Satisfiability of a Spatial Logic with Tree Variables

Emmanuel Filiot, Jean-Marc Talbot, Sophie Tison

Satisfiability of a Spatial Logic with Tree Variables

Emmanuel Filiot[1] Jean-Marc Talbot[2] Sophie Tison[1]

[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 Λ.

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

$$\mathtt{0}|h = h \qquad h|\mathtt{0} = h \qquad (h|h'')|h''' = h|(h''|h''')$$

$\mathtt{0}$ will be the *empty hedge*. We call respectively *trees* and *leaves* hedges of the form $a(h)$ and $a(\mathtt{0})$. We may omit $|$ and write $a(h)b(h')c(h'')$ instead of $a(h)|b(h')|c(h'')$. We define roots(h) as the word from Λ^* defined recursively as : (i) roots$(\mathtt{0}) = \epsilon$, (ii) roots$(a(h')) = a$ and, (iii) roots$(h_1|h_2) =$ roots(h_1)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 $\mathtt{nodes}(h)$ the set V and by $\mathtt{lab}_h(u)$ 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 $\alpha(L) \to q$ 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:*

$$R_A(h_1|h_2) = \{r_1|r_2 \mid r_1 \in R_A(h_1), r_2 \in R_A(h_2)\} \qquad R_A(\mathtt{0}) = \{\mathtt{0}\}$$
$$R_A(a(h)) = \{q(r) \mid \exists \alpha(L) \to q \in \Delta, a \in \alpha, r \in R_A(h), \mathit{roots}(r) \in L\}$$

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

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

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 $\{\mathsf{lab}_a \mid a \in \Lambda\}$ where lab_as are unary predicates. We associate with a hedge $h = (V, E_c, E_s, \lambda)$ a finite σ-structure $\mathcal{S}^h = \langle V, \{\mathsf{ch}, \mathsf{ns}\} \cup \{\mathsf{lab}_a^h \mid a \in \Lambda\}\rangle$, such that $\mathsf{lab}_a^h(v)$ (resp. $\mathsf{ch}(v, v')$, $\mathsf{ns}(v, v')$) 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 $x, y, z, \ldots$ and a countable set of second-order variables ranging over by $X, Y, Z, \ldots$.

MSO formulas are given by the following syntax:

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

$$
\begin{array}{lll}
\phi ::= & \mathtt{0} & \text{empty hedge} \\
& \top & \text{truth} \\
& \alpha[\phi] & \text{extension} \\
& \phi|\phi & \text{composition} \\
& \neg\phi & \text{negation} \\
& \phi \vee \phi & \text{disjunction} \\
& X & \text{tree variable} \\
& \xi & \text{recursion variables} \\
& \mu\xi.\phi & \text{least fixpoint} \\
& \phi^* & \text{iteration}
\end{array}
$$

(a) Syntax

$$
\begin{array}{ll}
[\![\mathtt{0}]\!]_{\rho,\delta} & = \{\mathtt{0}\} \\
[\![\top]\!]_{\rho,\delta} & = \mathbb{H}_\Lambda \\
[\![\alpha[\phi]]\!]_{\rho,\delta} & = \{a(h) \mid h \in [\![\phi]\!]_{\rho,\delta},\ a \in \alpha\} \\
[\![\phi|\phi']\!]_{\rho,\delta} & = \{h|h' \mid h \in [\![\phi]\!]_{\rho,\delta}, h' \in [\![\phi']\!]_{\rho,\delta}\} \\
[\![\neg\phi]\!]_{\rho,\delta} & = \mathbb{H}_\Lambda \backslash [\![\phi]\!]_{\rho,\delta} \\
[\![\phi \vee \phi']\!]_{\rho,\delta} & = [\![\phi]\!]_{\rho,\delta} \cup [\![\phi']\!]_{\rho,\delta} \\
[\![X]\!]_{\rho,\delta} & = \{\rho(X)\} \\
[\![\xi]\!]_{\rho,\delta} & = \delta(\xi) \\
[\![\mu\xi.\phi]\!]_{\rho,\delta} & = \bigcap\{S \subseteq \mathbb{H}_\Lambda \mid [\![\phi]\!]_{\rho,\delta[\xi \mapsto S]} \subseteq S\} \\
[\![\phi^*]\!]_{\rho,\delta} & = \mathtt{0} \cup \bigcup_{i>0} \underbrace{[\![\phi]\!]_{\rho,\delta}|\ldots|[\![\phi]\!]_{\rho,\delta}}_{i \text{ times}}
\end{array}
$$

(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) $\psi \models_\rho \mathsf{lab}_a(x)$ if $\mathsf{lab}_a(\rho(x))$ holds in $\mathcal{S}$, (ii) $\psi \models_\rho \mathsf{ch}(x,y)$ if $\mathsf{ch}(\rho(x),\rho(y))$ holds in $\mathcal{S}$, (iii) $\psi \models_\rho \mathsf{ns}(x,y)$ if $\mathsf{ns}(\rho(x),\rho(y))$ 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 $\Lambda[\mathtt{0}]$.
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 recvar(ϕ) (resp. var(ϕ)) 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 $\mathbb{T}_\Lambda$ 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 $[\![\phi]\!]_{\rho,\delta}$ is inductively defined and given on Figure 1(b).

