

Satisfiability of a Spatial Logic with Tree Variables

Emmanuel Filiot¹ Jean-Marc Talbot² Sophie Tison¹

¹ INRIA Futurs, Mostrare Project, University of Lille 1 (LIFL, UMR 8022 of CNRS)

² University of Provence (LIF, UMR 6166 of CNRS), Marseille

Abstract. We investigate in this paper the spatial logic TQL for querying semi-structured data, represented as unranked ordered trees over an infinite alphabet. This logic consists of usual Boolean connectives, spatial connectives (derived from the constructors of a tree algebra), tree variables and a fixpoint operator for recursion. Motivated by XML-oriented tasks, we investigate the guarded TQL fragment. We prove that for closed formulas this fragment is MSO-complete. In presence of tree variables, this fragment is strictly more expressive than MSO as it allows for tree (dis)equality tests, i.e. testing whether two subtrees are isomorphic or not. We devise a new class of tree automata, called TAGED, which extends tree automata with global equality and disequality constraints. We show that the satisfiability problem for guarded TQL formulas reduces to emptiness of TAGED. Then, we focus on bounded TQL formulas: intuitively, a formula is bounded if for any tree, the number of its positions where a subtree is captured by a variable is bounded. We prove this fragment to correspond with a subclass of TAGED, called bounded TAGED, for which we prove emptiness to be decidable. This implies the decidability of the bounded guarded TQL fragment. Finally, we compare bounded TAGED to a fragment of MSO extended with subtree isomorphism tests.

1 Introduction

In this paper, we consider the spatial logic TQL [7]. Spatial logics have been used to express properties of structures such as trees [7], graphs [6, 12] and heaps [19]. The main ingredients of spatial logics are spatial connectives: roughly speaking, these connectives are derived from operators that can be used to generate the domain of interpretation.

The logic we consider here is interpreted over hedges (*i.e.* unranked ordered trees) labelled over an infinite alphabet. The logic integrates Boolean connectives, spatial connectives (derived from the constructors of an unranked ordered tree algebra), tree variables and a fixpoint operator for recursion.

We focus on the satisfiability problem of TQL. It is quite simple to prove that the full TQL logic over unranked ordered trees even without tree variables is undecidable. We then focus on the guarded fragment which ensures that recursive variables have to be guarded by tree extension. We show that guarded TQL logic without tree variables is equivalent to the monadic second order logic (MSO).

However, when tree variables are considered, things are getting more complicated. Indeed, we can express that two non-empty paths starting from a node of a tree lead to two isomorphic subtrees, which goes beyond regularity over unranked trees.

Still about expressiveness of this logic, an infinite alphabet and the ability to test for tree equality allow us to consider some data values. We can write formulas whose models are hedges which violate some key constraints or some functional dependencies.

We focus on bounded TQL formulas: intuitively, a formula is bounded if for any tree, the number of equalities and disequalities that have to be tested – to check non-deterministically that the tree is a model of the formula – is bounded.

We introduce a new class of tree automata, called *tree automata with global equalities and disequalities* (TAGED for short), which extends unranked tree automata A with an equality relation $=_A$ and a disequality relation \neq_A on states. Subtrees of some tree t which evaluates by A to states which are in relation by $=_A$ (resp. by \neq_A) have to be isomorphic (resp. non isomorphic). Naturally, $=_A$ induces an equivalence relation on a subset of nodes of t , but the number of classes of this relation is bounded. E.g., TAGED can express that all subtrees of height n , for some fixed natural n , are equal, but not that for each node of the tree, all the subtrees rooted at its sons are equal. Although it is a natural extension of tree automata, this extension has never been considered.

We show that satisfiability of guarded TQL formulas reduces to emptiness of TAGED. We define a subclass of TAGED, called *bounded TAGED*, for which we can decide emptiness. Intuitively, boundedness ensures that the cardinality of every equivalence class is bounded, which may not be the case for full TAGED. We show emptiness decidability of bounded TAGED.

We complete our proof by constructing a TAGED from a guarded and bounded TQL formula. This construction extends the one from [4] with tree variables. This extension is non-trivial as the automata we have to consider are non-deterministic.

Finally, we define an extension of MSO with a binary relation \sim between nodes; two nodes are in relation if they are roots of two isomorphic subtrees. We consider MSO formulas extended with the predicate \sim . It is easy to see that this extension renders MSO undecidable. However, we prove that if the relation \sim concerns only variables belonging to a prefix of existentially quantified first-order variables, then this extension is decidable. The proof works by reduction to emptiness of bounded TAGED.

Automata dealing with data values have been studied in [18, 3]. However, our motivations are different and we obtain the capability to manage data values as a side-effect. In [3], the authors study two-variables FO logic extended with an equality relation on data values. The expressiveness of this formalism and the one presented here are not comparable: we can test tree isomorphisms while they can test data value equality only, but restricting our logic to data-value equality is strictly less expressive, as we do not have quantifiers.

The paper is organised as follows: in Section 2 we recall definitions for hedges, hedge automata and monadic second order logic. Section 3 describes the TQL logic and results we obtain concerning its satisfiability. Section 4 is dedicated to the tool we use to solve the satisfiability problem, namely tree automata with global equalities and disequalities (TAGED). In Section 5 we relate satisfiability of TQL formulas and emptiness of TAGED. Finally, in Section 6 we propose an extension of MSO with isomorphism tests whose satisfiability problem is decidable.

2 Preliminaries

We consider an infinite set of labels A .

Hedges - Trees Let Σ be the signature $\{0, |\} \cup \{a \mid a \in \Lambda\}$, where 0 is a constant, | a binary symbol and a s unary symbols. We call *hedge* an element of the Σ -algebra *Hedge* obtained by quotienting the free Σ -algebra by the following three axioms:

$$0|h = h \quad h|0 = h \quad (h|h'')|h''' = h|(h''|h''')$$

0 will be the *empty hedge*. We call respectively *trees* and *leaves* hedges of the form $a(h)$ and $a(0)$. We may omit | and write $a(h)b(h')c(h'')$ instead of $a(h)|b(h')|c(h'')$. We define $\text{roots}(h)$ as the word from Λ^* defined recursively as : (i) $\text{roots}(0) = \epsilon$, (ii) $\text{roots}(a(h')) = a$ and, (iii) $\text{roots}(h_1|h_2) = \text{roots}(h_1)\text{roots}(h_2)$.

We will also adopt the graph point of view and consider hedges as a set of vertices V , two disjoint sets of directed edges E_c, E_s and a mapping λ from V to Λ . In a hedge h , one associates a vertex with each occurrence of elements of Λ . There is an edge from E_c (resp. from E_s) from an occurrence a_1 to an occurrence a_2 if the hedge contains a pattern of the form $a_1(h_1|a_2(h)|h_2)$ for some hedges h_1, h_2 (resp. $a_1(h_1)|a_2(h_2)$ for some hedges h_1, h_2).

For every hedge $h = (V, E_c, E_s, \lambda)$, we denote by $\text{nodes}(h)$ the set V and by $\text{lab}_h(u)$ the label $\lambda(u)$, for $u \in V$. We denote $h|_u$ the subtree of h rooted at u , and by \leq the reflexive-transitive closure of E_c . E.g., the root is minimal for \leq in a tree.

For a set of labels L , we denote \mathbb{H}_L (resp. \mathbb{T}_L) the set of hedges (resp. trees) with nodes labelled by elements in L .

Hedge automata [17] A *hedge automaton* A is a 4-tuple (Λ, Q, F, Δ) where Δ is a finite set of rules $\alpha(L) \rightarrow q$ where α is a finite or cofinite set of labels, $L \subseteq Q^*$ is a regular language over states from Q , and $F \subseteq Q^*$ is an accepting regular language.

Definition 1 (runs). Let h be a hedge and A be a hedge automaton. The set of runs $R_A(h) \subseteq \mathbb{H}_Q$ of A on h is the set of hedges over Q inductively defined by:

$$\begin{aligned} R_A(h_1|h_2) &= \{r_1|r_2 \mid r_1 \in R_A(h_1), r_2 \in R_A(h_2)\} & R_A(0) &= \{0\} \\ R_A(a(h)) &= \{q(r) \mid \exists \alpha(L) \rightarrow q \in \Delta, a \in \alpha, r \in R_A(h), \text{roots}(r) \in L\} \end{aligned}$$

Let \bar{q} be a word of states, we denote by $R_{A, \bar{q}}(h) \subseteq R_A(h)$ the set of runs r such that $\text{roots}(r) = \bar{q}$, and often say that h evaluates to \bar{q} by A . The set of accepting runs of A on h , denoted by $R_A^{\text{acc}}(h)$, is defined by $\{r \mid \exists \bar{q} \in F, r \in R_{A, \bar{q}}(h)\}$.

The language accepted by A , denoted $L(A)$, is defined by $\{h \mid R_A^{\text{acc}}(h) \neq \emptyset\}$.

Testing emptiness of the language accepted by a hedge automaton is decidable [5].

MSO The logic MSO (Monadic Second Order logic) is the extension of the first-order logic FO with the possibility to quantify over unary relations, *i.e.* over sets.

Let σ be the signature $\{\text{lab}_a \mid a \in \Lambda\}$ where lab_a s are unary predicates. We associate with a hedge $h = (V, E_c, E_s, \lambda)$ a finite σ -structure $\mathcal{S}^h = \langle V, \{\text{ch}, \text{ns}\} \cup \{\text{lab}_a^h \mid a \in \Lambda\} \rangle$, such that $\text{lab}_a^h(v)$ (resp. $\text{ch}(v, v')$, $\text{ns}(v, v')$) holds in \mathcal{S}^h if $\lambda(v) = a$ (resp. $(v, v') \in E_c$, $(v, v') \in E_s$).

We assume a countable set of first-order variables ranging over by x, y, z, \dots and a countable set of second-order variables ranging over by X, Y, Z, \dots

MSO formulas are given by the following syntax:

$$\psi ::= \text{lab}_a(x) \mid \text{ch}(x, y) \mid \text{ns}(x, y) \mid x \in X \mid \psi \vee \psi \mid \neg \psi \mid \exists x. \psi \mid \exists X. \psi$$

$\phi ::= 0$	empty hedge	$\llbracket 0 \rrbracket_{\rho, \delta}$	$= \{0\}$
\top	truth	$\llbracket \top \rrbracket_{\rho, \delta}$	$= \mathbb{H}_\Lambda$
$\alpha[\phi]$	extension	$\llbracket \alpha[\phi] \rrbracket_{\rho, \delta}$	$= \{a(h) \mid h \in \llbracket \phi \rrbracket_{\rho, \delta}, a \in \alpha\}$
$\phi \phi'$	composition	$\llbracket \phi \phi' \rrbracket_{\rho, \delta}$	$= \{h h' \mid h \in \llbracket \phi \rrbracket_{\rho, \delta}, h' \in \llbracket \phi' \rrbracket_{\rho, \delta}\}$
$\neg\phi$	negation	$\llbracket \neg\phi \rrbracket_{\rho, \delta}$	$= \mathbb{H}_\Lambda \setminus \llbracket \phi \rrbracket_{\rho, \delta}$
$\phi \vee \phi'$	disjunction	$\llbracket \phi \vee \phi' \rrbracket_{\rho, \delta}$	$= \llbracket \phi \rrbracket_{\rho, \delta} \cup \llbracket \phi' \rrbracket_{\rho, \delta}$
X	tree variable	$\llbracket X \rrbracket_{\rho, \delta}$	$= \{\rho(X)\}$
ξ	recursion variables	$\llbracket \xi \rrbracket_{\rho, \delta}$	$= \delta(\xi)$
$\mu\xi.\phi$	least fixpoint	$\llbracket \mu\xi.\phi \rrbracket_{\rho, \delta}$	$= \bigcap \{S \subseteq \mathbb{H}_\Lambda \mid \llbracket \phi \rrbracket_{\rho, \delta[\xi \mapsto S]} \subseteq S\}$
ϕ^*	iteration	$\llbracket \phi^* \rrbracket_{\rho, \delta}$	$= 0 \cup \bigcup_{i>0} \underbrace{\llbracket \phi \rrbracket_{\rho, \delta} \dots \llbracket \phi \rrbracket_{\rho, \delta}}_{i \text{ times}}$

(a) Syntax

(b) Semantics

Fig. 1. TQL logic

Let \mathcal{S} be a σ -structure with domain V . Let ρ be a valuation mapping first-order variables to elements from V and second-order variables to subsets of V . We write $\mathcal{S} \models_\rho \psi$ when the structure \mathcal{S} is a model of the formula ψ under the valuation ρ ; this is defined in the usual Tarskian manner and we have in particular, (i) $\psi \models_\rho \text{lab}_a(x)$ if $\text{lab}_a(\rho(x))$ holds in \mathcal{S} , (ii) $\psi \models_\rho \text{ch}(x, y)$ if $\text{ch}(\rho(x), \rho(y))$ holds in \mathcal{S} , (iii) $\psi \models_\rho \text{ns}(x, y)$ if $\text{ns}(\rho(x), \rho(y))$ holds in \mathcal{S} .

A set of hedges S is *MSO-definable* if there is an MSO sentence ψ such that $S = \{h \mid h \models \psi\}$. It is well-known that a language is accepted by some hedge automata iff it is MSO-definable.

3 The Tree Query Logic

We consider here a fragment of the TQL logic defined in [7] and adapt it to unranked ordered trees.

Syntax We assume a countable set \mathcal{T} of tree variables ranging over by X, Y , and a countable set \mathcal{R} of recursion variables ranging over by ξ . Let α be a finite or co-finite set of labels from Λ . Formulas ϕ from TQL are given by the syntax on Figure 1(a). We allow cofinite sets in extensions, otherwise we could not express formula $\Lambda[0]$.

We assume that μ is the binder for recursion variables and the notions of bound and free variables are defined as usual. To ensure the existence of fixpoint, we will assume that in formulas $\mu\xi.\phi$, the recursion variable ξ occurs under an even number of negations. A formula is said to be *recursion-closed* if all the occurrences of its recursion variables are bound. A TQL *sentence* is a recursion-closed formula that does not contain tree variables. A TQL formula ϕ is *guarded* if for any of its subformula $\mu\xi.\phi'$, the variable ξ occurs in the scope of some extension operator $\alpha[\]$ in ϕ' .

We assume from now on that recursion variables are bound only once in formulas and denote $\text{recvar}(\phi)$ (resp. $\text{var}(\phi)$) the set of recursion variables (resp. tree variables) occurring in ϕ . We may write $a[\phi]$ instead of $\{a\}[\phi]$.

Semantics Interpretation maps a TQL formula to a set of hedges. Let ρ be an assignment of tree variables into \mathbb{T}_Λ and δ be an assignment of the recursion variables into sets of hedges. The interpretation of the formula ϕ under ρ and δ , denoted by $\llbracket \phi \rrbracket_{\rho, \delta}$ is inductively defined and given on Figure 1(b).

Examples Let us consider the following formulas:

$$\phi = a[\top]|\top \quad (1)$$

$$\phi_s = \mu\xi.(a[\top]|\xi \vee 0) \quad (2)$$

$$\phi_{dtd} = (\text{employee}[\text{name}[\Lambda[0]] | \text{dpt}[\Lambda[0]] | \text{manager}[\Lambda[0]]])^* \quad (3)$$

$$\phi_{dd} = \phi_{dtd} \wedge \top | \text{employee}[X] | \top | \text{employee}[X] | \top \quad (4)$$

$$\phi_{\text{path}(a),0} = \mu\xi.((\top | a[\xi] | \top) \vee 0) \quad (5)$$

The above formula ϕ is interpreted as the set of hedges of length at least 1, such that the root of the first tree is labelled a . The formula ϕ_s is interpreted as the set of hedges $\{a[h_1] \dots | a[h_n] | h_i \in \mathbb{H}_\Lambda, n \geq 0\}$. The formula ϕ_{dtd} is interpreted as the set of hedges defining employees by their name, the department they work in, and their manager whereas the formula ϕ_{dd} expresses that an employee occurs twice in the database. Finally, the models of the formula $\phi_{\text{path}(a),0}$ are hedges with a path labelled by a s from one of the roots to some leaf (*i.e.* the empty hedge 0).

The formula ϕ_{odd} is interpreted as the set of hedges having an odd number of nodes:

$$\begin{aligned} \phi_{\text{odd}} &= \mu\xi_o.(\Lambda[0] \vee \Lambda[\phi_{\text{even}}(\xi_o)]|\phi_{\text{even}}(\xi_o) \vee \Lambda[\xi_o]|\xi_o \\ &\text{where } \phi_{\text{even}}(\xi_o) = \mu\xi_e.(0 \vee (\Lambda[\xi_o]|\xi_e \vee \Lambda[\xi_e]|\xi_o)) \end{aligned}$$

Let us denote $\phi_{\text{path}(L),\psi} = \mu\xi.((\top | L[\xi] | \top) \vee \psi)$ the formula whose models are hedges containing a path labelled by elements from L from one of the roots to a hedge satisfying ψ . The models of the following formula are the hedges having two non-empty paths labelled respectively by a s and by b s from two of the top-level nodes, those two paths leading to some identical subtrees

$$\top | (a[\phi_{\text{path}(a),X}] | \top | b[\phi_{\text{path}(b),X}] \vee b[\phi_{\text{path}(b),X}] | \top | a[\phi_{\text{path}(a),X}] | \top$$

The formula $\phi_{\text{id.not.key}}$ is interpreted as the set of hedges for which two nodes labelled id have identical subtrees (roughly speaking “the (data) value of the element id can not be used as a key in this XML document”)

$$\phi_{\text{id.not.key}} = \top | \Lambda[\phi_{\text{path}(\Lambda),\text{id}[X]}] | \top | \Lambda[\phi_{\text{path}(\Lambda),\text{id}[X]}] | \top$$

The following formula states that two trees *employee* have identical subtrees rooted by *dpt* but different subtrees rooted by *manager*

$$\begin{aligned} \phi_{dtd} \wedge \top | \text{employee}[\top | \text{dpt}[X] | \text{manager}[Y]] | \top \\ | \text{employee}[\top | \text{dpt}[X] | \text{manager}[\neg Y]] | \top \end{aligned}$$

A hedge satisfying this formula may be considered as ill-formed assuming the existence of some functional dependency stating that *department* has only one *manager*.

Models of the formula ϕ_{branch} are the trees whose shapes are described on Figure 2.

$$\phi_{\text{branch}} = a[X]|\mu\xi.(a[X]|\xi \vee \neg X \wedge \Lambda[\top])$$

ϕ_s and ϕ_{odd} are the only two formulas that are not guarded; however, ϕ_s is equivalent to $a[\top]^*$, which is guarded and ϕ_{odd} to the following guarded formula

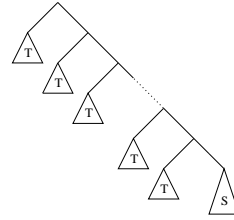


Fig. 2. A tree with $T \neq S$

$$\begin{aligned}\phi_{odd} &= \mu\xi_o.\Lambda[\phi_{even}(\xi_o) \vee (\xi_o|\phi_{even}(\xi_o)^*|\xi_o)^*] \\ \phi_{even}(\xi_o) &= \mu\xi_e.\Lambda[\xi_e^*\xi_o|(\phi_{even}(\xi_o) \vee (\xi_o|\phi_{even}(\xi_o)^*|\xi_o)^*) \vee 0\end{aligned}$$

But the formula $\mu\xi.(a[0]|\xi|b[0] \vee 0)$ is neither guarded nor equivalent to any guarded formula.

Definition 2 (satisfiability). *A recursion-closed TQL formula ϕ is satisfiable if there exists a hedge h and an assignment ρ such that $h \in \llbracket \phi \rrbracket_\rho$.*

TQL sentences It is easy to prove that TQL sentences can describe sets of hedges that are not MSO-definable; for instance, the TQL sentence $\mu\xi.(a[0]|\xi|b[0] \vee 0)$ describes a “flat” hedge of the form $(a[0]^n b[0]^n)$ for $n \in \mathbb{N}$.

Theorem 1. *For any set of hedges S , there exists a guarded TQL sentence ϕ such that $\llbracket \phi \rrbracket = S$ iff S is MSO-definable.*

As a consequence of Theorem 4, satisfiability is decidable for guarded TQL sentences. This restriction is crucial, since, by reduction of emptiness problem for the intersection of two context-free grammars, one can prove that:

Theorem 2. *Satisfiability for TQL sentences is undecidable.*

Adding quantification As in [7], one could also consider quantification over tree variables $\exists X$ and $\forall X$ with the following semantics:

$$\llbracket \exists X.\phi \rrbracket_{\rho,\delta} = \bigcup_{t \in \mathbb{T}} \llbracket \phi \rrbracket_{\rho[X \rightarrow t],\delta} \quad \llbracket \forall X.\phi \rrbracket_{\rho,\delta} = \bigcap_{t \in \mathbb{T}} \llbracket \phi \rrbracket_{\rho[X \rightarrow t],\delta}$$

where $\rho[X \rightarrow t]$ is the assignment identical to ρ except that it associates t with X .

Hence, the satisfiability problem from Definition 2 is equivalent to the one of closed formulas of the form $\exists X_1 \dots \exists X_n \phi$ where ϕ is a recursion-closed TQL formula, i.e. the existence of a tree satisfying this formula.

