv. $L(ta(\mathcal{A}), q) \neq \emptyset$.

$i \Rightarrow ii$ follows from the construction: if t_q is defined then r_q is a run of $L(rigid(\mathcal{A}))$ on t_q. This can be shown e.g. by induction on the number of iteration steps before t_q is defined.

$ii \Rightarrow iii$ and $iii \Rightarrow iv$ are immediate, because by definition we have $L(rigid(\mathcal{A}), q) \subseteq L(\mathcal{A}, q) \subseteq L(ta(\mathcal{A}), q)$.

$iv \Rightarrow i$ can be shown by induction on the number of transition rules of $\mathcal{A}$. This procedure terminates and at the end, t_q is defined iff $L(rigid(\mathcal{A}), q) \neq \emptyset$ iff $L(\mathcal{A}, q) \neq \emptyset$ iff $L(ta(\mathcal{A}), q) \neq \emptyset$. □

6.2. Membership

Membership is the problem of deciding, given an RTA $\mathcal{A}$ and a term $t \in \mathcal{T}(\Sigma)$, whether $t \in L(\mathcal{A})$. A similar proof of the following result (in the case of TAGED) already appeared in [6].

Theorem 9. *Membership is* NP*-complete for RTA (*PTIME *for DRTA).*

Proof. A non-deterministic algorithm for this problem consists in, given an RTA $\mathcal{A}$ and a term t, guessing a labelling of the nodes of t with states of $\mathcal{A}$ and checking that this labelling is a successful run of $\mathcal{A}$ on t. The checking operation can be performed in polynomial time.
In the deterministic case, there is at most one labelling of the term t compatible with the transition rules. It can be computed in PTIME and it can be checked in PTIME that this labelling is a successful run. Hence the membership problem is decidable in PTIME for DRTA.

In order to show NP-hardness for general RTA, we propose a reduction of 3-SAT for a formula ϕ into the membership for an RTA $\mathcal{A}$ and a term t representing ϕ.

Let us consider an instance ϕ of 3-SAT with variables from a set V. It is represented as a term t over the signature $\Sigma = \{0, 1 : 0, \neg : 1 \wedge :2, \vee : 3\} \cup \{x : 2 \mid x \in V\}$. Every variable x is represented by a subterm $x(0, 1)$, a 3 literal clause $\ell_1 \vee \ell_2 \vee \ell_3$ is encoded into $\vee(t_1, t_2, t_3)$ where t_1, t_2, t_3 encode respectively ℓ_1, ℓ_2, ℓ_3. Finally we encode a conjunction of disjunctions $D_1 \wedge \cdots \wedge D_n$ into $\wedge(t_1, \ldots, \wedge(t_{n-1}, t_n))$ where each t_i, $i \leq n$, is the encoding of D_i.

For instance, the tree encoding of the 3-SAT instance $(x \vee y \vee z) \wedge (\neg x \vee y \vee t) \wedge (\neg y, \neg t, z)$ is depicted in Figure 3.

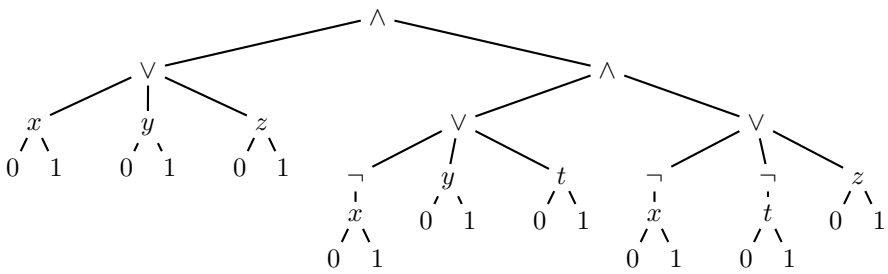

Figure 3: Membership NP-hardness: tree encoding of a 3 SAT instance.

We define an RTA $\mathcal{A} = \langle Q, R, F, \Delta \rangle$ on Σ by $R = \{q_x, q_{\neg x} \mid x \in V\}$, $Q = \{q_1, q_0\} \cup R$, $F = \{q_1\}$, and

$$\begin{array}{rcl}
\Delta & = & \{0 \to q_x | q_{\neg x}, 1 \to q_x | q_{\neg x} \mid x \in V\} \\
& \cup & \{x(q_x, q_{\neg x}) \to q_0, x(q_{\neg x}, q_x) \to q_1 \mid x \in V\} \\
& \cup & \{\vee(q_0, q_0, q_0) \to q_0\} \\
& \cup & \{\vee(q, q', q'') \to q_1 \mid \text{at least one of } q, q', q'' \text{ is } q_1, \text{ and the others are } q_0\} \\
& \cup & \{\neg(q_0) \to q_1, \neg(q_1) \to q_0\} \\
& \cup & \{\wedge(q_1, q_1) \to q_1, \wedge(q_0, q_1) \to q_0, \wedge(q_1, q_0) \to q_0, \wedge(q_0, q_0) \to q_0\}.
\end{array}$$

Both the automata $\mathcal{A}$ and the tree t are linear in size relatively to the size of the 3-SAT instance ϕ. The most important transitions of $\mathcal{A}$ are those of the two above lines involving the rigid states q_x and $q_{\neg x}$. The states q_0 and q_1 represent the value associated to x (they are propagated bottom-up along t) and the rigidity condition ensures that the same value is associated to all occurrences of the variable x in ϕ.

Let us show now in detail that $\mathcal{A}$ recognizes t iff the corresponding 3-SAT instance ϕ has a solution.

Assume that the given 3-SAT instance has a solution $\sigma : V \to \{0, 1\}$ (mapping of propositional variables into truth values). We define a successful run r of $\mathcal{A}$ in t as follows. For each variable $x \in V$ and for each position $p \in \mathcal{P}os(t)$ such that $t|_p = x$, we have by construction of t that $t|_{p.1} = 0$ and $t|_{p.2} = 1$. If $\sigma(x) = 0$, we define $r(p.1) = q_x$ and $r(p.2) = q_{\neg x}$, and if $\sigma(x) = 1$, we define $r(p.1) = q_{\neg x}$ and $r(p.2) = q_x$. Both options are possible thanks to the rules $0 \to q_{(\neg)x}$ and $1 \to q_{(\neg)x}$, and since we do the same thing for all occurrence of x in t, the rigid condition on q_x and $q_{\neg x}$ are satisfied for r. Only one rule can be applied at position p: $x(q_x, q_{\neg x}) \to q_0$ if $\sigma(x) = 0$ and $x(q_{\neg x}, q_x) \to q_1$ if $\sigma(x) = 1$. Therefore, for all $x \in V$ and $p \in \mathcal{P}os(t)$ such that $t|_p = x$, $r(p) = q_{\sigma(x)}$. It is obvious, considering the other rules of $\mathcal{A}$ that there is only one state possible for each other position in r, and that $r(\varepsilon) = q_1$ because σ is a solution. Hence $t \in L(\mathcal{A})$.

Conversely, let r be a successful run of $\mathcal{A}$ on t. The transition rules of $\mathcal{A}$ ensure that t is a representation of the given 3-SAT instance. We show that the

rigidity condition on r ensures that this instance is satisfiable. Let $x \in V$ and $p_1, p_2 \in \mathcal{P}os(t)$ such that $t|_{p_1} = t|_{p_2} = x$. By construction of t, $t|_{p_1.1} = t|_{p_2.1} = 0$ and $t|_{p_1.2} = t|_{p_2.2} = 1$. Only the two transition rules $x(q_x, q_{\neg x}) \to q_0$ and $x(q_{\neg x}, q_x) \to q_1$ can be applied on p_1 and p_2. Assume that $r(p_1) = q_0$, then $r(p_1.1) = q_x$. If $r(p_2) = q_1$, then $r(p_2.2) = q_x$ and since $t|_{p_1.1} \neq t|_{p_2.2}$ it does not respect the rigid condition. So the only possible values are $r(p_2.1) = q_x$, $r(p_2.2) = q_{\neg x}$ and $r(p_2) = q_0$, which respect the rigid condition of both q_x and $q_{\neg x}$. Following the same reasoning, if $r(p_1) = q_1$ then $r(p_2) = q_1$. So, for all $x \in V$, there exists $i_x \in \{0, 1\}$ such that for all $p \in \mathcal{P}os(t)$ such that $t|_p = x$, $r(p) = q_{i_x}$. Hence, by the construction of t and $\mathcal{A}$, it is obvious that the mapping $\sigma(x) = i_x$ is a solution for the 3-SAT instance. □

6.3. Intersection non-Emptiness

Intersection non-emptiness is the problem of deciding, given a finite sequence of RTA whether there exists a term recognized by each RTA of the sequence.

Theorem 10. *Intersection non-emptiness is* EXPTIME*-complete for RTA.*

Proof. The upper-bound is a consequence of Lemma 2, Theorem 1 and Theorem 8. The lower-bound follows from the EXPTIME-hardness of the problem for TA [22]. □

6.4. Universality

Universality is the problem of deciding, given an RTA $\mathcal{A}$ on Σ whether $L(\mathcal{A}) = \mathcal{T}(\Sigma)$.

Theorem 11. *Universality is undecidable for RTA.*

Proof. We reduce the non-existence of a solution of an instance P of the Post Correspondence Problem to the universality of an RTA. This RTA recognizes the set of terms which do not represent a solution of P. It is defined as a disjoint union of RTA, one for each case. Some cases involve the construction of an RTA testing disequalities between unary subterms like in Example 5.

