Florent Jacquemard
Francis Klay
Camille Vacher

Rigid Tree Automata

Research Report LSV-08-27

October 2008

Laboratoire Spécification et Vérification

CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE

Ecole Normale Supérieure de Cachan
61, avenue du Président Wilson
94235 Cachan Cedex France

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: two subtrees reaching the same rigid state in a run must be equal. 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 that is decidable 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 set 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 a 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 the addition of equality and disequality tests in TA transition rules (the classes [4, 11] 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 [8]. 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 [13]). 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 be very successful in verification see e.g. [5, 14]. 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 [1] 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. [16]).

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 restricted TRS called linear invisibly, whereas this closure is 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 $\mathsf{decrypt}$ [1]. It is possible to built an 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 (only a finite number of such state is needed). 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).

1 Preliminaries

A *signature* Σ is a finite set of function symbols with arity. We write Σ_m 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 Σ. 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* 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 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 said *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 rule 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 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 $t(p) = f \in \Sigma_n$ $(n \geq 0)$, $f\big(r(p.1), \ldots, r(p.n)\big) \to r(p) \in \Delta$. 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 on which there exists a run r of $\mathcal{A}$ such that $r(\varepsilon) \in q$. 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

We now introduce the class of rigid tree automata studied in this paper, demonstrate their expressiveness on some examples, study its closure properties, and compare them to other classes of TA.

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 like 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 $\mathcal{A} = \big\langle \{q, q_r, q_\mathsf{f}\}, \{q_r\}, \{q_\mathsf{f}\}, \{a \to q|q_r$[4]$, b \to q|q_r, f(q,q) \to q|q_r, f(q_r, q_r) \to q_\mathsf{f}\} \big\rangle$. A successful run of $\mathcal{A}$ on $f(f(a,a), f(a,a))$ 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 $\Delta = \{a \to q|q_r, b \to q|q_r, f(q,q) \to$

[4] $a \to q|q_r$ is an abbreviation for $a \to q$ and $a \to q_r$.

$q|q_r, f(q,q_r) \to q', f(q_r,q) \to q', f(q,q') \to q', f(q',q) \to q', <(q_r,q') \to q_\mathsf{f}\}$. 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. $\diamond$

This example cannot be generalized to the characterization of a maximal subterm amongst some subterms, as shown by a counter example in the proof of Proposition 2.
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 $\neq$). $\diamond$

2.2 Pumping Lemma

We generalize to RTA the pumping (or iteration) lemma for TA (or finite automata). 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.

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 non trivial (non-variable) 1-contexts C' and D 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 $d(t) > (|Q|+1)|R|$, and r be a successful run on t, and let p be a position in $\mathcal{P}os(t)$ of length at least $(|Q|+1)|R|$. By 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 occur 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 $D = (t_{p_1})[x_1]_{p_2-p_1}$.

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). Let π_1 be the maximal position of a rigid state in r smaller than p_0 wrt prefix ordering. Let $q_r = r(\pi_1)$ and let $\pi_2, \ldots, \pi_k$ 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 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 fresh 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 follow 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 RTA.

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 [13] 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 decidability of the emptiness problem is open for the whole TAGED class. A decidable subclass of TAGED is identified in [12] 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 [13] where a TAGED+ is transformed into a RTA in order to decide emptiness, at the price of an exponential blowup. This construction is used in the proof of Theorem 2 below. 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. [4, 11]). The emptiness problem undecidable in general [**?**]. Two decidable subclasses have been identified: BTTA [4] and Reduction Automata [11] (RA); the complexity of emptiness is at least EXPTIMEfor these subclasses. 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, not recognizable by RTA, is recognizable by TAC.

TA1M. TA with one memory [8, 10] compute bottom in terms with an auxiliary memory which has the shape of a tree. Moreover, they can perform equality tests on the memory contents, and hence simulate some tests of the TA with constraints, but are also limited to local tests. The classes of languages of TA1M and RTA are orthogonal. For instance, $\{f(g^n(f(x,y)), f(x, g^n(f(x,y)))) \mid x, y \in \mathcal{T}(\Sigma)\}$ can be recognized by an RTA but not by a TA1M.