For more complicated alternation of quantifiers, one can easily adapt the proof from [8] about the undecidability of the fragment of TQL without iteration, recursion and tree variable but with quantification over labels to prove that

Theorem 3. *Satisfiability for recursion-closed TQL formulas with quantification is undecidable (this holds even for recursion-free formulas).*

Bounded TQL formulas In bounded formulas, variables can occur in recursion only in a restricted manner: intuitively a formula is bounded if there exists a bound on the number of equality test performed to (non-deterministically) verify that a hedge is a model of the formula. Boundedness appears naturally in unification problems and in pattern languages (where variables appears a bounded number of times in terms or patterns). But defining boundedness in the presence of recursion is a bit more technical.

In the examples we have given so far, the only formula that is not bounded is ϕ_{branch} . The following formula is also not bounded as it asks each subtree of the hedge to be different from X : $\neg(\mu\xi.\top|(X \vee \Lambda[\xi])|\top)$

We let β be a recursion-closed formula s.t. no recursion variable is bound twice, and denote by ϕ_ξ the formula s.t. $\mu\xi.\phi_\xi$ is a subformula of β . To define *boundedness* formally, we introduce, for every subformula ϕ of β , the *generalized free variables* of ϕ , denoted $\text{var}_\beta^*(\phi)$, as the least solution of the following (recursive) equations:

$$\begin{aligned} \text{var}_\beta^*(0) &= \text{var}_\beta^*(\top) = \emptyset & \text{var}_\beta^*(a[\phi]) &= \text{var}_\beta^*(\neg\phi) = \text{var}_\beta^*(\mu\xi.\phi) = \text{var}_\beta^*(\phi^*) = \text{var}_\beta^*(\phi) \\ \text{var}_\beta^*(\xi) &= \text{var}_\beta^*(\phi_\xi) & \text{var}_\beta^*(X) &= \{X\} & \text{var}_\beta^*(\phi \vee \phi') &= \text{var}_\beta^*(\phi| \phi') = \text{var}_\beta^*(\phi) \cup \text{var}_\beta^*(\phi') \end{aligned}$$

Note that the least solution of these equations is computable. An operator occurs positively (resp. negatively) in a formula if it occurs under an even (resp. odd) number of negations.

A formula β is *bounded* if it satisfies the following properties:

1. for any subformula ϕ^* , $\text{var}_\beta^*(\phi) = \emptyset$;
2. for any subformula $\phi| \phi'$ where $|$ occurs negatively, $\text{var}_\beta^*(\phi) = \text{var}_\beta^*(\phi') = \emptyset$.
3. each formula $\phi| \phi'$ where $|$ occurs positively and each formula $\phi \vee \phi'$ where \vee occurs negatively satisfy $\forall \xi \in \text{recvar}(\phi), \text{var}_\beta^*(\xi) \cap \text{var}_\beta^*(\phi') = \emptyset$, and $\forall \xi' \in \text{recvar}(\phi'), \text{var}_\beta^*(\xi') \cap \text{var}_\beta^*(\phi) = \emptyset$.

As a consequence of Theorem 1, this fragment is strictly more expressive than guarded TQL sentences, as it allows for tree isomorphism tests. Combining Theorems 6 and 5 of the next two sections proves our main result:

Theorem 4. *Satisfiability is decidable for bounded guarded TQL formulas.*

Remarks Our logic can be seen as an extension of the (recursive) pattern-language of XDuce [13]. The main difference here is that we allow Boolean operators and drop the *linear* condition for variables of XDuce. The pattern-matching mechanism of CDuce [1] extends the one from XDuce with Boolean operations and weaker conditions on variables. However, no equality tests between terms can be performed making our logic more powerful. Since we consider an infinite alphabet and we allow equality tests between trees, we can, as a side effect, simulate data values as done in some of the examples we gave.

4 Tree Automata with Global Equalities and Disequalities

In this section, we present a new extension of hedge automata (called TAGED) with the ability to test tree equalities or disequalities globally on the run. We prove decidability of emptiness for a particular class of TAGED, called bounded TAGED, which we use to decide satisfiability of bounded TQL formulas.

4.1 Definitions

Definition 3 (TAGED). *A tree automaton with global equalities and disequalities (TAGED) is a 6-tuple $A = (\Lambda, Q, F, \Delta, =_A, \neq_A)$ such that:*

- (Λ, Q, F, Δ) is a hedge automaton;
- $=_A$ is an equivalence relation on a subset of Q ;
- \neq_A is a non-reflexive symmetric binary relation on Q ;

A TAGED is *positive* if $\neq_A = \emptyset$, and is simply denoted by $A = (\Lambda, Q, F, \Delta, =_A)$. The set $\{q \mid \exists q' \in Q, q =_A q'\}$ is denoted by E , and for all states $q \in E$, we denote by $[q]$ its equivalence class. The set $\{q \mid \exists q' \in Q, q \neq_A q'\}$ is denoted by D . The notion of run differs from hedge automata as we add equality and disequality constraints.

Definition 4 (runs). Let $A = (\Lambda, Q, F, \Delta, =_A, \neq_A)$ be a TAGED. A run r of the hedge automaton (Λ, Q, F, Δ) on a hedge h satisfies the equality constraints if the following holds: $\forall i \in \{1, \dots, n\}, \forall u, v \in \text{nodes}(h), \text{lab}_r(u) =_A \text{lab}_r(v) \implies h|_u = h|_v$.

Similarly, the run r on h satisfies the disequality constraints if the following holds: $\forall i \in \{1, \dots, n\}, \forall u, v \in \text{nodes}(h), \text{lab}_r(u) \neq_A \text{lab}_r(v) \implies h|_u \neq h|_v$.

The set of accepting runs of A on h , denoted $R_A^{\text{acc}}(h)$, is the set of accepting runs r of (Λ, Q, F, Δ) which satisfy the equality and disequality constraints. The language accepted by A , denoted $L(A)$, is the set of hedges h such that $R_A^{\text{acc}}(h) \neq \emptyset$.

Remark that $L(A)$ is not necessarily regular, as illustrated by Example 1.

Example 1. Let Λ be an infinite alphabet. Let $Q = \{q, q_X, q_f\}$, $F = \{q_f\}$, and let Δ be defined as the set of following rules: $\Lambda(q^*) \rightarrow q \quad \Lambda(q^*) \rightarrow q_X \quad a(q_X q_X) \rightarrow q_f$. Let A_1 be the positive TAGED $(\Lambda, Q, F, \Delta, \{q_X =_{A_1} q_X\})$. Then $L(A_1)$ is the set $\{a(t|t) \mid a \in \Lambda, t \in \mathbb{T}_\Lambda\}$, which is known to be non regular [10].

Example 2. Let $Q = \{q, q_X, q_{\bar{X}}, q_f\}$, $F = \{q_f\}$, and let Δ defined as the set of following rules:

$$\Lambda(q^*) \rightarrow q_{\bar{X}} \quad \Lambda(q^*) \rightarrow q \quad \Lambda(q^*) \rightarrow q_X \quad a(q_X(q_{\bar{X}} + q_f)) \rightarrow q_f$$

Let A_2 be the TAGED $(\Lambda, Q, F, \Delta, \{q_X =_{A_2} q_X\}, \{q_X \neq_{A_2} q_{\bar{X}}\})$. Then $L(A_2)$ is the set of hedges whose shapes are described on Figure 2.

Remarks Extensions of tree automata which allow for syntactic equality and disequality tests between subterms have already been defined by adding constraints to automaton rules. E.g., adding the constraint $1.2 = 2$ to a rule means that one can apply the rule at position π only if the subterm at position $\pi.1.2$ is equal to the subterm at position $\pi.2$. Testing emptiness of the recognized language is undecidable in general [16] but two classes with a decidable emptiness problem have been emphasised. In the first class, automata are deterministic and the number of equality tests along a path is bounded [11] whereas the second restricts tests to sibling subterms [2]. This latter class has recently been extended to unranked trees [15], the former one has been extended to equality modulo equational theories [14]. But, contrarily to TAGED, in all these classes, tests are performed locally, typically between sibling or cousin subterms. Finally, automata with local and global equality tests, using one memory, have been considered in [9]. Their emptiness problem is decidable, and they can simulate positive TAGEDs which use at most one state per runs to test equalities, but not general positive TAGEDs.

Definition 5 (bounded TAGED). A bounded TAGED is a 7-tuple $A = (\Lambda, Q, F, \Delta, =_A, \neq_A, k)$ where $A' = (\Lambda, Q, F, \Delta, =_A, \neq_A)$ is a TAGED and $k \in \mathbb{N}$ is a natural. An accepting run r of A on a tree t is an accepting run of A' on t such that the following is true: $|\{u \mid \text{lab}_r(u) \in E \cup D\}| \leq k$.

We say that A and its accepting runs are k -bounded and may write (A', k) instead of A . We say that a TAGED B is equivalent to a bounded TAGED A if $L(A) = L(B)$.

The TAGED of Example 1 is equivalent to the 2-bounded TAGED $(A_1, 2)$, whereas one can prove that the one of Example 2 is not equivalent to any bounded TAGED.

Theorem 5 (emptiness of bounded TAGED). *Let A be a bounded TAGED. It is decidable to know whether $L(A) = \emptyset$ holds or not.*

The rest of this subsection is dedicated to the proof of this theorem, first for positive bounded TAGED, then for full bounded TAGED.

4.2 Configurations

We define a tool called *configurations* used to decide emptiness of positive bounded TAGED. In this subsection, the 6-tuple $A = (\Lambda, Q, F, \Delta, =_A, k)$ always denotes a positive bounded TAGED. We suppose that A accepts trees only, i.e. $F \subseteq Q$. It is not difficult to adapt the decidability result to hedge acceptors. Moreover, we suppose that for any run r -even non accepting- on a tree, the cardinal of the set of nodes labelled by states of E is at most k . Indeed, it is easy to transform A to ensure this property by enriching states with a counter up to k . We show how to decompose a positive TAGED into an equivalent and computable finite set of configurations. Since testing emptiness of configurations is easily decidable, we get the decidability result for positive TAGED. Informally, configurations are (non-regular) tree acceptors which make explicit parent or ancestor relations between nodes for which equality tests are performed by A . These are DAG-like structures labelled by sets of states of A . A tree t is accepted by some configuration c if the unfolding of c can be embedded into a run r of A on t , such that labels of r belong to labels of c . By putting suitable rules on how sets of states occur as labels of c , we can enforce r to respect equality constraints.

Definition 6 (configurations). *A configuration c of A is a rooted directed acyclic graph such that: (i) every node carries a symbol from 2^Q , (ii) outgoing edges of a node are ordered, and (iii) for every equivalence class $[q]$ of $=_A$, there is at most one set $P \subseteq Q$ such that $[q] \cap P \neq \emptyset$ and P is a label of c .*

Nodes of c are denoted by $\text{nodes}(c)$. For every node $u \in \text{nodes}(c)$, we denote by $c|_u$ the subgraph of c induced by the nodes reachable from u in c , and by u_1, \dots, u_n the n successor nodes of u given in order. Note that it might be the case that $u_i = u_j$ for some $i, j \in \{1, \dots, n\}$. Finally, we always denote by $\text{lab}_c(u) \subseteq Q$ the label of node u in c . Fig. 3 illustrates a configuration whose nodes are labelled either by set of states $\{q, q'\}$, $\{s\}$, $\{p\}$ or $\{r\}$. Note that the second successor of the root is the node labelled $\{r\}$, while its first and third successor is the node labelled $\{q, q'\}$.

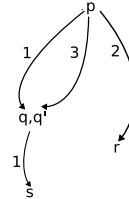


Fig. 3. A configuration (naturals represent the order on edges)

In order to define semantics of configurations, we first introduce some useful notions about contexts. For $n \geq 0$, we define n -ary contexts C s as usual, and the hole substitution with trees t_1, \dots, t_n is denoted by $C[t_1, \dots, t_n]$. Note that 0-ary contexts are just trees. See [10] for a formal definition. Given n states $p_1, \dots, p_n \in Q$ and an n -ary context C , we denote by $C[p_1, \dots, p_n]$ the tree over $\Lambda \cup Q$, where each p_i is

viewed as a constant symbol. We let A^* be the TAGED over alphabet $A \cup Q$ which is just A with additional rules $q(\epsilon) \rightarrow q$, for every $q \in Q$. We say that $C[p_1, \dots, p_n]$ evaluates to p , denoted $C[p_1, \dots, p_n] \rightarrow_A p$ if there is a run r of A^* on $C[p_1, \dots, p_n]$ such that $\text{roots}(r) = p$. For any set $S \subseteq Q$, we write $C[p_1, \dots, p_n] \rightarrow_{Q \setminus S} q$ if states from S does not occur in r , except at the root and at the leaves labelled p_1, \dots, p_n .

We now view configurations as a way to combine contexts to form trees t , with additional requirements which enforce existence of a run r of A on t . Condition (iii) of Definition 6 ensures r satisfies the equality constraints. This motivates the semantics of configurations. More formally, let c be a configuration of A . A mapping λ from $\text{nodes}(c)$ into contexts over A is an *interpretation* of c if for every node $u \in \text{nodes}(c)$, if u has exactly n successor nodes u_1, \dots, u_n in c , then $\lambda(u)$ is an n -ary context. Moreover, λ must satisfy the following: for every node $u \in \text{nodes}(c)$ and every nodes $u_1, \dots, u_n \in \text{nodes}(c)$, if u_1, \dots, u_n are the successor nodes of u then the following holds (called condition (P)):

$$\forall p \in \text{lab}_c(u), \exists p_1 \in \text{lab}_c(u_1) \dots \exists p_n \in \text{lab}_c(u_n), \lambda(u)[p_1, \dots, p_n] \rightarrow_{Q \setminus (E \cup D)} p$$

As A is positive, the set D is empty, but we keep this definition for uniformity reasons when dealing with disequalities. Every node $u \in \text{nodes}(c)$, together with the mapping λ , define a tree $t(u, \lambda)$ over A as follows: $t(u, \lambda) = \lambda(u)[t(u_1, \lambda), \dots, t(u_n, \lambda)]$, where $n \in \mathbb{N}$ and u_1, \dots, u_n are the successor nodes of u in c . Note that this definition is well-founded since c is a DAG. As we will see, condition (P) ensures the existence of a run of A on $t(u_0, \lambda)$, where u_0 is the root of c .

Definition 7 (tree language recognized by a configuration). *Let c be a configuration of A . The tree language recognized by c , denoted $L(A, c)$, is defined by the set of trees $t(u_0, \lambda)$, where u_0 is the root of c , and λ is an interpretation of c .*

Trees accepted by configuration of Fig. 3 are necessarily of form $C[C'[t], t', C'[t]]$, for some contexts C, C', C'' and trees t, t' . As already said, the constraints on the contexts and the configuration ensure the existence of a run on the trees of $L(A, c)$ which satisfies the equality constraints. In particular, we can prove the following:

Proposition 1. *Let c be a configuration of A such that $L(A, c)$ is nonempty. Let t be a tree of $L(A, c)$, and $u_0 \in \text{nodes}(c)$ be the root of c . For every $p \in \text{lab}_c(u_0)$, there is a run $r \in R_{A,p}(t)$ which respects the equality constraints.*

The converse holds too, and we can bound the size of configurations:

Proposition 2. *Let $t \in \mathbb{T}_A$ be a tree such that $t \in L(A)$. Then there is a configuration c of size at most $|Q| \cdot k^{|Q|}$ such that $t \in L(A, c)$.*

Sketch of proof We start from an accepting run r of A on t and define an equivalence relation on a subset of $\text{nodes}(t)$. Informally, two nodes u, v are equivalent if an equality test between $t|_u$ and $t|_v$ is performed in r . This is the case for instance when $\text{lab}_r(u) =_A \text{lab}_r(v)$. Each equivalence class will represent a node of c , to enforce equalities. \square

Hence, we can finitely represent the language recognized by A as a computable set of configurations of A , as stated below:

Corollary 1. *Let A be a k -bounded positive TAGED. We let $\mathcal{D}(A)$ be the set of configurations of A whose sizes are bounded by $|Q| \cdot k^{|Q|}$. The following holds:*

$$L(A) = \bigcup_{c \in \mathcal{D}(A)} L(A, c)$$

Moreover, we can decide emptiness of the language recognized by any configuration.

Lemma 1. *Given a configuration c , it is decidable to know whether $L(A, c) = \emptyset$ holds.*

Proof (Sketch). For all nodes u, u_1, \dots, u_n s.t. u_1, \dots, u_n are the successors of u , it suffices to test whether there is an n -ary context C s.t. for all state $p \in \text{lab}_c(u)$, there are $p_1 \in \text{lab}_c(u_1), \dots, p_n \in \text{lab}_c(u_n)$ s.t. $C[p_1, \dots, p_n] \rightarrow_{Q \setminus E} p$. We can represent the set of contexts C such that $C[p_1, \dots, p_n] \rightarrow_{Q \setminus E} p$ by a tree automaton $A_{(p_i)_i, p}$. Then, it suffices to test emptiness of $\bigcap_{p \in P} \bigcup_{(p_i)_i \in \prod_i \text{lab}_c(u_i)} L(A_{(p_i)_i, p})$, which is decidable, since regular languages are closed by Boolean operations.

As a corollary of Lemma 1 and Corollary 1, we get the following:

Proposition 3. *Emptiness of positive bounded TAGED is decidable.*

4.3 Adding disequalities to positive bounded TAGED

In the previous section, we have shown that emptiness of positive bounded TAGED is decidable. In this section, we extend this result to full bounded TAGED. A always denotes a k -bounded TAGED. The definition of configurations of A remains unchanged, and the set $\mathcal{D}(A)$ still denotes the set of configurations of A whose size is bounded by $|Q| \cdot k^{|Q|}$. We have the following inclusion: $L(A) \subseteq \bigcup_{c \in \mathcal{D}(A)} L(A, c)$, but the other one does not hold in general, since configurations do not require disequality constraints to be satisfied. We show how an additional test on configurations c allows us to decide whether $L(A) \cap L(A, c)$ is empty, which will be sufficient to decide whether $L(A)$ is empty. Informally, let c be a configuration and λ be an interpretation of c . We say that λ satisfies the disequalities of c if for all nodes $u, v \in \text{nodes}(c)$, if there are $p \in \text{lab}_c(u)$ and $q \in \text{lab}_c(v)$ such that $p \neq_A q$, then $t(u, \lambda) \neq t(v, \lambda)$.

We now relate the problem of finding such an interpretation to context disunification. For all nodes $u \in \text{nodes}(c)$, we let $\text{cxt}_c(u)$ be the set of contexts satisfying condition (P) of the definition of interpretation. We define the notion of *partial interpretation* β of c , as a mapping from $\text{nodes}(c)$ into contexts, such that it maps every node u such that $\text{cxt}_c(u)$ is finite into a context of $\text{cxt}_c(u)$, and every other node v to a context $@_v(\bullet, \dots, \bullet)$ with n holes (if v has n successors), where $@_v$ is a fresh symbol such that $@_v \notin \Lambda$. Note that trees $t(u, \beta)$ are trees over alphabet $\Lambda \cup \{@_v \mid v \in \text{nodes}(c)\}$. We can show the following, by using context disunification (symbols $@_v$ are viewed as context variables):

Lemma 2. *Let A be a bounded TAGED. We have $L(A) \neq \emptyset$ iff there exist a configuration $c \in \mathcal{D}(A)$ and a partial interpretation β of c such that β satisfies the disequality constraints of c . Moreover, it is decidable to know whether such an interpretation β exists.*

As a corollary, by combining Lemma 2 and Lemma 1, we get the proof of Theorem 5.

5 From TQL to Automata

In this section, ϕ denotes a recursion-closed and guarded TQL formula over tree variables X_1, \dots, X_n . We sketch the construction of a TAGED A_ϕ such that for all hedges $h \in \mathbb{H}_A$, we have $h \in L(A_\phi)$ iff there exists a valuation $\rho : \text{var}(\phi) \rightarrow \mathbb{T}_A$ such that $h \in \llbracket \phi \rrbracket_\rho$. Moreover, we prove A_ϕ to be equivalent to a computable bounded TAGED whenever ϕ is bounded.

In a first step, we transform ϕ into an equivalent system of fixpoint equations, and then sketch the construction of A_ϕ starting from this system. This construction extends the construction of [4]. This extension is non-trivial, since it manages tree variables, which induce non-determinism in the produced tree automaton. Moreover, even for sentences, this construction is different, since trees are ordered.

