Rigid Tree Automata

Florent Jacquemard, Francis Klay, Camille Vacher

Rigid Tree Automata *

Florent Jacquemard[1], Francis Klay[2], and Camille Vacher[3]

[1] INRIA Saclay & LSV (CNRS/ENS Cachan). florent.jacquemard@inria.fr
[2] FT/RD/MAPS/AMS/SLE. francis.klay@orange-ftgroup.com
[3] FT/RD & LSV (CNRS/ENS Cachan). vacher@lsv.ens-cachan.fr

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 restricted 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 a RTA language under a restricted family of term rewriting systems, whereas this closure is not a 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. Finally, a comparison of RTA with several classes of tree automata with local and global equality tests, and with dag automata is also provided.

Introduction

Tree automata (TA) are finite representations of infinite sets of terms. 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 by a 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

* This work was partly supported by the ANR Sesur 07 project AVOTÉ.

memory comparison [3]. 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 identification 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 and 3 we study issues like determinism, closure of languages under Boolean operations, comparison with related classes of automata and decision problems for RTA. RTA are a particular case of the more general class Tree Automata with General Equality and Disequality constraints (TAGED [4]). The study of the class RTA alone is motivated by the complexity results and applications mentioned below. But our most original contribution is the study of the rewrite closure of RTA languages.

Combining tree automata and term rewriting techniques has been very successful in verification see e.g. [5, 6]. In this context, term rewriting systems (TRS) can describe the transitions of a system, the evaluation of a program [5], the specification of operators used to build protocol messages [7] or also the transformation of documents. In these approaches, the rewrite closure $\mathcal{R}^*\big(L(\mathcal{A})\big)$ of the language $L(\mathcal{A})$ of a TA $\mathcal{A}$ using $\mathcal{R}$ represents the set of states reachable from states described by $L(\mathcal{A})$. When $\mathcal{R}^*\big(L(\mathcal{A})\big)$ is again a TA language, the verification of a safety property amounts to check 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. [8]).

In Section 4, 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 TRS called linear invisibly, whereas this closure is generally not a RTA language. Linear invisibly TRS can typically specify cryptographic operators like $\mathsf{decrypt}(\mathsf{encrypt}(x, \mathsf{pk}(\mathsf{A})), \mathsf{sk}(\mathsf{A})) \to x$.

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. For instance, our initial motivation for studying RTA was the analysis of security protocols in a model where a finite number of processes exchange messages (following a protocol) asynchronously over an insecure network controlled by an attacker who is able to tamper the messages. The messages are terms build over cryptographic operators and are interpreted modulo an invisibly TRS $\mathcal{R}$ with rules like the above one for $\mathsf{decrypt}$ [7]. It is possible to built a RTA $\mathcal{A}$ recognizing exactly the set of messages that can be exchanged by executing the protocol in presence of the active attacker. The RTA $\mathcal{A}$ models both the honest processes and the attacker, and uses one rigid state to memorize each message sent by an honest process In these settings, it is possible to express confidentiality problems as membership modulo $\mathcal{R}$ (does $t \in \mathcal{R}^*\big(L(\mathcal{A})\big)$), and authentication like problems as emptiness of intersection with a TA (does $L(\mathcal{A}) \cap L(\mathcal{B}) = \emptyset$ for a TA $\mathcal{B}$ recognizing error

traces). All the details and proofs omitted in this extended abstract due to space restrictions can be found in the long version [9].

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 $\mathcal{X}$ to $\mathcal{T}(\Sigma, \mathcal{X})$.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 of 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). Position are compared wrt the prefix ordering: $p_1 < p_2$ iff there exists $p \neq \varepsilon$ such that $p_2 = p_1.p$. In this case, p is denoted $p_2 - p_1$. A subterm of t at position p is written $t|_p$, 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 $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$.

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 (lhs)). A term $t \in \mathcal{T}(\Sigma, \mathcal{X})$ rewrites to s 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 note $R^*(L) = \{t \mid \exists s \in L, s \xrightarrow[R]{*} t\}$. A TRS is called *linear* if all the terms in its rules are linear and *collapsing* if every rhs of rule is a variable.