DAG automata (DA) [7] 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 [3]. 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

Definition 2. *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 $\mathcal{A} = \langle \{q, q_r\}, \{q_r\}, \{q\}, \{a \to q, g(q) \to q_r, g(q_r) \to q\} \rangle$, 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 bottom-up DTAs are as expressive as TAs. 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.

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

Proof. Let $\Sigma = \{f:2, a:0\}$. 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 $r'|_{p'} = r$. So 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 $t'[t'_{p'_2}]_{p'_1}$. 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' 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''
- a position $p'_2.\pi$, $\pi \in \mathcal{P}os(t'|_{p'_2})$, and then the position $p'_1.\pi$ in t'' has the same subtree and the same rigid state
- a position $p'_1.\pi$, $\pi \in \mathcal{P}os(t'|_{p'_1})$, incomparable with t_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.

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 $\{f(g(t), g(t)) \mid t \in \mathcal{T}(\Sigma \setminus \{g\}\}$ is recognized by a DRTA but not by a TA. □

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 exists 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. Let us assume *wlog* that the state sets 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 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 $\{f\big(\langle q_{11}, q_{21}\rangle, \ldots, \langle q_{1n}, q_{2n}\rangle\big) \to \langle q_1, q_2\rangle \mid f(q_{i1}, \ldots, q_{in}) \to q_i \in \Delta_i, i = 1, 2\}$. Moreover, the equality relation of $\mathcal{B}$ is $R_= = \big\{\big\langle\langle q_{r_1}, q_2\rangle, \langle q_{r_1}, q_2'\rangle\big\rangle \mid q_{r_1} \in R_1\big\} \cup \big\{\big\langle\langle q_1, q_{r_2}\rangle, \langle q_1', q_{r_2}\rangle\big\rangle \mid q_{r_2} \in R_2\big\}$. A construction is proposed in [13] 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. □

Note that the construction for the intersection of RTA preserves determinism but not for the union. The following lemma shows that the exponential size of the intersection automaton constructed in Theorem 2 is a lower bound.

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 $L(\mathcal{A}_\times) \cap L(\mathcal{A}_r) = \emptyset$.*

Proof. Let $\Sigma_d = \Sigma \uplus \{0 : 0, d : 2\}$. The RTA on Σ_d, $\mathcal{A}_r = \big\langle\{q, q_r, q_{\mathsf{f}}\}, \{q_r\}, \{q_{\mathsf{f}}\}, \{0 \to q_{\mathsf{f}}, d(q_r, q_{\mathsf{f}}) \to q_{\mathsf{f}}\} \cup \{f(q, \ldots, q) \to q | q_r \mid f \in \Sigma\}\big\rangle$ 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. □

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 TA (given n TAs $\mathcal{A}_1, \ldots, \mathcal{A}_n$, do we have $L(\mathcal{A}_1) \cap \ldots \cap L(\mathcal{A}_n) \neq \emptyset$?), a problem which is known to be EXPTIME-complete [17] Since by Theorem 2, the intersection of two RTA is an RTA, and the emptiness of RTA can be decided in linear time (Theorem 4 below), we conclude that an RTA for the intersection must be exponential in some cases. Moreover, in the above construction, $\mathcal{A}_\times$ is a TA if every $\mathcal{A}_i$ ($i \leq n$) is a TA. Hence the intersection of a RTA with a TA also leads to an exponential construction.

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 complete 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 chose 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 non-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
$\neg$	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		

Fig. 1. Summary of closure and decision results

We show below that deciding emptiness for an RTA amounts to decide emptiness for the underlying TA.

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

Proof. Let $\mathcal{A} = \langle\Sigma, Q, R, F, \Delta\rangle$ and let $\mathit{rigid}(\mathcal{A}) = \langle\Sigma, Q, Q, F, \Delta\rangle$ (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. [9]). with an algorithm marking

the inhabited states of $ta(\mathcal{A})$ and appropriate data structure for the transitions rules. The idea is that if $L(ta(\mathcal{A}))$ is not empty, then there exists $t \in L(ta(\mathcal{A}))$ and a run r of $ta(\mathcal{A})$ on t that respects the rigidity condition for all the states of $\mathcal{A}$ (two subterms of t at positions of the same state in r are equal).

