

# Tree Automata with Global Constraints

Emmanuel Filiot<sup>1</sup> Jean-Marc Talbot<sup>2</sup> Sophie Tison<sup>1</sup>

<sup>1</sup> INRIA Lille - Nord Europe, Mostrare Project, University of Lille 1 (LIFL, UMR 8022 of CNRS)

<sup>2</sup> University of Provence (LIF, UMR 6166 of CNRS), Marseille

**Abstract.** A tree automaton with global tree equality and disequality constraints, TAGED for short, is an automaton on trees which allows to test (dis)equalities between subtrees which may be arbitrarily faraway. In particular, it is equipped with an (dis)equality relation on states, so that whenever two subtrees  $t$  and  $t'$  evaluate (in an accepting run) to two states which are in the (dis)equality relation, they must be (dis)equal. We study several properties of TAGEDs, and prove decidability of emptiness of several classes. We give two applications of TAGEDs: decidability of an extension of Monadic Second Order Logic with tree isomorphism tests and of unification with membership constraints. These results significantly improve the results of [10].

## 1 Introduction

The emergence of XML has strengthened the interest in tree automata, as it is a clean and powerful model for XML tree acceptors [20, 21]. In this context, tree automata have been used, for example, to define schemas, and queries, but also to decide tree logics, to type XML transformations, and even to learn queries. However, it is known that sometimes, expressiveness of tree automata is not sufficient. This is the case for instance in the context of non-linear rewriting systems, for which more powerful tree acceptors are needed to decide interesting properties of those rewrite systems. For example, the set of ground instances of  $f(x, x)$  is not regular.

Tree automata with constraints have been introduced to overcome this lack of expressiveness [3, 9, 13, 14]. In particular, the transitions of these tree automata are fired as soon as the subtrees of the current tree satisfy some structural (dis)equality. But typically, these constraints are kept local to preserve decidability of emptiness and good closure properties – in particular, tests are performed between siblings or cousins –. In the context of XML, and especially to define tree patterns, one need global constraints. For instance, it might be useful to represent the set of ground instances of the pattern  $X(\mathbf{author}(x), \mathbf{author}(x))$ , where  $X$  is a binary context variable, and  $x$  is an author which occurs at least twice (we assume this pattern to be matched against XML trees representing a bibliography). In this example, the two subtrees corresponding to the author might be arbitrarily faraway, making the tree equality tests more global. Patterns might be more complex, by the use of negation (which allow to test tree disequalities),

Boolean operations, and regular constraints on variables. In [10], we study the spatial logic TQL, which in particular, allows to define such patterns. We proved decidability of a powerful fragment of this logic, by reduction to emptiness test of a new class of tree automata, called *Tree Automata with Global Equalities and Disequalities* (TAGEDs for short). These are tree automata  $A$  equipped with an equality  $=_A$  and a disequality  $\neq_A$  relation on (a subset of) states. A tree  $t$  is accepted by  $A$  if there is a computation of  $A$  on  $t$  which leads to an accepting state, and whenever two subtrees of  $t$  evaluate to two states  $q_1, q_2$  such that  $q_1 =_A q_2$  (resp.  $q_1 \neq_A q_2$ ), then these two subtrees must be structurally equal (resp. structurally different). TAGEDs may be interesting on their own, since they are somehow orthogonal to usual automata with constraints [3]. Indeed, if we view equality tests during computations as an equivalence relation on a subset of nodes (two nodes being equivalent if their rooted subtrees are successfully tested to be equal), in the former, there are a bounded number of equivalence classes of unbounded cardinality, while in the latter, there might be an unbounded number of equivalence classes of bounded cardinality.

The main result of [10] was decidability of emptiness of a subclass of TAGEDs, called bounded TAGEDs, which allow only a bounded number (by some constant independent of the tree) of (dis)equality tests on the run. In this paper, we prove several properties of TAGED-definable languages (closure by union and intersection, non-closure by complement). We prove results on TAGEDs (non-determinization, undecidability of universality). The other main results are decidability of emptiness of several classes of TAGEDs which significantly improves the result of [10], and uses different and simpler techniques. In particular, we prove a pumping lemma for TAGEDs which performs a bounded number of disequality tests along paths (and arbitrarily many equality tests).

We give two applications of TAGEDs. The first is decidability of an extension of MSO with tree isomorphism tests. The second application concerns a first-order disunification problems, with (regular) membership constraints. Dealing with membership constraints has been done in several papers. In [8], the authors prove solvability of first-order formulas whose atoms are either equations between terms or membership constraints  $t \in L$  where  $L$  is a regular tree language. In [16], the authors propose an algorithm to solve iterated matching of hedges against terms with flexible arity symbols, one-hole context and sequence variables constrained to range over a regular language. In this paper, we extend the logic of [8] with context variables (with arbitrarily many holes, and membership constraints) to allow arbitrary depth matching. Context unification is still an open problem, but motivated by XML tasks, we do not need to do full context unification. We prefer to impose a strong linearity condition on context variables. We prove that, even with this restriction, solvability of first-order formulas over these atoms is undecidable. We introduce an existential fragment for which satisfiability is decidable by reduction to emptiness of a class of TAGEDs.

*Related Work* Extensions of tree automata which allow for syntactic equality and disequality tests between subterms have already been defined by adding constraints to the rules of automata. E.g., adding the constraint  $1.2 = 2$  to

a rule means that one can apply the rule at position  $\pi$  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 [18] but two classes with a decidable emptiness problem have been emphasized. In the first class, automata are deterministic and the number of equality tests along a path is bounded [9] whereas the second restricts tests to sibling subterms [3]. This latter class has recently been extended to unranked trees [14], the former one has been extended to equality modulo equational theories [13]. But, contrarily to TAGEDs, in all these classes, tests are performed locally, typically between sibling or cousin subterms. Automata with local and global equality tests, using one memory, have been considered in [6]. Their emptiness problem is decidable, and they can simulate positive TAGEDs (TAGEDs performing only equality tests) which use at most one state per runs to test equalities. Finally, automata for DAGs are studied in [2,4], they cannot be compared to positive TAGEDs, as they run on DAG representations of trees (with maximal sharing), and in TAGEDs, we cannot impose every equal subtrees to evaluate in the same state in a successful run.