Tree automata. 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 $q_1, \ldots, q_n, q \in Q$. The *size* of $\mathcal{A}$ is the number of symbols in Δ. A *run* of the TA $\mathcal{A}$ on a term $t \in \mathcal{T}(\Sigma)$ is a function $r : \mathcal{P}os(t) \to Q$ such that for all $p \in \mathcal{P}os(t)$ with $t(p) = f \in \Sigma_n$ $(n \geq 0)$, $f\big(r(p.1), \ldots, r(p.n)\big) \to r(p) \in \Delta$. 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 $r(\varepsilon) = q$. If $q \in F$ then this run r is

called *successful*. 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.

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 First Properties

2.1 Definition and Examples

Definition 1. *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 that: for all positions $p_1, p_2 \in \mathcal{P}os(t)$, if $r(p_1) = r(p_2) \in R$ then $t|_{p_1} = t|_{p_2}$. Languages of RTA are defined the same way as for TA. Note that with these definitions, every regular language is a RTA language. We shall write below TA and RTA for the classes of TA and RTA languages.

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 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 $a \to q$ and $a \to q_r$. A successful run of $\mathcal{A}$ on $f(f(a,b), f(a,b))$ is $q_{\mathsf{f}}\big(q_r(q,q), q_r(q,q)\big)$. ◇

Note that the above RTA language is not regular; RTA generalize to non-linear pattern the (linear) pattern matching ability of TA.

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 $f(x,x)$. ◇

Proposition 1. *For all term $t \in \mathcal{T}(\Sigma, \mathcal{X})$, there exists a RTA recognizing the terms of $\mathcal{T}(\Sigma)$ which have a ground instance of t as a subterm.*

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

Example 3. 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 subterm of t is recognized by the RTA on Σ $\langle\{q, q_r, q', q_{\mathsf{f}}\}, \{q_r\}, \{q_{\mathsf{f}}\}, \Delta\rangle$ with $\Delta = \{a \to q|q_r, b \to q|q_r, f(q,q) \to q|q_r, f(q,q_r) \to q', f(q_r, q) \to q', f(q,q') \to q', f(q',q) \to q', <(q_r, q') \to q_{\mathsf{f}}\}$. 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 the state q' is reached immediately above q_r and propagated up to the root, in order to express that the right side t of $<$ is a superterm of s. ◇

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

Example 4. 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 RTA $\langle \Sigma, \{q, q_r, q_a, q_b, q_\mathsf{f}\}, \{q_r\}, \{q_\mathsf{f}\}, \Delta\rangle$ with $\Delta = \{c \to q | q_r, a(q) \to q | q_r, b(q) \to q | q_r, a(q_r) \to q_a, b(q_r) \to q_b\} \cup \big\{a(q_x) \to q_x, b(q_x) \to q_x \mid q_x \in \{q_a, q_b\}\big\} \cup \big\{\neq(q_1,q_2) \to q_\mathsf{f} \mid q_1, q_2 \in \{q_a, q_b, q_r\}, q_1 \neq q_2\big\}$. 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 symbol $\neq$). ◇

2.2 Pumping Lemma

We propose here 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 equality condition for rigid states. Moreover, we cannot repeat a term containing a rigid state, because the same rigid state cannot label two different positions on the same branch. The proof of the following lemma can be found in [9].

Lemma 1. *For all RTA $\mathcal{A} = \langle Q, R, F, \Delta\rangle$, for all term $t \in L(\mathcal{A})$ 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 $n \geq 0$, $C\big[C'[D^n[u]], \ldots, C'[D^n[u]]\big] \in L(\mathcal{A})$.*

Example 5. The set $\mathcal{B}$ of balanced binary trees built over the signature $\{a : 0, f : 2\}$ is not a RTA language. Assume indeed that it is recognized by a 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 $C\big[C'[D^n[u]], \ldots, C'[D^n[u]]\big] \in L(\mathcal{A})$. ◇

2.3 Related Classes of Tree Automata

We shall briefly present below some classes of extended TA and compare them to RTA. The decidability and complexity results presented in Section 3 and summarized in Figure 1 also offer a base of comparison.

TAGED [4] were introduced in the context of spatial logics for XML querying [10]. 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 decidability of the emptiness problem is open for the whole TAGED class. A decidable subclass of TAGED is identified in [10] where the number of equality tested in every run is bounded. The fragment of positive TAGED (with $R_\neq = \emptyset$, denoted TAGED+) has the same expressiveness as RTA. This is shown in [4] where a TAGED+ is transformed implicitely into a RTA in order to decide emptiness, at the price of an exponential blowup. The emptiness is EXPTIME-complete for TAGED+, and PTIME for RTA (see Section 3). To our knowledge, the rewrite closure of TAGED has not been studied so far.

