Tree automata with global constraints for the verification of security properties

Camille Vacher

THESE DE DOCTORAT
DE L'ECOLE NORMALE SUPERIEURE DE CACHAN

Présentée par

Monsieur VACHER Camille

pour obtenir le grade de

DOCTEUR DE L'ECOLE NORMALE SUPERIEURE DE CACHAN

Domaine :
Informatique

Sujet de la thèse :

Automates d'arbres à contraintes globales pour la vérification de propriétés de sécurité

Thèse présentée et soutenue à Cachan le 07/12/2010 devant le jury composé de :

Jean Goubault-Larrecq	Professeur	Directeur de thèse
Florent Jacquemard	Chargé de recherche	Co-directeur de thèse
Christof Loeding	Professeur	Examinateur
Jean-Marc Talbot	Professeur	Rapporteur
Sophie Tison	Professeure	Présidente
Igor Walukiewicz	Professeur	Rapporteur

Nom du Laboratoire : Laboratoire de Spécification et Vérification
ENS CACHAN/CNRS/UMR : 8643
61, avenue du Président Wilson, 94235 CACHAN CEDEX (France)

Remerciements

Je tiens tout d'abord à remercier Igor Walukiewicz et Jean-Marc Talbot pour m'avoir fait l'honneur d'accepter d'être rapporteurs de cette thèse. Leur lecture attentive de mon manuscrit et leurs commentaires m'ont été d'une aide précieuse. Je remercie également Sophie Tison et Christof Loeding d'avoir accepté de participer au jury.

Merci également à mon directeur de thèse officiel, Jean Goubault-Larrecq, d'avoir accepter cette responsabilité.

Je remercie Florent Jacquemard, mon co-directeur de thèse, avec qui j'ai travaillé avec plaisir pendant plus de trois ans. La plupart des résultats présentés ici doivent leur origine à de longs après-midi passés ensemble devant un tableau. Qu'il ait su gérer mes excès d'enthousiasme ou de scepticisme n'est pas le moindre de ses mérites.

Merci à Francis Klay, de France Télécom, pour son accueil, pour les échanges scientifiques, malheureusement trop peu nombreux, et pour s'être assuré que tout se déroulait au mieux pour moi tout au long de ma thèse. Merci également à Yves Quemener, pour avoir fait un suivi bien apprécié de ma situation administrative durant ces trois années.

Je remercie Guillem Godoy, pour son accueil, sa bonne humeur et ses idées éclairantes. Ces trois jours de travail en commun à Barcelone ont constitué une étape cruciale dans le déroulement de ma thèse. Je lui en suis redevable.

Si cette thèse a pu voir le jour, c'est grâce à la bonne ambiance qui règne dans les bâtiments du LSV et aux personnes qui s'y trouvent, contribuant à en faire un peu plus qu'un simple lieu de recherche. Merci notamment à ceux et celles avec qui j'ai partagé un bureau, des soirées, des repas, ou plus que cela. Une pensée donc pour Thomas, Pierre C., Étienne, Mathilde, Jules, Arnaud, Antoine, Diego, Amélie, Sylvain, Pierre B., Alban et Jean-Loup. D'autre part, je tiens à remercier Geneviève, Ahmad, Catherine, Audrey, Virginie et Isabelle pour tous les services rendus pendant mon séjour au LSV.

Ce manuscrit n'est pas que le résultat d'un travail en laboratoire, mais doit beaucoup à toutes les personnes que j'ai rencontrées ces dernières années, et avec qui j'ai pu discuter en dehors de tout cadre scientifique. Parmi elles se trouvent évidemment mes colocataires de

la rue Saint-Michel à Lyon, des rues Bobillot, Crimée et de l'Ourcq à Paris, ainsi que de la rue Solférino à Lille. Je lève un verre à leur santé.

Enfin, il est quelques personnes qui me soutiennent depuis le début, ce qui est d'autant plus apprécié et courageux que la réciprocité n'est pas mon fort. Pour cela, je présente mes excuses et j'adresse toute mon affection à Frédérique, Gilles, Barbara, Marthe et Jacques.

Introduction

In system and software verification, one classical problem is to modelize infinite structures with only finitely many informations. Automata on words have been used with this approach since decades. They are basically finite state machines that allow to represent potentially infinite languages of words. An intuitive way to describe automata is to say that they have a finite number of states, and each of these states will correspond to a (potentially infinite) class of words that have the same behavior when viewed as prefixes of other words. Then, the automaton will have several transition rules that define the relations between all these states. The languages recognized by automata are called regular. The use of automata spread quickly and they are still widely used in many fields of computer science. Among the most evident reasons of their success, one can cite their expressiveness, the decidability of many important problems, and the closure properties of regular languages.

Automata are as expressive as rational expressions: any language that can be described as a rational expression can be described as an automaton language. The problem to know whether a word can be recognized by a given automaton, or whether there even exists such a word, are decidable, in a reasonable complexity. Moreover, regular languages can be combined, and automata can recognize the union or intersection of two regular languages, and also the complement of a regular language.

Automata have been quickly generalized from words to terms in order to finitely represent infinite languages of structures shaped like trees. Most of the properties of automata on words are still true on trees. In particular, the problems and properties mentioned above are still valid for Tree Automata. They can be used to represent infinite sets of states of a system or a program (in the latter case, a term can represent the program itself), messages exchanged in a communication protocol, XML documents... In these settings, the closure properties of TA languages permit incremental constructions and verification problems can be reduced to TA problems decidable in polynomial time like emptiness (is the language recognized by a given TA empty) and membership (is a given term t recognized by a given TA).

Tree automata techniques are widely used in several domains like automated deduction (see *e.g.* [CLDG$^+$07]), static analysis of programs [BT05] or protocols [VGL07, FGT04], and XML process-

ing [Sch07]. Hence they are an important tool in security issues such as communications protocol verification, or XML updates analysis. A severe limitation of standard tree automata (TA) is however that they are not able to test for equality (isomorphism) or disequality between subterms in an input term. For instance, the language of terms described by a non-linear pattern of the form $f(x, x)$ is not regular (*i.e.* there exists no TA recognizing this language). Such tests would also be useful for expressing integrity constraints . Similar problems are also frequent in the context of XML documents processing. Intuitively, an XML document is a textual representation of the storage of data in a tree structure; in other words, it is a finite labeled unranked tree. XML documents are commonly represented as labeled trees, and they can be constrained by XML schemas, which define both typing restrictions and integrity constraints. All the typing formalisms currently used for XML are based on finite tree automata. The key constraints for databases are common integrity constraints expressing that every two distinct positions of a given type have different values (see *e.g.* [FL02]). This is typically the kind of constraints that can not be characterized by TA.

One first approach to overcome this limitation of TA consists in adding the possibility to make equality or disequality tests at each step of the computation of the automaton. The tests are performed *locally*, between subtrees at a bounded distance from the current computation position in the input tree. The emptiness problem, whether the language recognized by a given automaton is empty, is undecidable with such tests [Mon81]. A decidable subclass is obtained by restricting the tests to sibling subtrees [BT92] (see [CLDG$^+$07] for a survey).

Another approach to allow such tests is to add an auxiliary memory containing a tree and permit memory comparison [CLC05]. Pushdown tree automata [Gue83, CR07] also permit such tests. However, they are also all limited to local tests, at a bounded distance from the current position.

A new approach was proposed more recently in [FTT07, FTT08] with the definition of tree automata with *global* equality and disequality tests (TAGED). The TAGED do not perform the tests during the computation steps but globally on the tree, at the end of the computation, at positions which are defined by the states reached during the computation. For instance, they can express that all the subtrees that reached a given state q are equal, or that every two subtrees that reached respectively the states q and q' are different. The emptiness has been shown decidable for several subclasses of TAGED [FTT07, FTT08], but the decidability of emptiness for the whole class remained a challenging open question until this year.

In this thesis, we will mostly work with various formalisms of tree automata with global constraints, as it seems to be one of the most adapted in order to be applied to security issues. We will first show an application of a subclass of TAGED that only do equality tests to

the verification of cryptographic protocols. We will follow a classical approach, which is to decide whether a term can be inferred from an automata language using term rewriting rules. Closure of TA languages under term rewriting has been a well-studied field, especially in communication protocol verification (see *e.g.* [FGT04, GK00]). However, we present a result which is, to our knowledge, one of the first concerned with automata with (dis)equality constraints. The difficulty lies in the preservation of the equality constraints by the successive rewritings.

Then we will focus on the main aim of this thesis, which is to answer the question of decidability of emptiness for TAGED. The emptiness problem of the subclass of TAGED with only disequality constraints (negative TAGED) has been shown decidable by proving the equivalence with a set constraint problem. This problem was solved by Charatonik and Pacholsky in [CP94]. Charatonik then proved (in [Cha99]) the emptiness decidability of emptiness of another class of automata, DAG-automata, using similar techniques. In chapter 4, we modify this proof to get decidability in the TAGED case.

We also prove the emptiness decision problem for a class of tree recognizers more general than TAGED. We define (in chapter 1) a class of tree automata with global constraints (TAGC) which, roughly, corresponds to TAGED extended with the possibility to express disequalities between subtrees that reached the same state (specifying key constraints, which are not expressible with TAGEDs), and with arbitrary Boolean combinations (including negation) of constraints. We show in Chapter 5 that emptiness is decidable for TAGC.

The decision algorithm uses an involved pumping argument: every sufficiently large tree recognized by the given TAGC can be reduced by an operation of parallel pumping into a smaller tree which is still recognized. The existence of the bound is based on a particular well quasi-ordering.

This thesis is organized as follow:

In chapter 1 we give the formal definitions that we will need along this manuscript. We will mostly introduce the notions of terms and of automata, give the usual notations, and present some basic properties on them. We will also introduce term rewriting systems (TRS) that we will need in chapter 3 and directed acyclic graph representation of terms, which we will intensively use in chapter 4. Finally we give the definition of well quasi-orders that we will need in chapter 5.

In chapter 2 we define the general class of Tree Automata with Global Constraints. We then prove some easy decision results and some closure properties. Then, we will do a quite exhaustive comparative study with most of the automata classes that allow to do similar (dis)equalities test. For each of the presented class, we will give an example of a term that is recognized by TAGC but not by the class, and, when it exists, of a term that is recognized by the class but not by the TAGC. We will mostly

prove those propositions, but some of them that required an involved combinatorial argument will just be stated or presented as conjectures.

In chapter 3 we will focus on rigid tree automata (RTA) which are in fact an easy representation of TAGED that can only do equality tests. We justify the use of this representation in the context of protocol verification. We proved some results specific to RTA. Then, we prove the decidability of the emptiness problem of the rewriting closure of RTA languages, and give an application to security protocol verification.

In chapter 4 we first prove a result of equivalence of expressiveness between negative TAGED and t-dag automata. Then, we enhance a former proof of Charatonik of the emptiness problem of t-dag automata in order to apply it to the full class of TAGED.

Finally, **in chapter 5**, we prove the decidability of the emptiness problem for the whole class of TAGC, which generalizes the class of TAGED. In particular, they allow to handle key constraints, which is not possible with TAGED. We first prove that we can reduce to positive conjunctive constraints by using some arithmetic constraints. Then, we prove the decidability result on TAGC with those positive conjunctive constraints, using a pumping argument and a well quasi-order to prove the existence of a pumping for every big enough term. We then give an extension of this technique for automata that have (dis)equality constraints both global and between brother (as in [BT92]. Finally, we apply these results to prove the decidability of a strict extension of MSO on trees.

Contents

Chapter 1

Preliminaries

1 Ranked Terms

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. Function symbols of arity 0 are also called constant symbols. Given an infinite set $\mathcal{X}$ of variables, the set of terms built over Σ and $\mathcal{X}$ is denoted $\mathcal{T}(\Sigma, \mathcal{X})$, and the subset of ground terms (terms without variables) is denoted $\mathcal{T}(\Sigma)$. The set of variables occurring in a term $t \in \mathcal{T}(\Sigma, \mathcal{X})$ is denoted $vars(t)$. A term $t \in \mathcal{T}(\Sigma, \mathcal{X})$ is called *linear* if every variable of $vars(t)$ occurs at most once in t.

A *substitution* σ is a mapping from a finite subset of $\mathcal{X}$ into $\mathcal{T}(\Sigma, \mathcal{X})$. The application of a substitution σ to a term t is the homomorphic extension of σ to $\mathcal{T}(\Sigma, \mathcal{X})$.

The set of positions of a term t is a set of sequences of positive integers, $\mathcal{P}os(t) \subseteq \mathbb{N}^*$, defined inductively as follow

- if $t = a$ where $a \in \Sigma_0$ or if $t = x$ where $x \in \mathcal{X}$ then $\mathcal{P}os(t) = \{\varepsilon\}$
- if $t = f(t_1, \ldots, t_n)$ where $f \in \Sigma_n$ $\mathcal{P}os(t) = \{\varepsilon\} \cup \{1.p \mid p \in \mathcal{P}os(t_1)\} \cup \cdots \cup \{n.p \mid p \in \mathcal{P}os(t_n)\}$

We denote for all $i, 0 \leq i < |p|$, we denote $p[i]$ the i-th number of p. Positions are compared *wrt* the prefix ordering: $p_1 < p_2$ iff there exists $p \neq \varepsilon$ such that $p_2 = p_1 \cdot p$ (where $p_1 \cdot p$ denotes the concatenation of p_1 and p). In this case, p is denoted $p_2 - p_1$. Two positions p_1, p_2 incomparable with respect to the prefix ordering are called *parallel*, and it is denoted by $p_1 \parallel p_2$. Note that the set of positions is prefix-closed.

A term t can then be seen as a function from its set of *positions* $\mathcal{P}os(t)$ into function symbols or variables of $\Sigma \cup \mathcal{X}$. More precisely, for every position $p \in \mathcal{P}os(t)$ we define $t(p)$ as

- if $p = \varepsilon$ then
  - if $t = a$ with $a \in \Sigma_0$ then $t(p) = a$
  - if $t = x$ with $x \in \mathcal{X}$ then $t(p) = x$

  - if $t = f(t_1, \ldots, t_n)$ with $f \in \Sigma_n$ then $t(p) = n$

- if $p = i.p'$ and $t = f(t_1, \ldots, t_n)$ then $t(p) = t_i(p')$

The subterm of t at position p is denoted $t|_p$. More formally we define its set of positions of $t|_p$ is $\mathcal{P}os(t|_p) = \{p' \mid p.p' \in \mathcal{P}os(t)\}$ and for all $p' \in \mathcal{P}os(t|_p)$ we have $t|_p(p') = t(p.p')$. The replacement in t of the subterm at position p by u is denoted $t[u]_p$. More formally $\mathcal{P}os(t[u]_p) = \{p' \mid p \in \mathcal{P}os(t), p \not< p'\} \cup \{p' \mid p' = p.p'', p'' \in \mathcal{P}os(u)\}$ and for all $p' \in \mathcal{P}os(t[u]_p), t[u]_p(p') = t(p')$ if $p \not< p'$ and $t[u]_p(p') = u(p'')$ if $p' = p.p''$. 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 $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$.

2 Unranked Ordered Labeled Trees

Until now, we have only considered terms over signatures (or ranked alphabet). Here we will give a definition of trees over an unranked alphabet where each position may have an unbounded (but finite) number of successors.

Definition 1.1 *Let $\mathcal{U}$ be an unranked alphabet, and $\mathcal{F}$ a ranked alphabet. Let us define $\Sigma = \mathcal{U} \cup \mathcal{F}$. An* unranked ordered labeled tree *over Σ is a partial function $t : \mathcal{N}^* \to \Sigma$ with domain written $\mathcal{P}os(t)$ and such that*

1. *$\mathcal{P}os(t)$ is finite, nonempty and closed by prefix;*
2. *$\forall p \in \mathcal{P}os(t)$,*
   - *if $p \in \mathcal{F}_n$ then $\{i \mid p.i \in \mathcal{P}os(r)\} = \{1, 2, \ldots, n\}$;*
   - *if $p \in \mathcal{U}$ then $\{i \mid p.i \in \mathcal{P}os(r)\} = \{1, 2, \ldots, k\}$ for some $k > 0$.*

The set of all unranked ordered labeled tree over Σ is denoted $\mathcal{T}(\Sigma)$. A sequence of unranked trees over Σ, even the empty sequence ε, is called a hedge.

3 Term Rewriting

A *term rewrite system* (TRS) over a signature Σ is a finite set of rewrite rules $\ell \to r$, where $\ell \in \mathcal{T}(\Sigma, \mathcal{X})$ (it is called the left-hand side (*lhs*) of the rule) and $r \in \mathcal{T}(\Sigma, vars(\ell))$ (it is called right-hand-side (*rhs*)). A term $t \in \mathcal{T}(\Sigma, \mathcal{X})$ rewrites to $s \in \mathcal{T}(\Sigma, \mathcal{X})$ by a TRS $\mathcal{R}$ (denoted $t \xrightarrow[\mathcal{R}]{} s$) if there is a rewrite rule $\ell \to r \in \mathcal{R}$, a position $p \in \mathcal{P}os(t)$ and a substitution σ such that $t|_p = \sigma(\ell)$ and $s = t[\sigma(r)]_p$. In this case,

t is called *reducible*. An irreducible term is also called an $\mathcal{R}$-*normal-form*. The transitive and reflexive closure of $\xrightarrow[\mathcal{R}]{}$ is denoted $\xrightarrow[\mathcal{R}]{*}$. Given $\mathcal{L} \subseteq \mathcal{T}(\Sigma, \mathcal{X})$, we denote $R^*(L) = \{t \mid \exists s \in L, s \xrightarrow[R]{*} t\}$.

A TRS $\mathcal{R}$ is called *left-linear* (resp. *right-linear*)) if every variable occurs at most once in the left-hand side term (resp. right-hand side term) of the rule. It is called *linear* if it is both left-linear and right-linear. A TRS is called *collapsing* if every right-hand side term of rules of $\mathcal{R}$ is a variable.

4 DAG Representation of Terms

In Chapter 4, we will extensively use a canonical and succinct graph representation of ground terms. Intuitively, we will represent a ground term with a directed acyclic graph, such that each subterm corresponds to exactly one node in the graph, even if it may occurs several times in the original ground term. So we have to introduce several notations on graph.

Definition 1.2 *A directed ordered graph is a couple* $G = (V_G, succ_G)$ *where*

- V_G *is a finite set of nodes*
- *for all* $v \in V_G$ *there is an arity* $ar(v) \in \mathbb{N}$
- $succ_G : \mathbb{N} \times V_G \to V_G$ *is a partial function such that for all node* v, $succ_G(i, v)$ *is defined if and only if* $i \leq ar(v)$

If $succ_G(i, v) = v'$ we say that v' is *the i-th immediate successor* (or simply *an immediate successor*) of v. If all the nodes of a directed ordered graph G are of arity at most 1, we may forget the "ordered" qualification, and, if it exists, we denote $succ_G(v)$ the only immediate successor of v. We denote $v \to_G v'$ the relation "v' is an immediate successor of v". We denote $\xrightarrow{+}_G$ the transitive closure of $\to_G$ and $\xrightarrow{*}_G$ its reflexive transitive closure. If $v \xrightarrow{+}_G v'$ we say that v' is a successor of v. A subgraph G' of a directed graph G is *closed* if G' contains with every node v all the successors of v in G.

A directed ordered graph G is said *acyclic* if there does not exist a node $v \in V_G$ such that $v \xrightarrow{+} v$. A graph G is *rooted* at some node u if u is the only node such that there is no node $v \in V_G$ such that $v \to u$. We call u the *root* of G. Given a directed ordered graph G and a node $v \in V_G$, we denote $G|_v$ and we call *subgraph of G rooted at v* the directed ordered graph $G|_v = \langle V', succ' \rangle$, where $V' = \{v' \mid v \xrightarrow{*} v'\}$ and $succ'$ is the restriction of $succ_G$ to V'.

A position is a finite sequence of integers, and ε denotes the empty sequence. The set of positions $\mathcal{P}os(G)$ of a directed ordered acyclic graph $G = \langle V_G, succ_G \rangle$ rooted at u is defined inductively as follow:

- if $V_G = \{u\}$ then $\mathcal{P}os(G) = \varepsilon$
- otherwise, let $ar(u) = n$, then $\mathcal{P}os(G) = \{\varepsilon\} \cup \{1.p \mid p \in \mathcal{P}os(G|_{succ_G(1,u)})\} \cup \cdots \cup \{n.p \mid p \in \mathcal{P}os(G|_{succ_G(n,u)})\}$

Given a directed ordered graph G rooted at u, and $p \in \mathcal{P}os(G)$, we denote $G(p)$ the node:

- u if $p = \varepsilon$
- $G|_{succ(i,u)}(p')$ if $p = i.p'$

We use $G|_p$ as a short notation for $G|_{G(p)}$.

Given a directed ordered acyclic graph a *leaf* is a node without immediate successors, The *main path* of a given node v in G is the longest path leading from v to a leaf in G; if there are several paths of the same (biggest) length, then the leftmost of them is the main one. The length of the main path for a given node v is called the *height* of the node and is denoted $h(v)$. The height of a directed acyclic graph G, denoted $h(G)$, is the maximal height of its nodes.

In a directed ordered acyclic graph, the *main successor* of a node v is the immediate successor of v lying on the main path for v. The position of the main successor (that is, the number identifying this node in the ordered sequence of successors of v) is called the *main position*. Note the "leftmost" requirement above; it implies that if $v_1, \ldots, v_n$ are the immediate successors of v in G, and v_i lies on the main path for v, then the main paths for $v_1, \ldots, v_{i-1}$ are strictly shorter than the path for v_i. A position (resp. a successor) which is not the main position (resp. the main successor) is called *secondary*.

We say that a node v lies below a node v_0 if the main path for v is shorter than the main path for v_0 (that is, the depth of v is smaller than the depth of v_0).

A directed ordered *labeled* graph is a triplet $G = \langle V_G, succ_G, \lambda_G \rangle$ where $\langle V_G, succ_G \rangle$ is a directed ordered graph, and λ_G is mapping from V_G to another set. We will always precise this set when defining a new labeled graph.

Definition 1.3 *A DAG representation of a ground term (t-dag in short) over a signature Σ is a directed acyclic ordered labeled graph $G = \langle V_G, succ_G, \lambda_G \rangle$ such that*

- *G is rooted at some node u*
- *$\lambda_G : V_G \to \Sigma$ is a labeling of nodes of G by function symbols of Σ*
- *for each node $v \in V_G$ such that $\lambda(v) = f$, the arity of v in G is equal to the arity of f in Σ*
- *no two subgraphs of G rooted at two different nodes are isomorphic (wrt. structure and labelling)*

The last condition is known as *maximal sharing of structure*. The assumption that a t-dag is ordered (that is, the successors of each node are ordered) is needed to assure that the t-dags representing $f(a, b)$ and $f(b, a)$ are not isomorphic.

We will now formalize how a t-dag represents a ground term.

Definition 1.4 *Let $G = \langle V, succ, \lambda \rangle$ be a t-dag. We define inductively a mapping $term_G : V \to \mathcal{T}(\Sigma)$ the following way*

- *if $\lambda(v)$ is a constant, then $term_G(v) = \lambda(v)$*
- *if $\lambda(v)$ is a function symbol f of arity n, then $term_G(v) = f(term_G(succ(1, v)), \ldots, term_G(succ(n, v)))$*

We denote $term(G)$ the ground term $term_G(u)$ where u is the root of G, and we say that G represents *$term(G)$.*

Given a t-dag G representing a term t, we have that $h(G) = h(u) = h(t)$ where u is the root of G. Due to the maximal sharing property, it is quite obvious that given a ground term t, the t-dag representing t is unique up to isomorphism. Let us give a trivial construction of this t-dag.

Proposition 1.5 *Let t be a ground term on a signature Σ. Let us define the directed oriented labeled graph $\langle V, succ, \lambda \rangle$:*

- *$V = \{t' \mid \exists p \in \mathcal{P}os(t), t|_p = t'\}$*
- *for all $t' \in V$, such that $t' = f(t'_1, \ldots, t'_n)$, for all i, $1 \leq i \leq n, succ(i, t') = t'_i$.*

And let $\lambda : V \to \Sigma$ be defined as follow

- *$\lambda(t') = f$ if f is a function symbol of arity n and $t' = f(t'_1, \ldots, t'_n)$*

Then $\langle V, succ, \lambda \rangle$ is a t-dag representing t, and we denote it $dag(t)$.

proof. It is obvious that $dag(t)$ is a directed ordered labeled graph and that the arity or each node is equals to the arity of its labelling. A subterm t' of t has either no successor, if t' is a leaf of t, or all its successors are subterms of t', hence, are of height strictly lower than t'. So $dag(t)$ is also acyclic.

Let t_1, t_2 be be two different subterms of t, hence, two different nodes of $dag(t)$. Let us show that they are not isomorphic (in order to prove the maximal sharing property). If they are not of same height, then the main path of t_1 in $dag(t)$ is of different length than the main path of t_2 in $dag(t)$, and they are not isomorphic. Assume that they are of same height h. Let us prove by induction on h that they are not isomorphic. If $h(t_1) = h(t_2) = 1$, then t_1 and t_2 are two different leafs of t. Hence, they are two different constant symbols of Σ, which are also their labelling.

Hence, they are not isomorphic *wrt* λ. Assume now, that two different nodes of a same height lower than h are necessarily non-isomorphic. Let $h(t_1) = h(t_2) = h$. If $\lambda(t_1) \neq \lambda(t_2)$, they are not isomorphic. Hence, assume $\lambda(t_1) = \lambda(t_2) = f \in \Sigma_n$. We have that $t_1 = f(t_1^1, \ldots, t_n^1)$ and $t_2 = f(t_1^2, \ldots, t_n^2)$. Since $t_1 \neq t_2$ there exists a position $k, 1 \leq k \leq n$ such that $t_k^1 \neq t_k^2$. So t_k^1 and t_k^2 are two different nodes of height lower than h. Either they are of different height, and then they are clearly non-isomorphic, are they are both of same height lower than h, and then, by induction hypothesis, they are not isomorphic. In both cases, t_1 and t_2 are not isomorphic. □

A term and its t-dag representation coincide in many cases. Especially, they have the same set of positions.

Lemma 1.6 *Let t be a ground term, and $G = dag(t)$. Then $\mathcal{P}os(t) = \mathcal{P}os(G)$, and for all $p \in \mathcal{P}os(t)$, $G|_p$ represents $t|_p$.*

proof. It follows mainly from the remark that given a subterm t' of t, such that $\lambda(t') = f \in \Sigma_n$, t' viewed as a term, has the same number n of direct subterms, than the number of immediate successors he has when viewed as a node. Also, we have that, for all $i, 1 \leq i \leq n$, $succ(i, t') = t.i$. Then, by induction on the height of t, we prove that $\mathcal{P}os(t) = \mathcal{P}os(G)$. And then, by recurrence on the length of p, we have that, for all $p \in \mathcal{P}os(t)$, $G|_p$ represents $t|_p$. □

5 Tree Automata

Following definitions and notation of [CLDG+07], we consider tree automata which compute bottom-up (from leaves to root) on (finite) ground terms in $\mathcal{T}(\Sigma)$. At each stage of computation on a tree t, a tree automaton reads the function symbol f at the current position p in t and updates its current state, according to f and the respective states reached at the positions immediately under p in t.

Definition 1.7 *A* tree automaton *(TA) $\mathcal{A}$ on a signature Σ is a tuple $\langle \Sigma, Q, F, \Delta \rangle$ 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 $\mathcal{A}$, denoted $|\mathcal{A}|$, is the number of symbols in Δ. When given the name $\mathcal{A}$ of a tree automaton, we might sometimes use the notations $Q_{\mathcal{A}}$ or Δ_A to denote respectively the set of states of $\mathcal{A}$ or the set of its transition rules.

Definition 1.8 *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 $t(p) = f \in \Sigma_n$ $(n \geq 0)$, $f\big(r(p.1), \ldots, r(p.n)\big) \to r(p) \in \Delta$. A run r is called* successful *if $r(\varepsilon) \in F$.*

We will sometimes use term-like notation for runs. For instance, a run $\{\varepsilon \mapsto q, 1 \mapsto q_1, 2 \mapsto q_2\}$ will be denoted $q(q_1, q_2)$.

Definition 1.9 *The* language $\mathcal{L}(\mathcal{A}, q)$ *of a TA* $\mathcal{A}$ *in state* q *is the set of ground terms for which there exists a run* r *of* $\mathcal{A}$ *such that* $r(\varepsilon) = q$. *The language* $\mathcal{L}(\mathcal{A})$ *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.*

Definition 1.10 *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 ground term t, there is at most (resp. at least) one run r of $\mathcal{A}$ on t.

Proposition 1.11 *Let* $\mathcal{A}$ *be a tree automaton. There exists a deterministic tree automaton* $\mathcal{B}$ *such that* $\mathcal{L}(A) = \mathcal{L}(B)$.

The idea is to take all the subset of states of A as set of states and B, and all the subsets containing a final state of A as the set of final states of B. Then the transition rules are of the form $f(P_1, \ldots, P_n) \to P$, $P, P_1, \ldots, P_n \subseteq Q$, such that $q \in P$ if and only if there exists n states $q_1 \in P_1, \ldots, q_n \in P_n$ such that the transition rule $f(q_1, \ldots, q_n) \to q$ belongs to Δ. Hence, for each configuration $f \in Sigma_n$, $P_1, \ldots, P_n \subseteq Q$, there exists exactly one $P \subseteq Q$ verifying this, and $\mathcal{B}$ is actually a deterministic tree automaton. Note that the size of B may be exponential in the size of A, and that it is a lower bound. Hence, since decidability results may focus on deterministic tree automata, it is still important for complexity issues to work with non-deterministic ones.

Tree automata have nice closure properties. One can find the proofs of the following statement in [CLDG+07].

Proposition 1.12 *Let* $\mathcal{L}_1$ *and* $\mathcal{L}_2$ *be two tree regular languages. Then the following propositions holds*

1. $\bar{L_1}$ *is regular;*
2. $\mathcal{L}_1 \cup \mathcal{L}_2$ *is regular;*
3. $\mathcal{L}_1 \cap \mathcal{L}_2$ *is regular.*

In the general case, idea is to consider two tree automata A_1 and A_2 recognizing respectively $\mathcal{L}_1$ and $\mathcal{L}_2$ and do simple constructions. The size of the automaton to construct is $\mathcal{O}(2^{|A_1|})$ for the complementation, $\mathcal{O}(|A_1| + |A_2|)$ for the union and $\mathcal{O}(|A_1| \times |A_2|)$ for the intersection. Those sizes and the complexities may vary depending whether you have deterministic tree automata, and whether you want to preserve determinism.

We present now some classical decision problems on tree automata, and we will give their complexity here.

Definition 1.13 *Given a TA* $\mathcal{A} = \langle \Sigma, Q, F, \Delta \rangle$*:*

- *the* emptiness *problem is the problem to decide whether* $\mathcal{L}(\mathcal{A}) = \emptyset$
- *the* universality *problem is the problem to decide whether* $\mathcal{L}(\mathcal{A}) = \mathcal{T}(\Sigma)$
- *the* finiteness *problem is the problem to decide whether* $\mathcal{L}(\mathcal{A})$ *is finite*
-

Given a TA $\mathcal{A} = \langle \Sigma, Q, F, \Delta \rangle$ *and a term* t*, the* membership *problem is the problem to decide whether* $t \in$ ‘$\mathcal{L}(\mathcal{A})$.

The emptiness problem is decidable in linear time, the membership and the finiteness problems are in PTIME, whereas the universality problem is EXPTIME-complete (see *e.g.* [CLDG$^+$07]).

When introducing extensions of TA, we will often refer to those decision problems. They will be defined the same way, with the name of the concerned class of automata instead of TA.

6 Automata on Unranked Ordered Labeled Trees

We defined an automata model for ranked terms. Here we generalize this model to unranked labeled ordered trees (see definition 1.1 below). They are called *hedge automata*, in reference of the denomination of the sequences of unranked ordered labeled trees.

Definition 1.14 *A* Hedge Automaton *is a tuple* $\mathcal{A} = \langle \Sigma, Q, F, \Delta \rangle$ *where* $\Sigma = \mathcal{U} \cup \mathcal{F}$ *where* $\mathcal{U}$ *is an alphabet of unranked symbols, and* $\mathcal{F}$ *an alphabet of ranked symbols,* Q *is a finite set of states,* $F \subseteq Q$ *is a set of finite states, and* Δ *is a set of transition rules of the form* $a(R) \to q$ *where* $R \subseteq Q^*$ *is a regular word language over* Q*, defined by an automaton on word. We call the languages* R *the* horizontal languages.

A run *of a hedge automaton* $\mathcal{A}$ *on an unranked tree* t *is a mapping* $r : \mathcal{P}os(t) \to Q$ *such that for all position* $p \in \mathcal{P}os(t)$ *with* $a = t(p)$ *and* $q = r(p)$ *there is a transition rule* $a(R) \to q$ *such that* $r(p.1) \dots r(p.n) \in R$ *where* n *is the number of successors of* p*. In particular in order to apply a rule at a leaf, the empty word* ε *has to be in the horizontal language of the rule. A run* r *is said* successful *if* $r(\varepsilon) \in F$*.*

An unranked ordered labeled tree t *is* accepted *by* A *if there exists a successful run* r *of* $\mathcal{A}$ *on* t*. The* language $\mathcal{L}(\mathcal{A})$ *of* $\mathcal{A}$ *is the set of all unranked trees accepted by* $\mathcal{A}$*.*

7 Well Quasi-Ordering

In chapter 5, we use a pumping lemma that relies on the existence of two comparable elements in a given infinite sequence. To prove this existential property, we need to use a well quasi-order.

Definition 1.15 *A well quasi-ordering [Gal91]* $\leq$ *on a set* S *is a reflexive and transitive relation such that any infinite sequence of elements* $e_1, e_2, \ldots$ *of* S *contains an increasing pair* $e_i \leq e_j$ *with* $i < j$.

Chapter 2

Tree Automata with Equality and Disequality Tests

In this chapter, we define an extension of tree automata with global equality and disequality constraints. We then do a short survey of the existing classes of tree automata that allow similar tests, and compare their properties, the complexity of their decision procedures, and their expressiveness. We will describe many examples of languages of terms that are, or are not, recognized by a given class of tree automata. To prove that a language is recognized, we will only give the description of the corresponding automata: it will be quite obvious that the recognized languages matches. However, we will formally prove that a language is not recognized by some class of automata when we state it.

1 Tree Automata with Global Constraints

1.1 Definition and First Examples

We propose here a general class of TA with global constraints that we introduced in [BCG$^+$10]. It generalizes the class of TAGED that was firstly introduced in [FTT07].

Definition 2.1 *A* tree automaton with global constraints *(**TAGC**) over a signature Σ is a tuple $\mathcal{A} = \langle Q, \Sigma, F, C, \Delta \rangle$ such that $\langle Q, \Sigma, F, \Delta \rangle$ is a TA, denoted $ta(\mathcal{A})$, and C is a Boolean combination of atomic constraints of the form $q \approx q'$ or $q \not\approx q'$, where $q, q' \in Q$. A TAGC $\mathcal{A}$ is called* positive *if $C_\mathcal{A}$ is a disjunction of conjunctions of atomic constraints. A TAGC $\mathcal{A}$ is called* positive conjunctive *if $C_\mathcal{A}$ is a conjunction of atomic constraints. The subclasses of positive and positive conjunctive TAGC are denoted by **PTAGC** and **PCTAGC**, respectively.*

Definition 2.2 *A* run *r of the TAGC $\mathcal{A}$ on a term t is a run of $ta(\mathcal{A})$ on t such that r satisfies $C_\mathcal{A}$, denoted $r \models C_\mathcal{A}$, where the satisfiability*

of constraints is defined as follows. For atomic constraints, $r \models q \approx q'$ *holds (respectively* $r \models q \not\approx q'$*) if and only if for all different positions* $p, p' \in \mathcal{P}os(t)$ *such that* $r(p) = q$ *and* $r(p') = q'$*,* $t|_p = t|_{p'}$ *holds (respectively* $t|_p \neq t|_{p'}$ *holds). This notion of satisfiability is extended to Boolean combinations as usual. As for TAs, we say that* r *is a run of* $\mathcal{A}$ *on* t*.*

A run of $\mathcal{A}$ *on* $t \in \mathcal{T}(\Sigma)$ *is* successful *if* $r(\varepsilon) \in F_{\mathcal{A}}$*. The language* $\mathcal{L}(\mathcal{A})$ *of* $\mathcal{A}$ *is the set of terms* t *for which there exists a successful run of* $\mathcal{A}$*.*

It is important to note that the semantics of $\neg(q \approx q')$ and $q \not\approx q'$ differ, as well as the semantics of $\neg(q \not\approx q')$ and $q \approx q'$. This is because we have a "for all" quantifier in both definitions. The constraint $\neg(q \approx q')$ is satisfied by a run r on a term t if and only if $\exists p, p' \in Pos(t), r(p) = q, r(p') = q', t|_p \neq t|_{p'}$. The difference with $\not\approx$ is that it has to have a disequality between one occurrence of each state, and not all, but it forces the existence of at least one occurrence of each state.

We introduced TAGC as a generalization of Tree Automata with Global Equality and Disequality constraints (TAGED) which were first introduced in [FTT07]. They can be defined as PCTAGC where the disequality constraints are irreflexive, *i.e.* there is no atomic constraints of the form $q \not\approx q$. Hence, there are already several known results on PCTAGC, and many of them directly generalizes to TAGC. We will however prove most of them, as they are a good way to get familiar with the manipulation of these automata.

We will several times characterize the subclasses of TAGC with only atomic constraints either of type $\approx$ or of type $\not\approx$. Moreover, in chapter 5, we will introduce new atomic constraints. So from now, we will always specify between brackets the atomic constraints that are used in a given automaton (or class of automata). For example, we will use the notation TAGC$[\tau]$, where τ is either $\approx$ or $\not\approx$, to characterize the subclass of TAGC with only atomic constraints of type τ (and same for PTAGC and PCTAGC). And the general class of TAGC introduced in definition 2.1 will now be referred as TAGC$[\approx, \not\approx]$.

We will prove many undecidability and expressivity results only for PCTAGC$[\approx]$. Since they are clearly a subclass of TAGC$[\approx, \not\approx]$, those results can be directly generalized to this class.

We will here some examples for the expressivity of the equality constraints of the TAGC. The class of regular languages is strictly included in the class of PCTAGC$[\approx]$ languages due to the constraints.

Example 2.3 *Let* $\Sigma = \{a : 0, f : 2\}$*. The set* $\{f(t, t) \mid t \in \mathcal{T}(\Sigma)\}$ *is not a regular tree language (this can be shown using a classical* pumping *argument).*

However, it is recognized by the following PCTAGC$[\approx]$ $\mathcal{A} = \langle \{q_0, q_1, q_f\}, \Sigma, \{q_f\}, q_1 \approx q_1, \{a \to q_0 \mid q_1, f(q_0, q_0) \to q_0 \mid q_1, f(q_1, q_1) \to q_f\} \rangle$*, where* $a \to q \mid q_r$ *is an abbreviation for* $a \to q$ *and* $a \to$

q_r. *An example of successful run of* $\mathcal{A}$ *on* $t = f(f(a,a),f(a,a))$ *is* $q_{\mathsf{f}}\big(q_1(q_0,q_0),q_1(q_0,q_0)\big)$. $\diamond$

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

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

Example 2.4 *Let us extend the PCTAGC*$[\approx]$ *of Example 2.3 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 automaton obtained recognizes the set of terms of* $\mathcal{T}(\Sigma)$ *containing the pattern* $f(x,x)$. $\diamond$

The principle of the construction of Examples 2.3 and 2.4 can be generalized into the following result.

Proposition 2.5 *For every term* $t \in \mathcal{T}(\Sigma,\mathcal{X})$, *there exists a PCTAGC*$[\approx]$ *of size linear in the size of* t *and constructed in linear time which recognizes the terms of* $\mathcal{T}(\Sigma)$ *having a ground instance of* t *as a subterm.*

proof. The proposition is obvious when t is a variable. Let us assume that t is not a variable and let us associate one state q_s to every strict subterm s of t (including variables). The PCTAGC[$\approx$] for Proposition 2.5 has for set of states $Q = \{q_s \mid s \text{ strict subterm of } t\} \cup \{q, q_{\mathsf{f}}\}$, its constraint is of the form $C = \bigwedge_{x\in vars(t)} q_x \approx q_x$, (we recall that $vars(t)$ is the set of variables occurring in t), the subset of final states is $F = \{q_{\mathsf{f}}\}$, and its transition set is

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

The transitions in the first four lines ensure the recognition of the pattern t into the final state q_{f} and the transitions in the last line ensure the propagation of q_{f}. The equality constraints $q_x \approx q_x$ ensure that the non linearities in t are respected. $\square$

With disequality constraints, TAGC can recognize languages that are in some sense the duals of the ones it can recognize with equality constraints. For example, by replacing the constraint by $q_1 \not\approx q_1$ in the example 2.3, the TAGC can recognize the language $\{f(t_1,t_2) \mid t_1 \neq t_2\}$. Moreover, reflexive disequality constraints such as $q \not\approx q$ correspond to monadic *key constraints* for XML documents, meaning that every

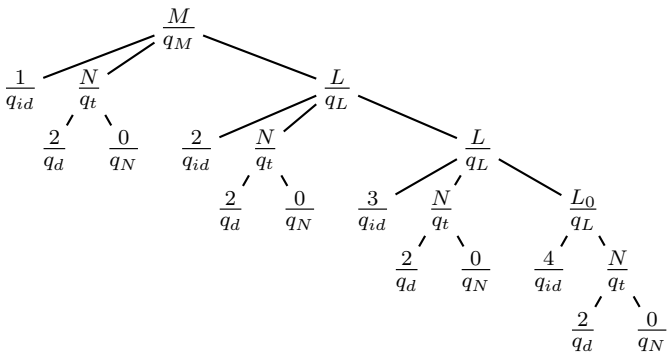

Figure 2.1: Term and successful run (Ex. 2.6).

two distinct positions of type q have different values. A state q of a TAGC$[\approx, \not\approx]$ can be used for instance to characterize unique identifiers as in the following example.

Example 2.6 *The TAGC$[\approx, \not\approx]$ of our example accepts (in state q_M) lists of dishes called menus, where every dish is associated with one identifier (state q_{id}) and the time needed to cook it (state q_t). We have other states accepting digits (q_d), numbers (q_N) and lists of dishes (q_L).*

The TAGC$[\approx, \not\approx]$ $\mathcal{A} = \langle Q, \Sigma, F, C, \Delta\rangle$ is defined as follows: $\Sigma = \{0, \ldots, 9 : 0, N, \mathcal{L}_0 : 2, \mathcal{L}, M : 3\}$, $Q = \{q_d, q_N, q_{id}, q_t, q_L, q_M\}$, $F = \{q_M\}$, and $\Delta = \{i \to q_d \mid q_N \mid q_{id} \mid q_t \mid 0 \leq i \leq 9\} \cup \{N(q_d, q_N) \to q_N \mid q_{id} \mid q_t, \mathcal{L}_0(q_{id}, q_t) \to q_L, \mathcal{L}(q_{id}, q_t, q_L) \to q_L, M(q_{id}, q_t, q_L) \to q_M\}$.
The constraint C ensures that all the identifiers of the dishes in a menu are pairwise distinct (i.e. that q_{id} is a key) and that the time to cook is the same for all dish: $C = q_{id} \not\approx q_{id} \wedge q_t \approx q_t$.
A term in $\mathcal{L}(\mathcal{A})$ together with an associated successful run are depicted in Figure 2.1. ◇

1.2 Decision Problems

We prove here some already known or easy decision problems on TAGC.