We only sketch the proofs, but all the missing proofs are in the full paper version [24] and in the thesis [11].

## 2 Trees and TAGED

*Binary trees* We start from a ranked alphabet  $\Sigma$  ranged over binary symbols  $f$  and constant symbols  $a$ . A *binary tree*  $t$  is a ground term over  $\Sigma$ . The set of binary trees over  $\Sigma$  is denoted by  $T_\Sigma$ . The set of *nodes* of a tree  $t \in T_\Sigma$ , denoted by  $N_t$ , is defined inductively as a set of words over  $\{1, 2\}$  by:  $N_a = \{\epsilon\}$  and  $N_{f(t_1, t_2)} = \{\epsilon\} \cup \{\alpha.u \mid \alpha \in \{1, 2\}, u \in N_{t_\alpha}\}$  ( $\epsilon$  denotes the empty word and  $.$  the concatenation). For any tree  $t$ , and any node  $u \in N_t$ , we define the subtree at node  $u$ , denoted by  $t|_u$ , inductively by:  $t|_\epsilon = t$ ,  $f(t_1, t_2)|_{\alpha.u} = t_\alpha|_u$ ,  $\alpha \in \{1, 2\}$ . Note that we have  $N_{t|_u} = \{v \mid u.v \in N_t\}$ . We also denote by  $O_t(u) \in \Sigma$  the label of node  $u$  in  $t$ . Finally, we denote by  $\triangleleft$  the strict descendant relation between nodes. Hence, for all  $u, v \in N_t$ ,  $u \triangleleft v$  if  $u$  is a prefix of  $v$  (therefore the root is minimal for  $\triangleleft$ ).

*Tree Automata* We define tree automata on binary trees, but the reader may refer to [7] for more details. A *tree automaton* is a 4-tuple  $A = (\Sigma, Q, F, \Delta)$  where  $Q$  is a finite set of states,  $F \subseteq Q$  is a set of final states, and  $\Delta$  is a set of rules of the form  $a \rightarrow q$  and  $f(q_1, q_2) \rightarrow q$ , where  $f$  is a binary function symbol,  $a$  a constant, and  $q_1, q_2, q$  are states from  $Q$ . A *run* of  $A$  on a tree  $t$  is a tree  $r$  over  $Q$  such that: (i)  $N_r = N_t$ , (ii) for all leaves  $u \in N_r$ , we have  $O_t(u) \rightarrow O_r(u) \in \Delta$ , and (iii) for all inner-nodes  $u \in N_r$ , we have  $O_t(u)(O_r(u.1), O_r(u.2)) \rightarrow O_r(u) \in \Delta$ . A run  $r$  is *successful* if  $O_r(\epsilon) \in F$ . The *language recognized* (or defined) by  $A$ , denoted  $L(A)$ , is the set of trees  $t$  for which there exists a successful run of  $A$ .

We consider binary and constant symbols only, but the two definitions above can be easily extended to symbols of other arity (in particular, we use unary symbols in several proofs and examples).

*Example 1.* Let  $\Sigma_b$  be the alphabet consisting of the two binary symbols  $\wedge, \vee$ , the unary symbol  $\neg$ , and the two constant symbols  $0, 1$ . Trees from  $T_{\Sigma_b}$  represents Boolean formulas. We define an automaton on  $\Sigma_b$  which accepts only Boolean formulas logically equivalent to 1. Its set of states (resp. final states) is defined by  $Q_b = \{q_0, q_1\}$  (resp.  $F_b = \{q_1\}$ ), and its set of rules  $\Delta_b$  by, for all  $b, b_1, b_2 \in \{0, 1\}$ , and all  $\oplus \in \{\wedge, \vee\}$ :

$$b \rightarrow q_b \quad \neg(q_b) \rightarrow q_{-b} \quad \oplus(q_{b_1}, q_{b_2}) \rightarrow q_{b_1 \oplus b_2} .$$

**Definition 1 (TAGED).** A TAGED is a 6-tuple  $A = (\Sigma, Q, F, \Delta, =_A, \neq_A)$  such that:

- $(\Sigma, Q, F, \Delta)$  is a tree automaton;
- $=_A$  is a reflexive and symmetric binary relation on a subset of  $Q$ ;
- $\neq_A$  is an irreflexive and symmetric binary relation on  $Q$ .

A TAGED  $A$  is said to be positive (resp. negative) if  $\neq_A$  (resp.  $=_A$ ) is empty.

The notion of successful run differs from tree automata as we add equality and disequality constraints. A run  $r$  of the tree automaton  $(\Sigma, Q, F, \Delta)$  on a tree  $t$  satisfies the equality constraints if  $\forall u, v \in N_t, O_r(u) =_A O_r(v) \Rightarrow t|_u = t|_v$ . Similarly, it satisfies the disequality constraints if  $\forall u, v \in N_t, O_r(u) \neq_A O_r(v) \Rightarrow t|_u \neq t|_v$ .

A run is *successful* (or *accepting*) if it is successful for the tree automaton  $(\Sigma, Q, F, \Delta)$  and if it satisfies the constraints. The language accepted by  $A$ , denoted  $L(A)$ , is the set of trees  $t$  having a successful run for  $A$ . We denote by  $\text{dom}(=_A)$  the domain of  $=_A$ , i.e.  $\{q \mid \exists q' \in Q, q =_A q'\}$ . The set  $\text{dom}(\neq_A)$  is defined similarly. Finally, two TAGEDs are *equivalent* if they accept the same language.

In [10], we introduced the class of bounded TAGED, where in successful runs, the number of occurrences of states from  $\text{dom}(=_A) \cup \text{dom}(\neq_A)$  is bounded by some fixed  $k \in \mathbb{N}$ . We proved emptiness to be decidable for that class. The classes we consider in this paper are either incomparable or strictly more expressive. All the results from Section 3 also hold for bounded TAGED. Note also that TAGED are strictly more expressive than tree automata, as illustrated by the next example.