Let Γ be a finite alphabet and $P = (u_i, v_i)_{1 \leq i \leq n}$ be an instance of PCP, with $u_i, v_i \in \Gamma^*$. A solution of P is a finite sequence $i_1, \ldots, i_k$ ($1 \leq i_j \leq n$ for all $j \leq k$) such that $u_{i_1} \ldots u_{i_k} = v_{i_1} \ldots v_{i_k}$.

Let $\Sigma = \{\bot : 0, \} \cup \{a : 1 \mid a \in \Gamma\} \cup \{f_i : 3 \mid 1 \leq i \leq n\}$. For the sake of clarity, a term of the form $a_1(a_2(\ldots a_n(t)))$, with $a_1, \ldots, a_n \in \Gamma$ will be denoted $a_1 a_2 \ldots a_n t$ below.

Every solution of P will be represented by a term of $\mathcal{T}(\Sigma)$. For the definition of this representative, we use two sets U and V of terms of $\mathcal{T}(\Sigma)$ defined recursively as the smallest sets containing $\bot$, V contains all $f_i(u_i\bot, \bot, v_i\bot)$ for $1 \leq i \leq n$, and such that

- if $u \in U$ then $f_i(u_i w_1, u, v_i w_3) \in U$ for all $w_1, w_3 \in \mathcal{T}\big(\Gamma \cup \{\bot\}\big)$,
- if $v \in V \setminus \{\bot\}$ then $f_i(u_i v|_1, v, v_i v|_3) \in V$.

Note that U is regular and $V \subset U$ (but V is not regular). Every solution of P is represented by a term $v \in V \setminus \{\bot\}$ such that $v|_1 = v|_3$.

Given the PCP instance P on Γ, we construct an RTA $\mathcal{A}$ on Σ recognizing the set of terms of $\mathcal{T}(\Sigma)$ which are not a representation of solution of P as above. Therefore, $L(\mathcal{A}) = \mathcal{T}(\Sigma)$ iff P has no solution. The RTA $\mathcal{A}$ is defined as the union of several RTA, each one corresponding to a possible case for a term for not representing a solution of P:

- a TA recognizing the complement of U in $\mathcal{T}(\Sigma)$,
- an RTA recognizing exactly the terms $u \in U \setminus \{\bot\}$ such that $u|_1 \neq u|_3$. This RTA can be constructed as the intersection of a TA recognizing U and an RTA similar to the one of Example 5.
- an RTA recognizing exactly $U \setminus V$, i.e. the terms of U with a subterm at a position 2^k (for some $k \geq 0$) of the form $f_i(u_i w_1, f_j(w'_1, u', w'_3), v_i w_3)$ with $w'_1 \neq w_1$ or $w'_3 \neq w_3$. Again, it is the intersection with a TA for U and the union of two RTA testing disequalities, like in Example 5. □

Together, Theorem 8 and Theorem 11 induce another proof that the class of RTA languages is not closed under complement (the result of Theorem 2).

6.5. Inclusion, Equivalence

Inclusion (resp. equivalence) is the problem of deciding, given two RTA $\mathcal{A}_1$ and $\mathcal{A}_2$ on Σ whether $L(\mathcal{A}_1) \subseteq L(\mathcal{A}_2)$ (resp. $L(\mathcal{A}_1) = L(\mathcal{A}_2)$).

Theorem 12. *Inclusion and equivalence are undecidable for RTA.*

Proof. The equivalence problem is reducible to inclusion. Hence both are undecidable as universality is a particular case of equivalence. □

6.6. Finiteness

Finiteness is the problem of deciding, given an RTA $\mathcal{A}$ on Σ whether $L(\mathcal{A})$ is finite or not. For an RTA $\mathcal{A}$, the finiteness of $L(ta(\mathcal{A}))$ implies the finiteness of $L(\mathcal{A})$, but the converse is not true: the language of the RTA of Example 9 is $\{a, g(g(a))\}$ whereas the language of its underlying TA is $\{a, g^2(a), g^4(a), \ldots\}$.

Theorem 13. *Finiteness is decidable in* PTIME *for RTA.*

Proof. Like for TA [11], checking finiteness amounts to detecting (in PTIME) some loops and paths in the accessibility graph of an RTA. The accessibility graph of a given RTA $\mathcal{A} = \langle Q, R, F, \Delta \rangle$ is an oriented graph $G_{\mathcal{A}} = \langle Q, E_{\mathcal{A}} \rangle$ whose set of vertexes is Q and set of edges is $E_{\mathcal{A}} := \{\langle q, q' \rangle \mid \exists f(\ldots q \ldots) \to q' \in \Delta\}$. A path in $G_{\mathcal{A}}$ is a finite sequence of states $q_1, \ldots, q_n$ such that $\langle q_i, q_{i+1} \rangle \in E_{\mathcal{A}}$ for all $1 \leq i < n$. We have that $L(\mathcal{A})$ is infinite iff there exists a state $q \in Q \setminus R$ such that $L(\mathcal{A}, q) \neq \emptyset$, a loop on q in $G_{\mathcal{A}}$ (path starting and ending with q) whose states are all in $Q \setminus R$, and a path in $G_{\mathcal{A}}$ starting with q and ending with a final state of F.

The *if* direction is easy. The other direction can be shown with arguments similar as those in the proof of Lemma 1. If $L(\mathcal{A})$ is infinite then it contains a term t of depth larger than $(|Q|+1)|R|$. The idea is that the loop on q is the path from the variable position up to the root of the context D in a successful run r of $\mathcal{A}$ on t, and the path from q to a final state is the path from the root of D up to the root of t in r.

Checking that $L(\mathcal{A}, q) \neq \emptyset$ can be done in linear time according to Theorem 8, and deciding the existence of the loop and the path can both be done in polynomial time in the size of $\mathcal{A}$. Altogether, the finiteness of $L(\mathcal{A})$ can be checked in polynomial time. □

7. Rewrite Closure

Following the motivations presented in the introduction, we study here the closure under term rewriting of RTA languages. We observe first that in general, the rewrite closure of an RTA language is not an RTA language (Section 7.1) and it is even not recursive (Section 7.2) for linear and collapsing TRS. This is in contrast with TA languages, which are closed under rewriting with such TRS [25].

We show next that, under a syntactical restriction, namely for a linear and so called *inverse visibly pushdown* TRS $\mathcal{R}$, it is decidable whether a given tree belongs to the rewrite closure of a given RTA language (Section 7.3).

7.1. Linear and Collapsing Rewrite Systems

We show first that the closure of an RTA language under rewriting is not an RTA language, even for a very restricted class of TRS.

Proposition 6. *In general $\mathcal{R}^*(L)$ is not an RTA language when L is an RTA language and $\mathcal{R}$ a linear and collapsing TRS.*

Proof. Let $\Sigma = \{h:2, f:1, g:1, 0:0\}$, let $\mathcal{R} = \{f(g(x)) \to x\}$, and let $\mathcal{A} = \langle Q, R, F, \Delta\rangle$ be the RTA on Σ with $Q = \{q_0, q_1, q_2, q_r, q_{\mathsf{f}}\}$, $R = \{q_r\}$, $F = \{q_{\mathsf{f}}\}$, and

$$\Delta = \left\{ \begin{array}{l} 0 \to q_0, g(q_0) \to q_0 | q_r, f(q_r) \to q_1, f(q_1) \to q_1, \\ h(q_r, q_{1,2}) \to q_{\mathsf{f}}, h(q_{1,2}, q_{1,2}) \to q_2, h(q_{\mathsf{f}}, q_{1,2}) \to q_{\mathsf{f}}, \end{array} \right\}$$

where $q_{1,2}$ is either q_1 or q_2. Every term of $L(\mathcal{A})$ has the form $H\big[g^m(0), f^*(g^m(0)), \ldots, f^*(g^m(0))\big]$ where H is an k-context made of the symbol h only (with $k \geq 2$), g^m represents a nesting of m symbols g and f^* represents a nesting of an arbitrary number of f's. In other word, the leftmost argument of the context H contains m symbols g, and the other arguments of the context consist in an arbitrary number of f's followed by m g's and finished by a 0. Indeed, the rigid state q_r enforces that each argument has the same number of g's.

The terms of the closure $\mathcal{R}^*\big(L(\mathcal{A})\big)$ of $L(\mathcal{A})$ by $\mathcal{R}$ have a similar form except that the number of g's in the different arguments might not be equal. They only

have to be all less than or equal to the number of g's in the leftmost argument. The intersection of this set $\mathcal{R}^*\big(L(\mathcal{A})\big)$ with the regular tree language containing the terms of the form $H\big[g^*(0),\ldots,g^*(0)\big]$ is the language of Example 6, which is not recognized by RTA. It follows that $\mathcal{R}^*\big(L(\mathcal{A})\big)$ is not an RTA language. □

Note that the terms $H\big[g^m(0), g^{n_1}(0),\ldots,g^{n_k}(0)\big]$ in the language of Example 6 are $\mathcal{R}$-normal forms in the above counterexample in the proof of Proposition 6 (since they do not contain the symbol f). Hence, restricting to the terms of the rewrite closure in normal form does not help: the intersection of $\mathcal{R}^*\big(L(\mathcal{A})\big)$ with $\mathcal{R}$-normal-forms is not an RTA language in general, when $\mathcal{A}$ is an RTA and $\mathcal{R}$ a linear and collapsing TRS.

7.2. Undecidability of Membership Modulo

We show in this section that the rewrite closure of an RTA under a linear collapsing TRS is even not recursive. Let us call *membership modulo* the problem of deciding whether $t \in \mathcal{R}^*(L(\mathcal{A}))$ given an RTA $\mathcal{A}$, a TRS $\mathcal{R}$ and a ground term $t \in \mathcal{T}(\Sigma)$.