TA with equality constraints (TAC) are TA whose transitions can perform local equality and disequality tests on the subterms of the term in input (e.g. [1, 2]). In contrast, the equality tests of RTA can be global. For instance, the language of terms t over $\{f : 2, g : 1, a : 0\}$ such that $s_1 = s_2$ for every two subterms $g(s_1)$, $g(s_2)$ of t is recognizable by a RTA, but not by a TAC. The RTA language of Examples 1 and 2 are recognizable by TAC, but not the one of Example 3. The language $\mathcal{B}$ of Example 5, which is not recognizable by RTA, is recognizable by TAC.

DAG automata (DA) [11] are defined as TA computing on DAG representation of terms with maximal sharing. Somehow, DA are the dual of RTA in the sense that in their runs, a unique state is associated to equal subtrees (which are rooted by the same node in the DAG representation) whereas for RTA, a unique subtree is associated to every occurrence of the same rigid state. However, the classes of languages defined by these two formalisms are orthogonal. On one hand, one can observe that the RTA language of Example 1 (terms $f(t, t)$) is not recognizable by a DA. On the other hand, the emptiness problem is PTIME for RTA and NP-complete for DA [12]. Actually, DA and RTA are defined for different purposes: DA are proposed for computing on compressed trees, and not for checking equalities like RTA. Moreover, deterministic DA coincide with DTA, and, as we show in Section 2.4, it is not the case for DRTA.

2.4 Determinism and Completeness

A *deterministic rigid tree automaton* (DRTA) (resp. complete RTA) on a signature Σ is a 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. 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 given run of $ta(\mathcal{A})$ on t might not be a run of $\mathcal{A}$ on t because of the rigidity condition.

Example 6. 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 $L\big(ta(\mathcal{A}), q_r\big)$, with a unique run $r = q_r(q(q_r(q)))$. However, r is not a run of $\mathcal{A}$, because the two subterms at the positions of q_r are distinct. $\diamond$

It is well-known that DTAs are as expressive as TAs. We show below that it is not the case for RTA.

Theorem 1. *DRTA $\subsetneq$ RTA and TA $\subsetneq$ DRTA.*

Proof. We show in [9] that the language of Example 1 is not recognized by a DRTA. The inclusion TA $\subset$ DRTA is immediate since DTA $\equiv$ TA and DTA are particular cases of DRTA. Let $\Sigma = \{f{:}2, g{:}1, a{:}0\}$. The language $\big\{f(g(t), g(t)) \mid t \in \mathcal{T}(\Sigma \setminus \{g\}\big\}$ is recognized by a DRTA but not by a TA. $\square$

2.5 Boolean Closure

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

Theorem 2. *Given two RTA $\mathcal{A}_1$ and $\mathcal{A}_2$, there exist two RTAs of respective sizes $O(|\mathcal{A}_1| + |\mathcal{A}_2|)$ and $O(2^{|\mathcal{A}_1||\mathcal{A}_2|})$ 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. For $L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$, it is easy to construct a positive TAGED $\mathcal{B}$ recognizing $L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$ by a product operation like for standard TA. The state set of $\mathcal{B}$ is $Q_1 \times Q_2$, its final state set $F_1 \times F_2$ and its transition rules $\{f\big(\langle q_{11}, q_{21}\rangle, \ldots, \langle q_{1n}, q_{2n}\rangle\big) \to \langle q_1, q_2\rangle \mid q_{i1} \ldots q_{in}, q_i \in Q_i f(q_{i1}, \ldots, q_{in}) \to q_i \in \Delta_i, i = 1, 2\}$. Moreover, the equality relation of $\mathcal{B}$ is $R_= = \big\{\big\langle\langle q_{r_1}, q_2\rangle, \langle q_{r_1}, q_2'\rangle\big\rangle \mid q_{r_1} \in R_1, q_2, q_2' \in Q_2\big\} \cup \big\{\big\langle\langle q_1, q_{r_2}\rangle, \langle q_1', q_{r_2}\rangle\big\rangle \mid q_1, q_1' \in Q_1, q_{r_2} \in R_2\big\}$. A construction is proposed in [4] for transforming any positive TAGED into an RTA (i.e. a TAGED with a reflexive state relation). This transformation causes an exponential blowup. It cannot be described here. Combining the two above steps results in an exponential construction for the intersection of RTA. $\square$

Note that the construction for the intersection of RTA preserves determinism but not for the union. The following lemma (its proof can be found in [9]) shows that the exponential time complexity for the construction of the intersection automaton constructed in Theorem 2 is a lower bound, by a reduction of the EXPTIME-complete problem of the non-emptiness of the intersection of n TA.

