On Nondeterministic Unranked Tree Automata with Sibling Constraints

Christof Löding and Karianto Wong[*]
RWTH Aachen University, Germany

ABSTRACT. We continue the study of bottom-up unranked tree automata with equality and disequality constraints between direct subtrees. In particular, we show that the emptiness problem for the nondeterministic automata is decidable. In addition, we show that the universality problem, in contrast, is undecidable.

1 Introduction

We continue the study of bottom-up unranked tree automata with equality and disequality constraints between direct subtrees, introduced in [12], which extend the corresponding automaton model known from the ranked setting [1]. This extension constitutes a part of the efforts in transferring the results known in the context of automata on ranked trees to the unranked case, which has attracted much attention from the research community as a formal model for XML documents; for references, see, e.g., the surveys [14, 15].

The distinguishing feature of unranked trees is that the number of children of the nodes, as opposed to ranked trees, is not a priori bounded by any fixed rank. In order to cope with this phenomenon, bottom-up automata on unranked trees, usually, incorporate regular languages in their transitions. It then turns out that finite automata on unranked trees enjoy many of the good properties of their counterpart in the ranked case. In many application domains, however, it is often desired to add some expressive power to the basic model without losing too many of the decidability results. A common approach to doing this is to add some constraints to the transitions of the automata. In fact, many models of finite automata with constraints on (ranked as well as unranked) trees have appeared in the literature.

An example of adding constraints in tree automata is counting constraints, which is particularly interesting in the unranked case as the number of successors of a node might be unbounded. For instance, in Presburger automata [17] (cf. also sheaves automata [7]) the application of a bottom-up transition is subject to the satisfaction of certain numerical conditions involving the subtrees (or the states reached at the root of these subtrees), such as “the number of a-rooted subtrees is twice the number of b-rooted subtrees”. It turns out that adding these constraints retains many of the good properties of the basic model; in particular, the emptiness problem remains decidable.

Another type of constraints that has been considered in the literature is the equality (and disequality) constraints. Here, the application of a bottom-up transition is subject to whether certain subtrees of the current node are equal. It turns out, however, that these constraints, in the most general form where comparisons between arbitrary subtrees are allowed, are too powerful, in the sense that the emptiness problem becomes undecidable [13].

[*] supported by DFG research grant *Algorithmische Theorie der Baumautomaten*

© Christof Löding, Karianto Wong; licensed under Creative Commons License-NC-ND.
Foundations of Software Technology and Theoretical Computer Science (Kanpur) 2009.
Editors: Ravi Kannan and K. Narayan Kumar; pp 311–322
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany.
Digital Object Identifier: 10.4230/LIPIcs.FSTTCS.2009.2328

In order to obtain decidability, thus, some restrictions on how equality constraints are used in ranked tree automata have been suggested. Such a restriction can be found in reduction automata [3, 8]; here, one requires that the number of equality and disequality tests in each path of a run tree must be bounded. Another possible restriction, suggested in [1], is that equality constraints may only be applied to sibling subtrees. For a more thorough overview of these automata models, the reader is referred to [4].

In [12], we extend tree automata with equality constraints between siblings to the unranked setting. In order to be able to address the (possibly) unbounded number of siblings to be compared, while still maintaining a finite representation, we suggest using formulas of monadic second-order logic. For this model, it has been shown that the emptiness problem for the deterministic case is decidable, while leaving open the nondeterministic case. It also turns out that the nondeterministic automata are strictly more expressive than the deterministic ones.

In this paper, we settle the nondeterministic case: we show that the emptiness problem, as in the deterministic case, is decidable. For this, despite the fact that determinization is not possible, we incorporate a kind of subset construction directly into our algorithm for the deterministic case, which then yields an emptiness algorithm for the nondeterministic case. In addition, we show that the universality problem in undecidable, which is achieved via a reduction of the halting problem for two-register machines.

There is a tight connection between our automaton model and data words that we want to point out. Generally speaking, a data word is a finite word to each position of which is attached a data value, i.e., a value from an infinite domain like the natural numbers. There are several automaton models on data words (and data trees) that have been proposed in the literature; for a recent survey, see, e.g., [16]. In order to maintain decidability results, these automaton models, usually, can only compare data values with respect to equality. Now, as trees can be used to represent data values, equality between data values amounts to equality between trees. Thus, with an appropriate encoding of data words as unranked trees, our automaton model can be used to describe languages of data words.

This paper is organized as follows. In Section 2, we fix our notations and recall our automaton model as well as some known results. Section 3 is devoted to our main result, namely that the emptiness problem for our automaton model is decidable. In Section 4 we show that the universality problem, in contrast, is undecidable. Then, in Section 5 we discuss the connection with data languages. Finally, Section 6 concludes with some remarks on the complexity issues and further prospects.