Theorem 14. *Membership modulo is undecidable for RTA and linear and collapsing TRS.*

Proof. Let Γ be a finite alphabet and let $P = (u_i, v_i)_{1\le i\le n}$ be an instance of PCP, with $u_i, v_i \in \Gamma^*$. A solution of P is a sequence $i_1,\ldots,i_m$ of integers smaller or equal to n such that $u_{i_1}\ldots u_{i_m} = v_{i_1}\ldots v_{i_m}$.

Let us consider the signature $\Sigma = \{g_i : 1, f_i : 1 \mid i \le n\} \cup \{a : 1 \mid a \in \Gamma\} \cup \{0 : 0, k : 1, h : 2\}$, and the language (for all $w = a_1,\ldots,a_p \in \Gamma^*$, the term $a_1(\ldots a_p(t))$ is written $w(t)$)

$$L = \{h(s, k(s)) \mid s = f_{i_m}(g_{i_m}(\ldots f_{i_1}(g_{i_1}(w(0)), 1 \le i_1,\ldots,i_m \le n, m > 0, w \in \Gamma^*\}$$

Let $\mathcal{R}$ be a TRS on Σ containing the rules

$$\begin{array}{rcll} f_i(g_i(u_i(x))) & \to & x & (i \le n), \\ g_i(x) & \to & x & (i \le n), \\ g_j(f_i(v_i(x))) & \to & x & (i, j \le n), \\ k(f_i(v_i(x))) & \to & x & (i \le n). \end{array}$$

The tree language L is recognizable by an RTA on Σ. Hence the following property permits us to conclude the proof of Theorem 14.

Lemma 3. *$h(0,0) \in \mathcal{R}^*(L)$ iff P has a solution.*

The if direction is easy. Let $i_1,\ldots,i_m$ be a solution of P and let $h(s, k(s))$ be the term of L corresponding to this solution (i.e. $s = f_{i_m}(g_{i_m}(\ldots f_{i_1}(g_{i_1}(w(0)))))$ and $w = u_{i_1}\ldots u_{i_m}(0) = v_{i_1}\ldots v_{i_m}(0)$). This term is depicted in Figure 4. The s in the left branch can be reduced to 0 using the first rule of $\mathcal{R}$, and the s in the second branch can be reduced to $k(f_{i_m}(v_{i_m}(0)))$ using the two next rules of $\mathcal{R}$. This latter term is in turn reduced to 0 using the last rule of $\mathcal{R}$.

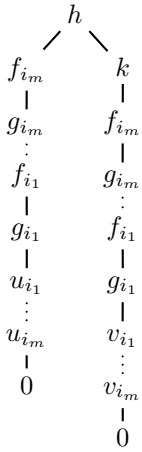

Figure 4: Undecidability of membership modulo: encoding of a solution of PCP.

For the only if direction, assume that $L \ni h(s, k(s)) \xrightarrow[\mathcal{R}]{*} h(0, 0)$. In order to show that in this case, s corresponds to a solution, it is sufficient to make the following observations. First, only the first rule of $\mathcal{R}$ (with u_i) can be applied in order to reduce the s in the left branch to 0. Indeed, the only other rule of $\mathcal{R}$ applicable to s is $g_i(x) \to x$ and after using this rule, only $g_j(f_i(v_i(x))) \to x$ can be applied, and s cannot be reduced to 0. Moreover, assuming s minimal, and having the k at the top of the right branch imposes us to use only the last three rules of $\mathcal{R}$ in order to reduce $k(s)$ to 0 (it is possible to start the reduction of the right branch with a sequence of applications of $f_i(g_i(u_i(x))) \to x$ but this would contradict the minimality of s). Altogether, it follows that s corresponds to a solution of P. □

7.3. Linear and Invisibly Pushdown Rewrite Systems

We show in this section that the problem of membership modulo becomes decidable with some further syntactic restrictions on $\mathcal{R}$, based on the definition of visibly pushdown automata [23].

The VPA (see also Section 5.2) recognize languages of words. They were generalized into tree recognizers in [5, 16]. In [5], Chabin and Rety show that the class of visibly pushdown tree automata (VPTA) languages is closed under rewriting with so called linear visibly context-free TRS. We use a similar definition in order to characterize a class of TRS modulo which membership is decidable for RTA. In the following definition, we assume a partition of the signature Σ into $\Sigma_c \uplus \Sigma_r \uplus \Sigma_\ell$.

Definition 5. A collapsing TRS $\mathcal{R}$ is called *inverse-visibly pushdown* (invisibly pushdown) if for every rule $\ell \to x \in \mathcal{R}$, $d(\ell) \geq 1$, x occurs once in ℓ, and if x occurs at depth 1 in ℓ then $\ell \in \mathcal{T}(\Sigma_\ell, \mathcal{X})$, otherwise, $\ell(\varepsilon) \in \Sigma_c$, the symbol immediately above x is in Σ_r and all the other symbols of ℓ are in Σ_ℓ.

Example 11. The TRS $\mathcal{R} = \{\mathsf{fst}(\mathsf{pair}(x_1, x_2)) \to x_1, \mathsf{snd}(\mathsf{pair}(x_1, x_2)) \to x_2, \mathsf{dec}(\mathsf{inc}(x)) \to x, \mathsf{decrypt}(\mathsf{crypt}(x, \mathsf{pk}(\mathsf{A})), \mathsf{sk}(\mathsf{A})) \to x\}$ is linear and invisibly pushdown with $\Sigma_c = \{\mathsf{fst}, \mathsf{snd}, \mathsf{dec}, \mathsf{decrypt}\}$ and $\Sigma_r = \{\mathsf{pair}, \mathsf{inc}, \mathsf{crypt}\}$, $\Sigma_\ell = \{\mathsf{pk}, \mathsf{sk}, \mathsf{A}\}$. $\diamond$

The TRS in the proof of Proposition 6, $\{f(g(x)) \to x\}$, is invisibly pushdown, but not the one in the proof of Theorem 14. Indeed, there exists no partition of Σ making this latter TRS invisibly pushdown. According to Definition 5, having a rule $g_i(x) \to x$ implies that $g_i \in \Sigma_\ell$ but also having a rule $g_j(f_i(v_i(x))) \to x$ implies that $g_i \in \Sigma_c$.

Theorem 15. *Membership modulo is decidable for RTA and linear and invisibly pushdown TRS.*

Proof. The decision algorithm involves the construction of a visibly pushdown automata recognizing the language of ancestors of t wrt $\mathcal{R}$ that belong to $L(\mathcal{A})$. We do this in three steps:

1. we compute a (big) context-free tree grammar that generates all terms that can rewrite to a single variable by $\mathcal{R}$ rules, such that there exists a run of $\mathcal{A}$ on them where the positions of the rigid states are not contradictory.
2. we add initial rules to the grammar in order to make it generate terms that rewrite to t instead of those rewriting to a variable.
3. finally, we transform the tree grammar obtained into a visibly pushdown tree automaton, and take the subterms under rigid states as independent languages. We replace each language under an occurrence of a rigid state by the intersection of all languages under all occurrences of the same rigid state.

After these constructions have been completed, we only need to check whether the given visibly pushdown language is empty or not in order to solve the problem.

Let $\mathcal{A} = \langle Q, R, F, \Delta \rangle$ be an RTA, $\mathcal{R}$ an inverse-visibly pushdown TRS and $t \in \mathcal{T}(\Sigma)$ a term in normal form. We want to construct a CF tree grammar which simulates the application of the rules of $\mathcal{R}$ backwards, by expanding subterms into left-hand side of rules, in a way that the application of rules of $\mathcal{A}$ is possible. For this purpose, we shall use some tuples of the following form $\langle \frac{q_1}{q_2}, \ell, lbl, occ, occ_x, < \rangle$ where

- $q_1, q_2 \in Q$,
- ℓ is either the lhs of some rule $\ell \to x \in \mathcal{R}$, or the single variable x,
- lbl is a labeling of ℓ by pairs of states denoted $\frac{q}{q'}$,
- occ is a set of pairs $\langle q_r, p \rangle$ where $q_r \in R$ and $p \in \mathcal{P}os(\ell)$,
- occ_x is a subset of occ where each rigid state occurs in at most one pair,
- $<$ is a strict partial order on the set of rigid states R.

For each tuple ψ of this form, we will denote $lq(\psi)$ for q_1, $rq(\psi)$ for q_2 and $pair(\psi)$ for $\frac{q_1}{q_2}$. When the use of lbl, occ, occ_x or $<$ is ambiguous we will index them with the tuple they are referencing (ex. occ^ψ). W.l.o.g. we will assume that the rule of $\mathcal{R}$ in which ℓ occurs rewrites to the variable x, and we will denote p_x the position of this variable in ℓ. For the rest of the proof, we will need the following definition of a *valid labelling* of a term by those tuples.

Definition 6. A labelling ξ of a term t by tuples of the form $\langle \frac{q_1}{q_2}, \ell, lbl, occ, occ_x, < \rangle$ is said to be *valid* if

1. $\forall p \in \mathcal{P}os(t)$, $t(p)(lq(\xi(p.1)), \ldots, lq(\xi(p.n))) \to rq(\xi(p)) \in \Delta$,
2. there do not exist two rigid states q_{r_1}, q_{r_2} and two positions p, p' such that $q_{r_1} <^{\xi(p)} q_{r_2}$ and $q_{r_2} <^{\xi(p')} q_{r_1}$,
3. there do not exist two positions p and $p.w$ such that a rigid state q_r appears in $occ_x^{\xi(p)}$ and in $occ^{\xi(p.w)}$,
4. there do not exist three positions $p, p.w$ and p' such that a rigid state q_{r_1} appears in $occ_x^{\xi(p)}$, a distinct rigid state q_{r_2} appears in $occ^{\xi(p.w)}$, and $q_{r_1} <^{\xi(p')} q_{r_2}$.