Proposition 2.7 *Membership is NP-complete for TAGC$[\approx, \not\approx]$.*

proof. Given a TAGC $\mathcal{A} = \langle Q, \Sigma, F, C, \Delta\rangle$ and a term $t \in \mathcal{T}(\Sigma)$, a non-deterministic algorithm consist in guessing a function r from $\mathcal{P}os(t)$ into Q, and checking that r is a successful run of $\mathcal{A}$ on t. The checking can be performed in polynomial time.

We show NP-hardness for the restricted class of PCTAGC$[\approx]$ as it has already been done in [FTT07] and [JKV09]. We propose a reduction of 3-SAT for a formula ϕ into the membership for a PCTAGC$[\approx]$ and a term t representing ϕ.

Let us consider an instance ϕ of 3-SAT with variables from a set V. It is represented as a term t over the signature $\Sigma = \{0, 1 : 0, \neg : 1 \wedge$

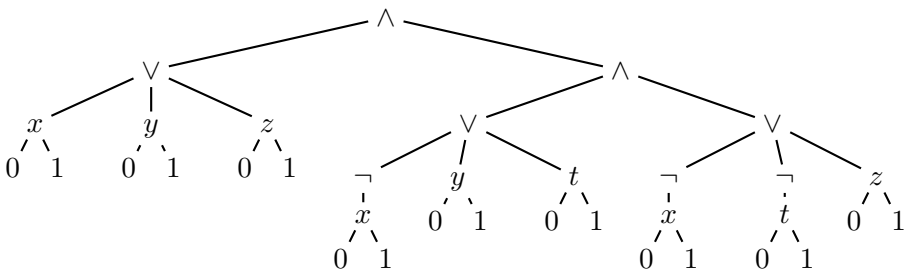

Figure 2.2: Membership NP-hardness: tree encoding of a 3 SAT instance.

$:2, \vee:3\} \cup \{x:2 \mid x \in V\}$. Every variable x is represented by a subterm $x(0,1)$, a 3 literal clause $\ell_1 \vee \ell_2 \vee \ell_3$ is encoded into $\vee(t_1,t_2,t_3)$ where t_1,t_2,t_3 encode respectively ℓ_1,ℓ_2,ℓ_3. Finally we encode a conjunction of disjunctions $D_1 \wedge \cdots \wedge D_n$ into $\wedge(t_1,\ldots,\wedge(t_{n-1},t_n))$ where each t_i, $i \leq n$, is the encoding of D_i.

For instance, the tree encoding of the 3-SAT instance $(x \vee y \vee z) \wedge (\neg x \vee y \vee t) \wedge (\neg y, \neg t, z)$ is depicted in Figure 2.2.

We define an PCTAGC[$\approx$] $\mathcal{A} = \langle \Sigma Q, F, C, \Delta \rangle$ by $Q = \{q_1, q_0\} \cup \{q_x, q_{\neg x} \mid x \in V\}$, $F = \{q_1\}$, $C = \bigwedge_{x \in V}(q_x \approx q_x) \wedge (q_{\neg x} \approx q_{\neg x})$ and

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

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

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

Assume that the given 3-SAT instance has a solution $\sigma : V \to \{0,1\}$ (mapping of propositional variables into truth values). We define a successful run r of $\mathcal{A}$ in t as follows. For each variable $x \in V$ and for each position $p \in \mathcal{P}os(t)$ such that $t|_p = x$, we have by construction of t that $t|_{p.1} = 0$ and $t|_{p.2} = 1$. If $\sigma(x) = 0$, we define $r(p.1) = q_x$ and $r(p.2) = q_{\neg x}$, and if $\sigma(x) = 1$, we define $r(p.1) = q_{\neg x}$ and $r(p.2) = q_x$. Both options are possible thanks to the rules $0 \to q_{(\neg)x}$ and $1 \to q_{(\neg)x}$, and since we do the same thing for all occurrence of x in t, the constraints on q_x and $q_{\neg x}$ in C are satisfied by 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 $t \in \mathcal{L}(\mathcal{A})$.

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 constraint C 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 atomic constraint $q_x \approx q_x$. 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 atomic constraints on 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.

□

We recall that for plain TA, membership is in PTIME.

The *universality* problem is known to be undecidable already for the small subclass of PCTAGC[≈], called positive TAGED in [FTT08] or rigid tree automata in [JKV09].

Proposition 2.8 *[FTT08, JKV09] Universality is undecidable for PCTAGC[≈].*

Given two automata $\mathcal{A}_1$ and $\mathcal{A}_1$ the inclusion (resp. equivalence) problem is the problem of deciding whether $\mathcal{L}(\mathcal{A}_1) \subseteq \mathcal{L}(\mathcal{A}_2)$ (resp. $\mathcal{L}(\mathcal{A}_1) = \mathcal{L}(\mathcal{A}_2)$). The following corollary is easily derived from the proposition 2.8.

Theorem 2.9 *Inclusion and equivalence are undecidable for PCTAGC[≈].*

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

The following original result is another consequence of proposition 2.8.

Proposition 2.10 *It is undecidable whether the language of a given PCTAGC[≈] is regular.*

proof. We show that universality is reducible to regularity. Let us define the quotient of a term language $\mathcal{L}$ by a term s *wrt* a function symbol f: $\mathcal{L}/s := \{t \mid f(s,t) \in \mathcal{L}\}$. This operation preserves regular languages: for all s and f, if $\mathcal{L}$ is regular then $\mathcal{L}/s$ is regular.

Let $\mathcal{A}$ be an input of universality for PTAGC[$\approx$], and let $\mathcal{L}'$ be a language of PTAGC[$\approx$] over Σ which is not regular (such a language exists). Let $\mathcal{L}_1 := f\big(\mathcal{L}(\mathcal{A}), \mathcal{T}(\Sigma)\big) \cup f\big(\mathcal{T}(\Sigma), \mathcal{L}'\big)$ where f is a binary symbol, possibly not in Σ ($f(\mathcal{L}, \mathcal{T}(\Sigma))$ denotes $\{f(s,t) \mid s \in \mathcal{L}, t \in \mathcal{T}(\Sigma)\}$). It is obvious that $\mathcal{L}_1$ is a TAGC language.

If $\mathcal{L} = \mathcal{T}(\Sigma)$, then $\mathcal{L}_1 = f\big(\mathcal{T}(\Sigma), \mathcal{T}(\Sigma)\big)$ and it is regular. Assume that $\mathcal{L}(\mathcal{A}) \neq \mathcal{T}(\Sigma)$ and let $s \in \mathcal{T}(\Sigma) \setminus \mathcal{L}(\mathcal{A})$. By construction, $\mathcal{L}_1/s = \mathcal{L}'$ which is not regular. Hence $\mathcal{L}_1$ is not regular. Therefore $\mathcal{L}(\mathcal{A}) = \mathcal{T}(\Sigma)$ iff the TAGC language $\mathcal{L}_1$ is regular. □

The *emptiness* is the problem to decide, given a TAGC $\mathcal{A}$, whether $\mathcal{L}(\mathcal{A}) = \emptyset$? The proof that it is decidable for TAGC is rather involved and is presented in section 2.

1.3 Closure Properties

Let us conclude this first section with the closure properties of the TAGC languages.

Proposition 2.11 *The class of TAGC languages is closed under union and intersection but not closed effectively under complementation.*

proof. We use a classical disjoint union for union and Cartesian product of state sets for intersection, with a careful redefinition of constraints on this product.

More precisely, let $\mathcal{A}_1 = \langle Q_1, \Sigma_1, F_1, C_1, \Delta_1\rangle$ and $\mathcal{A}_2 = \langle Q_2, \Sigma_2, F_2, C_2, \Delta_2\rangle$ be two TAGCs. We can assume *wlog* that Q_1 and Q_2 are disjoint.

The TAGC $\mathcal{A}_\cup = \langle Q_1 \uplus Q_2, \Sigma_1 \cup \Sigma_2, F_1 \uplus F_2, C_1 \wedge C_2, \Delta_1 \uplus \Delta_2\rangle$ recognizes $\mathcal{L}(\mathcal{A}_1) \cup \mathcal{L}(\mathcal{A}_2)$.

We define a TAGC $\mathcal{A}_\cap = \langle Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, F_1 \times Q_2 \cup Q_1 \times F_2, C_\cap, \Delta_\cap\rangle$ recognizing $\mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$. The constraint $C_\cap$ is obtained from $C_1 \wedge C_2$ by replacing every atom $q_1 \approx q_1'$ with $q_1, q_1' \in Q_1$ (resp. $q_2 \approx q_2'$ with $q_2, q_2' \in Q_2$) by $\bigwedge_{q_2, q_2' \in Q_2} \langle q_1, q_2\rangle \approx \langle q_1', q_2'\rangle$ (resp. $\bigwedge_{q_1, q_1' \in Q_1} \langle q_1, q_2\rangle \approx \langle q_1', q_2'\rangle$), and similarly for the atoms $q_1 \not\approx q_1'$, $q_2 \not\approx q_2'$. The set of transitions is $\Delta_\cap = \big\{ f\big(\langle q_{1,1}, q_{2,1}\rangle, \ldots, \langle q_{1,n}, q_{2,n}\rangle\big) \to \langle q_1, q_2\rangle \;\big|\; f(q_{i,1}, \ldots, q_{i,n}) \to q_i \in \Delta_i \text{ for } i = 1, 2\big\}$.
The effective closure under complementation of TAGC would contradict Proposition 2.8. □

Note that the two above constructions transform two PTAGC (resp. two PCTAGC) into a PTAGC (resp. a PCTAGC). Hence the subclasses of PTAGC and PCTAGC are also closed under union and intersection. Using a disjunctive normal form for the constraint of a PTAGC, it is easy to show that both classes of PTAGC and PCTAGC are equally expressive.

Corollary 2.12 *Let $\mathcal{B} = \langle Q, \Sigma, F, C, \Delta\rangle$ be a PTAGC. Then one can effectively constructs a PCTAGC $\mathcal{A}$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$.*

proof. Let $\mathcal{B} = \langle Q, \Sigma, F, C, \Delta \rangle$, and let $C' = C_1 \vee \ldots \vee C_k$ where $C_1, \ldots, C_k$ are conjunctions of atomic constraints of the form $q \approx q'$ or $q \not\approx q'$ for $q, q' \in Q$, be the disjunctive normal form of the constraint C. It is obvious that, given the PTAGC $B = \langle Q, \Sigma, F, C', \Delta \rangle$, $\mathcal{L}(\mathcal{B}) = \mathcal{L}(\mathcal{B}')$ For all $1 \leq i \leq k$, let $\mathcal{B}_i = \langle Q, \Sigma, F, C_i, \Delta \rangle$. Every such $\mathcal{B}_i$ is a PCTAGC. Let $\mathcal{L}(\mathcal{A})$ be the union construction of the proof of proposition 2.11, modulo state renaming, for $\bigcup_{i=1}^{k} \mathcal{L}(\mathcal{B}_i)$. We have that $\mathcal{L}(\mathcal{A}) = \bigcup_{i=1}^{k} \mathcal{L}(\mathcal{B}_i) = \mathcal{L}(\mathcal{B}') = \mathcal{L}(\mathcal{B})$. Since the construction in the proof of Proposition 2.11 preserves the positive conjunctive property of the constraints, $\mathcal{A}$ is a PCTAGC recognizing the same language as $\mathcal{B}$. □

2 Related Models with (Dis)Equalities Tests

The formalism of TAGC allows one to test (dis)equalities globally, and to have a succinct representation thanks to the grammar of the constraints. It is a quite natural extension of TAGED that we will present in subsection 2.1. But other models of tree automata already existed that allow to do similar tests, either directly, like the tree automata with local constraints, or by considering specific applications of their features, like tree automata with one memory, or dag automata. We here show the interest of TAGC by doing a short survey of those models. For each of them, we show how it compares to TAGC in expressiveness.

2.1 TAGED

Definitions and Known Results

Tree Automata with General Equality and Disequality constraints [FTT08] were introduced in the context of spatial logics for XML querying [FTT07].

Definition 2.13 *A* Tree Automaton with General Equality and Disequality constraints *(TAGED for short) is a tuple* $\mathcal{A} = \langle \Sigma, Q, F, \Delta, \approx_{\mathcal{A}}, \not\approx_{\mathcal{A}} \rangle$ *where*

- $\langle \Sigma, Q, F, \Delta \rangle$ *is a tree automaton;*
- $\approx_{\mathcal{A}}$ *is a reflexive and symmetric binary relation on a subset of* Q*;*
- $\not\approx$ *is an irreflexive and symmetric binary relation on* Q*.*

A TAGED $\mathcal{A}$ *is said to be* positive *(resp.* negative*) if* $\not\approx_{\mathcal{A}}$ *(resp.* $\approx_{\mathcal{A}}$*) is empty.*

A run r of a TAGED on a term $t \in \mathcal{T}(\Sigma)$ 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 $r(p_1) \approx_{\mathcal{A}} r(p_2)$ then $t|_{p_1} = t|_{p_2}$ and if $r(p_1) \not\approx_{\mathcal{A}} r(p_2)$ then $t|_{p_1} \neq t|_{p_2}$.

It is quite obvious that TAGED are equivalent to PCTAGC$[\approx, \not\approx]$ where $\not\approx$ is irreflexive. In fact, we were working on TAGED when we

introduced TAGCto generalize our results. Several results on these automata have been shown in [FTT08]. The emptiness problem is EXPTIME-complete for positive TAGED and in NEXPTIME for negative TAGED. Moreover, the authors of [FTT08] prove decidability for the emptiness problem of a subclass combining equality and disequality tests.

Definition 2.14 *Let Σ be a ranked alphabet. A vertically bounded TAGED (vbTAGED) over Σ is a pair (A, k) where A is a TAGED over Σ and $k \in \mathbb{N}$. A run r of k on a tree $t \in \mathcal{T}(\Sigma)$ is a run of A on t. It is an accepting run if r is accepting for A and the number of states from $dom(\not\approx_{\mathcal{A}})$ occurring along any root-to-leaf path is bounded by k:*

for all position p such that $t|p$ is a leaf, $|\{i \mid i < p \wedge r(i) \in dom(\not\approx_{\mathcal{A}})\}| \leq k$

The authors of [FTT08] show that the emptiness problem for vbTAGED is decidable in 2NEXPTIME. However they did not solve the emptiness problem for the full class.

One interesting property of TAGED is that the equality relation can be reduced to an identity relation.

Theorem 2.15 *[FTT08] Every TAGED A is equivalent to a TAGED A' (whose size might be exponential in the size of A) such that $\approx_{A'} \subseteq id_{Q_{A'}}$ where $id_{Q_{a'}}$ is the identity relation on $Q_{A'}$. Moreover, A' can be built in exponential time.*

Such a property allows one to manipulate TAGED more easily. Also, some results have a lower complexity when using such a TAGED. For example, the emptiness of a positive TAGED $\mathcal{A}$ such that $\approx_{\mathcal{A}} \subseteq id_{Q_{\mathcal{A}}}$ can be done in linear time. In chapter 3 we are concerned with Rigid Tree Automata. Those are actually positive TAGED respecting this property.

Comparison with TAGC

The link with PCTAGC is quite obvious.

Proposition 2.16 *The class of TAGED is equivalent to the class of PCTAGC$[\approx, \not\approx]$ where $\not\approx$ is irreflexive*

The requirement of irreflexivity weakens the expressive power of PCTAGC$[\approx, \not\approx]$. For example, we have seen that with a reflexive disequality constraint, one can express key constraints on XML documents.

Lemma 2.17 *Let $\Sigma = \{0:0, s:1, f:2\}$. The set $\mathcal{L}$ of terms of $\mathcal{T}(\Sigma)$ of the form $f(s^{n_1}(0), \ldots, f(s^{n_k}(0), 0))$, such that $k \geq 0$ and the integers n_i, for $i \leq k$, are pairwise distinct, is recognized by a TAGC$[\approx, \not\approx]$ with an irreflexive disequality relation, but cannot be recognized by a TAGED.*

proof. The following PCTAGC[$\approx, \not\approx$] recognizes $\mathcal{L}$.

$$\left\langle \{q_0, q, q_\mathsf{f}\}, \Sigma, \{q_\mathsf{f}\}, q \not\approx q, \left\{ \begin{array}{l} 0 \to q_0 \mid q \mid q_\mathsf{f}, \\ s(q_0) \to q_0 \mid q, \\ f(q, q_\mathsf{f}) \to q_\mathsf{f} \end{array} \right\} \right\rangle.$$

Assume that there exists a PCTAGC[$\approx, \not\approx$] like in proposition 2.16 $\mathcal{A}$ recognizing this language $\mathcal{L}$.

There exists an accepting run r of $\mathcal{A}$ on the term $t = f(s(0), f(s^2(0), \ldots f(s^{|Q|+1}(0))))$. We have therefore $r \models C_\mathcal{A}$ (the global constraint of $\mathcal{A}$, which is positive by hypothesis).

There are two different positions $p_i = 2^{i-1}.1$ and $p_j = 2^{j-1}.1$, $1 \leq i < j \leq |Q| + 1$ such that $r(p_i) = r(p_j)$. Let us show that $r' = r[r|_{p_i}]_{p_j}$ is an accepting run of $\mathcal{A}$ on $t' = t[t|_{p_i}]_{p_j}$. Since $r(p_i) = r(p_j)$ and r is a run of $\mathcal{A}$ on t, by replacing $t|_{p_j}$ with $t|_{p_i}$ and $r|_{p_j}$ with $r|_{p_i}$, r' is a run of $ta(\mathcal{A})$ on t'. Hence we only have to ensure that the constraint $C_\mathcal{A}$ is fulfilled by r'.

For all $p, p' \in \mathcal{P}os(t')$ such that $2^{j-1}.1$ is neither a prefix of p nor of p', and such that p and p' are no prefix of $2^{j-1}.1$, if $r'(p) \approx r'(p')$ is in C_1 (resp. $r'(p) \not\approx r'(p')$ is in C_1), we know that $r \models r(p) \approx r(p')$ (resp. $r \models r(p) \not\approx r(p')$), so the constraints are respected in t, hence also in t' since the positions p and p' are referring to common subterms of t and t'.

If a position $p = 2^{j-1}.1.v$ is involved in some constraint, let $p' = 2^{i-1}.1.v$. By construction, we have $r'|_p = r'|_{p'}$ and $t'|_p = t'|_{p'}$. Due to this last equality, any constraint involving p is satisfied iff the same constraint with p' instead of p also holds. And thanks to the equality $r'|_p = r'|_{p'}$, this other constraint holds and is satisfied by r'.

Finally, we have to consider constraints that involve a strict prefix p of $2^{j-1}.1$. It is clear that every subterm of t at a position 2^ℓ, for $0 \leq \ell \leq |Q|$, is unique, so every subterm at such a position can only satisfy a disequality constraint or an equality with itself (in that case $r(2^\ell)$ is unique in r). In the latter case, $r'(p)$ is also unique in r' and the equality is obviously satisfied. In the other case, it is easy to see that it also holds in t' that all subterms at positions 2^ℓ, and hence the subterm at position p, are unique, and satisfy all the disequalities. So t' is recognized by $\mathcal{A}$ but is not in the language $\mathcal{L}$, a contradiction. □

2.2 Tree Automata with Local Constraints

Definition and Known Results

A TA with local equality and disequality constraints is a TA whose transitions can perform local equality and disequality tests on the subterms of the term in input (see *e.g.* [BT92, DCC95]).

Definition 2.18 *A* tree automaton with local equality and disequality constraints *(TAC for short) is a tuple* $A = \langle \Sigma, Q, F, \Delta \rangle$ *where*

- Σ *is a signature;*
- Q *is a finite set of states;*
- $F \subseteq Q$ *is a set of final states;*
- Δ *is a set of transitions of the form* $f(q_1, \ldots, q_n) \xrightarrow{c} q$ *where* $f \in \Sigma_n$, $q_1, \ldots, q_n, q \in Q$, *and* c *is a conjunction of constraints of the form* $\pi = \pi'$ *or* $\pi \neq \pi'$ *where* π *and* π' *are positions (sequences of positive integers).*

A run of a TAC on a term t *is a function* $r : \mathcal{P}os(t) \to Q$ *such that for all* $p \in \mathcal{P}os(t)$ *with* $t(p) = f \in \Sigma_n$ $(n \geq 0)$, *there exists a transition* $f\big(r(p \cdot 1), \ldots, r(p \cdot n)\big) \xrightarrow{c} r(p) \in \Delta$ *such that for all constraints* $\pi = \pi'$ *(resp.* $\pi \neq \pi'$*) in* c, *we have* $p \cdot \pi, p \cdot \pi' \in \mathcal{P}os(t)$ *and* $t|_{p \cdot \pi} = t|_{p \cdot \pi'}$ *(resp.* $t|_{p \cdot \pi} \neq t|_{p \cdot \pi'}$*).*

A TAC is called positive *(resp* negative*) if all its transitions contain only equalities (resp. disequalities).*

Note that the RTA language of Examples 2.3 and 2.4 are recognizable by positives TAC:

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

for Example 2.3, and the same extended with the transitions $f(q, q_{\mathsf{f}}) \to q_{\mathsf{f}}$, $f(q_{\mathsf{f}}, q) \to q_{\mathsf{f}}$ for Example 2.4.

The emptiness problem is undecidable in general [Mon81] for positive TAC. Two decidable subclasses of TAC have been identified: tree automata with equality and disequality tests between brother positions [BT92] (TACB), where we have the constraint that for all constraint $\pi = \pi'$ or $\pi \neq \pi'$, $|\pi| = |\pi'| = 1$, and Reduction Automata [DCC95] (RA), that we will note define here; the complexity of emptiness is at least EXPTIME for these subclasses.

Comparison with TAGC

We can do an interesting comparison if we limit ourselves to positive conjunctive TAGC The tests of TAC are performed locally, on each application of a transition rules with constraints, while the ones of PC-TAGC are performed globally, on all positions labeled by a state of the domains of $\approx$ and $\not\approx$. Those different approaches lead to two incomparable expressiveness.

There is a good intuition to understand the respective power of those two classes. A TAC can test an unbounded number of "classes" of (dis)equalities (one for each application of a transition rule), but each "class" has a bounded number of element (the size of the constraints). Conversely, a PCTAGC $A = \langle \Sigma, Q, F, \Delta, C \rangle$ can test only a bounded number of "classes" of (dis)equalities (the size of the constraint C), but each of these "classes" may test an unbounded number of (dis)equalities (one for each occurrence of a state occurring in C).

Following this intuition it is easy to construct a term language that is recognized by one of these models but not by the other. For the sake of simplicity, we will restrict ourselves to positive TAC with test between brothers one one hand, and to PCTAGC[$\approx$] on the other hand. A good candidate for a language recognized by a TAC but not by a PCTAGC[$\approx$] would need to test equalities between two subterms an unbounded number of times.

Example 2.19 *Let* $\Sigma = \{f : 2, a : 0\}$ *and let* $\mathcal{L} = \{t \mid t \in \mathcal{T}(\Sigma), \forall p \in \mathcal{P}ost, t(p) = f \Rightarrow t|_{p.1} = t|_{p.2}\}$. *The language* $\mathcal{L}$ *is recognized by the positive TACB* $\mathcal{A} = \langle \Sigma, Q, F, \Delta \rangle$ *where*

- $Q = F = \{q\}$
- $\Delta = \{a \to q, f(q, q) \xrightarrow{1=2} q\}$

Proposition 2.20 *The language of the example 2.19 is not recognizable by a PCTAGC*[$\approx$].

proof. Let $\mathcal{B} = \langle \Sigma, Q, F, \Delta, C \rangle$ be a PCTAGC[$\approx$] such that for all $t \in \mathcal{L}$, $\mathcal{A}$ recognizes t. We will detail the construction of a term t' recognized by $\mathcal{B}$ which is not in $\mathcal{L}$. However, we will only give a hint why it is recognized by $\mathcal{B}$. Let N be the number of states of Q occurring in C. It is quite obvious that for all $i \in \mathbb{N}^*$ there is a unique term of height $|i|$ in $\mathcal{L}$ that we can define inductively as follow: $t_1 = 0$ and $t_i = f(t_{i-1}, t_{i-1})$ for $i > 1$. Let $t = t_{N(|Q|+2)}$, and let r be a run of $\mathcal{B}$ accepting t. Let $p = 1^{N(|Q|+2)-1}$, p is a position of the leftmost leaf of t. Let M be the number of prefixes p' of p such that $r(p')$ is involved in an equality test *wrt* r and C, that is, there exists $p'' \in \mathcal{P}os(t), p'' \neq p'$ such that the atomic constraint $r(p') \approx r(p'')$ occurs in C. Obviously $M \leq N$. There are at most N occurrences of states involved in an equality test in the branch from the root to the leaf at position p. And since $|p| = N(|Q|+2)$, by the pigeon hole principle, there exists two positions, p_1 and p_2 on the branch, such that $p_1 < p_2$, $|p_2 - p_1| \geq |Q|$, and for all $p', p_1 \leq p' \leq p_2$, $r(p')$ is not involved in an equality test *wrt* r and C. So there exists two positions p'_1 and p'_2, $p_1 \leq p'_1 < p'_2 \leq p_2$ such that $r(p'_1) = r(p'_2)$.

Let $\bar{p_1} < \cdots < \bar{p_k}$ be all the strict prefixes of p_1 such that for all $i, 1 \leq i \leq k, r(\bar{p_i})$ is a state involved in an equality test *wrt* r and C. If $k > 0$ Let us define inductively $\bar{t_k} = t|_{\bar{p_k}}[t|_{p'_2}]_{p'_1 - \bar{p_k}}$, $\bar{r_k} = r|_{\bar{p_k}}[r|_{p'_2}]_{p'_1 - \bar{p_k}}$ and $\bar{t_i}$ and $\bar{r_i}$ the term $t|_{\bar{p_i}}$ and the run $t|_{\bar{p_i}}$ where for each position $\bar{p}$ such that $r(\bar{p_i}.\bar{p}) = r(\bar{p_{i+1}})$ or $r(\bar{p_i}.\bar{p}) = r(\bar{p_{i+1}})$, we replace $t|_{\bar{p_i}.\bar{p}})$ by $\bar{t_{i+1}}$ and $r|_{\bar{p_i}.\bar{p}})$ by $\bar{r_{i+1}}$. We define t' and r' as t and r where for each position $\bar{p}$ such that $r(\bar{p}) = r(\bar{p_1})$ or $r(\bar{p}) \approx r(\bar{p_1})$, we replace $t|_{\bar{p}}$ by $\bar{t_1}$ and $r|_{\bar{p}})$ by $\bar{r_1}$.

If $k = 0$, that is there is no prefix p' of p_1 such that the state occurring at position p' in r is involved in an equality test, then we just define t' and r' as $t' = t[t'|_{p_2}]_{p'_1}$ and $r' = r[r'|_{p_2}]_{p'_1}$.

This pumping preserves the compatibility with Δ, since the initial pumping is done between two positions of occurrences of a same state in r: p'_1 and p'_2. This initial pumping does not change equality tests done with states that are at positions p_2. The equality tests done with states that are occurring above p_1 are ensured to be answered positively by the parallel pumping done at each position where a state involved in such an equality test occurs. Equality tests done only between positions parallel to p_1 are not changed. Proving these statements need an exhaustive case study that we do not do here. However, a very similar approach is used and developed later for the proof of lemma 5.20 in chapter 5. Hence, r' satisfies C and is a successful run of $\mathcal{B}$ on t'.

If $k > 0$, in the subterm $t_{\bar{k}}$, it is clear that the term at position $(p'_1 - p_{\bar{k}}) = 1^{|p'_1|-|p_{\bar{k}}|}$ is different from the term at position $(1)^{|p'_1|-|p_{\bar{k}}|-1}.2$, because the first one is a pumping of the second one. Hence at each position p' such that $t'(p') = t_{\bar{k}}$ we have that the constraint of the definition of $\mathcal{L}$ that states that all brothers are equals is not respected at position $p'.1^{|p'_1|-|p_{\bar{k}}|-1}$. If $k = 0$, the brother condition is not respected at the position $1^{|p'_1|-1}$. Hence t' does not belong to $\mathcal{L}$. □

Conversely, a good candidate for a language recognized by a PCTAGC and not recognized by a positive TACB would test a single equality between an unbounded number of subterms. However, the counter-example will not need to make the equality to occur an unbounded number of time, but only to occur at two positions not reachable at the same time by a transition rules testing equality. It is quite easy with TACB since transition rules can only test equalities between brothers, but would need further work to be applied to positive TAC.

Example 2.21 *Let* $\Sigma = \{f : 2, g : 1, a : 0\}$ *and let* $\mathcal{L} = \{t \mid t \in \mathcal{T}(\Sigma), \forall p_1, p_2 \in \mathcal{P}os(t), t(p_1) = g \wedge t(p_2) = g \Rightarrow t|_{p_1} = t|_{p_2}$. *The language* $\mathcal{L}$ *is recognized by the PCTAGC*$[\approx]$ $\mathcal{A} = \langle \Sigma, Q, F, \Delta, C \rangle$ *where*

- $F = Q = \{q_1, q_2\}$
- $\Delta = \{a \to q_1, f(q_1, q_1) \to q_1, f(q_1, q_2) \to q_1, f(q_2, q_1) \to q_1, f(q_2, q_2) \to q_1, g(q_1) \to q_2\}$
- $C = q_2 \approx q_2$

Proposition 2.22 *The language of the example 2.21 is not recognizable by a positive TACB.*

proof. Let $\mathcal{B} = \langle \Sigma, Q, F, \Delta \rangle$ be a TACB such that for all $t \in \mathcal{L}$, $\mathcal{B}$ recognizes t. Let us define t_i recursively as follow: $t_0 = f(0, 0)$ and $t_i = f(0, t_{i-1})$. Let $t = f(g(t_{|Q|+1}), f(0, g(t_{|Q|+1})))$, and let r be a run of $\mathcal{B}$ accepting t. It is clear that only two transition rules testing equalities may have been used in r: one at each occurrence of t_0. Let $p = 1.1$. For all i, $0 \leq i \leq |Q|$, let us denote $p_i = p.2^i$. We have that $t_{p_i} = t_{|Q|+1-i}$. By the pigeon hole principle, there exist $i, j, 0 \leq i < j \leq |Q|$ such that

$r(p_i) = r(p_j)$. Let $t' = t[p_j]_{p_i}$, and $r' = r[p_j]_{p_i}$. Since equalities may only have been tested on the two occurrences of t_0 in t, and that one is below, p_j and p_i, and the other one is at a parallel position, r' is still compatible with Δ. Hence r' is a successful run of $\mathcal{B}$ on t', and since $t' \notin \mathcal{L}$, $\mathcal{L}(\mathcal{B}) \neq \mathcal{L}$. □

2.3 Tree Automata with One Memory

Definitions and Known Properties

Like pushdown tree automata [Gue83], TA with one memory (TA1M) [CLC05, CLJP07] are TA extended in order to carry an unbounded amount of information along the states in computations. Instead of a stack, a TA1M stores this information in a memory with a tree structure. More precisely, this memory contains a ground term over a memory signature Γ. The memory is updated during the bottom-up computations. The general form of the transitions of TA1M is

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

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

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

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

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

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

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

with $1 \leq i \leq n$.

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

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

where $1 \leq i, j, k \leq n$.

Comparison with TAGC

The internal$_=$ transition rules make possible the simulation of some tests of the TA with constraints by storing some subterms in memory and comparing them. It even is possible to test an equality an unbounded number of time.

Proposition 2.23 *The language of terms of the example 2.21 is recognized by the TA1M* $A = \langle \Sigma, Q, F, \Delta \rangle$ *where*

- $Q = F = \{q_1, q_2, q_3\}$
- $\Delta = \{a- > q_1(a), g(q_1(x)) \to q_2(g(x)), f(q_1(x_1), q_1(x_2)) \to q_1(f(x_1, x_2)), f(q_1(x_1), q_2(x_2)) \to q_3(x_2), f(q_2(x_1), q_1(x_2)) \to q_3(x_1), f(q_2(x_1), q_2(x_2)) \xrightarrow{x_1=x_2} q_3(x_1), f(q_2(x_1), q_3(x_2)) \xrightarrow{x_1=x_2} q_3(x_1), f(q_3(x_1), q_2(x_2)) \xrightarrow{x_1=x_2} q_3(x_1), f(q_3(x_1), q_3(x_2)) \xrightarrow{x_1=x_2} q_3(x_1)\}$

However, since those automata can only store one memory at a time, it has a limitation in the equality tests it can perform. It can easily deal with a bounded number of equality "classes", if all the equal terms of different "classes" appear in distinct parallel subterms: in each of these parallel subterm, the run of the automaton only have to memorize one subterm. But if is not the case, the memory cannot be used to test several "classes" of equalities in full generality. Especially, in a term t, it generally cannot test at the same time that subterms at several positions are equals to a subterm t', and that subterms at other positions are equals to a subterm t'' of t'. However, since some encoding can be made by an involved use of both the states and the memory alphabet, it seems that proving this fact would be difficult. Also, we only state this as a conjecture.

Example 2.24 *Let* $\Sigma = \{f : 2, g : 1, s : 1, a : 0\}$ *and let* $\mathcal{L} = \{f(g^n(s^m(a)), f(s^m(a), g^n(s^m(a)))) \mid n, m \in \mathbb{N}^*\}$. *The language* $\mathcal{L}$ *is recognized by the PCTAGC*$[\approx]$ $A = \langle \Sigma, Q, F, \Delta, C \rangle$ *where*

- $Q = \{q_1, q_2, q_3, q_4, q_5, q_f\}, F = \{q_f\}$;
- $\Delta = \{a \to q_1, s(q_1) \to q_1, s(q_1) \to q_2, g(q_2) \to q_3, g(q_3) \to q_3, g(q_3) \to q_4, f(q_2, q_4) \to q_5, f(q_4, q_5) \to q_f\}$;
- $C = (q_2 \approx q_2) \wedge (q_4 \approx q_4)$.

Conjecture 2.25 *The language of the example 2.24 is not recognizable by a TA1M.*

2.4 Automata on DAG Representations of Terms

Definitions and Known Results

One way to test equalities or disequalities in subterms, is to use a DAG representation of terms and to ensure the maximal sharing properties. Hence instead of computing on terms, the automata will then compute on their DAG representations (called t-dags). Charatonik ([Cha99]) introduced the t-dag automata.

Definition 2.26 *A* t-dag automaton *is a tuple* $\langle \Sigma, Q, F, \Delta \rangle$ *where* Σ *is a finite signature,* Q *is a finite set of states,* $F \subseteq Q$ *is the set of final states, and* Δ *is a set of transitions of the form* $f(q1, \ldots, qn) \to q$ *with* $q, q1, \ldots, qn \in Q$ *and* $f \in \Sigma_n$.

Note that up till now there is no difference between t-dag automata and standard bottom-up tree automata. A t-dag can be *determinist* (resp. *complete*) the same way as a TA (existence of at most (resp. at least) one transition rule for every $f \in \Sigma_n$ and every sequence $q_1, \ldots q)n)$. The difference is that t-dag automata run on t-dags and not terms.

Definition 2.27 *A* run *of a t-dag automaton* $\langle \Sigma, Q, F, \Delta \rangle$ *on a given t-dag* $G = \langle V, succ, \lambda \rangle$ *on* Σ *is a mapping* r *from the set of nodes* V *to the set of states* Q *such that for each node* v *and each* $f \in \Sigma$, *if* $\lambda(v) = f$, *then* Δ *contains a transition* $f(r(succ(1, v)), \ldots, r(succ(n, v)) \to r(v)$. *A run* r *is said* successful *if it maps the root of* G *to a final state. We say that the t-dag automaton* $\mathcal{A}$ accepts *the t-dag* G, *if there is a successful run of* $\mathcal{A}$ *on* G.

A set of t-dags T is *recognizable* if there exists a t-dag automaton A such that $T = \{G \mid A \text{ accepts } G\}$. We denote $\mathcal{L}(A)$ the set of t-dags recognized by A

The classical TA constructions for recognizing the union or the intersection of two recognizable sets also work for t-dag automata as shown in [Cha99]. However, the complementation does not work, since the determinization procedure does not create an equivalent t-dag automaton, in the sense that the determinized t-dag automaton does not necessarily recognized the same set of t-dags. In [ANR05], a formal proof that t-dag automata are not determinizable is given.

If we consider the sets of terms $terms(\mathcal{L}(A))$ represented by the t-dags recognized by some t-dag automaton, we can compare expressiveness of t-dag automata and TA. A recognizable set of terms $\mathcal{L}$ can be described by a t-dag automaton: take a deterministic TA A recognizing $\mathcal{L}$, and $\mathcal{L}$ as a t-dag automaton will recognize $dag(\mathcal{L})$.

The reverse is not true: the description of a recognizable set of t-dags, whose represented terms is not a regular term language is given in [ANR05]. Hence, t-dag automata are more expressive than TA.

Comparison with TAGC

The t-dag structure seems to allow to test equalities and disequalities, since equal terms will necessarily have a same state in the run of a t-dag automaton. In particular, to operate a disequality test between two subterms, you only have to ensure that they will be labeled by two different states in the run.

Example 2.28 *Let* $\Sigma = \{f : 2; a : 0\}$*, and* $\mathcal{L} = \{f(t, t') \mid t, t' \in \mathcal{T}(\Sigma), t \neq t'\}$*. Then the following t-dag automaton* A *recognizes a language of t-dags* $\mathcal{L}'$ *such that* $term(\mathcal{L}') = \mathcal{L}$*.*

$A = \langle \Sigma, Q, F, \Delta \rangle$ *where*

- $Q = \{q_1, q_2, q_f\}$
- $F = \{q_f\}$
- Δ *contains the following transition rules*
  - $a \to q_1$, $a \to q_2$
  - $f(q_1, q_1) \to q_1$, $f(q_1, q_1) \to q_2$
  - $f(q_1, q_2) \to q_f$

Actually, we will show in chapter 4 that the t-dag automata seen as term language recognizers are equivalent to PCTAGC[$\not\approx$] where $\not\approx$ is irreflexive.

However, ensuring that two subterms have the same state labeling in a run does not mean they are represented by the same node of the t-dag. Since you cannot force the successors of a node to be equals, you cannot test equalities with a t-dag automaton

Lemma 2.29 *Let* $\Sigma = \{f : 2; a : 0\}$*, and* $\mathcal{L} = \{f(t, t) \mid t\}$*. There exists a PCTAGC*[$\approx$] *recognizing* $\mathcal{L}$ *but there is no recognizable t-dag language* $\mathcal{L}'$ *such that* $term(\mathcal{L}') = \mathcal{L}$

proof. The term language $\mathcal{L}$ is recognized by the PCTAGC[$\approx$] $A = \langle \Sigma, Q, F, \Delta, C \rangle$ where

- $Q = \{q_1, q_2, q_f\}$
- $F = \{q_f\}$
- Δ contains the following transition rules
  - $a \to q_1$, $a \to q_2$
  - $f(q_1, q_1) \to q_1$, $f(q_1, q_1) \to q_2$
  - $f(q_2, q_2) \to q_f$
- $C = \{q_2 \approx q_2\}$

Briefly, any term of $\mathcal{T}(\Sigma)$ can be recognized by $\mathcal{A}$ in the state q_2: it suffices to label the root by $q2$ and the other positions by q_1. excepand non-deterministically choose to stop and to label the root by q_2. Hence, the transition rule $f(q_2, q_2) \rightarrow$ ensures that $\mathcal{A}$ recognizes the terms $f(t_1, t_2)$ where t_1, t_2 can be any term of $\mathcal{T}(\Sigma)$, and the constraint $q_2 \approx q_2$ forces that t_1 and t_2 are always equals.

Assume $\mathcal{A}$ is a t-dag automaton, with a set of states Q, that recognizes the t-dag language $\mathcal{L}'$. Let $t \in \mathcal{T}(\Sigma)$ be a balanced term of height greater than $|Q|$, and $G = dag(t)$. Let r be an accepting run of $\mathcal{A}$ on $dag(f(t,t))$. Since a same term can occur only once in a dag, then $r(1) = r(2) = r(t)$. Let $q = r(t)$, then there exists a final state q_f in Q such that the transition rule $f(q,q) \rightarrow q_f$ belongs to $\mathcal{A}$. Since t is of height greater than $|Q|$, there is a path on which there are two different nodes u and v in G such that $r(u) = r(v)$. Let $q' = r(u)$, and let p_1 and p_2 be two positions of G such that $G(p_1) = u$ and $G(p_2) = v$. Assume *wlog* that $p_1 < p_2$. Then, let $t' = t[t|_{p_2}]_{p_1}$.

If $p_1 = \varepsilon$, then $dag(f(t,t'))$ and $dag(f(t,t))$ have the same sets of vertices. The only difference is that the second child of the root is v instead of u. Since $r(v) = r(u)$, by keeping the same labeling of nodes as in r, we have an accepting run of $\mathcal{A}$ on $dag(f(t,t'))$.

If $p_1 \neq \varepsilon$, then let $G' = dag(t')$. Then G' and G have the same set of vertices. The only difference is that all the edges oriented to u in G are oriented to V in G'. Hence, the run r' of $\mathcal{A}$ on G' by $r'(u) = r(u)$ respects the transition rules of $\mathcal{A}$ and labels the root of G' by g. Let $G'' = dag(f(t,t')$. All the subterms of t' at positions $p' < p_1$ are not balanced and hence are not subterms of t. Hence the nodes of G'' at positions $1.p'$ and $2.p'$ are distinct. All the subterms of t' at positions $p' = p1.p''$ or at positions p' incomparable with p_1 are also subterms of t at the same positions. Hence there is a unique node of G'' at both positions $1.p'$ and $2.p'$. So the the run r'' defined as $r''(\varepsilon) = q_f$, $r''(1.p) = r(1.p)$ and $r''(2.p) = r'(p)$ is a well-defined run of $\mathcal{A}$ on G'', *i.e.* each node of G'' is labeled by a single state. Hence $\mathcal{A}$ recognizes $dag(f(t,t')$ which is not in $\mathcal{L}'$. □

2.5 Automatic Clauses

This latter comparison is a bit more informal as the previous ones. We compare here a subclass of PCTAGC[≈] with a non-automata formalism: finite sets of Horn clauses with rigid variables [And81]. This formalism was used in several related works [DLL07, ACL09]. These papers do not mention the name of tree automata, they are targeted at the static analysis of security protocols. In chapter 3, we will apply tree automata with global equality constraints to the same application. Therefore, a comparison with this formalism is appropriate.

Definitions and Known Results

We do not recall here the definition of Horn clauses. For an introduction on Horn clauses, see *e.g.* [Pad88]. Following [FSVY91], it is a common approach to represent tree automata by Horn clause sets. A tree automata transition $f(q_1, \ldots, q_n) \to q$ can indeed be encoded into the following first order Horn clause (variables are implicitly universally quantified)

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

where $y_1, \ldots, y_n$ are distinct variables and $q_1, \ldots, q_n, q$ are unary predicate symbols. Let us call *regular clauses* the Horn clauses of the above form. Given a finite set $\mathcal{C}$ of regular clauses (an *automaton* in these settings) and a predicate q (a *state*), the language of $\mathcal{C}$ in q, denoted by $\mathcal{L}(\mathcal{C}, q)$, is the set of terms $t \in \mathcal{T}(\Sigma)$ such that $q(t)$ is a logical consequence of $\mathcal{C}$ ($q(t)$ is in the smallest Herbrand model of $\mathcal{C}$). This definition corresponds exactly to the language of the TA whose transition are encoded by the clauses of $\mathcal{C}$.

One advantage of this presentation of tree automata by Horn clause sets is that it permits to use classical first-order theorem proving techniques in order to decide TA problems. For instance, if $\mathcal{C}$ is a finite set of regular clauses, $t \in \mathcal{T}(\Sigma)$ then $t \in \mathcal{L}(\mathcal{C}, q)$ iff $\mathcal{C} \cup \{q(t) \Rightarrow\}$ is inconsistent, and $\mathcal{L}(\mathcal{C}, q) \neq \emptyset$ iff $\mathcal{C} \cup \{q(x) \Rightarrow\}$ is inconsistent. These sets can be finitely saturated by a resolution calculus with appropriate strategies [Gou05], hence, the above decision problems can be solved using first order theorem provers.