System of equations We define dual connectives for parallel composition and Kleene star. We let $\phi_1 \parallel \phi_2$ as a shortcut for $\neg(\neg\phi_1 \mid \neg\phi_2)$, ϕ_1^\diamond for $\neg(\neg\phi_1)^*$ and \overline{X} for $\neg X \wedge \Lambda[\top]$. A *system of fixpoint equations* Σ is a sequence of equations $\xi_1 = \text{rhs}_1, \dots, \xi_n = \text{rhs}_n$ where every rhs_i has one of the following form:

$$0 \quad \overline{0} \quad \xi \vee \xi' \quad \xi \wedge \xi' \quad \alpha[\xi] \quad \xi \mid \xi' \quad \xi \parallel \xi' \quad X \quad \overline{X} \quad \xi^* \quad \xi^\diamond$$

The last fixpoint variable, ξ_n , is denoted by $\text{last}(\Sigma)$. The set of tree variables occurring in Σ is denoted by $\text{var}(\Sigma)$. Systems of equations are interpreted over the complete lattice $(2^{\mathbb{H}_A}, \cup, \cap)$, modulo an assignment $\rho : \text{var}(\Sigma) \rightarrow \mathbb{T}_A$. We consider the following monotonic operations over $2^{\mathbb{H}_A}$, modulo ρ : 0 is interpreted as $\{0^{\mathbb{H}_A}\}$, $\overline{0}$ as $\overline{\{0^{\mathbb{H}_A}\}}$ (the overline denotes the complement in \mathbb{H}_A), \vee by \cup , \wedge by \cap , $\alpha[\cdot]$ as the unary operator which maps any set of hedges $H \subseteq \mathbb{H}_A$ into $\{a[h] \mid a \in \alpha, h \in H\}$, \mid as the binary operator which maps two sets of hedges H, H' into $H \mid H' = \{h \mid h' \mid h \in H, h' \in H'\}$, its dual \parallel maps H, H' into $H \parallel H' = \overline{H \mid H'}$. The Kleene star * and its dual $^\diamond$ are interpreted similarly. Finally, X is interpreted by $\rho(X)$ and \overline{X} by $\mathbb{T}_A \setminus \{\rho(X)\}$.

The solution of Σ over $(2^{\mathbb{H}_A}, \cup, \cap)$ modulo ρ is a mapping from fixpoint variables of Σ into $2^{\mathbb{H}_A}$, and is denoted by $\text{Sol}_{\mathbb{H}}(\Sigma, \rho)$. We can push down the negations to the leaves of ϕ , using the dual connectives, and introduce fixpoint variables for every position in ϕ , which allow to construct a system of equations S_ϕ such that $\text{var}(S_\phi) = \text{var}(\phi)$ and the following holds:

Lemma 3. *For all valuations $\rho : \text{var}(S_\phi) \rightarrow \mathbb{T}_A$, we have $\llbracket \phi \rrbracket_\rho = \text{Sol}_{\mathbb{H}}(S_\phi, \rho)(\text{last}(S_\phi))$*

E.g., the equation system associated with the formula $\mu\xi.(a[\xi] \vee (\mu\xi'.(b[\xi'] \vee X)))$ is $\{\xi' = \xi_2 \vee \xi_3; \quad \xi_2 = b[\xi']; \quad \xi_1 = a[\xi]; \quad \xi_3 = X; \quad \xi = \xi_1 \vee \xi'\}$.

Ideas of automaton construction for sentences In this paragraph, we assume that ϕ is a sentence. Checking whether a hedge h is a solution of S_ϕ is similar to a run of some hedge automaton on h . Let us consider the system $S = \{\xi = \xi_1 \vee \xi_2; \xi_1 = a[\xi]; \xi_2 = 0\}$, where the last variable is ξ . Solutions of S are chains labelled by a . To check whether hedge $a(0)$ is a solution of ξ , first verify that $a(0)$ is a solution of ξ_1 or a solution of ξ_2 . One can easily see that $a(0)$ is not a solution of ξ_2 . It remains to verify whether $a(0)$ is a solution of $\xi_1 = a[\xi]$, by verifying that 0 is a solution of ξ , etc... This can be done by an automaton with transition rules $a(\epsilon + q_{a[\xi]}) \rightarrow q_{a[\xi]}$, where $q_{a[\xi]}$ is

a final state. We define the set of (final) states by $Q = F = \{q_{a[\xi]}\}$. Let us interpret S over 2^{Q^*} , where Q^* is the set of words over Q . We interpret \vee by \cup , 0 by $\{\epsilon\}$, and $a[\xi]$ by $\{q_{a[\xi]}\}$. Solutions of ξ are denoted by $s(\xi)$ (and similarly for other variables). Hence we get $s(\xi_2) = \{\epsilon\}$, $s(\xi_1) = \{q_{a[\xi]}\}$, and $s(\xi) = \{\epsilon, q_{a[\xi]}\}$. Which trees evaluate to $q_{a[\xi]}$? Trees of the form $a(h)$ where h evaluates to some word of states from $s(\xi)$. Hence, we can define transition $a(s(\xi)) \rightarrow q_{a[\xi]}$.

Things get more complicated when adding intersection. For instance, consider the system $S' = \{\xi_1 = a[\xi]; \xi_2 = a[\xi']; \xi' = 0; \xi = \xi_1 \wedge \xi_2\}$. If we interpret this system as before, with states $q_{a[\xi]}$ and $q_{a[\xi']}$, we would get $s(\xi) = \emptyset$. Hence, states should carry enough information to know if the current tree is a solution of $a[\xi]$, $a[\xi']$, or both. In the construction we provide, every state is a tuple of atoms of the form $\alpha[\xi]$, $\bar{\alpha}[\top]$, or $\alpha[-\xi]$, for every right-hand side of the form $\alpha[\xi]$ occurring in the system. If some tree t evaluates to a tuple which has a component equal to $\alpha[\xi]$, it means that t is of the form $a(h)$, where $a \in \alpha$ and h is solution of ξ . If the component is $\bar{\alpha}[\top]$ or $\alpha[-\xi]$, it means that $a(h)$ is not a solution of $\alpha[\xi]$, because, in the first case, $a \notin \alpha$, and in the second one, $a \in \alpha$, but h is not a solution of ξ . Knowing this complete information, i.e. which right-hand sides of the form $\alpha[\xi]$ are satisfied or not by the current tree, we are able to construct exactly one rule per state, by solving the system on sets of words of states, with suitable interpretations. As the formula is guarded, solutions of the system are regular state word languages. We then get a deterministic hedge automaton whose accepted trees are the solutions of the system.

Adding tree variables When adding variables, we cannot keep the automaton deterministic, since subtrees will be captured non-deterministically. For instance, consider the system $S = \{\xi'' = X; \xi' = \xi''|\xi''; \xi_X = a[\xi']; \xi_2 = 0; \xi_3 = \xi_1^*; \xi_1 = \Lambda[\xi_3]; \xi_\top = \xi_1 \vee \xi_2; \xi = \xi_\top \wedge \xi_X\}$, where the last variable is ξ . Given a valuation $\rho : \text{var}(S) \rightarrow \mathbb{T}_A$, the system S has a unique solution, modulo ρ , which is $a(\rho(X)|\rho(X))$. A TAGED accepting the solutions of S is the TAGED of Example 1 of Section 4. It is non-deterministic, since it has to choose to go in a state q_X which will enforce the TAGED to test whether the two sons of the root are equal.

As tree variables induce a kind of non-determinism, we emphasize two kinds of recursion variables: deterministic recursion variables, for which the problem of checking whether a given hedge is a solution does not involve tree variables, such as ξ_\top , ξ_1 , ξ_2 and ξ_3 in our example; and non-deterministic one: ξ, ξ_X, ξ' and ξ'' .

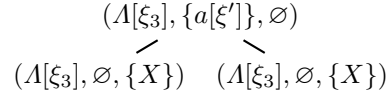


Fig. 4. Run of A_ϕ on $a(a(0)a(0))$

States of the automaton we construct have three components. The first component is induced by deterministic recursion variables and simulate a classical hedge automaton, as for the case of sentences. The second component is induced by non-deterministic recursion variables, and collects atoms of the form $\alpha[\xi]$, where ξ is non-deterministic, for which the current tree is the solution. In other words, it guesses the positions in the tree under which capture variables occur. Third components are sets of variables X or \bar{X} , meaning that equality or disequality tests are performed on the current tree.

Transition rules are obtained by suitable interpretation of the system over words of states. Finally, two states are equivalent for the automaton if their third component

shares a tree variable. Disequalities are defined similarly. For instance, an accepting run of the automaton for S , on the tree $a(a(0)a(0))$, is represented in Fig. 4. The state $(A[\xi_3], \emptyset, \{X\})$ is equivalent to itself.

Theorem 6. *Let ϕ be a guarded TQL formula. There is a computable TAGED A_ϕ such that for all hedges h , we have $h \in L(A_\phi)$ iff there exists a valuation $\rho : \text{var}(\phi) \rightarrow \mathbb{T}_A$ such that $h \in \llbracket \phi \rrbracket_\rho$.*

Moreover, if ϕ is bounded, the TAGED A_ϕ is equivalent to the bounded TAGED (A_ϕ, B) , for some computable bound $B \in \mathbb{N}$.

To compute the bound B , we interpret the system S_ϕ on naturals, with suitable interpretations (for instance, X is interpreted by 1, \wedge by $+$, and \vee by \max).

6 Extending MSO with Tree Isomorphism Tests

In this section, we propose an extension of MSO for unranked trees with isomorphism tests between trees.

Let \sim be a binary predicate s.t. for a structure \mathcal{S}^h associated with a hedge h and a mapping ρ from $\{x, y\}$ to nodes of h , $\mathcal{S}^h \models_\rho x \sim y$ holds if the two subtrees rooted at respectively $\rho(x)$ and $\rho(y)$ in h are isomorphic. We consider sentences of the form

$$Q_1 x_1 Q_2 x_2 \dots Q_n x_n \psi$$

where $Q_i \in \{\exists, \forall\}$ and ψ is an MSO formula extended with atoms $x_i \sim x_j$ ($1 \leq i, j \leq n$). We call MSO_\sim this logic. We will also consider the fragment MSO_\sim^\exists for which formulas satisfy $Q_1 = Q_2 = \dots = Q_n = \exists$. Remark that ψ can again contain quantifiers. Hence, since tree isomorphism cannot be expressed in MSO [10], MSO_\sim and MSO_\sim^\exists are strictly more expressive than MSO . By reduction of the Post correspondence problem, we can prove that:

Theorem 7. *Satisfiability for MSO_\sim is undecidable.*

However, MSO_\sim^\exists and bounded TAGED are equally expressive: for any formula φ in MSO_\sim^\exists , one can compute a bounded TAGED, whose size is non-elementary in the size of φ , accepting the models of φ . The converse holds too. As a consequence of decidability of emptiness for bounded TAGED, we have:

Theorem 8. *Satisfiability is decidable for MSO_\sim^\exists .*

7 Conclusion

In this paper, we have considered the spatial logic TQL with tree variables. We have proved that for the guarded fragment when variables appear in a bounded way then the satisfiability problem is decidable. To do so, we have introduced a new class of tree automata, called TAGED, permitting to test global equalities and disequalities on the accepted trees. Finally, we have used TAGED to prove decidability for an extension of MSO with isomorphism tests interpreted over unranked trees.

We speculate that the boundedness condition is not required for the decidability of emptiness of TAGED, as pumping technics dealing with constraints may be applicable. However, it is non-trivial, since TAGED are not determinizable in general. This would imply that the full guarded TQL with trees variables is decidable. Another extension

would be to consider hedge variables. This problem seems to be non trivial as the satisfiability problem for such formulas could encode word equations.

We emphasized a correspondence between MSO_{\sim}^{\exists} and bounded TAGED. It would be interesting to exhibit a fragment of MSO_{\sim} equivalent to full TAGED.

The TQL system also includes a transformation language; we aim at using TAGED automata to type these transformations and more generally, tree transducers.

References

1. V. Benzaken, G. Castagna, and A. Frisch. CDuce: an XML-centric general-purpose language. In *8th ACM International Conf. on Functional Programming*, pages 51–63, 2003.
2. B. Bogaert and S. Tison. Equality and disequality constraints on direct subterms in tree automata. In *9th Annual Symposium on Theoretical Aspects of Computer Science*, volume 577 of *LNCS*, pages 161–171, 1992.
3. M. Bojanczyk, C. David, A. Muscholl, T. Schwentick, and L. Segoufin. Two-variable logic on data trees and xml reasoning. In *ACM 25th Symp. on Principles of database systems*, pages 10–19, 2006.
4. I. Boneva, JM. Talbot, and S. Tison. Expressiveness of a spatial logic for trees. In *20th IEEE Symposium on Logic in Computer Science*, pages 280–289, 2005.
5. A. Brüggemann-Klein, M. Murata, and D. Wood. Regular tree languages over non-ranked alphabets. unpublished manuscript, 1998.
6. L. Cardelli, P. Gardner, and G. Ghelli. A spatial logic for querying graphs. In *29th International Colloquium on Automata, Languages and Programming*, volume 2380 of *LNCS*, pages 597–610. Springer, 2002.
7. L. Cardelli and G. Ghelli. TQL: A Query Language for Semistructured Data Based on the Ambient Logic. *Mathematical Structures in Computer Science*, 14:285–327, 2004.
8. W. Charatonik and JM. Talbot. The Decidability of Model Checking Mobile Ambients. In *15th Annual Conference of the European Association for Computer Science Logic*, volume 2142 of *LNCS*, pages 339–354. Springer, 2001.
9. H. Comon and V. Cortier. Tree automata with one memory, set constraints and cryptographic protocols. *TCS*, 331(1):143–214, 2005.
10. H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications, 1997. release October, 1rst 2002.
11. M. Dauchet, A.-C. Caron, and J.-L. Coquidé. Reduction properties and automata with constraints. 20:215–233, 1995.
12. A. Dawar, P. Gardner, and G. Ghelli. Expressiveness and complexity of graph logic. In *Information and Computation*, number 205, pages 263–310. 2007.
13. H. Hosoya and B. C. Pierce. XDuce: A statically typed xml processing language. *ACM Trans. Internet Techn.*, 3(2):117–148, 2003.
14. F. Jacquemard, M. Rusinowitch, and L. Vigneron. Tree automata with equality constraints modulo equational theories. Research Report LSV-06-07, LSV, ENS Cachan, France, 2006.
15. W. Karianto and C. Löding. Unranked tree automata with sibling equalities and disequalities. In *34th International Colloquium on Automata, Languages and Programming*, 2007.
16. J. Mongy. *Transformation de noyaux reconnaissables d'arbres. Forêts RATEG*. PhD thesis, Université de Lille, 1981.
17. M. Murata. Hedge automata: A formal model for xml schemata. Technical report, Fuji Xerox Information Systems, 1999.
18. Frank Neven, Thomas Schwentick, and Victor Vianu. Towards regular languages over infinite alphabets. In *MFCS*, pages 560–572. SV, 2001.
19. J. C. Reynolds. Separation logic: A logic for shared mutable data structures. In *17th IEEE Symp. on Logic in Computer Science*, pages 55–74. IEEE, 2002.

A Deciding emptiness of bounded TAGED

A.1 Positive bounded TAGED

Proof of Proposition 1 Since $t \in L(A, c)$, there exists an interpretation λ of c such that $t = t(u_0, \lambda)$.

Let u_0, u_1, \dots, u_n a topological order of the nodes of c . For any i , let $t_i = t(u_i, \lambda)$. We will prove that for any i , for any state q_i belonging to $\text{lab}_c(u_i)$, there is a run on t_i whose root is q_i , that satisfies the strong following equality constraints: if a node is labelled by q in E , then q belongs to one and only one $\text{lab}_c(u_k)$ and the corresponding subterm is isomorphic to t_k .

This can be proven by induction. If $i = n$, then $t_n = \lambda(u_n)$ and the property is directly obtained. Now let us suppose the property is true if $i > p$. Then $t_p = \lambda(u_p)(t_{i_1}, \dots, t_{i_k})$ where the t_{i_j} satisfy the required property by induction hypothesis. Using the condition (P) of interpretations of configurations, we get easily the required property for t_p . Now, as $\text{lab}_c(u_0)$ contains a final state, we get the proposition.

Proof of Proposition 2 Since $t \in L(A)$, there exists a successful run $r \in R_A^{acc}(t)$. We associate to (t, r) a configuration c , such that $|c| \leq |Q| \cdot k^{|Q|}$ and $t \in L(A, c)$. Informally, we define an equivalence relation \sim on nodes of t , two nodes being equivalent if they are labelled, in r , by two states in relation by $=_A$ (hence, the two induced subtrees of t are equal), or they can be reached by two isomorphic paths, whose respective node starting points are equivalent. In particular, this definition implies that if nodes u and v are equivalent, then $t|_u = t|_v$ holds, and, roughly speaking, $t|_u$ and $t|_v$ have been tested to be equal by the automaton, or are embedded in two equal trees which have been tested to be equal by the automaton.

Equivalence classes are intended to represent the nodes of the configuration. By defining a suitable order on equivalence classes, we can define the configuration. A full example is given at the end of the proof.

Node equivalence The node equivalence relation is defined on a subset of $\text{nodes}(t)$. Given two nodes $u, v \in \text{nodes}(t)$, we define the relation \sim as the least relation satisfying the following:

- $u_0 \sim u_0$, where u_0 is the root of t ;
- for all nodes $u, v \in \text{nodes}(t)$, if $\text{lab}_r(u) =_A \text{lab}_r(v)$, then $u \sim v$;
- for all nodes $u, v, u', v' \in \text{nodes}(t)$ such that $u' < u$ and $v' < v$, if $u' \sim v'$ and there exists a node $w \in \text{nodes}(t)$ such that $u \sim w$, and the downward paths from respectively u' to u and v' to v are isomorphic, then $u \sim v$;
- for all nodes $u, v, w \in \text{nodes}(t)$, if $u \sim w$ and $w \sim v$, then $u \sim v$.

We denote by $\text{nodes}(t)|_{\sim}$ the set of nodes u such that $\exists v \in \text{nodes}(t), u \sim v$. We can show that \sim is symmetric by induction on pairs of nodes of r (using the lexicographic order induced by \leq). It is transitive by definition, hence reflexive on $\text{nodes}(t)|_{\sim}$. Thus, \sim is an equivalence relation on $\text{nodes}(t)|_{\sim}$.

Fig. 5 illustrates this relation. It represents a tree t with two isomorphic paths: u, u', v, v' are nodes of t , while p, p', q, q' are states of A , which represent the labels

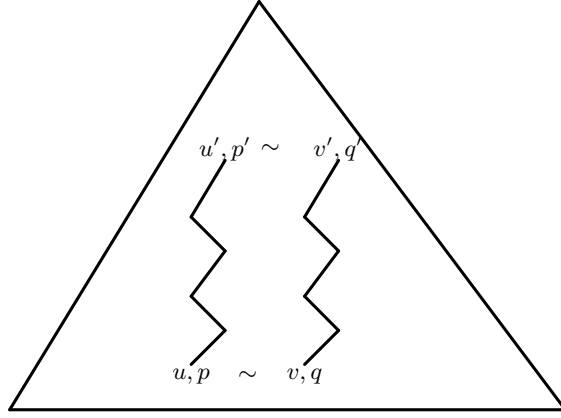


Fig. 5. Node equivalence

at nodes u, u', v, v' respectively in a run of A on t . Suppose that $p' =_A q'$, and $p \in E$. Hence, $u' \sim v'$, and, since the path from u' to u in t is isomorphic to the path from v' to v in t , we get $u \sim v$ (the labels in t of the nodes on the respective paths must be the same, by definition of path isomorphism).

Remark that the number of equivalence classes is bounded by some $b \leq |Q|$. We denote by \bar{u} the equivalence class of any node $u \in \text{nodes}(t)|_{\sim}$, and by $\text{states}(\bar{u})$ the set $\{q \in Q \mid \exists v \in \bar{u}, \text{lab}_r(v) = q\}$. Remark that for every equivalence class \bar{u} , for all states $q \in \text{states}(\bar{u})$, $R_{A,q}(t|_u)$ is non-empty.

Frontier We let \leq_{do} be the document order on nodes, i.e. the order induced by a depth-first traversal of the tree. Given a node $u \in \text{nodes}(t)|_{\sim}$, we let $\text{frontier}(u)$ be the sequence defined as the maximal subset (ordered by document order) of $\text{nodes}(t)|_{\sim}$, let us called (u_1, \dots, u_n) , such that every u_i is minimal for \leq and below u . More formally, (u_1, \dots, u_n) is defined by

- for all $i \in \{1, \dots, n\}$, $u < u_i$;
- for all $i \in \{1, \dots, n\}$, for all $v \in \text{nodes}(t)|_{\sim}$, if $u < v \leq u_i$ then $v = u_i$ (called property (P));
- for all $i \in \{1, \dots, n-1\}$, $u_i \leq_{do} u_{i+1}$, for all $v \in \text{nodes}(t)|_{\sim}$, if $u_i <_{do} v' <_{do} u_{i+1}$, then $u_i \leq v$.

Configuration We define configuration c of A by:

- nodes of c are the equivalence classes \bar{u} , for all $u \in \text{nodes}(t)|_{\sim}$;
- for all $u \in \text{nodes}(t)|_{\sim}$, $\text{lab}_c(\bar{u})$ is defined by $\text{states}(\bar{u})$;
- for all $u \in \text{nodes}(t)|_{\sim}$, let us denote $\text{frontier}(u) = (u_1, \dots, u_n)$ the frontier of u . The n successors (given in order) of node \bar{u} in c are defined by $\bar{u}_1, \dots, \bar{u}_n$.

Remark that the definition of the successors of any node \bar{u} in c are independent of the choice of the class representatives. In other words, let $u, u' \in \text{nodes}(t)$ be two nodes such that $u \sim u'$.