We will build a set T of tuples by induction. We start with

$$T_0 = \{\langle \frac{q}{q}, x, \frac{q}{q}, \emptyset, \emptyset, \emptyset \rangle \mid q \in Q \setminus R\} \cup \{\langle \frac{q_r}{q_r}, x, \frac{q_r}{q_r}, \{(q_r, \varepsilon)\}, \{(q_r, \varepsilon)\}, \emptyset \rangle \mid q_r \in R\}.$$

The set T_{i+1} is built from T_i as follows

- for every $\psi \in T_i$, we add ψ to T_{i+1},
- for every lhs ℓ of $\mathcal{R}$ and every valid labelling ξ of ℓ by tuples of T_i, we add the tuple $\langle \frac{q_1}{q_2}, \ell, lbl, occ, occ_x, < \rangle$ to T_{i+1}, where:
  - $q_1 = lq(\xi(\varepsilon))$, $q_2 = rq(\xi(p_x))$,
  - $\forall p \in \mathcal{P}os(\ell), lbl(p) = \frac{lp(\xi(l))}{rp(\xi(l))}$,
  - $occ = \{\langle q_r, p \rangle \mid q_r$ appears in $occ^{\xi(p)}\}$,
  - $occ_x = \{\langle q_r, p \rangle \mid p \leq p_x$ and q_r appears in $occ_x^{\xi(p)}\}$,
  - $q_{r_1} < q_{r_2}$ iff $\exists p, q_{r_1} <^{\xi(p)} q_{r_2}$ or $\exists p, p.w \in \mathcal{P}os(\ell), q_{r_1}$ appears in $occ_x^{\xi(p)}$ and q_{r_2} appears in $occ^{\xi(p.w)}$.

By induction, each tuple $\langle \frac{q_1}{q_2}, \ell, lbl, occ, occ_x, < \rangle$ added in T verifies

1. q_1 is the top state of $lbl(\varepsilon)$,
2. q_2 is the bottom state of $lbl(p_x)$,
3. $<$ is a partial strict order on R,
4. $occ_x \subseteq occ$,
5. each position in occ_x is a prefix of p_x.

Since the number of lhs of $\mathcal{R}$, the size of each lhs, the number of (rigid) states of $\mathcal{A}$ are all finite, each step takes a finite (in fact polynomial) amount of time. Also, the number of distinct tuples that can be added in some T_i is also finite, so we will eventually reach a set T_i where no new tuple can be added. We define T as the first T_i where no new tuple can be added.

The terminal symbols of our CF tree grammar are the function symbols of Σ. Its non-terminals symbols are elements of $\mathcal{N} = \Sigma \cup \{\top\} \times T$, and $\langle f, \psi \rangle$ has arity n if $f \in \Sigma_n$ and every $\langle \top, \psi \rangle$ has arity zero.

Let us first define the main production rules of the grammar: for every $f \in \Sigma \cup \top$ and every tuple $\phi = \langle \frac{q_1}{q_2}, \ell, lbl, occ, occ_x, < \rangle$,
– if $\ell \neq x$, then we have in the grammar all the production rules:

$$\langle f, \psi \rangle(x_1, \ldots, x_n) := u$$

where, $f \in \Sigma_n$ and u is a term of $\mathcal{T}\big(\Sigma \cup \mathcal{N}, \{x_1, \ldots, x_n\}\big)$ such that $\mathcal{P}os(u) = \mathcal{P}os(\ell)$ and defined by, for every position p, $u(p) = \langle f', \psi_p \rangle$ with

- $\psi_p \in T$
- $pair(\psi_p) = lbl(p)$
- if $\langle q_r, p \rangle \in occ$, then q_r occurs in occ^{ψ_p}
- if $\langle q_r, p \rangle \in occ_x$, then q_r occurs in $occ_x^{\psi_p}$
- $<^{\psi_p} \subseteq <$

By construction of T there exists a tuple ψ satisfying these conditions. Moreover,

- $f' = \ell(p)$ if $\ell(p) \in \Sigma$ (i.e. if p is not a variable position in ℓ),
- $f' = f(x_1, \ldots, x_n)$ if $p = p_x$ (the position of x in ℓ),
- $f' = \top$ elsewhere.

– if $\ell = x$, then $q_1 = q_2 = q$, and we add to our grammar the production rule

$$\langle f, \psi \rangle(x_1, \ldots, x_n) := f(x_1, \ldots, x_n)$$

if $f \neq \top$. We also add to the tree grammar some non-terminal of arity zero and production rules that generates the terms of $L(A, q)$, which is a regular language.

With this construction, the rules of our CF tree grammar generate the terms that rewrite to a single variable x with $\mathcal{R}$, and that have a run r of $\mathcal{A}$ on them and where positions of rigid states are not contradictory with the rigid conditions. But we still need to ensure that we generate terms that rewrite to t instead of x and that subterms under rigid states are equal.

In order to generate terms rewriting to t instead of x, we just need to add initial rules to the grammar. Let S be the initial non-terminal symbol (of arity

zero). For each valid labelling ξ of t such that $lq(\xi(\varepsilon))$ is a finite state of $\mathcal{A}$, we add to our grammar the production rule:

$$S := u$$

where $u(p) = (t(p), \xi(p))$ for all $p \in \mathcal{P}os(t)$.

The CF tree grammar constructed generates all the terms rewriting to t with $\mathcal{R}$ and with a run of $\mathcal{A}$ that have non-contradictory positions of rigid states. Only the rigidity condition is missing.

For the rigidity condition, we need to compare the languages generated by the grammar's production rules, starting from the non-terminal symbols of the form $\langle f, \psi \rangle$ with $\psi = \langle \frac{q_r}{q_r}, x, \frac{q_r}{q_r}, \{\langle q_r, \varepsilon \rangle\}, \{\langle q_r, \varepsilon \rangle\}, < \rangle$ for some rigid state $q_r \in R$. Let us call such languages the language of the grammar associated to q_r. For this purpose, we use the fact that $\mathcal{R}$ is a linear and invisibly pushdown rewrite system. Indeed, it ensures that the above languages of the grammar associated to rigid states are languages of visibly pushdown tree automata (VPTA). Such languages are closed under intersection, and the emptiness is decidable.

We consider the languages of the grammar associated to rigid states, beginning by the maximal rigid states according to the partial order. We compute the intersection of every language that can be generated at different occurrences of a same rigid state. We do that for each rigid state. Then, the intersection language of the minimal rigid states (according to the partial order) is used in the languages of greater rigid sates and in the general language of ancestors of t instead of the different languages of the different occurrences. We repeat this procedure, following the partial order, until having replaced each language of an occurrence of a rigid state by the corresponding intersection. Finally, we just have to decide the emptiness of the general language to know whether a term recognized by $\mathcal{A}$ (with a run respecting the rigidity condition for all rigid states) does rewrite to t. □

8. Application to the Verification of Security Protocols

In this final section, we would like to present an application of RTA to the verification of security protocols; this application was actually our original motivation for studying rigid tree automata. Our purpose is not to propose new results in this domain, but rather to illustrate the potential of RTA for the automatic verification of some infinite state systems, in particular communicating processes.

Using automata for protocol analysis is a quite popular approach, see e.g. [3, 8, 21]. In particular it is possible to analyze protocols with infinitely many sessions. But this kind of analyses has limitations due to approximations with regular sets. Such approximations may conduct to false alarms, as discussed e.g. in [26] or [19]. The approach with RTA overcomes several sources of imprecisions such as incorrect chaining of messages sent by agents, or ignoring the multiple occurrences of variables in the body of messages sent. Moreover, rigid state also permit to model a local finite memory in which both honest and dishonest

agents can store read messages. This feature is generally not supported in other models.

8.1. Protocol Model

We consider a model of security protocols where a finite number of agents exchange messages, following a protocol, asynchronously over an insecure network. The messages are ground terms of $\mathcal{T}(\Sigma)$ build over cryptographic operators and are interpreted modulo an invisibly TRS $\mathcal{R}$ with rules like the above one for $\mathsf{decrypt}$. For instance, we assume that Σ contains the binary operator crypt for encryption of data (in the first argument) with a public or secret key (in the second argument), and $\mathsf{decrypt}$ for the decryption with the associated secret key. We also use two unary operators pk and sk, associating to the name of an agent its public, respectively secret, key. For the name of agents, we use a finite set of constant function symbols of Σ_0, A, B... Amongst the set of function symbols Σ, we distinguish a subset Σ_{pub} of *public function symbols*, which represent the function publicly known. Below we assume that Σ_{pub} contains crypt and $\mathsf{decrypt}$, all the agent's names, the function pk (hence we assume that knowing somebody's name is sufficient to know his public key), a function inc for incrementation, but Σ_{pub} does not contain the function sk.

We have the following rewrite rules in $\mathcal{R}$ (for each agent's name A)

$$\mathsf{decrypt}(\mathsf{crypt}(x, \mathsf{pk}(A)), \mathsf{sk}(A)) \to x$$

(a message x encrypted using the function crypt with the public key of A, $\mathsf{pk}(A)$, can be recovered using $\mathsf{decrypt}$ and the secret key of A, $\mathsf{sk}(A)$).
We also have the symmetric rules

$$\mathsf{decrypt}(\mathsf{crypt}(x, \mathsf{sk}(A)), \mathsf{pk}(A)) \to x$$

We also consider a binary constructor pair and two unary operators fst and snd for pairing and projection, and the associated rewrite rules

$$\mathsf{fst}(\mathsf{pair}(x_1, x_2)) \to x_1, \quad \mathsf{snd}(\mathsf{pair}(x_1, x_2)) \to x_2$$

As noticed in Section 7.3, the TRS containing the above rules is linear invisibly (see Example 11).