Lemma 2. *Given n RTA $\mathcal{A}_1, \ldots, \mathcal{A}_n$ 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 $L(\mathcal{A}_\times) \cap L(\mathcal{A}_r) = \emptyset$.*

Theorem 3. *The class of RTA languages is not closed under complement.*

Proof. We have seen in Example 5 that the set $\mathcal{B}$ of balanced binary trees over $\Sigma := \{a : 0, f : 2\}$ is not a 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 3: 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 $\Delta = \{a \to q|q_r, f(q,q) \to q|q_r, f(q,q_r) \to q', f(q_r,q) \to q', f(q,q') \to q', f(q',q) \to q', f(q_r,q') \to q_\mathsf{f}, f(q',q_r) \to q_\mathsf{f}, f(q_\mathsf{f},q) \to q_\mathsf{f}, f(q,q_\mathsf{f}) \to q_\mathsf{f}\}$. The last two transition rules ensure the propagation of the final state q_f up to the root, like in Example 2. □

3 Decision problems

We study in this section several decision problems for RTA; *emptiness*: given a RTA $\mathcal{A}$ on Σ, does $L(\mathcal{A}) = \emptyset$, *universality*: does $L(\mathcal{A}) = \mathcal{T}(\Sigma)$, *finiteness*: is $L(\mathcal{A})$ finite, *membership*: given additionally $t \in \mathcal{T}(\Sigma)$, is t in $L(\mathcal{A})$; *inclusion*: given two RTA $\mathcal{A}_1$ and $\mathcal{A}_2$, does $L(\mathcal{A}_1) \subseteq L(\mathcal{A}_2)$, *equivalence*: does $L(\mathcal{A}_1) = L(\mathcal{A}_2)$, and *intersection emptiness*: given n RTA $\mathcal{A}_1, \ldots \mathcal{A}_n$, does $L(\mathcal{A}_1) \cap \ldots \cap L(\mathcal{A}_n) = \emptyset$. Figure 1 provides a summary of closure and decision results and a comparison with other classes of extended TA mentioned in Section 2.3.

	TA	**RTA**	**TAGED+**	**DA**
$\cup$	PTIME	PTIME	PTIME	PTIME
$\cap$	PTIME	EXPTIME	EXPTIME	not [12]
$\neg$	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		

Table 1. Summary of closure and decision results

Theorem 4. *The emptiness problem is decidable in linear time for RTA.*

Proof. We show [9] that the emptiness of $L(\mathcal{A})$ and $L\big(ta(\mathcal{A})\big)$ are equivalent. The latter problem (emptiness for standard TA) is known to be decidable in linear-time (see e.g. [13]). The idea is that if $L(ta(\mathcal{A}))$ is not empty, then the classical "state marking" algorithm builts a witness which respects the rigidity condition for all states, and is therefore a witness for $L(\mathcal{A})$ non-emptiness. □

Theorem 5. *Membership is* NP*-complete for RTA (*PTIME *for DRTA).*

Proof. A non-deterministic algorithm for this problem consist in, given a 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 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.

In order to show NP-hardness, we propose [9] a reduction from 3-SAT for a formula ϕ into the membership to an RTA $\mathcal{A}$ of a term t_ϕ representing ϕ. Each variable x of ϕ is represented in t_ϕ by a subterm $x(0,1)$, where x is a binary symbol and $0, 1$ constants. The most important transitions of $\mathcal{A}$ are $0, 1 \to q_x|q_{\neg x}$ for each variable x of ϕ and $x(q_x, q_{\neg x}) \to q_0$, $x(q_{\neg x}, q_x) \to q_1$, where the states q_x and $q_{\neg x}$ are rigid. The states q_0 and q_1 represent the value associated to x (they are propagated bottom-up along t_ϕ) and the rigidity condition ensures that the same value is associated to all occurrences of the variable x in ϕ. □