*Example 2.* Let  $Q = \{q, q_=:, q_f\}$ ,  $F = \{q_f\}$ , and let  $\Delta$  be defined as the set of following rules:  $a \rightarrow q$ ,  $a \rightarrow q_=:$ ,  $f(q, q) \rightarrow q$ ,  $f(q, q) \rightarrow q_=:$ ,  $f(q_=:, q_=:) \rightarrow q_f$ , for all  $a, f \in \Sigma$ . Let the positive TAGED  $A_1 = (\Sigma, Q, F, \Delta, \{q_=: =_{A_1} q_=: \})$ . Then  $L(A_1)$  is the set  $\{f(t, t) \mid f \in \Sigma, t \in T_\Sigma\}$ , which is known to be non regular [7].

*Example 3.* Let  $\mathcal{X}$  be a finite set of variables. We now define a TAGED  $A_{\text{sat}}$  which accepts tree representations of satisfiable Boolean formulas with free variables  $\mathcal{X}$ . We let  $A_b = (\Sigma_b, Q_b, F_b, \Delta_b)$  be the automaton defined in Example 1.

Every variable is viewed as a binary symbol, and we let  $\Sigma_{\mathcal{X}} = \Sigma_b \cup \mathcal{X}$ . Every Boolean formula is naturally viewed as a tree, except for variables  $x \in \mathcal{X}$  which are encoded as trees  $x(0,1)$  over  $\Sigma_{\mathcal{X}}$ . For instance, the formula  $(x \wedge y) \vee \neg x$  is encoded as the tree  $\vee(t_1, t_2)$ , where  $t_1 = \wedge(x(0,1), y(0,1))$  and  $t_2 = \neg(x(0,1))$ . Now, we let  $Q = Q_b \cup \{q_x \mid x \in \mathcal{X}\} \cup \{p_0, p_1\}$ , for two fresh states  $p_0, p_1$ , and  $F = F_b$ . The idea is to choose non-deterministically to evaluate the leaf 0 or 1 below  $x$  to  $q_x$ , but not both, for all  $x \in \mathcal{X}$ . This means that we affect 0 or 1 to a particular occurrence of  $x$ . Then, by imposing that every leaf evaluated to  $q_x$  are equal, we can ensure that we have chosen to same Boolean value for all occurrences of  $x$ , for all  $x \in \mathcal{X}$ . This can be done with the set of rules  $\Delta_b$  extended with the following rules, for all  $b \in \{0,1\}$  and all  $x \in \mathcal{X}$ :

$$b \rightarrow p_b \quad b \rightarrow q_x \quad x(p_0, q_x) \rightarrow q_1 \quad x(q_x, p_1) \rightarrow q_0.$$

Finally, for all  $x \in \mathcal{X}$ , we let  $q_x =_{A_{sat}} q_x$ .

The (*uniform*) *membership problem* is “given a TAGED  $A$ , given a tree  $t$ , does  $t$  belong to  $L(A)$ ?”. We can prove the following:

**Proposition 1.** *Membership is NP-complete for TAGED.*

*Proof.* Example 3 gives a polynomial reduction of SAT to membership of TAGEDs. To show it is in NP, it suffices to guess a labeling of the nodes of the tree by states, and then to verify that it is a run, and that equality and disequality constraints are satisfied. This can be done in linear time both in the size of the automaton and of the tree.  $\square$

### 3 Closure Properties of TAGEDs and Decision Problems

In this section, we prove closure properties of TAGED-definable languages.

**Proposition 2 (Closure by union and intersection).** *TAGED-definable languages are closed by union and intersection.*

*Proof.* Let  $A = (\Lambda, Q, F, \Delta, =_A, \neq_A)$  and  $A' = (\Lambda, Q', F', \Delta', =_{A'}, \neq_{A'})$  be two TAGEDs. Wlog, we suppose that  $Q \cap Q' = \emptyset$ . A TAGED accepting  $L(A) \cup L(A')$  is defined by  $A \cup A' = (\Lambda, Q \cup Q', F \cup F', \Delta \cup \Delta', =_A \cup =_{A'}, \neq_A \cup \neq_{A'})$ . For the closure by intersection, we use the usual product construction  $A \times A'$  [7], whose set of final states is  $F \times F'$ . State equality  $=_{A \times A'}$  is defined by  $\{((q, q'), (p, p')) \mid q =_A p \text{ or } q =_{A'} p\}$ , while  $\neq_{A \times A'}$  is defined by  $\{((q, q'), (p, p')) \mid q \neq_A p \text{ or } q \neq_{A'} p\}$ .  $\square$

Prop 2 also holds for the class of languages defined by positive or negative TAGEDs. A TAGED is *deterministic* if all rules have different left-hand sides (hence there is at most one run per tree). For a deterministic TAGED  $A$ , we can prove that one can compute a non-deterministic TAGED accepting the complement of  $L(A)$ : we have to check if the tree evaluates in a non-accepting state or in an accepting state but in this case we non-deterministically guess a position where a constraint is not satisfied. However:

**Proposition 3.** *TAGEDs are not determinizable.*

*Proof.* Let  $\Sigma = \{f, a\}$  an alphabet where  $f$  is binary and  $a$  constant. Consider the language  $L_0 = \{f(t, t) \mid t \in T_\Sigma\}$  of Example 2. It is obvious that  $L_0$  is definable by a non-deterministic (bounded) TAGED. Suppose that there is a deterministic TAGED  $A = (\Sigma, Q, F, \Delta, =_A, \neq_A)$  such that  $L(A) = L_0$ . Let  $t$  be a tree whose height is strictly greater than  $|Q|$ . Since  $f(t, t) \in L_0$ , there are a successful run  $q_f(r, r)$  of  $A$  on  $f(t, t)$  for some final state  $q_f$ , two nodes  $u, v$  and a state  $q \in Q$  such that  $t|_v$  is a strict subtree of  $t|_u$ , and  $O_r(u) = O_r(v) = q$ . Since  $f(t|_u, t|_u) \in L_0$ , and  $A$  is deterministic, there is a final state  $q'_f \in F$  and a rule  $f(q, q) \rightarrow q'_f \in \Delta$ . Hence  $q'_f(r|_u, r|_v)$  is a run of  $A$  on  $f(t|_u, t|_v)$ . Since  $q_f(r, r)$  satisfies the constraints,  $q'_f(r|_u, r|_v)$  also satisfies the constraints. Hence  $f(t|_u, t|_v) \in L(A)$ , which contradicts  $t|_u \neq t|_v$ .  $\square$