Example 12. We consider as a running example a simplified version of the mutual authentication protocol `SPLICE/AS` [27], which consists of the following two messages exchanged between a client C and a server S.

$$\begin{array}{llllll} 1. & C & \to & S & : & \mathsf{pair}\big(\mathsf{pair}(C, S), \mathsf{crypt}(\mathsf{pair}(C, \mathsf{crypt}(N, \mathsf{pk}(S))), \mathsf{sk}(C))\big) \\ 2. & S & \to & C & : & \mathsf{pair}\big(\mathsf{pair}(S, C), \mathsf{crypt}(\mathsf{pair}(S, \mathsf{inc}(N)), \mathsf{pk}(C))\big) \end{array}$$

The purpose of this protocol is to establish a handshake between C and S: C sends to S some integer value N, encrypted with crypt and the public key of S, $\mathsf{pk}(S)$. Then, S sends to C in reply the successor of N, $\mathsf{inc}(N)$, encrypted with crypt and the public key of C, $\mathsf{pk}(C)$, in order to prove that he was the

real receiver of the first message – since only S has the secret key $\mathsf{sk}(S)$ which is necessary in order to recover N from $\mathsf{crypt}(N, \mathsf{pk}(S))$.

Moreover, in the first message, C further encrypts $\mathsf{crypt}(N, \mathsf{pk}(S))$ using the function crypt and its secret key of C, $\mathsf{sk}(C)$. The purpose of this second step of encryption is to act as a signature: only C is supposed to know his secret key, and the receiver of the message S, who knows C's public key $\mathsf{pk}(C)$ can check whether this part of the message was really encrypted with $\mathsf{sk}(C)$.

Finally, some more information is wrapped in the messages: C recalls his name in the signed part of the first message, for the purpose of the double check of the signature described above; moreover both messages start with a "header", i.e. a pair containing the name of the sender and the intended receiver.

The original `SPLICE/AS` protocol [27] contains additional messages for the distribution of public keys $\mathsf{pk}(S)$ and $\mathsf{pk}(S)$ by a trusted authority `AS`, and timestamps. Here, we make the simplifying assumption that every public key is known by everyone (since everyone can obtain it from `AS`), and we skip the timestamps. ◇

We consider a simple formal representation of programs executing cryptographic protocols which should fit with most of the formalisms in use.

A *program* is a finite set of agents, and each agent is a finite sequence of pairs of *instructions* of the form $\mathsf{recv}(x).\mathsf{send}(s).i$ where i is a program point (in an arbitrary domain), $x \in \mathcal{X}$ and $s \in \mathcal{T}(\Sigma, \mathcal{X})$. We assume that moreover every agent starts with an initial program point and that all the program points in a program are pairwise distinct. Note that every message is received as a variable (the argument of recv is always a variable). Hence recv acts as a variable binder, like in [9]. Every agent is supposed to be closed, i.e. every variable x occurring in a $\mathsf{send}(s)$ is in the scope of a binder $\mathsf{recv}(x)$. For convenience, we assume that the variables in different instructions $\mathsf{recv}(x)$ of a program are distinct.

Example 13. An example of program executing the simplified version of the `SPLICE/AS` protocol of Example 12 is made of the two following agents called C and S.

$$\begin{array}{ll} C: & c_0.\mathsf{recv}(x).\mathsf{send}\big(\mathsf{pair}\big(\mathsf{pair}(C,S), \mathsf{crypt}(\mathsf{pair}(C, \mathsf{crypt}(N, \mathsf{pk}(S))), \mathsf{sk}(C))\big)\big).c_1 \\ S: & s_0.\mathsf{recv}(y).\mathsf{send}\big(\mathsf{pair}\big(\mathsf{pair}(t_s, t_c), \mathsf{crypt}(\mathsf{pair}(t_s, \mathsf{inc}(t_n)), \mathsf{pk}(t_c))\big)\big).s_1 \\ \text{where } t_s = \mathsf{snd}(\mathsf{fst}(y)), t_c = \mathsf{fst}(\mathsf{fst}(y)), & \\ \qquad t_n = \mathsf{decrypt}\big(\mathsf{snd}\big(\mathsf{decrypt}(\mathsf{snd}(y), \mathsf{pk}(t_c))\big), \mathsf{sk}(S)\big) & \end{array}$$

The terms t_s, t_c and t_n describe the recipes used by S to recover respectively the values S, C and N from the message received y. The variable x is useless. It is only for technical purpose that we assume that C receives an arbitrary value before sending the initial message of the protocol. ◇

In order to define a semantics for the execution of these programs, we describe in the next section a model of the network used for the communications.

8.2. Protocol Semantics

The network is assumed to be under the control of an active attacker who is able to read and divert messages and to sent newly forged messages under fake identities. The attacker is able to use terms and function symbols that he knows (like terms sent by honest processes and public functions such as crypt or $\mathsf{decrypt}$), in order to forge new messages.

To summarize, in this network model, the communication of one message m between two agents C and S can be decomposed into three phases:

- C sends the message m to the attacker (send instruction),
- the attacker analyze the message as much as he can (applying public functions like $\mathsf{decrypt}$, known public keys like $\mathsf{pk}(S)$ and the rewrite rules of $\mathcal{R}$) and possibly changes m into m' (but m' may be equal to m),
- the attacker transfers m' to S, pretending that the sender is C, and S reads m' (recv instruction).

Later, S may reuse m' in order to prepare an answer to C, following the rules of the protocol.

A *configuration* of a program P is a triple (S, σ, N) where S is a set of programs points (one for each agent), σ is a substitution whose domain is the variables of P and codomain is a subset of $\mathcal{T}(\Sigma)$ and $N \subset \mathcal{T}(\Sigma)$. Intuitively, S contains the current program point of each agent of P, σ is the list of messages read by the agents so far with instructions $\mathsf{recv}(x)$, and N represents the set of terms known by the attacker. Hence, according to the above hypotheses, N corresponds to the content of the network (at a step of execution defined by S and σ) i.e. it is the set of all terms which can be read (with $\mathsf{recv}(x)$) by the agents.

We define now small step semantics for the execution of programs. Each step changes the running configuration $\big(S, \sigma, N\big)$ of P into $\big(S', \sigma', N'\big)$ if $S = \{i\} \cup U$, the program point i appears in one agent of P (this agent is unique by assumption that the program points are pairwise distinct) and in this agent, i is followed by the instructions $\mathsf{recv}(x).\mathsf{send}(s).i'$, x is not in the domain of σ and

- $S' = \{i'\} \cup U$,
- $\sigma' = \sigma \cup \{x \mapsto m\}$ for some $m \in N$,
- N' is the closure of $cl(N \cup \{\sigma(s)\})$ under application of public function symbols and $\mathcal{R}$, as defined below.

The closure $cl(M)$ of a set $M \subseteq \mathcal{T}(\Sigma)$ is defined recursively as the smallest set containing all the terms of M and such that for all $f \in \Sigma_{\mathsf{pub}}$ of arity n, for all $t_1, \ldots, t_n \in cl(M)$, $\mathcal{R}^*\big(\{f(t_1, \ldots, t_n)\}\big) \subseteq cl(M)$. Intuitively, $cl(M)$ represents the set of terms than the attacker can deduce from the terms of M.

Example 14. The following sequence of configurations represents a valid execution of the program in Example 13, where the agent's messages are smoothly transferred without tampering.

$$(\{c_0, s_0\}, \emptyset, N_0), (\{c_1, s_0\}, \{x \mapsto t_0\}, N_1), (\{c_1, s_1\}, \{x \mapsto t_0, y \mapsto s_0\}, N_2)$$

The set N_0 in the first configuration contains the initial knowledge of the attacker. For instance, N_0 contains A, $\mathsf{pk}(A)$, $\mathsf{sk}(A)$ (A is the official identity of the attacker), the identity of other agents and their public keys C, S, $\mathsf{pk}(C)$, $\mathsf{pk}(S)$... and the terms build with these terms by application of the public function symbols pair, fst, snd, inc, crypt, $\mathsf{decrypt}$, e.g. $\mathsf{pair}(A, S)$, $\mathsf{pair}(A, \mathsf{pair}(A, S))$... In other word,

$$N_0 = cl\big(\{A, \mathsf{pk}(A), \mathsf{sk}(A), C, S, \mathsf{pk}(C), \mathsf{pk}(S)\}\big).$$

Note that this set N_0 is infinite but regular.
The term t_0 is an arbitrary element of N_0, and

$$\begin{array}{rcl}
N_1 & = & cl(N_0 \cup \{s_0\}), \\
s_0 & = & \mathsf{pair}\big(\mathsf{pair}(C, S), \mathsf{crypt}(\mathsf{pair}(C, \mathsf{crypt}(N, \mathsf{pk}(S))), \mathsf{sk}(C))\big), \\
N_2 & = & cl(N_1 \cup \{s_1\}), \\
s_1 & = & \mathsf{pair}\big(\mathsf{pair}(t_s, t_c), \mathsf{crypt}(\mathsf{pair}(t_s, \mathsf{inc}(t_n)), \mathsf{pk}(t_c))\big), \\
t_s & = & \mathsf{snd}(\mathsf{fst}(s_0)), t_c = \mathsf{fst}(\mathsf{fst}(s_0)), \\
t_n & = & \mathsf{decrypt}\big(\mathsf{snd}\big(\mathsf{decrypt}(\mathsf{snd}(s_0), \mathsf{pk}(t_c))\big), \mathsf{sk}(S)\big).
\end{array}$$

Note that in the sequence, the two agents exchange the messages of the protocol, as described in Example 12, because $s_1 \xrightarrow[\mathcal{R}]{*} \mathsf{pair}\big(\mathsf{pair}(S, C), \mathsf{crypt}(\mathsf{pair}(S, \mathsf{inc}(N)), \mathsf{pk}(C))\big)$. $\diamond$

With the above semantics, every message is build on the top of former messages (either by an agent or the attacker). The monotonicity of the definition of the messages sets N_i makes bottom-up RTA suitable for their representation.