Theorem 6. *Intersection non-emptiness is* EXPTIME*-complete for RTA.*

Proof. The upper-bound is a consequence of Lemma 2 and Th. 2 & 4. The lower-bound follows from the EXPTIME-hardness of the problem for TA [14]. □

Theorem 7. *Universality is undecidable for RTA.*

Proof. In [9] we reduce the non-existence of a solution of an instance P of the Post Correspondence Problem to the universality of a RTA. This RTA recognizes the set of terms which do not represent a solution of P. It is defined as a disjoint union of RTAs, one for each case. Some cases involve the construction of a RTA testing disequalities between unary subterms like in Example 4. □

Theorem 8. *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. □

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 6 is $\{a, g(g(a))\}$ whereas the language of its underlying TA is $\{a, g^2(a), g^4(a), \ldots\}$.

Theorem 9. *Finiteness is decidable in* PTIME *for RTA.*

Like for TA, checking finiteness amounts to detect (in PTIME) some loops and paths in the accessibility graph of an RTA (see [9] for details).

4 Rewrite Closure

The closure of a RTA language under rewriting is unfortunately not a RTA language, even for a TRS as simple as $\mathcal{R} = \{f(g(x)) \to x\}$. Let $\Sigma = \{h : 2, f : 1, g : 1, 0 : 0\}$, and let $\mathcal{A} = \langle Q, R, \Delta \rangle$ be the RTA on Σ with $Q = \{q_0, q_1, q_2, q_r, q_\mathrm{f}\}$,

$R = \{q_r\}$, $F = \{q_f\}$, and $\Delta = \{0 \to q_0, g(q_0) \to q_0 | q_r, f(q_r) \to q_1, f(q_1) \to q_1, h(q_r, q_{1,2}) \to q_f, h(q_{1,2}, q_{1,2}) \to q_2, h(q_f, q_{1,2}) \to q_f, \}$ 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 a k-context made of the symbol h only, and g^m and f^* represent nesting of m symbol g and an arbitrary number of f, respectively. The rigid state q_r enforces that each argument has the same number of g. 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 in the different arguments might not be equal. They only have to be all less than or equal to the number of g on the leftmost argument. We show in [9] that it is not a RTA language, with arguments similar to those of Section 2.2. The rewrite closure of a RTA under a linear collapsing TRS is even not recursive.

Theorem 10. *The problem to know whether $t \in \mathcal{R}^*(L(\mathcal{A}))$ or not given a RTA $\mathcal{A}$, a collapsing and linear TRS $\mathcal{R}$ and a term t, is undecidable.*

Proof. Let $u_1, v_1 \ldots u_n, v_n$ be words on an alphabet Γ seen as a PCP instance P. Let us consider the signature $\Sigma = \{g_i : 1, f_i : 1 \mid i \leq n\} \cup \{a : 1 \mid a \in \Gamma\} \cup \{0 : 0, k : 1, h : 2\}$, and $L = \{h(s, k(s)) \mid s = f_{i_1}(g_{i_1}(\ldots f_{i_m}(g_{i_m}(w(0))))), m > 0, w \in \Gamma^*\}$[4] where $1 \leq i_1, \ldots, i_m \leq n$. Let $\mathcal{R}$ be a TRS on Σ with the rules $f_i(g_i(u_i(x))) \to x$ $(i \leq n)$, $g_i(x) \to x$ $(i \leq n)$, $g_j(f_i(v_i(x))) \to x$ $(i, j \leq n)$, and $k(f_i(v_i(x))) \to x$ $(i \leq n)$. The tree language L is recognizable by a RTA on Σ and we show [9] that $h(0, 0) \in \mathcal{R}^*(L)$ iff P has a solution. □

The problem of Theorem 10, *membership modulo*, becomes decidable with some further syntactical restrictions on $\mathcal{R}$ based on the theory of visibly pushdown automata (VPA) [15]. VPA define a subset of context-free languages closed under intersection, and were generalized to tree recognizers in [16, 17]. 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 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.

In [16], Chabin and Rety show that the class of visibly pushdown tree automata (VPTA) languages is closed under rewriting with so called linear context-free visibly TRS. We use a similar definition in order to characterize a class of TRS for which membership modulo is decidable.