Comparison with TAGC

This approach can also be suitable for studying Rigid Tree Automata. We recall that we define Rigid Tree Automata (definition 3.1) as PCTAGC[$\approx$] where $\approx$ is the identity relation on a subset of the set of states, and that it has been shown to be equally expressive to the full class of PCTAGC[$\approx$] in [FTT08]. We can fit the same presentation of tree automata by Horn clause sets as above for RTA, by distinguishing, in regular clauses, some variables as so called *rigid variables* [And81]. We use below uppercase letters $X, Y...$ for rigid variables and lowercase $x, y...$ for other variables, called *flexible* variables. Recently [DLL07, ACL09], some models of Horn clauses with rigid variables (including regular clauses) have been studied in the context of the verification of security protocols. We recall the definitions and results of [ACL09] in order to establish connections with RTA.

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

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

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

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

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

In [ACL09], a translation of clauses with rigid variables into first order clauses (without rigid variables) preserving satisfiability is proposed. In the case of the above regular clause with rigid variables (reg'), the translation returns

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

where $\overline{x} = (x_q)_{q \in R}$ is a sequence of $|R|$ flexible variables, one variable x_q for each rigid state $q \in R$ (hence one for each rigid variable X_q). Every variable α'_i, $i \leq n$, is either x_{q_i} if α_i is the rigid variable X_{q_i} (*i.e.* if $q_i \in R$) and α'_i is the (flexible) variable $\alpha_i = y_i$ otherwise. Such clauses can alternatively be seen as transitions of tree automata extended with $|R|$ auxiliary registers storing terms of $\mathcal{T}(\Sigma)$. For such an automaton, the values are stored in the registers once and for all at the beginning of the computation (in the variables of $\overline{x}$) and during the application of a transition, the current subterm can be compared to the content of one register (in the case where $\alpha'_i = y_i$).

It is shown in [ACL09] that binary resolution with an appropriate ordered strategy terminates on such clauses as long as there is only one unary predicate; [ACL09] consider also other kinds of clauses, some of them that can be seen as a generalization of RTA to two way and alternating rigid tree automata.

Hence, in the result of [ACL09], termination is limited to automata with one state. The resolution strategy of [ACL09] does not terminate on automata with more than one state and a terminating resolution strategy for this case is not known. Some progress in this direction would enable the application of first order theorem proving techniques to decision problem for RTA. This could permit in particular to consider extensions of RTA with *e.g.* equational tests or language modulo equational theories, like what was done in [JRV08] for standard tree automata using a Horn clauses approach and a paramodulation calculus. In particular, the latter extension (modulo equational theories) is related to the problem of Section 4.3 of chapter 3, and in this context, first order theorem proving tools could provide an efficient alternative to the complicated decision algorithm described this section.

Chapter 3

Rigid Tree Automata and Rewrite Closure

As presented in previous chapter, given a TAGED, one can always compute an equivalent TAGED such that the equality relation is the identity relation on a subset of the states of the automaton. We defined the class of Rigid Tree Automata (RTA), that we first introduced in [JKV09], as the class of positive TAGED, where this property is respected. The additional constraint on the equality relation allows to have decision procedure with lower complexities. 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. On the other side, positive TAGED give a more succinct way to represent equivalent languages. In this chapter, the applications that we aim may be intuitively modelized into some RTA. We will try as much as possible to keep the RTA property when operating transformations on our automata, in order to keep a complexity as low as possible.

Properties like determinism, pumping lemma, Boolean closure, and several decision problems are studied in detail.

Term rewriting systems (TRS) is a general formalism for the symbolic evaluation of terms by replacement of some patterns by others, following rewrite rules. Combining tree automata and term rewriting techniques has been very successful in verification see *e.g.* [BT05, GK00]. In this context, term rewriting systems (TRS) can describe the transitions of a system, the evaluation of a program [BT05], the specification of operators used to build protocol messages [AF01] or also transformation of documents. If a TA $\mathcal{A}$ is used to finitely represent an infinite set $\mathcal{L}(\mathcal{A})$ of states of a system, the rewrite closure $\mathcal{R}^*\big(\mathcal{L}(\mathcal{A})\big)$ of the language $\mathcal{L}(\mathcal{A})$ using $\mathcal{R}$ represents the set of states reachable from states described by $\mathcal{A}$. When $\mathcal{R}^*\big(\mathcal{L}(\mathcal{A})\big)$ is again a TA language, the verification of a safety property amounts to checking for the existence of an error state in $\mathcal{R}^*\big(\mathcal{L}(\mathcal{A})\big)$ (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.* [JRV08]). The reason is the difficulty to capture the behavior of constraints after the application of rewrite rules.

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 linear TRS called invisibly pushdown, whereas this closure is generally not an RTA language. Linear and invisibly pushdown TRS can typically specify cryptographic operators like $\mathsf{decrypt}(\mathsf{encrypt}(x, \mathsf{pk}(A)), \mathsf{sk}(A)) \to x$. A potential application of RTA to the verification of security protocola is described in Section 5. Using RTA instead of positive TAGED is at the same time more intuitive in the field of protocol verification, where we want terms built by the same rules to be equals, and simpler to handle with rewrite systems. It is an original use of tree automata with global constraints, wince they were at first introduced in the field of databases and semi-structured document verifications.

1 RTA: Definition, Examples and Properties

1.1 Definition

We mentioned in section 2.1 of chapter 2 Rigid Tree Automata as being positive TAGED (*i.e.* PCTAGC[$\approx$]) where the equality relation $\approx$ is the identity relation on a subset of the set of states.

Definition 3.1 *A* Rigid Tree Automaton *(RTA for short) is a positive TAGED $\mathcal{A} = \langle \Sigma, Q, F, \Delta, \approx_{\mathcal{A}} \rangle$ such that $\approx_{\mathcal{A}} \subseteq id_Q$. A state q of A is said* rigid *if $q \in dom(\approx_{\mathcal{A}})$. We denote $R = dom(\approx_{\mathcal{A}})$ and we call it the set of* rigid states *of $\mathcal{A}$.*

We may latter define an RTA the following way: $\mathcal{A} = \langle \Sigma, Q, F, R, \Delta \rangle$, where R is the set of rigid states, *i.e.* $\approx_{\mathcal{A}}$ is implicitely defined as the identity relation on $R \subseteq Q$.

1.2 Examples

The PCTAGC[$\approx$] given in example 2.3 and in proposition 2.5 are actually rigid tree automata. Hence we have a first idea of what RTA can do, *i.e.* testing local equalities, or recognizing terms embedding a non-linear pattern-matching. However, it is also possible to test equalities between subterms at arbitrary position in a term.

Example 3.2 *Let $\Sigma = \{a : 0, g : 1, f : 2\}$. The set of terms $t \in \mathcal{T}(\Sigma)$ such that $s_1 = s_2$ for every two subterms $g(s_1)$, $g(s_2)$ of t is recognizable by the following RTA: $\mathcal{A} = \langle \{q, q_r\}, \{q_r\}, \{q, q_r\}, \{a \to q, g(q') \to q_r, f(q', q') \to q \mid q' \in \{q, q_r\}\} \rangle$.* ◇

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

Example 3.3 *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 strict subterm of* t *is recognized by the following RTA on* Σ, $\langle\{q, q_r, q', q_{\mathsf{f}}\}, \{q_r\}, \{q_{\mathsf{f}}\}, \Delta\rangle$, *with*

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

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

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

Example 3.4 *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 following RTA on* Σ, $\langle\{q, q_r, q_a, q_b, q_{\mathsf{f}}\}, \{q_r\}, \{q_{\mathsf{f}}\}, \Delta\rangle$, *with*

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

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

The construction of Example 3.3 cannot be generalized to the characterization of a maximal subterm amongst some subterms. This is shown in the following counter example, using a pumping argument.

Example 3.5 *Let* $\Sigma = \{0:0, g:1, h:2\}$, *and let* $\mathcal{L}_{\max}$ *be the set of terms of the form* $H\big[g^m(0), g^{n_1}(0), \ldots, g^{n_k}(0)\big]$ *where* k *is an arbitrary positive integer,* $m \geq n_1, \ldots, n_k$, H *is an* $k+1$*-context made of the symbol* h *only, and* g^n *represents* n *nested symbols* g.

Fact 3.6 $\mathcal{L}_{\max}$ *is not an RTA language.*

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

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

1.3 Pumping Lemma

Following some ideas developed in the proof of Fact 3.6 above, we propose a weak form, adapted to RTA, of the pumping (or iteration) lemma for TA. Pumping on runs of RTA is not as easy as for standard TA. Indeed, we must take care of the position of rigid states in order to preserve recognizability. For this reason, the transformation of a subterm must be performed in several branches in parallel (instead of one single branch for TA) in order to preserve the rigidity condition. Moreover, we cannot repeat a term containing a rigid state, because the same rigid state cannot label two different positions on the same branch.

Lemma 3.7 *For all RTA $\mathcal{A} = \langle Q, R, F, \Delta \rangle$, for all terms $t \in \mathcal{L}(\mathcal{A})$ 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 $n \geq 0$, $C\big[C'[D^n[u]], \ldots, C'[D^n[u]]\big] \in \mathcal{L}(\mathcal{A})$.*

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

With the rigidity condition in the definition of successful runs, a rigid state can occur at most once on a path of r. Hence, there exist two positions $p_0 < p'_0 < p$ such that $|p'_0| - |p_0| > |Q|$ and no rigid state occurs between p_0 and p'_0 in r. By a pigeon-hole principle, there exist two positions p_1, p_2 with $p_0 \leq p_1 < p_2 \leq p'_0$ labeled with the same state of $Q \setminus R$ in r. We let $u := t|_{p_2}$ and $D = (t|_{p_1})[x_1]_{p_2 - p_1}$. This situation is depicted in Figure 3.1.

In order to preserve the property of being a run while iterating D, we need to take care of rigid states above p_0 in r (rigid states below p'_0 and below D are not affected by iteration of D). Let π_1 be the maximal position of a rigid state in r smaller than p_0 wrt the prefix ordering. Let $q_r = r(\pi_1)$ and let $\pi_2, \ldots, \pi_k$ be the other positions of q_r in r. Note that by definition of r being a run the positions $\pi_1, \ldots, \pi_k$ are pairwise incomparable wrt the prefix ordering. We let $C = t[x_1]_{\pi_1} \ldots [x_k]_{\pi_k}$ and $C' = (t|_{\pi_1})[x_1]_{p_1 - \pi_1}$ ($x_1, \ldots, x_k$ are distinct variables).

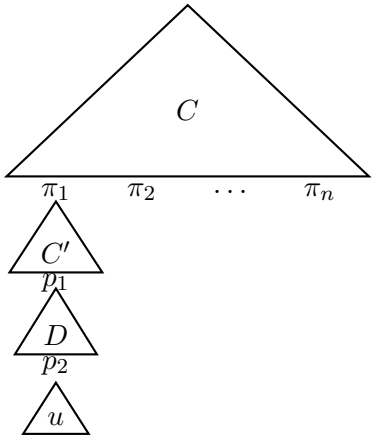

Figure 3.1: Pumping lemma

Since $r(p_1) = r(p_2)$ and there are no rigid states between p_1 and p_2, we can construct a run on every $C'[D^n[u]]$. Moreover, $t|_{\pi_i} = t|_{\pi_j}$ for all $i, j \in \{1..k\}$, hence we may assume wlog that the subruns $r|_{\pi_i}$ are equal for all $i \in \{1..k\}$. It follows that we can perform the same operation as in $C'[D^n[u]]$ under each $r|_{\pi_i}$, and that $C\big[C'[D^n[u]], \ldots, C'[D^n[u]]\big] \in \mathcal{L}(\mathcal{A})$. □ As usual, such a lemma can be used to show that a language is not in RTA.

Example 3.8 *As a consequence of the above pumping lemma, we can show that the set $\mathcal{B}$ of balanced binary trees built over the signature $\{a : 0, f : 2\}$ is not an RTA language. Assume indeed that it is recognized by an RTA $\mathcal{A} = \langle Q, R, F, \Delta\rangle$ and let $t \in \mathcal{L}(\mathcal{A})$ such that $d(t) > (|Q|+1)|R|$ and C, C', D, u be as in Lemma 3.7. By hypothesis, $C'[D[u]]$ is balanced, but for any $n > 1$, $C'[D^n[u]]$ is not balanced since C' and D are not trivial. It contradicts the fact that $C\big[C'[D^n[u]], \ldots, C'[D^n[u]]\big] \in \mathcal{L}(\mathcal{A})$ by Lemma 3.7.* ◇

1.4 Boolean Closure

We show below that the class of RTA languages is closed under union and intersection but not under complement. These are not new results, since they use the same proofs as the ones already done for TAGED (see [FTT08]). However, we give here a lower bound for intersection, and an effective example of a RTA language whose complement is not a RTA language.

Theorem 3.9 *Given two RTA $\mathcal{A}_1$ and $\mathcal{A}_2$, there exist two RTA of respective sizes $O(|\mathcal{A}_1| + |\mathcal{A}_2|)$ and $O(2^{|\mathcal{A}_1||\mathcal{A}_2|})$, constructed respectively in polynomial and exponential time, and recognizing respectively $\mathcal{L}(\mathcal{A}_1) \cup \mathcal{L}(\mathcal{A}_2)$ and $\mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$.*

proof. Let $\mathcal{A}_i = \langle Q_i, R_i, F_i, \Delta_i\rangle$ with $i = 1, 2$. For $\mathcal{L}(\mathcal{A}_1) \cup \mathcal{L}(\mathcal{A}_2)$, we do a classical disjoint union of automata. Let us assume *wlog* that the

state sets Q_1, Q_2 of $\mathcal{A}_1$ and $\mathcal{A}_2$ are disjoint. Like for the union of TA, the RTA $\mathcal{A}$ is obtained by disjoint union of the state sets, rigid state sets, final state sets and transitions sets.

For $\mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$, it is easy to construct a positive TAGED $\mathcal{A}'$ recognizing $\mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$ by a Cartesian product operation like for standard TA.

We can use Lemma 1 of [FTT08] in order to transform this positive TAGED into an RTA recognizing the same language, at the price of an exponential blowup. Combining the two above steps results in an exponential construction for the intersection of RTA. □

The following lemma shows that the exponential time complexity for the construction of the intersection automaton in Theorem 3.9 is a lower bound, with a reduction of the EXPTIME-complete problem of the non-emptiness of the intersection of n TA.

Lemma 3.10 *Given n TA $\mathcal{A}_1, \ldots, \mathcal{A}_n$ on Σ, we can compute in polynomial time two RTA $\mathcal{A}_\times$ and $\mathcal{A}_r$, both of size $O\big(|\mathcal{A}_1| + \ldots + |\mathcal{A}_n|\big)$, and such that $\mathcal{L}(\mathcal{A}_1) \cap \ldots \cap \mathcal{L}(\mathcal{A}_n) = \emptyset$ iff $\mathcal{L}(\mathcal{A}_\times) \cap \mathcal{L}(\mathcal{A}_r) = \emptyset$.*

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

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

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

Note that the above construction also works (hence Lemma 3.10 also holds) for n given RTA. With Lemma 3.10, we have a polynomial time reduction into the non-emptiness of the intersection of two RTA of the problem of the intersection non-emptiness for n TA (given n TA $\mathcal{A}_1, \ldots, \mathcal{A}_n$, do we have $\mathcal{L}(\mathcal{A}_1) \cap \ldots \cap \mathcal{L}(\mathcal{A}_n) \neq \emptyset$?), a problem which is known to be EXPTIME-complete [Sei94]. Since by Theorem 3.9, the intersection of two RTA is an RTA, and the emptiness of RTA can be decided in linear time (Theorem 3.19 below), we conclude that EXPTIME is a lower bound for the construction of an RTA for the intersection. Moreover, in the above construction, $\mathcal{A}_\times$ is a TA if every $\mathcal{A}_i$ ($1 \leq i \leq n$) is a TA. Hence the intersection of an RTA with a TA also leads to an exponential construction.

2 Deterministic and Visibly Rigid Tree Automata

Non-determinism is crucial for an RTA recognizing the terms of the form $f(t,t)$ like in Example 2.3. Indeed, in a bottom-up computation, such an automaton needs to guess both positions of the two occurrences of t under the symbol f, and put one rigid state at these positions.

Example 3.11 *Let us come back to Example 2.3, where* $\Sigma = \{a:0, b:0, f:2\}$ *and the RTA* $\mathcal{A}$ *with transition set* $\{a \to q|q_r, b \to q|q_r, f(q,q) \to q|q_r, f(q_r,q_r) \to q_f\}$ *recognizing* $\{f(t,t) \mid t \in \mathcal{T}(\Sigma)\}$. *Applying a classical subset construction to the transition set of* $\mathcal{A}$ *returns a deterministic set of transitions*

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

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

In this section, we will study the expressiveness of deterministic RTA. We will show that they are strictly less expressive than RTA, but still more expressive than standard TA. Hence RTA cannot be determinized, and moreover, determinism is not sufficient here to provide the closure by complementation. Therefore, we will define a subclass of RTA, inspired by a previous work on tree automata with memory [CLJP07], that can be determinized, but which is still not closed under complementation.

2.1 Determinism and Completeness

Definition 3.12 *A* deterministic rigid tree automaton *(DRTA) on a signature* Σ *is an RTA* $\mathcal{A}$ *whose underlying TA* $ta(\mathcal{A})$ *is deterministic.*

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

Theorem 3.13 *DRTA* $\subsetneq$ *RTA.*

proof. Let $\Sigma = \{a:0, f:2\}$. The language $\mathcal{L} = \{f(t,t) \mid t \in T(\Sigma)\}$ is recognized by the RTA of Example 2.3, without the transitions rules for symbol b.

We show now that $\mathcal{L}$ is not recognized by a DRTA. Assume that there is a DRTA $\mathcal{A} = \langle Q, R, F, \Delta\rangle$ recognizing $\mathcal{L}$. For every term t such that there exists a run r of $\mathcal{A}$ on t, and for every path $p \in \mathcal{P}os(t)$ from the root to a leaf, let us denote $|p|_R = |\{p' \mid p' \leq p, r(p') \in R\}|$,

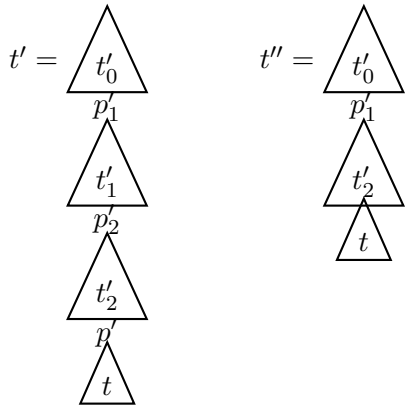

Figure 3.2: Proof that DRTA $\subsetneq$ RTA

the number of rigid states occurring on the path p. Then let us denote $|t|_R$ the maximal $|p|_R$ among all the paths p from the root to a leaf of t. Then, we define $max_R = max_{t \in \mathcal{L}(\mathcal{A})} |t|_R$.

Since each rigid state can only appear once on a path, otherwise it would not respect the rigid condition, there is at most $|R|$ occurrences of rigid states on every path. Hence max_R is well-defined and is lower or equals to $|R|$ Given Let t be a term recognized by $\mathcal{A}$ such that $|t|_R = max_R$. Let r be the unique run of $\mathcal{A}$ on t, and p be a path of t from the root to a leaf such that $|p|_R = max_R$.

We build a tree t' such that there exists a position $p' \in \mathcal{P}os(t')$, $|p'| > |Q| - |R|$ and $t'|_{p'} = t$. Since $f(t', t')$ is recognized by $\mathcal{A}$, there exists a (unique) run r' on t'. Since $\mathcal{A}$ is deterministic, we know that $r'|_{p'} = r$. Hence there exists a path in r' from the position p' to a leaf that contains the maximal number of rigid states. So for each strict prefix p'_0 of p', $r'(p'_0) \in Q \setminus R$. Since $|p'| > |Q| - |R|$, there exists two strict prefixes p'_1, p'_2 of p', such that p'_1 is a strict prefix of p'_2 and $r'(p'_1) = r'(p'_2)$. Let t'' be the tree $t'[t'_{p'_2}]_{p'_1}$. This construction is illustrated in Figure 3.2.

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

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

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

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

Theorem 3.14 *TA $\subsetneq$ DRTA.*

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

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

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

But this language if not regular. □

2.2 Visibly Rigid Tree Automata

We propose here a class of restricted RTA which can be determinized. The definition of the restriction is inspired by the theory of visibly pushdown automata (VPA) [AM04]. VPA define a subset of context-free languages closed under intersection and complement. They were generalized to tree recognizers in [CR07, CLJP07]. The idea in these works is that the signature Σ is partitioned into $\Sigma = \Sigma_c \uplus \Sigma_r \uplus \Sigma_\ell$ and the operation performed by the VPA on the stack depends on the current symbol in the input: if it is a *call* symbol of Σ_c, the VPA can only do a push, for a *return* symbol of Σ_r it can do a pop and it must leave the stack untouched for a *local* symbol of Σ_ℓ. The transition of a VPA follows this discipline, determinization is possible for this class of pushdown automata, and it is closed under intersection and complement.

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

Definition 3.15 *A visibly rigid tree automaton (VRTA) is an RTA $\mathcal{A} = \langle Q, R, F, \Delta\rangle$ on a signature Σ such that there exists a partial function ν from Σ to R such that for every transition $f(q_1, \ldots, q_n) \to q \in \Delta$, $q = \nu(f)$ if ν is defined on f and $q \in Q \setminus R$ otherwise.*

Example 3.16 *The RTA of Example 3.2 is visibly rigid, with a function ν defined only on g by $\nu(g) = q_r$. The DRTA in the above proof of Theorem 3.14 is also visibly rigid, with the same function.*

Conversely, the RTA of Example 2.3 (recognizing the terms $f(t,t)$ with $t \in \mathcal{T}(\{a:0, b:0, f:2\})$ is not visibly rigid. Intuitively, some nondeterminism is needed for the bottom-up recognition of this language (because t may contain the symbol f), and it is not compatible with the visibly rigid condition. Indeed, the above language is not regular, hence at least one rigid state is necessary for the definition of a RTA recognizing it. Defining rigid states for $\nu(a)$ and $\nu(b)$, is pointless (it can be simulated by standard tree automata). Hence,$\nu(f)$ must be defined in order to ensure the visibly rigid condition, but this would contradict the recognition a term such as e.g. $f(f(a,a), f(a,a))$. ◇

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

Theorem 3.17 *Given a VRTA $\mathcal{A}$ on Σ, a deterministic VRTA $\mathcal{A}'$ on Σ of size exponential in $|\mathcal{A}|$ and such that $\mathcal{L}(\mathcal{A}') = \mathcal{L}(\mathcal{A})$ can be constructed in exponential time.*

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

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

The RTA $\mathcal{A}'$ is deterministic. Moreover, because of the visibly rigid condition for $\mathcal{A}$, every state of $\mathcal{A}'$ occurring in Δ (*i.e.* every state of $\mathcal{A}'$ with a non empty language) is either a subset of $Q \setminus R$ (and it is not a rigid state of $\mathcal{A}'$) or is a singleton subset of R (and it is a rigid state of $\mathcal{A}'$). Hence, given a function ν associated to the VRTA $\mathcal{A}$ like in Definition 3.15, there exists a function ν' making $\mathcal{A}'$ a VRTA, defined by $\nu'(f) = \{q_r\}$ iff $\nu(f) = q_r \in R$.

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

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

Theorem 3.18 *The class of VRTA languages is not closed under complement.*

proof. Let us consider the language $\mathcal{L}_g$ of Example 3.2: the set of terms $t \in \mathcal{T}(\Sigma)$, with $\Sigma = \{a : 0, g : 1, f : 2\}$, such that $s_1 = s_2$ for every two

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

Table 3.1: Summary of closure and decision results

subterms $g(s_1)$, $g(s_2)$ of t. $\mathcal{L}_g$ is recognized by the VRTA $\mathcal{A}$ given in Example 3.2 but its complement is not a language of VRTA.

Assume that the complement $\mathcal{T}(\Sigma) \setminus \mathcal{L}_g$ of $\mathcal{L}_g$ is recognized by a VRTA $\mathcal{A}'$ and let ν' be the function associated to $\mathcal{A}'$ like in Definition 3.15. Since $\mathcal{L}_g$ is not regular, $\mathcal{L}(\mathcal{A}')$ is neither regular, and hence $\mathcal{A}'$ has to contain at least one rigid state q_r such that $\mathcal{L}(\mathcal{A}', q_r) \neq \emptyset$. Hence, there exists a function symbol $h \in \Sigma$ such that $\nu(h) = q_r$. It cannot be g, otherwise $\mathcal{A}'$ would not be able to recognize any term of the form $f(g(t_1), g(t_2))$ with $t_1 \neq t_2$ (such a term is in the complement of $\mathcal{L}_g$). It cannot be f either, otherwise $\mathcal{A}'$ would not be able to recognize terms of the form $f(g(t_1), g(t_2))$ with $t_1 = f(t_3, t_4)$ and $t_1 \neq t_2$ (those terms are also all is in the complement of $\mathcal{L}_g$). Hence h has to be a, but with rigid states bound to constant symbols, VRTA do not have more expressive power than standard TA. It follows that there does not exist any VRTA $\mathcal{A}'$ recognizing $\mathcal{T}(\Sigma) \setminus \mathcal{L}_g$. □

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

3 Decision problems

All decision problems studied in section 1.2 naturally apply to RTA. However, the decidability of the emptiness problem for PCTAGC[≈], *i.e.* positive TAGED has not been proved. We will do it here, especially because we have a lower complexity with RTA. We will also study two decision problems for RTA: intersection non-emptiness, that have not been studied for TAGED, and finiteness. Table 3.1 provides a summary of closure and decision results and a comparison with plain TA and full positive TAGED.

3.1 Emptiness

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

decide emptiness for the underlying TA.

Theorem 3.19 *The emptiness problem is decidable in linear time for RTA.*

proof. Let $\mathcal{A} = \langle \Sigma, Q, R, F, \Delta \rangle$ and let $rigid(\mathcal{A}) = \langle \Sigma, Q, Q, F, \Delta \rangle$ be a copy of $\mathcal{A}$ where every state is rigid. We show that the emptiness of $\mathcal{L}(\mathcal{A})$ and $\mathcal{L}\big(rigid(\mathcal{A})\big)$ and $\mathcal{L}\big(ta(\mathcal{A})\big)$ are equivalent. The latter problem (emptiness for standard TA) is known to be decidable in linear-time (see *e.g.* [CLDG$^+$07]) with an algorithm marking the inhabited states of $ta(\mathcal{A})$ and appropriate data structure for the transitions rules. The idea of the proof is that if $\mathcal{L}(ta(\mathcal{A}))$ is not empty, then the classical "state marking" algorithm builds a witness which respects the rigidity condition for all states, and is therefore a witness for $\mathcal{L}(\mathcal{A})$ non-emptiness.

In order to establish the above equivalence, we use a similar algorithm for $\mathcal{A}$ except that every inhabited state q is marked by a witness (minimal) term $t_q \in \mathcal{L}(rigid(\mathcal{A}), q)$ 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 [CLDG$^+$07]) 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. $\mathcal{L}(rigid(\mathcal{A}), q) \neq \emptyset$,

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

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

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

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

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

In the deterministic case, there is at most one labelling of the term t compatible with the transition rules. It can be computed in PTIME

and it can be checked in PTIME that this labelling is a successful run. Hence the membership problem is decidable in PTIME for DRTA.

3.2 Intersection non-Emptiness

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

Theorem 3.20 *Intersection non-emptiness is* EXPTIME*-complete for RTA.*

proof. The upper-bound is a consequence of Lemma 3.10, Theorem 3.9 and Theorem 3.19. The lower-bound follows from the EXPTIME-hardness of the problem for TA [Sei94]. □

3.3 Finiteness

Theorem 3.21 *Finiteness is decidable in* PTIME *for RTA.*

proof. Like for TA [CLDG+07], checking finiteness amounts to detecting (in PTIME) some loops and paths in the accessibility graph of an RTA. The accessibility graph of a given RTA $\mathcal{A} = \langle Q, R, F, \Delta \rangle$ is an oriented graph $G_{\mathcal{A}} = \langle Q, E_{\mathcal{A}} \rangle$ whose set of vertexes is Q and set of edges is $E_{\mathcal{A}} := \{\langle q, q' \rangle \mid \exists f(\dots q \dots) \to q' \in \Delta\}$. 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 $\mathcal{L}(\mathcal{A})$ is infinite iff there exists a state $q \in Q \setminus R$ such that $\mathcal{L}(\mathcal{A}, q) \neq \emptyset$, 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 3.7. If $\mathcal{L}(\mathcal{A})$ is infinite then it contains a term t of depth larger than $(|Q| + 1)|R|$. The idea is that the loop on q is the path from the variable position up to the root of the context D in a successful run r of $\mathcal{A}$ on t, and the path from q to a final state is the path from the root of D up to the root of t in r.

Checking that $\mathcal{L}(\mathcal{A}, q) \neq \emptyset$ can be done in linear time according to Theorem 3.19, 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 $\mathcal{L}(\mathcal{A})$ can be checked in polynomial time. □

4 Rewrite Closure

Following the motivations presented in the introduction of this chapter, we study here the closure under term rewriting of RTA languages. We observe first that in general, the rewrite closure of an RTA language is not an RTA language (Section 4.1) and even not recursive (Section 4.2) for linear and collapsing TRS. This is in contrast with TA languages,

which are closed under rewriting with such TRS [Sal88]. We show next that, under a syntactical restriction, namely for a linear and *invisibly pushdown* TRS $\mathcal{R}$, it is decidable whether a given tree belongs to the rewrite closure of a given RTA language (Section 4.3).

4.1 Linear and Collapsing Rewrite Systems

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

Proposition 3.22 *$\mathcal{R}^*(\mathcal{L})$ is not an RTA language in general when $\mathcal{L}$ is an RTA language and $\mathcal{R}$ a linear and collapsing TRS.*

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

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

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

The terms of the closure $\mathcal{R}^*\big(\mathcal{L}(\mathcal{A})\big)$ of $\mathcal{L}(\mathcal{A})$ by $\mathcal{R}$ have a similar form except that the number of g's in the different arguments might not be equal. They only have to be all less than or equal to the number of g's in the leftmost argument. The intersection of this set $\mathcal{R}^*\big(\mathcal{L}(\mathcal{A})\big)$ with the regular tree language containing the terms of the form $H\big[g^*(0), \ldots, g^*(0)\big]$ is the language of Example 3.5, which is not recognized by RTA. It follows that $\mathcal{R}^*\big(\mathcal{L}(\mathcal{A})\big)$ is not an RTA language. □

Note that the terms $H\big[g^m(0), g^{n_1}(0), \ldots, g^{n_k}(0)\big]$ in the language of Example 3.5 are $\mathcal{R}$-normal forms in the above counterexample in the proof of Proposition 3.22 (since they do not contain the symbol f). Hence, restricting to the terms of the rewrite closure in normal form does not help: the intersection of $\mathcal{R}^*\big(\mathcal{L}(\mathcal{A})\big)$ with $\mathcal{R}$-normal-forms is not an RTA language in general, when $\mathcal{A}$ is an RTA and $\mathcal{R}$ a linear and collapsing TRS.

4.2 Undecidability of Membership Modulo

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

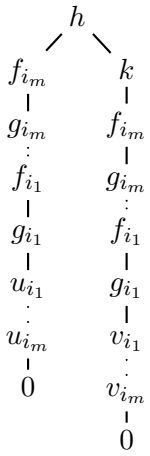

Figure 3.3: Undecidability of membership modulo: encoding of a solution of PCP.

Theorem 3.23 *Membership modulo is undecidable for RTA and linear and collapsing TRS.*

proof. Let Γ be a finite alphabet and let $u_1, v_1 \dots u_n, v_n \in \Gamma^*$ be a PCP instance P. A solution of this PCP instance is a sequence $i_1, \dots, i_m$ of integers smaller or equal to n such that $u_{i_1} \dots u_{i_m} = v_{i_1} \dots v_{i_m}$.

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 the language (for all $w = a_1, \dots, a_p \in \Gamma^*$, the term $a_1(\dots a_p(t))$ is written $w(t)$)

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

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

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

The tree language $\mathcal{L}$ is recognizable by an RTA on Σ. Hence the following property permits us to conclude the proof of Theorem 3.23.

Lemma 3.24 *$h(0, 0) \in \mathcal{R}^*(\mathcal{L})$ iff P has a solution.*

The if direction is easy. Let $i_1, \dots, i_m$ be a solution of P and let $h(s, k(s))$ be the term of $\mathcal{L}$ corresponding to this solution (*i.e.* $s = f_{i_m}(g_{i_m}(\dots f_{i_1}(g_{i_1}(w(0)))))$ and $w = u_{i_1} \dots u_{i_m}(0) = v_{i_1} \dots v_{i_m}(0)$). This term is depicted in Figure 3.3. The s in the left branch can be reduced to 0 using the first rule of $\mathcal{R}$, and the s in the second branch can be reduced to $k(f_{i_m}(v_{i_m}(0)))$ using the two next rules of $\mathcal{R}$. This latter term is in turn reduced to 0 using the last rule of $\mathcal{R}$.

For the only if direction, assume that $\mathcal{L} \ni h(s, k(s)) \xrightarrow[\mathcal{R}]{*} h(0, 0)$. In order to show that in this case, s corresponds to a solution, it is

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

4.3 Linear and Invisibly Pushdown Rewrite Systems

We show in this section that the problem of membership modulo becomes decidable with some further syntactic restrictions on $\mathcal{R}$, based on the definition of visibly pushdown automata [AM04].

The VPA (see also Section 2.2) recognize languages of words. They were generalized into tree recognizers in [CR07, CLJP07]. In [CR07], Chabin and Rety show that the class of visibly pushdown tree automata (VPTA) languages is closed under rewriting with so called linear visibly context-free TRS. We use a similar definition in order to characterize a class of TRS for which membership modulo is decidable. In the following definition, we assume a partition of the signature Σ into $\Sigma_c \uplus \Sigma_r \uplus \Sigma_\ell$.

Definition 3.25 *A collapsing TRS $\mathcal{R}$ is called* inverse-visibly pushdown *(invisibly pushdown) if for every rule $\ell \to x \in \mathcal{R}$, $d(\ell) \geq 1$, x occurs once in ℓ, and if x occurs at depth 1 in ℓ then $\ell \in \mathcal{T}(\Sigma_\ell, \mathcal{X})$, otherwise, $\ell(\varepsilon) \in \Sigma_c$, the symbol immediately above x is in Σ_r and all the other symbols of ℓ are in Σ_ℓ.*

Example 3.26 *The TRS* $\mathcal{R} = \{\mathsf{fst}(\mathsf{pair}(x_1, x_2)) \to x_1, \mathsf{snd}(\mathsf{pair}(x_1, x_2)) \to x_2, \mathsf{dec}(\mathsf{inc}(x)) \to x, \mathsf{decrypt}(\mathsf{encrypt}(x, \mathsf{pk}(\mathsf{A})), \mathsf{sk}(\mathsf{A})) \to x\}$ *is linear and invisibly pushdown with* $\Sigma_c = \{\mathsf{fst}, \mathsf{snd}, \mathsf{dec}, \mathsf{decrypt}\}$ *and* $\Sigma_r = \{\mathsf{pair}, \mathsf{inc}, \mathsf{encrypt}\}$, $\Sigma_\ell = \{\mathsf{pk}, \mathsf{sk}, \mathsf{A}\}$. ◇

The TRS in the proof of Proposition 3.22, $\{f(g(x)) \to x\}$, is invisibly pushdown, but not the one in the proof of Theorem 3.23. Indeed, there exists no partition of Σ making this latter TRS invisibly pushdown. According to Definition 3.25, having a rule $g_i(x) \to x$ implies that $g_i \in \Sigma_\ell$ but also having a rule $g_j(f_i(v_i(x))) \to x$ implies that $g_i \in \Sigma_c$.

Theorem 3.27 *Membership modulo is decidable for RTA and linear and invisibly pushdown TRS.*

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

1. we compute a (big) context-free tree grammar that generates all terms that can rewrite to a single variable by $\mathcal{R}$ rules, such that there exists a run of $\mathcal{A}$ on them where the positions of the rigid states are not contradictory.

2. we add initial rules to the grammar in order to make it generate terms that rewrite to t instead of those rewriting to a variable.

3. finally, we transform the tree grammar obtained into a visibly pushdown tree automaton, and take the subterms under rigid states as independent languages. We replace each language under an occurrence of a rigid state by the intersection of all languages under all occurrences of the same rigid state.

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

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

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

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

Definition 3.28 *A labelling ξ of a term t by tuples of the form $\langle \frac{q_1}{q_2}, \ell, \mathit{lbl}, \mathit{occ}, \mathit{occ}_x, < \rangle$ is said to be* valid *if*

1. $\forall p \in \mathcal{P}os(t)$, $t(p)(lq(\xi(p.1)), \ldots, lq(\xi(p.n))) \to rq(\xi(p)) \in \Delta$,

2. *there do not exist two rigid states* q_{r_1}, q_{r_2} *and two positions* p, p' *such that* $q_{r_1} <^{\xi(p)} q_{r_2}$ *and* $q_{r_2} <^{\xi(p')} q_{r_1}$,

3. *there do not exist two positions* p *and* $p.w$ *such that a rigid state* q_r *appears in* $occ_x^{\xi(p)}$ *and in* $occ^{\xi(p.w)}$,

4. *there do not exist three positions* $p, p.w$ *and* p' *such that a rigid state* q_{r_1} *appears in* $occ_x^{\xi(p)}$, *a distinct rigid state* q_{r_2} *appears in* $occ^{\xi(p.w)}$, *and* $q_{r_1} <^{\xi(p')} q_{r_2}$.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

In order to generate terms rewriting to t instead of x, we just need to add initial rules to the grammar. Let S be the initial non-terminal symbol (of arity zero). For each valid labelling ξ of t such that $lq(\xi(\varepsilon))$ is a finite state of $\mathcal{A}$, we add to our grammar the production rule:

$$S := u$$

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

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

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

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

5 Application to the Verification of Security Protocols

In this final section, we would like to present an application of RTA to the verification of security protocols; this model is also presented in [JKV] (extends version of [JKV09]). Our purpose is not to propose new results in this domain, but rather to illustrate the potential of RTA for the automatic verification of some infinite state systems, in particular communicating processes.

Using automata for protocol analysis is a quite popular approach, see *e.g.* [CLC05, GK00, Gou05]. In particular it is possible to analyze protocols with infinitely many sessions. But this kind of analysis has limitations due to approximations with regular sets. Such approximations may conduct to false alarms, as discussed *e.g.* in [AC02] or [ACL09]. The approach with RTA overcomes several sources of imprecision such as incorrect chaining of messages sent by agents, or ignoring the multiple occurrences of variables in the body of messages sent. Moreover, rigid state also permit to model a local finite memory in which both honest and dishonest agents can store read messages. This feature is generally not supported in other models.

5.1 Protocol Model

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

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

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

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

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

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

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

As noticed in Section 4.3, the TRS containing the above rules is linear invisibly (see Example 3.26).

Example 3.29 *We consider as a running example a simplified version of the mutual authentication protocol* `SPLICE/AS` *[YOM91], which consists of the following two messages exchanged between a client C and a server S.*

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

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

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

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

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

The original `SPLICE/AS` protocol [YOM91] contains additional messages for the distribution of public keys $\mathsf{pk}(S)$ and $\mathsf{pk}(S)$ by a trusted authority `AS`, and timestamps. Here, we make the simplifying assumption that every public key is known by everyone (since everyone can obtain it from `AS`), and we skip the timestamps. ◇

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

A *program* is a finite set of agents, and each agent is a finite sequence of pairs of *instructions* of the form $\mathsf{recv}(x).\mathsf{send}(s).i$ where i is a program point (in an arbitrary domain), $x \in \mathcal{X}$ and $s \in \mathcal{T}(\Sigma, \mathcal{X})$. We assume that moreover every agent starts with an initial program point and that all the program points in a program are pairwise distinct. Note that every message is received as a variable (the argument of recv is always a variable). Hence recv acts as a variable binder, like in [AF01]. Every agent is supposed to be closed, *i.e.* every variable x occurring in a $\mathsf{send}(s)$ is in the scope of a binder $\mathsf{recv}(x)$. For convenience, we assume that the variables in different instructions $\mathsf{recv}(x)$ of a program are distinct.

Example 3.30 *An example of program executing the simplified version of the `SPLICE/AS` protocol of Example 3.29 is made of the two following agents called C and S.*

$$\begin{array}{ll} C: & c_0.\mathsf{recv}(x).\mathsf{send}\big(\mathsf{pair}\big(\mathsf{pair}(C,S), \mathsf{encrypt}(\mathsf{pair}(C, \mathsf{encrypt}(N, \mathsf{pk}(S))), \mathsf{sk}(C))\big)\big).c_1 \\ S: & s_0.\mathsf{recv}(y).\mathsf{send}\big(\mathsf{pair}\big(\mathsf{pair}(t_s, t_c), \mathsf{encrypt}(\mathsf{pair}(t_s, \mathsf{inc}(t_n)), \mathsf{pk}(t_c))\big)\big).s_1 \end{array}$$

$$\text{where } t_s = \mathsf{snd}(\mathsf{fst}(y)), t_c = \mathsf{fst}(\mathsf{fst}(y)),$$
$$t_n = \mathsf{decrypt}\big(\mathsf{snd}\big(\mathsf{decrypt}(\mathsf{snd}(y), \mathsf{pk}(t_c))\big), \mathsf{sk}(S)\big)$$

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

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

5.2 Protocol Semantics

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

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

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

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

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

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

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

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

Example 3.31 *The following sequence of configurations represents a valid execution of the program in Example 3.30, where the agent's messages are smoothly transferred without tampering.*

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

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

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

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

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

Note that in the sequence, the two agents exchange the messages of the protocol, as described in Example 3.29, because $s_1 \xrightarrow[\mathcal{R}]{*} \mathsf{pair}\big(\mathsf{pair}(S, C), \mathsf{encrypt}(\mathsf{pair}(S, \mathsf{inc}(N)), \mathsf{pk}(C))\big)$. ◇

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

5.3 RTA Construction

We show below that it is possible to build an RTA $\mathcal{A}$ recognizing exactly the sets of messages N (representing the state of the attacker's knowledge) in reachable configurations of a given program P. By reachable, we mean reachable from an initial configuration, which is specified precisely below and is assumed to be part of the problem. The RTA $\mathcal{A}$ models both the behavior of the honest agents and of the attacker. It uses uses one rigid state to memorize every message received by the

honest agents (the codomain of the substitution σ in configuration). The first component of configurations (program points of all agent) is encoded directly into the states (as the amount of information needed is finite).

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

Example 3.32 *For the program of Example 3.30 (for the simplified version of the* `SPLICE/AS` *protocol presented in Example 3.29), with two agents called C and S, we have the states $q_{c_0 s_0}$, $q_{c_1 s_0}$, $q_{c_0 s_1}$, $q_{c_1 s_1}$.* $\diamond$

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

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

Example 3.33 *For the program of Example 3.30, we have the rigid states $q^x_{c_0 s_0}$, $q^y_{c_0 s_0}$, $q^x_{c_1 s_0}$, $q^y_{c_1 s_0} \ldots$ Intuitively, $\mathcal{A}$ will be such that $\mathcal{L}(\mathcal{A}, q^y_{c_1 s_0})$ is exactly the set of terms t such that there exists a reachable configuration $(\{c_1, s_0\}, \{x \mapsto t_0, y \mapsto t\}, N)$, for some t_0 and N.* $\diamond$

Now, we will describe the transitions of $\mathcal{A}$ modeling the operations recv and send of the agents. The idea is that when an agent P_i has an instruction $\mathsf{send}(s)$, then $\mathcal{A}$ will perform pattern matching of s, using transitions similar to the ones described in the construction of Proposition 2.5. Like in Proposition 2.5, we consider for this purpose some auxiliary states of the form $q^u_{i_1 \ldots i_n}$ for every strict subterm u of s and tuple $(i_1, \ldots, i_n)$ of program points values. Note that for every variable $x \in vars(s)$, one rigid states $q^x_{i_1 \ldots i_n}$ has already been added above.