8.3. RTA construction

We show below that it is possible to build an RTA $\mathcal{A}$ recognizing exactly the sets of messages N (representing the state of the attacker's knowledge) in reachable configurations of a given program P. By reachable, we mean reachable from an initial configuration, which is specified precisely below and is assumed to be part of the problem. The RTA $\mathcal{A}$ models both the behaviour of the honest agents and of the attacker. It uses uses one rigid state to memorize every message received by the honest agents (the codomain of the substitution σ in configuration). The first component of configurations (program points of all agent) is encoded directly into the states (as the amount of information needed is finite).

Assume that the program P contains n agents $P_1, \ldots, P_n$. We detail below the construction of the states and transitions of $\mathcal{A}$. We have in $\mathcal{A}$ one state $q_{i_1 \ldots i_n}$ for each tuple of values $(i_1, \ldots, i_n)$ of program points of respectively $P_1, \ldots, P_n$.

Example 15. For the program of Example 13 (for the simplified version of the SPLICE/AS protocol presented in Example 12), with two agents called C and S, we have the states $q_{c_0 s_0}$, $q_{c_1 s_0}$, $q_{c_0 s_1}$, $q_{c_1 s_1}$. $\diamond$

Intuitively, $\mathcal{A}$ will be such that $L(\mathcal{A}, q_{i_1 \ldots i_n})$ is the set of messages N such that a configuration $(\{i_1 \ldots i_n\}, \sigma, N)$ is reachable, for some σ. Hence this language contains the set of terms readable (at this point) by the agents and the attacker.

For each state $q_{i_1 \ldots i_n}$ and each (bound) variable x occurring in P, we consider one copy denoted $q^x_{i_1 \ldots i_n}$, which is a rigid state.

Example 16. For the program of Example 13, we have the rigid states $q^x_{c_0 s_0}$, $q^y_{c_0 s_0}$, $q^x_{c_1 s_0}$, $q^y_{c_1 s_0} \ldots$ Intuitively, $\mathcal{A}$ will be such that $L(\mathcal{A}, q^y_{c_1 s_0})$ is exactly the set of terms t such that there exists a reachable configuration $(\{c_1, s_0\}, \{x \mapsto t_0, y \mapsto t\}, N)$, for some t_0 and N. $\diamond$

Now, we will describe the transitions of $\mathcal{A}$ modeling the operations recv and send of the agents. The idea is that when an agent P_i has an instruction $\mathsf{send}(s)$, then $\mathcal{A}$ will perform pattern matching of s, using transitions similar to the ones described in the construction of Proposition 1. Like in Proposition 1, we consider for this purpose some auxiliary states of the form $q^u_{i_1 \ldots i_n}$ for every strict subterm u of s and tuple $(i_1, \ldots, i_n)$ of program points values. Note that for every variable $x \in \mathit{vars}(s)$, one rigid states $q^x_{i_1 \ldots i_n}$ has already been added above.

Let $(i_1, \ldots, i_n)$ be a tuple of program point values, such that i_j occurs in the agent P_j and is followed by the instructions $\mathsf{recv}(x).\mathsf{send}(s).i'_j$. Then the following transitions are added to $\mathcal{A}$ for the recognition of s (like in Proposition 1, we assume that s is not a variable):

$$\begin{array}{ll} g(q^{u_1}_{i_1 \ldots i_n}, \ldots, q^{u_m}_{i_1 \ldots i_n}) \to q^{g(u_1, \ldots, u_m)}_{i_1 \ldots i_n} & \text{s.t. } g \in \Sigma_m, g(u_1, \ldots, u_m) \text{ strict subterm of } s, \\ g(q^{u_1}_{i_1 \ldots i_n}, \ldots, q^{u_m}_{i_1 \ldots i_n}) \to q_{\vec{i'}} & \text{s.t. } g(u_1, \ldots u_m) = s, \vec{i'} = i_1 \ldots i_{j-1} i'_j i_{j+1} \ldots i_n. \end{array}$$

Note that since the states $q^x_{i_1 \ldots i_n}$ are rigid (when x is a variable), the non linearities in s are respected.

Example 17. Let us consider for instance the instructions of the agent S in the program of Example 13

$$s_0.\mathsf{recv}(y).\mathsf{send}\big(\mathsf{pair}\big(\mathsf{pair}(t_s, t_c), \mathsf{crypt}(\mathsf{pair}(t_s, \mathsf{inc}(t_n)), \mathsf{pk}(t_c))\big)\big).s_1$$

with $t_s = \mathsf{snd}(\mathsf{fst}(y))$, $t_c = \mathsf{fst}(\mathsf{fst}(y))$, and $t_n = \mathsf{decrypt}\big(\mathsf{snd}\big(\mathsf{decrypt}(\mathsf{snd}(y), \mathsf{pk}(t_c))\big), \mathsf{sk}(S)\big)$. We have the following transitions in $\mathcal{A}$

$$\begin{array}{rcl} \mathsf{fst}\big(q^y_{c_1 s_0}\big) & \to & q^{\mathsf{fst}(y)}_{c_1 s_0} \\ \mathsf{snd}\big(q^{\mathsf{fst}(y)}_{c_1 s_0}\big) & \to & q^{t_s}_{c_1 s_0} \\ & \vdots & \\ \mathsf{pair}\big(q^{pair(t_s, t_c)}_{c_1 s_0}, q^{crypt(\ldots)}_{c_1 s_0}\big) & \to & q_{c_1 s_1} \end{array}$$

$\diamond$

We need next some transitions in $\mathcal{A}$ modeling the behaviour of the attacker. As said above, the purpose of a state $q_{i_1 \ldots i_n}$ is to characterize the set of messages N in a reachable configuration $(\{i_1 \ldots i_n\}, \sigma, N)$. In other words, this state characterizes the knowledge of the attacker when the n agents reached the respective steps $i_1, \ldots, i_n$.

Let us consider first the tuple $(i_1^0, \ldots, i_n^0)$ of the initial program points of the agents $P_1, \ldots, P_n$ of P. The corresponding set of terms N_0 is characterized explicitly by a set of transitions of $\mathcal{A}$ using the states $q_{i_1^0 \ldots i_n^0}$ and $q^x_{i_1^0 \ldots i_n^0}$ (and possibly some auxiliary states used only for that purpose, see the Example 18 below). This set N_0 defines a unique *initial configuration* $(\{i_1^0, \ldots, i_n^0\}, \emptyset, N_0)$, which was mentioned when we discussed the reachable configurations, and N_0 is assumed to be part of the verification problem. Note that with this approach, it is possible to consider an infinite initial knowledge for the attacker. Moreover, the regular language N_0 is defined in a way that $cl(N_0) = N_0$, in order to conform to the above semantics.

Example 18. The initial set of the attacker's knowledge N_0 which was mentioned in Example 14 is defined by the following transitions of $\mathcal{A}$ (for the sake of readability, we denote below the state $q_{c_0 s_0}$ by q_0 and the states $q_{c_0 s_0} | q^x_{c_0 s_0} | q^y_{c_0 s_0}$ by q_0^{xy})

$$\begin{array}{l} A \to q_0^{xy}, A \to q_A, C \to q_0^{xy}, S \to q_0^{xy}, \\ \mathsf{pk}(q_0) \to q_0^{xy}, \mathsf{sk}(q_A) \to q_0^{xy}, \mathsf{fst}(q_0) \to q_0^{xy}, \mathsf{snd}(q_0) \to q_0^{xy}, \mathsf{inc}(q_0) \to q_0^{xy}, \\ \mathsf{crypt}(q_0, q_0) \to q_0^{xy}, \mathsf{decrypt}(q_0, q_0) \to q_0^{xy}, \mathsf{pair}(q_0, q_0) \to q_0^{xy}. \end{array}$$

where q_A is an auxiliary state that occurs only in the above 2 transitions of $\mathcal{A}$, in order to have $\mathsf{sk}(A) \in N_0$. $\diamond$

Next, we define some transitions modeling the evolution of the attacker's knowledge during the execution of the protocol. With the transitions defined above, we know that the states $q_{i_1 \ldots i_n}$ characterise the messages that can be sent to the network by the agents. Moreover, we want to enrich the languages of these states with the information that the attacker is able to learn form the messages sent. According to the semantics presented above, the technique used by the attacker to learn information from messages consists in applying public function symbols of Σ_{pub} at the top of the terms of its knowledge, i.e. the terms recognized in states $q_{i_1 \ldots i_n}$. It is expressed by transitions of the form:

$$f(q_{\vec{i_1}}, \ldots, q_{\vec{i_m}}) \to q_{\vec{i}} | q^x_{\vec{i}} \quad f \in \Sigma_{\mathsf{pub}}, \vec{i} = \max_{1 \leq j \leq m} \vec{i_j}, x \text{ variable of } P$$

where the operator max is applied componentwise to the vectors $\vec{i_j}$ and refers to an order defined on the program points of each agents by their order of appearance in the P_j's.