Definition 2. *A collapsing TRS $\mathcal{R}$ is called* inverse-visibly *(invisibly) 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 7. The TRS $\mathcal{R} = \{\mathsf{fst}(\mathsf{pair}(x_1, x_2)) \to x_1, \mathsf{snd}(\mathsf{pair}(x_1, x_2)) \to x_2, \mathsf{decrypt}(\mathsf{encrypt}(x, \mathsf{pk}(\mathsf{A})), \mathsf{sk}(\mathsf{A})) \to x\}$ is linear and invisibly with $\Sigma_c = \{\mathsf{fst}, \mathsf{snd}, \mathsf{decrypt}\}$ and $\Sigma_r = \{\mathsf{pair}, \mathsf{encrypt}\}$, $\Sigma_\ell = \{\mathsf{pk}, \mathsf{sk}, \mathsf{A}\}$. ◇

The TRS $\{f(g(x)) \to x\}$ is invisibly but not the one for Theorem 10.

[4] For all $w = a_1, \ldots, a_p \in \Gamma^*$, the term $a_1(\ldots a_p(t))$ is written $w(t)$.

Theorem 11. *The problem to know whether $t \in \mathcal{R}^*(L(\mathcal{A}))$ or not, given a RTA $\mathcal{A}$, a linear and invisibly TRS $\mathcal{R}$ and a term t, is decidable.*

Proof. The proof [9] is long and technical, and due to space restrictions, we only sketch it below for a TRS $\mathcal{R}$ containing the two first rewrite rules of Example 7.

Let $\mathcal{A} = \langle Q, R, F, \Delta \rangle$ be a RTA on $\Sigma = \{\mathsf{f} : 2, \mathsf{fst} : 1, \mathsf{snd} : 1, \mathsf{pair} : 2, 0 : 0\}$ with $Q = \{q_0, q_r, q_1, q_\mathsf{f}\}$, $R = \{q_r\}$, $F = \{q_\mathsf{f}\}$, and $\Delta = \{0 \to q_0, \mathsf{pair}(q_0, q_0) \to q_0 | q_r, \mathsf{fst}(q_r | q_1) \to q_1, \mathsf{snd}(q_r | q_1) \to q_1, \mathsf{f}(q_1, q_1) \to q_\mathsf{f}\}$ and let $t = \mathsf{f}(\mathsf{pair}(0,0),0)$. Very roughly, the decision algorithm guesses the existence of one tree $t' \in L(\mathcal{A})$ such that $t' \xrightarrow[\mathcal{R}]{*} t$, by application of $\mathcal{R}$ backwards starting from t, expanding subterms into lhs of rules. In order to ensure that $t' \in L(\mathcal{A})$, we consider pairs of states $\frac{q_\varepsilon}{q_x}$ which intuitively correspond to a run r of $\mathcal{A}$ on ℓ (for $\ell \to x \in \mathcal{R}$) such that $q_\varepsilon = r(\varepsilon)$ and $q_x = r(p_x)$ where $\ell(p_x) = x$ (this position p_x is unique by hypothesis). If $q_\varepsilon = q_x$, the pair is simply denoted q_ε. In a first step, we label the lhs of $\mathcal{R}$ with such pairs. For both $\mathsf{fst}(\mathsf{pair}(x_1, x_2))$ and $\mathsf{snd}(\mathsf{pair}(x_1, x_2))$, the only possible labelling is $\ell_1 := q_1(\frac{q_1}{q_0}(q_0, q_0))$. The condition for such a labelling is indeed that there exists a transition in $\mathcal{A}$ from the first components of labels at sibling positions into the second component of the label at the father position, like $\mathsf{fst}(q_1) \to q_1$ and $\mathsf{pair}(q_0, q_0) \to q_0$ for ℓ_1 above. Intuitively, ℓ_1 describes a nesting of runs on lhs of $\mathcal{R}$ which permits to recover a run r' of $\mathcal{A}'$ on t'. In other terms, t' can be generated by a context-free tree grammar with non-terminals from Q (nullary) or of the form $\frac{q_\varepsilon}{q_x}$ (unary). For instance, the production rule $\frac{q_1}{q_0}(x) := \mathsf{fst}(\frac{q_1}{q_0}(\mathsf{pair}(x, q_0))$ corresponds to $\mathsf{fst}(\mathsf{pair}(x_1, x_2)) \to x_1$. The grammar generates a visibly pushdown tree language, thanks to the hypotheses on $\mathcal{R}$.