Let (u_1, \dots, u_n) be equal to $\text{frontier}(u)$. We denote by u'_1 (resp. u'_2, \dots, u'_n) the node of t reachable from u' by the downward path isomorphic to the path from u to u_1 (resp. u_2, \dots, u_n). By definition of \sim , we have $u_i \sim u'_i$, for every $i \in \{1, \dots, n\}$. Similarly, it is not difficult to prove that (u'_1, \dots, u'_n) is equal to $\text{frontier}(u')$.

Order on equivalence classes In order to complete the proof of Proposition 2, we define a strict order \prec_c on equivalence classes as the transitive but not reflexive closure of the relation (\mathcal{R}) defined by:

$$\bar{u} (\mathcal{R}) \bar{v} \text{ iff } \exists u' \in \bar{u}, \exists v' \in \bar{v}, u' < v'$$

We can verify that \prec_c is a strict order. We show antisymmetry. Let $i, j \in \{1, \dots, m\}$. Suppose that $\bar{u} \prec_c \bar{v}$ and $\bar{v} \prec_c \bar{u}$. Then there exists nodes $u_1, u_2 \in \bar{u}$, and $v_1, v_2 \in \bar{v}$ such that $u_1 \leq v_1$ and $v_2 \leq u_2$. By definition of \sim , $t|_{u_1} = t|_{u_2}$, and $t|_{v_1} = t|_{v_2}$, so that $t|_{v_1} = t|_{u_1}$ and $t|_{u_2} = t|_{v_2}$. Hence $v_1 = u_1$ and $v_2 = u_2$, and $\bar{u} = \bar{u}_1 = \bar{v}_1 = \bar{v}$.

Proof of correctness We now prove the following claims, which will conclude the proof of the whole Proposition:

Claim 1 Let $u, v \in \text{nodes}(t)|_{\sim}$. If \bar{v} is a successor of \bar{u} in c , then $\bar{u} \prec_c \bar{v}$.

Claim 2 c is a configuration of A .

Claim 3 It holds that $t \in L(A, c)$.

Claim 4 Let $u \in \text{nodes}(t)|_{\sim}$ such that u is not the root of t . Let $\downarrow(\bar{u})$ be the set of equivalence classes less than \bar{u} , i.e. $\downarrow(\bar{u}) = \{\bar{v} \mid \bar{v} \prec_c \bar{u}\}$. The cardinality of \bar{u} , denoted $|\bar{u}|$, is bounded by $k \cdot \max_{\bar{v} \in \downarrow(\bar{u})} |\bar{v}|$.

Claim 5 The size of c is bounded by $|Q|k^{|Q|}$.

Proof of Claim 1 Let $u, v \in \text{nodes}(t)|_{\sim}$. Suppose that \bar{v} is a successor of \bar{u} in c . It means that there exist $u' \in \bar{u}$ and $v' \in \bar{v}$ such that $v' \in \text{frontier}(u')$. Hence, by definition of the frontier, $u' < v'$, and we get $\bar{u} \prec_c \bar{v}$. \square

Proof of Claim 2 By Claim 1, we get the proof of acyclicity of c , otherwise it would contradict the fact that \prec_c is an order on the nodes of c . It is not difficult to see that c is uniquely rooted by \bar{u}_0 , where u_0 is the root of t , and that Conditions (i) and (ii) of the definition of configurations are satisfied (Definition 6).

For condition (iii), suppose that there exist two different nodes \bar{u} and \bar{v} of c (i.e. $u \not\sim v$), and an equivalence class $[q]$ of $=_A$ such that $\text{lab}_c(\bar{u}) \cap [q] \neq \emptyset$ and $\text{lab}_c(\bar{v}) \cap [q] \neq \emptyset$. It means that there exist two nodes $u' \in \bar{u}$ and $v' \in \bar{v}$, and two states $q', q'' \in [q]$, such that $\text{lab}_r(u') = q'$, and $\text{lab}_r(v') = q''$. Since $q' =_A q''$, by definition of \sim , we get $u' \sim v'$, hence $u \sim v$, which is a contradiction. \square

Proof of Claim 3 We define a mapping λ from nodes of c into contexts, and prove, in a first step, that λ is an interpretation of c , and in second step, that $t = t(\lambda, \bar{u}_0)$, where u_0 is the root of t , which will conclude the proof, by definition of $L(A, c)$.

For all nodes $u \in \text{nodes}(t)|_{\sim}$, we let (u_1, \dots, u_n) be equal to $\text{frontier}(u)$. The mapping λ is defined by:

$$\lambda(\bar{u}) = C \text{ where } C \text{ is the context s.t. } t|_u = C[t|_{u_1}, \dots, t|_{u_n}]$$

Since any two equivalent nodes have the same frontier, modulo isomorphism, this definition is independent of the choice of the class representative: if (u_1, \dots, u_n) (resp. (u'_1, \dots, u'_n)) is the frontier of u (resp. of u'), then for every $i \in \{1, \dots, n\}$, the path from u to u_i is isomorphic to the path from u' to u'_i . Since $t|_u = t|_{u'}$, we get $\lambda(\bar{u}) = \lambda(\bar{u}')$.

Now, let $u \in \text{nodes}(t)|_{\sim}$ be a node of t , and (u_1, \dots, u_n) its frontier. Let $p \in \text{lab}_c(\bar{u})$, i.e. $p \in \text{states}(\bar{u})$. We have to show that there exist $p_1 \in \text{lab}_c(\bar{u}_1), \dots, p_n \in \text{lab}_c(\bar{u}_n)$ such that $C[p_1, \dots, p_n] \rightarrow_{Q \setminus E} p$. Since $p \in \text{states}(\bar{u})$, there exist a node $u' \in \text{nodes}(t)$ such that $u \sim u'$ and $\text{lab}_r(u') = p$. We let (u'_1, \dots, u'_n) be the frontier of u' , and for every $i \in \{1, \dots, n\}$, we let p_i equal to $\text{lab}_r(u'_i)$. By definition of the frontier, there is no node $v \in \text{nodes}(t)$, except possibly u' or u'_1, \dots, u'_n which are labelled by a state of E in the run r . Otherwise, we would have $v \in \text{nodes}(t)|_{\sim}$ and $u < v < u'_i$, for every $i \in \{1, \dots, n\}$. Hence, we have $C[p_1, \dots, p_n] \rightarrow_{Q \setminus E} p$. Now, we show that for every $i \in \{1, \dots, n\}$, since $u'_i \in \bar{u}_i$, it is clear that p_i belongs to $\text{states}(\bar{u}_i)$, i.e. $\text{lab}_c(\bar{u}_i)$. Therefore, we can conclude that λ is an interpretation of c .

In a second step, we prove that $t \in L(A, c)$, or equivalently, $t = t(\bar{u}_0, \lambda)$. By induction on equivalence classes, we show that for all nodes $u \in \text{nodes}(t)|_{\sim}$, $t|_u = t(\bar{u}, \lambda)$.

By definition of λ , it is obvious when \bar{u} is minimal for \prec_c , since the frontier of u is empty, so that $\lambda(\bar{u}) = t|_u$.

Suppose now that (u_1, \dots, u_n) is the frontier of u , where $n > 0$. By induction hypothesis (since for every $i \in \{1, \dots, n\}$, $u \prec_c u_i$), we already know that for every $i \in \{1, \dots, n\}$, $t|_{u_i} = t(\bar{u}_i, \lambda)$. Since $t(\bar{u}, \lambda) = \lambda(\bar{u})[t(\bar{u}_1, \lambda), \dots, t(\bar{u}_n, \lambda)]$, we get $t(\bar{u}, \lambda) = \lambda(\bar{u})[t|_{u_1}, \dots, t|_{u_n}]$, hence, by definition of λ , we have $t|_u = t(\bar{u}, \lambda)$. \square

Proof of Claim 4 The proof is divided in several steps: first, we associate with every node $v \in \bar{u}$ such that $\text{lab}_r(v) \in E$ a set of nodes $\mathcal{N}(v) \subseteq \bar{u}$, for which we can bound the cardinality. Then, we prove that \bar{u} consists of the union of all sets $\mathcal{N}(v)$, for every node $v \in \bar{u}$ such that $\text{lab}_r(v) \in E$. For this, we introduce a relation $\rightarrow \in \bar{u}^2$ which we prove to be transitive. Finally, since the set of nodes $v \in \bar{u}$ such that $\text{lab}_r(v) \in E$ is bounded by k , and we can bound every sets $\mathcal{N}(v)$, we can bound the cardinality of \bar{u} .

We let $v \in \bar{u}$ such that $\text{lab}_r(v) \in E$ (there is at least one such node in \bar{u} , by definition of \sim , and since u is not the root of t). For any node pairs $(v_1, v_2), (v'_1, v'_2) \in \text{nodes}(t)^2$ such that $v_1 \leq v_2$ and $v'_1 \leq v'_2$, we denote by $\text{path}_\downarrow(v_1, v_2)$ (resp. $\text{path}_\downarrow(v'_1, v'_2)$) the downward path from v_1 to v_2 (resp. from v'_1 to v'_2), and write $\text{path}_\downarrow(v_1, v_2) \equiv \text{path}_\downarrow(v'_1, v'_2)$ if the two paths are isomorphic.

We let w be the maximal element of the set $\{v' \mid v' \in \text{nodes}(t)|_{\sim} \wedge v' < v\}$. We define the set of nodes $\mathcal{N}(v)$ by $v' \in \mathcal{N}(v)$ if there exists $w' \in \bar{w}$ such that we

have $\text{path}_\perp(w, v) \equiv \text{path}_\perp(w', v')$. Fig. 6 illustrates $\mathcal{N}(v)$. Remark that we have $|\mathcal{N}(v)| = |\bar{w}|$. This is because all subtrees $t|_{w'}$ are equal, for all nodes $w' \in \bar{w}$.

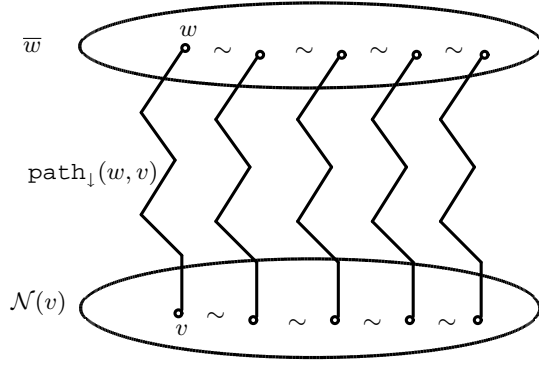


Fig. 6. The set of nodes $\mathcal{N}(v)$

We now prove that $\bigcup_{v \in \bar{u} \text{ and } \text{lab}_r(v) \in E} \mathcal{N}(v) = \bar{u}$. The forth inclusion is obvious. For the converse, we define a relation \rightarrow on nodes v', v'' of \bar{u} by $v' \rightarrow v''$ if $\text{lab}_r(v') \notin E$ and there exists two nodes w', w'' such that $w' \sim w''$, $w' < v'$, $w'' < v''$ and $\text{path}_\perp(w', v') \equiv \text{path}_\perp(w'', v'')$. We denote by \rightarrow^* its transitive closure. For every nodes $v' \in \bar{u}$, if $\text{lab}_r(v') \notin E$, then v' has a successor by \rightarrow . We also can prove that there exists a node $v'' \in \bar{u}$ such that $\text{lab}_r(v'') \in E$ and $v' \rightarrow^* v''$.

Now, we show that this relation is transitive. Let v, v', v'' be nodes of \bar{u} such that $v \rightarrow v'$ and $v' \rightarrow v''$. It means that there exists nodes w, w'_1, w'_2, w'' such that the following holds (illustrated by Fig. 7):

- $w \sim w'_1, w'_2 \sim w''$;
- $w < v, w'_1 < v', w'_2 < v', w'' < v''$;
- $\text{path}_\perp(w, v) \equiv \text{path}_\perp(w'_1, v')$;
- $\text{path}_\perp(w'_2, v') \equiv \text{path}_\perp(w'', v'')$.

We have either $w'_1 \leq w'_2$ or $w'_2 \leq w'_1$. Suppose that $w'_1 \leq w'_2$ (the other case is symmetric). Hence, $\text{path}_\perp(w'_2, v')$ is somehow a suffix of $\text{path}_\perp(w'_1, v')$. Since $\text{path}_\perp(w'_1, v') \equiv \text{path}_\perp(w, v)$, there exists a node $w_0 \in \text{nodes}(t)$ such that $w_0 \leq v$ and $\text{path}_\perp(w_0, v) \equiv \text{path}_\perp(w'_2, v')$. As a consequence, we get $\text{path}_\perp(w, w_0) \equiv \text{path}_\perp(w'_1, w'_2)$. Hence, since $w'_2 \sim w'_2$, we have $w_0 \sim w'_2$, and by transitivity, we get $w_0 \sim w''$. Again, since $\text{path}_\perp(w_0, v) \equiv \text{path}_\perp(w'', v'')$, we can conclude that $v \rightarrow v''$.

Finally, let v' be an element of \bar{u} such that $\text{lab}_r(v') \notin E$. There exists an element $v \in \bar{u}$ such that $\text{lab}_r(v) \in E$ and $v' \rightarrow^* v$. Since \rightarrow is transitive, we also have $v' \rightarrow v$. Hence, there exists $w', w \in \text{nodes}(t)$ such that $w' \sim w, w' > v'$,

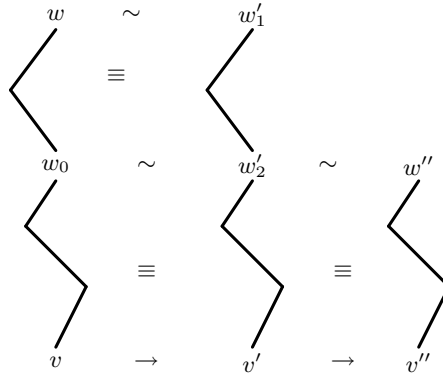


Fig. 7. Transitivity of \rightarrow

$w > v$ and $\text{path}_\perp(w', v') \equiv \text{path}_\perp(w, v)$. We let w_0 be the greatest node (for \leq) on the path from w to v such that $w_0 \in \text{nodes}(t)|_{\sim}$. We let w'_0 its corresponding node on $\text{path}_\perp(w', v')$. Since we have $\text{path}_\perp(w', w'_0) \equiv \text{path}_\perp(w, w_0)$, and $w_0 \in \text{nodes}(t)|_{\sim}$, we get $w'_0 \sim w_0$. Hence, since $\text{path}_\perp(w'_0, v') \equiv \text{path}_\perp(w_0, v)$, we get $v' \in \mathcal{N}(v)$.

We have proven that the following holds $\bigcup_{v \in \bar{u} \text{ and } \text{lab}_r(v) \in E} \mathcal{N}(v) = \bar{u}$. Since $|\mathcal{N}(v)| = |\bar{v}| \leq \max_{\bar{u} \in \downarrow(\bar{v})} |\bar{u}|$, and there are at most k nodes labelled by an element of E , we get $|\bar{u}| \leq k \cdot \max_{\bar{v} \in \downarrow(\bar{u})} |\bar{v}|$. \square

Proof of Claim 5

By definition of \sim , every equivalence classes contain some node u such that $\text{lab}_r(u) \in E$. Moreover, any two nodes u, v such that $\text{lab}_r(v) =_A \text{lab}_r(u)$ are necessarily equivalent. Hence, the number of \sim -equivalence classes is bounded by the number of $=_A$ -equivalence classes, which is also bounded by $|Q|$.

As a consequence of Claim 4, every \sim -equivalence class \bar{u} has a cardinality bounded by k^{n-1} , where n is the length of the longest \prec_c -chain from \bar{u}_0 to \bar{u} , where u_0 is the root of c .

Hence, the cardinality of every \sim -equivalence class is bounded by $k^{|Q|-1}$.

Hence, for each node $u \in \text{nodes}(t)|_{\sim}$, the frontier of u has at most $k^{|Q|-1}$ elements, so that every node of c has arity at most $k^{|Q|-1}$. Hence, the size of c is bounded by $|Q| \cdot k^{|Q|-1}$. \square

Example We let A be a TAGED such that $Q = \{q_f, q, q', q_x, q'_x, q_y, q_z, q'_z\}$, $F = \{q_f\}$ and $=_A$ is the reflexive, symmetric and transitive closure of the binary relation $\{(q_x, q'_x), (q_z, q'_z), (q_y, q_y)\}$. The set S is therefore defined by $\{q_x, q'_x, q_z, q'_z, q_y\}$. Figure 8 represent a tree t and a run r of A on t . Nodes are represented as naturals. For any node u , we denote by \bar{u} its \sim -equivalence classes. The following table gives all the equivalence classes, and their associated sets of states:

class \bar{u}	$\{3, 8\}$	$\{4, 9\}$	$\{6, 7, 11, 12\}$	$\{1\}$
states(\bar{u})	$\{q_x, q'_x\}$	$\{q_y, q'_y\}$	$\{q_z, q'_z, q, q'\}$	$\{q_f\}$

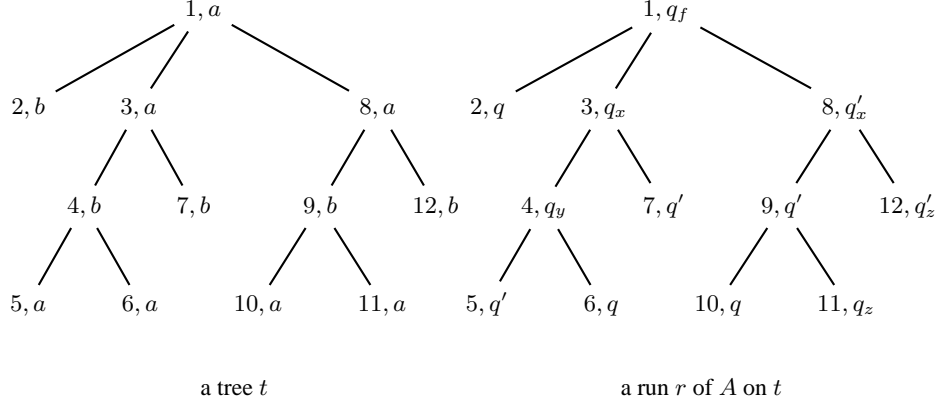


Fig. 8. A tree t (on the left) and a successful run r of a TAGED A on t (on the right), such that $Q = \{q_f, q, q', q_x, q'_x, q_y, q_z, q'_s\}$, $F = \{q_f\}$ and $=_A$ is the reflexive, symmetric and transitive closure of $\{(q_x, q'_x), (q_z, q'_z), (q_y, q_y)\}$. Naturals $1, \dots, 12$ represent the nodes of the two structures, while second components represent the labels (from alphabet $\{a, b\}$ for t , and from alphabet Q for r)

For instance, $3 \sim 8$, since $\text{lab}_r(3) =_A \text{lab}_r(8)$. We also have $6 \sim 11$, since $\text{lab}_r(11) \in S$, $3 \sim 8$, and the path from 3 to 6, in t , is isomorphic to the path from 8 to 11.

The order \prec_c is given by:

$$\{1\} \prec_c \{3, 8\} \prec_c \{4, 9\} \prec_c \{6, 7, 11, 12\}$$

We sum up the frontiers of these nodes in the following table:

node u	1	3	8	4	9	6	7	11	12
frontier(u)	(3,8)	(4,7)	(9,12)	(6)	(11)	()	()	()	()

For every node $u \in \text{nodes}(t)|_{\sim}$, we denote by $\overline{\text{frontier}(u)}$ the sequence $(\bar{u}_1, \dots, \bar{u}_k)$, where $\text{frontier}(u) = (u_1, \dots, u_k)$, we can remark that, for any nodes u, u' such that $u \sim u'$, we have $\overline{\text{frontier}(u)} = \overline{\text{frontier}(u')}$, by definition of the frontier and \sim . For instance:

- $\overline{\text{frontier}(1)} = (\{3, 8\}, \{3, 8\})$
- $\overline{\text{frontier}(u)} = (\{4, 9\}, \{6, 7, 11, 12\}) \forall u \in \{3, 8\}$;
- $\overline{\text{frontier}(u)} = (\{6, 7, 11, 12\}) \forall u \in \{4, 9\}$;
- $\overline{\text{frontier}(u)} = (), \forall u \in \{6, 7, 11, 12\}$

Remark that for any equivalence class \bar{u} , the definition of its successors in c does not depend on the choice of the class representative u .

Finally, the configuration associated with t and r is represented on Figure 9. Naturals indicates the successor order.

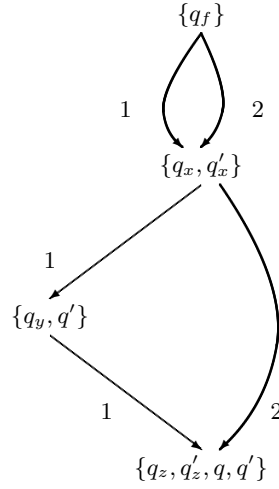


Fig. 9. Configuration obtained from tree and run of Figure 8

A.2 Proof of Lemma 2

In order to prove Lemma 2, we relate the problem to solving a particular case of context disunification, where context variables can take their values in an infinite domain. We first introduce context desunification and prove some useful results. In a second subsection, we prove Lemma 2.

Context Disunification We consider a particular case of context disunification, for which we give sufficient and necessary conditions to ensure the existence of a solution.