Let $(i_1, \ldots, i_n)$ be a tuple of program point values, such that i_j occurs in the agent P_j and is followed by the instructions $\mathsf{recv}(x).\mathsf{send}(s).i'_j$. Then the following transitions are added to $\mathcal{A}$ for the recognition of s (like in Proposition 2.5, we assume that s is not a variable):

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

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

Example 3.34 *Let us consider for instance the instructions of the agent S in the program of Example 3.30*

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

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

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

◇

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

Let us consider first the tuple $(i^0_1, \ldots, i^0_n)$ of the initial program points of the agents $P_1, \ldots, P_n$ of P. The corresponding set of terms N_0 is characterized explicitly by a set of transitions of $\mathcal{A}$ using the states $q_{i^0_1 \ldots i^0_n}$ and $q^x_{i^0_1 \ldots i^0_n}$ (and possibly some auxiliary states used only for that purpose, see the Example 3.35 below). This set N_0 defines a unique *initial configuration* $(\{i^0_1, \ldots, i^0_n\}, \emptyset, N_0)$, which was mentioned when we discussed the reachable configurations, and N_0 is assumed to be part of the verification problem. Note that with this approach, it is possible to consider an infinite initial knowledge for the attacker. Moreover, the regular language N_0 is defined in a way that $cl(N_0) = N_0$, in order to conform to the above semantics.

Example 3.35 *The initial set of the attacker's knowledge* N_0 *which was mentioned in Example 3.31 is defined by the following transitions of* $\mathcal{A}$ *(for the sake of readability, we denote below the state* $q_{c_0 s_0}$ *by* q_0 *and the states* $q_{c_0 s_0} | q^x_{c_0 s_0} | q^y_{c_0 s_0}$ *by* q^{xy}_0*)*

$$\begin{array}{l} A \to q^{xy}_0, A \to q_A, C \to q^{xy}_0, S \to q^{xy}_0, \\ \mathsf{pk}(q_0) \to q^{xy}_0, \mathsf{sk}(q_A) \to q^{xy}_0, \mathsf{fst}(q_0) \to q^{xy}_0, \mathsf{snd}(q_0) \to q^{xy}_0, \mathsf{inc}(q_0) \to q^{xy}_0, \\ \mathsf{encrypt}(q_0, q_0) \to q^{xy}_0, \mathsf{decrypt}(q_0, q_0) \to q^{xy}_0, \mathsf{pair}(q_0, q_0) \to q^{xy}_0. \end{array}$$

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

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

symbols of Σ_{pub} at the top of the terms of its knowledge, *i.e.* the terms recognized in states $q_{i_1 \ldots i_n}$. It is expressed by transitions of the form:

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

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

Example 3.36 *For the program of Example 3.30, we have the following attacker's transitions (additionally to the one presented in Example 3.35 above)*

$\mathsf{pk}(q_{c_0 s_1}) \to q_{c_0 s_1} | q^x_{c_0 s_1} | q^y_{c_0 s_1}, \mathsf{pk}(q_{c_1 s_0}) \to q_{c_1 s_0} | q^x_{c_1 s_0} | q^y_{c_1 s_0},$
$\mathsf{pk}(q_{c_1 s_1}) \to q_{c_1 s_1} | q^x_{c_1 s_1} | q^y_{c_1 s_1}$ (*and idem for* $\mathsf{fst}, \mathsf{snd}, \mathsf{inc}$),
$\mathsf{encrypt}(q_{c_0 s_1}, q_{c_0 s_1}) \to q_{c_0 s_1} | q^x_{c_0 s_1} | q^y_{c_0 s_1},$
$\mathsf{encrypt}(q_{c_0 s_1}, q_{c_1 s_0}) \to q_{c_1 s_1} | q^x_{c_1 s_1} | q^y_{c_1 s_1}, \ldots$ (*and idem for* $\mathsf{decrypt}, \mathsf{pair}$).

◇

5.4 Verification of Security Properties

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

Example 3.37 *The protocol of Example 3.29 is supposed to ensure the authenticity of the message of S and also the confidentiality of* $\mathsf{inc}(N)$ *(for instance the value* $\mathsf{inc}(N)$ *can be supposed to be reused later as a key for symmetric encryption of a communication tunnel). However, both these properties can be attacked with a replay attack described in the following counter example.*

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

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

Actually, the attacker is not able to decrypt $\mathsf{encrypt}(N, \mathsf{pk}(S))$*, because he does not know the secret key* $\mathsf{sk}(S)$*. However, he is able to decrypt* $\mathsf{encrypt}(\mathsf{pair}(C, \mathsf{encrypt}(N, \mathsf{pk}(S))), \mathsf{sk}(C))$*, using the public key*

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

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

The existence of a confidentiality flaw like the one described in Example 3.37 is reducible to the problem of membership modulo $\mathcal{R}$ ($t \in \mathcal{R}^*\big(\mathcal{L}(\mathcal{A})\big)$, see Section 4.2), for the RTA $\mathcal{A}$ constructed above.

Example 3.38 *The confidentiality attack described in Example 3.37 occurs with two parallel sessions, involving 3 agents: 1 agent C playing the role of the client in the first session, and 2 agents playing the role of the server, respectively in the first and second session. The server agent in the first session is inactive. The role of the client in the second session is played by the attacker.*

We can recognize this attack by analyzing a program made of the 2 agents C and S defined in Example 3.30 plus the following second instance of a server

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

This agent has the same identity S as the first one in Example 3.30. Despite the renaming of the variable y into y' and of the program points (for technical convenience), this agent is the same as the one of Example 3.30. Let us construct the RTA $\mathcal{A}$ for this 3 agents as above. The states of $\mathcal{A}$ have the form $q_{c_i s_j s'_k}$ or $q^z_{c_i s_j s'_k}$ for $i, j, k \in \{0, 1\}$ and $z \in \{x, y, y'\}$.

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

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

This term t_1 corresponds to the message 1 *(of C) in Example 3.37. Using the transitions of the attacker, we obtain that*

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

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

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

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

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

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

Let us make a few remarks on the above analysis. The construction of two generic agents like in Example 3.30 is the specification of the protocol, written by the user. These agents represent the 2 roles (client and server in the protocol). Adding several instances of each agent in a program (to be verified) can be done automatically, just by variable and program point renaming as described above. The second part of the user specification is the construction of a TA recognizing the initial attacker's knowledge N_0, like in Example 3.35. The construction of the rest of the RTA, on the top of the agents and the TA for N_0 is automatic and the definition of the signature Σ and the TRS $\mathcal{R}$ are generic and independent of the protocol.

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

- a definition of the set of public symbols Σ_{pub},
- a user specification (as programs) of the roles of the protocol,
- the number and identities of the agents playing the different roles of the protocol (generic results like [CC04] can help),
- a finite representation of the initial knowledge of the attacker N_0, and
- a ground term whose confidentiality must be ensured.

Note that the verification technique described above is exact: it requires no approximation on the protocol and attacker model (as long as the protocol is a program in the syntax of Section 5.1). Hence, every attack reported is a real attack, all the confidentiality attacks are reported and

a negative answer is reported to the problem of membership modulo only if there exists no confidentiality attack, under the above hypotheses.

Authentication flaws like the one described in Example 3.37 can be reduced to the problem of emptiness of the intersection between the RTA $\mathcal{A}$ and a TA $\mathcal{E}$ (does $\mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{E}) = \emptyset$). The idea is to add some tags in the agent's messages, for instance marking the end of every agent. The tags are built with function symbols which are not public (hence they can only be added by the agents, with special instructions), but we can also consider other public functions that the attacker can apply to remove the tags (modulo some rules in $\mathcal{R}$ for that purpose). Then the TA $\mathcal{E}$ characterizes some traces corresponding to authentication errors. For instance, the authentication flaw described in Example 3.37 can be characterized by the fact that C has received a message 2 (in the first session) and entered program point c_1 (this is characterized by the presence of a tag T_c in the term) while S did not send it, and is still at program point s_0 (this is characterized by the absence of a tag above T_c in the term). The emptiness of the intersection of $\mathcal{L}(\mathcal{E})$ and $\mathcal{L}(\mathcal{A})$ (note that this language is not considered modulo $\mathcal{R}$ in this case) means that there is no authentication flaw.

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

6 Conclusion

In this chapter, we have studied the closure of RTA languages under term rewriting, and proposed an algorithm to decide that a given term belong to the closure of an RTA language by linear and invisibly pushdown TRS.

This class of tree automata is thought to be well suited for the automatic verification of some infinite state systems, and in particular for the verification of traces or equivalence properties of security protocols, using regular tree model checking like techniques. In this context, it would be interesting to extend the result of Theorem 3.27 to invisibly pushdown (non-linear) TRS, in order to handle axioms like $\mathsf{decrypt}(\mathsf{encrypt}(x, y), y) = x$. We are also interested about the symmetric form of the TRS of [CR07], whose rhs's are not single variables but have the form $f(x_1, \ldots, x_n)$.

In section 2.5, of chapter 2 we have also mentioned the possibility to define RTA as sets of Horn clauses and the use of general purpose first order theorem proving tools in order to decide properties like the ones related to the rewrite closure. Such an approach could be interesting for instance for the extension of RTA with equational tests like in [JRV08],

in order to be able to capture conditionals in the model of security protocols presented in Section 5.

Another use of TAGED in model checking has been recently studied in [CHK09]. We did not manage to fully study this work, but it seems that, even if the application is very close, the use of TAGED is very different than the one presented here.

In section 2.2, we add a visibly constraints to RTA that allows to determinize them, but not to complement them. In the context of the analysis and processing of trees containing data, like XML documents, it could be interesting to find a decidable class of tree automata with (dis)equality constraints closed under complement.

Chapter 4

Emptiness Decision for TAGED

As mentioned in section 2.1 the emptiness decision problem has been shown decidable for several subclasses of TAGED in [FTT08]. It is decidable for positive TAGED, for negative TAGED but also for vertically bounded TAGED which allow to mix equalities and disequalities up to a certain point. The decision procedure of the emptiness problem decidability for vbTAGED uses elegant intermediate results. However, we did not manage to enhance these techniques to have a positive result for a full TAGED class.

The decision procedure for negative TAGED is a reduction of the emptiness problem to a problem of verification of a system of set constraints. The proof of decidability of this result has been done in [CP94]: it translates the system of set constraints into a formula, and show that if there exists a model for the formula then there exists a finite one. Then, the rest of the proof shows how to produce an herbrand model (*i.e.* a model which is a term) from a generic model. To extend to the full class of TAGED, we need to add equality tests. Unfortunately they lead to more complex formula, where the proof of the existence of a Herbrand formula seems harder. On the other hand, the encoding of TAGED do not require all the logic formula used for the translation of set constraints.

Charatonik also proved in [Cha99] the decidability of emptiness problem for the class of DAG-automata, using similar techniques and notations as the ones in [CP94]. In fact, we will show an equivalence between t-dag automata and negative TAGED in the first section. The idea is that the maximal sharing property of t-dags ensure that two different nodes necessarily represents two different terms. Hence to provide the inequality $q \not\approx q'$ in a term it suffices to guarantee that, in the dag representation, no node represents a term recognized by both q and q'. Once again, this technique does not work with equality tests, because t-dag automata cannot enforce multiple use of a state in a run to correspond to only one node in the t-dag. However, we managed to

enhance the proof in [Cha99] to deal with equalities and be applied to the decidability of the emptiness problem of the full class of TAGED. That is what we will do in the second section of this chapter, by defining a pumping lemma on TAGED.

1 Negative TAGED and t-dag Automata are Equally Expressive

Negative TAGED recognize terms languages whereas t-dag automata recognize languages of t-dag representations of terms. As in chapter 2, we will here compare the expressiveness of these two models by considering the terms represented by the t-dags of the t-dag automaton language. The easy way is to build a negative TAGED recognizing the same language as a t-dag automaton.

Theorem 4.1 *Let $\mathcal{A} = \langle \Sigma, Q, F, \Delta \rangle$ be a t-dag automaton. There exists a negative TAGED $\mathcal{B}$ such that, for all $t \in \mathcal{T}(\Sigma)$, t is recognized by $\mathcal{B}$ if and only if $dag(t)$ is recognized by $\mathcal{A}$.*

proof. Let us define $\mathcal{B} = \langle \Sigma, Q, F, \Delta, \not\approx \rangle$ where, for all $q_1, q_2 \in Q, q_1 \neq q_2$ implies that $q_1 \not\approx q_2$. Let $G = dag(t)$ be a t-dag recognized by $\mathcal{A}$ and let r be a successful run of on G. Obviously, r' defined as $\forall p \in \mathcal{P}os(t), r'(p) = r(G(p))$ is an accepting run of the underlying tree automaton $ta(B)$ on t, since the states, the final states, and the transition rules are the same. For all $p_1, p_2 \in \mathcal{P}os(t)$ such that $r'(p_1) = q_1 \neq q_2 = r'(p_2)$, we have that $r(G(p_1)) \neq r(G(p_2))$, hence the nodes $G(p_1)$ and $G(p_2)$ are distinct. Since G is a t-dag, $G|_{p_1}$ and $G|_{p_2}$ are not isomorphic, which implies that $t|_{p_1} \neq t|_{p_2}$. The disequalities $q_1 \not\approx q_2$ are respected, and r' is a successful run of $\mathcal{B}$ on t.

Reciprocally, let t be recognized by a run r' of $\mathcal{B}$. Since for all $p_1, p_2 \in \mathcal{P}os(t)$ such that $r(p_1) \neq r(p_2)$, the disequality constraint $\not\approx$ implies that $t_{p_1} \neq t_{p_2}$, we have that for all subterm t' of t there exists a unique state $q_{t'}$ such that, for all position p, if $t|_p = t'$ then $r'(p) = q'$. Considering the construction of proposition 1.5, where each node of $dag(t)$ corresponds to a subterm of t, we build the run r such that, for all subterm t' of t, $r(t') = q_{t'}$. Since the states, the final states, and the transition rules are the same in $\mathcal{A}$ and $\mathcal{B}$, r is a successful run of $\mathcal{A}$ on $dag(t)$. □

The other way, encoding a negative TAGED into a t-dag automaton, is a bit more complex, and may involve an exponential blowup of the size of the automaton.

Theorem 4.2 *Let $\mathcal{A} = \langle \Sigma, Q, F, \Delta, \not\approx \rangle$ a negative TAGED.*

There exists a t-dag automaton $\mathcal{B}$ such that for all term $t \in \mathcal{T}(\Sigma)$, t is recognized by $\mathcal{A}$ if and only if $dag(t)$ is recognized by $\mathcal{B}$.

proof. Let $P = 2^Q \setminus \{Q' \subseteq Q \mid \exists q_1, q_2 \in Q', q_1 \not\approx q_2\}$. Let Δ' be the set of transition rules of the form $\{f(Q_1, \ldots, Q_n) \to Q'$ where $f \in \Sigma_n, Q_1, \ldots Q_n, Q' \in P$ and for all $q \in Q'$, there exists $q_1 \in Q_1, \ldots, q_n \in Q_n$ such that $f(q_1, \ldots, q_n) \to q \in \Delta\}$ Let $F' = \{\{q_f\} \mid q_f \in F\}$ We define the t-dag automaton $\mathcal{B} = \langle \Sigma, P, F', \Delta' \rangle$.

$\Rightarrow$: Let t be a term recognized by $\mathcal{A}$ and let r be a successful run of $\mathcal{A}$ on t. Let us define a mapping r' from the nodes of $G = dag(t)$ to 2^Q. Using the canonical description of the nodes of G of being the set of subterms of t, we define $r'(t') = \{q \mid \exists p \in \mathcal{P}os(t), t|_p = t', r(p) = q\}$ for all subterm t' of t. Since, there is no subterm t' of t such that $\exists p_1, p_2 \in \mathcal{P}os(t), t|_{p_1} = t|_{p_2} = t'$ and $r(p_1) \not\approx r(p_2)$, then, there is no subterm t' of t such that $r'(t')$ contains two different states q_1 and q_2 such that $q_1 \not\approx q_2$. Hence, for all subterm t' of t, $r'(t') \in P$. Moreover, since r is an accepting run, $r'(t)$ is a singleton $\{q_f\}$ where $q_f \in F$, so $r'(t) \in F'$.

Let us show that r' is compatible with Δ'. Let t' be a subterm of t such that $t'(\varepsilon) \in \Sigma_n$. If $n = 0$, then, by construction, for all $q \in r'(t')$, there exists some leaf position p of t such that $r(p) = q$, hence, there exists $t(p) \to r(p) \in \Delta$. So $t(p) \to r'(t') \in \Delta'$.

If $n > 0$, then for all $q \in r'(t')$, there exists a position p such that $t|_p = t'$ and $r(p) = q$. Hence, there exists a transition rule $t(p)(q_1, \ldots, q_n) \to \in \Delta$, such that, for all i, $1 \leq i \leq n$, $r(p.i) = q_i$ and $t|_{p.i} = t'|_i$. Hence, for all i, $1 \leq i \leq n$, $q_i \in r'(t'|_i)$, so we have that $t(p)(r'(t'|_1), \ldots, r'(t'|_n)) \to r'(p) \in \Delta'$. So r' is compatible with Δ' and is a successful run of $\mathcal{B}$ on G.

$\Leftarrow$: Let $G = dag(t)$ be a t-dag recognized by $\mathcal{B}$, and r' a successful run of $\mathcal{B}$ on G. Let us define by induction a mapping r from $\mathcal{P}os(t)$ to Q, such that for all $p \in \mathcal{P}os(t)$, $r(p) \in r'(t|_p)$, and which is compatible with Δ. We start with $r(\varepsilon) = q_f$ where $r' = \{q_f\}$, q_f being a state of F. By induction, for all position $p \in \mathcal{P}os(t)$, such that $t(p) \in \Sigma_n$, $n > 0$, and $r(p) = q \in r'(t|_p)$, by construction of Δ' there exists $q_1 \in r'(t|_{p.1}), \ldots, q_n \in r'(t|_{p.n})$ such that $t(p)(q_1, \ldots, q_n) \to q \in \Delta$. We define $r(p.i) = q_i$ for all $i, 1 \leq i \leq n$, and the position p of r is compatible with Δ. And for all position $p \in \mathcal{P}os(t)$ such that $t(p) \in \Sigma_0$ and $r(p) \in r'(t|_p)$, by construction of Δ' there exists $t(p) \to r(p) \in \Delta$. The mapping r is then compatible with Δ, and $r(\varepsilon) \in F$, so r is a successful run of $ta(\mathcal{A})$.

In order to prove that r is a successful run of $\mathcal{A}$, we only have to prove that the relation $\not\approx$ is respected. For all $q_1, q_2 \in Q$ such that $q_1 \not\approx q_2$, there is no set $Q' \subseteq Q, Q' \in P$ such that Q' contains both q_1 and q_2. So, by construction of r, there is no subterm t' of t, such that there exists two positions $p_1, p_2 \in \mathcal{P}os(t)$ with $r(p_1) = q_1, r(p_2) = q_2$ and $t|_{p_1} = t|_{p_2} = t'$. So $\not\approx$ is respected by r and t, and r is a successful run of $\mathcal{A}$ on t $\square$

2 Deciding Emptiness of TAGED

We use here the same vocabulary and the similar constructions as the ones introduced by Charatonik in [Cha99] for the emptiness decision for t-dag automata. The general idea is to prove that for each term recognized by a TAGED, one can build another term of bounded height that is also recognized. In order to do this, we will need to work on the t-dag representation of the terms to preserve the (dis)equality properties required by the TAGED. The main difference with the pumping for t-dag automata is that, here, the t-dag is labelled by sets of states, like in the proof of Theorem 4.2, since each node corresponds to several positions in the term recognized by the TAGED, instead of states, since t-dag automata directly runs on t-dags.

We first introduce an example that we will refer to all along this section.

Example 4.3 *Let* $\mathcal{A} = \langle \Sigma, Q, F, \Delta, \approx_{\mathcal{A}}, \not\approx_{\mathcal{A}} \rangle$ *be the TAGED such that*

- $\Sigma = \{a : 0, g : 1, f : 2, h : 3\}$;
- $Q = \{q_0, q_1, q_2, q_3, q_4, q_f\}$;
- $F = \{q_f\}$;
- $\Delta = \{a \to q_0, g(q_0) \to q_0, g(q_0) \to q_1$
  $f(q_1, q_0) \to q_2,$
  $g(q_2) \to q_3, g(q_3) \to q_3$
  $g(q_2) \to q_4, h(q_2, q_3, q_4) \to q_f\}$
- $q_1 \approx_{\mathcal{A}} q_1$;
- $q_3 \not\approx_{\mathcal{A}} q_4$

Let $t = h(f(g(g(a)), g(g(a))), g(g(f(g(g(a)), g(a)))), g(f(g(g(a)), a)))$. *The figure 4.1 gives a graphic representation of* t, *and of a successful run of* $\mathcal{A}$ *on* t. *Note that* $g(g(a))$ *is the only term recognized (several times) by* q_1, *and that the terms recognized respectively by* q_3 *and* q_4 *are different.*

2.1 Mapping Nodes to a Set of States

We begin by defining a mapping from the set of nodes of a t-dag to the subsets of states of a TAGED, according to a given run.

Definition 4.4 *Let* t *be a term recognized by an accepting run* r *of* $\mathcal{A}$ *Let* $G = (V, succ, \lambda) = dag(t)$. *We define the function* $states_r : V \to 2^Q$, *the following way:* $states_r(v) = \{q \mid \exists p \in \mathcal{P}os(t), t|_p = term(G|_v) \wedge r(p) = q\}$. *For all node* v *and for all set* $P \subseteq Q$, *if* $states_r(v) = P$, *we say that* v provides P.

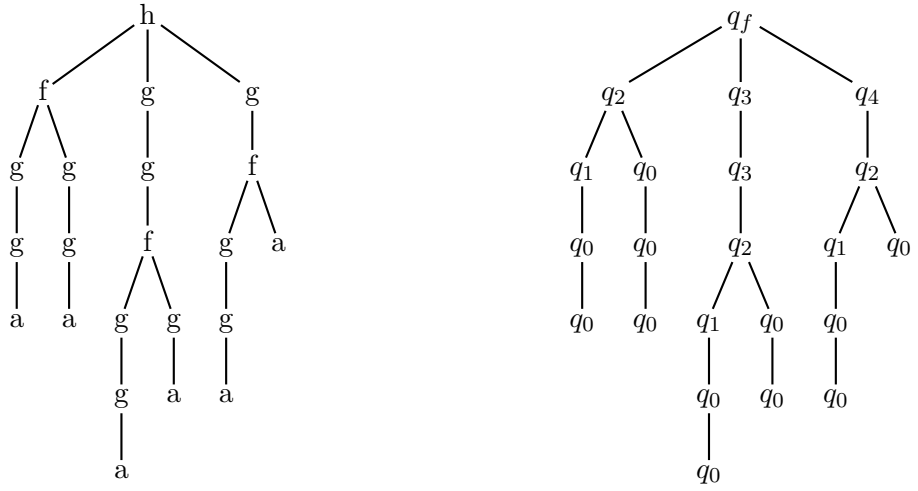

Figure 4.1: The term and the run of example 4.3

Intuitively, $states_r(v)$ is, in the run r, the set of states recognizing the term represented by G_v.

The figure 4.2 shows the t-dag representation G of the term t of example 4.3, where we write at each node v, the couple $\lambda_G(v) : states_r(v)$.

We will present several transformations on t-dags. Given a t-dag $G = (V, succ, \lambda) = dag(t)$ and a run r of a TAGED on t, we will define several subgraphs or t-dag whose set of nodes will be a subset of V. However, we will always manipulate $state_r$ as it is defined on all V (and hence, on all subset of V). In particular, the following properties will still hold for every node of V, even if we consider these nodes in the set of nodes of a smaller t-dag.

Proposition 4.5 *Given a t-dag G representing a term t and an accepting run r of a TAGED $\mathcal{A}$ on t, the following properties hold:*

1. *let u be the root of G, then $states_r(u) = \{q_f\}$ for some $q_f \in F$*
2. *if $q_1 \approx_{\mathcal{A}} q_2$ there exists at most one node $v \in V$ such that q_1 and/or $q_2 \in states_r(v)$*
3. *if $q_1 \not\approx_{\mathcal{A}} q_2$ there is no node $v \in V$ such that $states_r(v)$ contains both q_1 and q_2*
4. *$\forall v \in V$ labeled by $f \in \Sigma_n$, $\forall q \in states_r(v)$, $\exists f(q_1, \ldots, q_n) \to q \in \Delta$ such that $\forall i, 1 \leq i, \leq n$, $q_i \in states_r(succ(i, v))$*

proof. 1 - r is an accepting run so there exists some $q_f \in F$ such that $r(\varepsilon) = q_f$. By definition of $states_r$, since $term_G(u) = t = t|_\varepsilon$, we have that $states_r(u)$ is the singleton $\{q_f\}$.

2 - If neither q_1 nor q_2 occur in r, then by definition of $states_r$ there is no node v such that $states_r(q)$ contains q_1 or q_2, and we are done. Otherwise, either q_1 or q_2 occurs at least once in r. Wlog, assume that q_1

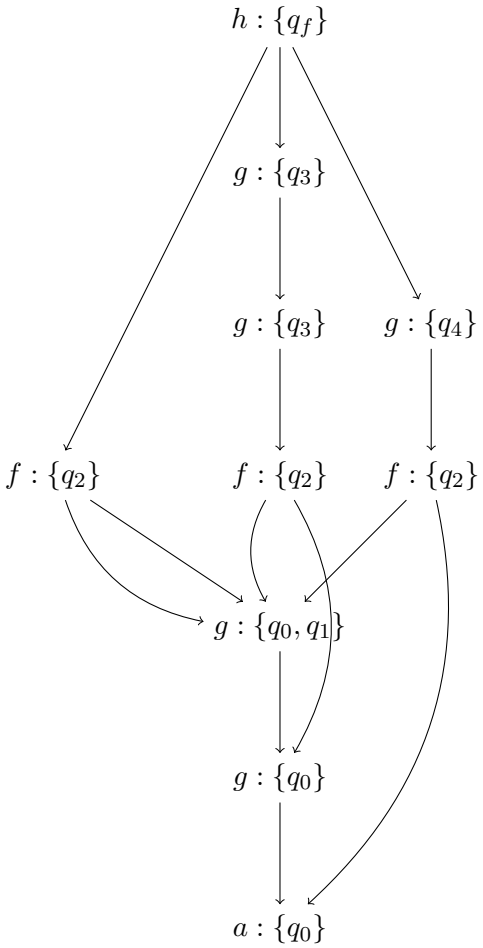

Figure 4.2: The t-dag representation of the term of example 4.3 with the set $states_r(v)$ for each node v.

occurs in $states_r(v)$ for some node v. By definition of $states_r$, there exists some position $p \in \mathcal{P}os(t)$ such that $r(p) = q_1$, and $term_G(v) = t|_p$. For every node v' of G such that either q_1 or q_2 belongs to $states_r(v')$, there exists a position $p' \in \mathcal{P}os(t)$ such that $r(p') = q_1$ or q_2. Since r satisfies the equality constraints and since $\approx_{\mathcal{A}}$ is reflexive and symmetric, we have that $t|_p = t|_{p'}$. Hence $term_G(v) = term_G(v')$, so $v = v'$. v is therefore the unique node of G such that $states_r(v)$ may contain q_1 and q_2.

3 - If r contains only one of the states q_1 or q_2, or none of them, then the property is obviously satisfied. So assume there is at least one occurrence of each of them in r. Let v_1 be a node of G such that $q_1 \in states_r(v_1)$. By definition of $states_r$ there is a position $p_1 \in \mathcal{P}os(t)$ such that $term_G(v_1) = t|_{p_1}$ and $r(p_1) = q_1$. For every node v_2 of G such that $q_2 \in states_r(v_2)$, there exists $p_2 \in \mathcal{P}os(t)$ such that $r(p_2) = q_2$ and $term_G(v_2) = t|_{p_2}$. Since r satisfies the disequality condition, we have that $t|_{p_1} \neq t|_{p_2}$, so $term_G(v_1) \neq term_G(v_2)$. Hence $v_1 \neq v_2$, and $q_2 \notin states_r(v_1)$. By a symmetric reasoning, we have that for every node v_2 such that $q_2 \in states_r(v_2)$, $q_1 \notin states_r(v_2)$. So there is no node v such that $states_r(v)$ contains both q_1 and q_2.

4 - Let v be a node of G labeled by $f \in \Sigma_n$, and let q be a state in $states_r(v)$. By definition of $states_r$, there exists a position $p \in \mathcal{P}os(t)$ such that $r(p) = q$ and $term_G(v) = t|_p$. By definition of a run, we know that there exist a rule $f(q_1, \ldots, q_n) \to q \in \Delta$ where $q_i = r(p.i)$ for all i. For all i, $1 \leq i \leq n$, we have that $term_G(succ(i, v)) = t|_{p.i}$. Hence, by definition of $states_r$, $q_i = r(p.i) \in states_r(v_i)$ for all i. The rule $f(q_1, \ldots, q_n) \to q$ satisfies the wanted condition. □

2.2 Definitions and Notations

In order to prove the existence of a small enough term for every non-empty TAGED language, we will need to manipulate some new objects that we introduce here.

Let $G = (V, succ, \lambda)$ be a t-dag on a signature Σ, and some linear ordering $<$ on function symbols of Σ. By induction, we define $\prec$ on V to extend the "lies below" partial order into a linear ordering the following way: let v_1, v_2 be in V, $v_1 \prec v_2$ if and only if:

- v_1 lies below v_2; or
- v_1 and v_2 have the same height, and $\lambda(v_1) < \lambda(v_2)$; or
- v_1 and v_2 are of same height, $\lambda(v_1) = \lambda(v_2) \in \Sigma_n$, and $\langle succ(1, v_1), \ldots, succ(n, v_1)\rangle$ is lower than $\langle succ(1, v_2), \ldots, succ(n, v_2)\rangle$ *wrt* the lexicographic extension of $\prec$.

Let us define the order $\sqsubset$ on tuples of nodes of G:
$\langle v_1^1, \ldots, v_1^k\rangle \sqsubset \langle v_2^1, \ldots, v_2^m\rangle$ if and only if

- $\max_{1\leq i\leq k}\{h(v_1^i)\} < \max_{1\leq j\leq m}\{h(v_2^j)\}$ **or**
- 1. $\max_{1\leq i\leq k}\{h(v_1^i)\} = \max_{1\leq i\leq m}\{h(v_2^j)\}$ and
  2. $\langle v_1^1, \ldots v_1^k\rangle$ is strictly smaller than $\langle v_2^1, \ldots v_2^m\rangle$ *wrt* the lexicographic extension of $\prec$.

Note that since $\prec$ is a linear ordering on nodes of G, $\sqsubset$ is a linear ordering on tuples of nodes of G.

A *pointer* is a pair (P, i), where P is a subset of Q and i is a number. We write $P[i]$ instead of (P, i). Given a t-dag G representing a term t, r an accepting run of a TAGED $\mathcal{A}$ on t, we say that the pointer $P[i]$ points to the i-th (according to the order $\prec$) node v such that $states_r(v) = P$. By index of a given node v (in symbols, $ind_r(v)$) we mean the number i such that the pointer $states_r(v)[i]$ points to v. A *transition* with pointers, is an expression of the form $f(P_1[i_1], \ldots, P_{k-1}[i_{k-1}], P_k, P_{k+1}[i_{k+1}], \ldots, P_n[i_n]) \to P$, where $P, P_1, \ldots, P_n \subseteq Q_A$ We say that such a transition is *compatible* with some node v of G if

- k is the main position of v;
- $P = states_r(v)$;
- for all $i, 1 \leq i \leq n$, $P_i = states_r(succ(i, v))$.

2.3 Building a Skeleton

Our goal here will be to define a directed acyclic graph, whose node set is a subset of the node set of G, from which we can infer a t-dag representing a shorter term, that will be recognized by the same TAGED. The construction will first keep all the main path of the roof of the t-dag. But to have a smaller t-dag, it will try to use as less nodes as possibles for every secondary position. However, we cannot use a single node for each set of states in $states_r(V_G)$, since it may broke some (dis)equalities. We define more formally this situation.

Definition 4.6 *Let G be a t-dag representing a term t, and r be a successful run of a TAGED $\mathcal{A}$ on t. Let $P_1, \ldots, P_n \subseteq Q$ and v a node of G. We say that there is a* fork *of degree m at the node v and at position k wrt $P_1, \ldots, P_n$ if there exists m nodes $v_1, \ldots v_m$ in G such that*

- $\lambda_G(v_i) = \cdots = \lambda_G(v_m) = f \in \Sigma_n$;
- *for all $i, 1 \leq i \leq m, k$ is the main position of v_i;*
- *for all $i, 1 \leq i \leq m$, $succ(k, v_i) = v$;*
- *and for all $i, j, 1 \leq i \leq m, 1 \leq j \leq n$, $states_r(succ(j, v_i)) = P_j$.*

Intuitively, a fork of degree m is a situation where m nodes are labeled by the same function symbol, have the same main successor, and all their successors represent terms that are respectively recognized by the same states by $\mathcal{A}$ in r. Since those m nodes represent m different terms, if we replace all their secondary successors by the same nodes, we may break some disequality constraints. So to build our skeleton, we will need to generate m different sequences of nodes providing the sets $P_1, \ldots, P_n$. Note that a fork may be of degree 1. Here is the construction of our skeleton, respecting these observations.

Definition 4.7 *Given a t-dag G representing t, a TAGED $\mathcal{A}$ and an accepting run r of $\mathcal{A}$ on t, we want to build a directed acyclic S^r_G whose node set is a subset of V_G, and such that each node is labeled by a transition with pointers to some other nodes below.*

We apply the following procedure that marks nodes of V_G and define a labeling of each marked node by a transition rule with pointers:

1. *Let u be the root of G we mark u and all the nodes in its main path.*

2. *For $h = h(G)$ to 1 for each fork of degree m at the node v' and position k wrt $P_1, \ldots, P_n$, we define*

$$\lambda_S(v_i) = f(P_1[ind_r(v_i^{1\prime})], \ldots P_{k-1}[ind_r(v_i^{k-1\prime})], P_k, P_{k+1}[ind_r(v_i^{k+1\prime})],$$

$$\ldots, P_n[ind_r(v_k^{n\prime})]) \rightarrow states_r(v_i)$$

for all $i, 1 \leq i \leq m$, where $\langle v_1^{1\prime}, \ldots v_1^{k-1\prime}, v_1^{k+1\prime}, \ldots, v_1^{n\prime}\rangle \sqsubset \ldots,$ $\sqsubset \langle v_m^{1\prime}, \ldots, v_m^{k-1\prime}, v_m^{k+1\prime}, \ldots, v_m^{n\prime}\rangle$ are the first (according to the linear ordering $\sqsubset$) m sequences of nodes of G providing the sets $P_1, \ldots, P_{k-1}, P_{k+1}, P_n$. We say that the fork defines those labels. Then we mark the nodes pointed by all the pointers $P_j[ind_r(v_i^{j\prime})]$ and all the nodes on their main path. We say that the fork marked those nodes.

3. *for each marked leaf v of G, we define $\lambda_S(v) = \lambda_G(v) \rightarrow states_r(v)$*

We define V_S as the set of nodes of V_G that have been marked during the procedure. We define the successor relation $succ_S$ over V_S by $v' = succ_S(v)$ if and only if v' is the main successor of v in G.

We define the direct acyclic labeled graph $S^r_G = \langle V_S, succ_S, \lambda_S\rangle$.

The figure 4.3 shows the result of this construction applied to the t-dag of figure 4.2. Note that there is a fork of degree 2 involving the two nodes labeled by f.

We will show that this construction is well-defined and that it satisfies several useful properties. First, we will prove the following lemma on the construction.

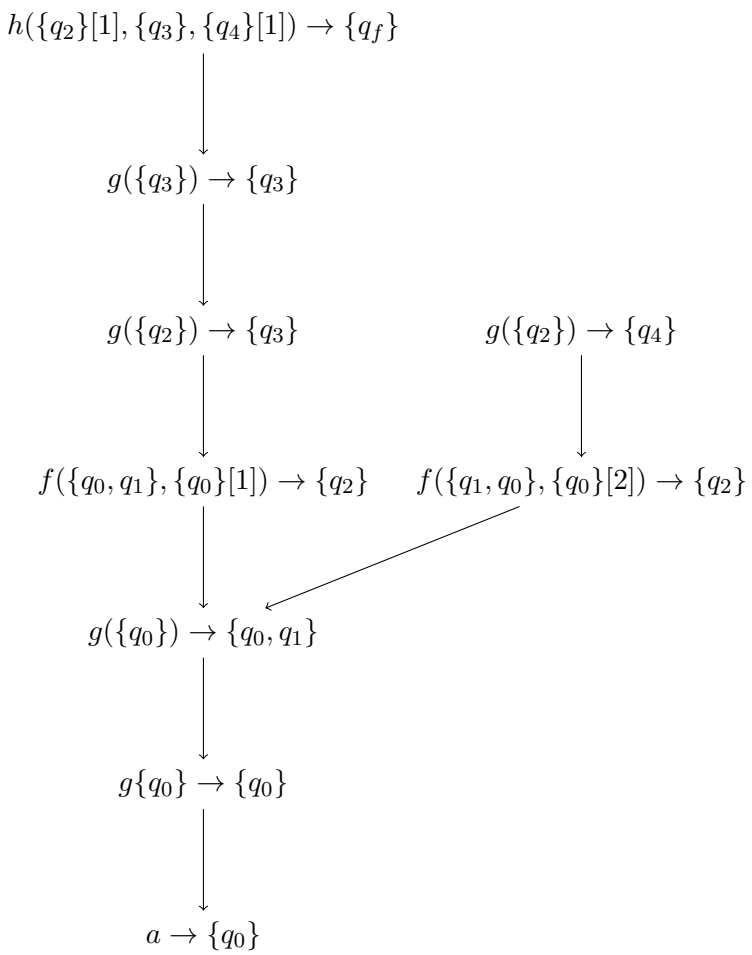

Figure 4.3: The graph S^r_G where G is the t-dag of figure 4.2

Lemma 4.8 *Let G be a t-dag representing a term t, and r a successful run of a TAGED $\mathcal{A}$ on t. Then, given a fork of degree m occurring in the procedure of definition 4.7, there is at most $m-1$ nodes of indexes greater than 1 used in the transition rules defined by this fork, and the maximal index used in such a transition is m.*

proof. Given a fork of degree m at a marked node v' at position k *wrt* $P_1, \ldots, P_n$ that labels m marked nodes $v_1 \ldots v_m$ with $\lambda_S(v_j) = f(P_1[ind_r(v_j^{1\prime})], \ldots, P_k, \ldots P_n[ind_r(v_j^{n\prime})])$. Let $k1, \ldots, k_n$ the numbers of different nodes $v_j^{1\prime}, \ldots, v_j^{n\prime}$ whose indexes are used in these transitions. We can construct $k_1.k_2.\ldots.k_n$ different sequences of the form $\langle v_1{}', \ldots, v_n{}' \rangle$. As we need m different sequences, so we have that $k_1.\ldots.k_n \geq m$. Since we take the m first sequences according to $\sqsubset$, and that $\langle P_1[1], \ldots, P_n[1] \rangle$ is the lowest sequence, there is a node of index 1 for each of the subsets of states, so there are $(k_1-1)+\cdots+(k_n-1)$ nodes of index greater than 1. And since $k_1.\ldots.k_n \geq (k_1-1)+\cdots+(k_n-1)+1$, it is sufficient to have $(k_1-1)+\cdots+(k_n-1) \geq m-1$. Therefore, a fork of degree m requires at most $m-1$ nodes of index greater than 1. In particular, the maximal index used in a transition rule defined by a fork of degree m is m. □

Theorem 4.9 *Given a t-dag G representing t, a TAGED $\mathcal{A}$ and an accepting run r of $\mathcal{A}$ on t, the directed acyclic labeled graph S_G^r is well-defined. Moreover, for all node $v \in V_S$ the height of v in S_G^r is equals to its height in G, all the indexes of the pointers in the transition $\lambda_S(v)$ are lower or equal to $2^{|Q|}$, and the pointers point to nodes below v.*

proof. First, when a node is marked, in either the initial step, or during the loop, we also mark all the nodes in its main path. Hence, since the successor of a node v in S_G^r is its main successor in G, we have that its height is equal in both graphs.

We will then prove the well-definition of V_G and $\lambda_S(v)$, and the property on the indexes by proving some invariants in the procedure.

For each height $h, 1 \leq h \leq h(G)$, at a given moment of the procedure, we will define the width $w(h)$ as the number of marked nodes of G of height h. We will define $mil^1_=(h)$ as the number of nodes $v \in V_G$ of index 1 (v is of index 1 if and only if $v = states_r(v)[1]$) of height h, and $mil^1_<(h)$ as the number of nodes $v \in V_G$ of index 1 and of height strictly lower than h. Note that for all $h, 1 \leq h < h(G)$, $mil^1_<(h+1) = mil^1_<(h) + mil^1_=(h)$. We define also $mil^{\geq}_<(h)$ as the number of nodes of index greater than 1 and of height lower than h that are pointed by some transition labelling of some marked node of height greater or equals to h, and $mil^{\geq}_=(h)$ as the number of nodes of index greater than 1 and of height h that are pointed by some transition labelling some marked node of height greater than h.

For each fork of degree m, we call *extra degree* the number $m-1$. For each height $h, 1 \leq h \leq h(G)$, we denote $ex(h)$ the sums of the extra degrees of all the forks at the marked nodes of height $h-1$.

For each height $h, 1 \leq h \leq h(G)$, we will prove the following invariants on the procedure.

1. the step h define $\lambda_S(v)$ for all marked node v of height h;
2. The step h only mark nodes of height lower than h;
3. at the end of the step h the following inequations hold:
   - $w(h) + mil^1_<(h) + mil^{\geq}_<(h) + ex(h) \leq 2^{|Q|} + ex(h)$.
   - $w(h) + mil^1_<(h) \leq 2^{|Q|}$.
4. for all marked node v, $\lambda_S(v)$ points only to nodes of index lower or equal to $2^{|Q|}$;

First, note that to prove the invariant 2, it is enough to prove that the transitions with pointers that are defined for the marked nodes of height h, all the pointers point to node of height lower than h.

Let us prove these invariants recursively. For the first step, $h = h(G)$, $w(h) = 1$ since the root u of G is the only node of height h, and it is marked. There is only one fork of degree 1 at the main successor u' of u which is at this moment the only marked node of height $h-1$. Since the fork is of degree 1, we have that $ex(h) = 0$, and we only take the indexes of the lowest tuple of nodes. The labelling $\lambda_S(u)$ is hence defined and is of the form

$$f(P_1[1], \ldots P_{k-1}[1], P_k, P_{k+1}[1], P_n[1])$$

where $f = \lambda_G(u) \in \Sigma_n$, for all $i, 1 \leq i \leq n, P_i = states_r(G(i))$, and k is the main position of u. Since the node pointed belong to G they are necessarily of height lower than u. This transition points only to nodes of indexes $1 \leq 2^{|Q|}$, and hence $mil^{\geq}_<(h) = 0$. Since there is at most one node of index 1 for each set of states $P \subseteq Q$, and since we assume that $states_r(u)$ is the only set of $states_r(V_G)$ containing a final state, we have that $mil^1_<(h) \leq 2^{|Q|} - 1$. Hence at the end of the first step, we have that $w(h) + mil^1_<(h) \leq 2^{|Q|}$ and $w(h) + mil^1_<(h) + mil^{\geq}_<(h) \leq 2^{|Q|} + ex(h)$.