2 Preliminaries

We denote the set of (positive) natural numbers by $\mathbb{N}$ (respectively, $\mathbb{N}_+$).

For every set A, we denote by 2^A the power set of A and by $\mathbb{N}^A$ the set of mappings assigning a natural number to each member of A. We denote the set of all finite (nonempty) words over A by A^* (respectively, A^+). We denote the empty word by ε. For every word $w \in A^*$, we denote its length by $|w|$.

Let A be a finite, nonempty alphabet. A nonempty word w over A defines a logical structure with the set of w's positions as its universe, equipped with the successor predicate $S(x,y)$, the order predicate $x < y$, and the label predicate $a(x)$, for each $a \in A$; these predicates are interpreted over $\{1,\dots,|w|\}$ as usual. The formulas of monadic second-order (MSO) logic over words over A are built up from: first-order variables $x, y, z, \dots$ (ranging over positions); set variables $X, Y, Z, \dots$; atomic formulas $x = y$, $x < y$, $S(x,y)$, $X(x)$, and $a(x)$, for all $a \in A$; Boolean connectives; and first-order as well as set quantifiers. We write $\varphi(x_1,\dots,x_n,X_1,\dots,X_m)$ to indicate that the MSO-formula φ may contain free occurrences of the variables $x_1,\dots,x_n,X_1,\dots,X_m$.

In the sequel, let Σ be a nonempty, finite (tree-labeling) alphabet. A tree domain D is a nonempty, prefix-closed subset of $\mathbb{N}_+^*$ such that, for each $u \in D$ and $i > 0$, if $ui \in D$, then also $uj \in D$, for each $j \in \{1,\dots,i\}$. A finite unranked tree t over Σ (Σ-labeled tree, for short) is a mapping $t\colon \mathrm{dom}_t \to \Sigma$ where dom_t is a finite tree domain. The elements of dom_t are called the nodes of t, and the node ε is called the root of t. A node $u \in \mathrm{dom}_t$ is said to have $k \geq 0$ successors if $uk \in \mathrm{dom}_t$ but $u(k+1) \notin \mathrm{dom}_t$. In this case, we call ui the i-th successor of u, and we say that ui and uj are sibling nodes, for each $i, j \in \{1,\dots,k\}$. A leaf of t is a node without any successor. Given a node u of t, the subtree of t at u is the tree given by $t|_u$ with $\mathrm{dom}_{t|_u} = \{v \in \mathbb{N}_+^* \mid uv \in \mathrm{dom}_t\}$ and $t|_u(v) = t(uv)$, for all $v \in \mathrm{dom}_{t|_u}$. Further, $t|_u$ is called a direct subtree of t if $|u| = 1$. We write t as $a(t_1 \dots t_k)$ to indicate that its root is labeled with a and that it has k successors at which the subtrees $t_1,\dots,t_k$ are rooted. We denote the set of all Σ-labeled trees by $\mathcal{T}_\Sigma$.

Let Q be a finite, nonempty set (of tree automaton states). An *atomic sibling constraint over Q* is given by an MSO-formula $\varphi(x,y)$ over words over Q, with two free first-order variables x and y, and has either of the following forms:

$$\begin{array}{llll} (\forall^=) & \forall x \forall y \,.\, \varphi(x,y) \;\to\; t_x = t_y & \qquad (\exists^=) & \exists x \exists y \,.\, \varphi(x,y) \;\wedge\; t_x = t_y \\ (\forall^{\neq}) & \forall x \forall y \,.\, \varphi(x,y) \;\to\; t_x \neq t_y & \qquad (\exists^{\neq}) & \exists x \exists y \,.\, \varphi(x,y) \;\wedge\; t_x \neq t_y \end{array}$$

Intuitively, an $\exists^=$-constraint (respectively, $\exists^{\neq}$) says that "there is a pair of positions that satisfies φ, and the subtrees at these positions are equal (or distinct, respectively)", and a $\forall^=$-constraint (respectively, $\forall^{\neq}$) says that "for each pair of positions that satisfies φ the subtrees at these positions must be equal (or distinct, respectively)". A nonempty word w over Q together with a sequence $t_1 \dots t_{|w|}$ of Σ-labeled trees (attached to w's positions) are said to satisfy an atomic sibling constraint if, depending on the constraint type, the following holds:

- $\exists^=$- or $\exists^{\neq}$-constraint: There exist some positions x, y in w such that $\varphi(x,y)$ is satisfied and $t_x = t_y$ (respectively, $t_x \neq t_y$).
- $\forall^=$- or $\forall^{\neq}$-constraint: For all positions x, y in w, if $\varphi(x,y)$ is satisfied, then $t_x = t_y$ (respectively, $t_x \neq t_y$).