We start from an unranked and countable alphabet Σ , a countable set \mathcal{X} of ranked context variables ranging over by X, Y, \dots , and an arity function ar mapping context variables into naturals. We suppose that \mathcal{X} and Σ are disjoint. The set of terms over Σ and \mathcal{X} is denoted by $\mathcal{T}(\Sigma, \mathcal{X})$. The set of *ground term* is given by $\mathcal{T}(\Sigma, \emptyset)$ and is simply denoted by $\mathcal{T}(\Sigma)$. Examples of terms from $\mathcal{T}(\Sigma, \mathcal{X})$ are $a(X(a, b))$ or $X(Y, b)$, where $a, b \in \Sigma$, and X (resp. Y) is a context variable of arity 2 (resp. 0). We identify terms and trees, so that the notions of nodes, labels, order \leq on nodes, carry over to terms. In particular, for any term $t \in \mathcal{T}(\Sigma, \mathcal{X})$, we denote by $\text{nodes}(t)$ its set of nodes, and for all nodes $u \in \text{nodes}(t)$, by $\text{lab}_t(u)$ the label of t at node u .

A *substitution* σ is a mapping from \mathcal{X} into $\mathcal{T}(\Sigma, \mathcal{X})$ which respects arities, i.e. for every variable $X \in \mathcal{X}$, $\sigma(X)$ is of arity $\text{ar}(X)$. We extend σ to terms naturally: $\sigma(a(t_1, \dots, t_n)) = a(\sigma(t_1), \dots, \sigma(t_n))$ and $\sigma(X(t_1, \dots, t_n)) = \sigma(X)[\sigma(t_1), \dots, \sigma(t_n)]$, where $a \in \Sigma$, $t_1, \dots, t_n \in \mathcal{T}(\Sigma, \mathcal{X})$ and $X \in \mathcal{X}$. A substitution σ is *ground* if $\sigma(X)$ is a context over Σ .

A *disequation* is a pair of terms $(t, s) \in \mathcal{T}(\Sigma, \mathcal{X})^2$ and is often denoted by $t \not\approx s$. A *disunification problem* S is a set of disequations $\{t_1 \not\approx s_1, \dots, t_n \not\approx s_n\}$. A *solution* of S is a substitution σ such that for all $i \in \{1, \dots, n\}$, we have $\sigma(t_i) \neq \sigma(s_i)$. A solution

is *ground* if σ is a ground substitution. We can constrain the range of solutions. Let Γ be a mapping from \mathcal{X} into context languages over Σ . A Γ -solution of S is a solution σ of S such that $\sigma(X) \in \Gamma(X)$, for every variable $X \in \mathcal{X}$. In particular, for every disequation $(t \not\approx s) \in S$, $\sigma(t)$ and $\sigma(s)$ are ground.

We consider the following particular form of context disunification problems:

Definition 8. A disunification problem $S = \{t_1 \not\approx t_2, \dots, t_{2n-1} \not\approx t_{2n}\}$ is simple if the following hold:

1. for all $i \in \{1, 3, 5, \dots, 2n-1\}$, $t_i \neq t_{i+1}$;
2. for all variables $X \in \mathcal{X}$, all the subtrees whose root is labelled X , among the subtrees of t_i 's, are equal. In other words, for all $i, j \in \{1, \dots, 2n\}$, and all nodes $u \in \text{nodes}(t_i)$ and $v \in \text{nodes}(t_j)$, if $\text{lab}_{t_i}(u) = \text{lab}_{t_j}(v) = X$, then $t_i|_u = t_j|_v$.

We prove the following theorem in the rest of the section:

Theorem 9. Let S be a disunification problem, and Γ be a mapping from \mathcal{X} into context languages. If S is simple, and $\Gamma(X)$ is infinite, for all $X \in \mathcal{X}$, S admits a Γ -solution.

We start by defining a normal form for simple disunification problems:

Definition 9. A simple disunification problem $S = \{t_1 \not\approx s_1, \dots, t_n \not\approx s_n\}$ is in normal form if for all $i \in \{1, \dots, n\}$, either t_i or s_i (or both) has its root labelled by some variable $X \in \mathcal{X}$.

We denote by $\bigvee_{i=1}^n S_i$ the disjunction of n disunification problems. A substitution σ is a solution of $\bigvee_{i=1}^n S_i$ if it is a solution of one of the S_i 's.

Two disunification problems are said to be *equivalent* if they have the same set of solutions. This definition carries over to disjunction of problems. We have the following:

Lemma 4. Every simple disunification problem is equivalent to a disjunction of simple disunification problems in normal form.

Proof. Starting from a simple problem S , we construct a set of simple problems P , such that S is equivalent to $\bigvee_{S' \in P} S'$. It can be obtained by pushing the disequality constraints down to a context variable, as formalized by the following rewrite rules:

1. $P \cup \{S' \cup \{a(t_1, \dots, t_n) \not\approx a(t'_1, \dots, t'_n)\}\} \implies P \cup \bigcup_{t_i \neq t'_i} \{S' \cup \{t_i \not\approx t'_i\}\}$
2. $P \cup \{S' \cup \{a(t_1, \dots, t_n) \not\approx b(t'_1, \dots, t'_n)\}\} \implies P \cup \{S'\}$
3. $P \cup \{S' \cup \{a(t_1, \dots, t_n) \not\approx a(t'_1, \dots, t'_m)\}\} \implies P \cup \{S'\}$

where $a \in \Sigma$, $n, m \in \mathbb{N}$, $n \neq m$.

First, it is not difficult to see that these rules preserve the two conditions of the definition of simple problems. Let us describe the three rules. Rule 1 push the disequality

1. $S' \cup \{a(t_1, \dots, t_n) \approx a(t'_1, \dots, t'_n)\} \implies S' \cup \{t_i \approx t'_i \mid t_i \neq t'_i, 1 \leq i \leq n\}$
2. $S' \cup \{a(t_1, \dots, t_n) \approx a(t'_1, \dots, t'_m)\} \implies \perp$
3. $S' \cup \{a(t_1, \dots, t_n) \approx b(t'_1, \dots, t'_m)\} \implies \perp$

where $n, m \geq 0, n \neq m, a, b \in \Sigma$.

Fig. 10. Rewrite rules to put a simple context unification problem in normal form

constraint down to the respective sons of the two terms, only for those which are different. Indeed, a problem with a disequation of the form $t \not\approx t$, for some term t , has no solutions, so that it is not necessary to add the problems $S' \cup \{t_i \not\approx t'_i\}$ where $t_i = t'_i$ to the disjunction. This is correct only if the disjunction is nonempty, which is the case since $a(t_1, \dots, t_n) \neq a(t'_1, \dots, t'_n)$.

Rules 2 and 3 remove all the useless disequations, i.e. disequations which are always true. This is because either the root symbols are different, or the number of sons are different.

Again, it is easy to see that applying these rules extensively with starting set $P = \{S\}$ leads to a set of simple problems S_1, \dots, S_n such that $\bigvee_{i=1}^n S_i$ is equivalent to S and every S_i is a simple problem in normal form (otherwise one could apply one of the three rewrite rules). \square

The following lemma, combined with Lemma 4, proves Theorem 9.

Lemma 5. *Let S be a simple disunification problem in normal form, and Γ be a mapping from \mathcal{X} into infinite context languages. It holds that S admits a Γ -solution.*

To prove this lemma, we need an intermediate lemma on context unification. We do not define context unification problem formally as it is similar to context desunification. For instance, a substitution σ is a solution of some unification problem $\{t_1 \approx s_1, t_2 \approx s_2\}$ if $\sigma(t_1) = \sigma(s_1)$ and $\sigma(t_2) = \sigma(s_2)$. The previously defined notions carry over to context unifications. In particular, the definition of *simple* unification problems remains unchanged. We have:

Lemma 6. *Let X be a context variable, and S be a simple unification problem such that for every equation $t \approx s \in S$, we have $t, s \in \mathcal{T}(\Sigma, \{X\})$. There are a finite number of solutions for S .*

Proof. Consider the rewrite rules of Fig. 10, where \perp denotes a problem which has no solutions. We first prove that every rule rewrites a simple problem into an equivalent simple problem. It is clear that the resulting problem is again simple. Let us prove equivalence: it is clear for rules 2 and 3. For rule 1, we know that $a(t_1, \dots, t_n) \neq a(t'_1, \dots, t'_n)$ (hypothesis of the lemma). Hence, for every $i \in \{1, \dots, n\}$, if t_i and t'_i are equal, they are useless to find solutions. In other words, for all contexts C , and all substitutions σ mapping X to C , σ is a solution of $\{a(t_1, \dots, t_n) \approx a(t'_1, \dots, t'_n)\}$

iff σ is a solution of $\{a(t_1, \dots, t_{i-1}, t_{i+1}, \dots, t_n) \approx a(t'_1, \dots, t'_{i-1}, t'_{i+1}, \dots, t'_n)\}$. The forth direction is obvious. For the back direction, suppose that σ is a solution of $\{a(t_1, \dots, t_{i-1}, t_{i+1}, \dots, t_n) \approx a(t'_1, \dots, t'_{i-1}, t'_{i+1}, \dots, t'_n)\}$. Since $t_i = t'_i$, we also have $\sigma(t_i) = \sigma(t'_i)$, hence σ is a solution of $\{a(t_1, \dots, t_n) \approx a(t'_1, \dots, t'_n)\}$.

Consequently, applying these rewrite rules exhaustively on S (it is not difficult to see that it terminates) leads to an equivalent simple problem S' . Moreover, the problem S' is either \perp or has the following form: every equation $t \approx s \in S'$ satisfies $t = X(t_1, \dots, t_n)$ or $s = X(t_1, \dots, t_n)$, for some natural $n \geq 0$ and some terms $t_1, \dots, t_n \in \mathcal{T}(\Sigma, \{X\})$. The solutions of S' are easy to find. If S' is \perp , there are no solutions. If $S' \neq \perp$, then equations of S' are all of the form $X(t_1, \dots, t_n) \approx a(t'_1, \dots, t'_m) \in S'$. Indeed, if $X(t_1, \dots, t_n) \approx X(t'_1, \dots, t'_m)$ is an equation of S' , for some naturals n, m and terms $t_1, \dots, t_n, t'_1, \dots, t'_m$, then by condition 2 of Definition 8, we get $n = m$ and $t_i = t'_i$, for all $i \in \{1, \dots, m\}$. This contradicts condition 1 of Definition 8. Thus, for any equation $X(t_1, \dots, t_n) \approx a(t'_1, \dots, t'_m) \in S'$, there are two cases:

- if some t'_i is not ground, there are no solutions to S' . Indeed, again by condition 2 of Definition 8, it means that $X(t_1, \dots, t_n)$ is a subtree of $a(t'_1, \dots, t'_m)$, so that this equation will never be satisfied;
- if all t'_i 's are ground, since X cannot occur in any t_i 's (again by condition 2 of Definition 8), there are a finite number of contexts C such that $C[t_1, \dots, t_n] = a(t'_1, \dots, t'_m)$, thus a finite number of such contexts which are solutions of S' .

The number of contexts solutions of any equation of S' are finite, so that the number of solutions of S' is finite. \square

We can now prove Lemma 5:

Proof of Lemma 5 We prove this lemma by induction on the number of variables of S . If S has no variables, by definition of normal forms, this implies that $S = \emptyset$, so that every substitution is a solution of S .

Suppose that S has at least one variable. We eliminate some variable X occurring in S by substituting it by some context C from $\Gamma(X)$. We let $S[X \leftarrow C]$ be the problem obtain after substituting X by C . We choose C such that $S[X \leftarrow C]$ satisfies $t \neq s$, for all disequations $(t \not\approx s) \in S[X \leftarrow C]$. We show that it is possible as $\Gamma(X)$ is infinite. It is not difficult to see that $S[X \leftarrow C]$ is a simple problem, so it is equivalent to a disjunction of simple problems $\bigvee_i S_i$. Every S_i has one variable left, so that we can apply induction hypothesis. Hence there exists an Γ -solution σ of $\bigvee_i S_i$. We then show that $\sigma[X \leftarrow C]$ is a solution of $S[X \leftarrow C]$, which will conclude the proof.

Let us now detail all the steps of this proof. We let $\{t_1 \not\approx s_1, \dots, t_n \not\approx s_n\}$ be the problem S . We let X be a variable appearing in S .

Now, we want to substitute X by some context C such that for every $i \in \{1, \dots, n\}$, we have $t_i[X \leftarrow C] \neq s_i[X \leftarrow C]$. For every $i \in \{1, \dots, n\}$, we can view the set $\{t_i \approx s_i\}$ as a simple unification problem in one variable X . The terms t_i and s_i are viewed as terms of $\mathcal{T}(\Sigma \cup (\mathcal{X} - X), \{X\})$. By Lemma 6, there is a finite number of solutions for this unification problem, let us denote them $\sigma_1^i, \dots, \sigma_{n_i}^i$, for some natural

n_i . Taking C different from $\sigma_j^i(X)$, for every $j \in \{1, \dots, n_i\}$ ensures that $t_i[X \leftarrow C] \neq s_i[X \leftarrow C]$.

More generally, we let C be an element of $\Gamma(X) \setminus \bigcup_{i=1}^m \{\sigma_j^i(X) \mid 1 \leq j \leq n_i\}$ (it exists since $\Gamma(X)$ is infinite). Hence, for every disequation $t \not\approx s \in S$, we have $t[X \leftarrow C] \neq s[X \leftarrow C]$.

As we already said, we denote by $S[X \leftarrow C]$ the desunification problem obtained by substituting X with context C . From what we saw in the last paragraph, we deduce that $S[X \leftarrow C]$ is simple. By Lemma 4, $S[X \leftarrow C]$ is equivalent to a disjunction $\bigvee_{i=1}^m S_i$ of problems in normal form, for some $m \geq 0$. Since every S_i has one variable left, we know by induction hypothesis that every S_i admits a Γ -solution σ_i . Let $i \in \{1, \dots, m\}$. Since σ_i is a Γ -solution of S_i , it is also a Γ -solution of $S[X \leftarrow C]$, and we obviously get that $\sigma_i[X \mapsto C]$ is a Γ -solution of S , which conclude the proof. \square

Proof of Lemma 2 We only prove the back direction, since the forth direction is similar to the proof of Proposition 2.

First remark that if there exists a partial interpretation β of c , it implies that $L(A, c) \neq \emptyset$. Suppose that β satisfies the disequality constraints. We let \mathcal{X} be the set $\{\@_u \mid u \in \text{nodes}(c)\}$. For all nodes $u \in \text{nodes}(t)$, we can view $t(u, \beta)$ as a term of $\mathcal{T}(A, \mathcal{X})$, where symbols $\@_u$'s are viewed as context variables. We define a context desunification problem $S_{c, \beta}$ associated with c and β by the set of disequations $t(u, \beta) \not\approx t(v, \beta)$, such that $u, v \in \text{nodes}(c)$, and there exist states $p \in \text{lab}_c(u)$ and $q \in \text{lab}_c(v)$ such that $p \neq_A q$.

Since β satisfies the disequality constraints of c , for all disequations $t \not\approx s \in S_{c, \beta}$, we have $t \neq s$. Moreover, by definition of $t(u, \beta)$, for all nodes $u \in \text{nodes}(c)$, condition 2 of Definition 8 is satisfied by $S_{c, \beta}$. Hence, $S_{c, \beta}$ is a simple disunification problem. We let Γ be a mapping from \mathcal{X} into context languages defined by $\Gamma(\@_u) = \text{cxt}_c(u)$, for all nodes $u \in \text{nodes}(c)$. Hence, $\Gamma(\@_u)$ is infinite, for all context variables $\@_u \in \mathcal{X}$. By Theorem 9, $S_{c, \beta}$ admits a Γ -solution σ . We let λ be the mapping from nodes of c into contexts defined by $\lambda(u) = \beta(u)$ if cxt_u is finite, and by $\lambda(u) = \sigma(u)$ otherwise. It is clear that λ is an interpretation of c which satisfies the disequality constraints of c . Hence, $t(u_0, \lambda) \in L(A)$, where u_0 is the root of c .

Decidability By Lemma 1, the set $\mathcal{D}(A)$ is finite and computable. For every configuration $c \in \mathcal{D}(A)$, there are a finite number of partial interpretations of c , and we can test in polynomial time whether a given partial interpretation satisfies the disequalities of c .

B Proof of Theorem 6

In this section, ϕ is a guarded and recursion-closed TQL formula. We denote by X_1, \dots, X_n its tree variables.

B.1 Preliminaries and notations

Remind that the dual operator $\phi_1 \parallel \phi_2$ stands for $\neg(\neg\phi_1 \mid \neg\phi_2)$, ϕ_1° for $\neg(\neg\phi_1)^*$, \overline{X} for $\neg X \wedge A[\top]$, and $\overline{0}$ for $\neg 0$.

Given two hedges h, h' , we say that h and h' have *the same shape* if $\text{nodes}(h)$ and $\text{nodes}(h')$ are isomorphic and their edge relations are preserved. In this case, we identify $\text{nodes}(h)$ and $\text{nodes}(h')$.

The set of fixpoint variables of any fixpoint equation system Σ is denoted by $\text{fpvar}(\Sigma)$. Remind that its set of tree variables is denoted by $\text{var}(\Sigma)$. We also denote by $\overline{\text{var}(\Sigma)}$ the set $\{\overline{X} \mid X \in \text{var}(\Sigma)\}$. For the sake of uniformity, we define a notion of *boundedness* of equations systems. First, we need to define the set of tree variables of any fixpoint variable of the system.

Let Σ be a system of fixpoint equations. We first define a binary relation \rightarrow_Σ on $\text{fpvar}(\Sigma)$ by $\xi \rightarrow_\Sigma \xi'$ if there exists an equation $\xi = \text{rhs} \in \Sigma$ where ξ' occurs in rhs , for any fixpoint variables $\xi, \xi' \in \text{fpvar}(\Sigma)$. We denote by \rightarrow_Σ^* its transitive closure.

For every $\xi \in \text{fpvar}(\Sigma)$, we define $\text{var}(\xi)$ as the tree variables or negated tree variables that can be reached starting from ξ and following relation \rightarrow_Σ . More formally, $\text{var}(\xi)$ is a subset of $\text{var}(\Sigma) \cup \overline{\text{var}(\Sigma)}$ which consists of variables X (or \overline{X}) such that there exists $\xi' \in \text{fpvar}(\Sigma)$ and a right-hand side $\text{rhs}_{\xi'}$ such that $\xi \rightarrow_\Sigma^* \xi'$, $(\xi' = \text{rhs}_{\xi'}) \in \Sigma$ and X (or \overline{X}) occurs in $\text{rhs}_{\xi'}$.

We are now able to define boundedness of equation systems.

Definition 10 (boundedness of equation systems). *A system of fixpoint equations Σ is bounded if, for all fixpoint variables $\xi, \xi_1, \xi_2 \in \text{fpvar}(\Sigma)$, the following holds:*

- if $(\xi = \xi_1 \parallel \xi_2) \in \Sigma$, then $\text{var}(\xi_1) = \text{var}(\xi_2) = \emptyset$;
- if $(\xi = \xi_1^*) \in \Sigma$, then $\text{var}(\xi_1) = \emptyset$;
- if $(\xi = \xi_1^\circ) \in \Sigma$, then $\text{var}(\xi_1) = \emptyset$;
- if $(\xi = \xi_1 \xi_2) \in \Sigma$, or $(\xi = \xi_1 \wedge \xi_2) \in \Sigma$, then, if $\xi_1 \rightarrow_\Sigma^+ \xi$, we have $\text{var}(\xi_2) = \emptyset$.
Similarly, if $\xi_2 \rightarrow_\Sigma^+ \xi$, we have $\text{var}(\xi_1) = \emptyset$.

B.2 Construction of the fixpoint equation system

In this subsection, we associate ϕ with a system of fixpoint equations S_ϕ . Remark that we do not mention if the system is a system of least fixpoint equations or greatest fixpoint equations, since least and greatest fixpoint collapse as ϕ is guarded.

First, all subformulas \top of ϕ are replaced by $\overline{0} \vee 0$.

Then we push the negations down to the variables or the constant 0. For instance, $\neg\mu\xi.\phi$ is rewritten into $\mu\xi.\neg\phi < \xi \leftarrow \neg\xi >$, where $\phi < \xi \leftarrow \neg\xi >$ is the formula ϕ in which occurrences of the recursion variable ξ have been replaced by $\neg\xi$. Usually, negations of least fixpoints are replaced by greatest fixpoints, but, since every recursion variable is guarded, least fixpoints and greatest fixpoints collapse. Subformulas $\neg(\phi \mid \phi')$ are replaced by $\neg\phi \mid \neg\phi'$, and $\neg(\phi^*)$ by $(\neg\phi)^\circ$. Other connectives are replaced as usual.

Then, every subformula $\neg 0$ is replaced by $\overline{0}$, and every subformula $\neg X$ is replaced by $\overline{X} \vee (\overline{0} \mid \overline{0})$. Since X captures trees only, this rewriting is correct. Hence, one obtains a formula ϕ' equivalent to ϕ , in which no negations occur.

In order to construct system S_ϕ , we introduce a fixpoint variable for each position in the formula. In particular, the last fixpoint variable of S_ϕ corresponds to the root

position in ϕ . For instance, from the formula $\mu\xi.(a[\mu\xi'.(b[\xi'] \vee \xi)] \vee X)$ we construct the following system:

$$\begin{aligned}\xi &= \xi_1 \vee \xi_2 & \xi_1 &= a[\xi'] \\ \xi' &= \xi_3 \vee \xi & \xi_3 &= b[\xi'] \\ \xi_2 &= X\end{aligned}$$

where the last variable is ξ .