Proposition 3 is not surprising, since:

**Proposition 4.** *The class of TAGED-definable languages is not closed by complement.*

*Proof.* (Sketch) We exhibit a tree language whose complement is easily definable by a TAGED, but which is not TAGED-definable. This language is the union of sets  $T_n$ , for all  $n \in \mathbb{N}$ , where  $T_n = \{f(g(t, t), t') \mid t \in T_\Sigma, t' \in T_{n-1}\}$ , and  $T_0 = \{a\}$ . To check whether a tree is in  $T_n$ , a TAGED would have to perform  $n$  equality tests, for each subtree rooted by  $g$ . This would require  $n$  states. This is only an intuition. The proof is a bit more complicated as the TAGED could also perform inequality tests.  $\square$

We end up this section with an undecidability result:

**Proposition 5.** *Testing universality of TAGEDs is undecidable.*

*Proof.* (Sketch) We adapt the proof of [18] for undecidability of emptiness of classical tree automata with equality constraints. We start from an instance of the Post Correspondence Problem (PCP). We encode the set of solutions of PCP as a tree language whose complement is easily definable by a TAGED. Hence, the complement is universal iff PCP has no solution.  $\square$

Even if TAGEDs are not determinizable, we can assume that testing an equality between subtrees can be done using the same state, as stated by the following lemma:

**Lemma 1.** *Every TAGED  $A$  is equivalent to a TAGED  $A'$  (whose size might be exponential in the size of  $A$ ) such that  $=_{A'} \subseteq id_{Q_{A'}}$ , where  $id_{Q_{A'}}$  is the identity relation on  $Q_{A'}$ . Moreover,  $A'$  can be built in exponential time (and may have exponential size).*

*Proof.* (Sketch) Intuitively, we can view an accepting run  $r$  of  $A$  on a tree  $t$  as a DAG structure. Let  $U \subseteq N_t$  such that all subtrees  $t|_u$ ,  $u \in U$ , have been successfully tested equal by  $A$  in the run  $r$  (i.e.  $\forall u, v \in U, O_r(u) =_A O_r(v)$ ).

Let  $t_0 = t|_u$ , for some  $u \in U$ . We replace all nodes of  $U$  by a single node  $u_0$  which enroots  $t_0$ . The parent of any node of  $U$  points to  $u_0$ . We maximally iterate this construction to get the DAG. Note that this DAG is not maximal sharing<sup>1</sup>, since only subtrees which have been successfully tested to be equal are shared. We construct  $A'$  such that it simulates a run on this DAG, obtained by overlapping the runs on every equal subtrees for which a test has been done.  $\square$

## 4 Emptiness of Positive and Negative TAGEDs

In this section we prove decidability of emptiness of positive and negative TAGEDs respectively. For positive TAGEDs, it uses Lemma 1, and the classical reachability method for tree automata. For negative TAGEDs, we reduce the problem to testing satisfiability of set constraints.

**Theorem 1.** *Testing emptiness of positive TAGEDs is EXPTIME-complete.*

*Proof. upper bound* Let  $A$  be a positive TAGED such that its equality relation is a subset of the identity relation (otherwise we transform  $A$  modulo an exponential blow-up, thanks to Lemma 1). Let  $A^-$  be its associated tree automaton (i.e.  $A$  without the constraints). We have  $L(A) \subseteq L(A^-)$ .

Then it suffices to apply a slightly modified version of the classical reachability method used to test emptiness of a tree automaton [7]. In particular, we can make this procedure associate with any state  $q$  a unique tree  $t_q$ . When a new state is reached, it can possibly activate many rules  $f(q_1, q_2) \rightarrow q$  whose rhs are the same state  $q$ . The algorithm has to make a choice between this rules in order to associate a unique tree  $t_q = f(t_{q_1}, t_{q_2})$  to  $q$ . This choice can be done for instance by giving an identifier to each rule and choosing the rule with the least identifier.

If  $L(A^-)$  is empty, then  $L(A)$  is also empty. If  $L(A^-)$  is non-empty, we get a tree  $t$  and a run  $r$  which obviously satisfies the equality constraints, since a state  $q$  such that  $q =_A q$  is mapped to unique tree  $t_q$  (if  $q$  is reachable).

**lower bound** We reduce the problem of testing emptiness of the intersection of  $n$  tree automata  $A_1, \dots, A_n$  (see [7]), which is known to be EXPTIME-complete. We assume that their sets of states are pairwise disjoint ( $Q_i \cap Q_j = \emptyset$  whenever  $i \neq j$ ), and for all  $i = 1, \dots, n$ ,  $A_i$  has exactly one final state  $q_{f_i}$ , and  $q_{f_i}$  does not occur in lhs of rules of  $A_i$  (otherwise we slightly modify  $A_i$ , modulo a factor 2 in the size of  $A_i$ ). We let  $L = \{f(t_1, \dots, t_n) \mid f \in \Sigma, \forall i, t_i \in L(A_i), \forall i, j, t_i = t_j\}$ . It is clear that  $L$  is empty iff  $L(A_1) \cap \dots \cap L(A_n)$  is empty. It is not difficult to construct a TAGED  $A$  (with  $|A| = O(\sum_i |A_i|)$ ), such that  $L = L(A)$ : it suffices to take the union of  $A_1, \dots, A_n$  and to add the rule  $f(q_{f_1}, \dots, q_{f_n}) \rightarrow q_f$ , where  $q_f$  is a fresh final state of  $A$ . Then we add the following equality constraints:  $\forall i, j, q_{f_i} =_A q_{f_j}$ .  $\square$

If  $=_A \subseteq id_Q$ , in a successful run we can assume that the subruns rooted at states  $q$  such that  $q =_A q$  are the same. Hence, we can introduce a pumping technique