Assume now that the invariants are true at the step of height $h+1$, $h > 1$. We will prove the correctness of the invariants one by one

Invariant 1: For every marked node v of height h, its main successor v' is also marked and is of height $h-1$. Therefore, v will be involved in a fork at v' and will then be labeled.

Invariant 2: Let us now prove that each node pointed by a transition rule defined at this step is of height lower than h. For each fork of degree m at some marked node v' of height $h-1$, at position k and *wrt* $P_1, \ldots, P_n$, where $\langle v_1^{1\prime}, \ldots v_1^{k-1\prime}, v_1^{k+1\prime}, \ldots, v_1^{n\prime}\rangle, \ldots, \langle v_m^{1\,\prime}, \ldots, v_m^{k-1\prime}, v_m^{k+1\prime}, \ldots, v_m^{n\,\prime}\rangle$ are the first (according to the linear ordering $\sqsubset$) m sequences of nodes providing $P_1, \ldots, P_{k-1}, P_{k+1}, \ldots, P_n$.

We have to show that, for all $i, 1 \leq i \leq m$, and for all $j, 1 \leq j \leq n, j \neq k, h(v_i^j)' \leq h$. Let $v_i^j = succ_G(j, v_i)$. The sequences $\langle v_1^1, \ldots v_1^{k-1}, v_1^{k+1}, \ldots, v_1^n \rangle, \ldots, \langle v_m^1, \ldots, v_m^{k-1}, v_m^{k+1}, \ldots, v_m^n \rangle$ are all distinct and are such that for all $i, 1 \leq i \leq n$ and all $j, 1 \leq j \leq m, j \neq k$, $states_r(v_j^{i\prime} = P_i)$. Let j be the number such that $\langle v_j^1, \ldots v_j^{k-1}, v_j^{k+1}, \ldots, v_j^n \rangle$ is the greatest of those sequences *wrt* $\sqsubseteq$. For all $i, 1 \leq i \leq m$ we have that $\langle v_i^{1\prime}, \ldots, v_i^{k-1\prime}, v_i^{k+1\prime}, \ldots, v_i^{n\prime} \rangle \sqsubseteq \langle v_j^1, \ldots, v_j^{k-1}, v_j^{k+1}, \ldots, v_j^n \rangle$, which implies that $\max_{1 \leq s \leq n, s \neq k}\{h(v_i^{s\prime})\} \leq \max_{1 \leq s \leq n, s \neq k}\{h(v_j^s)\}$, and since all v_i^j are of height less or equal to the main successor of v_i, they are all of height lower or equal to h.

Invariant 3: A marked node of height h at which occurs a fork of degree m, is the common main successor of m different marked nodes of height $h+1$, so the number of marked nodes that are main successors of marked nodes of height $h+1$ is $w(h+1) - ex(h+1)$. Since a node of height h is either a main successor of m nodes for some m, or a node of index 1, or a node of index greater than 1 pointed by some node of height $h > 1$, we have that $w(h) \leq w(h+1) - ex(h+1) + mil_{=}^{1}(h) + mil_{=}^{\geq}(h)$. Since $mil_{<}^{1}(h) = mil_{<}^{1}(h+1) - mil_{=}^{1}(h)$, we have that $w(h) + mil_{<}^{1}(h) \leq w(h+1) + mil_{<}^{1}(h+1) - ex(h+1) + mil_{=}^{\geq}(h)$. By recurrence hypothesis, we have that $w(h+1) + mil_{<}^{1}(h+1) + mil_{<}^{\geq}(h+1) \leq 2^{|Q|} + ex(h+1)$, from which we can derive $w(h+1) + mil_{<}^{1}(h+1) - ex(h+1) \leq 2^{|Q|} - mil_{<}^{\geq}(h+1)$. Therefore, $w(h) + mil_{<}^{1}(h) \leq 2^{|Q|} - mil_{<}^{\geq}(h+1) + mil_{=}^{\geq}(h)$. By definition, every node counted in $mil_{=}^{\geq}(h)$ is also counted in $mil_{<}^{\geq}(h+1)$, so we clearly have $mil_{=}^{\geq}(h) \leq mil_{<}^{\geq}(h+1)$. Hence, the invariant $w(h) + mil_{<}^{1}(h) \leq 2^{|Q|}$ is respected.

For each fork of degree m at a marked node of height $h-1$, by lemma 4.8, the number of nodes of index greater than 1 pointed by some pointer in a transition defined by this fork is at most $m-1$. Hence, the total number of nodes of index greater than 1 and of height lower than h pointed by some pointer at some transition labelling a node of height h is bounded by $ex(h)$. So we have the following inequality: $mil_{<}^{\geq}(h) \leq mil_{<}^{\geq}(h+1) - mil_{=}^{\geq}(h) + ex(h)$. Using the previously proved inequality, it follows directly that the invariant $w(h) + mil_{<}^{1}(h) + mil_{<}^{\geq}(h) = 2^{|Q|} + ex(h)$ is also respected.

Invariant 4: from the previous invariant, we can deduce $w(h) \leq 2^{|Q|}$, so there cannot be a fork of degree greater than $w(h)$ at a marked node of height $h-1$. Then, by lemma 4.8, we have that the maximal index of a node pointed by a labelling of a marked node of height h is $2^{|Q|}$. □

The use of the order $\sqsubset$ ensures that every pointer points to nodes below. In [Cha99], the order used is the lexicographic extension of $\preceq$ without consideration of the maximal height amongst the nodes of the sequences. Hence, assume that at some fork of arity 2, we need two distinct sequences $\langle v_1^1, \ldots, v_n^1$ and $\langle v_1^2, \ldots, v_2^n \rangle$. Then, if they are two different nodes $v_n \prec v_n{}'$ in G which are the two lowest nodes such that

$state_r(v_n) = state_r(v_n{}') = P_n$, the two first sequences according to the lexicographic extension of $\prec$ will be $\langle v_1, v_2, \dots v_n \rangle$ and $\langle v_1, v_2, \dots v_n{}' \rangle$. But nothing can guarantee that $v_n{}'$ lies below the fork.

Intuitively, S^r_G gives a reduced construction of a t-dag representation of a term recognized by the same TAGED. It can be seen as a *skeleton*, since it only contains the longest paths of the t-dag, but give the hints to build the whole t-dag. The t-dag that we can infer from S^r_G will have less states than G since we bound the number of nodes used at secondary positions. However, we are not yet able to bound the total number of states that such a t-dag will have. We will need to operate some transformations on S^r_G. To be sure that we keep the properties that allow one to build a t-dag, we will regroup these properties under the name of *skeleton*.

Definition 4.10 *Given a t-dag $G = \langle V_G, succ_G, \lambda_G \rangle$ representing a term t, given a TAGED $\mathcal{A}$, and an accepting run r of $\mathcal{A}$ on t, a* semi-skeleton *of G and r is a directed acyclic labeled graph $S = (V_S, succ_S, \lambda_S)$ such that*

- *$V_S \subseteq V_G$,*
- *V_S contains the root of G,*
- *for each node v in V_S, $\lambda_S(v)$ is a transition with pointers, compatible with v,*
- *each node v such that $\lambda_S(v) = f(P_1[i_1], \dots, P_k, \dots, P_n[i_n]) \to P$ has a unique immediate successor $succ_S(v)$ such that $states_r succ_S(v) = P_k$,*
- *for all pointer $P[i]$ occurring in a transition labelling a node v such that $P[i]$ points to some node v_0 in V_G, v_0 is in V_S.*

Moreover, a semi-skeleton S of G is a skeleton *if for all pointer $P[i]$ used in a transition labelling a node v such that $P[i]$ points to some node v_0, v_0 lies below v in S.*

A node v of S is a milestone *if it is either the root u of G or if it is pointed by some pointer $P[i]$ in a transition labeling some node of S. It is called an* ordinary node *if it is neither a milestone nor a node with more than one predecessor.*

Note that each node of a semi-skeleton has at most one immediate successor. In particular, a semi-skeleton not a t-dag. We want the properties of a skeleton to characterize S^r_G.

Proposition 4.11 *Let G be a t-dag representing a term t, and r be an accepting run of a TAGED $\mathcal{A}$ on t. S^r_G is a skeleton, and has at most $2^{2|Q|}$ milestones*

proof. It follows directly from the above definition and from the theorem that S^r_Q is a skeleton. The maximal index of a node pointed in some transition in S^r_G is $2^{|Q|}$. Since a milestone of S^r_Q is either such a node or the root u of G, and since u is the unique node such that $states_r(u)$ contains a final state, we have that the number of milestones of S^r_Q is bounded by $2^{|Q|} \times 2^{|Q|} = 2^{2|Q|}$. □

As previously mentioned, we want to generate from S^r_G, and more generally, from any semi-skeleton, some t-dag with a bounded number of states which will be eventually represent term recognized by the TAGED. We define here how we can obtain a directed ordered labeled graph from a semi-skeleton.

Definition 4.12 *Given a semi-skeleton S of a t-dag G and a run r, the* induced graph *of S is the unique directed ordered labeled graph $G' = \langle V_{G'}, succ_{G'}, \lambda_{G'} \rangle$ such that*

- $V_{G'} = V_S$
- *for all $v \in V_S$, if $\lambda_S(v) = f(P_1[i_1], \ldots, P_k, \ldots, P_n[i_n]) \to P$ then*
  - $\lambda_{G'}(v) = f$
  - $succ_{G'}(k, v) = succ_S(v)$
  - *for all $j = 1, \ldots, k-1, k+1, \ldots, n$, $succ_{G'}(i, v)$ is the node pointed by $P_j[i_j]$.*

We will focus on semi-skeleton whose induced graphs are t-dags:

Definition 4.13 *A (semi-)skeleton is said* perfect *if its induced graph is a t-dag.*

Let us prove that we can build a t-dag from S^r_G.

Proposition 4.14 *Let G be a t-dag representing a term t, and r be an accepting run of a TAGED $\mathcal{A}$ on t. The directed acyclic labeled graph S^r_G is a perfect skeleton.*

proof. Let $G' = \langle V_{G'}, succ_{G'}, \lambda_{G'} \rangle$ be the induced graph of S^r_G. In order to prove that G' is a t-dag, we first have to prove that it is acyclic. An immediate successor of a node v in G' is either an immediate successor of v in G or a node pointed by the labelling for v in S. In both cases, it is a node below v in G, and hence, G' is an acyclic graph. Moreover, G' is labeled on Σ, and each node labeled with a function symbol of arity n has exactly n immediate successors.

We still have to prove that two subdags of G' rooted at two different nodes are not isomorphic. Let us first recall that G is a t-dag, and the height of each node in G' is equals to its height in S^r_G which is equals to its height in G. Let ${v_1}'$ and ${v_2}'$ be two different nodes in $V_{G'}$ and let ${G_1}' = G'|_{{v_1}'}$ and ${G_2}' = G'|_{{v_2}'}$. If ${v_1}'$ and ${v_2}'$ have different height, they

cannot represent the same term, and we are done. So we can assume that ${v_1}'$ and ${v_2}'$ are of same height. Let us prove that they represent different terms by induction on their common height.

If ${v_1}'$ and ${v_2}'$ are of height 1 in G', then, they are of height 1 in G and hence, are representing constant symbols. Since ${v_1}' \neq {v_2}'$ and G is a t-dag, they represent two different constant symbols.

Assume by induction that no two closed subgraphs of G' rooted at nodes of height h are isomorphic, and that ${v_1}'$ and ${v_2}'$ are of height $h+1$. If $\lambda({v_1}') \neq \lambda({v_2}')$, then ${G'}_1$ and ${G_2}'$ are not isomorphic. Assume that $\lambda({v_1}') = \lambda({v_2}')$. If their main successors are at different positions k^1 and k^2 (assume *wlog* $k^1 < k^2$), then the k^1-th immediate successor of ${v_1}'$ has a greater height than the k^1-th immediate successor of ${v_2}'$ in G'. Hence the closed subgraphs rooted at those two nodes are not isomorphic and neither are ${G_1}'$ and ${G_2}'$.

Assume that the main successors of ${v_1}'$ and ${v_2}'$ are at the same position k. If they are different, then by induction hypothesis, the closed subgraphs rooted at them are non-isomorphic and neither are ${G_1}'$ and ${G_2}'$. So we can assume that ${v_1}'$ and ${v_2}'$ are labeled by the same n-ary symbol f in G', and that they have the same main successor v' at the same position k.

Let $f(P_1^1[i_1^1], \ldots, P_k^1, \ldots P_n^1) \to P^1$ be the labelling of ${v_1}'$ in S and $f(P_1^2[i_1^2], \ldots, P_k^2, \ldots P_n^2[i_n^2]) \to P^2$ be the labelling of ${v_2}'$ in S.

If there exists some j such that $P_j^1 \neq P_j^2$, then $P_j^1[i_j^1]$ and $P_j^2[i_j^2]$ necessarily point to two different nodes. Hence, the j-th immediate successors of ${v_1}'$ and ${v_2}'$ are different, and by induction their closed subgraphs are non-isomorphic. So neither are ${G'}_1$ and ${G'}_2$.

Otherwise, we have that $P_j^1 = P_j^2$ for all j. So according to the fork conditions, we have that the two tuples $\langle i_1^1, \ldots, i_{k-1}^1, i_{k+1}^1, \ldots, i_n^1 \rangle$ and $\langle i_1^1, \ldots, i_{k-1}^1, i_{k+1}^1, \ldots, i_n^1 \rangle$ are distinct. So there is some j such that $i_j^1 \neq i_j^2$ and hence $P_j^1[i_j^1]$ and $P_j^1[i_j^1]$ point to two different nodes. So $succ_{G'}(j, {v_1}') \neq succ_{G'}(j, {v_2}')$, and by induction, their closed subgraphs are non-isomorphic, and neither are ${G_1}'$ and ${G_2}'$ □

2.4 Properties of Semi-Skeleton

We are able to build a perfect skeleton from a given t-dag and the run of a TAGED, and to build its induced graph that we proved to be a t-dag. It is not sufficient for our purpose, since we want this t-dag to be recognized by the same TAGED. In this section, we will prove that this is the case. First, we have to show that given an induced graph of a perfect semi-skeleton, and that for each node, one can always find a transition rule of the TAGED consistent with the mapping of the nodes.

Lemma 4.15 *Let $G = \langle V_G, succ_G, \lambda_G \rangle$ be a t-dag representing a term t, let $\mathcal{A} = \langle Q, F, \Delta, C \rangle$ be a TAGED, and r an accepting run of $\mathcal{A}$ on t. Let $S = \langle V_S, succ_S, \lambda_S \rangle$ be a perfect semi-skeleton of G and r, and $G' = \langle V_{G'}, succ_{G'}, \lambda_{G'} \rangle$ its induced graph. Then, for all $v \in V_{G'}$ such that*

$\lambda_{G'} = f$, *and for all* $q \in states_r(v)$, *there exists* $f(q_1, \ldots, q_n) \to q \in \Delta$ *such that for all* $i, 1 \leq i, \leq n$, $q_i \in states_r(succ(i, v))$

proof. Let v be a node in G'. By definition of a semi-skeleton, $\lambda_S(v) = f(P_1[i_1], \ldots, P_k, \ldots, P_n[i_n]) \to P$ such that $P = states_r(v)$, and $P_i = states_r[succ_G(i, v)]$ for all $i, 1 \leq i \leq n$. By definition of a semi-skeleton and by construction of the induced graph, we have that $states_r[succ_{G'}(i, v)] = P_i$ for all $i, 1 \leq i \leq n$. where $v_i{}'$ is the i-th immediate successor of v in G'. And by Proposition 4.5, we have that for all $q \in P$ there exists $f(q_1, \ldots, q_n) \to q \in \Delta$ where for all $i, q_i \in P_i$. □

Now, we can prove that the induced graph of a perfect semi-skeleton is in the language of the TAGED.

Theorem 4.16 *Let* $G = \langle V_G, succ_G, \lambda_G \rangle$ *be a t-dag representing a term* t, *let* $\mathcal{A} = \langle Q, F, \Delta, C \rangle$ *be a TAGED, and* r *an accepting run of* $\mathcal{A}$ *on* t. *Let* $S = \langle V_S, succ_S, \lambda_S \rangle$ *be a perfect semi-skeleton of* G *and* r, *and* $G' = \langle V_{G'}, succ_{G'}, \lambda_{G'} \rangle$ *its induced graph. Then* G' *represents a term recognized by* $\mathcal{A}$.

proof. We prove this proposition constructively by building a run r' of $\mathcal{A}$ on the term t' represented by G', and showing that this is an accepting run. We define r', $\mathcal{P}os(r') = \mathcal{P}os(t')$, recursively on the length of the positions, respecting the following invariant: for all position $p, r'(p) \in states_r(v)$ where v is the root of the subdag $G'|_p$.

- for length 0, we define $r'(\varepsilon) = q_f$. Since the root of G' is the root u of G and $states_r(u) = q_f$ with q_f the final state used in r, the invariant is respected
- if $r'(p)$ is defined for all position p of length n, and such that $r'(p)$ is a state q occurring at $states_r(v')$ where v' is the root of the subdag $G'|_p$. If $t(p)$ is a function symbol of arity $n > 0$, then, by proposition 4.5 there exists a rule $f(q_1, \ldots, q_n) \to q \in \Delta$ such that for all $i, 1 \leq i \leq n, q_i \in states_r(succ_{G'}(i, v'))$. For all $i, 1 \leq i \leq n$, we define $r(p.i) = q_i$. Since $succ_{G'}(i, v')$ is the root of the subdag $G'|_{p.i}$, we respect the invariant

Clearly with this construction, we have that r' is a run of the underlying tree automaton of the TAGED $\mathcal{A}$ on t'. Let us prove that it is an accepting one. First, we have that $r'[\varepsilon] = q_f$ which is a final state. So r is an accepting run of the underlying tree automaton, and we only have to ensure that (dis)equalities are respected.

Let $q_1 \approx q_2 \in C$ and p_1 and p_2 be two positions such that $r'(p_1) = q_1$ and $r'(p_2) = q_2$. According to proposition 4.5, there is at most one node v of G such that $states_r(v)$ may contain both q_1 and q_2. We have that v is the root of both the subdags $G'|_{p_1}$, $G'|_{p_2}$ which are, hence, equals. Especially, $t'|_{p_1} = t'|_{p_2}$.

Let $q_1 \not\approx q_2 \in C$ and p_1 and p_2 be two positions such that $r'(p_1) = q_1$ and $r'(p_2) = q_2$. Since that, according to proposition 4.5, no node v of

G is such that $states_r(v)$ may contain both q_1 and q_2, we have that the subgraphs of G' representing $t'|_{p_1}$ and $t'|_{p_2}$ are rooted by two different nodes $v_1{}'$ and $v_2{}'$. Since G' is a t-dag, we hence have that $t'|_{p_1} \neq t'|_{p_2}$. □

2.5 Pumping Within a Skeleton

We now are able to build a skeleton with a bounded number of milestones from which we can build a t-dag that is recognized by the TAGED we want. However, bounding the number of milestones is not enough, since there can be an unbounded number of ordinary nodes. In this section, we show than we can pump on ordinary nodes as we would have done in TA.

Proposition 4.17 *Given a (perfect) semi-skeleton S of a t-dag G and of a run r of a TAGED $\mathcal{A}$, if there exists a sequence $(v_0, v_1, \dots v_n)$ of ordinary nodes of S such that*

- *for all $i, 0 \leq i \leq m-1$, v_{i+1} is the immediate successor of v_i, and*
- *the set of states provided by v_1 and v_m are the same.*

then the graph S' obtained from S by removing the nodes $v_1, \dots, v_{m-1}$ and defining v_m as the immediate successor of v_0 S is a (perfect) semi-skeleton of G and r.

proof. It is obvious that the set of vertices is still a subset of the vertices of G. Since the root of G is a milestone, it will not be removed, and neither will the nodes pointed by any labelling since they are all milestones. The transitions with pointers label the same nodes, so they still are compatible with their respective nodes. Let $f(P_1[i_1], \dots, P_k, \dots, P_n[i_n])$ be the transition labelling v_0. Thus, since v_1 and v_m provides the same set of states, and since S is a semi-skeleton, v_m provides P_k. The other nodes' immediate successors have not changed. So S' is a semi-skeleton

If S is a perfect semi-skeleton, then its induced graph G' is a t-dag. Let G'' be the induced graph of S'. It is the same graph excepted that it does not contain the nodes $v_1, \dots, v_{m-1}$ and that $succ_{G''}(k, v_0) = v_m$. It is easy to see that there is a path between two nodes of G'' if and only if there is a path between those two nodes in G'. Hence G'' is acyclic. Since the labeling and the arity of each node in G'' are the same than the ones in G', it is sufficient to prove that two subgraphs of G'' rooted at two different nodes are not isomorphic.

Let $v_1{}'$ and $v_2{}'$ be two different nodes of G''. Let $G_1{}' = G'|_{v_1{}'}$, $G'_2 = G'|_{v_2{}'}$ $G_1{}'' = G''|_{v_1{}'}$ and $G_2{}'' = G''|_{v_2{}'}$. The node v_m is a successor of $v_1{}'$ in S' if and only if $v_1{}'$ is v_0 or v_0 is one of its successors. So if v_m is not a successor of $v_1{}'$ in S', then none of the nodes $v_1, \dots, v_m$ is a successor of $v_1{}'$ in S. And since $v_1, \dots, v_m$ are not milestones, none of these nodes belongs to G'_1, and so no node of $G_1{}'$ is removed from S to obtain S'. Hence $G_1{}' = G_1{}''$.

If v_m is neither a successor of ${v_1}'$ in S' nor one of ${v_2}'$, then, since ${G_1}'$ is not isomorphic to ${G_2}'$, neither are ${G_1}''$ and ${G_2}''$. If v_m is a successor of both ${v_1}'$ and ${v_2}'$ in S', then, since they are different, they have different height in S', and hence different height in G''. Hence ${G_1}''$ and ${G_2}''$ are not isomorphic.

Assume that v_m is a successor of v_1' in S' but not of ${v_2}'$. Then we have ${G_2}' = {G_2}''$. If ${G_1}''$ and ${G_2}''$ are isomorphic, they have the same set of positions. Moreover, let p be the position of v_m in ${G_1}''$, then p is a position of ${G_2}''$, and ${G_1}''|_p$ and ${G_2}''|_p$ are isomorphic. Since, ${G_1}''|_p = G''|_{v_m} = G'|_{v_m}$ and ${G_2}''|_p = {G_2}'|_m$ it means that $G'|_{v_m}$ is isomorphic to ${G_2}'|_p$. But v_m does not belong to ${G_2}'$, so ${G_2}'[p] \neq v_m$, and since G' is a t-dag, $G'|_{v_m}$ and ${G_2}'|_p$ are not isomorphic.

Hence ${G_1}''$ and ${G_2}''$ are not isomorphic, so G'' is a t-dag and S' is a perfect semi-skeleton. □

Note that this transformation does not preserve the skeleton property, since a labelling of a node below v_m may point to a node that will be below it in S but not in S'.

The figure 4.4 shows a pumping of the skeleton of figure 4.3. Note that it only removed one node. Then, a graphic representation of its induced graph G' is done in figure 4.5 where at each node v we wrote $\lambda'_G(v) : states_r(v)$. The order of the edges of the nodes is misleading. The leftmost edge on the graphic is actually its second edge.

Now, we have all the components we need in order to reduce the emptiness problem of TAGED to the existence of a small term recognized by the automaton.

Theorem 4.18 *Let $\mathcal{A}$ be a TAGED. The language $\mathcal{L}(\mathcal{A})$ is not empty if and only if it contains a term t such that $dag(t)$ has less than $2^{3|Q|} + 2^{2|Q|+1} + 2^{|Q|}$ nodes.*

proof. It is obvious that if such a term exists, $\mathcal{L}(\mathcal{A})$ is not empty. Reciprocally, if $\mathcal{L}(\mathcal{A})$ is not empty, then, there exists a term $t \in \mathcal{L}(\mathcal{A})$. Let r be a successful run of $\mathcal{A}$ on t. Let $G = dag(t)$. Then according to propositions 4.11 and 4.14, S^r_G is a perfect skeleton of G that has less than $2^{2|Q|}$ milestones. Since each node that does not have a predecessor is necessarily a milestone, there are at most $2^{|Q|}$ node with more than one predecessors. By repeating several times the construction of proposition 4.17 we can obtain a perfect semi-skeleton such that there is at most $2^{|Q|}$ successive ordinary nodes. So there exists a perfect semi-skeleton with less than $(2^{|Q|} + 1)(2^{2|Q|} + 2^{|Q|}) = 2^{3|Q|} + 2^{2|Q|+1} + 2^{|Q|}$ nodes. The induced graph G' of this perfect-semi skeleton is a t-dag whose node set is the same and representing a term t' which is recognized by $\mathcal{A}$ according to theorem 4.16. □

It follows directly from this theorem a non-deterministic procedure to decide whether the language of a TAGED is empty.

Corollary 4.19 *The emptiness problem for TAGED is decidable in NEXPTIME*

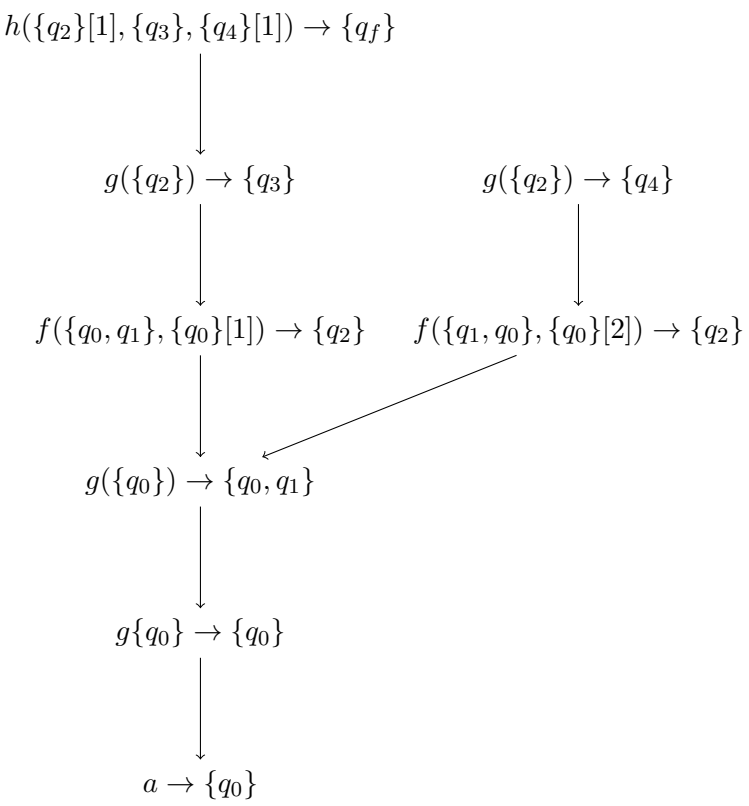

Figure 4.4: A pumping of the skeleton of figure 4.3

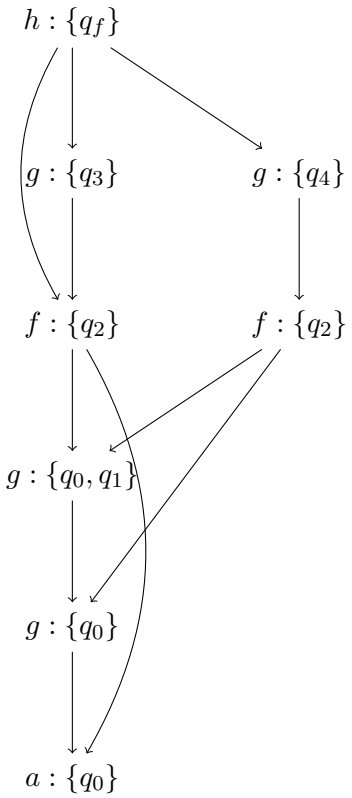

Figure 4.5: The induced graph of the semi-skeleton of figure 4.4

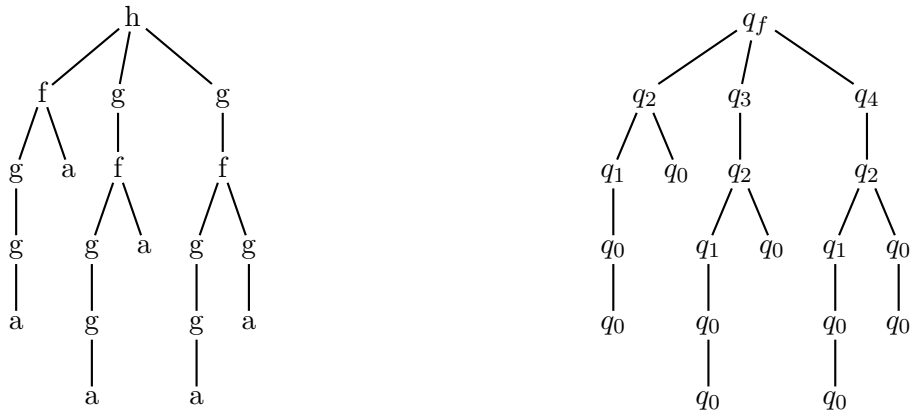

Figure 4.6: The term and the run inferred from the t-dag of figure 4.5

proof. Following the previous theorem, to decide whether the language of a TAGED $\mathcal{A}$ is empty, we can guess a t-dag G with at most $\mathcal{O}(2^{3|Q|})$ nodes, and a labelling $states : V_G \to 2^Q$ and verify that we can deduce an accepted run of $\mathcal{A}$ on the term t represented by G. Such a run exists, if the root of $\mathcal{A}$ is labeled by a single final state, if no node is labeled by two different states q_1, q_2 such that $q_1 \not\approx q_2$, and if each labeling verifies the property 4 of Proposition 4.5. We can check the labeling of the root in constant time. Then, for each node of G, we can verify that no two nodes in disequality relations label G in $\mathcal{O}(|Q|^2)$, and the property 4 in $\mathcal{O}(|Q|^m)$ where m is the maximal arity of the function symbols of Σ. Since we can assume that m is a constant, the verification for each node can be done in polynomial time. As there are $\mathcal{O}(2^{3|Q|})$ nodes to check, we can verify that G is the t-dag representation of an accepted term in exponential time. Hence, the emptiness problem for TAGED is in NEXPTIME.

□

With our running example, given the induced graph of the pumping of the initial skeleton, we obtain the term and the successful run of the figure 4.6.

3 Conclusion

In this chapter, we answered positively the open problem of decidability of the emptiness problem for the TAGEDS [FTT08]. We give an upper bound on the complexity, NEXPTIME. Indeed, the emptiness decision problem for t-dag automata presented in [Cha99], that we adapt to prove our result, has been shown NP-complete. Since we use quite a same construction but with a data input exponential in size, it could be possible that emptiness for TAGED is NEXPTIME-complete.

As a direct consequence, our result extends the satisfiability result of the fragment of the tree query-logic described in [FTT08] (TQL was introduces in [CG03]). Moreover, it can also be used to encode some

structural constraints on XML documents, and satisfies some extension of MSO on trees. However, these results cannot be applied to the full class of TAGC, presented in chapter 2 because the disequality is not limited to irreflexive relations. The emptiness decision procedure for this full class is more complex and will be presented in the next chapter.

Chapter 5

Deciding Emptiness for TAGC

We proved in the last chapter that the emptiness problem is decidable for TAGED. This result can be used to decide the satisfiability of structural constraints on XML documents. As we showed in the example 2.6, some constraints known as "key constraints" cannot be encoded into a TAGED. Yet, they are common integrity constraints on XML documents (see [FL02]). Those require to have reflexive disequality constraints, which are provided by TAGC.

In order to prove the decision result of emptiness to TAGC, there are two problems to solve. First, the constraints of TAGED are only positive conjunctive, as for PCTAGC. And second, reflexive disequalities (constraints of the form $q \not\approx q$) are not handled by TAGED.

We solve the first problem in the first section of this chapter. We show that TAGC and PCTAGC are equally expressive. In order to do this, we add some arithmetic constraints on the number of different terms occurring in a given state. We show that negations of atomic constraints $\approx$ and $\not\approx$ can be eliminated using new states and arithmetic constraints. Then, we show that TAGC with those arithmetic constraints can be encoded by PCTAGC. We also prove some properties of this extension of TAGC.

The pumping technique described in chapter 4 cannot deal with reflexive disequality constraints. So in the second section we present a new pumping lemma and apply it to PCTAGC to prove the decidability of the emptiness problem. The pumping lies on some well quasi-order, leading to a non-primitive recursive complexity. The complexity gap between this emptiness decision procedure, and the one seen in the previous chapter justifies the interest of presenting both of them, one being slightly more complex, but dealing with a more expressive class than the other.

We show that the emptiness decision algorithm can also be applied to the combination of TAGC with local tests between sibling subtrees a la [BT92] (section 3), and to unranked ordered labeled trees (section 4).

This demonstrates the robustness of the method.

As an application of our results, in section 5 we present a (strict) extension of the monadic second order logic on trees whose existential fragment corresponds exactly to TAGC. In particular, we conclude its decidability.

1 TAGC with Arithmetic Constraints

We study first the addition of counting constraints to TAGC.

Definition 5.1 *Let Q be a set of states. A* linear inequality *over Q is an expression of the form $\sum_{q\in Q} a_q.|q| \geq a$ or $\sum_{q\in Q} a_q.\|q\| \geq a$ where every a_q and a belong to $\mathbb{Z}$.*

Let r be a run on a term t of a TA or TAGC $\mathcal{A}$ over $\mathcal{F}$ and with state set Q, and let $q \in Q$. The interpretations of $|q|$ and $\|q\|$ *wrt* r (and t) are defined respectively by the following cardinalities $[\![\, |q| \,]\!]_r = |r^{-1}(q)|$ and $[\![\, \|q\| \,]\!]_r = \big|\{s \in \mathcal{T}(\mathcal{F}) \mid \exists p \in \mathcal{P}os(t), r(p) = q, s = t|_p\}\big|$.
This permits to define the satisfiability of linear equalities *wrt* t and r and the notion of successful runs for extensions of TAGC with atomic constraints which can have the form of the above linear inequalities.

Example 5.2 *Let us add a new argument to the dishes of to the menu of Example 2.6 which represents the price coded on two digits by a term $N(d_1, d_0)$. We add a new state q_p for the type of prices, and other states q_{cheap}, $q_{moderate}$, $q_{expensive}$, q_{chic} describing price level ranges, and transitions $0|1 \to q_{cheap}$, $2|3 \to q_{moderate}$, $4|5|6 \to q_{expensive}$, $7|8|9 \to q_{chic}$ and $N(q_{cheap}, q_d) \to q_p$,... The price is a new argument of $\mathcal{L}_0$, $\mathcal{L}$ and M, hence we replace the transitions with these symbols in input by $\mathcal{L}_0(q_{id}, q_t, q_p) \to q_L$, $\mathcal{L}(q_{id}, q_t, q_p, q_L) \to q_L$, $M(q_{id}, q_t, q_p, q_L) \to q_M$. We can use a linear equality $|q_{cheap}| + |q_{moderate}| - |q_{expensive}| - |q_{chic}| \geq 0$ to characterize the moderate menus, and $|q_{expensive}| + |q_{chic}| \geq 6$ to characterize the menus with too many expensive dishes. A linear equality $\|q_p\| \leq 1$ expresses that all the dishes have the same price.*

1.1 Relative Linear Inequalities

Let us denote by $|.|_{\mathbb{Z}}$ and $\|.\|_{\mathbb{Z}}$ the types of the above linear inequalities, seen as atomic constraints of TAGC. The class TAGC$[\,|.|_{\mathbb{Z}}]$ has been studied under different names (*e.g.* Parikh automata in [KR02], linear constraint tree automata in [BMSS09]) and it has a decidable emptiness test. Indeed, the set of successful runs of a given TA with state set Q describes a context free language (over Q^*), and the Parikh projection (the set of tuples over $\mathbb{N}^{|Q|}$ whose components are the $[\![\, |q| \,]\!]_r$ for every run r) of such a language is a semi-linear set. The idea for deciding emptiness for a TAGC$[|.|_{\mathbb{Z}}]$ $\mathcal{A}$ is to compute this semi linear set and test

the emptiness of its intersection with the set of solutions in $\mathbb{N}^{|Q|}$ of $C_{\mathcal{A}}$, (a Boolean combinations of linear inequalities of type $|.|_{\mathbb{Z}}$) which is also semi-linear. This can be done in NPTIME, see [BMSS09].

Combining constraints of type $\approx$ and counting constraints of type $|.|_{\mathbb{Z}}$ however leads to undecidability.

Theorem 5.3 *Emptiness is undecidable for PTAGC*$[\approx,|.|_{\mathbb{Z}}]$.

proof. We reduce Hilbert's tenth problem. Let $a_1\, x_1^{i_{1,1}} \dots x_m^{i_{1,m}} + \dots + a_n\, x_1^{i_{n,1}} \dots x_m^{i_{n,m}} = 0(\phi)$ be a Diophantine equation with m variables $x_1, \dots, x_m$, $i_{j,k} \geq 0$ for all $j \leq n$, $k \leq m$ and $a_1, \dots, a_n \in \mathbb{Z}$. We will construct a TAGC$[\approx, |.|_{\mathbb{Z}}]$ $\mathcal{A}$ such that $\mathcal{L}(\mathcal{A}) \neq \emptyset$ iff there exists a valuation of $x_1, \dots, x_m$ in $\mathbb{N}$ satisfying the above equation.

The most important step of the construction is that we can encode (non negative) integer multiplication with TAGC$[\approx, |.|_{\mathbb{Z}}]$.

Let us consider for instance the product $x_1.x_2$ and three TAGC states q_1, q_2 and q_{12}, whose number of occurrences will represent the respective valuations of x_1, x_2 and their product (with our notations, $|q_1| = x_1$, $|q_2| = x_2$, $|q_{12}| = x_1.x_2$). We use a signature with symbols a and b of arity 0, f of arity 2, and $*$ of arity 3. For $x \geq 1$, let t_x be the right comb $f(a, f(a, \dots, f(a, b)))$ with x occurrences of a and one occurrence of b. The product $x_1.x_2$ is represented by $t_{x_1,x_2} := f(t_{x_1}, \dots, f(t_{x_1}, b))$ with x_2 occurrences of t_{x_1}.

Let us now construct a TAGC$[\approx, |.|_{\mathbb{Z}}]$ $\mathcal{A}_*$ recognizing all $*(t_{x_1}, t_{x_2}, t_{x_1,x_2})$. During the computation of $\mathcal{A}_*$, in the first argument t_{x_1}, every leaf labeled with a goes to the state q_1, with the transition rule $a \to q_1$, and t_{x_1} goes to a state r_1. For the intermediate steps, we use a state p_1 and transitions $b \to p_1$, $f(q_1, p_1) \to p_1$, and $f(q_1, p_1) \to r_1$. We proceed in the same manner for t_{x_2} with the states q_2, r_2, p_2. Finally, for the recognition of the third argument t_{x_1,x_2} by $\mathcal{A}_*$, every leaf labeled with a goes to the state q_{12}, every subterm t_{x_1} goes to a state r'_1 and the term t_{x_1,x_2} goes to r_{12}. For this purpose we have the transitions $a \to q_{12}$, $b \to p'_1$, $f(q_{12}, p'_1) \to p'_1$, $f(q_{12}, p'_1) \to r'_1$, and $b \to p_{12}$, $f(r'_1, p_{12}) \to p_{12}$, $f(r'_1, p_{12}) \to r_{12}$. Finally, we add a rule $*(r_1, r_2, r_{12}) \to q_*$ to $\mathcal{A}_*$.

Let r be a run of $\mathcal{A}_*$ on a term $t = *(t_{x_1}, t_{x_2}, t_3)$ similar to the one presented in Figure 5.1. By construction, the number of occurrences of q_1 in r is x_1, *i.e.* $[\![\, |q_1| \,]\!]_r = x_1$. Similarly, $[\![\, |q_2| \,]\!]_r = x_2$. Let us consider the constraint $C_* = \big(|r'_1| = |q_2| \wedge r_1 \approx r'_1\big)$ (for simplicity, we use linear equalities representing the conjunction of two linear inequalities). By imposing the constraint C_* to the run r of $\mathcal{A}_*$, we ensure that t_3, the third argument of t, contains x_2 nested copy of t_{x_1}. It follows that the number $[\![\, |q_{12}| \,]\!]_r$ of occurrences of q_{12} in the run r is equal to $x_1.x_2$.

The general construction of the TAGC$[\approx, |.|_{\mathbb{Z}}]$ $\mathcal{A} = \langle Q, \mathcal{F}, F, C, \Delta \rangle$ associated to the above Diophantine equation (ϕ) follows the same principle, except that we may have several level of nesting in a term like t_3 above. For encoding the product of more than two integers, we have in $\mathcal{A}$ some constraints $\approx$ and $|.|_{\mathbb{Z}}$ similar to the above ones, and an addi-

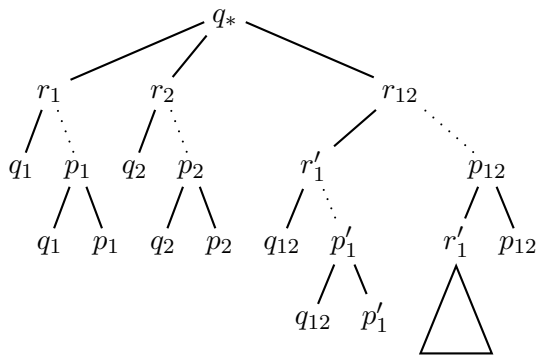

Figure 5.1: A run on $*(t_{x_1}, t_{x_2}, t_{x_1.x_2})$

tional constraint $\sum a_j.|q_j| = 0$ where each q_j is uniquely associated to a variable x_j, as above.

Let $\mathcal{W}$ be the set containing all the variables of (ϕ), $\{x_1, \ldots, x_m\}$ and all the prefixes of the monomials of (ϕ) $x_j^{i_{j,1}} \ldots x_m^{i_{j,m}}$ for all $j \leq n$ (we consider every monomial as a word over the alphabet $\{x_1, \ldots, x_m\}$). Let us denote the prefix ordering over strings by $\preceq$, and for every words v, v', w such that $w = v.v'$ (*i.e.* $v \preceq w$) we recall that $w - v = v'$ and let $w \div v$ be the first letter of v' when this latter word is not empty. We assume below that $\mathcal{W}$ is ordered arbitrarily in a sequence without duplicates $\overrightarrow{w} = (w_1, \ldots, w_N)$.

The signature $\mathcal{F}$ contains the symbols a, b, and f with the respective arity 0, 0 and 2 as above, and two symbols c of arity 0 and g of arity 2. Let $Q = \{q_w, s_w, r_w^v, p_w^v \mid v, w \in \mathcal{W}, v \preceq w\} \cup \{s_0\}$ and $F = \{s_{w_N}\}$. The transitions of Δ are the following: $a \to q_w$, $b \to p_w^v$, $f(q_w, p_w^v) \to p_w^v$, $f(q_w, p_w^v) \to r_w^v$ for all $v, w \in \mathcal{W}$ with $v \preceq w$, and $f(r_w^v, p_w^{v.x_i}) \to p_w^{v.x_i}$, $f(r_w^v, p_w^{v.x_i}) \to r_w^{v.x_i}$, for all $v, w \in \mathcal{W}$ and x_i variable of (ϕ) $(i \leq m)$, with $v.x_i \preceq w$. Moreover, we also have: $c \to s_0$, $g(r_{w_1}^{w_1}, s_0) \to s_{w_1}$ and $g(r_{w'}^{w'}, s_w) \to s_{w'}$ for all $w, w' \in \mathcal{W}$ such that $w = w_i$ and $w' = w_{i+1}$ in the sequence $\overrightarrow{w}$, for some $i < N$.