Moreover, wlog, for technical reasons (to ensure that the produced TAGED has at least one state) we add an equation $\xi_0 = \Lambda[\xi_0]$ to the system, for a fresh fixpoint variables ξ_0 .

We also require that for every $\xi \in \text{fpvar}(S_\phi)$, there exists exactly one equation in S_ϕ whose left-hand side is ξ .

It is not difficult to see that if ϕ is bounded, then S_ϕ is also bound, as the definition of boundedness for equations systems mimics one for TQL formulas.

We can prove the following lemma, which strenghten Lemma 3:

Lemma 7. *We have $\text{var}(\phi) = \text{var}(S_\phi)$, and, for all valuations $\rho : \text{var}(\phi) \rightarrow \mathbb{T}_A$, the following holds:*

$$\llbracket \phi \rrbracket_\rho = \text{Sol}_{\mathbb{H}}(S_\phi, \rho)(\text{last}(S_\phi))$$

Moreover, S_ϕ contain at most one equation $\xi_0 = \Lambda[\xi_0]$, for some ξ_0 , such that $\text{var}(\xi_0) = \emptyset$, and for every fixpoint variable $\xi \in \text{fpvar}(S_\phi)$, there is exactly one equation whose left-hand side is ξ .

If ϕ is bounded, then S_ϕ is also bounded.

We can characterize boundedness of S_ϕ in terms of the existence of a solution when solving the system with the following interpretation on $(\mathbb{N}, \max, +)$: disjunction is interpreted as \max , conjunction as $+$, \cdot as $+$, $\|$ as 0 (viewed as a constant function), Kleene star \cdot^* as 0, \cdot^\diamond as 0, 0 as 0, $\bar{0}$ as 0, \bar{X} and X as 1, and $\alpha[\cdot]$ as the identity function.

Lemma 8. *If ϕ is bounded, the equation system S_ϕ has a computable solution over $(\mathbb{N}, \max, +)$.*

Proof. Suppose that S_ϕ has no finite solution over $(\mathbb{N}, \max, +)$. It means that there exists $\xi, \xi' \in \text{fpvar}(S_\phi)$ such that $\xi \rightarrow_{S_\phi}^+ \xi' \rightarrow_{S_\phi}^+ \xi$, and ξ' occurs in a right-hand side interpreted by $+$, hence of the form $\xi' \wedge \xi''$ or $\xi' \| \xi''$, where $\text{var}(\xi'') \neq \emptyset$, which contradicts boundedness of S_ϕ .

As every interpretation is monotonic, we can use the classical least fixpoint computation of Tarski.

If ϕ is bounded, we let $b(\xi)$ the solution of any fixpoint variable $\xi \in \text{fpvar}(S_\phi)$ over $(\mathbb{N}, \max, +)$.

B.3 Automata Construction

In this section, we start from S_ϕ , the equation system defined in the previous subsection, and construct a TAGED A_ϕ , which we prove in the next subsection that it satisfies the conditions of Theorem 6.

Let us introduce several notations. We denote by $\text{extvar}(S_\phi) \subseteq \text{fpvar}(S_\phi)$ the set of fixpoint variables of S_ϕ such that there exists an equation $(\xi = \alpha[\xi']) \in S_\phi$, for some ξ' and α . In this case, we denote by $\text{ext}(\xi)$ the extension $\alpha[\xi']$.

We denote by $\text{DVars}(S_\phi)$ (resp. $\text{NDVars}(S_\phi)$) the set of fixpoint variables $\xi \in \text{extvar}(S_\phi)$ such that $\text{var}(\xi) = \emptyset$ (resp. $\text{var}(\xi) \neq \emptyset$).

States For all $\xi \in \text{DVars}(S_\phi)$, we define formulas $\text{Pos}(\xi)$, $\text{Neg}(\xi)$ and $\text{Compl}(\xi)$ by:

$$\text{Pos}(\xi) = \alpha[\xi'] \quad \text{Neg}(\xi) = \alpha[\neg\xi'] \quad \text{Compl}(\xi) = \bar{\alpha}[\top]$$

where $\alpha[\xi'] = \text{ext}(\xi)$. We let $\text{split}(\xi) = \{\text{Pos}(\xi), \text{Neg}(\xi), \text{Compl}(\xi)\}$.

The set of states Q_ϕ of A_ϕ is defined by:

$$Q_\phi = \left(\prod_{\xi \in \text{DVars}(S_\phi)} \text{split}(\xi) \right) \times 2^{\{\text{ext}(\xi) \mid \xi \in \text{DVars}(S_\phi)\}} \times 2^{\text{var}(S_\phi) \cup \overline{\text{var}(S_\phi)}}$$

Intuitively, the first component contain full information on satisfiability (or not) of every atoms of the form $\alpha[\xi]$ such that $\text{var}(\xi) = \emptyset$. The second component contain information on satisfiability of some atoms of the form $\alpha[\xi]$ where $\text{var}(\xi) \neq \emptyset$, as their satisfiability depends on whether subtrees have been captured or not. The last component collects the tree variables (resp. negated tree variables) which capture the current tree.

Let $q \in \prod_{\xi \in \text{DVars}(S_\phi)} \text{split}(\xi)$. We denote by $q(\xi)$ the component of q at rank ξ . States are often denoted by $p = (q_d, q_{nd}, V)$. We also denote by $\pi_d(p)$ the projection of p on its deterministic component, in other words, we let $\pi_d(p) = q_d$. Hence, we have $\pi_d(p) \in \left(\prod_{\xi \in \text{DVars}(S_\phi)} \text{split}(\xi) \right)$.

Transition rules Let $p = (q_d, q_{nd}, V)$ and $p' = (q'_d, q'_{nd}, V')$ be two states from Q_ϕ such that $\pi_d(p) = \pi_d(p')$, i.e. $q_d = q'_d$. We define an *merge operation* on states, denoted \oplus , defined by $p \oplus p' = (q_d, q_{nd} \cup q'_{nd}, V \cup V')$. We denote by Q_ϕ^* the set of words over alphabet Q_ϕ . The operator \oplus is naturally extended to word of states, and to subsets of Q_ϕ^* (in this case, it is denoted \bigoplus). As we will see further, this operation is intended to merge different runs of A_ϕ on the same tree, which come from different branches of an intersection.

Interpretation of S_ϕ over word of states In order to define regular languages of the transitions, we interpret S_ϕ over $(2^{Q_\phi^*}, \cup, \bigoplus)$, with the following interpretations:

- 0 (resp. $\bar{0}$) is interpreted as $\{\epsilon\}$ (resp. $\overline{\{\epsilon\}}^1$);
- disjunction \vee is interpreted as \cup and conjunction \wedge as \bigoplus ;
- every right-hand side of the form $\alpha[\xi']$ is viewed as a constant, so that if $(\xi = \alpha[\xi']) \in S_\phi$, then the interpretation of $\alpha[\xi']$ is:

$$\begin{cases} \{(q_d, q_{nd}, V) \in Q_\phi \mid \alpha[\xi'] \in q_{nd}\} & \text{if } \xi' \in \text{NDVars}(S_\phi) \\ \{(q_d, q_{nd}, V) \in Q_\phi \mid q_d(\xi) = \alpha[\xi']\} & \text{if } \xi' \in \text{DVars}(S_\phi) \end{cases}$$

¹ ϵ denotes the empty word over Q_ϕ , and the overline denotes the complement relatively to Q_ϕ^*

- the binary operator $.\mid.$ is interpreted as the concatenation “.” of words, so that $P\mid P' = \{\bar{p}.\bar{p}' \mid \bar{p} \in P, \bar{p}' \in P'\}$, for all $P, P' \subseteq Q_\phi^*$;
- the binary operator $.\|\.$ is interpreted as $P\|\ P' = \overline{P\mid P'}$, for any sets $P, P' \subseteq Q_\phi^*$;
- Kleene star $.*$ and its dual $.\diamond$ are interpreted similarly to $.\mid.$ and $.\|\.$;
- X is interpreted as $\{(q_d, q_{nd}, V) \in Q_\phi \mid X \in V\}$, and \bar{X} as $\{(q_d, q_{nd}, V) \in Q_\phi \mid \bar{X} \in V\}$.

The solution of S_ϕ , if it exists, is a mapping from $\text{fpvar}(S_\phi)$ into $2^{Q_\phi^*}$, and is denoted by $\text{Sol}_{Q_\phi}(S_\phi)$.

Since ϕ is guarded, and every right-hand side of the form $\alpha[\xi]$ is interpreted as constants, S_ϕ is acyclic when interpreted over $(2^{Q_\phi^*}, \cup, \oplus)$. Hence, it is not difficult to prove the following Lemma:

Lemma 9. *The system S_ϕ has a solution over $(2^{Q_\phi^*}, \cup, \oplus)$, and, for every variable $\xi \in \text{fpvar}(S_\phi)$, the language $\text{Sol}_{Q_\phi}(S_\phi)(\xi)$ is a regular word language over alphabet Q_ϕ .*

Transition rule definition We now define the set of transition rules of A_ϕ , denoted Δ_ϕ . Let $p = (q_d, q_{nd}, V) \in Q_\phi$. For any variable $\xi \in \text{DVars}(S_\phi)$, we define $\text{lab}_p(\xi)$ as the set of labels occurring in $q_d(\xi)$. We associate with p the transition $\alpha_p(L_p) \rightarrow p$, where α_p and L_p are defined by:

$$\alpha_p = \bigcap_{\xi \in \text{DVars}(S_\phi)} \text{lab}_p(\xi) \cap \bigcap_{\alpha[\xi] \in q_{nd}} \alpha \quad L_p = L_{pos,p} \cap L_{neg,p} \cap L_{nd,p}$$

where

$$\begin{aligned} L_{pos,p} &= \bigcap_{\xi \in \text{DVars}(S_\phi)} \{\text{Sol}_{Q_\phi}(S_\phi)(\xi') \mid \exists \alpha, q_d(\xi) = \alpha[\xi']\} \\ L_{neg,p} &= \bigcap_{\xi \in \text{DVars}(S_\phi)} \{Q_\phi^* \setminus \text{Sol}_{Q_\phi}(S_\phi)(\xi') \mid \exists \alpha, q_d(\xi) = \alpha[-\xi']\} \\ L_{nd,p} &= \bigoplus_{\alpha[\xi'] \in q_{nd}} \text{Sol}_{Q_\phi}(S_\phi)(\xi') \end{aligned}$$

Remark that the definitions of α_p and L_p do not depend on V , so that $\alpha_p = \alpha_{p'}$, and $L_p = L_{p'}$, for any state $p' = (q_d, q_{nd}, V')$, for some set of variables V' . Hence, every trees for which the rule $\alpha_p(L_p)$ applies evaluates non-deterministically to a state (q_d, q_{nd}, V') , for some set of variables V' .

Acceptance Condition The final regular state language of A_ϕ , denoted by F_ϕ , is defined by $\text{Sol}_{Q_\phi}(S_\phi)(\text{last}(S_\phi))$.

Equality and disequality relations Finally, we define $=_{A_\phi}$ by $(q_d, q_{nd}, V) =_{A_\phi} (q'_d, q'_{nd}, V')$ if $V \cap V' \cap \text{var}(\phi) \neq \emptyset$ and $\{X \mid X \in V, \bar{X} \in V'\} = \{X \mid X \in V', \bar{X} \in V\} = \emptyset$

The relation \neq_{A_ϕ} is defined by $(q_d, q_{nd}, V) \neq_{A_\phi} (q'_d, q'_{nd}, V')$ if: $(q_d, q_{nd}, V) \neq (q'_d, q'_{nd}, V')$ and, there exists $X \in V$ st $\bar{X} \in V'$ or there exists $X \in V'$, st $\bar{X} \in V$.

B.4 Proof of Correctness

In this section, we prove Theorem 6. We first define several well-founded orders on fixpoint variables and hedges, in order to proofs inductively.

Then, we prove several properties of automaton A_ϕ defined in the latter subsection. In particular, we prove that for all hedges h , and for all runs of A_ϕ on h , runs are equal when projected on their deterministic component. This proves our intuition that A_ϕ simulates an deterministic hedge automaton on its first component.

We then prove that two runs on the same hedge h can be combine via the \oplus operation to form another run on h . This will be usefull when dealing with intersection in equation systems.

We show that, for all fixpoint variables ξ , any word of states which is solution of ξ can be extended via \oplus and is still a solution of ξ .

Finally, we prove the two main lemmata which prove Theorem 6 in both directions.

Well-founded orders We now define a well-founded order on $\mathbb{H}_\Lambda \times \text{fpvar}(S_\phi)$, using the fact that ϕ is guarded, which will allow to do the proof inductively.

We first define a relation (\mathcal{R}) on $\text{fpvar}(S_\phi)$: $\xi(\mathcal{R})\xi'$ if there exists a rule $(\xi = \text{rhs}) \in S_\phi$ such that ξ' occurs in rhs, and rhs is not of the form $\alpha[\xi']$, for some α . We denote by \prec_Ξ its transitive closure. Since ϕ is guarded, by construction of S_ϕ , there does not exist $\xi \in \text{fpvar}(S_\phi)$ such that $\xi \prec_\Xi \xi$. Hence, we can verify that \prec_Ξ is a strict partial and well-founded order on $\text{fpvar}(S_\phi)$.

We also define a partial order on \mathbb{H}_Λ , denoted $\preceq_{\mathbb{H}}$, by $h' \preceq_{\mathbb{H}} h$ if one of the following conditions hold:

- there exists $h_1, h_2 \in \mathbb{H}_\Lambda$ such that $h_1 \neq 0$ or $h_2 \neq 0$, and $h = h_1|h'|h_2$;
- there exists $h_1, h_2 \in \mathbb{H}_\Lambda$, $a \in \Lambda$, and $u \in \text{nodes}(h)$ such that $h|_u = a(h_1|h'|h_2)$.

Finally, we define a strict well-founded order $\prec_{\mathbb{H} \times \Xi}$ as the lexicographic order induced by the well-founded orders $\prec_{\mathbb{H}}$ and \prec_Ξ .

Properties of A_ϕ We prove several properties of A_ϕ . We extend the mapping π_d naturally to words of states. We often denote word of states over Q_ϕ by \bar{p} .

For the sake of clarity we prove the following lemma at the end of the section. It states that being solution of a variable ξ over word of states only depends on the deterministic component.

Lemma 10. *Let $\xi \in \text{fpvar}(S_\phi)$, and let $\bar{p}, \bar{p}' \in Q_\phi^*$. If $\pi_d(\bar{p}) = \pi_d(\bar{p}')$ and $\text{var}(\xi) = \emptyset$, then $\bar{p} \in \text{Sol}_{Q_\phi}(S_\phi)(\xi)$ iff $\bar{p}' \in \text{Sol}_{Q_\phi}(S_\phi)(\xi)$.*

For any word of states \bar{p} , we denote by $|\bar{p}|$ its length. We can show the following lemma similarly to Lemma 10.

Lemma 11. *Let $\xi \in \text{fpvar}(S_\phi)$, and let $\bar{p}, \bar{q} \in Q_\phi^*$ such that $|\bar{p}| = |\bar{q}|$ and $\pi_d(\bar{p}) = \pi_d(\bar{q})$. If $\bar{p} \in \text{Sol}_{Q_\phi}(S_\phi)(\xi)$, then $\bar{p} \oplus \bar{q} \in \text{Sol}_{Q_\phi}(S_\phi)(\xi)$.*

The following lemma expresses then that A_ϕ is deterministic on its first component:

Lemma 12. *Let $h \in \mathbb{H}_\Lambda$ be a hedge, and $r, r' \in R_{A_\phi}(h)$ two runs of A_ϕ on h , then we have:*

$$\forall u \in \text{nodes}(h), \pi_d(\text{lab}_r(u)) = \pi_d(\text{lab}_{r'}(u))$$

Proof. The proof goes by induction on h

- if $h = 0$, then $r = r' = 0$;
- if $h = a(0)$, there exists two states $p = (q_d, q_{nd}, V)$ and $p' = (q'_d, q'_{nd}, V')$ such that $r = p(0)$ and $r' = p'(0)$. Hence, $a \in \alpha_p \cap \alpha_{p'}$, and $\epsilon \in L_p \cap L_{p'}$. Suppose that $q_d \neq q'_d$, then there exists ξ such that $q_d(\xi) \neq q'_d(\xi)$. There are three cases (other cases are symmetric):
 - $q_d(\xi)$ is of the form $\alpha[\xi']$ and $q'_d(\xi)$ is of the form $\bar{\alpha}[\top]$. In this case, we get the contradiction $a \in \alpha \cap \bar{\alpha}$.
 - $q_d(\xi)$ is of the form $\alpha[\neg\xi']$ and $q'_d(\xi)$ is of the form $\bar{\alpha}[\top]$. This case is similar to the previous case.
 - $q_d(\xi)$ is of the form $\alpha[\xi']$ and $q'_d(\xi)$ of the form $\alpha[\neg\xi']$. In this case, we get the contradiction $\epsilon \in \text{Sol}_{Q_\phi}(S_\phi)(\xi') \cap Q_\phi^* \setminus \text{Sol}_{Q_\phi}(S_\phi)(\xi')$.
- if $h = a(h_1)$, then by induction hypothesis, there exists two runs r_1, r'_1 on h_1 , and two states $p = (q_d, q_{nd}, V)$ and $p' = (q'_d, q'_{nd}, V')$ such that $r = p(r_1)$ and $r' = p'(r'_1)$. Suppose that $q_d \neq q'_d$. There is three cases (other cases are symmetric):
 - $q_d(\xi)$ is of the form $\alpha[\xi']$ and $q'_d(\xi)$ is of the form $\bar{\alpha}[\top]$. In this case, we get the contradiction $a \in \alpha \cap \bar{\alpha}$.
 - $q_d(\xi)$ is of the form $\alpha[\neg\xi']$ and $q'_d(\xi)$ is of the form $\bar{\alpha}[\top]$. This case is similar to the previous case.
 - $q_d(\xi)$ is of the form $\alpha[\xi']$ and $q'_d(\xi)$ of the form $\alpha[\neg\xi']$. In this case, we have $\text{roots}(r_1) \in \text{Sol}_{Q_\phi}(S_\phi)(\xi')$ and $\text{roots}(r'_1) \notin \text{Sol}_{Q_\phi}(S_\phi)(\xi')$. Since $\xi' \in \text{DVars}(S_\phi)$, we get a contradiction by Lemma 10.
- if $h = h_1|h_2$, with $h_1 \neq 0$ and $h_2 \neq 0$, then it is obvious by induction hypothesis on h_1 and h_2 , and by Lemma 10.

□.

Definition 11 (run constrained by a valuation). *Let $\rho : \text{var}(\phi) \rightarrow \mathbb{T}_\Lambda$ be a valuation, and $h \in \mathbb{H}_\Lambda$ be a hedge. The set of runs of A_ϕ on h constrained by ρ , denoted $R_{A_\phi}^\rho(h)$, is defined as set of runs r of A_ϕ on h such that: for all nodes $u \in \text{nodes}(r)$, for all state $p = (q_d, q_{nd}, V) \in Q_\phi$, if $\text{lab}_r(u) = p$, then for all tree variable $X \in \text{var}(\phi)$, the following holds:*

- if $X \in V$, then $h|_u = \rho(X)$;
- if $\bar{X} \in V$, then $h|_u \neq \rho(X)$.

Remark that a constrained run necessarily respects the equality and disequality constraints.

On merging runs As already said, we extend the \oplus operator on runs with the same shape. Let $h \in \mathbb{H}_\Lambda$ be a hedge, and for any two runs $r, r' \in R_{A_\phi}(h)$ on h , $r \oplus r'$ is the tree which has the same shape as r and r' and such that each of its labels is obtained by applying \oplus on the respective labels of r and r' . More formally, for every

node $u \in \text{nodes}(r)$, $\text{lab}_{r \oplus r'}(u)$ is defined by $\text{lab}_r(u) \oplus \text{lab}_{r'}(u)$. Note that $r \oplus r'$ necessarily exists, since $\pi_d(r) = \pi_d(r')$, by Lemma 12.

The following lemma expresses that the merging of two runs which respect the constraints is still a run which respect the constraints.

Lemma 13. *Let $h \in \mathbb{H}$ and $\rho : \text{var}(\phi) \rightarrow \mathbb{T}_A$. Let $r, r' \in R_{A_\phi}^\rho(h)$, then $r \oplus r'$ exists and belongs to $R_{A_\phi}^\rho(h)$.*

Proof. The proof goes by induction on h . Intuitively, the only things which differ between r, r' and $r \oplus r'$ are the second and last components, but this does not disturb the transitions, by definition of set of transition rules.

- if $h = 0$, then $r = r' = r \oplus r' = 0$;
- if $h = a(h_1)$, then there exists two runs $r_1, r'_1 \in R_{A_\phi}^\rho(h_1)$, and two states $p = (q_d, q_{nd}, V)$ and $p' = (q'_d, q'_{nd}, V')$ such that $r = p(r_1)$ and $r' = p(r'_1)$. By induction hypothesis, $r_1 \oplus r'_1 \in R_{A_\phi}^\rho(h_1)$.