<sup>1</sup> there might be two isomorphic subgraphs occurring at different positions.

for positive TAGEDs satisfying this property. The idea is to pump similarly in parallel below all states  $q$  such that  $q =_A q$ , while keeping the equality constraints satisfied. The pumping technique is described in the full version of the paper [24]. Thanks to this, if there is a loop in a successful run, we can construct infinitely many accepted trees. In particular:

**Theorem 2.** *Let  $A$  be a positive TAGED. It is decidable whether  $L(A)$  is infinite or not, in  $O(|A||Q|^2)$  if  $=_A \subseteq id_Q$ , and in EXPTIME otherwise.*

We now prove decidability of emptiness of negative TAGEDs ( $=_A = \emptyset$ ), by reduction to positive and negative set constraints (PNSC for short). Set expressions are built from set variables, function symbols, and Boolean operations. Set constraints are either positive,  $e_1 \subseteq e_2$ , or negative,  $e_1 \not\subseteq e_2$ , where  $e_1, e_2$  are set expressions. Set expressions are interpreted in the Herbrand structure while set constraints are interpreted by Booleans 0,1. Testing the existence of a solution of a system of set constraints has been proved to be decidable in several papers [5, 1, 22, 12]. In particular, it is known to be NEXPTIME-complete. We do not formally define set constraints and refer the reader to [5, 1, 22, 12].

Consider for instance the constraint  $f(X, X) \subseteq X$ . It has a unique solution which is the empty set. Consider now  $X \subseteq f(X, X) \cup a$ , where  $a$  is a constant symbol. Every set of terms over  $\{f, a\}$  closed by the subterm relation is a solution. More generally, we can encode the emptiness problem of tree automata as a system of set constraints. Let  $A = (\Sigma, Q, F, \Delta)$  be a tree automaton. Wlog, we assume all state  $q \in Q$  to occur in the rhs of a rule. We associate with  $A$  the system  $S_A$  defined by:

$$(S_A) \quad \begin{cases} X_q \subseteq \bigcup_{f(q_1, q_2) \rightarrow q \in \Delta} f(X_{q_1}, X_{q_2}) \cup \bigcup_{a \rightarrow q \in \Delta} a & \text{for all } q \in Q \\ \bigcup_{q \in F} X_q \not\subseteq \emptyset \end{cases}$$

We can prove that  $L(A)$  is non-empty iff  $S_A$  has a solution. Let  $(A, \neq_A)$  be a negative TAGED, and consider the system  $S'_A$  consisting in  $S_A$  extended with the constraints  $X_q \cap X_p = \emptyset$ , for all  $q, p \in Q$  such that  $q \neq_A p$ . We can prove that  $L(A, \neq_A) \neq \emptyset$  iff  $S'_A$  has a solution. Since deciding existence of a solution of a system of PNSC is in NEXPTIME, we get:

**Theorem 3.** *Emptiness of negative TAGEDs is decidable in NEXPTIME.*

## 5 Emptiness when Mixing Equality and Disequality Constraints

In this section, we mix equality and disequality constraints. This has already been done in [10] for bounded TAGEDs. Emptiness was proved by decomposition of runs, but here we use a pumping technique that allows to decide emptiness for a class of TAGEDs that significantly extends the class considered in [10]. In particular, we allow an unbounded number of positive tests, but boundedly many negative tests along root-to-leaves paths, *i.e.* branches. While this class

subsumes positive TAGEDs, the upper-bound for testing emptiness is bigger than the bound obtained in Section 4.

Formally, a *vertically bounded TAGED* (vbTAGED for short) is a pair  $(A, k)$  where  $A$  is a TAGED, and  $k \in \mathbb{N}$ . A run  $r$  of  $(A, k)$  on a tree  $t \in T_\Sigma$  is a run of  $A$  on  $t$ . It is successful if  $r$  is successful for  $A$  and the number of states from  $\text{dom}(\neq_A)$  occurring along a root-to-leaves path is bounded by  $k$ : in other words, for all root-to-leaves path  $u_1 \triangleleft \dots \triangleleft u_n$  of  $t$  (where each  $u_i$  is a node), one has  $|\{u_i \mid O_r(u_i) \in \text{dom}(\neq_A)\}| \leq k$ .

We now come to the main result of the paper:

**Theorem 4.** *Emptiness of vbTAGEDs  $(A, k)$  is decidable in  $2\text{NEXPTIME}$ .*

*Proof. Sketch* We first transform  $A$  so that it satisfies  $=_A \subseteq id_Q$ , thanks to Lemma 1 (modulo an exponential blow-up). Let  $t \in T_\Sigma$ , and  $r$  a run of  $A$  on it which satisfies the equality constraints (but not necessarily the disequality constraints), and such that its root is labeled by a final state. We introduce sufficient conditions on  $t$  and  $r$  (which can be verified in polynomial-time, in  $|t|$ ,  $|r|$  and  $|A|$ ) to be able to repair the unsatisfied inequality constraints in  $t$  in finitely many rewriting steps. These rewritings can be done while keeping the equality constraints satisfied. In particular, since  $=_A \subseteq id_Q$ , we can assume that for all  $u, v \in N_t$  such that  $u \sim_{t,r} v$ ,  $r|_u = r|_v$ . Hence, we can use a “parallel” pumping technique similar to the pumping technique for positive TAGEDs. The pumping is a bit different however: indeed, if  $t$  and  $r$  satisfies the sufficient conditions, we increase the size of some contexts of  $t$  and  $r$ , called *elementary contexts*, in order to repair all the unsatisfied inequality constraints. The repairing process is inductive. In particular, we introduce a notion of frontier below which all inequality constraints have been repaired. The process stops when the frontier reach the top of the tree (and in this case the repaired tree is in the language). From a tree and a run that satisfy the sufficient conditions, and a frontier  $F$ , one can create a new tree and a new run satisfying the conditions, and a new frontier which is strictly contained in  $F$ . Conversely, if  $L(A, k) \neq \emptyset$ , then there is a tree  $t$  and a run  $r$  satisfying the conditions such that the height of  $t$  is smaller than  $2(k + |Q|)|Q|$  (and by  $(k + 2^{|Q|})2^{|Q|+1}$  if  $=_A \not\subseteq id_Q$ ). Hence, it suffices to guess a tree and a run satisfying the conditions to decide emptiness of  $A$ .