We can observe that with these transitions, if r is an accepting run of $ta(\mathcal{A})$ on some term t, then for all $w \in \mathcal{W}$, the state s_w occurs exactly once in r at some position $p_w \in 2^*$, and the leaves of $r|p_w.1$ are all labelled by the state q_w. Moreover, the state q_w occurs only in this subterm of r.

Finally, the constraint C of $\mathcal{A}$ is defined as the conjunction of the following atomic constraints:

i. $|r_w^v| = |q_{w \div v}|$ for all $v, w \in \mathcal{W}$ with $v \prec w$,

ii. $r_w^v \approx r_{w'}^v$ for all $v, w, w' \in \mathcal{W}$ with $v \preceq w, w'$,

iii. $\sum_{j=1}^{n} a_j.|q_{w_j}| = 0.$

In *iii.*, the a_j's are the coefficients of (ϕ) and $w_j = x_1^{i_{j,1}} \dots x_m^{i_{j,m}}$ (we assume that the n first elements of the sequence $\overrightarrow{w}$ correspond to these monomials).

With the above observation, the construction of the transition rules of $\mathcal{A}$, and the above constraints *i.* and *ii.* we have that for all run r of $\mathcal{A}$ on a term t, all variables x_i of (ϕ) and all $w.x_i \in \mathcal{W}$, $[\![\, |q_{w.x_i}| \,]\!]_r = [\![\, |q_w| \,]\!]_r \,.\, [\![\, |q_{x_i}| \,]\!]_r$. It follows, using the constraint *iii*, that $\mathcal{L}(\mathcal{A}) \neq \emptyset$ iff (ϕ) has a solution in $\mathbb{N}$. □

1.2 Natural Linear Inequalities

We present now a restriction on linear equalities which enables the reduction to PTAGC when combined with $\approx$ and $\not\approx$ as global constraints.

Definition 5.4 *A* natural linear inequality *over Q is a linear inequality as in Definition 5.1 whose coefficients a_q and a all have the same sign.*

Note that it is equivalent to consider inequalities in both directions whose coefficients are all non-negative, like $\sum a_q.|q| \leq a$, with $a_q, a \in \mathbb{N}$, to refer to $\sum -a_q.|q| \geq -a$. We also consider linear equalities $\sum a_q.|q| = a$, with $a_q, a \in \mathbb{N}$, to refer to a conjunction of two natural linear disequalities. The types of the natural linear inequalities are denoted by $|.|_{\mathbb{N}}$ and $\|.\|_{\mathbb{N}}$. Below, we shall abbreviate these two types by $\mathbb{N}$.

The main difference between the linear inequalities of type $|.|_{\mathbb{Z}}$ and $|.|_{\mathbb{N}}$ (and respectively $\|.\|_{\mathbb{Z}}$ and $\|.\|_{\mathbb{N}}$) is that the former permits to compare the respective number of occurrences of two states, like *e.g.* in $|q| \leq |q'|$, whereas the latter only permit to compare the number of occurrences of one state (or a sum of the number occurrences of several states with coefficients) to a constant as *e.g.* in $|q| \leq 4$ or $|q|+2.|q'| \leq 9$. This difference permits to encode the (in)equalities with additional states and (dis)equalities constraints, and to reduce TAGC$[\approx, \not\approx, \mathbb{N}]$ to PCTAGC$[\approx, \not\approx]$, which we will prove emptiness decidability in next section.

The proof that TAGC$[\approx, \not\approx, \mathbb{N}]$ are equally expressive as PTAGC$[\approx, \not\approx]$ works in two steps. In the first step (Proposition 5.5) we restrict ourselves to positive TAGC (let us recall that PTAGC denotes the class of positive TAGC$[\approx, \not\approx]$) and show that we can get rid of $|.|_{\mathbb{N}}$ and $\|.\|_{\mathbb{N}}$ while keeping the same class of recognized languages. Then, in Proposition 5.9, we show how to get rid of negative constraints in TAGC$[\approx, \not\approx, \mathbb{N}]$.

Proposition 5.5 *For all PTAGC$[\approx, \not\approx, \mathbb{N}]$ $\mathcal{A}$, one can effectively construct a PTAGC$[\approx, \not\approx]$ $\mathcal{A}'$ such that $\mathcal{L}(\mathcal{A}') = \mathcal{L}(\mathcal{A})$.*

The first step of this construction consists in rewriting conjunctions of constraints with $|.|_{\mathbb{N}}$ and $\|.\|_{\mathbb{N}}$ into disjunctions of conjunctions of simpler constraints.

Lemma 5.6 *Every conjunction φ of natural linear inequalities over Q can be effectively rewritten into an equivalent disjunction $\bigvee_{1\leq i\leq n}\bigwedge_{q\in Q}\alpha_{|q|}\wedge\alpha_{\|q\|}$ for some $n\in\mathbb{N}$, where $\alpha_{|q|}$ (resp. $\alpha_{\|q\|}$) is either $|q|=a_q$ or $|q|\geq a_q$ for some $a_q\in\mathbb{N}$, or $\top$ (resp. $\|q\|=b_q$ or $\|q\|\geq b_q$ for some $b_q\in\mathbb{N}$, or $\top$).*

proof. First, we separate the conjunction of inequalities in two parts: we denote by $\varphi_{|.|}$ the conjunction of atomic constraints of type $|.|_{\mathbb{N}}$ (inequalities on occurrences), and $\varphi\|.\|$ the conjunction of atomic constraints of type $\|.\|_{\mathbb{N}}$ (inequalities on cardinality). So $\varphi=\varphi_{|.|}\wedge\varphi_{\|.\|}$. Then, we further separate the two parts into $\varphi_{|.|}=\chi_{|.|}\wedge\psi_{|.|}$ and $\varphi_{\|.\|}=\chi_{\|.\|}\wedge\psi_{\|.\|}$ where $\chi_{|.|}$ contains all the inequalities of the form $|q|=a$ or $|q|\geq a$ where q occurs there only, and $\psi_{|.|}$ contains all the other inequalities. Note that each state occurring in $\chi_{|.|}$ occurs only in this subformula and does not occur in $\psi_{|.|}$. Respectively, $\chi_{\|.\|}$ is the conjunction of all the inequalities of $\varphi_{\|.\|}$ of the form $\|q\|=a$ or $\|q\|\geq a$ where q does not occur in another inequality, and $\psi_{\|.\|}$ contains all the other inequalities.

Let

$$\psi_{|.|}=\bigwedge_{1\leq i\leq k}\sum_{q\in Q}a^i_q.|q|\leq a^i\wedge\bigwedge_{1\leq j\leq l}\sum_{q\in Q}b^j_q.|q|\geq b^j$$

for some k and l, with $a^i_q,a^i,b^j_q,b^j\in\mathbb{N}$ for all $i\leq k$, $j\leq l$.

Let q be a state for which either there exists $i\leq k$ such that $a^i_q\neq 0$ or there exists $j\leq l$ such that $b^j_q\neq 0$. Then, we do one of the following transformations.

1. If there exists some i such that $a^i_q\neq 0$, then we define

$$s_q=\inf\big\{\lfloor\frac{a^i}{a^i_q}\rfloor\;\big|\;1\leq i\leq k,a^i_q\neq 0\big\}.$$

If $|q|>s_q$, then there exists some inequality which can not be satisfied. So in order to erase the occurrences of $|q|$ in the inequalities, we can make an enumeration of the values of $|q|$ between 0 and s_q.

We replace $\varphi_{|.|}$ by a disjunction $\bigvee_{0\leq s\leq s_q}\varphi^s_{|.|}$, where $\varphi^s_{|.|}=\chi^s_{|.|}\wedge\psi^s_{|.|}$ and $\chi^s_{|.|}=\chi_{|.|}\wedge|q|=s$, and $\psi^s_{|.|}$ is defined as follows: First, we replace every inequality $\sum_{q'\in Q}a^i_{q'}\cdot|q'|\leq a^i$ of $\psi_{|.|}$ by $\sum_{q'\in Q\setminus\{q\}}a^i_{q'}\cdot|q'|\leq\sup\{0,a^i-s\cdot a^i_q\}$ and we replace every inequality $\sum_{q'\in Q}b^j_{q'}\cdot|q'|\geq b^i$ of $\psi_{|.|}$ by $\sum_{q'\in Q\setminus\{q\}}b^j_{q'}\cdot|q'|\geq\sup\{0,b^j-s\cdot b^j_q\}$.

Then, for every inequality with no occurrence of a state in the left-hand side, which can be seen as an inequality between integers with a 0 on the left-hand side:

- if the inequality is true, then we erase it,
- if the inequality is false, then we delete the whole conjunction, which cannot be satisfied with the value given to $|q|$.

2. Otherwise (for all $a_q^i = 0$), there exists some j such that $b_q^i \neq 0$ and we define

$$s_q = \sup\{\lceil \frac{b^j}{b_q^j} \rceil \mid 1 \leq j \leq l, b_q^j \neq 0\}.$$

Note that if $|q| \geq s_q$, then all the inequalities involving q are satisfied. So in order to erase the occurrences of $|q|$, we only have to do an enumeration of the value of $|q|$ between 0 and $s_q - 1$ and to erase all the inequalities with an occurrence of $|q|$ if $|q| \geq s_q$. We replace $\varphi_{|.|}$ by a disjunction $\bigvee_{0 \leq s \leq s_q} \varphi_{|.|}^s$ where $\varphi_{|.|}^s = \chi_{|.|}^s \wedge \psi_{|.|}^s$ and $\chi_{|.|}^s = \chi_{|.|} \wedge q = s$ if $s < s_q$, $\chi_{|.|}^s = \chi_{|.|} \wedge q \geq s$ is $s = s_q$ and $\psi_{|.|}^s$ is defined as above (except that there is no $a_q^i \neq 0$) if $s < s_q$ and $\psi_{|.|}^{s_q}$ is as $\psi_{|.|}$ where every inequality $\sum_{q \in Q} b_q^j \geq b^j$ where $b_q^j \neq 0$ is deleted.

Note that, at this point of the procedure, q does not occur anymore in any $\psi_{|.|}^s$ and occurs only once in each $\chi_{|.|}^s$. While there is still a non-empty conjunction $\psi_{|.|}^s$, we apply the procedure to $\varphi_{|.|}^s$. At the end, we have an expression equivalent to $\varphi_{|.|}$ that can be written $\bigvee_{1 \leq i \leq n} \chi_{|.|}^i$ for some n where each $\chi_{|.|}^i$ is a conjunction of (in)equalities of the form $|q| = a$ or $|q| \geq a$.
We apply the same procedure to $\varphi_{\|.\|}$, which becomes a disjunction $\bigvee_{1 \leq j \leq m} \chi_{\|.\|}^j$ for some n where each $\chi_{\|.\|}^j$ is a conjunction of (in)equalities of the form $\|q\| = a$ or $\|q\| \leq a$. Hence, we have that ϕ is equivalent to $\bigvee_{1 \leq i \leq n} \chi_{|.|}^i \wedge \bigvee_{1 \leq j \leq m} \chi_{\|.\|}^j$.
Since each q occurs at most once in each $\chi_{|.|}^i$ and once in each $\chi_{\|.\|}^j$, we can obtain what we are looking for by rewriting this expression into disjunctive normal form and adding a $\top$ symbol for each q than does not occur in $\chi_{|.|}^i$, and one for each that does not occur in $\chi_{\|.\|}^i$. □

Let C be the constraint of a PTAGC$[\approx, \not\approx, \mathbb{N}]$ $\mathcal{A}$. The PTAGC$[\approx, \not\approx, \mathbb{N}]$ $\mathcal{A}'$, defined as a copy of $\mathcal{A}$ where the constraint C' is the disjunctive normal form of $\mathcal{C}$ is obviously equivalent to $\mathcal{A}$. By associativity of $\wedge$ one can rewrite C' the following way

$$C' = \bigvee_{1 \leq i \leq n} (C_+^i \wedge C_{\mathbb{N}}^i)$$

where, for all $i, 1 \leq i \leq n$, C_+^i is a conjunction of atomic constraints of the form $q \approx q'$ or $q \not\approx q'$, and $C_{\mathbb{N}}^i$ is a conjunction of natural arithmetic constraints. Let $\mathcal{A}_i$ be a copy of $\mathcal{A}$ with the constraint $C^i = C_+^i \wedge C_{\mathbb{N}}^i$. We have that $\mathcal{L}(\mathcal{A}) = \cup_{1 \leq i \leq n} \mathcal{L}(\mathcal{A}_i)$. Since PCTAGC$[\approx, \not\approx]$ are closed by union (see proposition 2.11), we only have to prove that each $\mathcal{A}_i$ is equivalent to a PCTAGC.

Lemma 5.7 *Given a PTAGC$[\approx, \not\approx, \mathbb{N}]$ $\mathcal{A} = \langle \Sigma, Q, \mathcal{F}, F, C, \Delta \rangle$, with $C = C_+ \wedge C_{\mathbb{N}}$, where C_+ is a conjunction of atomic constraints of the form $q \approx q'$ or $q \not\approx q'$, and $C_{\mathbb{N}}$ is a conjunction of natural arithmetic*

constraints, one can effectively compute a PTAGC$[\approx, \not\approx]$ $\mathcal{A}'$ *such that* $\mathcal{L}(\mathcal{A}') = \mathcal{L}(\mathcal{A})$.

proof. First, we just rewrite C_+ into a disjunction of conjunction of simple arithmetic constraints like in Lemma 5.6.

Since PCTAGC are closed by union, it suffices to effectively constructs a PCTAGC for each PTAGC$[\approx, \not\approx, \mathbb{N}]$ with constraints of the form $C_+ \wedge C'_{\mathbb{N}}$ where $C'_{\mathbb{N}}$ is of the form $\bigwedge_{q\in Q} p_{|.|}(q) \wedge p_{\|.\|}(q)$. Let us then assume that $C_{\mathbb{N}}$

The general idea is to have multiple copies of the states involved in natural (in)equality constraints, and to create $\approx$ and $\not\approx$ relations between them, in order to ensure that the wanted cardinality of the original state is reached by any successful run. Then we count the first occurrence of each state, in order to ensure that any successful run also have the correct number of occurrences of each original state.
First, we compute for each $q \in Q$ the number $c(q)$ of copies of this state that we need, depending on the constraints on q in $D^{\mathbb{N}}$.

- if $\alpha_{\|q\|}$ is $\top$ then $c(q) = 1$,
- if $\alpha_{\|q\|}$ is $\|q\| = a_q$ then $c(q) = a_q$, each state will recognize only one of the a_q different subterms,
- if $\alpha_{\|q\|}$ is $\|q\| \geq a_q$ then $c(q) = a_q + 1$, the extra state will recognize the possible extra terms.

We construct now a PCTAGC$[\approx, \not\approx]$ $\mathcal{A}' = \langle Q', \mathcal{F}, F', C'_+, \Delta' \rangle$ recognizing $\mathcal{L}(\mathcal{A})$. Let us define

$$M = \max\Big(\sup\big\{b_q \mid q \in Q, \alpha_{\|q\|}\text{is}\|q\| = b_q\big\}, \sup\big\{b_q \mid q \in Q, \alpha_{\|q\|}\text{is}\|q\| \geq b_q\big\}\Big).$$

Intuitively, in a run, M is the maximal number of occurrences of a given state in a run that is needed to satisfy the arithmetic constraints.

Then we define a new set of copies of states of Q,

$$\overline{Q} = \bigcup_{q\in Q, c(q)>0} \{q^1, \ldots, q^{c(q)}\}, \text{ and } Q' = \overline{Q} \times \Big(\{0, \ldots, M+1\}^Q\Big).$$

Hence a state of Q' is a pair $\langle q^i, \delta \rangle$ where $q \in Q$, $1 \leq i \leq c(q)$, $q^i \in \overline{Q}$ and δ is a mapping $\delta : \overline{Q} \to \{0, \ldots, M+1\}$.

The number $M + 1$ is needed to deal with constraints of the form $|q| = M$. If there are strictly more than M occurrences of q, then the term should not be accepted. So if more than M occurrences occur, it suffices to say that $M + 1$ copies occurred. Hence, when counting the number of occurrences of a given state, we will bound the count at $M + 1$.

The intuition behind the definition of states of Q' is that the index i in q^i is the number (up to $M + 1$) of terms encountered so far in the

bottom-up computation of the automaton (for the evaluation of $\|q\|$), and $\delta(q)$ is the number (up to $M+1$) of occurrences of q encountered so far in the computation (for the evaluation of $|q|$).

We define the sum m_1+m_2 of two elements $m_1, m_2 \in \{0,\ldots,M+1\}$ as the result of the addition of m_1 and m_2 in $\mathbb{N}$ if it is lower or equals to M, or as $M+1$ otherwise (if the result of the addition in $\mathbb{N}$ is greater than $M+1$).

Then, we define the constraints, transitions rules, and final states of $\mathcal{A}'$.
For each atomic constraint c occurring in C we define a conjunction of atomic constraints $const(c)$ on Q' as follow:

- if $c = (q_1 \approx q_2)$, then $const(c) = \bigwedge_{1\leq i\leq c(q_1),1\leq j\leq c(q_2)}, \langle q_1^i, \delta_1\rangle \approx \langle q_2^j, \delta_2\rangle$
- if $c = (q_1 \not\approx q_2)$, then $const(c) = \bigwedge_{1\leq i\leq c(q_1),1\leq j\leq c(q_2)}, \langle q_1^i, \delta_1\rangle \not\approx \langle q_2^j, \delta_2\rangle$.
- if $c = (\|q\| = b_q)$ or $c = (\|q\| \geq b_q)$, then $const(c) = \left(\bigwedge_{1\leq i\leq b_q}\langle q^i, \delta_1\rangle \approx \langle q^i, \delta_2\rangle\right) \wedge \left(\bigwedge_{1\leq i<j\leq b_q}\langle q^i, \delta_1\rangle \not\approx \langle q^j, \delta_2\rangle\right)$.

We define $C'_+ = \bigwedge_{c\in C'} const(c)$, where $c \in C$ means that c is an atomic constraint occurring in C.
The set of transition rules Δ' contains all the transition rules $f(\langle q_1^{i_1}, \delta_1\rangle, \ldots, \langle q_n^{i_n}, \delta_n\rangle) \rightarrow \langle q^i, \delta\rangle$ such that $q_1,\ldots, q_n, q \in Q$, $f(q_1,\ldots,q_n) \rightarrow q \in \Delta$, for all $j \leq n$, $1 \leq i_j \leq c(q_j)$, $i = i_1 + \ldots i_n$, for all $q' \in Q \setminus \{q\}$, $\delta(q') = \sum_{1\leq j\leq n} \delta_j(q')$, and $\delta(q) = 1 + \sum_{1\leq j\leq n} \delta_j(q)$.
The set of final states F' contains the states $\langle q_{\mathsf{f}}^i, \delta\rangle$, such that $q_{\mathsf{f}} \in F$ and δ satisfies the followings conditions for all $q \in Q$

- if $\alpha_{|q|}$ is $|q| = a_q$ then $\sum_{1\leq i\leq c(q)} \delta(q^i) = a_q$,
- if $\alpha_{|q|}$ is $|q| \geq a_q$ then $\sum_{1\leq i\leq c(q)} \delta(q^i) \geq a_q$ or $\sum_{1\leq i\leq c(q)} \delta(q^i) = M+1$,
- if $\alpha_{\|q\|}$ is $\|q\| = b_q$ or $\|q\| \geq b_q$, then $\forall 1 \leq i \leq b_q, \delta(q^i) \geq 1$ or $\delta(q^i) = M+1$.

We now have to prove the correctness of this construction.

Lemma 5.8 *For all $t \in \mathcal{T}(\Sigma)$ there exists an accepting run r of $\mathcal{A}$ on t iff there exists an accepting run r' of $\mathcal{A}'$ on t.*

proof. only if direction : Let r be an accepting run of $\mathcal{A}$ on t, such that $r \models C$. First, we compute a labelled tree $\overline{r} : \mathcal{P}os(t) \rightarrow \overline{Q}$ such that for all $p \in \mathcal{P}os(t)$, $\overline{r}(p)$ is a copy $q^i \in \overline{Q}$ of the state $r(p) = q$.
For every $q \in Q$, if $\alpha_{\|q\|}$ is $\top$, then there is only one copy q^1 of q in $\overline{Q}$. So, for all p with $r(p) = q$ we define $\overline{r}(p) = q^1$. If $\alpha_{\|q\|}$ is $\|q\| = b_q$ (resp. $\alpha_{\|q\|}$ is $\|q\| \geq b_q$) there exists b_q (resp $b_q + 1$) copies of q. Since $r \models \varphi$, there

exists exactly (resp. at least) b_q different terms $t_1, \ldots, t_{b_q}$ such that there exists b_q positions $p_1, \ldots, p_{b_q}$ verifying $r(p_1) = \cdots = r(p_{b_q}) = q$ and $t|_{p_1} = t_1, \ldots, t|_{p_{b_q}} = t_{b_q}$. For every position $p \in \mathcal{P}os(t)$ such that $r(p) = q$ and $t|_p = t_i$, we define $\overline{r}(p) = q^i$. And, if $\alpha_{\|q\|}$ is $\|q\| \geq b_q$, for every $p \in \mathcal{P}os(t)$ such that $r(p) = q$ and $t|_p$ is different from all the $t_1, \ldots, t_{b_q}$, we define $\overline{r}(p) = q^{b_q+1}$.

Then, we define $r' : \mathcal{P}os(t) \to Q'$ as follows. For all position $p \in \mathcal{P}os(t)$, $r'(p) = \langle \overline{r}(p), \delta_p \rangle$ where for all $q \in Q$, $\delta_p(q) = [\![\, |q| \,]\!]_{\overline{r}|_p}$ if $[\![\, |q| \,]\!]_{\overline{r}|_p} \leq M$ and $\delta_p(q) = M + 1$ otherwise. It is easy to see that the rules of Δ' allow us to define such a run of $\mathcal{A}'$ on t. We now have to check that the constraint C' is satisfied by r' and t.

This construction satisfies the atomic constraints $\langle q_1^i, \delta_1 \rangle \approx \langle q_2^j, \delta_2 \rangle$ (resp. $\langle q_1^i, \delta_1 \rangle \not\approx \langle q_2^j, \delta_2 \rangle$), that were added to C' because every constraint $q_1 \approx q_2$ (resp. $q_1 \not\approx q_2$) was already in C. Whatever copies q_1^i and q_1^j we have chosen in $\overline{r}$ in order to replace occurrences of q_1 and q_2 at positions p_1 and p_2 in r, we know that those constraints will be respected because C already implied that $t|_{p_1} = t_{p_2}$ (resp. $t|_{p_1} \neq t|_{p_2}$).

For every state $q \in Q$ such that $\alpha_{\|q\|}$ is $\|q\| = b_q$, or $\alpha_{\|q\|}$ is $\|q\| \geq b_q$, there are constraints of two types: $\langle q^i, \delta_1 \rangle \not\approx \langle q^j, \delta_2 \rangle$ for $1 \leq i < j \leq b_q$, for all δ_1, δ_2 and $\langle q^i, \delta_1 \rangle \approx \langle q^i, \delta_2 \rangle$ for $1 \leq i \leq b_q$ and for all δ_1, δ_2. By construction of r', for all $1 \leq i \leq b_q$ all the positions p such that $r'(p) = \langle q^i, \delta \rangle$, for all δ, have the same subterm $t|_p = t_i$. Moreover, for all $1 \leq i < j \leq b_q$ we have that $t_i \neq t_j$. So both types of constraints are respected.

Now we have to make sure that r' is an accepting run, that is that $r'(\varepsilon) \in F'$. Since r is an accepting run of $\mathcal{A}$ we know that $r'(\varepsilon) = \langle q_{\mathsf{f}}^i, \delta \rangle$ where $q_{\mathsf{f}} \in F$ and $\delta(q') = [\![\, |q'| \,]\!]_{r'}$ if $[\![\, |q'| \,]\!]_{r'} \leq M$ and $|q'| = M + 1$ otherwise. Since $q_{\mathsf{f}} \in F$ we just have to make sure that δ satisfies the conditions. By construction of r', every occurrence of a state $q \in Q$ in r corresponds to an occurrence of one of the copies $q^i \in Q'$ in r'. So, for all $q \in Q$ it holds that $[\![\, |q| \,]\!]_r$ is equals to $\sum_{1 \leq i \leq c(q)} [\![\, |q^i| \,]\!]_{r'}$ and that $\sum_{1 \leq i \leq c(q)} \delta(q^i)$ is either $[\![\, |q| \,]\!]_r$ if $[\![\, |q| \,]\!]_r \leq M$ or $M + 1$ otherwise. Hence, if $\alpha_{|q|}$ is $|q| = a_q$ or $|q| \geq a_q$, then the associated conditions on δ are satisfied since φ (and in particular $\alpha_{|q|}$) is satisfied by r. By construction, for every $q \in Q$ such that $\alpha_{\|q\|}$ is $\|q\| = b_q$ or $\|q\| \geq b_q$, we have instantiated each copy q^i for all $1 \leq i \leq b_q$ at least once in r'. Hence the associated conditions are also respected: r' is an accepting of $\mathcal{A}'$ on t.

if direction : Let r' be an accepting run of $\mathcal{A}'$ on t. We define a run r of $\mathcal{A}$ on t in the following way: for all $p \in \mathcal{P}os(t)$, $r(p)$ is the original state $q \in Q$ corresponding to the copy q^i such that $r'(p) = \langle q^i, \delta \rangle$. By construction of $\mathcal{A}'$ we have that r is a valid run *wrt* Δ and that $r(\varepsilon) \in F$. For every positions $p_1, p_2 \in \mathcal{P}os(t)$ with $r(p_1) = q_1$, $r(p_2) = q_2$, such that there exists a constraint $q_1 \approx q_2$ (resp. $q_1 \not\approx q_2$), there was, by construction of $\mathcal{A}'$, a constraint $\langle q_1^i, \delta_1 \rangle \approx \langle q_2^j, \delta_2 \rangle$ (resp. $\langle q_1^i, \delta_1 \rangle \not\approx \langle q_2^j, \delta_2 \rangle$) for all copies q_1^i of q_1 and q_2^j of q_2 and for all δ_1, δ_2. In particular, there is in $\mathcal{A}'$ the

constraint $r'(p_1) \approx r'(p_2)$ (resp. $r'(p_1) \not\approx r'(p_2)$), so $t|_{p_1} = t|_{p_2}$ (resp. $t|_{p_1} \neq t|_{p_2}$). Hence, all the constraints of $\mathcal{A}$ are satisfied by r and t.

We now have to ensure that the arithmetic constraints in C are satisfied by r and t. As above, by construction of r', it is easy to see that for all $q \in Q$ we have that $\sum_{1 \leq i \leq c(q)} \delta(q^i)$ is either $[\![\, |q| \,]\!]_r$ if $[\![\, |q| \,]\!]_r \leq M$ or $M+1$ otherwise. Hence, all the expressions $\alpha_{|q|}$ of C are satisfied. And for every expression $\alpha_{\|q\|}$ equal to $\|q\| = b_q$, we have exactly b_q copies of q in $\overline{Q}$. The constraints $\langle q^i, \delta_1 \rangle \approx \langle q^i, \delta_2 \rangle$ and $\langle q^i, \delta_1 \rangle \not\approx \langle q^j, \delta_2 \rangle$ for all δ_1, δ_2, $i \neq j$ ensure that each copy q^i is used for only one subterm t_i and that, for all $1 \leq i < j \leq b_q$ $t_i \neq t_j$. Since all the copies q^i are guaranteed to occur at least once in r', by the arithmetic constraints, we know that we have exactly b_q different subterms at the positions labelled by q. And if $\alpha_{\|q\|}$ is $\|q\| \geq b_q$, the extra copy q^{b_q+1} recognizes all the extra subterms that are recognized in q if $\|q\| > b_q$ is satisfied by r and t. Hence, C is fully satisfied by r and t, and r is a successful run of $\mathcal{A}$ on t. □ □

We can know construct a PCTAGC$[\approx, \not\approx]$ equivalent to a given PTAGC$[\approx, \not\approx, \mathbb{N}]$. The missing step is to get such an automaton from a TAGC$[\approx, \not\approx, \mathbb{N}]$.

Proposition 5.9 *For all TAGC$[\approx, \not\approx, \mathbb{N}]$ $\mathcal{A}$, one can effectively construct a PTAGC$[\approx, \not\approx, \mathbb{N}]$ $\mathcal{A}'$ s.t. $\mathcal{L}(\mathcal{A}') = \mathcal{L}(\mathcal{A})$.*

proof. Let $\mathcal{A} = \langle Q, \mathcal{F}, F, C, \Delta \rangle$. We can assume *wlog* that $C = C_+ \wedge C_- \wedge C_{\mathbb{N}}$ where C_+ is a conjunction of atomic constraints of the form $q \approx q'$ or $q \not\approx q'$, C_- is a conjunction of negations of atomic constraints of the form $\neg(q \approx q')$ or $\neg(q \not\approx q')$, and $C_{\mathbb{N}}$ is a conjunction of natural linear inequalities. Otherwise, we can put C is disjunctive normal form $C = \bigvee_{i=1}^{n} C_i$, where every C_i has the above form, apply the transformation below to each $\mathcal{A}_i = \langle Q, F, C_i, \Delta \rangle$, obtaining $\mathcal{A}'_i$ and let $\mathcal{A}'$ be the disjoint union of the $\mathcal{A}'_i$'s.

We construct below a PTAGC$[\approx, \not\approx, \mathbb{N}]$ $\mathcal{A}' = \langle Q', \mathcal{F}, F', C', \Delta' \rangle$ recognizing $\mathcal{L}(\mathcal{A})$. The idea for the construction of $\mathcal{A}'$ is to replace the negative constraints of C_- by positive constraints and arithmetic constraints, using copies of states of Q. For instance, assume that C_- contains only $\neg(q_1 \approx q_2)$. By definition, for every successful run r of $\mathcal{A}$ on a term $t \in \mathcal{T}(\mathcal{F})$, there exist two positions $p_1, p_2 \in \mathcal{P}os(t)$ such that $r(p_1) = q_1$, $r(p_2) = q_2$ and $t|_{p_1} \neq t|_{p_2}$. For expressing this property without the negative constraint, we add two copies q'_1, q'_2 of respectively q_1 and q_2 (*i.e.* we let $Q' = Q \cup \{q'_1, q'_2\}$) and define Δ' by adding to Δ all the transitions obtained from Δ by replacing at least one occurrence of q_1 by q'_1 or one occurrence of q_2 by q'_2. We define similarly C'_+ and $C'_{\mathbb{N}}$ from C_+ and $C_{\mathbb{N}}$. Also let $F' = F \cup \{q'_i \mid q_i \in F, i = 1, 2\}$. Finally, we let $C' = C'_+ \wedge q'_1 \not\approx q'_2 \wedge C'_{\mathbb{N}} \wedge |q'_1| = 1 \wedge |q'_2| = 1$.

In general, when C_- contains several negative constraints, we have to take care of the multiple occurrences of states in the different negative constraints of C_-. For instance, let $C_- = \neg(q_1 \approx q_2) \wedge \neg(q_1 \approx q_3)$. For

a successful run r of $\mathcal{A}$ on t, there exist positions $p_1^1, p_2, p_1^2, p_3 \in \mathcal{P}os(t)$ such that $r(p_1^1) = r(p_1^2) = q_1$, $r(p_2) = q_2$, $r(p_3) = q_3$ and and $t|_{p_1^1} \neq t|_{p_2}$ and $t|_{p_1^2} \neq t|_{p_3}$. In order to replace C_- by positive constraints as above, we must decide how many copies of q_1, q_2 and q_3 we will need. If $p_1^1 \neq p_1^2$, then we can choose two copies q_1^1 and q_1^2 of q_1, that will reach respectively p_1^1 and p_1^2 in a computation of $\mathcal{A}'$ on t. With constraints $q_1^1 \not\approx q_1^2$, $|q_1^1| = 1$ and $|q_1^2| = 1$ as above, we are done. However, if $p_1^1 = p_1^2$, then we must have only one copy of q_1, because only one state can reach this position in a computation of $\mathcal{A}'$ on t. We shall enumerate below all the cases of the number of copies needed for each state, using the number of finite models of a first-order formula.

Let $Q = \{q_1, \ldots, q_p\}$ and let $C_- = d_1 \wedge \ldots \wedge d_m \wedge e_1 \wedge \ldots \wedge e_n$ where for all $k \leq m$, $d_k = \neg(q_{u_k} \approx q_{u'_k})$ and for all $\ell \leq n$, $e_\ell = \neg(q_{v_\ell} \not\approx q_{v'_\ell})$. Let us associate the following closed first order formula to $\mathcal{A}$:

$$\begin{array}{rcl} \phi_{\mathcal{A}} & = & \exists x_{u_1}^{d_1}, y_{u'_1}^{d_1}, \ldots x_{u_m}^{d_m}, y_{u'_m}^{d_m}, x_{v_1}^{e_1}, y_{v'_1}^{e_1}, \ldots x_{v_m}^{e_n}, y_{v'_n}^{e_n}. \\ & & \bigwedge_{k=1}^{m} x_{u_k}^{d_k} \neq y_{u'_k}^{d_k} \wedge \bigwedge_{\ell=1}^{n} x_{v_\ell}^{e_\ell} = y_{v'_\ell}^{e_\ell} \end{array}$$

Let N be the number of variables in $\phi_{\mathcal{A}}$. Every variable of $\phi_{\mathcal{A}}$ is uniquely associated to an occurrence of a state of Q in C_-. For instance, $x_{u_k}^{d_k}$ (resp. $y_{u'_k}^{d_k}$) is associated to the occurrence of q_{u_k} is the left (resp. right) hand side of d_k. For a $\mathcal{A}$ with the above example of C_-, we have $\phi_{\mathcal{A}} = \exists x_1^{d_1}, y_2^{d_1}, x_1^{d_2}, y_4^{d_2} \,.\, x_1^{d_1} \neq y_2^{d_1} \wedge x_1^{d_1} \neq y_3^{d_1}$.

For each $1 \leq i \leq p$, we define X_i as the set of variables associated to occurrences of q_i: $X_u = \{x_{u_k}^{d_k} \mid 1 \leq k \leq m, u_k = i\} \cup \{y_{u'_k}^{d_k} \mid 1 \leq k \leq m, u'_k = i\} \cup \{x_{v_\ell}^{e_\ell} \mid 1 \leq \ell \leq n, v_\ell = i\} \cup \{y_{v'_\ell}^{e_\ell} \mid 1 \leq \ell \leq n, v'_\ell = i\}$.

It is clear that $\phi_{\mathcal{A}}$ is satisfiable iff it admits a finite model with at most N elements. Let D be a set of N distinct elements. A solution of $\phi_{\mathcal{A}}$ is a mapping σ from $\{x_{u1}^{d_1}, y_{u'_1}^{d_1}, \ldots\}$ into D which makes true the conjunction of $\phi_{\mathcal{A}}$. Let $sol(\phi_{\mathcal{A}})$ be the set of solutions of $\phi_{\mathcal{A}}$. To each $\sigma \in sol(\phi_{\mathcal{A}})$, we associate a PTAGC$[\approx, \not\approx, \mathbb{N}]$ $\mathcal{A}_\sigma = \langle Q_\sigma, F_\sigma, C_\sigma, \Delta_\sigma \rangle$, where Q_σ contains the states of Q and, for each $i \leq p$, $n_i = |\sigma(X_i)|$ copies of q_i: $q_i^1, \ldots, q_i^{n_i}$. The transition set Δ_σ contains all the transitions of Δ plus additional transitions obtained from Δ by replacing at least one occurrence of one of the q_i by one of its copies q_i^j. The final states set is defined by $F_\sigma = \{q_i^j \mid q_i^j \in Q_\sigma, q_i \in F\}$. Finally, we let

$$C_\sigma = C_+^\sigma \wedge C_{\mathbb{N}}^\sigma \wedge \bigwedge_{\substack{i \leq p \\ j \leq n_i}} \left(|q_i^j| = 1 \wedge \bigwedge_{\substack{j' \leq n_i \\ j' \neq j}} q_i^j \not\approx q_i^{j'} \right)$$

where C_+^σ contains all the positive constraints of C_+ where some (possibly zero or more) instances of states of Q are replaced by copies, and $C_{\mathbb{N}}^\sigma$ is obtained from $C_{\mathbb{N}}$ by replacement of every $|q_i|$ by $|q_i| + |q_i^1| + \ldots + |q_i^{n_i}|$ and of every $\|q_i\|$ by $\|q_i\| + \|q_i^1\| + \ldots + \|q_i^{n_i}\|$. Finally, let us rename the respective states of the $\mathcal{A}_\sigma$ such that there states sets are pairwise

disjoint, and let $\mathcal{A}'$ be the disjoint union of all the $\mathcal{A}_\sigma$ (constructed as in Proposition 2.11). It is clear that $\mathcal{A}'$ is positive. Let us show that $\mathcal{L}(\mathcal{A}') = \mathcal{L}(\mathcal{A})$.

Lemma 5.10 $\mathcal{L}(\mathcal{A}') = \mathcal{L}(\mathcal{A})$.

proof. Let $t \in \mathcal{L}(\mathcal{A}')$. By construction, there exists a solution $\sigma \in sol(\phi_\mathcal{A})$ such that $t \in \mathcal{L}(\mathcal{A}_\sigma)$. Let r' be a successful run of $\mathcal{A}_\sigma$ on t. By definition of C_σ, for each $i \leq p$, there exists $n_i = |\sigma(X_i)|$ copies of q_i: $q_i^1, \ldots, q_i^{n_i}$, and each copy q_i^j occurs exactly once in r' at a position called p_i^j (*i.e.* $r^{-1}(q_i^j) = \{p_i^j\}$). Moreover, still by definition of C_σ, the subterms $t|_{p_i^j}$ are pairwise disjoint. The mapping θ which associate to each variable in X_i the corresponding position p_i^j is isomorphic to σ. Let r be obtained from r' by replacement of every subrun $r'|_{p_i^j}$ by a run of $ta(\mathcal{A})$ headed by q_i. It is clear by construction of Δ_σ that r is a run of $ta(\mathcal{A})$. Moreover, following the property of θ above, $t, r \models C_-$ and $t, r \models C_+ \wedge C_\mathbb{N}$. It follows that r is a successful run of $\mathcal{A}$ on t.

Conversely, let $t \in \mathcal{L}(\mathcal{A})$, and let r be a successful run of $\mathcal{A}$ on t. By definition, for every $d_k = \neg(q_{u_k} \approx q_{u'_k})$ in C_-, there exists two positions $p_{u_k}^{d_k}$ and $s_{u_k}^{d_k}$ in $\mathcal{P}os(t)$ such that $t|_{p_{u_k}^{d_k}} \neq t|_{s_{u'_k}^{d_k}}$, and for every $e_\ell = \neg(q_{v_\ell} \not\approx q_{v'_\ell})$ in C_- there exists two positions $p_{v_\ell}^{e_\ell}$ and $s_{v'_\ell}^{e_\ell}$ in $\mathcal{P}os(t)$ such that $t|_{p_{v_\ell}^{e_\ell}} = t|_{s_{v'_\ell}^{e_\ell}}$. The set of subterms of t at these positions define therefore a solution σ of $\phi_\mathcal{A}$. We can construct from r a run successful run r' of $\mathcal{A}_\sigma$ on t. The principle of the construction is to replace every subrun of r at the position *e.g.* $p_{u_k}^{d_k}$ by a run of $\mathcal{A}_\sigma$ headed by a copy of q_{u_k}. Therefore, $t \in \mathcal{L}(\mathcal{A}_\sigma) \subseteq \mathcal{L}(\mathcal{A}')$. □ □

Since TAGC[$\approx, \not\approx$] are a particular case of TAGC[$\approx, \not\approx, \mathbb{N}$], one can construct an equally expressive PCTAGC. Indeed, given a TAGC[$\approx, \not\approx$] $\mathcal{A}$, Proposition 5.9 permits to construct a PTAGC[$\approx, \not\approx, \mathbb{N}$] $\mathcal{B}'$ with $\mathcal{L}(\mathcal{B}') = \mathcal{L}(\mathcal{A})$ and then Proposition 5.5 permits to construct a PTAGC $\mathcal{B}$ with $\mathcal{L}(\mathcal{B}) = \mathcal{L}(\mathcal{B}') = \mathcal{L}(\mathcal{A})$. Using DNF for the constraints of $\mathcal{B}$ and the construction of Proposition 2.11 for union (which preserves the property of being PCTAGC), we obtain a PCTAGC$\mathcal{A}'$ such that $\mathcal{L}(\mathcal{A}') = \mathcal{L}(\mathcal{A})$. Hence the following corollary.

Corollary 5.11 *Given a TAGC $\mathcal{A}$, one can effectively construct a PCTAGC $\mathcal{A}'$such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$.*

The complexity of this effective construction has not been studied. However, the constructions used for the propositions and lemma of this section are all primitive recursive. The decidability procedure that we will present in next section for PCTAGC implies a non-primitive recursive complexity. Regarding this, we can consider to have the translation from TAGC into PCTAGC for free.

2 Emptiness Decision Algorithm

In this section we prove the decidability of the *emptiness* problem for TAGC. Using the corollary 5.11 it suffices to prove the decidability of the emptiness problem for PCTAGC.

The decidability of emptiness for PCTAGC is proved in three steps. In subsection 2.1, we present a new notion of pumping which allows to transform a run into a smaller run under certain conditions. In subsection 2.2, we define a well quasi-ordering $\leq$ on a certain set S. In subsection 2.3, we connect the two previous subsections by describing how to compute, for each run r with height $h = h(r)$, a sequence $e_h, \ldots, e_0$ of elements of S satisfying the following fact: there exists a pumping on r if and only if $e_i \leq e_j$ for some $h \geq i > j \geq 0$. Finally, all of these constructions are used as follows. Suppose the existence of an accepting run r on a term t. If r is "too high", the fact that $\leq$ is a well quasi-ordering and the property of the sequence imply the existence of such i, j. Thus, it follows the existence of a pumping providing a smaller accepting run r'. We conclude the existence of a computational bound for the height of an accepting run, and hence, decidability of emptiness.

2.1 Global Pumpings

Pumping lemma is a traditional concept in automata theory, and in particular, they are very useful to reason about tree automata. The basic idea is to convert a given run r into another run by replacing a subrun at a certain position p in r by a run r', thus obtaining a run $r[r']_p$. Pumpings are useful for deciding emptiness: if a "big" run can always be reduced by a pumping, then decision of emptiness is obtained by a search of an accepting "small" run.

For plain tree automata, a necessary and sufficient condition to ensure that $r[r']_p$ is a run is that the resulting states of $r|_p$ and r' coincide, since the correct application of a rule at a certain position depends only on the resulting states of the subruns of the direct children. In this case, an accepting run with height bounded by the number of states exists, whenever the accepted language is not empty.

When the tree automaton has global equality and disequality constraints, the constraints may be falsified when replacing a subrun by a new run. For PCTAGC, we will define a notion of pumping ensuring that the constraints are satisfied. This notion of pumping requires to perform several replacements in parallel. We first define the sets of positions involved in such a kind of pumping.