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 $t_q \in L(rigid(\mathcal{A}), q) \subseteq L(\mathcal{A}, q) \subseteq L(ta(\mathcal{A}), q)$ and a run r_q of $rigid(\mathcal{A})$ on t_q. At the beginning, each t_q and r_q are set 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 [9]) 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 by definition.

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

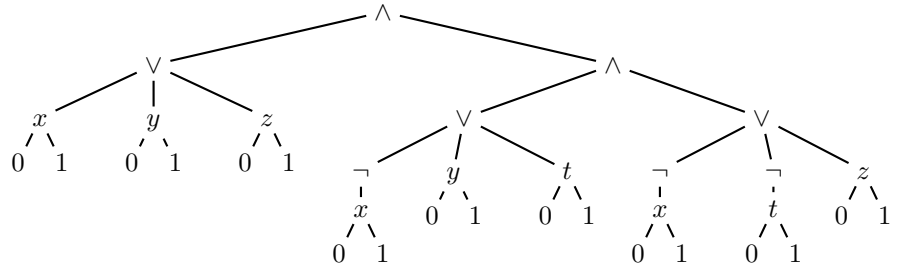
Theorem 5. *Membership is* NP*-complete for RTA (*PTIME *for DRTA).*

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. It can be computed in PTIME and it can be checked in PTIME that this labelling is a successful run.

In order to show that the membership problem on RTAs is NP-hard, we propose a reduction of 3-SAT. Let us consider an instance 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 by $\vee(t_1, t_2, t_3)$ where t_1, t_2, t_3 encode respectively ℓ_1, ℓ_2, ℓ_3. Finally we encode a conjunction of disjunctions $D_1 \wedge \cdots \wedge D_n$ by $\wedge(t_1, \ldots, \wedge(t_{n-1}, t_n))$ where each t_i, $i \leq n$, is the encoding of D_i.

For instance, the 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 the following tree:



We define a 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 $\Delta = \{0 \to q_x | q_{\neg x}, 1 \to q_x | q_{\neg x} \mid x \in V\} \cup \{x(q_x, q_{\neg x}) \to q_0, x(q_{\neg x}, q_x) \to q_1 \mid x \in V\} \cup \{\vee(q_0, q_0, q_0) \to q_0\} \cup \{\vee(q, q', q'') \to q_1 \mid \text{at least one of } q, q', q'' \text{ is } q_1, \text{ and the others are } q_0\} \cup \{\neg(q_0) \to q_1, \neg(q_1) \to q_0, \wedge(q_1, q_1) \to q_1, \wedge(q_0, q_1) \to q_0, \wedge(q_1, q_0) \to q_0, \wedge(q_0, q_0) \to q_0\}$. Both the automata and the tree are linear in size relatively to the size of the 3-SAT instance. We show that $\mathcal{A}$ recognizes t iff the corresponding 3-SAT instance 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 rigid 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. □

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 [17]. □

Theorem 7. *Universality is undecidable for RTA.*

Proof. We reduce the Post Correspondence Problem (PCP). 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 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. We define two sets U and V of terms of $\mathcal{T}(\Sigma)$ as the smallest sets containing $\bot$ and such that if $u \in U$ then $u' := f_i(u_i w_1, u, v_i w_3) \in U$ for all $w_1, w_3 \in \mathcal{T}\big(\Gamma \cup \{\bot\}\big)$. If moreover $u \neq \bot$ and $w_1 = u|_1$ and $w_3 = u|_3$ then $u' \in V$. Note that U is regular and $V \subset U$.

Every solution of P is represented by a term $v \in V \setminus \{\bot\}$ such that $v|_1 = v|_3$. We can construct a RTA $\mathcal{A}$ on Σ recognizing the set of terms which are not a representation of solution as above. It is defined as the union of several TAs and RTAs, each one corresponding to one reason for not being a solution of P:

- a TA recognizing the complement of U in $\mathcal{T}(\Sigma)$,
- a RTA recognizing exactly the terms of $u \in U$ of the form $f_i(\ldots)$ and such that $u|_1 \neq u|_3$. This RTA can be constructed as the intersection of TA for U and a RTA similar to the one of Example 4.
- a RTA recognizing exactly the terms of $u \in U$ 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 4.