We now prove that $a \in \alpha_{p \oplus p'}$ and $\text{roots}(r_1) \oplus \text{roots}(r'_1) \in L_{p \oplus p'}$. It is not difficult to see that $\alpha_{p \oplus p'} = \alpha_p \cap \alpha_{p'}$, so that $a \in \alpha_p \cap \alpha_{p'}$.

Now, remind that L_p is of the form $L_{pos,p} \cap L_{neg,p} \cap L_{nd,p}$, and $L_{p'}$ of the form $L_{pos,p'} \cap L_{neg,p'} \cap L_{nd,p'}$. It remains to show that $\text{roots}(r_1) \oplus \text{roots}(r'_1) \in L_{pos,p \oplus p'} \cap L_{neg,p \oplus p'} \cap L_{nd,p \oplus p'}$.

First remark that the definitions of $L_{pos,p}, L_{neg,p}$, (resp. $L_{pos,p'}$ and $L_{neg,p'}$), only depend on q_d (resp. q'_d). Therefore, since $q_d = q'_d$, we get $L_{pos,p} = L_{pos,p'} = L_{pos,p \oplus p'}$, and $L_{neg,p} = L_{neg,p'} = L_{neg,p \oplus p'}$.

Let $\xi \in \text{DVars}(S_\phi)$ be a fixpoint variable. We consider two cases, depending on the form of $\text{ext}(\xi)$:

- if $\text{ext}(\xi)$ is of the form $\alpha[\xi']$, for some α and ξ' , by definition of $L_{pos,p}$, we have $\text{roots}(r_1) \in \text{Sol}_{Q_\phi}(\xi')$. As $\text{var}(\xi') = \emptyset$ and $\pi_d(\text{roots}(r_1) \oplus \text{roots}(r'_1)) = \pi_d(\text{roots}(r_1) \oplus \text{roots}(r'_1))$, by Lemma 10, we get $\text{roots}(r_1) \oplus \text{roots}(r'_1) \in \text{Sol}_{Q_\phi}(\xi')$;
- if $\text{ext}(\xi)$ is of the form $\alpha[\neg\xi']$, for some α and ξ' , by definition of $L_{neg,p}$, we have $\text{roots}(r_1) \notin \text{Sol}_{Q_\phi}(\xi')$. As $\text{var}(\xi') = \emptyset$ and $\pi_d(\text{roots}(r_1) \oplus \text{roots}(r'_1)) = \pi_d(\text{roots}(r_1) \oplus \text{roots}(r'_1))$, by Lemma 10, we get $\text{roots}(r_1) \oplus \text{roots}(r'_1) \notin \text{Sol}_{Q_\phi}(\xi')$.

Hence, by definition of $L_{p \oplus p'}$, we can conclude that $\text{roots}(r_1) \oplus \text{roots}(r'_1) \in L_{p \oplus p'}$.

Now, since, $\text{roots}(r_1) \in L_{nd,p}$ and $\text{roots}(r'_1) \in L_{nd,p'}$, we get $\text{roots}(r_1) \oplus \text{roots}(r'_1) \in (L_{nd,p} \oplus L_{nd,p'})$. Hence, we have $\text{roots}(r_1) \oplus \text{roots}(r'_1) \in L_{nd,p \oplus p'}$, by definition of $L_{nd,p \oplus p'}$.

Finally, $\text{roots}(r_1) \oplus \text{roots}(r'_1) \in L_{p \oplus p'}$, so that we have $(p \oplus p')(r_1 \oplus r'_1) \in R_{A_\phi}(a(h_1))$.

- if $h = h_1|h_2$, with $h_1 \neq 0$ and $h_2 \neq 0$, then it is obvious by induction hypothesis on h_1 and h_2 .

□.

For any hedge $h \in \mathbb{H}_A$ and any run $r \in R_{A_\phi}(h)$, we say that r is *closed* if for all nodes $u \in \text{nodes}(h)$, $\text{lab}_r(u)$ is of the form $(q_d, \emptyset, \emptyset)$.

We prove at the end of the section:

Proposition 4. *For all hedges $h \in \mathbb{H}_A$, there exists exactly one run $r \in R_{A_\phi}(h)$, such that r is closed.*

Main Lemmata We are now able to prove the two main lemmata of this subsection.

We denote by $R_{A_\phi, \bar{p}}^\rho(h)$ the set of runs r of A_ϕ on hedge h , constrained by valuation ρ such that it evaluated h to the word of states \bar{p} .

Remind if ϕ is bounded, we denote by $b(\xi)$ the solution of S_ϕ over $(\mathbb{N}, \text{max}, +)$ in variable ξ .

Lemma 14. *Let $\rho : \text{var}(\phi) \rightarrow \mathbb{T}_A$ be a valuation, let $\xi \in \text{fpvar}(S_\phi)$ be a fixpoint variable, and $h \in \text{Sol}_{\mathbb{H}}(S_\phi, \rho)(\xi)$ be a hedge.*

There exist a word of states $\bar{p} \in \text{Sol}_{Q_\phi}(S_\phi)(\xi)$ and a run $r \in R_{A_\phi, \bar{p}}^\rho(h)$. Moreover, if ϕ is bounded, then r is bounded by $b(\xi)$.

Proof. The proof goes by induction on (h, ξ) . At each step, (h, ξ) decreases by $\prec_{\mathbb{H} \times \Xi}$.

- if $(\xi = 0)$, then necessarily $h = 0$, and we get the result, since $\epsilon \in \text{Sol}_{Q_\phi}(S_\phi)(\xi)$;
- if $(\xi = \alpha[\xi']) \in S_\phi$, then there exists $h' \in \mathbb{H}_A$ and $a \in \alpha$ such that $h' \in \text{Sol}_{\mathbb{H}}(S_\phi, \rho)(\xi')$. By induction hypothesis on (h', ξ') , there exist a word of states $\bar{p}' \in \text{Sol}_{Q_\phi}(S_\phi)(\xi')$ and a run $r' \in R_{A_\phi, \bar{p}'}^\rho(h')$.

Moreover, if ϕ is bounded, then r' is bounded by $b(\xi')$.

Now, we define a state $(q_d, q_{nd}, V) \in Q_\phi$.

For any variable $\xi_1 \in \text{DVars}(S_\phi)$ and extension $\alpha_1[\xi'_1]$ such that $(\xi_1 = \alpha_1[\xi'_1]) \in S_\phi$, for some ξ'_1 and α_1 , we define $s(\xi_1)$, an element of $\text{split}(\xi_1)$, by:

$$s(\xi_1) = \begin{cases} \bar{\alpha}_1[\top] & \text{if } a \notin \alpha_1 \\ \alpha_1[\xi'_1] & \text{if } a \in \alpha_1 \text{ and } \bar{p}' \in \text{Sol}_{Q_\phi}(S_\phi)(\xi'_1) \\ \alpha_1[\neg \xi'_1] & \text{if } a \in \alpha_1 \text{ and } \bar{p}' \notin \text{Sol}_{Q_\phi}(S_\phi)(\xi'_1) \end{cases}$$

We define q_d as an element of $\prod_{\xi_1 \in \text{DVars}(S_\phi)} \text{split}(\xi_1)$ such that for all variables $\xi_1 \in \text{DVars}(S_\phi)$, $q_d(\xi_1) = s(\xi_1)$.

Now, we define q_{nd} by:

$$\begin{cases} \emptyset & \text{if } \xi \notin \text{NDVars}(S_\phi) \\ \{\alpha[\xi']\} & \text{if } \xi' \in \text{NDVars}(S_\phi) \end{cases}$$

Finally, we define V by \emptyset and let $p = (q_d, q_{nd}, V)$. By definition of p , it is not difficult to see that $a \in \alpha_p$, and $\bar{p}' \in L_p$ (assuming that $\bigoplus \emptyset = Q_\phi^*$).

Since $V = \emptyset$, we get $p[r'] \in R_{A_\phi, p}^\rho(h)$.

Moreover, if ϕ is bounded, since $b(\xi) = b(\xi')$, $V = \emptyset$, and r' is bounded by $b(\xi')$, $p[r']$ is bounded by $b(\xi)$.

- if $(\xi = \xi_1 \wedge \xi_2) \in S_\phi$, then $h \in \text{Sol}_{\mathbb{H}}(S_\phi, \rho)(\xi_1)$ and $h \in \text{Sol}_{\mathbb{H}}(S_\phi, \rho)(\xi_2)$. By induction hypothesis, there exist two words of states $\bar{p}_1 \in \text{Sol}_{Q_\phi}(S_\phi)(\xi_1)$ and $\bar{p}_2 \in \text{Sol}_{Q_\phi}(S_\phi)(\xi_2)$, and two runs $r_1 \in R_{A_\phi, \bar{p}_1}^\rho(h)$, $r_2 \in R_{A_\phi, \bar{p}_2}^\rho(h)$. By Lemma 13, $r_1 \oplus r_2 \in R_{A_\phi, p_1 \oplus p_2}^\rho(h)$. It remains to show that $p_1 \bar{p}_2 \in \text{Sol}_{Q_\phi}(S_\phi)(\xi)$, which is obvious, since $\text{Sol}_{Q_\phi}(S_\phi)(\xi) = \text{Sol}_{Q_\phi}(S_\phi)(\xi_1) \oplus \text{Sol}_{Q_\phi}(S_\phi)(\xi_2)$. Moreover, if ϕ is bounded, by induction hypothesis, r_1 is bounded by $b(\xi_1)$ and r_2 by $b(\xi_2)$, so that $r_1 \oplus r_2$ is bounded by $b(\xi_1) + b(\xi_2)$, i.e. $b(\xi)$.
- if $\xi = \xi_1 || \xi_2$, then by Proposition 4, there exists a unique closed run $r \in R_{A_\phi}(h)$. Since r is closed, we also have $r \in R_{A_\phi}^\rho(h)$. Let h_1, h_2 be two hedges such that $h = h_1 | h_2$. We denote by r_{h_1, h_2} the hedge over Q defined by:

$$r_{h_1, h_2} = \bigoplus_{(r_1, r_2) \in R_{A_\phi}^\rho(h_1) \times R_{A_\phi}^\rho(h_2)} r_1 | r_2$$

First remark that r_{h_1, h_2} exists, bu Lemma 13. Moreover, again by Lemma 13, we have $r_{h_1, h_2} \in R_{A_\phi}^\rho(h_1 | h_2)$.

We now define a hedge r over Q by:

$$r = \bigoplus_{h_1 | h_2 = h} r_{h_1, h_2}$$

Still by Lemma 13, we have $r \in R_{A_\phi}^\rho(h_1 | h_2)$.

It remains to prove that $\text{roots}(r) \in \text{Sol}_{Q_\phi}(S_\phi)(\xi)$. Let $\bar{p}_1, \bar{p}_2 \in Q_\phi^*$ such that $\text{roots}(r) = \bar{p}_1 \cdot \bar{p}_2$. Let $h_1, h_2 \in \mathbb{H}_A$, such that $h = h_1 | h_2$, $|h_1| = |\bar{p}_1|$, and $|h_2| = |\bar{p}_2|$. By definition of $|\cdot|$, either $h_1 \in \text{Sol}_{\mathbb{H}}(S_\phi, \rho)(\xi_1)$ or $h_2 \in \text{Sol}_{\mathbb{H}}(S_\phi, \rho)(\xi_2)$. Suppose that $h_1 \in \text{Sol}_{\mathbb{H}}(S_\phi, \rho)(\xi_1)$. By induction hypothesis, there exists a run $r_1 \in R_{A_\phi}^\rho(h_1)$ such that $\text{roots}(r_1) \in \text{Sol}_{Q_\phi}(S_\phi)(\xi_1)$.

Let r'_1, r'_2 be the two hedges over Q such that $r = r'_1 | r'_2$ and the length of r'_1 is equal to the length of r_1 . By definition of r , there exists a hedge r''_1 over Q such that $r'_1 = r_1 \oplus r''_1$. Since $\text{roots}(r_1) \in \text{Sol}_{Q_\phi}(S_\phi)(\xi_1)$, we also get, by Lemma 11, we have $\text{roots}(r'_1) \in \text{Sol}_{Q_\phi}(S_\phi)(\xi_1)$. In other words, we have $\bar{p}_1 \in \text{Sol}_{Q_\phi}(S_\phi)(\xi_1)$. Symmetrically, if we suppose that $h_2 \in \text{Sol}_{\mathbb{H}}(S_\phi, \rho)(\xi_2)$, we can prove that $\bar{p}_2 \in \text{Sol}_{Q_\phi}(S_\phi)(\xi_2)$.

Hence, we have either $\bar{p}_1 \in \text{Sol}_{Q_\phi}(S_\phi)(\xi_1)$ or $\bar{p}_2 \in \text{Sol}_{Q_\phi}(S_\phi)(\xi_2)$, which suffices to conclude.

The proof is a bit different when ϕ is bounded. If ϕ is bounded, we have $\text{var}(\xi) = \text{var}(\xi_1) = \text{var}(\xi_2) = \emptyset$, so that $\xi, \xi_1, \xi_2 \in \text{DVars}(S_\phi)$ and $b(\xi) = 0$. By Proposition 4, there exists a unique closed run r of A_ϕ over h . This run is obviously bounded by $b(\xi) = 0$, since it is closed. It remains to prove that $\text{roots}(r) \in \text{Sol}_{Q_\phi}(S_\phi)(\xi)$.

Let $\bar{p}_1, \bar{p}_2 \in Q_\phi^*$ such that $\text{roots}(r) = \bar{p}_1 \cdot \bar{p}_2$. Let $h_1, h_2 \in \mathbb{H}_A$, such that $h = h_1 | h_2$, $|h_1| = |\bar{p}_1|$, and $|h_2| = |\bar{p}_2|$. By definition of $|\cdot|$, either $h_1 \in \text{Sol}_{\mathbb{H}}(S_\phi, \rho)(\xi_1)$ or $h_2 \in \text{Sol}_{\mathbb{H}}(S_\phi, \rho)(\xi_2)$. Suppose that $h_1 \in \text{Sol}_{\mathbb{H}}(S_\phi, \rho)(\xi_1)$. By induction hypothesis, there exists a run $r_1 \in R_{A_\phi}(h_1)$ such that $\text{roots}(r_1) \in \text{Sol}_{Q_\phi}(S_\phi)(\xi_1)$. By Lemma 12, $\pi_d(\text{roots}(r_1)) = \pi_d(\bar{p}_1)$, and by Lemma 10, $\bar{p}_1 \in \text{Sol}_{Q_\phi}(S_\phi)(\xi_1)$.

Similarly, when $h_2 \in \text{Sol}_{\mathbb{H}}(S_\phi, \rho)(\xi_2)$, one can prove that $\bar{p}_2 \in \text{Sol}_{Q_\phi}(S_\phi)(\xi_1)$. Finally, either $\bar{p}_1 \in \text{Sol}_{Q_\phi}(S_\phi)(\xi_1)$ or $\bar{p}_2 \in \text{Sol}_{Q_\phi}(S_\phi)(\xi_2)$, which is sufficient to conclude that $\text{roots}(r) \in \text{Sol}_{Q_\phi}(S_\phi)(\xi)$.

- if $(\xi = X) \in S_\phi$, then necessarily $h = \rho(X)$ and h is a tree. By Proposition 4, there exists a closed run $r \in R_{A_\phi}(h)$, and *a fortiori*, we have $r \in R_{A_\phi}^\rho(h)$. We let p be equal to $\text{roots}(r)$, and let r' such that $r = p(r')$. The state p is of the form $(q_d, \emptyset, \emptyset)$ for some q_d . Let $p' = (q_d, \emptyset, \{X\})$. Since the definition of transition rules does not depend on the third component, it is not difficult to see that $p'(r') \in R_{A_\phi}^\rho(h)$. Moreover, $p' \in \text{Sol}_{Q_\phi}(S_\phi)(\xi)$. If ϕ is bounded, since r' is bounded by 0, r is bounded by 1, i.e. by $b(\xi)$;
- all other cases are either similar to the previous cases, or obvious.

□.

We now prove the converse. Let $h \in \mathbb{H}_\Lambda$, and let $r \in R_{A_\phi}(h)$ be a run which respects the equality and disequality constraints of A_ϕ . We let Val_r be the set of valuations $\rho : \text{var}(\phi) \rightarrow \mathbb{T}_\Lambda$ such that for all tree variables $X \in \text{var}(\phi)$, all nodes $u \in \text{nodes}(r)$, and all states $p \in Q_\phi$ of the form (q_d, q_{nd}, V) , if $\text{Lab}_r(u) = p$, then the following hold:

- if $X \in V$, then $\rho(X) = h|_u$;
- if $\bar{X} \in V$, then $\rho(X) \neq h|_u$

Note that we have $\text{Val}_r \neq \emptyset$, by definition of $=_{A_\phi}$ and \neq_{A_ϕ} .

Lemma 15. *Let $h \in \mathbb{H}_\Lambda$, $\xi \in \text{fpvar}(S_\phi)$, and $\bar{p} \in \text{Sol}_{Q_\phi}(S_\phi)(\xi)$. If there exists a run $r \in R_{A_\phi, \bar{p}}(h)$ which respects the equality and disequality constraints of A_ϕ , then for all valuations $\rho \in \text{Val}_r$, we have $h \in \text{Sol}_{\mathbb{H}}(S_\phi, \rho)(\xi)$.*

Proof. The proof goes by induction on (h, ξ) .

- if $(\xi = 0) \in S_\phi$, then necessarily, $\bar{p} = \epsilon$ and $h = 0$, hence $h \in \text{Sol}_{\mathbb{H}}(S_\phi, \rho)(\xi)$, for any valuation ρ ;
- if $(\xi = \bar{0}) \in S_\phi$, then every hedge belongs to $\text{Sol}_{\mathbb{H}}(S_\phi, \rho)(\xi)$, for any valuation ρ ;
- if $(\xi = \alpha[\xi']) \in S_\phi$, then $|\bar{p}| = 1$ and \bar{p} is of the form $(q_d, q_{nd}, V) \in Q_\phi$. Moreover, $h = a(h')$ for some $h' \in \mathbb{H}_\Lambda$ and $a \in \Lambda$, and there exist a word of states $\bar{p}' \in Q_\phi^*$ and a run $r' \in R_{A_\phi, \bar{p}'}(h')$ such that $r = \bar{p}(r')$. We show that $\bar{p}' \in \text{Sol}_{Q_\phi}(S_\phi)(\xi')$. We consider two cases, depending on whether $\xi \in \text{DVars}(S_\phi)$ or $\xi \in \text{NDVars}(S_\phi)$:
 - if $\xi \in \text{DVars}(S_\phi)$, then necessarily, $q_d(\xi) = \alpha[\xi']$, and by definition of $L_{\bar{p}}$, since $\bar{p}' \in L_{\bar{p}}$, we have $\bar{p}' \in \text{Sol}_{Q_\phi}(S_\phi)(\xi')$;
 - if $\xi \in \text{NDVars}(S_\phi)$, then necessarily, $\alpha[\xi'] \in q_{nd}$. Hence, there exist $\bar{p}_1, \bar{p}_2 \in Q_\phi^*$, such that $\bar{p}_1 \in \text{Sol}_{Q_\phi}(S_\phi)(\xi')$ and $\bar{p}' = \bar{p}_1 \oplus \bar{p}_2$. By Lemma 11, we get $\bar{p}' \in \text{Sol}_{Q_\phi}(S_\phi)(\xi')$.

Hence, we can apply the induction hypothesis on (h', ξ') , so that for all valuations $\rho' \in \text{Val}_{r'}$, we have $h' \in \text{Sol}_{\mathbb{H}}(S_\phi, \rho')(\xi')$. Moreover, since $\text{Val}_r \subseteq \text{Val}_{r'}$, we also have, for all valuations $\rho \in \text{Val}_r$, $h' \in \text{Sol}_{\mathbb{H}}(S_\phi, \rho)(\xi')$. Hence, the following holds: $\forall \rho \in \text{Val}_r$, $h \in \text{Sol}_{\mathbb{H}}(S_\phi, \rho)(\xi)$;

- if $(\xi = \xi_1 || \xi_2) \in S_\phi$, then let $\rho \in \text{Val}_r$ and $h_1, h_2 \in \mathbb{H}_A$ such that $h = h_1 | h_2$. Let r_1, r_2 be the runs associated to h_1, h_2 , i.e. such that $r = r_1 | r_2$, $r_1 \in R_{A_\phi, \text{roots}(r_1)}(h_1)$ and $r_2 \in R_{A_\phi, \text{roots}(r_2)}(h_2)$. Since $p = \text{roots}(r) \in \text{Sol}_{Q_\phi}(S_\phi)(\xi)$, either $\text{roots}(r_1) \in \text{Sol}_{Q_\phi}(S_\phi)(\xi_1)$ or $\text{roots}(r_2) \in \text{Sol}_{Q_\phi}(S_\phi)(\xi_2)$. By induction hypothesis, $\exists i \in \{1, 2\}$ such that $\rho_i \in \text{Val}_{r_i}$, $h \in \text{Sol}_{\mathbb{H}}(S_\phi, \rho_i)(\xi_i)$. Since $\text{Val}_{r_i} \subseteq \text{Val}_r$, we get $h \in \text{Sol}_{\mathbb{H}}(S_\phi, \rho)(\xi)$. Hence, $h \in \text{Sol}_{\mathbb{H}}(S_\phi, \rho)(\xi)$.
- if $(\xi = X)$, let $\rho \in \text{Val}_r$. Necessarily, $h = \rho(X)$, and $h \in \text{Sol}_{\mathbb{H}}(S_\phi, \rho)(\xi)$;
- other cases are either similar or obvious.