Example 19. For the program of Example 13, we have the following attacker's

transitions (additionally to the one presented in Example 18 above)

$$\begin{array}{ll}
\mathsf{pk}(q_{c_0 s_1}) \to q_{c_0 s_1}|q^x_{c_0 s_1}|q^y_{c_0 s_1}, \mathsf{pk}(q_{c_1 s_0}) \to q_{c_1 s_0}|q^x_{c_1 s_0}|q^y_{c_1 s_0}, & \\
\mathsf{pk}(q_{c_1 s_1}) \to q_{c_1 s_1}|q^x_{c_1 s_1}|q^y_{c_1 s_1} & \text{(and idem for } \mathsf{fst}, \mathsf{snd}, \mathsf{inc}), \\
\mathsf{crypt}(q_{c_0 s_1}, q_{c_0 s_1}) \to q_{c_0 s_1}|q^x_{c_0 s_1}|q^y_{c_0 s_1}, & \\
\mathsf{crypt}(q_{c_0 s_1}, q_{c_1 s_0}) \to q_{c_1 s_1}|q^x_{c_1 s_1}|q^y_{c_1 s_1}, \ldots & \text{(and idem for } \mathsf{decrypt}, \mathsf{pair}).
\end{array}$$

◇

8.4. Verification of Security Properties

We will see that in our setting, it is possible to express and verify confidentiality and authentication properties for a protocol by a reduction to decision problems for the RTA $\mathcal{A}$ constructed above.

Example 20. The protocol of Example 12 is supposed to ensure the authenticity of the message of S and also the confidentiality of $\mathsf{inc}(N)$ (for instance the value $\mathsf{inc}(N)$ can be supposed to be reused later as a key for symmetric encryption of a communication tunnel). However, both these properties can be attacked with a replay attack described in the following counter example.

$$\begin{array}{rrcll}
1. & C & \to & A(S) & : \mathsf{pair}\big(\mathsf{pair}(C,S), \mathsf{crypt}(\mathsf{pair}(C, \mathsf{crypt}(N, \mathsf{pk}(S))), \mathsf{sk}(C))\big) \\
1'. & A & \to & S & : \mathsf{pair}\big(\mathsf{pair}(A,S), \mathsf{crypt}(\mathsf{pair}(A, \mathsf{crypt}(N, \mathsf{pk}(S))), \mathsf{sk}(A))\big) \\
2'. & S & \to & A & : \mathsf{pair}\big(\mathsf{pair}(S,A), \mathsf{crypt}(\mathsf{pair}(S, \mathsf{inc}(N)), \mathsf{pk}(A))\big) \\
2. & A(S) & \to & C & : \mathsf{pair}\big(\mathsf{pair}(S,C), \mathsf{crypt}(\mathsf{pair}(S, \mathsf{inc}(N)), \mathsf{pk}(C))\big)
\end{array}$$

This counter example involves two parallel sessions of the protocol. In the first session (messages 1 and 2), the client C contacts the server S, following the protocol. But the first message is diverted by the attacker, (i.e. the message 1 stays in the network without being delivered to S) as indicated by the receiver denoted by $A(S)$. Then the attacker opens a new session (messages $1'$ and $2'$), between himself, A, (acting as a client) and the same server S. It is important to note that in message $1'$, the attacker reuses the same number N as in 1.

Actually, the attacker is not able to decrypt $\mathsf{crypt}(N, \mathsf{pk}(S))$, because he does not know the secret key $\mathsf{sk}(S)$. However, he is able to decrypt $\mathsf{crypt}(\mathsf{pair}(C, \mathsf{crypt}(N, \mathsf{pk}(S))), \mathsf{sk}(C))$, using the public key of C. Hence he reuses this ciphertext $\mathsf{crypt}(N, \mathsf{pk}(S))$ in $1'$, as a ciphertex protecting a fresh value of N. The server, who is not aware that this is a replay, replies with $\mathsf{crypt}(\mathsf{pair}(S, \mathsf{inc}(N)), \mathsf{pk}(A))$, a message that the attacker is able to decrypt, with his own secret key $\mathsf{sk}(A)$. Hence the attacker learns the value $\mathsf{inc}(N)$ which is supposed to be shared only by S and C. It means that N is also compromised if we assume that inc is invertible, i.e. that there exists a public unary function $\mathsf{dec} \in \Sigma_{\mathsf{pub}}$ and a rewrite rule $\mathsf{dec}(\mathsf{inc}(x)) = x \in \mathcal{R}$ (we did not consider these additional symbols and rules in our example above because they are not necessary for our purpose).

Moreover, the attacker can send the last message 2, impersonating S (this is denoted by the sender $A(S)$). Hence this is also an attack on the authenticity of this message (the server S was actually not involved in the session of the protocol made of messages 1 and 2). ◇

The existence of a confidentiality flaw like the one described in Example 20 is reducible to the problem of membership modulo $\mathcal{R}$ ($t \in \mathcal{R}^*\big(L(\mathcal{A})\big)$, see Section 7.2), for the RTA $\mathcal{A}$ constructed above.

Example 21. The confidentiality attack described in Example 20 occurs with two parallel sessions, involving 3 agents: 1 agent C playing the role of the client in the first session, and 2 agents playing the role of the server, respectively in the first and second session. The server agent in the first session is inactive. The role of the client in the second session is played by the attacker.

We can recognize this attack by analysing a program made of the 2 agents C and S defined in Example 13 plus the following second instance of a server

$$\begin{array}{l} S: \quad s'_0.\mathsf{recv}(y').\mathsf{send}\big(\mathsf{pair}\big(\mathsf{pair}(t'_s, t'_c), \mathsf{crypt}(\mathsf{pair}(t'_s, \mathsf{inc}(t'_n)), \mathsf{pk}(t'_c))\big)\big).s'_1 \\ \text{where } t'_s = \mathsf{snd}(\mathsf{fst}(y')), t'_c = \mathsf{fst}(\mathsf{fst}(y')), \\ \qquad\qquad t'_n = \mathsf{decrypt}\big(\mathsf{snd}\big(\mathsf{decrypt}(\mathsf{snd}(y'), \mathsf{pk}(t'_c))\big), \mathsf{sk}(S)\big) \end{array}$$

This agent has the same identity S as the first one in Example 13. Despite the renaming of the variable y into y' and of the program points (for technical convenience), this agent is the same as the one of Example 13. Let us construct the RTA $\mathcal{A}$ for this 3 agents as above. The states of $\mathcal{A}$ have the form $q_{c_i s_j s'_k}$ or $q^z_{c_i s_j s'_k}$ for $i, j, k \in \{0, 1\}$ and $z \in \{x, y, y'\}$.

We have that $\mathsf{inc}(N) \in R^*\big(L(\mathcal{A}, q_{c_1 s_0 s'_1})\big)$. Since the closure under $\mathcal{R}$ of the language in state $q_{c_1 s_0 s'_1}$ represents the knowledge of the attacker, it means that the value $\mathsf{inc}(N)$ has been compromised. Indeed, following the construction of $\mathcal{A}$, we have

$$t_1 = \mathsf{pair}\big(\mathsf{pair}(C, S), \mathsf{crypt}(\mathsf{pair}(C, \mathsf{crypt}(N, \mathsf{pk}(S))), \mathsf{sk}(C))\big) \in L(\mathcal{A}, q_{c_1 s_0 s'_0}).$$

This term t_1 corresponds to the message 1 (of C) in Example 20. Using the transitions of the attacker, we obtain that

$$t_{1'} = \mathsf{pair}\big(\mathsf{pair}(A, S), \mathsf{crypt}(\mathsf{pair}(A, \mathsf{snd}(\mathsf{decrypt}(\mathsf{snd}(t_1), pk(C)))), \mathsf{sk}(A))\big) \in L(\mathcal{A}, q^{y'}_{c_1 s_0 s'_0}).$$

Note that $\mathsf{snd}(\mathsf{decrypt}(\mathsf{snd}(t_1), pk(C))) \xrightarrow[\mathcal{R}]{*} \mathsf{crypt}(N, \mathsf{pk}(S))$. With the transitions for the pattern matching of the message of the second agent playing the role of S, we have

$$\begin{array}{lcl} t_{2'} & = & \mathsf{pair}\big(\mathsf{pair}(t'_s, t'_c), \mathsf{crypt}(\mathsf{pair}(t'_s, \mathsf{inc}(t'_n)), \mathsf{pk}(t'_c))\big) \in L(\mathcal{A}, q_{c_1 s_0 s'_1}) \\ \text{with} & & \\ t'_s & = & \mathsf{snd}(\mathsf{fst}(t_{1'})), \\ t'_c & = & \mathsf{fst}(\mathsf{fst}(t_{1'})), \\ t'_n & = & \mathsf{decrypt}\big(\mathsf{snd}\big(\mathsf{decrypt}(\mathsf{snd}(t_{1'}), \mathsf{pk}(t'_c))\big), \mathsf{sk}(S)\big). \end{array}$$

Next, using again the transitions of the attacker, we obtain

$$t' = \mathsf{snd}(\mathsf{decrypt}(\mathsf{snd}(t_{2'}), sk(A))) \in L(\mathcal{A}, q_{c_1 s_0 s'_1}),$$

and we have $t' \xrightarrow[\mathcal{R}]{*} \mathsf{inc}(N)$. Hence there is a positive answer to the problem of membership modulo for $\mathcal{A}$, $\mathcal{R}$ and $\mathsf{inc}(N)$, meaning that there exists a confidentiality attack. $\diamond$

Let us make a few remarks on the above analysis. The construction of two generic agents like in Example 13 is the specification of the protocol, written by the user. These agents represent the 2 roles (client and server in the protocol). Adding several instances of each agent in a program (to be verified) can be done automatically, just by variable and program point renaming as described above. The second part of the user specification is the construction of a TA recognizing the initial attacker's knowledge N_0, like in Example 18. The construction of the rest of the RTA, on the top of the agents and the TA for N_0 is automatic and the definition of the signature Σ and the TRS $\mathcal{R}$ are generic and independent of the protocol.