With this construction, $L(\mathcal{A}) = \mathcal{T}(\Sigma)$ iff P has no solution. □

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.*

Proof. 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 $E_{\mathcal{A}} := \{\langle q, q' \rangle \mid \exists f(\dots q \dots) \to q' \in \Delta\}$. A path in $G_{\mathcal{A}}$ is a finite sequence of states $q_1, \dots, q_n$ 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 it 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 4, 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. □

4 Rewrite Closure

Following the motivations presented in introduction, we study here the closure under term rewriting of RTA languages. We observe first that in general the rewrite closure of a RTA language is not an RTA language and even not recursive for linear collapsing TRS. We show then that, under a syntactical restriction, namely for a linear *invisibly* TRS $\mathcal{R}$, is decidable whether a given tree belongs to the rewrite closure of a given RTA language under rewriting with $\mathcal{R}$.

4.1 Linear Collapsing Rewrite Systems

The closure of a RTA language under rewriting is unfortunately not a RTA language, even for very restrictive TRS.

Proposition 2. *$\mathcal{R}^*(L)$ is not a RTA language in general when L is a RTA language and $\mathcal{R}$ a linear and collapsing TRS.*

Proof. 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 $Q = \{q_0, q_1, q_r, q_{\mathsf{f}}\}$, $R = \{q_r\}$, $F = \{q_{\mathsf{f}}\}$, and $\Delta = \{0 \to q_0, g(q_0) \to q_0 | q_r, f(q_r) \to q_1, f(q_1) \to q_1, h(q_r, q_{1,2}) \to q_{\mathsf{f}}, h(q_{1,2}, q_{1,2}) \to q_2, h(q_{\mathsf{f}}, q_{1,2}) \to q_{\mathsf{f}}, \}$ where $q_{1,2}$ is either q_1 or q_2. Every term of $L(\mathcal{A})$ has the form $H\big[g^m(0), f^*(g^m(0)), \dots, f^*(g^m(0))\big]$ 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.

- H is a context made of the symbol h only,
- the leftmost argument of the context contains m symbol g,
- the other arguments of the context consist in an arbitrary number of f followed by m g and finished by a 0.

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 or equal than the number of g on the leftmost argument. We show in [15] that it is not a RTA language, with arguments similar to those of Section 2.2. Assume that $\mathcal{R}^*\big(L(\mathcal{A})\big)$ is recognized by a RTA $\mathcal{B}$ with n states and d rigid states. We can assume wlog that $d < n$. Otherwise $\mathcal{A}$ could not recognize terms of depth larger than n. Let $t \in \mathcal{R}^*\big(L(\mathcal{A})\big)$ be of the form $H[t_0, \ldots, t_{d+1}]$ where for each $0 \leq i \leq d+1$, $t_i = g^{(r+2-i)(n+1)}(0))$ (not that t contains no symbol f). Let r be a run of $\mathcal{B}$ on t. We show, by a pumping argument, that there is one $i \geq 1$ such that we can we can increase as much as we want the number of g in t_i, while keeping the term recognized (a contradiction). It is not possible to apply the pumping Lemma 1 here, because we do not have enough control on the context C in this lemma. However, it is possible to apply the same principle in order to expand one t_i while preserving the rigidity condition, with a construction specific for the example.

First, note that the t_i's are pairwise distinct. It follows that there is 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 $\mathcal{B}$ cannot occur twice in some t_i. By a pigeon hole 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 exist one non-rigid state of $\mathcal{B}$ 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 an accepting run of $\mathcal{B}$ on $t'_j := H[t_0, \ldots, t_{i-1}, g^{jk}(t_i), t_{i+1}, \ldots, t_{d+1}]$. But for for a j sufficiently large, $t'_j \notin \mathcal{R}^*\big(L(\mathcal{A})\big)$, a contradiction. □