Since the class of vbTAGEDs subsumes the class of positive TAGEDs, we also get an EXPTIME lower bound for emptiness of vbTAGEDs, by Theorem 1. Moreover, if  $=_A \subseteq id_Q$  and  $k \leq |Q|$  (or  $k$  is unary encoded), emptiness of  $A$  is in NEXPTIME.  $\square$

## 6 Applications

### 6.1 MSO with Tree Isomorphism Tests

We study an extension of MSO with isomorphism tests between trees. Trees over an alphabet  $\Sigma$  are viewed as structures over the signature consisting of unary predicates  $O_a$ , for all  $a \in \Sigma$ , to test the labels, and the two successor relations  $S_1$

and  $S_2$  which relates the parent to its first child and its second child respectively. The domain of the structure is the set of nodes.

We consider node variables  $x, y$  and set variables  $X, Y$ . MSO consists of the closure of atomic formulas  $O_a(x)$  (for  $a \in \Sigma$ ),  $S_1(x, y)$ ,  $S_2(x, y)$ ,  $x \in X$ , by conjunction  $\wedge$ , negation  $\neg$ , and existential quantifications  $\exists x, \exists X$ . We refer the reader to [17] for the semantics of MSO. It is well-known that MSO sentences and tree automata define the same tree languages [23]. We use similar back and forth translations to prove that an extension of MSO with tree isomorphism tests effectively defines the same language as vertically bounded TAGED. This significantly improves the result of [10].

We consider a predicate  $\text{eq}(X)$ , which holds in a tree  $t$  under assignment  $\rho : X \mapsto U$  (denoted by  $t, \rho \models \text{eq}(X)$ ), for some  $U \subseteq N_t$ , if for all  $u, v \in U$ , the trees  $t|_u$  and  $t|_v$  are isomorphic. For all  $k \in \mathbb{N}$ , we consider the predicate  $\text{diff}_k(X, Y)$ , which holds in  $t$  under assignment  $\rho$  if (i) the maximal length of a descendant chain in  $\rho(X)$  and  $\rho(Y)$  is bounded by  $k$ , (ii) for all  $u \in \rho(X), v \in \rho(Y)$ , the trees  $t|_u$  and  $t|_v$  are **not** isomorphic. We consider  $\text{MSO}^{\exists}$  the extension of MSO whose formulas are of the form  $\exists X_1 \dots \exists X_n \phi$ , where  $\phi$  is an MSO formula extended with atoms  $\text{eq}(X_i)$  and  $\text{diff}_k(X_i, X_j)$  ( $1 \leq i, j \leq n$ )<sup>2</sup>.  $\text{MSO}^{\exists}$  is strictly more expressive than MSO as tree isomorphism is not expressible in MSO [7], but as a corollary of Theorem 4, we obtain:

**Theorem 5.**  *$\text{MSO}^{\exists}$  and  $\text{vbTAGEDs}$  effectively define the same tree languages, and satisfiability of  $\text{MSO}^{\exists}$  formulas is decidable.*

If we allow universal quantification of set variables  $X_1, \dots, X_n$ , the logic becomes undecidable (even if the  $X_i$ s denote singletons) [10].

## 6.2 Unification with Membership Constraints

We show that TAGEDs are particularly suitable to represent sets of ground instances of terms. Then we investigate a particular unification problem with tree and context variables where context variables can occur only in a restricted manner. In particular, we consider first-order logic (FO) over term equations  $t \approx t'$ , where  $t, t'$  are terms with tree and context variables, such that in a formula, every context variable can occur at most once. Tree and context variables might be constrained to range over regular languages (membership constraints). We prove this logic to be undecidable and exhibit a decidable existential fragment. This is particularly relevant for XML queries, as we can express tree patterns with negations. For instance, let  $L_{\text{DTD}}$  be a regular tree language representing the DTD of a bibliography,  $d$  a ground term representing a bibliography,  $L_{\text{path}}$  the set of unary contexts denoted by the XPath expression  $\text{bib/books}$  (i.e. contexts whose hole is reachable by the path  $\text{bib/books}$ ), and  $X$  a unary context variable. The formula  $\phi(y, z) = \exists X, d \approx X(\text{book}(\text{author}(y), \text{title}(z))) \wedge d \in L_{\text{DTD}} \wedge X \in L_{\text{path}}$  checks that  $d$  conforms to the DTD and extracts from  $d$  all (author,title) pairs reachable from the root by a path  $\text{bib/books/book}$ . The formula  $\exists z \exists z', \phi(y, z) \wedge$

<sup>2</sup> We assume that  $X_1, \dots, X_n$  are not quantified in  $\phi$

$\phi(y, z') \wedge \neg(z \approx z')$  extracts from  $d$  all authors  $y$  who published at least two books.

The restriction on context variables allows to test (dis)equalities arbitrarily deeply but can not be used to test context (dis)equalities. Even with this restriction, FO is undecidable, while it is known that without context variables, FO on atoms  $t \approx t'$  with membership constraints is decidable [8].