For the sake of simplicity, we will sometimes refer to atomic sibling constraints simply by the underlying MSO-formulas whenever no confusion might arise. A *sibling constraint over Q* is a Boolean combination of atomic constraints. As a remark, $\forall^=$-constraints are, with respect to negation, dual to $\exists^{\neq}$-constraints, and, similarly, $\forall^{\neq}$-constraints are dual to $\exists^=$-constraints. Thus, without loss of generality, we will consider only positive Boolean combinations (i.e., without negation) of atomic constraints.

A (nondeterministic) *unranked tree automaton with equality and disequality constraints between siblings* (*UTACS*) over Σ is defined as a tuple $\mathfrak{A} = (Q, \Sigma, \Lambda, \Delta, F)$ where: Q is a finite, nonempty set of states; $F \subseteq Q$ is the set of accepting or final states; $\Lambda \subseteq \Sigma \times Q$ is the set of leaf transitions; and Δ is the set of inner-node transitions of the form (L, α, a, q), where $L \subseteq Q^+$ is a regular set, α is a sibling constraint over Q, $a \in \Sigma$, and $q \in Q$. Note that we can assume, without loss of generality, that α is a conjunction of atomic constraints; starting from sibling constraints in disjunctive normal form, each transition with a disjunction of sibling constraints can be split into several transitions each of which contains only a conjunction of atomic constraints.

For every Σ-labeled tree t, a run of $\mathfrak{A}$ on t is a Q-labeled tree $\rho\colon \mathrm{dom}_t \to Q$ such that: (a) for each leaf node $u \in \mathrm{dom}_t$, we have $(t(u), \rho(u)) \in \Lambda$; (b) for each node $u \in \mathrm{dom}_t$ with $k \geq 1$ successors, there exists a transition $(L, \alpha, t(u), \rho(u))$ in Δ such that the word $\rho(u1) \ldots \rho(uk)$ belongs to L and, together with the tree sequence $t|_{u1} \ldots t|_{uk}$, satisfies α. In case such a run exists, we write $t \rightarrow_{\mathfrak{A}} \rho(\varepsilon)$ (or simply $t \rightarrow \rho(\varepsilon)$), and say that t reaches or evaluates to $\rho(\varepsilon)$. Further, $\delta(t)$ denotes the set of states reached by t, i.e., $\delta(t) = \{q \in Q \mid t \rightarrow q\}$. Note that $\delta(t)$ can effectively be determined. The tree t is accepted by $\mathfrak{A}$ if $\delta(t) \cap F \neq \emptyset$. The set of trees accepted by $\mathfrak{A}$ is denoted by $T(\mathfrak{A})$. We call $\mathfrak{A}$ *deterministic* if, for each tree t, there exists at most one state q with $t \rightarrow q$.

Let us recall some properties of UTACS (cf. [12]). The class of UTACS is closed under union and intersection, and the class of deterministic UTACS are closed under all Boolean operations. Moreover, UTACS, in general, cannot be determinized; that is, there exists some UTACS-definable tree language which cannot be recognized by any deterministic UTACS.

3 The Emptiness Problem

The main result of this section is:

THEOREM 1. *The emptiness problem for nondeterministic UTACS is decidable.*

In [12], we have given an emptiness algorithm for the *deterministic* case. Toward showing Theorem 1, we propose incorporating a kind of subset construction into this algorithm in order to obtain an emptiness algorithm for the nondeterministic case; in this way, we have thus avoided determinization, which, in general, is not possible for nondeterministic UTACS. In order to accomplish this, we will need to refine some notions we have used in the deterministic case and adapt the algorithm appropriately. For the sake of clarity, in the following we will directly present our method for the nondeterministic case, while sometimes making reference to the deterministic case as we see fit.

Throughout this section, let $\mathfrak{A} = (Q, \Sigma, \Lambda, \Delta, F)$ be a nondeterministic UTACS.

The key difference between nondeterministic and deterministic UTACS is that for the former the state reached by a tree is, in general, not unique. Nevertheless, the *set of states* reached by every tree is unique; in other words, we have:

REMARK 2. *For every pair, t and t', of Σ-labeled trees, if $\delta(t) \neq \delta(t')$, then $t \neq t'$.*

Our emptiness algorithm is actually an adaptation of the standard marking algorithm (see, e.g., [4, Chapter 8]). The main idea is to maintain, for each set of states $S \subseteq Q$, a collection T_S containing trees t with $\delta(t) = S$. For this, we iteratively construct new trees $a(t_1 \ldots t_m)$, for some $a \in \Sigma$, where the trees $t_1, \ldots, t_m$ have been constructed in previous rounds, by checking whether some *transition that reaches S* is *applicable*. The algorithm then terminates as soon as some tree t with $\delta(t) \cap F \neq \emptyset$ has been constructed, or, otherwise, if we have constructed *enough* trees to conclude that the language recognized by the underlying automaton is empty.