□

Proof of Theorem 6 Now we can prove the main theorem of the section.

Suppose that there exists $h \in L(A_\phi)$. By Lemma 15, there exists a valuation $\rho : \text{var}(\phi) \rightarrow \mathbb{T}_A$ such that $h \in \text{Sol}_{\mathbb{H}}(S_\phi, \rho)(\text{last}(S_\phi))$, and by Lemma 7, $h \in \llbracket \phi \rrbracket_\rho$.

Conversely, if there exists a valuation $\rho : \text{var}(\phi) \rightarrow \mathbb{T}_A$ and a hedge $h \in \llbracket \phi \rrbracket_\rho$, then by Lemma 7, $h \in \text{Sol}_{\mathbb{H}}(S_\phi, \rho)(\text{last}(S_\phi))$. By Lemma 14, there exists $\bar{p} \in \text{Sol}_{Q_\phi}(S_\phi)(\text{last}(S_\phi))$ such that $R_{A_\phi, \bar{p}}^\rho(h) \neq \emptyset$. Since \bar{p} is a final, i.e. $\bar{p} \in F_\phi$, we get $h \in L(A_\phi)$.

Now, suppose that ϕ is bounded. We let $k = \text{Sol}_{\mathbb{N}}(S_\phi)(\text{last}(S_\phi))$. One can add to A_ϕ a regular control which ensures that every accepting run of the TAGED A_ϕ is bounded by k . Let us denote $A_\phi^{\leq k}$ the resulting TAGED. Hence, $A_\phi^{\leq k}$ is equivalent to the bounded TAGED $(A_\phi^{\leq k}, k)$, which is obviously computable. □

B.5 Omitted Proofs of Section B

Proof of Lemma 10 The proof goes by induction on ξ . At each step, ξ decreases by \prec_Ξ .

- $(\xi = \alpha[\xi']) \in S_\phi$. By definition of the interpretation of $\alpha[\cdot]$, if $\bar{p} \in \text{Sol}_{Q_\phi}(S_\phi)(\xi)$, then \bar{p} is of the form (q_d, q_{nd}, V) for some q_d, q_{nd}, V , and $q_d(\xi) = \alpha[\xi']$. If $\pi_d(\bar{p}) = \pi_d(\bar{p}')$, then $\pi_d(\bar{p}') = q_d$ and $q_d(\xi) = \alpha[\xi']$. Again by definition of the interpretation of $\alpha[\cdot]$, every state $(q'_d, q'_{nd}, V') \in Q_\phi$ such that $q'_d(\xi) = \alpha[\xi']$ is a solution of ξ in Q_ϕ^* . A fortiori, $\bar{p}' \in \text{Sol}_{Q_\phi}(S_\phi)(\xi)$;
- $(\xi = \xi_1 || \xi_2) \in S_\phi$. Suppose that $\bar{p} \in \text{Sol}_{Q_\phi}(S_\phi)(\xi)$. Let $\bar{p}'_1, \bar{p}'_2 \in Q_\phi^*$ such that $\bar{p}' = \bar{p}'_1 \cdot \bar{p}'_2$. Let $\bar{p}_1, \bar{p}_2 \in Q_\phi^*$ such that $\bar{p} = \bar{p}_1 \cdot \bar{p}_2$, $|\bar{p}_1| = |\bar{p}'_1|$ and $|\bar{p}_2| = |\bar{p}'_2|$ (\bar{p}_1 and \bar{p}_2 exist since $|\bar{p}| = |\bar{p}'|$). By definition of the interpretation of \cdot , either $\bar{p}_1 \in \text{Sol}_{Q_\phi}(S_\phi)(\xi_1)$ or $\bar{p}_2 \in \text{Sol}_{Q_\phi}(S_\phi)(\xi_2)$. One apply the induction hypothesis to get $\bar{p}'_1 \in \text{Sol}_{Q_\phi}(S_\phi)(\xi_1)$ or $\bar{p}'_2 \in \text{Sol}_{Q_\phi}(S_\phi)(\xi_2)$. Hence, $\bar{p}' \in \text{Sol}_{Q_\phi}(S_\phi)(\xi)$.
- other cases are either similar or obvious

□

Proof of Proposition 4 The proof goes by induction on h :

- if $h = 0$, then $r = 0$;
- if $h = h_1|h_2$, then $r = r_1|r_2$ where r_1 and r_2 are obtained by induction hypothesis on h_1 and h_2 respectively;
- if $h = a(h')$, by induction hypothesis, there exists a unique closed run $r' \in R_{A_\phi}(h')$. We define $q_d \in \prod_{\xi \in \text{DVars}(S_\phi)} \text{split}(\xi)$ as follows: for all equations $(\xi = \alpha[\xi']) \in S_\phi$ – there exists at least one such equation by Lemma 7 – such that $\xi \in \text{DVars}(S_\phi)$, we define $q_d(\xi)$ as:

$$\begin{aligned} \alpha[\xi'] & \text{ if } a \in \alpha \text{ and } \text{roots}(r') \in \text{Sol}_{Q_\phi}(S_\phi)(\xi') \\ \alpha[\neg\xi'] & \text{ if } a \in \alpha \text{ and } \text{roots}(r') \notin \text{Sol}_{Q_\phi}(S_\phi)(\xi') \\ \bar{\alpha}[\top] & \text{ if } a \notin \alpha \end{aligned}$$

We let $p = (q_d, \emptyset, \emptyset)$. It is not difficult to see that $a \in \alpha_p$, and $\text{roots}(r') \in L_p$ (since $L_{nd,p} = Q_\phi^*$).

It remains to show that $p[r']$ is unique. Suppose that there exists $p' = (q'_d, \emptyset, \emptyset) \in Q_\phi$ such that $a \in \alpha_{p'}$ and $\text{roots}(r') \in L_{p'}$. Let $(\xi = \alpha[\xi']) \in S_\phi$. There are several cases to consider, depending on whether $a \in \alpha$ or not, and $\text{roots}(r') \in \text{Sol}_{Q_\phi}(S_\phi)(\xi')$ or not. In all cases, it is easy to show that $q_d(\xi) = q'_d(\xi)$, which conclude the proof. \square .

C Extending MSO

Proof (Proof of Theorem 7). We adapt the proof of emptiness undecidability of automata with equality constraints of [16]. In this proof, it is shown that one can construct an automata with equality constraints which can recognize the solution of an instance of the Post Correspondence Problem (PCP).

Let Σ be a finite alphabet, and $u_1, \dots, u_m, v_1, \dots, v_m$ be an instance of PCP, $\forall i \in \{1, \dots, m\}, u_i, v_i \in \Sigma^*$. We denote by $\Sigma \cup \{f, c\}$ the ranked alphabet obtained by extending Σ with a fresh ternary function symbol f and a constant 0 . Symbols from Σ are viewed as unary function.

Let $(u_{i_1}, v_{i_1}), \dots, (u_{i_n}, v_{i_n})$ be a solution of PCP. This solution can be represented as a term over $\Sigma \cup \{f, c\}$, as in Figure 11. For all $1 \leq j \leq m$, the notation $u_j(x)$ stands for the context $u_{j,1}(u_{j,2}(\dots u_{j,k}(x) \dots))$, where $u_{j,1}, \dots, u_{j,k}$ are symbols from Σ and $u_j = u_{j,1} \dots u_{j,k}$.

Informally, we have to check with an MSO_{\sim} whether the tree has the shape of Figure 11. This can easily be done by a sentence ϕ_{shape} . Necessarily, if some tree t satisfies ϕ_{shape} , then all its leaves are labeled c , the ternary nodes are labeled f , and the unary node labeled Σ .

Then, for each pattern $f(x_1, f(x_2, x_3, x_4), x_5)$ occurring in the tree, we have to verify that x_1 is of the form $u_i(x_2)$ and x_5 is of the form $v_i(x_4)$ for some i . It is easy to write an MSO formula $\phi_{pattern}(x_1, x_2, x_3, x_4, x_5)$ which checks whether the nodes bounded to x_1, \dots, x_5 form a pattern $f(x_1, f(x_2, x_3, x_4), x_5)$ in the tree. For every $i \in \{1, \dots, m\}$, we define $\phi_{u_i}(x, y)$ as a binary formula which checks whether $x < y$ and the sequence of labels along the path from x down to the parent of y is u_i . We define $\phi_{v_i}(x, y)$ similarly.

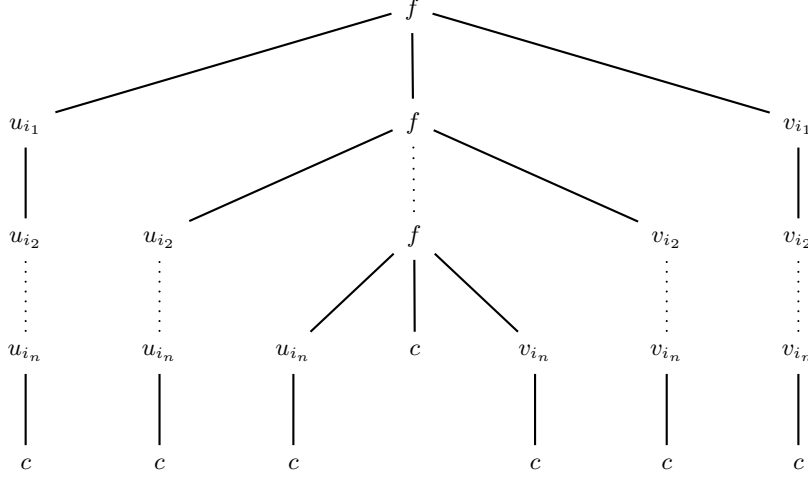


Fig. 11. representation of a solution of PCP

Finally, let x_r denotes the root of the tree, we define ϕ_{init} as the sentence which checks whether the tree rooted at the first child of x_r is isomorphic to the tree rooted at the third child of x_r .

Now, the following sentence accepts the tree representations of the solutions of the PCP instance:

$$\begin{aligned} & \phi_{shape} \wedge \phi_{init} \wedge \\ & \forall x_1 \dots \forall x_5 \exists y \exists y', \neg \phi_{pattern}(x_1, \dots, x_5) \vee \\ & (y \sim x_2 \wedge y' \sim x_4 \wedge \bigvee_{i=1}^m \phi_{u_i}(x_1, y) \wedge \phi_{v_i}(x_5, y')) \end{aligned}$$

Proposition 5. For any formula φ in MSO_{\sim}^{\exists} , one can compute a bounded TAGED, whose size is non-elementary in the size of φ , accepting the models of φ . Conversely, for any bounded TAGED A , one can compute a formula φ in MSO_{\sim}^{\exists} whose models are the trees accepted by A . The size of φ is doubly exponential in the size of A .

Proof. **Forth direction**

We first show that bounded TAGED are closed under union. Let $A = (\Lambda, Q, F, \Delta, =_A, \neq_A)$ and $A' = (\Lambda, Q', F', \Delta', =_{A'}, \neq_{A'})$ be two TAGED, bounded respectively by k_A and $k_{A'}$. Wlog, we suppose that $Q \cap Q' = \emptyset$. Then the TAGED which recognizes $L(A) \cup L(A')$ is defined by $A \cup A' = (\Lambda, Q \cup Q', F \cup F', \Delta \cup \Delta', =_A \cup =_{A'}, \neq_A \cup \neq_{A'})$. Of course, $A \cup A'$ is bounded by $\max(k_A, k_{A'})$.

Let ϕ be an MSO_{\sim}^{\exists} formula. By definition of MSO_{\sim}^{\exists} , ϕ is equivalent to a disjunction of formulas of the form $\Phi = \exists \bar{x}, \psi(\bar{x}) \wedge \psi_{test}(\bar{x})$, for some tuple of variable \bar{x} . Formula ψ is an MSO-formula, and ψ_{test} is a conjunction of atoms $x_i \sim x_j$ or $\neg(x_i \sim x_j)$, where $x_i, x_j \in \bar{x}$. Since bounded TAGED are closed under disjunction, it is sufficient to show the proposition for every Φ .

We let \bar{x} being equal to x_1, \dots, x_n . We use the classical Thatcher and Wright's construction to transform $\psi(\bar{x})$ into a hedge automata $A = (A \times \{0, 1\}^n, Q, F, \Delta)$, such that the i -th component of the tuple of any label corresponds to the i -th variable in \bar{x} , namely x_i . Then, projecting A on its first component results in a hedge automata recognizing the models of $\exists \bar{x}, \psi(\bar{x})$. It is known that several regular languages over $A \times \{0, 1\}^n$ are equivalent when projected on their first component. In particular, we can suppose that A recognizes trees t 's over $A \times \{0, 1\}^n$ such that $\forall i \in \{1, \dots, n\}$, there is at most one node whose i -th Boolean component is set to 1 [20]. Those trees are called *tuple tree*.

Now, instead of projecting the automata as usual, we project the Booleans from the labels into states [21]. We denote by $\text{proj}(A) = (A, Q_{\text{proj}(A)}, F_{\text{proj}(A)}, \Delta_{\text{proj}(A)})$ the resulting automaton. It is defined by: $Q_{\text{proj}(A)} \subseteq Q \times \{0, 1\}^n$, $F_{\text{proj}(A)} = F \times \{0, 1\}^n$ and $\Delta_{\text{proj}(A)}$ is constructed as follows:

$$\text{if } (a, \bar{b})(L) \rightarrow q \in \Delta, \text{ then } a(L') \rightarrow (q, \bar{b}) \in \Delta_{\text{proj}(A)}$$

where L' is defined to be the regular language over $Q_{\text{proj}(A)}$ such that, for any $q_1, \dots, q_n \in Q$, and $\bar{b}_1, \dots, \bar{b}_n \in \{0, 1\}^n$, $(q_1, \bar{b}_1) \dots (q_n, \bar{b}_n) \in L'$ iff $q_1 \dots q_n \in L$.

Let $(q, \bar{b}) \in Q_{\text{proj}(A)}$, where $\bar{b} \in \{0, 1\}^n$ and \bar{b} contains at least one component set to 1. Since A recognizes tuple trees, (q, \bar{b}) occurs at most one time in each accepting run of $\text{proj}(A)$ (*).

Finally, we define two relations $(\mathcal{R})_E$ and $(\mathcal{R})_D$ on states of $\text{proj}(A)$. For any n -ary tuple of Booleans \bar{b} , we denote by $\bar{b}(i)$ its i -th projection. Now, $\forall (q_1, \bar{b}_1), (q_2, \bar{b}_2) \in Q_{\text{proj}(A)}$, we let:

- $(q_1, \bar{b}_1) (\mathcal{R})_E (q_2, \bar{b}_2)$ iff $\exists i, j \in \{1, \dots, n\}$ st $\bar{b}_1(i) = 1$ and $\bar{b}_2(j) = 1$ and $x_i \sim x_j$ is an atom of ϕ_{test} ;

- $(q_1, \bar{b}_1) (\mathcal{R})_D (q_2, \bar{b}_2)$ iff $(q_1, \bar{b}_1) \neq (q_2, \bar{b}_2)$ and $\exists i, j \in \{1, \dots, n\}$ st $\bar{b}_1(i) = 1$ and $\bar{b}_2(j) = 1$ and $\neg x_i \sim x_j$ is an atom of ϕ_{test} ;

We define $=_{\text{proj}(A)}$ by $(\mathcal{R})_E^*$, the transitive closure of $(\mathcal{R})_E$, and $\neq_{\text{proj}(A)}$ by $(\mathcal{R})_D$. Finally, the TAGED equivalent to Φ is defined by $\text{proj}(A)$ equipped with $=_{\text{proj}(A)}$ and $\neq_{\text{proj}(A)}$.

It is a bit tedious, but not difficult, to prove that $\forall h \in \mathbb{H}_A, t \in L(A)$ iff $t \models \Phi$. By the remark (*), one can see that $(\text{proj}(A), =_{\text{proj}(A)}, \neq_{\text{proj}(A)})$ is bounded by $|\bar{x}|$.

Complexity The classical Thatcher and Wright's construction which transform an MSO-formula into an equivalent tree automata is known to be non-elementary in time complexity (in the size of the formula). The size of the tree automata is also non-elementary in the size of the input formula. Since our TAGED construction relies on this construction, the output TAGED has also a non-elementary size in the size of the input MSO $_{\sim}$ -formula.

Back direction Conversely, let $A = (A, Q, F, \Delta, =_A, \neq_A, k)$ be a bounded TAGED. It is already known (see [10]) that the hedge automata (A, Q, F, Δ) is equivalent to an MSO-formula of the form:

$$\phi = \exists X_{q_1} \dots \exists X_{q_n} \phi_{\Delta}(X_{q_1}, \dots, X_{q_n})$$

where $\{q_1, \dots, q_n\} = Q$. Set variables X_{q_i} 's are intended to capture the set of nodes labeled by state q_i in a successful run of A . Formula $\phi_{\Delta}(X_{q_1}, \dots, X_{q_n})$ describes the behavior of the hedge automata, in terms of runs. We do not make this formula explicit and refer the reader to [10] for more details.

Now, we could add the constraints $\forall x \forall y (x \in X_{q_i} \wedge y \in X_{q_i}) \implies x \sim y$ but this would not result in an MSO^{\exists} formula. However, there are at most k nodes for which equality or disequality tests are performed, so that we can existentially quantify these nodes. We partition Q into sets S and $(Q \setminus S)$, where $S = E \cup D$.

In every successful run of A , the number of nodes labeled by S is bounded by k , so that a disjunction has to enumerate all the number of occurrences n_q of states from S , such that $\sum_{q \in S} n_q \leq k$.

States of S are denoted by $q_1, \dots, q_{|S|}$, and other states by $q_{|S|+1}, \dots, q_{|Q \setminus S|}$.

We define a formula denoted ϕ_A by:

$$\phi_A = \begin{cases} \exists X_{q_1} \dots \exists X_{q_{|Q \setminus S|}} \phi_{\Delta}(X_{q_1}, \dots, X_{q_{|Q \setminus S|}}) \wedge \\ \bigvee_{n_1 + \dots + n_{|S|} \leq k} \exists x_{1,1} \dots \exists x_{1,n_1} \dots \exists x_{|S|,1} \dots \exists x_{|S|,n_{|S|}} \\ \wedge \bigwedge_{i=1}^{|S|} X_{q_i} = \{x_{i,1}, \dots, x_{i,n_i}\} \\ \wedge \bigwedge_{q_i = A q_j} \bigwedge_{\ell \in \{1, \dots, n_i\}} \bigwedge_{\ell' \in \{1, \dots, n_j\}} x_{i,\ell} \sim x_{j,\ell'} \\ \wedge \bigwedge_{q_i \neq A q_j} \bigwedge_{\ell \in \{1, \dots, n_i\}} \bigwedge_{\ell' \in \{1, \dots, n_j\}} \neg x_{i,\ell} \sim x_{j,\ell'} \end{cases}$$

By construction, the models of ϕ_A are the trees of $L(A)$, and conversely.

While formula ϕ_{Δ} is polynomial in the size of A , the formula ϕ_A is exponential in k , hence doubly exponential is the size of A .

D Other omitted proofs

D.1 Proof of Theorem 1

The forth direction is a consequence of the construction given in the proof of Theorem 6. In this proof, the constructed TAGED A_{ϕ} satisfies $=_{A_{\phi}} \neq_{A_{\phi}} = \emptyset$, since ϕ is a sentence. Hence A_{ϕ} is nothing else than a classic hedge automaton, which is known to be MSO-definable.

Conversely, if S is defined by some MSO formula, it is well-known that S is also recognizable by a hedge automaton A . We can easily encode the behavior of A by a guarded TQL sentence. We give the construction for binary tree automaton, as it can be easily lift to the unranked case.

Let $A = (A, Q, F, \Delta)$ be a binary tree automaton. We define ϕ_A by $\phi_A = \bigvee_{q \in F} \phi_q^{\emptyset}$ where, for every $q \in Q$, and every set $P \subseteq Q$ (the environment), we define ϕ_q^P as:

$$\phi_q^P = \begin{cases} \xi_q & \text{if } q \in P \\ \mu \xi_q. (\bigvee_{\alpha(q_1, q_2) \rightarrow q \in \Delta} \alpha[\phi_{q_1}^{P \cup \{q\}} | \phi_{q_2}^{P \cup \{q_2\}}] \\ \vee \bigvee_{\alpha \rightarrow q \in \Delta} \alpha[0]) & \text{otherwise} \end{cases}$$

This definition is recursive but terminating since the environment increases a finite number of times.

E Additional References

References

- [20] Aurélien Lemay, Joachim Niehren and Rémi Gilleron, Learning n-ary Node Selecting Tree Transducers from Completely Annotated Examples *International Colloquium on Grammatical Inference, Springer, 2006*
- [21] Joachim Niehren, Laurent Planque, Jean-Marc Talbot and Sophie Tison. N-ary Queries by Tree Automata *10th International Symposium on Database Programming Languages, Springer, 2005*