Note that the term t in the above counter example is a $\mathcal{R}$-normal form (it does 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 neither an RTA language in general, when $\mathcal{A}$ is a RTA and $\mathcal{R}$ a linear and collapsing TRS. 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. A solution of this PCP instance 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 $\Sigma = \{g_i{:}1, f_i{:}1 \mid i \leq n\} \cup \{a{:}1 \mid a \in \Gamma\} \cup \{0{:}0, k{:}1, h : 2\}$, and $L = \{h(s, k(s)) \mid s = f_{i_1}(g_{i_1}(\ldots f_{i_m}(g_{i_m}(w(0))))), m > 0, w \in \Gamma^*\}$[5] 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 $k(f_i(v_i(x))) \to x$

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

$(i \leq n)$. The tree language L is recognizable by a RTA on Σ and $h(0,0) \in \mathcal{R}^*(L)$ 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. $s = f_{i_1}(g_{i_1}(\ldots f_{i_m}(g_{i_m}(w(0)))))$ and $w = u_{i_1} \ldots u_{i_m} = v_{i_1} \ldots v_{i_m}$). 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 $k(f_{i_1}(v_{i_1}(0)))$ using the two next rules of $\mathcal{R}$. This latter term is in turn reduced to 0 using the last rule of $\mathcal{R}$.

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 do 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. On the other hand, assuming s minimal, and having the k at the top of the right branch imposes to use only the last three rule 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 describes a solution of P. □

4.2 Linear Invisibly Rewrite Systems

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) [2]. VPA define a subset of context-free languages closed under intersection, and were generalized to tree recognizers in [6, 10]. The idea is 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 [6], 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 the class TRS for which membership modulo is decidable.

Definition 3. *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 in the proof of Proposition 2 is invisibly but not the one for Theorem 10.

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. Let us first sketch the proof with a TRS $\mathcal{R}$ containing the two first rewrite rules of Example 7, namely $\mathcal{R} = \{\mathsf{fst}(\mathsf{pair}(x_1, x_2)) \to x_1, \mathsf{snd}(\mathsf{pair}(x_1, x_2)) \to x_2\}$.

Let $\mathcal{A} = \langle Q, R, F, \Delta\rangle$ be an 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 $t = \mathsf{f}(\mathsf{pair}(0, 0), 0)$. The language of $\mathcal{A}$ is the set of terms of the form $\mathsf{f}(\{\mathsf{fst}, \mathsf{snd}\}^*(s), \mathsf{fst}, \mathsf{snd}\}^*(s))$ for some $s \in \mathcal{T}\big(\{0, \mathsf{pair}\}\big)$.

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 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 $\frac{q_1}{q_0}$. Indeed, there is only one possible pair for the root position: $\frac{q_\mathsf{f}}{q_\mathsf{f}}$ (we want a final state to occur at this position). Consequently, according to the transitions of $\mathcal{A}$, there are two possible pairs for the position 1: $\frac{q_r}{q_r}$ and $\frac{q_1}{q_0}$ and $\frac{q_0}{q_0}$ for all the positions below 1. Since there is only one transition for the symbol 0 there is only one possible pair for the position 2: $\frac{q_1}{q_0}$. 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 corresponding to the distinct occurrences of q_r, which are generated by non-terminal of grammars as above. We use the fact that the emptiness of intersection is decidable for visibly context free tree grammars.

Let us now describe the general algorithm. Let $\mathcal{A} = \langle Q, R, F, \Delta\rangle$ be a RTA, $\mathcal{R}$ an inverse-visibly TRS and $t \in \mathcal{T}(\Sigma)$ a term in normal form. As in the example, for each lhs term of $\mathcal{R}$, we consider every labeling with pairs of states. More than a labeling of a left-hand term, we want to know which rigid states occur in this labeling, directly or indirectly, at which positions, and the partial order between terms induced by the labeling. So we will work on tuples of the following form $\langle \frac{q_1}{q_2}, \ell, \mathit{lbl}, \mathit{occ}, \mathit{occ}_x, <\rangle$ where

- $q_1, q_2 \in Q$, they may be equal,
- 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,
- occ is a set of couple $\langle q_r, p\rangle$ where $q_r \in R$ is a rigid state 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.