Examples Let us consider the following formulas:

$$\begin{aligned}
\phi &= a[\top]|\top && (1)\\
\phi_s &= \mu\xi.(a[\top]|\xi \ \vee \ \mathtt{0}) && (2)\\
\phi_{dtd} &= (employee[name[\Lambda[0]] \,|\, dpt[\Lambda[0]] \,|\, manager[\Lambda[\mathtt{0}]]])^* && (3)\\
\phi_{dd} &= \phi_{dtd} \wedge \top \,|\, employee[X] \,|\, \top \,|employee[X] \,|\, \top && (4)\\
\phi_{path(a),\mathtt{0}} &= \mu\xi.((\top|a[\xi]|\top) \ \vee \ \mathtt{0}) && (5)
\end{aligned}$$

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 $\{a[h_1]|\ldots|a[h_n] \mid h_i \in \mathbb{H}_\Lambda, n \geq 0\}$. 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 $\phi_{path(a),\mathtt{0}}$ are hedges with a path labelled by as from one of the roots to some leaf (*i.e.* the empty hedge $\mathtt{0}$).

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

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

Let us denote $\phi_{path(L),\psi} = \mu\xi.((\top|L[\xi]|\top) \ \vee \ \psi)$ 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 as and by bs from two of the top-level nodes, those two paths leading to some identical subtrees

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

The formula $\phi_{id\_not\_key}$ 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")

$$\phi_{id\_not\_key} = \top|\Lambda[\phi_{path(\Lambda),id[X]}]|\top|\Lambda[\phi_{path(\Lambda),id[X]}]|\top$$

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

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

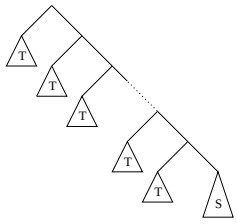

Fig. 2. A tree with $T \neq S$

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.

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

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

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

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 $h \in [\![\phi]\!]_\rho$.*

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 $[\![\phi]\!] = S$ 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:

$$[\![\exists X.\phi]\!]_{\rho,\delta} = \bigcup_{t\in\mathbb{T}} [\![\phi]\!]_{\rho[X\to t],\delta} \qquad [\![\forall X.\phi]\!]_{\rho,\delta} = \bigcap_{t\in\mathbb{T}} [\![\phi]\!]_{\rho[X\to t],\delta}$$

where $\rho[X \to t]$ 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 $\exists X_1 \ldots \exists X_n \phi$ 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:

$$\text{var}^*_\beta(\mathtt{0}) = \text{var}^*_\beta(\top) = \varnothing \quad \text{var}^*_\beta(a[\phi]) = \text{var}^*_\beta(\neg\phi) = \text{var}^*_\beta(\mu\xi.\phi) = \text{var}^*_\beta(\phi^*) = \text{var}^*_\beta(\phi)$$
$$\text{var}^*_\beta(\xi) = \text{var}^*_\beta(\phi_\xi) \qquad \text{var}^*_\beta(X) = \{X\} \quad \text{var}^*_\beta(\phi \vee \phi) = \text{var}^*_\beta(\phi|\phi') = \text{var}^*_\beta(\phi) \cup \text{var}^*_\beta(\phi')$$

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

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 $\neq_A = \varnothing$, 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: $\forall i \in \{1, \ldots, n\}$, $\forall u, v \in \mathtt{nodes}(h)$, $\mathtt{lab}_r(u) =_A \mathtt{lab}_r(v) \implies h|_u = h|_v$.*

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

The set of accepting runs of A on h, denoted $R_A^{acc}(h)$, 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 $R_A^{acc}(h) \neq \varnothing$.