Let  $\Sigma$  be a ranked alphabet (assumed to be of binary and constant symbols for the sake of clarity). Let  $\mathcal{X}_t$  be a countable set of tree variables  $x, y$ , and  $\mathcal{X}_c$  a countable set of multi-ary context variables  $X, Y$  (we assume the existence of a mapping  $\text{ar} : \mathcal{X}_c \rightarrow \mathbb{N}$  giving the arity of any context variable). The set of terms over  $\Sigma$ ,  $\mathcal{X}_t$  and  $\mathcal{X}_c$  is denoted by  $\mathcal{T}(\Sigma, \mathcal{X}_t, \mathcal{X}_c)$ . For instance  $X(f(x, X(y), x))$  is a term where  $X \in \mathcal{X}_c$  (arity 1),  $f \in \Sigma$  (arity 3) and  $x, y \in \mathcal{X}_t$ . A term is *ground* if it does not contain variables. The set of ground terms over  $\Sigma$  is simply denoted  $T_\Sigma$ . We also denote by  $\mathcal{C}_\Sigma$  the set of contexts over  $\Sigma$ , and by  $\mathcal{C}_\Sigma^n$  the set of  $n$ -ary contexts over  $\Sigma$ , for all  $n \in \mathbb{N}$ . For all  $C \in \mathcal{C}_\Sigma^n$ , and terms  $t_1, \dots, t_n \in \mathcal{T}(\Sigma, \mathcal{X}_t, \mathcal{X}_c)$ , we denote by  $C[t_1, \dots, t_n]$  the term obtained by substituting the holes in  $C$  by  $t_1, \dots, t_n$  respectively (see [7] for a formal definition of contexts). A *ground substitution*  $\sigma$  is a function from  $\mathcal{X}_t \cup \mathcal{X}_c$  into  $T_\Sigma \cup \mathcal{C}_\Sigma$  such that for all  $x \in \mathcal{X}_t$ ,  $\sigma(x) \in T_\Sigma$ , and for all  $X \in \mathcal{X}_c$ ,  $\sigma(X) \in \mathcal{C}_\Sigma$  and  $\text{ar}(X) = \text{ar}(\sigma(X))$ . The ground term obtained by applying  $\sigma$  on a term  $t$  is denoted  $t\sigma$ . A ground term  $t'$  is a *ground instance* of a term  $t$  if there is  $\sigma$  such that  $t' = t\sigma$ . Finally, a term  $t$  is *context-linear* if every context variables occurs at most once in  $t$ .

**Proposition 6.** *Let  $t \in \mathcal{T}(\Sigma, \mathcal{X}_t, \mathcal{X}_c)$  be context-linear. The set of ground instances of  $t$  is definable by a positive TAGED.*

*Proof.* (Sketch) It suffices to introduce states for each subterm of  $t$ , and a special state  $q_\forall$  in which every ground term evaluates. Then we add state equalities  $q_x =_A q_x$  for all variable  $x$  occurring in  $t$ .  $\square$

We now introduce unification problems. An *equation*  $e$  is a pair of terms denoted by  $t \approx t'$ , where  $t, t' \in \mathcal{T}(\Sigma, \mathcal{X}_t, \mathcal{X}_c)$ . A ground substitution  $\sigma$  is a solution of  $e$  if  $t\sigma$  and  $t'\sigma$  are ground terms, and  $t\sigma = t'\sigma$ . Let  $n \in \mathbb{N}$ . A regular  $n$ -ary context language  $L$  is a regular language over  $\Sigma \cup \{\circ_1, \dots, \circ_n\}$ , where  $\circ_1, \dots, \circ_n$  are fresh symbols denoting the holes, and such that every symbol  $\circ_i$  occurs exactly once in terms (this can be ensured by a regular control). A *membership constraint* is an atom of the form  $x \in L_x$ , or  $X \in L_X$ , where  $x \in \mathcal{X}_t$ ,  $X \in \mathcal{X}_c$ ,  $L_x$  is a regular tree language, and  $L_X$  is a regular  $\text{ar}(X)$ -ary context language.

We consider FO over equations and membership constraint atoms, with the following restriction: for all formulas  $\phi$ , and all context variables  $X \in \mathcal{X}_c$ , there is at most one equation  $e$ , and one term  $t$  in  $e$  such that  $X$  occurs in  $t$ . We denote by FO[ $\approx, \in$ ] this logic. FO[ $\approx, \in$ ]-formulas are interpreted over ground substitutions  $\sigma$ . We define the semantics  $\sigma \models \phi$  inductively:  $\sigma \models e$  if  $\sigma$  is a solution of  $e$ ,  $\sigma \models x \in L_x$  if  $\sigma(x) \in L_x$  (and similarly for  $X \in L_X$ ),  $\sigma \models \exists x \phi$  if there is a ground term  $t$  such that  $\sigma[x \mapsto t] \models \phi$  (and similarly for  $\exists X \phi$ ). Disjunction and negation are interpreted as usual.

We can show the following by reducing PCP:

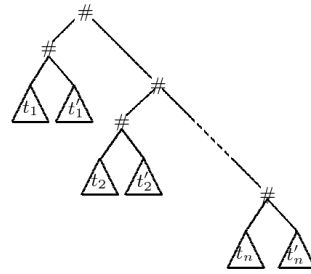
**Proposition 7.** *Satisfiability of  $FO[\approx, \in]$  is undecidable.*

However, it is known that satisfiability of  $FO[\approx, \in]$  in which no context variable occurs is decidable [8]. We consider the existential fragment  $FO^{\exists}[\approx, \in]$  of  $FO[\approx, \in]$  formulas where existential quantifiers  $\exists x$  or  $\exists X$  cannot occur below an odd number of negations.

**Theorem 6.** *Satisfiability of  $FO^{\exists}[\approx, \in]$  is decidable.*

*Proof.* (Sketch) Wlog, we consider only closed formulas. We define a normal form which intuitively can be viewed as a set of pairs  $(E, M)$ , where  $E$  is a set of equations  $e$  (or negated equations  $\neg e$ ), and  $M$  is a set of membership constraints. For each pair  $(E, M)$ , we construct a vbTAGED  $(A_{E,M}, |E|)$  which defines the ground instances of the term  $t_0$  depicted in Fig. 1 satisfying: (i)  $\#$  is a fresh symbol, (ii) for all terms  $t, t'$ , there exists  $i \in \{1, \dots, n\}$  s.t.  $t = t_i$  and  $t' = t'_i$  iff either  $(t \approx t') \in E$  or  $\neg(t \approx t') \in E$ , (iii) if  $t_0\sigma$  is a ground instance of  $t_0$ , then the membership constraints are satisfied, and  $\sigma$  is a solution of every equation of  $E$  (this can be done for instance by adding state inequalities  $q_t \neq_A q_{t'}$ , if  $\neg(t \approx t') \in E$ ). The formula is satisfiable iff there is a pair  $(E, M)$  such that  $L(A_{E,M}, |E|) \neq \emptyset$ .  $\square$