Several constraints are assumed for lbl:

- if $\frac{q}{q'}$ is the pair at position ε, then $q = q_1$,
- if $\frac{q}{q'}$ is the pair at the position of the variable x, then $q' = q_2$,
- for every position $p \in \mathcal{P}os(\ell)$, such that $\ell(p)$ is a function symbol of arity n, $q_1 \dots q_n$ are the first states of the pairs at respective positions $p.1, \dots p.n$ in lbl, and q is the second state of the pair at position p, we have $\ell(p)\big(q_1 \dots q_n\big) \to q \in \Delta$,
- If $\langle q_r, p\rangle \in occ_x$, p is a position in ℓ between the root and the position of the variable x.

The restrictions on the partial order $<$ will be given in the next step.

First, we consider the set of tuples T consisting, for each non-rigid state q, of the tuple $\langle \frac{q}{q}, x, \frac{q}{q}, \emptyset, \emptyset, \emptyset\rangle$, and for each rigid state q_r, of the tuple $\big\langle \frac{q_r}{q_r}, x, \frac{q_r}{q_r}, \{\langle q_r, \varepsilon\rangle\}, \{\langle q_r, \varepsilon\rangle\}, \emptyset\big\rangle$. Note that in the first tuple, there is no rigid state in the labeling, so occ and occ_x are both empty, while in the second tuple, q_r occurs at position ε which is the only position, so both the root and the x position, so $occ = occ_x = \{\langle q_r, \varepsilon\rangle\}$. Note also that in both cases, the partial order $<$ is empty since there is not more than one rigid state in the labeling.

We recursively build every tuple possible respecting the conditions previously listed, and add them to the T set. For each lhs term ℓ of $\mathcal{R}$ we consider each function ξ that associates to each position of ℓ a tuple of T. We check that those tuples verify the following conditions

- There is no couple of distinct states q_1 and q_2 such that $q_1 < q_2$ in one tuple and $q_2 < q_1$ in another tuple
- There is no rigid state q_r such that there exists a couple of distinct positions p and $p' = p.\omega$ such that $q_r \in occ_x$ in $\xi(p)$ and $q_r \in occ$ in $\xi(p')$

If they do, we try to build a new tuple $\big\langle \frac{q_1}{q_2}, \ell, lbl, occ, occ_x, <\big\rangle$ such that

- q_1 is the first state of the pair of $\xi(\varepsilon)$
- q_2 is the second state of the pair of $\xi(p_x)$ where p_x is the position of the variable x in ℓ
- lbl is the labeling of ℓ where for each position p $lbl(p)$ is the pair of state of $\xi(p)$
- $occ = \{\langle q_r, p\rangle \mid$ it exists a pair $\langle q_r, p'\rangle$ in the set occ of $\xi(p)\}$
- $occ_x = \{\langle q_r, p\rangle \mid$ it exists a pair $\langle q_r, p'i\rangle$ in the set occ_x of $\xi(p)$, and p is a position between the root and the variable x of $\ell\}$

This construction respects the constraints given in the tuples definition, but the partial order is not yet given. We first define a relation $\prec$ by

– if $q_{r_1} < q_{r_2}$ in some tuple, then $q_{r_1} \prec q_{r_2}$
– for all p and $p' = p.\omega$, $|\omega| > 0$, if there exists q_{r_1} in occ_x in $\xi(p)$ and q_{r_2} in occ in $\xi(p')$, then $q_{r_1} \prec q_{r_2}$

We take the transitive closure of $\prec$: $\prec^+$. If $\prec^+$ is irreflexive and asymmetric, it defines a strict partial order which is compatible with the order of $\xi(p)$ for each $p \in \mathcal{P}os(\ell)$. So we have defined a correct tuple $\left\langle \frac{q_1}{q_0}, \ell, lbl, occ, occ_x, < \right\rangle$ with $<=\prec^+$ and we add it to the set T. Otherwise, we do not add it, and we consider a new function ξ that associates a tuple to each position of ℓ or we consider another lhs term of $\mathcal{R}$.

We repeat this procedure until we do not add any new tuple to T. It will end since the number of tuple is finite: the number of states, the number of lhs term of ℓ, the number of labeling of a lhs term by pairs of state and the number of strict partial orders on rigid states are all finite. Note that the way we built the tuples allows us to make two remarks:

1. The tuples such that ℓ is a single variable x instead of a lhs term of $\mathcal{R}$ are the ones in the initial T set and only them
2. The tuples which have for pair of states a pair $\frac{q_r}{q_r}$ with $q_r \in R$ are all in the initial T set.