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: $\Lambda(q^*) \to q \qquad \Lambda(q^*) \to q_X \qquad a(q_X q_X) \to q_f$ 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 $Q = \{q, q_X, q_{\overline{X}}, q_f\}$, $F = \{q_f\}$, and let Δ defined as the set of following rules:

$$\Lambda(q^*) \to q_{\overline{X}} \qquad \Lambda(q^*) \to q \qquad \Lambda(q^*) \to q_X \qquad a(q_X(q_{\overline{X}} + q_f)) \to q_f$$

Let A_2 be the TAGED $(\Lambda, Q, F, \Delta, \{q_X =_{A_2} q_X\}, \{q_X \neq_{A_2} q_{\overline{X}}\})$. 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: $|\{u \mid \mathtt{lab}_r(u) \in E \cup D\}| \leq k$.*
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 $L(A) = \varnothing$ 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 $[q] \cap P \neq \varnothing$ and P is a label of c.*

Nodes of c are denoted by $\mathtt{nodes}(c)$. For every node $u \in \mathtt{nodes}(c)$, we denote by $c|_u$ the subgraph of c induced by the nodes reachable from u in c, and by $u_1, \ldots, u_n$ the n successor nodes of u given in order. Note that it might be the case that $u_i = u_j$ for some $i, j \in \{1, \ldots, n\}$. Finally, we always denote by $\mathtt{lab}_c(u) \subseteq Q$ 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 $\{q, q'\}$.

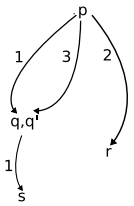

Fig. 3. A configuration (naturals represent the order on edges)

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

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

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 $\mathtt{nodes}(c)$ into contexts over Λ is an *interpretation* of c if for every node $u \in \mathtt{nodes}(c)$, if u has exactly n successor nodes $u_1, \ldots, u_n$ in c, then $\lambda(u)$ is an n-ary context. Moreover, λ must satisfy the following: for every node $u \in \mathtt{nodes}(c)$ and every nodes $u_1, \ldots, u_n \in \mathtt{nodes}(c)$, if $u_1, \ldots, u_n$ are the successor nodes of u then the following holds (called condition (P)):

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

As A is positive, the set D is empty, but we keep this definition for uniformity reasons when dealing with disequalities. Every node $u \in \mathtt{nodes}(c)$, together with the mapping λ, define a tree $t(u, \lambda)$ over Λ as follows: $t(u, \lambda) = \lambda(u)[t(u_1, \lambda), \ldots, t(u_n, \lambda)]$, where $n \in \mathbb{N}$ and $u_1, \ldots, u_n$ 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 $u_0 \in \mathtt{nodes}(c)$ be the root of c. For every $p \in \mathtt{lab}_c(u_0)$, 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 $t \in \mathbb{T}_\Lambda$ be a tree such that $t \in L(A)$. Then there is a configuration c of size at most $|Q|.k^{|Q|}$ 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 $\mathtt{nodes}(t)$. 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 $\mathtt{lab}_r(u) =_A \mathtt{lab}_r(v)$. Each equivalence class will represent a node of c, to enforce equalities. □

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 $\mathcal{D}(\mathcal{A})$ be the set of configurations of A whose sizes are bounded by $|Q|.k^{|Q|}$. The following holds:*

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

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

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

Proof (Sketch). For all nodes $u, u_1, \ldots, u_n$ s.t. $u_1, \ldots, u_n$ are the successors of u, it suffices to test whether there is an n-ary context C s.t. for all state $p \in \mathtt{lab}_c(u)$, there are $p_1 \in \mathtt{lab}_c(u_1)$,...,$p_n \in \mathtt{lab}_c(u_n)$ s.t. $C[p_1, \ldots, p_n] \rightarrow_{Q \setminus E} p$. We can represent the set of contexts C such that $C[p_1, \ldots, p_n] \rightarrow_{Q \setminus E} p$ by a tree automaton $A_{(p_i)_i, p}$. Then, it suffices to test emptiness of $\bigcap_{p \in P} \bigcup_{(p_i)_i \in \prod_i \mathtt{lab}_c(u_i)} L(A_{(p_i)_i, p})$, 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 $\mathcal{D}(\mathcal{A})$ still denotes the set of configurations of A whose size is bounded by $|Q|.k^{|Q|}$. We have the following inclusion: $L(A) \subseteq \bigcup_{c \in \mathcal{D}(\mathcal{A})} L(A, c)$, 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 $u, v \in \mathtt{nodes}(c)$, if there are $p \in \mathtt{lab}_c(u)$ and $q \in \mathtt{lab}_c(v)$ 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 $u \in \mathtt{nodes}(c)$, we let $\mathrm{cxt}_c(u)$ 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 $\mathtt{nodes}(c)$ into contexts, such that it maps every node u such that $\mathrm{cxt}_c(u)$ is finite into a context of $\mathrm{cxt}_c(u)$, and every other node v to a context $@_v(\bullet, \ldots, \bullet)$ 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 $\Lambda \cup \{@_v \mid v \in \mathtt{nodes}(c)\}$. 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 $L(A) \neq \varnothing$ iff there exist a configuration $c \in \mathcal{D}(\mathcal{A})$ 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 $X_1, \ldots, X_n$. We sketch the construction of a TAGED A_ϕ such that for all hedges $h \in \mathbb{H}_\Lambda$, we have $h \in L(A_\phi)$ iff there exists a valuation $\rho : \text{var}(\phi) \to \mathbb{T}_\Lambda$ such that $h \in [\![\phi]\!]_\rho$. 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 $\phi_1||\phi_2$ as a shortcut for $\neg(\neg\phi_1|\neg\phi_2)$, $\phi_1^\diamond$ 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 $\xi_1 = \mathsf{rhs}_1, \ldots, \xi_n = \mathsf{rhs}_n$ where every rhs_i has one of the following form:

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

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

The solution of Σ over $(2^{\mathbb{H}_\Lambda}, \bigcup, \bigcap)$ modulo ρ is a mapping from fixpoint variables of Σ into $2^{\mathbb{H}_\Lambda}$, 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 var(S_ϕ) = var(ϕ) and the following holds:

Lemma 3. *For all valuations* $\rho : \text{var}(S_\phi) \to \mathbb{T}_\Lambda$, *we have* $[\![\phi]\!]_\rho = \text{Sol}_\mathbb{H}(S_\phi, \rho)(\text{last}(S_\phi))$

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 $S = \{\xi = \xi_1 \vee \xi_2; \xi_1 = a[\xi]; \xi_2 = \mathtt{0}\}$, where the last variable is ξ. Solutions of S are chains labelled by as. To check whether hedge $a(0)$ is a solution of ξ, first verify that $a(\mathtt{0})$ is a solution of ξ_1 or a solution of ξ_2. One can easily see that $a(\mathtt{0})$ is not a solution of ξ_2. It remains to verify whether $a(\mathtt{0})$ is a solution of $\xi_1 = a[\xi]$, by verifying that $\mathtt{0}$ is a solution of ξ, etc... This can be done by an automaton with transition rules $a(\epsilon + q_{a[\xi]}) \to q_{a[\xi]}$, 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 $\cup$, $\mathtt{0}$ 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 $a(s(\xi)) \to q_{a[\xi]}$.

Things get more complicated when adding intersection. For instance, consider the system $S' = \{\xi_1 = a[\xi]; \xi_2 = a[\xi']; \xi' = \mathtt{0}; \xi = \xi_1 \wedge \xi_2\}$. If we interpret this system as before, with states $q_{a[\xi]}$ and $q_{a[\xi']}$, we would get $s(\xi) = \varnothing$. 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 $\alpha[\xi]$, $\overline{\alpha}[\top]$, or $\alpha[\neg\xi]$, for every right-hand side of the form $\alpha[\xi]$ occuring 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 $\overline{\alpha}[\top]$ or $\alpha[\neg\xi]$, 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 $S = \{\xi'' = X; \xi' = \xi''|\xi''; \xi_X = a[\xi']; \xi_2 = \mathtt{0}; \xi_3 = \xi_\top^*; \xi_1 = \Lambda[\xi_3]; \xi_\top = \xi_1 \vee \xi_2; \xi = \xi_\top \wedge \xi_X\}$, where the last variable is ξ. Given a valuation $\rho : \mathrm{var}(S) \to \mathbb{T}_\Lambda$, 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 $\xi_\top$, ξ_1, ξ_2 and ξ_3 in our example; and non-deterministic one: ξ,ξ_X,ξ' and ξ''.

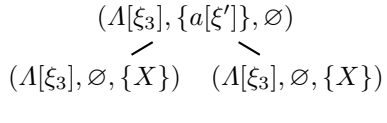

Fig. 4. Run of A_ϕ on $a(a(\mathtt{0})a(\mathtt{0}))$

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 $\overline{X}$, 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 $a(a(\mathtt{0})a(\mathtt{0}))$, is represented in Fig. 4. The state $(\Lambda[\xi_3], \varnothing, \{X\})$ 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 $\rho : \text{var}(\phi) \to \mathbb{T}_\Lambda$ such that $h \in [\![\phi]\!]_\rho$.*

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 $MSO^\exists_\sim$ for which formulas satisfy $Q_1 = Q_2 = \ldots = Q_n = \exists$. Remark that ψ can again contain quantifiers. Hence, since tree isomorphism cannot be expressed in MSO [10], $MSO_\sim$ and $MSO^\exists_\sim$ are stricly more expressive than MSO. By reduction of the Post correspondence problem, we can prove that:

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

However, $MSO^\exists_\sim$ and bounded TAGED are equally expressive: for any formula φ in $MSO^\exists_\sim$, 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 $MSO^\exists_\sim$.*

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 $MSO^{\exists}_{\sim}$ 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.