Definition 5.12 *Let t be a term of $\mathcal{T}(\Sigma)$. Let i be an integer between 0 and $h(t)$. We define H_i as $\{p \in \mathcal{P}os(t) \mid h(t|_p) = i\}$ and $\check{H}_i$ as $\{p.j \in \mathcal{P}os(t) \mid h(t|_{p.j}) < i \wedge h(t|_p) > i\}$.*

Example 5.13 *According to Definition 5.12, for our running example (Example 2.6), we have the H_i and $\check{H}_i$ presented in Figure 5.2.* $\diamond$

i	H_i	$\check{H}_i$
5	$\{\varepsilon\}$	$\emptyset$
4	$\{3\}$	$\{1,2\}$
3	$\{3.3\}$	$\{1,2,3.1,3.2\}$
2	$\{3.3.3\}$	$\{1,2,3.1,3.2,3.3.1,3.3.2\}$
1	$\{2,3.2,3.3.2,3.3.3.2\}$	$\{1,3.1,3.3.1,3.3.3.1\}$
0	$\{1,2.1,2.2,3.1,3.2.1,$ $3.2.2,3.3.1,3.3.2.1,3.3.2.2,$ $3.3.3.1,3.3.3.2.1,3.3.3.2.2\}$	$\emptyset$

Figure 5.2: H_i and $\check{H}_i$ (Example 5.13).

The following lemma is rather straightforward from the previous definition.

Lemma 5.14 *Let t be a term of $\mathcal{T}(\Sigma)$. Let i be an integer between 0 and $h(r)$. Then, any two different positions in $H_i \cup \check{H}_i$ are parallel, and for any arbitrary position p in $\mathcal{P}os(r)$ there is a position $\bar{p}$ in $H_i \cup \check{H}_i$ such that, either p is a prefix of $\bar{p}$, or $\bar{p}$ is a prefix of p.*

proof. For the first fact, note that any proper prefix p of a position $\bar{p}$ in $H_i \cup \check{H}_i$ satisfies $h(t|_p) > i$. Thus, such a p is not in $H_i \cup \check{H}_i$. For the second fact, consider any p in $\mathcal{P}os(t)$. If $h(t|_p) \leq i$ holds, then the smallest position $\bar{p}$ satisfying $\bar{p} < p$ and $h(t|_{\bar{p}}) \leq i$ is in $H_i \cup \check{H}_i$, and we are done. Otherwise, if $h(t|_p) > i$ holds, then the smallest position $\bar{p}$ of the form $p.1.\ldots.1$ and satisfying $h(t|_{\bar{p}}) \leq i$ is in $H_i \cup \check{H}_i$, and we are done. □

Definition 5.15 *Let $\mathcal{A}$ be a PCTAGC. Let r be a run of $\mathcal{A}$ on a term t. Let i, j be integers satisfying $0 \leq j < i \leq h(r)$. A* pump-injection *$I : (H_i \cup \check{H}_i) \to (H_j \cup \check{H}_j)$ is an injection function such that the following conditions hold:*

(C_1) $I(H_i) \subseteq H_j$ and $I(\check{H}_i) \subseteq \check{H}_j$.

(C_2) For each $\bar{p}$ in $H_i \cup \check{H}_i$, $r(\bar{p}) = r(I(\bar{p}))$.

(C_3) For each $\bar{p}_1, \bar{p}_2$ in $H_i \cup \check{H}_i$, $(t|_{\bar{p}_1} = t|_{\bar{p}_2}) \Leftrightarrow (t|_{I(\bar{p}_1)}) = t|_{I(\bar{p}_2)}))$.

Let $\{\bar{p}_1, \ldots, \bar{p}_n\}$ be $H_i \cup \check{H}_i$ more explicitly written. The pair formed by the run $r[r|_{I(\bar{p}_1)}]_{\bar{p}_1} \ldots [r|_{I(\bar{p}_n)}]_{\bar{p}_n}$ and the term $t[t|_{I(\bar{p}_1)}]_{\bar{p}_1} \ldots [t|_{I(\bar{p}_n)}]_{\overline{(p)}_n}$ is called a global pumping *on r and t with indexes i, j, and injection I.*

By the condition C_2, $r[r|_{I(\bar{p}_1)}]_{\bar{p}_1} \ldots [r|_{I(\bar{p}_n)}]_{\bar{p}_n}$ is clearly a run of $ta(\mathcal{A})$ on $t[t|_{I(\bar{p}_1)}]_{\bar{p}_1} \ldots [t|_{I(\bar{p}_n)}]_{\overline{(p)}_n}$, but it is still necessary to prove that it is a run of $\mathcal{A}$. By abuse of notation, when we write $r[r|_{I(\bar{p}_1)}]_{\bar{p}_1} \ldots [r|_{I(\bar{p}_n)}]_{\bar{p}_n}$, or $t[t|_{I(\bar{p}_1)}]_{\bar{p}_1} \ldots [t|_{I(\bar{p}_n)}]_{\overline{(p)}_n}$ we sometimes consider that I and $\{\bar{p}_1, \ldots, \bar{p}_n\}$ are still explicit, and say that it is a global pumping with some indexes $0 \leq j < i \leq h(r)$.

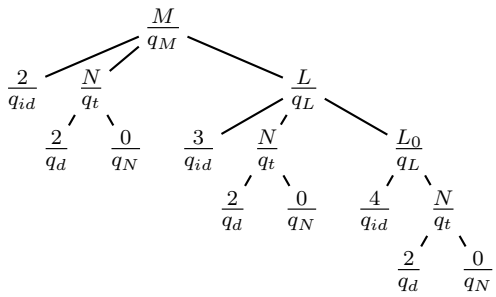

Figure 5.3: Pump-injection of Example 5.16.

Example 5.16 *Following our running example, we define a pump-injection* $I : (H_4 \cup \check{H}_4) \to (H_3 \cup \check{H}_3)$ *as follows:* $I(1) = 3.1$, $I(2) = 2$, $I(3) = 3.3$. *We note that* I *is a correct pump-injection:* $I(H_4) \subseteq H_3$ *and* $I(\check{H}_4) \subseteq \check{H}_3$ *hold, thus* (C_1) *holds. For* (C_2), *we have* $r(1) = r(I(1)) = q_{id}$, $r(2) = r(I(2)) = q_t$, *and* $r(3) = r(I(3)) = q_L$. *Regarding* (C_3), *for each different* $\bar{p}_1, \bar{p}_2$ *in* $H_4 \cup \check{H}_4$, $t|_{\bar{p}_1} \neq t|_{\bar{p}_2}$ *and* $t|_{I(\bar{p}_1)} \neq t|_{I(\bar{p}_2)}$ *hold.*

After applying the pump-injection I, *we obtain the term* t' *and run* r' *of Figure 5.3.* ◇

Our goal is to prove that any global pumping $r[r|_{I(\bar{p}_1)}]_{\bar{p}_1} \dots [r|_{I(\bar{p}_n)}]_{\bar{p}_n}$ is a run on $t[t|_{I(\bar{p}_1)}]_{\bar{p}_1} \dots [t|_{I(\bar{p}_n)}]_{\bar{(p)}_n}$, and in particular, that all global equality and disequality constraints are satisfied. To this end we first state the following intermediate statement, which determines the height of the terms pending at some positions after the pumping action.

Lemma 5.17 *Let* $\mathcal{A}$ *be a PCTAGC. Let* r *be a run of* $\mathcal{A}$ *on a term* t. *Let* $\langle r', t' \rangle$ *be the global pumping on* r *and* t *with indexes* $0 \leq j < i \leq h(r)$ *and injection* I, *such that* $r' = r[r|_{I(\bar{p}_1)}]_{\bar{p}_1} \dots [r|_{I(\bar{p}_n)}]_{\bar{p}_n}$ *and* $t' = t[t|_{I(\bar{p}_1)}]_{\bar{p}_1} \dots [t|_{I(\bar{p}_n)}]_{\bar{(p)}_n}$, *Let* $k \geq 0$ *be a natural number and let* p *be a position of* t *such that* $h(t|_p)$ *is* $i + k$. *Then,* p *is also a position of* t' *and* $h(t'|_p)$ *is* $j + k$.

proof. Position p is obviously a position of t' since no position in $H_i \cup \check{H}_i$ is a proper prefix of p. We prove the second part of the statement by induction on k. First, assume that $k = 0$. Then, $h(t|_p)$ is i. Thus, p is in H_i, say p is $\bar{p}_1$. Therefore, $t'|_p$ is $t|_{I(\bar{p}_1)}$. By Condition (C_1) of the definition of pump-injection, $I(\bar{p}_1) \in H_j$ holds. Hence, $h(t'|_p) = h(t|_{I(\bar{p}_1)}) = j$.

Now, assume that $k > 0$. Let m be the arity of $t(p)$. Thus, $p.1, \dots, p.m$ are all the child positions of p in t. Since $h(t|_p)$ is $i + k$, all $h(t|_{p.1}), \dots, h(t|_{p.m})$ are smaller than or equal to $i + k - 1$, and at least one of them is equal to $i + k - 1$.

Consider any α in $\{1, \dots, m\}$. If $h(t|_{p.\alpha})$ is $i + k'$ for some $0 \leq k' \leq k - 1$, then, by induction hypothesis, $h(r'|_{p.\alpha})$ is $j + k'$. Otherwise, if $h(r|_{p.\alpha})$ is strictly smaller than i, then $p.\alpha$ is one of the positions in $\check{H}_i$, say $\bar{p}_1$. Moreover, $t'|_{\bar{p}_1}$ is $t|_{I(\bar{p}_1)}$, and by Condition (C_1) of the definition

of I, $I(\bar{p}_1)$ belongs to $\check{H}_j$. Therefore, $h(t|_{I(\bar{p}_1)}) < j$ holds, and hence, $h(t'|_{p.\alpha}) = h(t'|_{\bar{p}_1}) = h(t|_{I(\bar{p}_1)}) < j \leq j + k - 1$ holds.

From the above cases we conclude that, if $h(t|_{p.\alpha})$ is $i + k - 1$, then $h(t'|_{p.\alpha})$ is $j+k-1$, and if $h(t|_{p.\alpha})$ is smaller than $i+k-1$, then $h(t'|_{p.\alpha})$ is smaller than $j + k - 1$. It follows that all $h(t'|_{p.1}), \ldots, h(t'|_{p.m})$ are smaller than or equal to $j + k - 1$, and at least one of them is equal to $j + k - 1$. As a consequence, $h(t'|_p)$ is $j + k$. □

Corollary 5.18 *Let $\mathcal{A}$ be a PCTAGC. Let r be a run of $\mathcal{A}$ on a term t. Let $\langle r', t' \rangle$ be a global pumping of r and t. Then, $h(t') < h(t)$.*

The following lemma states that equality and disequality relations are preserved, not only for terms pending at the positions of the domain of I, but also for terms pending at prefixes of positions of such domain.

Lemma 5.19 *Let $\mathcal{A}$ be a PCTAGC. Let r be a run of $\mathcal{A}$ on a term t. Let $\langle r', t' \rangle$ be the global pumping with indexes $0 \leq j < i \leq h(r)$ and injection I, such that $r' = r[r|_{I(\bar{p}_1)}]_{\bar{p}_1} \ldots [r|_{I(\bar{p}_n)}]_{\bar{p}_n}$ and $t' = t[t|_{I(\bar{p}_1)}]_{\bar{p}_1} \ldots [t|_{I(\bar{p}_n)}]_{(\bar{p})_n}$, Let p_1, p_2 be positions of p satisfying $h(t|_{p_1}), h(t|_{p_2}) \geq i$. Then, p_1, p_2 are also positions of t' and $(t|_{p_1} = t|_{p_2}) \Leftrightarrow (t'|_{p_1} = t'|_{p_2})$ holds.*

proof. The first statement follows by Lemma 5.17. We prove the second part of the statement by induction on $h(t|_{p_1}) + h(t|_{p_2})$. We distinguish the following cases:
i. Assume that $h(t|_{p_1}) \neq h(t|_{p_2})$. Then, $t|_{p_1} \neq t|_{p_2}$ hold and, moreover, $h(t|_{p_1}) = i + k_1$ and $h(t|_{p_2}) = i + k_2$ hold for some different natural numbers k_1 and k_2. By Lemma 5.17, $h(t'|_{p_1}) = j + k_1$ and $h(t'|_{p_2}) = j + k_2$. Thus, $t'|_{p_1} \neq t'|_{p_2}$ hold, and we are done.

ii. Assume that $h(t|_{p_1}) = h(t|_{p_2}) = i + k$ for some k. We start by assuming the case $k = 0$. Then, $h(t|_{p_1})$ is i. Thus, p_1, p_2 are in H_i, say p_1 is $\bar{p}_1$ and p_2 is $\bar{p}_2$. Therefore, $t'|_{p_1}$ is $t|_{I(\bar{p}_1)}$ and $t'|_{p_2}$ is $t|_{I(\bar{p}_2)}$. By Condition (C_3) of the definition of pump-injection, $(t|_{\bar{p}_1} = t|_{\bar{p}_2}) \Leftrightarrow (t|_{I(\bar{p}_1)} = t|_{I(\bar{p}_2)})$ holds. Thus, $(t|_{p_1} = t|_{p_2}) \Leftrightarrow (t'|_{p_1}) = t'|_{p_2}))$ holds and we are done.

Now, we assume that $k > 0$. Note that, in this case, $t'(p_1) = t(p_1)$ and $t'(p_2) = t(p_2)$ hold. In the case where $t(p_1)$ differs from $t(p_2)$), it is clear that $t|_{p_1}) \neq t|_{p_2})$ and $t'|_{p_1}) \neq t'|_{p_2}$ hold, and we are done. Hence, we consider the remaining case where $t(p_1) = t(p_2)$ holds. Let m be the arity of $t(p_1)$. Thus, $p_1.1, \ldots, p_1.m$ and $p_2.1, \ldots, p_2.m$ are all the child positions of p_1 and p_2 in t, respectively. In order to prove $(t|_{p_1} = t|_{p_2}) \Leftrightarrow (t'|_{p_1}) = t'|_{p_2})$ it suffices to prove $(t|_{p_1.\alpha} = t|_{p_2.\alpha}) \Leftrightarrow (t'|_{p_1.\alpha} = t'|_{p_2.\alpha})$ for all α in $\{1, \ldots, m\}$. Thus, we consider any of such α's and distinguish the following cases:

i.a. If $h(t|_{p_1.\alpha}), h(t|_{p_2.\alpha}) \geq i$ holds, then the result follows by induction hypothesis.

i.b. If $h(t|_{p_1.\alpha}) \geq i$ and $h(t|_{p_2.\alpha}) < i$, then $t|_{p_1.\alpha} \neq t|_{p_2.\alpha}$ holds, and moreover, $h(t|_{p_1.\alpha}) = i + k_1$ for some $k_1 \geq 0$, and $p_2.\alpha$ belongs to $\check{H}_i$. By Lemma 5.17, $h(t'|_{p_1.\alpha}) = j + k_1$ holds. By Condition (C_1) of the definition of pump-injection, $h(t'|_{p_2.\alpha}) < j$ holds. Thus, $t'|_{p_1.\alpha} \neq t'|_{p_2.\alpha}$ holds, and we are done.

i.c. The case where $h(t|_{p_1.\alpha}) < i$ and $h(t|_{p_2.\alpha}) \geq i$ hold is analogous to the previous one.

i.d. If $h(t|_{p_1.\alpha}), h(t|_{p_2.\alpha}) < i$, then $p_1.\alpha, p_2.\alpha$ belong to $\check{H}_i$, say $p_1.\alpha$ is $\bar{p}_1$ and $p_2.\alpha$ is $\bar{p}_2$. Therefore, $t'|_{p_1.\alpha}$ is $t|_{I(\bar{p}_1)}$ and $t'|_{p_2.\alpha}$ is $t|_{I(\bar{p}_2)}$. By Condition (C_3) of the definition of pump-injection, $(t|_{\bar{p}_1} = t|_{\bar{p}_2}) \Leftrightarrow (t|_{I(\bar{p}_1)} = t|_{I(\bar{p}_2)})$ holds. Thus, $(t|_{p_1.\alpha} = t|_{p_2.\alpha}) \Leftrightarrow (t'|_{p_1.\alpha} = t'|_{p_2.\alpha})$ holds and we are done. □

As a consequence of the previous lemmas, we prove that the result of a global pumping is a run.

Lemma 5.20 *Let $\mathcal{A}$ be a PCTAGC. Let r be a run of $\mathcal{A}$ on a term t. Let $\langle r', t' \rangle$ be the global pumping with indexes $0 \leq j < i \leq h(r)$ and injection I such that $r' = r[r|_{I(\bar{p}_1)}]_{\bar{p}_1} \dots [r|_{I(\bar{p}_n)}]_{\bar{p}_n}$ and $t' = t[t|_{I(\bar{p}_1)}]_{\bar{p}_1} \dots [t|_{I(\bar{p}_n)}]_{\bar{(p)}_n}$. Then, r' is a run of $\mathcal{A}$ on t'.*

proof. By Condition (C_2) of the definition of pump-injection, in order to see that r' is a run, it suffices to see that all global constraints are satisfied. Thus, let us consider two different positions p_1, p_2 of $\mathcal{P}os(r')$ involved in an atom of the constraint of $\mathcal{A}$, *i.e.* either $r'(p_1) \approx r'(p_2)$ or $r'(p_1) \not\approx r'(p_2)$ occurs in the constraint of $\mathcal{A}$. According to Lemma 5.14, we can distinguish the following cases:

• Suppose that a position in $H_i \cup \check{H}_i$, say $\bar{p}_1$, is a prefix of both p_1, p_2. Then, $r'|_{p_1} = r|_{I(\bar{p}_1).(p_1 - \bar{p}_1)}$ and $r'|_{p_2} = r|_{I(\bar{p}_1).(p_2 - \bar{p}_1)}$ hold. Hence, $r'|_{p_1}$ and $r'|_{p_2}$ are also subruns of r occurring at different positions. Thus, since r is a run, they satisfy the atom involving $r'(p_1)$ and $r'(p_2)$.

• Suppose that two different positions in $H_i \cup \check{H}_i$, say $\bar{p}_1$ and $\bar{p}_2$, are prefixes of p_1 and p_2, respectively. Then, $r'|_{p_1} = r|_{I(\bar{p}_1).(p_1 - \bar{p}_1)}$ and $r'|_{p_2} = r|_{I(\bar{p}_2).(p_2 - \bar{p}_2)}$ hold. By the injectivity of I, $I(\bar{p}_1) \neq I(\bar{p}_2)$ holds. Moreover, by Lemma 5.14, $I(\bar{p}_1) \parallel I(\bar{p}_2)$ holds. Hence, as before, $r'|_{p_1}$ and $r'|_{p_2}$ are subruns of r occurring at different (in fact, parallel) positions. Thus, they satisfy the atom involving $r'(p_1)$ and $r'(p_2)$.

• Suppose that one of p_1, p_2, say p_1, is a proper prefix of a position in $H_i \cup \check{H}_i$, and that p_2 satisfies that some position in $H_i \cup \check{H}_i$ is a prefix of p_2. It follows that $h(r'|_{p_2})$ is smaller than or equal to j, and $r'|_{p_2}$ is also a subrun of r. Moreover, p_1 is also a position of r, $r'(p_1) = r(p_1)$ holds, and $h(r|_{p_1}) = i + k$ holds for some $k > 0$. Hence, $t|_{p_1} \neq t'|_{p_2}$ holds. Since r is a run and $r'|_{p_2}$ is a subrun of r, the atom involving $r(p_1)$ and $r'(p_2)$ is necessarily of the form $r(p_1) \not\approx r'(p_2)$. Thus, the atom involving $r'(p_1)$ and $r'(p_2)$ is necessarily of the form $r'(p_1) \not\approx r'(p_2)$. By Lemma 5.17, $h(t'|_{p_1})$ is $j + k$. Therefore, $t'|_{p_1} \neq t'|_{p_2}$ holds, and hence, such an atom is satisfied for such positions in r'.

• Suppose that both p_1, p_2 are proper prefixes of positions in $H_i \cup \check{H}_i$. Then, p_1, p_2 are positions of t satisfying $h(r|_{p_1}), h(r|_{p_2}) \geq i$. Moreover, $r(p_1) = r'(p_1)$ and $r(p_2) = r'(p_2)$ hold. Since r is a run, the atom involving $r(p_1)$ and $r(p_2)$ is satisfied in the run r for positions p_1 and p_2. By Lemma 5.19, $(t|_{p_1} = t|_{p_2}) \Leftrightarrow (t'|_{p_1}) = t'|_{p_2})$ holds. Thus, the atom involving $r'(p_1)$ and $r'(p_2)$ is satisfied in the run r' for positions p_1 and p_2. □

2.2 A Well Quasi-Ordering

In this subsection we define a well quasi-ordering. It assures the existence of a computational bound for certain sequences of elements of the corresponding well quasi-ordered set. It will be connected with global pumpings in the next subsection.

Definition 5.21 *Let $\leq$ denote the usual quasi-ordering on natural numbers. Let n be a natural number. We define*

- *the extension of $\leq$ to n-tuples of natural numbers as $\langle x_1, \ldots, x_n \rangle \leq \langle y_1, \ldots, y_n \rangle$ if $x_i \leq y_i$ for each i in $\{1, \ldots, n\}$.*
- $\mathtt{sum}(\langle x_1, \ldots, x_n \rangle) := x_1 + \cdots + x_n$.
- *the extension of $\leq$ to multisets of n-tuples of natural numbers as $[e_1, \ldots, e_\alpha] \leq [e'_1, \ldots, e'_\beta]$ if there is an injection $I : \{1, \ldots, \alpha\} \to \{1, \ldots, \beta\}$ satisfying $e_i \leq e'_{I(i)}$ for each i in $\{1, \ldots, \alpha\}$.*
- $\mathtt{sum}([e_1, \ldots, e_\alpha]) := \mathtt{sum}(e_1) + \cdots + \mathtt{sum}(e_\alpha)$.
- *the extension of $\leq$ to pairs of multisets of n-tuples of natural numbers as $\langle P_1, \check{P}_1 \rangle \leq \langle P_2, \check{P}_2 \rangle$ if $P_1 \leq P_2$ and $\check{P}_1 \leq \check{P}_2$.*

As a direct consequence of Higman's Lemma [Gal91] we have the following:

Lemma 5.22 *Given n, $\leq$ is a well quasi-ordering for pairs of multisets of n-tuples of natural numbers.*

In any infinite sequence $e_1, e_2, \ldots$ of elements from a well quasi-ordered set there always exist two indexes $i < j$ satisfying $e_i \leq e_j$. In general, this fact does not imply the existence of a bound for the length of sequences without such indexes. For example, the relation $\leq$ between natural numbers is a well quasi-ordering, but there may exist arbitrarily long sequences $x_1, \ldots, x_k$ of natural numbers such that (‡) $x_i > x_j$ for all $1 \leq i < j \leq k$. In order to bound the length of sequences satisfying (‡), it is sufficient to force that the first element and each next element of the sequence are chosen among a finite number of possibilities. Indeed in this this case, by König's lemma, the prefix trees describing all such (finite) sequences is finite. As a particular case of this fact we have the following result.

Lemma 5.23 *There exists a computable function $B : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ such that, given two natural numbers a, n, $B(a, n)$ is a bound for the length ℓ of the maximum-length sequence $\langle T_1, \check{T}_1\rangle, \ldots, \langle T_\ell, \check{T}_\ell\rangle$ of pairs of multisets of n-tuples of natural numbers such that the following conditions hold:*

1. *The tuple $\langle 0, \ldots, 0\rangle$ does not occur in any T_i, $\check{T}_i$ for i in $\{1, \ldots, \ell\}$.*
2. *$\mathtt{sum}(T_1) = 1$ and $\mathtt{sum}(\check{T}_1) = 0$.*
3. *For each i in $\{1, \ldots, \ell - 1\}$, $\mathtt{sum}(T_{i+1}) + \mathtt{sum}(\check{T}_{i+1}) \leq a \cdot \mathtt{sum}(T_i) + \mathtt{sum}(\check{T}_i)$.*
4. *There are no i, j satisfying $1 \leq i < j \leq \ell$ and $\langle T_i, \check{T}_i\rangle \leq \langle T_j, \check{T}_j\rangle$*

proof. We construct a rooted tree $S = (V, E)$ labelled by sequences of pairs of multiset of n-tuples, where the depth of each node is equal to the length of the sequence labeling it and such that the set of internal nodes of S corresponds exactly to the set of sequences satisfying conditions (1) to (4). Then we show that S is finite, which implies that $B(a, n)$ exists and is the maximal depth of S.

Let $V = \{\varepsilon\} \cup W$, where W is the set of all the sequences $\langle T_1, \check{T}_1\rangle, \ldots, \langle T_\ell, \check{T}_\ell\rangle$ of pairs of multisets of n-tuples satisfying the conditions (1) to (3) and such that there are no i, j, satisfying $1 \leq i < j < \ell$ and $\langle T_i, \check{T}_i\rangle \leq \langle T_j, \check{T}_j\rangle$. This last condition, that we will refer to as (5), is weaker than (4). In particular, all sequences satisfying conditions (1) to (4) belong to W. Let $E \subseteq V^2$ be the set of edges containing

- $\varepsilon \longrightarrow \langle T_1, \check{T}_1\rangle$ for all sequence $\langle T_1, \check{T}_1\rangle$ of length 1 in W,
- $\langle T_1, \check{T}_1\rangle, \ldots, \langle T_i, \check{T}_i\rangle \longrightarrow \langle T_1, \check{T}_1\rangle, \ldots, \langle T_i, \check{T}_i\rangle, \langle T_{i+1}, \check{T}_{i+1}\rangle$ for every such couple of sequences in W.

It is quite obvious that $S = (V, E)$ is a tree rooted at ε: ε does not have an input edge, each sequence of length 1 has a unique input edge coming from ε, and each sequence of length $i > 1$ has a unique input edge coming from its unique prefix sequence of length $i - 1$.

We will call *depth* of a node v of S, and note $d(v)$ the number $d(v) = 0$ if $v = \varepsilon$ and $d(v) = \ell$ if v is a sequence of length ℓ. It is obvious that there is always a path of length $d(v) + 1$ from the root to a node $v \in V$.

If a sequence $\langle T_1, \check{T}_1\rangle, \ldots, \langle T_\ell, \check{T}_\ell\rangle$ in W satisfies the condition (4), then any sequence $\langle T_1, \check{T}_1\rangle, \ldots, \langle T_\ell, \check{T}_\ell\rangle, \langle T_{\ell+1}, \check{T}_{\ell+1}\rangle$ satisfying the conditions (1) to (3) also belongs to W, because there is is no i, j such that $1 \leq i < j < \ell + 1$ and $\langle T_i, \check{T}_i\rangle \leq \langle T_j, \check{T}_j\rangle$.

Since there always exists such a sequence (take $\langle T_{\ell+1}, \check{T}_{\ell+1}\rangle = \langle T_1, \check{T}_1\rangle$), then every sequence satisfying condition (1) to (4) is an internal node of S.

If a sequence $\langle T_1, \check{T}_1\rangle, \ldots, \langle T_\ell, \check{T}_\ell\rangle$ in W does not satisfy condition (4), then, by condition (5), it means that $\langle T_1, \check{T}_1\rangle \leq \langle T_\ell, \check{T}_\ell\rangle$.

Hence no sequence $\langle T_1, \check{T}_1\rangle, \ldots, \langle T_\ell, \check{T}_\ell\rangle, \langle T_{\ell+1}, \check{T}_{\ell+1}\rangle$ can satisfy condition (5), and hence do not belong to W. So any sequence satisfying conditions (1) to (3) and (5) but not (4) is a leaf of S.

Moreover, with the previous result, we also know that the set of internal nodes of S is exactly the set of sequences satisfying conditions (1) to (4), and the set of leaves of S is exactly the set of sequences satisfying conditions (1) to (3), (5), but not (4).

Let us show that S is finite. First, each node $v \in V$ has a finite branching: ε links to all the sequences of length 1, the number of which is bounded by condition (2); and each sequence of length i, $\langle T_1, \check{T}_1\rangle, \ldots, \langle T_i, \check{T}_i\rangle$ can only link to sequences of length $i+1$ of the following form $\langle T_1, \check{T}_1\rangle, \ldots, \langle T_i, \check{T}_i\rangle, \langle T_{i+1}, \check{T}_{i+1}\rangle$, the number of which is bounded by condition (3).

We now have to show that there is no infinite branch in S. Assume that we have an infinite branch $v_0, v_1, v_2, v_3, \ldots$ ($\forall i$, $v_i \in V$, and $(v_i, v_{i+1}) \in E$). By construction, we have $v_0 = \varepsilon$, and for all $i \geq 1$ and all $j \geq i$, the prefix of length i of the sequence v_j is equal to v_i. Consider the infinite sequence $\langle T_1, \check{T}_1\rangle, \langle T_2, \check{T}_2\rangle, \ldots$ where for all $i \geq 1$, $\langle T_i, \check{T}_i\rangle$ is the last element of the sequence v_i. Since $\leq$ on pairs of multisets of n-tuples is a well quasi-ordering, there exists two indexes i, j, such that $i < j$ and $\langle T_i, \check{T}_i\rangle \leq \langle T_j, \check{T}_j\rangle$. Hence, all sequences v_k such that $k > j$ do not satisfy condition (5) and do not belong to V, and there is not infinite branch in S.

Now we can show the existence of $B(a, n)$. Since S is finite we can define a number $D(S) = \max_{v \in V} d(v)$. Consider a sequence $\langle T_1, \check{T}_1\rangle, \ldots, \langle T_\ell, \check{T}_\ell\rangle$ of length $\ell \geq D(S)$, satisfying conditions (1) to (3). If this sequence belongs to W, since $D(S)$ is the maximal depth of the nodes in V, it is necessarily a leaf, and hence, does not satisfy (4). And if it does not belong to W, since it satisfies conditions (1) to (3), it does not satisfy condition (5), and hence, does not satisfy condition (4). So there is no sequence of length $\ell \geq D(S)$ satisfying conditions (1) to (4). So $B(a, n) = D(S)$.

$D(S)$ is computable because S is. It can easily be built by a depth-first algorithm. Initialize S with the root ε linking to all the sequences $\langle T_1, \check{T}_1\rangle$ of length 1 (there is a finite and computable number of them). Now, in order to construct nodes of depth $(\ell + 1)$ from nodes of depth ℓ, one can consider each previously built sequence $\langle T_1, \check{T}_1\rangle, \ldots, \langle T_\ell, \check{T}_\ell\rangle$ of length ℓ. If $\langle T_1, \check{T}_1\rangle \leq \langle T_\ell, \check{T}_\ell\rangle$, then we reached a leaf, and do not add new nodes. Otherwise, we add all the sequences $\langle T_1, \check{T}_1\rangle, \ldots, \langle T_\ell, \check{T}_\ell\rangle, \langle T_{\ell+1}, \check{T}_{\ell+1}\rangle$ satisfying the conditions (1) to (3) (there is a finite and computable number of them). The algorithm stops when at some depth ℓ, all nodes of depth ℓ are leaves: then we finished the construction of S, and $B(a, n) = D(S) = \ell$. □

In order to bound the height of a term accepted by a given PCTAGC $\mathcal{A}$ (and of minimum height), Lemma 5.23 will be used by making a to be the maximum arity of the signature of $\mathcal{A}$, and making n to be the number of states of $\mathcal{A}$.

i	r_{H_i}	$r_{\check{H}_i}$
5	$[\langle 0,0,0,0,0,1\rangle]$	$[\,]$
4	$[\langle 0,0,0,0,1,0\rangle]$	$[\langle 0,0,1,0,0,0\rangle, \langle 0,0,0,1,0,0\rangle]$
3	$[\langle 0,0,0,0,1,0\rangle]$	$[\langle 0,0,1,0,0,0\rangle, \langle 0,0,0,2,0,0\rangle, \langle 0,0,1,0,0,0\rangle]$
2	$[\langle 0,0,0,0,1,0\rangle]$	$[\langle 0,0,1,0,0,0\rangle, \langle 0,0,0,3,0,0\rangle, \langle 0,0,1,0,0,0\rangle, \langle 0,0,1,0,0,0\rangle]$
1	$[\langle 0,0,0,4,0,0\rangle]$	$[\langle 0,0,1,0,0,0\rangle, \langle 0,0,1,0,0,0\rangle, \langle 0,0,1,0,0,0\rangle, \langle 0,0,1,0,0,0\rangle]$
0	$[\langle 0,0,1,0,0,0\rangle, \langle 4,0,1,0,0,0\rangle, \langle 0,4,0,0,0,0\rangle, \langle 0,0,1,0,0,0\rangle, \langle 0,0,1,0,0,0\rangle\,]$	$[\,]$

Figure 5.4: Multisets r_{H_i}, $r_{\check{H}_i}$ (Example 5.26).

2.3 Mapping a Run to a Sequence of the Well Quasi-Ordered Set

We will associate, to each number i in $\{0,\ldots,h(r)\}$, a pair of multisets of tuples of natural numbers, which can be compared with other pairs according to the definition of $\leq$ in the previous subsection. To this end, we first associate tuples to terms and multisets of tuples to sets of positions.

Definition 5.24 *Let $\mathcal{A}$ be a PCTAGC. Let $q_1,\ldots,q_n$ be the states of $\mathcal{A}$. Let r be a run of $\mathcal{A}$ on a term t. Let P be a set of positions of r. Let t' be a term. We define $r_{t',P}$ as the following tuple of natural numbers: $\langle |\{p \in P \mid t|_p = t' \wedge r(p) = q_1\}|, \ldots, |\{p \in P \mid t|_p = t' \wedge r(p) = q_n\}| \rangle$*

Definition 5.25 *Let $\mathcal{A}$ be a PCTAGC. Let r be a run of $\mathcal{A}$ on a term t. Let P be a set of positions of r. Let $\{t_1,\ldots,t_k\}$ be the set of terms $\{t' \mid \exists p \in P : t|_p = t'\}$. We define r_P as the multiset $[r_{t_1,P},\ldots,r_{t_k,P}]$.*

Example 5.26 *Following our running example, for the representation of the tuples of natural numbers we order the states as $\langle q_d, q_N, q_{id}, q_t, q_L, q_M\rangle$. The multisets r_{H_i} and $r_{\check{H}_i}$ are presented in Figure 5.4.* ◇

The following lemma connects the existence of a pump-injection with the quasi-ordering relation.

Lemma 5.27 *Let $\mathcal{A}$ be a PCTAGC. Let r be a run of $\mathcal{A}$ on a term t. Let i,j be integers satisfying $0 \leq j < i \leq h(r)$. Then, there exists a pump-injection $I : (H_i \cup \check{H}_i) \to (H_j \cup \check{H}_j)$ if and only if $\langle r_{H_i}, r_{\check{H}_i}\rangle \leq \langle r_{H_j}, r_{\check{H}_j}\rangle$.*

proof. $\Rightarrow$. Assume that there exists a pump-injection $I : (H_i \cup \check{H}_i) \to (H_j \cup \check{H}_j)$. We just prove $r_{H_i} \leq r_{H_j}$, since $r_{\check{H}_i} \leq r_{\check{H}_j}$ can be proved

analogously. By Condition (C_1) of the definition of pump-injection, $I(H_i) \subseteq H_j$. We write $\{t|_p \mid p \in H_i\}$ and $\{t|_p \mid p \in H_j\}$ more explicitly as $\{t_{i,1}, \ldots, t_{i,\alpha}\}$ and $\{t_{j,1}, \ldots, t_{j,\beta}\}$, respectively. Hence, it remains to prove that $[r_{t_{i,1},H_i}, \ldots, r_{t_{i,\alpha},H_i}] \leq [r_{t_{j,1},H_j}, \ldots, r_{t_{j,\beta},H_j}]$. To this end we define the function $I' : \{1, \ldots, \alpha\} \to \{1, \ldots, \beta\}$ as follows. For each γ in $\{1, \ldots, \alpha\}$, we choose a position p in H_i satisfying $t|_p = t_{i,\gamma}$, determine the index δ of the term $t_{j,\delta}$ satisfying $t_{j,\delta} = t|_{I(p)}$, and define $I'(\gamma) := \delta$. This function I' is injective due to Condition (C_3) of the definition of pump-injection. In order to conclude, it suffices to prove $r_{t_{i,\gamma},H_i} \leq r_{t_{j,I'(\gamma)},H_j}$ for each γ in $\{1, \ldots, \alpha\}$. We just prove it for $\gamma = 1$. For proving $r_{t_{i,1},H_i} \leq r_{t_{j,I'(1)},H_j}$ it suffices to prove the following statement for each state q of $\mathcal{A}$: $\big|\{p \in H_i \mid t|_p = t_{i,1} \wedge r(p) = q\}\big| \leq \big|\{p \in H_j \mid t|_p = t_{j,I'(1)} \wedge r(p) = q\}\big|$.

To this end, since I is injective, it suffices to prove that $I(\{p \in H_i \mid t|_p = t_{i,1} \wedge r(p) = q\})$ is included in $\{p \in H_j \mid t|_p = t_{j,I'(1)} \wedge r(p) = q\}$ for each state q of $\mathcal{A}$. Thus, consider any $\bar{p}$ of $\{p \in H_i \mid t|_p = t_{i,1} \wedge r(p) = q\}$. Let p' be the chosen position for defining $I'(1)$. In particular, $t|_{p'} = t_{i,1}$ and $t|_{I(p')} = t_{j,I'(1)}$ hold. Note that $t|_{\bar{p}} = t|_{p'} = t_{i,1}$ holds. Thus, by Condition (C_3) of the definition of pump-injection, $t|_{I(\bar{p})} = t|_{I(p')}$ holds. Therefore, $t|_{I(\bar{p})} = t_{j,I'(1)}$ holds. In order to show the inclusion $I(\bar{p}) \in \{p \mid t|_p = t_{j,I'(1)} \wedge r(p) = q\}$ it rests to see $r(I(\bar{p})) = q$. Note that, since $\bar{p}$ belongs to $\{p \mid t|_p = t_{i,1} \wedge r(p) = q\}$, $r(\bar{p}) = q$ holds. By Condition (C_2) of the definition of pump-injection, $r(I(\bar{p})) = r(\bar{p}) = q$ holds, and we are done.

$\Leftarrow$. Assume that $\langle r_{H_i}, r_{\check{H}_i}\rangle \leq \langle r_{H_j}, r_{\check{H}_j}\rangle$ holds. We have to construct a pump-injection $I : (H_i \cup \check{H}_i) \to (H_j \cup \check{H}_j)$. We just define $I : H_i \to H_j$ and prove Conditions (C_2) and (C_3) for $\bar{p}, \bar{p}_1, \bar{p}_2$ in H_i. This is because $I : \check{H}_i \to \check{H}_j$ can be defined analogously, and Conditions (C_2) and (C_3) for the corresponding positions can be checked analogously. Moreover, for positions $\bar{p}'_1 \in H_i$ and $\bar{p}'_2 \in \check{H}_i$, Condition (C_3) holds whenever Condition (C_1) holds since in this case $t|_{\bar{p}'_1} \neq t|_{\bar{p}'_2}$ and $t|_{I(\bar{p}'_1)} \neq t|_{I(\bar{p}'_2)}$ hold. Hence, this simple case is enough to prove the whole statement.

We write $\{t|_p \mid p \in H_i\}$ and $\{t|_p \mid p \in H_j\}$ more explicitly as $\{t_{i,1}, \ldots, t_{i,\alpha}\}$ and $\{t_{j,1}, \ldots, t_{j,\beta}\}$, respectively. Since $\langle r_{H_i}, r_{\check{H}_i}\rangle \leq \langle r_{H_j}, r_{\check{H}_j}\rangle$ holds, $r_{H_i} \leq r_{H_j}$ also holds. Thus, there exists an injective function $I' : \{1, \ldots, \alpha\} \to \{1, \ldots, \beta\}$ satisfying the following statement for each δ in $\{1, \ldots, \alpha\}$ and each state q of $\mathcal{A}$:
$\big|\{p \in H_i \mid t|_p = t_{i,\delta} \wedge r(p) = q\}\big| \leq \big|\{p \in H_j \mid t|_p = t_{j,I'(\delta)} \wedge r(p) = q\}\big|$ (†).

In order to define $I : H_i \to H_j$, we define I for each of such sets $\{p \in H_i \mid t|_p = t_{i,\delta} \wedge r(p) = q\}$ as any injective function $I : \{p \in H_i \mid t|_p = t_{i,\delta} \wedge r(p) = q\} \to \{p \in H_j \mid t|_p = t_{j,I'(\delta)} \wedge r(p) = q\}$, which is possible by the above inequality (†). The global I is then injective thanks to the injectivity of I'. Conditions (C_2) and (C_3) trivially follow from this definition. □

Example 5.28 *Following our running example, we first prove* $\langle r_{H_4}, r_{\check{H}_4}\rangle \leq \langle r_{H_3}, r_{\check{H}_3}\rangle$. *To this end just note that* $[\langle 0,0,0,0,1,0\rangle] \leq [\langle 0,0,0,0,1,0\rangle]$, $[\langle 0,0,1,0,0,0\rangle] \leq [\langle 0,0,1,0,0,0\rangle]$, *and* $[\langle 0,0,0,1,0,0\rangle] \leq [\langle 0,0,0,2,0,0\rangle]$ *hold. We can define* $I : (H_4 \cup \check{H}_4) \to (H_3 \cup \check{H}_3)$ *from this relation according to Lemma 5.27. Doing the adequate guess we obtain the following definition:* $I(1) = 3.1$, $I(2) = 2$, $I(3) = 3.3$ *which is the pump-injection considered above for our running example.* ◇

The following lemma follows directly from the definition of the sets H_i and $\check{H}_i$, and allows to connect such definitions with Lemma 5.23.

Lemma 5.29 *Let* $\mathcal{A}$ *be a PCTAGC. Let* a *be the maximum arity of the symbols in the signature of* $\mathcal{A}$. *Let* r *be a run of* $\mathcal{A}$ *on a term* t. *Then, the following conditions hold:*

(1) $|H_{h(t)}| = 1$ *and* $|\check{H}_{h(t)}| = 0$.
(2) For each i *in* $\{1, \ldots, h(r)\}$, $|H_{i-1}| + |\check{H}_{i-1}| \leq a \cdot |H_i| + |\check{H}_i|$.
(3) For each i *in* $\{0, \ldots, h(r)\}$, $|H_i| = \mathtt{sum}(r_{H_i})$ *and* $|\check{H}_i| = \mathtt{sum}(r_{\check{H}_i})$.

proof. Item (1) is trivial by definition of H_i and $\check{H}_i$ for $i = h(r)$. For Item (2), it suffices to observe that the positions in $H_{i-1} \cup \check{H}_{i-1}$ are all the positions of $\check{H}_i$ plus all the child positions in H_i, and that each position has at most a children. For Item (3) we just prove $|H_i| = \mathtt{sum}(r_{H_i})$, since $|\check{H}_i| = \mathtt{sum}(r_{\check{H}_i})$ can be proved analogously. We write $\{t|_p \mid p \in H_i\}$ more explicitly as $\{t_1, \ldots, t_\alpha\}$.