Once this procedure is done, we repeat it another time, but instead of considering any lhs term of $\mathcal{R}$, we apply it to the term t, with the extra constraint that the first state of the pair of $\xi(\varepsilon)$ should be a final state of $\mathcal{A}$. So we associate each position of t with a tuple, we check that the different partial orders are coherent, that the induced labeling of t by pairs of state respect the transition rules of $\mathcal{A}$ and that the transitive closure of the new relation between rigid states induced by the labeling is still a partial order.

We now define a visibly pushdown language that describes all the terms of $ta(\mathcal{A})$ with accepting run coherent with the induced labeling (the final state at root is the first state of $\xi(\varepsilon)$, each node of t is labeled by the second state of the tuple associated to its position ...) which rewrite to t in a way that they respect the partial orders on rigid states of the tuples but not necessary the rigid condition. Instead of giving an automaton, we give a visibly tree grammar that describes the language. We consider each tuple of T as a non-terminal symbol of arity 1 of the grammar. We also add non-terminal symbols of arity 1 of the form $\langle \frac{q_1}{q_2}, rig, rig_x, \prec \rangle$ where $q_1, q_2 \in Q$, $rig_x \subseteq rig \subseteq R$ and $\prec$ is a strict partial order on R. The terminal symbols are the function symbols of Σ.

For each tuple tup such that $\ell = x$ we add the rule $tup(z) \to z$ to the grammar. For each tuple $tup = \left\langle \frac{q_1}{q_2}, \ell, lbl, occ, occ_x, < \right\rangle$ where ℓ is a lhs term of $\mathcal{R}$ we add the following grammar rule: $tup(z) \to tr[tup, \varepsilon](z)$ where $tr[tup, p]$, with p a position of ℓ is a term with non-terminals defined by $tr[tup, p](z) = \left\langle lbl(p), \{q_r \mid \langle q_r, p \rangle \in occ\}, \{q_r \mid \langle q_r, p \rangle \in occ_x \text{ and } p \text{ is a position between the root and the variable } x \text{ of } \ell\}, < \right\rangle (\ell(p)(tr[tup, p.1] \dots tr[tup, p.n]))$ where n is the arity of $\ell(p)$. $tr[tup, \varepsilon](z)$ denotes $tr[tup, \varepsilon]$ where z instantiates x.

For each non-terminal symbol $\langle \frac{q_1}{q_2}, rig, rig_x, \prec \rangle$, we add to the grammar the rules $\langle \frac{q_1}{q_2}, rig, rig_x, \prec \rangle \to tup = \left\langle \frac{q_1}{q_2}, \ell, lbl, occ, occ_x, < \right\rangle$ such that

– For all $q_r \in rig$, it exists a pair $\langle q_r, p\rangle \in occ$
– For all $q_r \in rig_x$, it exists a pair $\langle q_r, p\rangle \in occ_x$
– If $q_{r_1} < q_{r_2}$ then $q_{r_1} \prec q_{r_2}$

Note that reciprocity is never assumed.

Finally, the initial symbol is S and it can only operates one rule $S \to tr[t]$ where $tr[t]$ is built like the $tr[tup, t]$ seen above, except that instead of adding non-terminal of shape $\langle \frac{q_1}{q_2}, rig, rig_x, \prec\rangle$ in top of each function symbol, it appends the tuple associated to the current position.

The above grammar is invisibly and generates terms of $ta(\mathcal{A})$ that rewrite to t and respect a partial order on rigid states. However, they do not necessary respect rigidity conditions. In order to ensure that the constraints is fully respected, we consider separately the sublanguages under the non-terminal symbols of the shape $\langle \frac{q_r}{q_r}, x, \frac{q_r}{q_r}, \{\langle q_r, \varepsilon\rangle\}, \{\langle q_r, \varepsilon\rangle\}, \emptyset\rangle$. Since they can be generated by an invisibly grammar, they are also visibly pushdown languages, so they are closed under intersection, and the emptiness is decidable.