*Anti-pattern matching* [15] considers terms with negations (called anti-patterns). For instance, the anti-pattern  $f(x, \neg x)$  denotes all the ground terms  $f(t_1, t_2)$  such that  $t_1 \neq t_2$ . More generally, negations can occur at any position in the term:  $\neg(g(\neg a))$  denotes all ground terms which are not rooted by  $g$  or  $g(a)$ , and  $\neg f(x, x)$  denotes ground terms which are not of the form  $f(t, t)$ . A ground term matches an anti-pattern if it belongs to its denotation. [15] proves it to be decidable. We can easily define a vbTAGED  $A_p$  which accepts the denotation of an anti-pattern  $p$  where negations occur at variables only. Thus the anti-pattern matching problem reduces to test membership to  $L(A_p)$ . When negations occur arbitrarily, the translation is not so clear since the semantics of anti-patterns is universal (a ground term  $t$  matches  $\neg f(x, x)$  if  $\forall x, t \neq f(x, x)$ ). We let as future work this translation (for instance by pushing down the negations).



**Fig. 1.** term  $t_0$

## 7 Future Work

In [10], TAGEDs are based on hedge automata [19], so that they accept unranked trees. We can encode unranked trees over  $\Sigma$  as terms over the signature  $\Sigma \cup \{\text{cons}\}$ , where  $\text{cons}$  is a binary symbol denoting concatenation of an unranked tree to an hedge. For instance,  $f(a, b, c)$  maps to  $f(\text{cons}(a, \text{cons}(b, c)))$ . Hedge automata can be translated into tree automata over those encodings. Moreover, the encoding is unique, and any subtree  $t$  becomes a subtree rooted by a symbol of  $\Sigma$  in the encoding. Hence testing constraints between subtrees

in unranked trees is equivalent to test constraints between subtrees rooted by  $\Sigma$  in binary encodings. Therefore, TAGEDs over unranked trees can be translated into TAGEDs over encodings, and we can prove that all the results presented in this paper carry over to unranked trees. Moreover, in [10], we consider the TQL logic over unranked trees, and prove a fragment of it to be decidable, by reduction to emptiness of bounded TAGEDs (over unranked trees). This work could be used to decide larger fragments of TQL, via a binary encoding. Concerning the unification problem considered here, we would like to use TAGEDs to test whether there are finitely many solutions, and to represent the set of solutions. A question remains: deciding emptiness of full TAGEDs. It is not easy, even for languages of trees of the form  $f(t_1, t_2)$ , where  $t_1$  and  $t_2$  are unary. Finally, it could be interesting to consider more general tests, like recognizable relations on trees (since tree (dis)equality is a particular recognizable binary relation).

## References

1. Alexander Aiken, Dexter Kozen, and Edward L. Wimmers. Decidability of systems of set constraints with negative constraints. *Information and Computation*, 122(1):30–44, 1995.
2. Siva Anantharaman, Paliath Narendran, and Michael Rusinowitch. Closure properties and decision problems of dag automata. *Inf. Process. Lett.*, 94(5):231–240, 2005.
3. B. Bogaert and S. Tison. Equality and disequality constraints on direct subterms in tree automata. In *STACS'92*, volume 577 of *LNCS*, pages 161–171.
4. Witold Charatonik. Automata on dag representations of finite trees. Technical report, 1999.
5. Witold Charatonik and Leszek Pacholski. Set constraints with projections are in NEXPTIME. In *IEEE Symposium on Foundations of Computer Science*, 1994.
6. H. Comon and V. Cortier. Tree automata with one memory, set constraints and cryptographic protocols. *TCS*, 331(1):143–214, 2005.
7. H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available at <http://www.grappa.univ-lille3.fr/tata>, 2007.
8. Hubert Comon and Catherine Delor. Equational formulae with membership constraints. *Information and Computation*, 112(2):167–216, 1994.
9. M. Dauchet, A.-C. Caron, and J.-L. Coquidé. Reduction properties and automata with constraints. *JSC*, 20:215–233, 1995.
10. E. Filiot, J.-M. Talbot, and S. Tison. Satisfiability of a spatial logic with tree variables. In *CSL'07*, pages 130–145.
11. E. Filiot. *Logics for n-ary queries in trees*. PhD thesis, Lille University, 2008.
12. Rémi Gilleron, Sophie Tison, and Marc Tommasi. Some new decidability results on positive and negative set constraints. In *Proceedings of the International Conference on Constraints in Computational Logics*, pages 336–351, 1994.
13. F. Jacquemard, M. Rusinowitch, and L. Vigneron. Tree automata with equality constraints modulo equational theories. Research Report, LSV, ENS Cachan, 2006.
14. W. Kriantanto and C. Löding. Unranked tree automata with sibling equalities and disequalities. Research Report, RWTH Aachen, 2006.
15. Claude Kirchner, Radu Kopetz, and Pierre-Etienne Moreau. Anti-pattern matching. In *ESOP'07*, 2007.

16. Temur Kutsia and Mircea Marin. Solving regular constraints for hedges and contexts. In *UNIF'06*, pages 89–107.
17. L. Libkin. Logics over unranked trees: an overview. *LMCS'06*, 3(2):1–31, 2006.
18. J. Mongy. *Transformation de noyaux reconnaissables d'arbres. Forêts RATEG*. PhD thesis, Université de Lille, 1981.
19. M. Murata. Hedge automata: A formal model for xml schemata. Technical report, Fuji Xerox Information Systems, 1999.
20. Frank Neven. Automata, logic, and xml. In *CSL'02*, pages 2–26.
21. T. Schwentick. Automata for xml – a survey. *J. Comput. Syst. Sci.* 73, 3 (2007), 289–315.
22. Kjartan Stefansson. Systems of set constraints with negative constraints are nexttime-complete. In *LICS*, 1994.
23. J. W. Thatcher and J. B. Wright. Generalized finite automata with an application to a decision problem of second-order logic. *Mathematical System Theory*, 2:57–82, 1968.
24. Full paper version. Available at <http://hal.inria.fr/inria-00292027>.