Note that H_i is the disjoint union $\{p \in H_i \mid t|_p = t_1\} \cup \ldots \cup \{p \in H_i \mid t|_p = t_\alpha\}$. Thus, $|H_i|$ equals $|\{p \in H_i \mid t|_p = t_1\}| + \ldots + |\{p \in H_i \mid t|_p = t_\alpha\}|$. We conclude by observing that $|\{p \in H_i \mid t|_p = t_1\}| = \mathtt{sum}(r_{t_1,H_i})$, $\ldots$, $|\{p \in H_i \mid t|_p = t_\alpha\}| = \mathtt{sum}(r_{t_\alpha,H_i})$ hold. □

Lemma 5.30 *Let* $B : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ *be the computable function of Lemma 5.23. Let* $\mathcal{A}$ *be a PCTAGC. Let* a *be the maximum arity of the symbols in the signature of* $\mathcal{A}$. *Let* n *be the number of states of* $\mathcal{A}$. *Let* r *be a run of* $\mathcal{A}$ *on a term* t *satisfying* $h(t) \geq B(a, n)$. *Then, there is a global pumping on* r.

proof. Consider the sequence $\langle r_{H_{h(r)}}, r_{\check{H}_{h(r)}}\rangle, \ldots, \langle r_{H_0}, r_{\check{H}_0}\rangle$. Note that the n-tuple $\langle 0, \ldots, 0\rangle$ does not appear in the multisets of the pairs of this sequence. By Lemma 5.29, $|H_{h(t)}| = 1$ and $|\check{H}_{h(t)}| = 0$ hold, and for each i in $\{1, \ldots, h(t)\}$, $|H_{i-1}| + |\check{H}_{i-1}| \leq a \cdot |H_i| + |\check{H}_i|$ holds. Moreover, for each i in $\{0, \ldots, h(r)\}$, $|H_i| = \mathtt{sum}(r_{H_i})$ and $|\check{H}_i| = \mathtt{sum}(r_{\check{H}_i})$. Thus, $\mathtt{sum}(r_{H_{h(r)}}) = 1$, $\mathtt{sum}(r_{\check{H}_{h(r)}}) = 0$, and for each i in $\{1, \ldots, h(r)\}$, $\mathtt{sum}(r_{H_{i-1}}) + \mathtt{sum}(r_{\check{H}_{i-1}}) \leq a \cdot \mathtt{sum}(r_{H_i}) + \mathtt{sum}(r_{\check{H}_i})$. Hence, since $h(t) \geq B(a, n)$ holds, by Lemma 5.23 there exist i, j satisfying $h(t) \geq i > j \geq 0$ and $\langle r_{H_i}, r_{\check{H}_i}\rangle \leq \langle r_{H_j}, r_{\check{H}_j}\rangle$. By Lemma 5.27, There exists a pump-injection $I : (H_i \cup \check{H}_i) \to (H_j \cup \check{H}_j)$. Therefore, there exists a global pumping on r. □

Theorem 5.31 *Emptiness is decidable for PCTAGC.*

proof. Let a be the maximum arity of the symbols in the signature of $\mathcal{A}$. Let n be the number of states of $\mathcal{A}$. Let r be an accepting run of $\mathcal{A}$ on a term t with minimum height.

Suppose that $h(r) \geq B(a,n)$ holds. Then, by Lemma 5.30, there exists a global pumping $\langle r', t' \rangle$ on r and t. By Corollary 5.18, $h(t') < h(t)$ holds. Moreover, by the definition of global pumping, $r'(\varepsilon) = r(\varepsilon)$ holds. Finally, by Lemma 5.20, r' is a run of $\mathcal{A}$ on t'. Thus, t' contradicts the minimality of t. We conclude that $h(t) < B(a,n)$ holds.
The decidability of emptiness of $\mathcal{A}$ follows, since the existence of successful runs implies that one of them can be found among a computable and finite set of possibilities. □

Using Lemma 5.11 and Theorem 5.31, we can conclude the decidability of emptiness for TAGC.

Corollary 5.32 *Emptiness is decidable for TAGC.*

Since, we used TAGC with natural arithmetic constraints to obtain this reduction, another consequence is the decidability of this class.

Corollary 5.33 *Emptiness is decidable for TAGC*$[\approx, \not\approx, \mathbb{N}]$.

3 Equality Tests Between Brothers

The constraints of TAGC are checked once for all on a whole run. There exists another kind of equality and disequality constraints for extending TA which are tested locally at every transition step, as in the TAC that we mentioned in section 2.2.

Definition 5.34 *A* tree automaton with constraints between brothers *(TACB) is a tuple* $\mathcal{A} = \langle \Sigma, Q, \mathcal{F}, F, \Delta \rangle$ *where* Σ*,* Q*,* F*,* $\mathcal{F}$ *are defined as with TA and the transitions rules of* Δ *have the form:* $f(q_1, \ldots, q_n) \xrightarrow{c} q$*, where* c *is a conjunction of atoms of the form* $i = j$ *or* $i \neq j$ *with* $1 \leq i, j \leq n$*.*

A run *of the TACB* $\mathcal{A}$ *on a term* $t \in \mathcal{T}(\mathcal{F})$ *is a function* r *from* $\mathcal{P}os(t)$ *into* Q *such that, for all* $p \in \mathcal{P}os(t)$*, there exists a rule* $t(p)\big(r(p.1), \ldots, r(p.m)\big) \xrightarrow{c} r(p) \in \Delta$ *satisfying* $t(p) \in \mathcal{F}_m$*, and moreover, for all* $i = j$ *in the conjunction* c*,* $t|_{p.i} = t|_{p.j}$ *holds, and for all* $i \neq j$ *in the conjunction* c*,* $t|_{p.i} \neq t|_{p.j}$ *holds.*

The notions of successful runs and languages can be extended straightforwardly from TA to TACB. Global constraints can also be added to TACB in the natural way. The automata of the resulting classes TACB$[\approx, \not\approx, ...]$ will therefore perform global and local test during their computations.

Example 5.35 *Assume that the terms of Example 2.6 are now used to record the activity of a restaurant, and that we add a second argument of type q_t to $\mathcal{L}_0$, $\mathcal{L}$ and M, so that the first argument q_t will characterize the theoretical time to cook, and the second q_t will characterize the real time that was needed to cook the dish. Let us replace the transitions with $\mathcal{L}_0$, $\mathcal{L}$ and M in input by $\mathcal{L}_0(q_{id}, q_t, q_t) \xrightarrow{2=3} q_{\mathcal{L}}$, $\mathcal{L}_0(q_{id}, q_t, q_t) \xrightarrow{2\neq 3} q'_{\mathcal{L}}$, $\mathcal{L}(q_{id}, q_t, q_t, q_{\mathcal{L}}) \xrightarrow{2=3} q_{\mathcal{L}}$, $\mathcal{L}(q_{id}, q_t, q_t, q_{\mathcal{L}}) \xrightarrow{2\neq 3} q'_{\mathcal{L}}$, $M(q_{id}, q_t, q_t, q_{\mathcal{L}}) \xrightarrow{2=3} q_M$, $M(q_{id}, q_t, q_t, q_{\mathcal{L}}) \xrightarrow{2\neq 3} q'_M$, where $q'_{\mathcal{L}}$ is a new state meaning that there was an anomaly. We also also add a transition $\mathcal{L}(q_{id}, q_t, q_t, q'_{\mathcal{L}}) \to q'_{\mathcal{L}}$ to propagate $q'_{\mathcal{L}}$ and $M(q_{id}, q_t, q_t, q'_{\mathcal{L}}) \to q'_M$.*

The language of the TAGC obtained is the set of records well cooked, i.e. such that for all dishes, the real time to cook is equal to the theoretical time. ◇

The emptiness decision algorithm of section 2 still works for this extension of TAGC with local brother constraints. This is because a global pumping $r[r|_{I(\bar{p}_1)}]_{\bar{p}_1} \dots [r|_{I(\bar{p}_n)}]_{\bar{p}_n}$ on a run r can be proved to satisfy the constraints between brothers in a completely analogous way as in the proof of Lemma 5.20 and Theorem 5.33.

Theorem 5.36 *Emptiness is decidable for TACB$[\approx, \not\approx, \mathbb{N}]$.*

4 Unranked Ordered Trees

Our tree automata models and results can be generalized from ranked to unranked ordered terms. In this setting, $\mathcal{F}$ is called an *unranked signature*, meaning that there is no arity fixed for its symbols, *i.e.* that in a term $a(t_1, \dots, t_n)$, the number n of children is arbitrary and does not depend on a. Let us denote by $\mathcal{U}(\mathcal{F})$ the set of unranked ordered terms over $\mathcal{F}$. The notions of positions, subterms *etc* are defined for unranked terms of $\mathcal{U}(\mathcal{F})$ as for ranked terms of $\mathcal{T}(\mathcal{F})$. We extend the definition of automata for unranked ordered terms, called hedge automata [Mur99], with global constraints.

Definition 5.37 *A* hedge automaton with global constraints *(**HAGC**) is a tuple $\mathcal{A} = \langle Q, \mathcal{F}, F, C, \Delta \rangle$ where Q, F and C are as in the definition of TAGC (def. 2.1) and the transitions of Δ have the form $a(\mathcal{L}) \to q$ where $a \in \mathcal{F}$, $q \in Q$ and $\mathcal{L}$ is a regular (word) language over Q^*, assumed given by a NFA with input alphabet Q.*

A run *of $\mathcal{A}$ on an unranked ordered term $t \in \mathcal{U}(\mathcal{F})$ is a function r from $\mathcal{P}os(t)$ into Q such that for all position $p \in \mathcal{P}os(t)$ with n child, there exists a transition $t(p)(\mathcal{L}) \to r(p)$ in Δ such that $r(p.1) \dots r(p.n) \in L$. The run r is called* successful *if moreover $r(\varepsilon) \in F$ and $r \models C$, where satisfiability is defined like in definition 2.1. As above, we use the notation HAGC$[\tau_1, \dots, \tau_n]$ where the τ_i's can be $\approx$, $\not\approx$, $|.|_{\mathbb{N}}$, $\|.\|_{\mathbb{N}}$...*

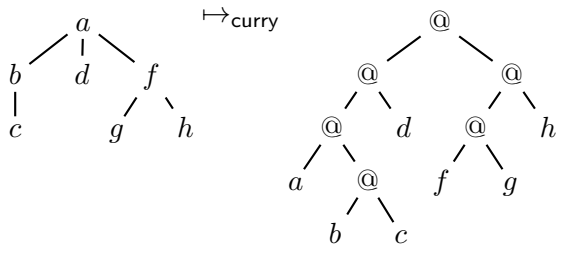

Figure 5.5: Currying an unranked term

The emptiness decision results of Corollary 5.32 can be extended from TAGC to HAGC using a standard transformation from unranked to ranked binary terms, like the *extension* encoding described in [CLDG+07], Chapter 8.

Let us associate to the unranked signature Σ the (ranked) signature $\mathcal{F}_@ := \{a:0 \mid a \in \mathcal{F}\} \cup \{@:2\}$ where @ is a new symbol. The operator curry, recursively defined in the following, is a bijection from $\mathcal{U}(\mathcal{F})$ into $\mathcal{T}(\mathcal{F}_@)$.

$$\begin{array}{rcl} \mathsf{curry}(a) & = & a \quad \text{for all } a \in \mathcal{F} \\ \mathsf{curry}\big(a(t_1,\ldots,t_n)\big) & = & @\big(\mathsf{curry}\big(a(t_1,\ldots,t_{n-1})\big), \mathsf{curry}(t_n)\big) \end{array}$$

An example of application of this operator is presented in Figure 5.5. We extend the application of the operator curry to set of trees by

$$\mathsf{curry}(L) = \{\mathsf{curry}(t) \mid t \in L\}.$$

Proposition 5.38 *For all HAGC*$[\approx, \not\approx, \mathbb{N}]$ $\mathcal{A}$ *over* $\mathcal{F}$*, one can construct effectively in PTIME a TAGC*$[\approx, \not\approx, \mathbb{N}]$ $\mathcal{A}'$ *over* $\mathcal{F}_@$ *such that* $\mathcal{L}(\mathcal{A}') = \mathsf{curry}\big(\mathcal{L}(\mathcal{A})\big)$.

proof. Let $\mathcal{A} = \langle Q, \mathcal{F}, F, C, \Delta\rangle$. We call $ha(\mathcal{A})$ the underlying hedge automaton of $\mathcal{A}$: $ha(\mathcal{A}) = \langle Q, \mathcal{F}, F, \mathit{true}, \Delta\rangle$ *i.e.* $ha(\mathcal{A})$ computes the same way as $\mathcal{A}$ but without the global constraints (note however that the top state of successful runs must be a final state).

We can assume *wlog* that Δ contains at most one transition $a(\mathcal{L}) \to q$ for each pair $\langle a, q\rangle \in \Sigma \times Q$. Otherwise, we can replace two transitions $a(\mathcal{L}) \to q$ and $a(\mathcal{L}') \to q$ by the unique transition $a(\mathcal{L} \cup \mathcal{L}') \to q$, preserving the language recognized. Let $B_{a,q}$ be the NFA recognizing $\mathcal{L}$ in the (unique) transition $a(\mathcal{L}) \to q$ of Δ associated to a and q. Note that such an automaton has Q as input alphabet. Let P be the union (assumed disjoint) for all transitions in Δ of the state sets of the automata $B_{a,q}$.

The transitions of the automaton $\mathcal{A}'$, when computing on $\mathsf{curry}(t)$ for some $t \in \mathcal{U}(\mathcal{F})$, will simulate both the transitions of $\mathcal{A}$ (vertical transitions) and the transitions of the NFAs $B_{a,q}$ (horizontal transitions).

Let $\mathcal{A}' = \langle Q \cup P, \mathcal{F}, F, C, \Delta'\rangle$ where Δ' contains the transitions:

$a \to q$ if $B_{a,q}$ recognizes the empty word,

$a \to init_{a,q}$ for all $q \in Q$, where $init_{a,q}$ is the initial state of $B_{a,q}$,

$@(p,q) \to p'$ if there is a transition $p \xrightarrow{q} p'$ in some $B_{a',q'}$, and

$@(p,q) \to q'$ if there is a transition $p \xrightarrow{q} p'$ in some $B_{a',q'}$, where p' is a final state of $B_{a',q'}$.

We can show by induction on $t \in \mathcal{U}(\mathcal{F})$ that $t \in \mathcal{L}\big(ha(\mathcal{A})\big)$ iff $\mathsf{curry}(t) \in \mathcal{L}\big(ta(\mathcal{A}')\big)$. More precisely, for all $t \in \mathcal{U}(\mathcal{F})$, to every run r of $ha(\mathcal{A})$ on t, we can associate a run r' of $ta(\mathcal{A}')$ on $\mathsf{curry}(t)$ with $r'(\varepsilon) = r(\varepsilon)$, and reciprocally. Moreover, by construction, there exists an injective mapping θ from $\mathcal{P}os(t)$ into $\mathcal{P}os(\mathsf{curry}(t))$, such that for all $p \in \mathcal{P}os(t)$, the states $r(p)$ and $r'(\theta(p))$ coincide and $\mathsf{curry}(t|_p) = \mathsf{curry}(t)|_{\theta(p)}$. Since curry is bijective, it follows that the run r is successful for $\mathcal{A}$ iff the run r' is successful for $\mathcal{A}'$. □

The following emptiness decision result is a direct consequence of Proposition 5.38 and Theorem 5.33.

Theorem 5.39 *Emptiness is decidable for HAGC*$[\approx, \not\approx, \mathbb{N}]$.

5 Monadic Second Order Logic

In this section, we discuss the application of our results to second order logics interpreted over domains defined by terms. We propose a strict extension of the second order monadic logic of the tree with predicates corresponding to the above equality, disequality and arithmetic constraints, and show that satisfiability is decidable for this extension thanks to a correspondence with TAGC$[\approx, \not\approx, \mathbb{N}]$.

5.1 MSO on Ranked Terms

A ranked term $t \in \mathcal{T}(\Sigma)$ over Σ can be seen as a model for logical formulae, with an interpretation domain which is the set of positions $\mathcal{P}os(t)$. We consider monadic second order formulae interpreted on such models, built with the usual Boolean connectors, with quantifications over first order variables (interpreted as positions), denoted $x, y \ldots$ and over unary predicates (*i.e.* second order variables interpreted as sets of positions), denoted $X, Y \ldots$, and with the following predicates,

1. equality: $x = y$, and membership: $X(x)$
2. labeling: $a(x)$, for $a \in \mathcal{F}$
3. navigation: $S_i(x, y)$, for all i smaller than or equal to the maximal arity of symbols of $\mathcal{F}$ (we call $+1$ the type of such predicates),
4. term equality: $X \approx Y$, term disequality: $X \not\approx Y$ (predicate types $\approx$ and $\not\approx$),
5. linear inequalities: $\sum a_i.|X_i| \geq a$ or $\sum a_i.\|X_i\| \geq a$, where every a_i and a belong to $\mathbb{Z}$ (predicate types $|.|_{\mathbb{Z}}$ and $\|.\|_{\mathbb{Z}}$).

We write MSO$[\tau_1, \ldots, \tau_k]$ for the set of monadic second order logic formulae with equality, membership, labeling predicates and other predicates of type $\tau_1, \ldots, \tau_k$, amongst the above types $+1$, $\approx$, $\not\approx$, and $|.|_{\mathbb{Z}}$, $\|.\|_{\mathbb{Z}}$. We also use the notations $|.|_{\mathbb{N}}$ and $\|.\|_{\mathbb{N}}$ for natural linear inequalities and the abbreviations $\mathbb{Z}$ and $\mathbb{N}$ as in section 1. Let EMSO$[\tau_1, \ldots, \tau_k]$ be the fragment of MSO$[\tau_1, \ldots, \tau_k]$ containing the formulae of the form $\exists X_1 \ldots \exists X_n\, \phi$ such that all the atoms of type $\approx$, $\not\approx$, $\mathbb{Z}$ or $\mathbb{N}$ involve only second order variables amongst $X_1, \ldots, X_n$.

A variable assignment into a term $t \in \mathcal{T}(\Sigma)$ is a function σ mapping first order variables into positions of $\mathcal{P}os(t)$ and second order variables into subsets of $\mathcal{P}os(t)$. The validity of a formula ϕ in a term $t \in \mathcal{T}(\Sigma)$ under the variable assignment σ, denoted $t, \sigma \models \phi$ is defined in the usual Tarksian manner, with (below, $|S|$ denotes the cardinality of the set S):

$$
\begin{array}{lcll}
t,\sigma & \models & x = y & \text{iff } \sigma(x) = \sigma(y) \\
t,\sigma & \models & X(x) & \text{iff } \sigma(x) \in \sigma(X) \\
t,\sigma & \models & a(x) & \text{iff } t(\sigma(x)) = a \\
t,\sigma & \models & S_i(x,y) & \text{iff } \sigma(y) = \sigma(x).i \\
t,\sigma & \models & X \approx Y & \text{iff for all } p \text{ in } \sigma(X) \text{ and } p' \text{ in } \sigma(Y),\ t|_p = t|_{p'} \\
t,\sigma & \models & X \not\approx Y & \text{iff for all } p \text{ in } \sigma(X) \text{ and } p' \text{ in } \sigma(Y),\ t|_p \neq t|_{p'} \\
t,\sigma & \models & \sum a_i.|X_i| \geq a & \text{iff } \sum_i a_i.|\sigma(X_i)| \geq a \\
t,\sigma & \models & \sum a_i.\|X_i\| \geq a & \text{iff } \sum_i a_i.\left|\{t|_p \mid p \in \sigma(X_i)\}\right| \geq a
\end{array}
$$

Example 5.40 *The following formula of EMSO$[\approx, \not\approx]$, $\exists X_a\, (\forall x\, X_a(x) \leftrightarrow a(x)) \wedge X_a \not\approx X_a$ expresses that all the subterms headed by a in a term t are pairwise different. It other words, a is used to mark monadic keys in t (see Example 2.6). This can also be expressed by $\|X_a\| \leq 1$.*

It is well known that MSO$[+1]$ has exactly the same expressiveness as TA [TW68] and therefore it is decidable. The extension MSO$[+1, \approx]$ is undecidable, see *e.g.* [FTT07], as well as MSO$[+1, |.|_{\mathbb{Z}}]$ [KR02] (the latter extension is also undecidable for unranked ordered terms when counting constraints are applied to sibling positions [SSM03]), but the fragment EMSO$[+1, |.|_{\mathbb{Z}}]$ is decidable [KR02].

In [FTT08] the fragment EMSO with $\approx$ and a restricted form of $\not\approx$ is shown decidable, with a two way correspondence between these formulae and a decidable subclass of TAGEDs. This construction can be straightforwardly adapted to establish a two way correspondence between EMSO$[+1, \approx, \not\approx, \mathbb{N}]$ and TAGC$[\approx, \not\approx, \mathbb{N}]$.

Theorem 5.41 *EMSO$[+1, \approx, \not\approx, \mathbb{N}]$ is decidable on ranked terms.*

proof. Following the same proof scheme as [FTT08], we show that for every formula ϕ in EMSO$[+1, \approx, \not\approx, \mathbb{N}]$, we can construct a TAGC$[\approx, \not\approx, \mathbb{N}]$ recognizing exactly the set of models of ϕ. Then, the decidability of the logic follows from Theorem 5.33.

We may assume *wlog* that ϕ has the form

$$\phi = \exists X_1 \ldots \exists X_n \ (\phi_0(\overline{X}) \wedge \phi_{\approx}(\overline{X}) \wedge \phi_{\mathbb{N}}(\overline{X}))$$

where $\phi_0(\overline{X})$ is a MSO[+1] formula with free variables $\overline{X} = X_1, \ldots, X_n$, and $\phi_{\approx}(\overline{X})$ and $\phi_{\mathbb{N}}(\overline{X})$ are Boolean combinations of atoms of the respective form $X_i \approx X_j$, $X_i \not\approx X_j$ and $\sum a_j.|X_{i_j}| \geq a$, $\sum a_j.\|X_{i_j}\| \geq a$. Moreover, we shall also assume that $\phi_{\approx}(\overline{X})$ and $\phi_{\mathbb{N}}(\overline{X})$ are conjunctions of the atoms or negations of atoms of the above form. Otherwise, we put them into disjunctive normal form and then split ϕ into an equivalent formula $\phi_1 \vee \ldots \vee \phi_k$, where each ϕ_i, $i \leq k$, is of the form requested ($\phi_i = \exists X_1 \ldots \exists X_n \ (\phi^i_0(\overline{X}) \wedge \phi^i_{\approx}(\overline{X}) \wedge \phi^i_{\mathbb{N}}(\overline{X}))$, with $\phi^i_0(\overline{X}) \in$ MSO[+1] and $\phi^i_{\approx}(\overline{X})$ and $\phi^i_{\mathbb{N}}(\overline{X})$ are conjunctions of atoms or negations of atoms as above) and we solve satisfiability separately for each ϕ_i.

First, using the construction of [TW68], we associate to $\phi_0(\overline{X})$ a TA $\mathcal{A}_0 = \langle Q, \mathcal{F} \times \{0,1\}^n, F, \Delta_0 \rangle$ which recognizes the set of set of terms $t \otimes \sigma \in \mathcal{T}\big(\mathcal{F} \times \{0,1\}^n\big) \mid t, \sigma \models \phi_0(\overline{X})$. Here, the arity of a symbol $\langle f, b_1, \ldots, b_n \rangle$ in $\Sigma \times \{0,1\}^n$ is the arity of f in $\mathcal{F}$ and $t \otimes \sigma$ denotes the term obtained from $t \in \mathcal{T}(\mathcal{F})$ by relabeling every position $p \in \mathcal{P}os(t)$ by $\langle t(p), b_1, \ldots, b_n \rangle$ where for each $i \leq n$, $b_i = 1$ if $p \in \sigma(X_i)$ and $b_i = 0$ otherwise. Note that the size of the above automaton $\mathcal{A}_0$ is in general non-elementary in the size of ϕ.

Then, following a construction in [NPTT05], we shift in $\mathcal{A}_0$ the bit-vectors from the signature into the state symbols, obtaining $\mathcal{A}'_0 = \langle Q \times \{0,1\}^n, \Sigma, F \times \{0,1\}^n, \Delta \rangle$ where Δ contains all the transition rules

$$f\big(\langle q_1, b_{1,1}, \ldots, b_{1,n} \rangle, \ldots, \langle q_m, b_{m,1}, \ldots, b_{m,n} \rangle\big) \to \langle q, b_1, \ldots, b_n \rangle$$

such that $f \in \mathcal{F}$, $\langle f, b_1, \ldots, b_n \rangle\big(q_1, \ldots, q_m\big) \to q \in \Delta_0$ and $b_{1,1}, \ldots, b_{1,n}$, $\ldots$, $b_{m,1}, \ldots, b_{m,n} \in \{0,1\}$. This automaton $\mathcal{A}'_0$ recognizes the projection (on the first components) of the terms recognized by $\mathcal{A}_0$, *i.e.* it recognizes the set of terms $t \in \mathcal{T}(\mathcal{F})$ such that there exists $\sigma : \{X_1, \ldots, X_n\} \to 2^{\mathcal{P}os(t)}$ with $t, \sigma \models \phi_0(\overline{X})$.

Now, let us construct a constraint C by rewriting the atoms of $\phi_{\approx}(\overline{X}) \wedge \phi_{\mathbb{N}}(\overline{X})$, following the rules:

$$\begin{array}{rcl}
X_i \approx X_j & \mapsto & \displaystyle\bigwedge_{b_i = b'_j = 1} \langle q, b_1, \ldots, b_n \rangle \approx \langle q', b'_1, \ldots, b'_n \rangle \\
X_i \not\approx X_j & \mapsto & \displaystyle\bigwedge_{b_i = b'_j = 1} \langle q, b_1, \ldots, b_n \rangle \not\approx \langle q', b'_1, \ldots, b'_n \rangle \\
\displaystyle\sum_j a_j.|X_{i_j}| \geq a & \mapsto & \displaystyle\sum_j \sum_{b_{i_j} = 1} a_j.|\langle q, b_1, \ldots, b_n \rangle| \geq a \\
\displaystyle\sum_j a_j.\|X_{i_j}\| \geq a & \mapsto & \displaystyle\sum_j \sum_{b_{i_j} = 1} a_j.\|\langle q, b_1, \ldots, b_n \rangle\| \geq a
\end{array}$$

Finally, we can show straightforwardly that the TAGC[$\approx, \not\approx, \mathbb{N}$] $\mathcal{A} = \langle Q \times \{0,1\}^n, \Sigma, F \times \{0,1\}^n, \Delta, C \rangle$ recognizes exactly $\{t \in \mathcal{L}(\mathcal{A}) \mid t \models \phi\}$. □ □

The above transformation also works in the other direction (this result is not necessary for the proof of Theorem 5.41 though): for every TAGC$[\approx, \not\approx, \mathbb{N}]$, we can construct a formula ϕ in EMSO$[+1, \approx, \not\approx, \mathbb{N}]$, whose set of models is $\mathcal{L}(\mathcal{A})$.

Note that EMSO$[+1, \approx]$ is strictly more expressive than MSO, since the equality between subterms is not expressible in MSO (see *e.g.* [CLDG$^+$07]). The TA construction of [TW68] for the decidability of MSO[+1] involves the closure under projection on components for TA languages over signatures made tuples of symbols (for the elimination of $\exists$ quantifiers). TAGC languages are not closed under projection, as it is already the case for simpler form tree automata with equality [Tre00].

5.2 MSO on Unranked Ordered Terms

In unranked ordered terms of $\mathcal{U}(\Sigma)$, the number of children of a position is unbounded. Therefore, for navigating in such terms with logical formulae, the successor predicates $S_i(x, y)$ of Section 5.1 are not sufficient. In order to describe unranked ordered terms as models, we replace these above predicates S_i by:

- $S_\downarrow(x, y)$ (y is a child of x),
- $S_\rightarrow(x, y)$ (y is the successor sibling of x).

The type of these predicates is still called $+1$. Note that the above predicates $S_1, S_2, \ldots$ can be expressed using these two predicates only.

The validity of the above atoms in a term $t \in \mathcal{U}(\Sigma)$ under a variable assignment σ is defined as follows.

$$\begin{array}{lcll} t, \sigma & \models & S_\downarrow(x, y) & \text{iff there exists } i \text{ such that } \sigma(y) = \sigma(x).i, \\ t, \sigma & \models & S_\rightarrow(x, y) & \text{iff there exists } p \in \mathcal{P}os(t) \text{ and } i \text{ such that } \sigma(x) = p.i \\ & & & \text{and } \sigma(y) = p.i + 1. \end{array}$$

It is shown in [SSM03] that the extension MSO$[+1, |.|_{\mathbb{Z}}]$ is undecidable for unranked ordered terms when counting constraints are applied to sibling positions.

Using the results of Section 4, and an easy adaptation of the automata construction in the proof of Theorem 5.41, we can generalize Theorem 5.41 to EMSO over unranked ordered terms.

Theorem 5.42 *EMSO*$[+1, \approx, \not\approx, \mathbb{N}]$ *is decidable on unranked ordered terms.*

6 Conclusion

We have proposed a decision algorithm for the emptiness problem of the class of TAGC, generalizing the result we had in chapter 4 for TAGED. Our method for emptiness decision, presented in section 2 appeared to be robust enough to deal with several extensions like global counting

constraints, local equality tests between sibling subterms and extension to unranked terms. It has been since extended to equality modulo commutativity in a yet unpublished work. Another interesting subject mentioned in the introduction is the combination of the HAGC of section 4 with the unranked tree automata with tests between siblings [WL07, LW09]. We come back to this point in the conclusion of the thesis.

There are some similarities between the technique presented in chapter 4 and the one presented here. They are both handling (dis)equalities depending on the height of the subterms. For each given height, they seek for terms below that are able to satisfy the constraints, and at the same time, to provide the states expecting by the transition rules. The termination of such a procedure depends on whether the number of those sets of terms for a given height is finite or not. In chapter 4, this property is satisfied because we manage to use only a finite number of milestones. Here, it works because we can order these sets with a well quasi-order, hence the non-primitive recursive complexity. A deeper study of these common properties could help in refining the complexity of those emptiness decision problems.

Another branch of research related to TAGC concerns automata and logics for *data trees*, *i.e.* trees labeled over an infinite (countable) alphabet (see [Seg06] for a survey). Indeed, data trees can be represented by terms over a finite alphabet, with an encoding of the data values into terms. This can be done in several ways, and with such encodings, the data equality relation becomes the equality between subterms. Therefore, this could be worth studying in order to relate our results on TAGC to decidability results on automata or logics on data trees like those in [Jur07, BMSS09]. Testing (dis)equalities of data with these techniques does not require the use of different tests at positions that are may be prefixes one from another. Hence, it would only use a rather limited class of TAGC, and maybe lead to some better complexity. On the other hand, representing data as terms allows to manipulate the structures of the data in order to test relations more complex than simple (dis)equalities, which are generally avoided by data automata since it easily leads to undecidability. Therefore, this approach would probably provide a less expressive model for representing XML data data documents.

Conclusion

With this thesis, we contribute to a deeper and better understanding of the tree automata with global equality and disequality constraints. We provide a model, that is TAGC[$\approx, \not\approx$], that generalizes the previously existing model of TAGED. We proposed an extensive comparative study of this model, relatively to the previously existing ones.

Our major contributions are the following:

- we provided one of the first result of decidability of membership for the closure of a language of tree automata with equality constraints by a term rewriting system;
- we answer positively the emptiness problem for TAGED, and show that it is in NEXPTIME;
- we prove the decidability of the emptiness problem for the more general class of TAGC. This procedure is robust enough to handle tree automata with both global and local equality and disequality constraints.

In addition to these results, we also made some more modest contributions, which can help a better understanding of the behavior of tree automata with global constraints:

- we proved the undecidability of the regularity problem of languages of tree automata with global equality constraints;
- we showed that the class of TAGED containing only disequality constraints and the t-dag automata are equally expressive.

We already discussed longly how our result on rewrite closure of the Rigid Tree Automata could be applied to communication protocols verification, and more generally how tree automata with global constraints could help in this domain. It seems that those intuitions were right, as TAGED have been since applied in a new model-checking result (see [CHK09]), but the way they are used is apparently slightly different from our approach. Those applications need efficient implementations of decision algorithms. Some work has been done to encode the membership decision problem in a SAT-solver in an efficient way in [HHK10]. Both

for communication protocols and for the context of XML programming, it would be very useful, but also challenging, to have a (quite) efficient implementation of the emptiness problem for tree automata with global constraints.

One major problem to handle is a lower bound for the complexity of the emptiness problem. For TAGED, we show that the problem is in NEXPTIME, but we do not have a lower bound. However, the proof of the decidability of emptiness for t-dag automata that inspired our work has been proved NP-complete. Since, we do a very similar proof using an automaton which is exponential in the size of the input, it seems reasonable to assume that one can prove that the emptiness decision problem is NEXPTIME-complete.

Another complexity issue that this thesis arises, is whether the decidability of the emptiness problem for TAGC is elementary, *i.e.* does our solution provide an optimal complexity. In order to prove this, one will need to find an explicit elementary decision or procedure. However, the stronger expressiveness of TAGC compared to TAGED, and the robustness of our decision procedure let us think that there is a complexity gap between our two results. It is maybe possible to encode this problem into one that is already known to be, for example, non-primitive recursive. Some recent works provide such problems, like decision problems on lossy-channel systems, or the Post Embedding Problem [CS07].

An interesting further work would be to compare the expressiveness of TAGC with automata on data trees (like the ones studied in [BMSS09]). As explained earlier, it would require to encode data on trees as extra subterms, which causes several complications that we already discussed. Another suggestion, that we were given when presenting our results, is to use query-like techniques to select nodes that will later be tested for equality or disequality in global constraints. At a first approach, it seems that we may have a model of automata that could be both powerful and with a decidable emptiness problem. Moreover, using queries on unranked ordered labeled trees could be a way to decide emptiness for unranked tree automata with both global constraints of TAGC, and constraints between sibling subterms like in the model UTASC of [LW09] which are not handled by TAGC. Hence, we would get a very powerful model combining the expressive power of both TAGC and UTASC.

Bibliography

[AC02] Roberto M. Amadio and Witold Charatonik. On name generation and set-based analysis in the dolev-yao model. In *13th International Conference on Concurrency Theory, CONCUR*, volume 2421 of *LNCS*, pages 499–514. Springer, 2002. ISBN 3-540-44043-7.

[ACL09] Reynald Affeldt and Hubert Comon-Lundh. Verification of security protocols with a bounded number of sessions based on resolution for rigid variables. In *Formal to Practical Security*, volume 5458 of *LNCS, State-of-the-Art Survey*, pages 1–20. Springer, May 2009. URL `http://staff.aist.go.jp/reynald.affeldt/documents/rigid.pdf`.

[AF01] Martin Abadi and Cédric Fournet. Mobile values, new names, and secure communication. In *28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL*, pages 104–115, 2001.

[AM04] Rajeev Alur and Parthasarathy Madhusudan. Visibly pushdown languages. In *36th Annual ACM Symposium on Theory of Computing, STOC*, pages 202–211. ACM, 2004. ISBN 1-58113-852-0.

[And81] Peter B. Andrews. Theorem proving via general matings. *Journal of the ACM*, 28(2):193–214, 1981.

[ANR05] Siva Anantharaman, Paliath Narendran, and Michaël Rusinowitch. Closure properties and decision problems of dag automata. *Information Processing Letters*, 94(5):231–240, 2005.

[BCG+10] Luis Barguñó, Carles Creus, Guillem Godoy, Florent Jacquemard, and Camille Vacher. The emptiness problem for tree automata with global constraints. In *25th Annual IEEE Symposium on Logic In Computer Science (LICS)*, pages 263–272, 2010.

[BMSS09] Mikoljav Bojańczyk, Anca Muscholl, Thomas Schwentick, and Luc Segoufin. Two-variable logic on data trees and

applications to xml reasoning. *Journal of the ACM*, 56(3), 2009.

[BT92] Bruno Bogaert and Sophie Tison. Equality and disequality constraints on direct subterms in tree automata. In *9th Symp. on Theoretical Aspects of Computer Science, STACS*, volume 577 of *Lecture Notes in Computer Science*, pages 161–171. Springer, 1992.

[BT05] Ahmed Bouajjani and Tayssir Touili. On computing reachability sets of process rewrite systems. In *Term Rewriting and Applications, 16th International Conference, RTA 2005, Nara, Japan, April 19-21, 2005, Proceedings*, volume 3467 of *Lecture Notes in Computer Science*, pages 484–499. Springer, 2005.

[CC04] Hubert Comon-Lundh and Véronique Cortier. Security properties: Two agents are sufficient. *Science of Computer Programming*, 50(1-3):51–71, March 2004.

[CG03] Luca Cardelli and Giorgio Ghelli. Tql: A query language for semistructured data based on the ambient logic. In *Mathematical Structures in Computer Science*, page 2004, 2003.

[Cha99] Wiltord Charatonik. Automata on dag representations of finite trees. Technical report, Max-Planck-Institut für Informatik, Saarbrücken, 1999.

[CHK09] Roméo Courbis, Pierre-Cyrille Héam, and Olga Kouchnarenko. Taged approximations for temporal properties model-checking. In *Proceedings of the 14th International Conference on Implementation and Application of Automata (CIAA'09)*, volume 5642 of *Lecture Notes in Computer Science*, pages 115–124, 2009.

[CLC05] Hubert Comon-Lundh and Véronique Cortier. Tree automata with one memory, set constraints and cryptographic protocols. *Theoretical Computer Science*, 331(1): 143–214, 2005.

[CLDG+07] Hubert Comon-Lundh, Max Dauchet, Rémi Gilleron, Florent Jacquemard, Christof Löding, Denis Lugiez, Sophie Tison, and Marc Tommasi. Tree automata techniques and applications. `http://tata.gforge.inria.f`, 2007.

[CLJP07] Hubert Comon-Lundh, Florent Jacquemard, and Nicolas Perrin. Tree automata with memory, visibility and structural constraints. In *Proceedings of the 10th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS)*. Springer, 2007.

[CP94] Witold Charatonik and Leszek Pacholski. Set constraints with projections are in NEXPTIME. In *Proceedings of the* 35^{th} *Symposium Foundations of Computer Science*, pages 642–653, 1994.

[CR07] Jacques Chabin and Pierre Réty. Visibly pushdown languages and term rewriting. In *6th International Symposium on Frontiers of Combining Systems, FroCos*, volume 4720 of *LNCS*, pages 252–266. Springer, 2007. ISBN 978-3-540-74620-1.

[CS07] Pierre Chambart and Philippe Schnoebelen. Post embedding problem is not primitive recursive, with applications to channel systems. In V. Arvind and Sanjiva Prasad, editors, *Proceedings of the 27th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'07)*, volume 4855 of *Lecture Notes in Computer Science*, pages 265–276. Springer, New Delhi, India, December 2007.

[DCC95] Max Dauchet, Anne-Cécile Caron, and Jean-Luc Coquidé. Automata for Reduction Properties Solving. *Journal of Symbolic Computation*, 20(2):215–233, 1995.

[DLL07] Stéphanie Delaune, Hai Lin, and Christopher Lynch. Protocol verification via rigid/flexible resolution. In *14th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning, LPAR*, volume 4790 of *LNCS*, pages 242–256. Springer, 2007.

[FGT04] Guillaume Feuillade, Thomas Genet, and Valérie Viet Triem Tong. Reachability Analysis over Term Rewriting Systems. *Journal of Automated Reasoning*, 33(3-4):341–383, 2004.

[FL02] Wenfei Fan and Leonid Libkin. On XML integrity constraints in the presence of DTDs. *Journal of the ACM*, 39: 368–406, 2002.

[FSVY91] Thom W. Frühwirth, Ehud Y. Shapiro, Moshe Y. Vardi, and Eyal Yardeni. Logic programs as types for logic programs. In *6th IEEE Symposium on Logic in Computer Science*, pages 300–309, 1991.

[FTT07] Emmanuel Filiot, Jean-Marc Talbot, and Sophie Tison. Satisfiability of a spatial logic with tree variables. In *Proceedings of the 21st International Workshop on Computer Science Logic (CSL 2007)*, volume 4646 of *Lecture Notes in Computer Science*, pages 130–145. Springer, 2007.

[FTT08] Emmanuel Filiot, Jean-Marc Talbot, and Sophie Tison. Tree automata with global constraints. In *12th International Conference in Developments in Language Theory (DLT 2008)*, volume 5257 of *Lecture Notes in Computer Science*, pages 314–326. Springer, 2008.

[Gal91] Jean H. Gallier. What's so special about kruskal's theorem and the ordinal γ_0? a survey of some results in proof theory. *Annals of Pure Applied Logic*, 53(3):199–260, 1991.

[GK00] Thomas Genet and Francis Klay. Rewriting for Cryptographic Protocol Verification. In *Proc. of 17th Int. Conf. on Automated Deduction, CADE*, volume 1831 of *LNCS*, pages 271–290. Springer, 2000.

[Gou05] Jean Goubault-Larrecq. Deciding $\mathcal{H}_1$ by Resolution. *Information Processing Letters*, 95(3):401–408, 2005.

[Gue83] Irène Guessarian. Pushdown tree automata. *Mathematical Systems Theory*, 16(1):237–263, 1983.

[HHK10] Pierre-Cyrille Héam, Vincent Hugot, and Olga Kouchnarenko. Sat solvers for queries over tree automata with constraints. In *Proceedings of the 2nd Workshop on Constraints in Software Testing, Verification and Analysis (CSTVA)*. IEEE publication, 2010.

[JKV] Florent Jacquemard, Francis Klay, and Camille Vacher. Rigid tree automata. *Information and Computation*. To appear.

[JKV09] Florent Jacquemard, Francis Klay, and Camille Vacher. Rigid tree automata. In *Proceedings of the 3rd International Conference on Language and Automata Theory and Applications, (LATA'09)*, volume 5457 of *Lecture Notes in Computer Science*, pages 446–457. Springer, 2009.

[JRV08] Florent Jacquemard, Michaël Rusinowitch, and Laurent Vigneron. Tree automata with equality constraints modulo equational theories. *Journal of Logic and Algebraic Programming*, 75(2):182–208, April 2008.

[Jur07] *Alternation-free modal mu-calculus for data trees*. IEEE Comp. Society, 2007.

[KR02] Felix Klaedtke and Harald Ruess. Parikh automata and monadic second-order logics with linear cardinality constraints. Technical Report TR 177, Intitute of Computer Science at Freiburg University, 2002.

[LW09] Christof Löding and Karianto Wong. On nondeterministic unranked tree automata with sibling constraints. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2009)*. Leibniz International Proceedings in Informatics, 2009.

[Mon81] Jocelyne Mongy. *Transformation de noyaux reconnaissables d'arbres. Forêts RATEG*. PhD thesis, Laboratoire d'Informatique Fondamentale de Lille, Université des Sciences et Technologies de Lille, 1981.

[Mur99] Makoto Murata. Hedge automata: a formal model for XML schemata. Technical report, Fuji Xerox Information Systems, 1999.

[NPTT05] Joachim Niehren, Laurent Planque, Jean-Marc Talbot, and Sophie Tison. N-ary queries by tree automata. In *Foundations of Semistructured Data*, 2005.

[Pad88] Peter Padawitz. *Computing in Horn clause theories*. Springer-Verlag New York, Inc., New York, NY, USA, 1988. ISBN 0-387-19427-4.

[Sal88] Kai Salomaa. Deterministic tree pushdown automata and monadic tree rewriting systems. *J. Comput. Syst. Sci.*, 37 (3):367–394, 1988.

[Sch07] Thomas Schwentick. Automata for xml - a survey. *Journal of Computer and System Sciences*, 73(3):289–315, 2007.

[Seg06] Luc Segoufin. Automata and logics for words and trees over an infinite alphabet. In *Computer Science Logic*, volume 4207 of *Lecture Notes in Computer Science*. Springer, 2006.

[Sei94] Helmut Seidl. Haskell overloading is dexptime-complete. *Information Processing Letters*, 52(2):57–60, 1994. ISSN 0020-0190.

[SSM03] Helmut Seidl, Thomas Schwentick, and Anca Muscholl. Numerical document queries. In *Principle of Databases Systems (PODS)*, pages 155–166. ACM Press, 2003.

[Tre00] Ralf Treinen. Predicate logic and tree automata with tests. In *FoSSaCS*, pages 329–343, 2000.

[TW68] James W. Thatcher and Jesse B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical System Theory*, 2:57–82, 1968.

[VGL07] Kumar Neeraj Verma and Jean Goubault-Larrecq. Alternating two-way ac-tree automata. *Information and Computation*, 205(6):817–869, 2007.

[WL07] Karianto Wong and Christof Löding. Unranked tree automata with sibling equalities and disequalities. In *Proceedings of the 34th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 4596 of *Lecture Notes in Computer Science*, pages 875–887. Springer, 2007.

[YOM91] Suguru Yamaguchi, Kiyohiko Okayama, and Hideo Miyahara. The design and implementation of an authentication system for the wide area distributed environment. *IEICE Transactions on Information and Systems*, E74(11):3902–3909, November 1991.