Next, in order to guess t', we label the positions of t by pairs of states. We obtain $q_\mathsf{f}(p(q_0, q_0), \frac{q_1}{q_0})$ where p is either q_r or $\frac{q_1}{q_0}$. We can observe that the rigid state q_r occurs, possibly at nested depth bigger than one, in both cases for p. The tricky part of the algorithm it to check that there exists at least one term in the intersection of the languages (generated by grammars as above) corresponding to the distinct occurrences of q_r. We use the fact that the emptiness of intersection is decidable for visibly context free tree grammars. □

Conclusion and Further Work

We want to use RTA for the automatic verification of traces or equivalence properties of security protocols, using regular tree model checking like techniques. In this context, we are planning to extend the result of Theorem 11 to invisibly (non-linear) TRS, in order to handle axioms as $\mathsf{decrypt}(\mathsf{encrypt}(x, y), y) = x$. We are also interested about the symmetric form of the TRS of [16], whose rhs are not single variables but have the form $f(x_1, \ldots, x_n)$.

One may also study the extension of RTA to equality tests modulo equational theories like in [8], or the addition of disequality constraints in order to obtain closure under complement and correspondence with logics.

References

1. Bogaert, B., Tison, S.: 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), 161–171.
2. Dauchet, M., Caron, A.C., Coquidé, J.L.: Automata for Reduction Properties Solving. Journal of Symbolic Computation **20**(2) (1995), 215–233.
3. Comon, H., Cortier, V.: Tree automata with one memory, set constraints and cryptographic protocols. Theoretical Computer Science **331**(1) (2005), 143–214.
4. Filiot, E., Talbot, J.M., Tison, S.: Tree automata with global constraints. In: 12th International Conference in Developments in Language Theory (DLT 2008). Volume 5257 of LNCS, Springer (2008), 314–326.
5. Bouajjani, A., Touili, T.: On computing reachability sets of process rewrite systems. In: 16th International Conference Term Rewriting and Applications, RTA. Volume 3467 of LNCS, Springer (2005), 484–499.
6. Genet, T., Klay, F.: Rewriting for Cryptographic Protocol Verification. In: 17th Int. Conf. on Automated Deduction, CADE. Volume 1831 of LNCS, Springer (2000).
7. Abadi, M., Fournet, C.: Mobile values, new names, and secure communication. In: 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL. (2001), 104–115.
8. Jacquemard, F., Rusinowitch, M., Vigneron, L.: Tree automata with equality constraints modulo equational theories. Journal of Logic and Algebraic Programming **75**(2) (2008), 182–208,
9. Jacquemard, F., Klay, F., Vacher, C.: Rigid tree automata. Technical Report RRLSV-0827, Laboratoire Spécification et Vérification (2008).
10. Filiot, E., Talbot, J.M., Tison, S.: Satisfiability of a spatial logic with tree variables. In: 21st International Workshop on Computer Science Logic (CSL 2007). Volume 4646 of LNCS, Springer (2007), 130–145.
11. Charatonik, W.: Automata on dag representations of finite trees. Technical Report Technical Report MPI-I-99-2-001, Max-Planck-Institut für Informatik, Saarbrücken, Germany (1999).
12. Anantharaman, S., Narendran, P., Rusinowitch, M.: Closure properties and decision problems of dag automata. Inf. Process. Lett. **94**(5) (2005), 231–240.
13. Comon, H., Dauchet, M., Gilleron, R., Jacquemard, F., Löding, C., Lugiez, D., Tison, S., Tommasi, M.: Tree Automata Techniques and Applications. `http://tata.gforge.inria.fr` (2007).
14. Seidl, H.: Haskell overloading is DEXPTIME-complete. Information Processing Letters **52**(2) (1994), 57–60.
15. Alur, R., Madhusudan, P.: Visibly pushdown languages. In: 36th Annual ACM Symposium on Theory of Computing, STOC. ACM (2004), 202–211.
16. Chabin, J., Réty, P.: Visibly pushdown languages and term rewriting. In: 6th International Symposium on Frontiers of Combining Systems (FroCos). Volume 4720 of LNCS, Springer (2007), 252–266.
17. Comon-Lundh, H., Jacquemard, F., Perrin, N.: Tree automata with memory, visibility and structural constraints. In: 10th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS). Volume 4423 of LNCS, Springer (2007), 168–182.