To summarize, RTA techniques permit an automatic analysis of the confidentiality property for security protocols, by reduction to the problem of membership modulo for RTA, given

- a definition of the set of public symbols Σ_{pub},
- a user specification (as programs) of the roles of the protocol,
- the number and identities of the agents playing the different roles of the protocol (generic results like [28] can help),
- a finite representation of the initial knowledge of the attacker N_0, and
- a ground term whose confidentiality must be ensured.

Note that the verification technique described above is exact: it requires no approximation on the protocol and attacker model (as long as the protocol is a program in the syntax of Section 8.1). Hence, every attack reported is a real attack, all the confidentiality attacks are reported and a negative answer is reported to the problem of membership modulo only if there exists no confidentiality attack, under the above hypotheses.

Authentication flaws like the one described in Example 20 can be reduced to the problem of emptiness of the intersection between the RTA $\mathcal{A}$ and a TA $\mathcal{E}$ (does $L(\mathcal{A}) \cap L(\mathcal{E}) = \emptyset$). The idea is to add some tags in the agent's messages, for instance marking the end of every agent. The tags are built with function symbols which are not public (hence they can only be added by the agents, with special instructions), but we can also consider other public functions that the attacker can apply to remove the tags (modulo some rules in $\mathcal{R}$ for that purpose). Then the TA $\mathcal{E}$ characterizes some traces corresponding to authentication errors. For instance, the authentication flaw described in Example 20 can be characterized by the fact that C has received a message 2 (in the first session) and entered program point c_1 (this is characterised by the presence of a tag T_c in the term) while S did not send it, and is still at program point s_0 (this is characterised by the absence of a tag above T_c in the term). The emptiness of the intersection of $L(\mathcal{E})$ and $L(\mathcal{A})$ (note that this language is not considered modulo $\mathcal{R}$ in this case) means that there is no authentication flaw.

However, with the above model this approach is quite limited, since the agents can accept any message in the input. Hence, many false authentication attacks will be reported. This verification technique, related to regular tree model checking, would make more sense with a model with some conditionals in the agents, between instructions $\mathsf{recv}(x)$ and $\mathsf{send}(s)$.

Conclusion and Further Work

We have presented the class of Rigid Tree Automata and its properties. We have also studied the closure of RTA languages under term rewriting, and proposed an algorithm to decide that a given term belong to the closure for linear and invisibly pushdown TRS.

This class of tree automata is thought to be well suited for the automatic verification of some infinite state systems, and in particular for the verification of traces or equivalence properties of security protocols, using regular tree model checking like techniques. In this context, it would be interesting to extend the result of Theorem 15 to invisibly pushdown (non-linear) TRS, in order to handle axioms like $\mathsf{decrypt}(\mathsf{encrypt}(x, y), y) = x$. We are also interested about the symmetric form of the TRS of [5], whose rhs's are not single variables but have the form $f(x_1, \ldots, x_n)$.

In Section 3.5, we have also mentioned the possibility to define RTA as sets of Horn clauses and the use of general purpose first order theorem proving tools in order to decide properties like the ones related to the rewrite closure. Such an approach could be interesting for instance for the extension of RTA with equational tests like in [10], in order to be able to capture conditionals in the model of security protocols presented in Section 8.

A comparison with restricted TA with registers was also mentioned in Section 3.5. Another potential application is indeed concerned with the processing of trees containing data from an infinite domain (this was for instance the motivation for the definition of TAGED in [12]). With a modeling of such data into subterms in $\mathcal{T}(\Sigma')$ (where Σ' is an auxiliary alphabet), some comparisons between data values can be expressed with rigid states. In this context, it could be interesting to find a decidable extension of VRTA with some other constraints complementary to the rigidity condition, in order to obtain a class closed under complement. This closure could permit to ensure a correspondence with a logic in which some queries on data trees can be expressed.

References

[1] B. Bogaert, S. Tison, Equality and disequality constraints on direct subterms in tree automata, in: 9th Symp. on Theoretical Aspects of Computer Science, STACS, volume 577 of *LNCS*, Springer, 1992, pp. 161–171.

[2] M. Dauchet, A.-C. Caron, J.-L. Coquidé, Automata for Reduction Properties Solving, Journal of Symbolic Computation 20 (1995) 215–233.

[3] H. Comon, V. Cortier, Tree automata with one memory, set constraints and cryptographic protocols, Theoretical Computer Science 331 (2005) 143–214.

[4] I. Guessarian, Pushdown tree automata, Mathematical Systems Theory 16 (1983) 237–263.

[5] J. Chabin, P. Réty, Visibly pushdown languages and term rewriting, in: 6th International Symposium on Frontiers of Combining Systems, FroCos, volume 4720 of *LNCS*, Springer, 2007, pp. 252–266.

[6] E. Filiot, J.-M. Talbot, S. Tison, Tree automata with global constraints, in: 12th International Conference in Developments in Language Theory, DLT, volume 5257 of *LNCS*, Springer, 2008, pp. 314–326.

[7] A. Bouajjani, T. Touili, On computing reachability sets of process rewrite systems, in: 16th International Conference on Term Rewriting and Applications, RTA, volume 3467 of *LNCS*, Springer, 2005, pp. 484–499.

[8] T. Genet, F. Klay, Rewriting for Cryptographic Protocol Verification, in: Proc. of 17th Int. Conf. on Automated Deduction, CADE, volume 1831 of *LNCS*, Springer, 2000, pp. 271–290.

[9] M. Abadi, C. Fournet, Mobile values, new names, and secure communication, in: 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL, pp. 104–115.

[10] F. Jacquemard, M. Rusinowitch, L. Vigneron, Tree automata with equality constraints modulo equational theories, Journal of Logic and Algebraic Programming 75 (2008) 182–208.

[11] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, C. Löding, D. Lugiez, S. Tison, M. Tommasi, Tree Automata Techniques and Applications, `http://tata.gforge.inria.fr`, 2007.

[12] E. Filiot, J.-M. Talbot, S. Tison, Satisfiability of a spatial logic with tree variables, in: 21st International Workshop on Computer Science Logic, CSL, volume 4646 of *LNCS*, Springer, 2007, pp. 130–145.

[13] L. Barguñó, C. Creus, G. Godoy, F. Jacquemard, C. Vacher, The emptiness problem for tree automata with global constraints, in: Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science (LICS'10), IEEE Computer Society Press, Edinburgh, Scotland, UK, 2010, pp. 263–272.

[14] J. Mongy, Transformation de noyaux reconnaissables d'arbres. Forêts RATEG, Ph.D. thesis, Laboratoire d'Informatique Fondamentale de Lille, UST, Villeneuve d'Ascq, France, 1981.

[15] W. Charatonik, Automata on DAG Representations of Finite Trees, Technical Report Technical Report MPI-I-99-2-001, Max-Planck-Institut für Informatik, Saarbrücken, Germany, 1999.

[16] H. Comon-Lundh, F. Jacquemard, N. Perrin, Visibly tree automata with memory and constraints, Logical Methods in Computer Science 4 (2008).

[17] P. B. Andrews, Theorem proving via general matings, Journal of the ACM 28 (1981) 193–214.

[18] S. Delaune, H. Lin, Ch. Lynch, Protocol verification via rigid/flexible resolution, in: 14th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning, LPAR, volume 4790 of *LNCS*, Springer, 2007, pp. 242–256.

[19] R. Affeldt, H. Comon-Lundh, Verification of security protocols with a bounded number of sessions based on resolution for rigid variables, in: Formal to Practical Security, volume 5458 of *LNCS, State-of-the-Art Survey*, Springer, 2009, pp. 1–20.

[20] T. Frühwirth, E. Shapiro, M. Vardi, E. Yardeni, Logic programs as types for logic programs, in: 6th IEEE Symposium on Logic in Computer Science, pp. 300–309.

[21] J. Goubault-Larrecq, Deciding $\mathcal{H}_1$ by Resolution, Information Processing Letters 95 (2005) 401–408.

[22] H. Seidl, Haskell overloading is dexptime-complete, Information Processing Letters 52 (1994) 57–60.

[23] R. Alur, P. Madhusudan, Visibly pushdown languages, in: 36th Annual ACM Symposium on Theory of Computing, STOC, ACM, 2004, pp. 202–211.

[24] S. Anantharaman, P. Narendran, M. Rusinowitch, Closure properties and decision problems of dag automata, Inf. Process. Lett. 94 (2005) 231–240.

[25] K. Salomaa, Deterministic tree pushdown automata and monadic tree rewriting systems, J. Comput. Syst. Sci. 37 (1988) 367–394.

[26] R. M. Amadio, W. Charatonik, On name generation and set-based analysis in the dolev-yao model, in: 13th International Conference on Concurrency Theory, CONCUR, volume 2421 of *LNCS*, Springer, 2002, pp. 499–514.

[27] S. Yamaguchi, K. Okayama, H. Miyahara, The design and implementation of an authentication system for the wide area distributed environment, IEICE Transactions on Information and Systems E74 (1991) 3902–3909.

[28] H. Comon-Lundh, V. Cortier, Security properties: Two agents are sufficient, Science of Computer Programming 50 (2004) 51–71.