To fully and correctly generate the language of terms of $L(\mathcal{A})$, we use the grammar defined above to find the sublanguage under each occurrence of a rigid state, beginning by the maximal ones 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 of $L(\mathcal{A})$, respecting the rigidity condition for all rigid state, does rewrite to t. □

Conclusion and Further Work

We have presented the class of Rigid Tree Automata and its main closure and decision 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 invisibly TRS.

We believe that this class of tree automata is accurate in procedures of verification of some infinite state systems. 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 like $\mathsf{decrypt}(\mathsf{encrypt}(x, y), y) = x$. We are also interested to the symmetric form of the TRS of [6], whose rhs are not single variables but have the form $f(x_1, \ldots, x_n)$.

Some other possible extensions are the study of binary relations on terms defined by RTA (like in [9], chapter 3), the extension of RTA to equalities test

modulo equational theories like in [16], or the addition of disequality constraints in order to obtain closure under complement and correspondence with logics.

References

1. M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *28th ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages (POPL)*, pages 104–115, 2001.
2. R. Alur and P. Madhusudan. Visibly pushdown languages. In *Proc. of the 36th Annual ACM Symposium on Theory of Computing, Chicago (STOC)*, pages 202–211. ACM, 2004.
3. S. Anantharaman, P. Narendran, and M. Rusinowitch. Closure properties and decision problems of dag automata. *Inf. Process. Letters*, 94(5):231–240, 2005.
4. B. Bogaert and 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*, pages 161–171, 1992.
5. A. Bouajjani and T. Touili. On computing reachability sets of process rewrite systems. In *Proc. 16th Int. Conf. on Term Rewriting and Applications (RTA)*, vol. 3467 of *LNCS*, pages 484–499, 2005.
6. J. Chabin and P. Réty. Visibly pushdown languages and term rewriting. In *Proc. 6th Int. Symp. Frontiers of Combining Systems (FroCos)*, vol. 4720 of *LNCS*, pages 252–266, 2007.
7. W. Charatonik. Automata on dag representations of finite trees. Technical Report Technical Report MPI-I-99-2-001, MPI Saarbrücken, Germany, 1999.
8. H. Comon and V. Cortier. Tree automata with one memory, set constraints and cryptographic protocols. *Theoretical Computer Science*, 331(1):143–214, Feb. 2005.
9. H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, C. Löding, D. Lugiez, S. Tison, and M. Tommasi. *Tree Automata Techniques and Applications.* `http://tata.gforge.inria.fr`, 2007.
10. H. Comon-Lundh, F. Jacquemard, and N. Perrin. Tree automata with memory, visibility and structural constraints. In *Proc. 10th Int. Conf. on Found. of Soft. Sc. and Computation Struct. (FoSSaCS)*, vol. 4423 of *LNCS*, pages 168–182, 2007.
11. M. Dauchet, A.-C. Caron, and J.-L. Coquidé. Automata for Reduction Properties Solving. *Journal of Symbolic Computation*, 20(2):215–233, 1995.
12. E. Filiot, J.-M. Talbot, and S. Tison. Satisfiability of a spatial logic with tree variables. In *Proc. 21st Int. Workshop on Computer Science Logic (CSL)*, vol. 4646 of *LNCS*, pages 130–145, 2007.
13. E. Filiot, J.-M. Talbot, and S. Tison. Tree automata with global constraints. In *12th Int. Conf. in Developments in Lang. Theory (DLT)*, vol. 5257 of *LNCS*, pages 314–326, 2008.
14. T. Genet and F. Klay. Rewriting for Cryptographic Protocol Verification. In *Proc. 17th Int. Conf. on Automated Deduction (CADE)*, vol. 1831 of *LNCS*, 2000.
15. F. Jacquemard, F. Klay, and C. Vacher. Rigid tree automata. Technical Report RRLSV-0827, Laboratoire Spécification et Vérification, 2008.
16. F. Jacquemard, M. Rusinowitch, and L. Vigneron. Tree automata with equality constraints modulo equational theories. *Journal of Logic and Algebraic Programming*, 75(2):182–208, 2008.
17. H. Seidl. Haskell overloading is DEXPTIME-complete. *Inf. Process. Letters*, 52(2):57–60, 1994.