In the sequel, we define the notions needed for our algorithm. First, we introduce transitions that have state sets as target. Second, we consider the applicability of such transitions. Third, we provide a bound on the number of trees we need to collect. Due to space limitations, however, we will omit most of the technical details.

Subset transitions and suitable words. As we are considering sets of states instead of mere states as the target of a transition, we are going to consider a collection of transitions instead of a single transition, which reflects the possibility to reach, with each tree, more than one target state. Such a collection of transitions, intuitively, specifies which 'normal transitions' we can apply in order to reach the states in S.

DEFINITION 3. *Let a be a symbol from Σ, and let S be a nonempty subset of Q. A* subset transition w.r.t. a and S *(or, for short, (a, S)-transition) is a collection of transitions given by a mapping $\theta\colon S \to \Delta$ such that, for each $q \in S$, the transition $\theta(q) \in \Delta$ reads the symbol a and has q as its target state. We denote the set of all (a, S)-transitions by Θ^a_S.*

An application of a subset transition $\theta \in \Theta^a_S$ to a tree $t = a(t_1 \dots t_m)$, actually, consists of applying all the transitions referred to therein to t; that is, for each of these transitions, say (L, α, a, q), there is a sequence of states $w = q_1 \dots q_m$, with $q_i \in \delta(t_i)$, for each $i = 1, \dots, m$, such that, firstly, w belongs to L and, secondly, w and $t_1 \dots t_m$ satisfy the constraint α.

Note that with this definition the result of applying a subset transition $\theta \in \Theta^a_S$ to a tree $t = a(t_1 \dots t_m)$ is, in general, not exactly S; instead, $\delta(t)$ might be a superset of S, as the definition does not forbid other transitions than the ones mentioned in θ to be applied to t.

In order to analyze the conditions under which a subset transition is applicable, we focus on the sequences of state sets that underlie an application of the subset transition (i.e., the state sets occurring at the children of the node under consideration). Let $\theta \in \Theta^a_S$ be a subset transition. A nonempty word over the power set of Q, say $\xi = S_1 \dots S_m \in (2^Q)^+$, is called *suitable* for θ (or θ-suitable, for short) if it can be used in an application of θ under the assumption that a sequence of trees $t_1, \dots, t_m$ with $\delta(t_i) = S_i$, for each $i = 1, \dots, m$, exists (thus resulting in a tree $t = a(t_1 \dots t_m)$ with $S \subseteq \delta(t)$). We denote the set of θ-suitable words by $\mathrm{suit}(\theta)$.

Note that with the notion of suitability we ignore the actual fact whether the trees needed to apply a subset transition exist. Instead, we focus on the sequences of state sets that can possibly be used for applying a subset transition under the assumption that the trees needed for the application exist; if an application is indeed possible, we then just have to arrange these trees appropriately. Thus, not surprisingly, analyzing the suitable words for a subset transition amounts to analyzing whether the equality and disequality constraints of the subset transition under consideration do not contradict one another.

More precisely, $\xi = S_1 \dots S_m \in (2^Q)^+$ is suitable for θ if there exists a family of words $(w^\tau)_{\tau \in \theta(S)}$ such that the following holds:

- For each transition $\tau \in \theta(S)$, say, $\tau = (L^\tau, \alpha^\tau, a, q^\tau)$, we have that $w^\tau \in L^\tau$ and, for each $i = 1, \dots, m$, the state at the i-th position of w^τ, say q^τ_i, belongs to S_i.

- Every two positions that are required to be equal (w.r.t. the evaluation of the constraint α^τ on w^τ, for *all* $\tau \in \theta(S)$), due to Remark 2, are labeled with the same state set.
- The equality and disequality constraints (again, with respect to the evaluation of α^τ on w^τ, for *all* $\tau \in \theta(S)$) do not contradict one another.

In particular, the satisfaction of these conditions allows an assignment of trees (if these exist) to the positions $1, \dots, m$ in order to apply all the transitions τ under consideration using the corresponding words w^τ.

Later in the emptiness algorithm, we want to look for some subset transition that is applicable using only the trees we have constructed in the previous rounds. To this end, we introduce a further restriction on the notion of suitable words. Let $\mathfrak{R} \subseteq 2^Q$ be a set of state sets, and let $\bar{d}\colon \mathfrak{R} \to \mathbb{N}$ be a mapping assigning to each state set $K \in \mathfrak{R}$ a natural number. A word $\xi = S_1 \dots S_m \in (2^Q)^+$ is called *suitable for θ with respect to $\mathfrak{R}$ and $\bar{d}$* (or $(\theta, \mathfrak{R}, \bar{d})$-suitable, for short) if it can be used in an application of θ under the assumption that there is a sequence of trees $t_1, \dots, t_m$ satisfying the following:

- for each $i = 1, \dots, m$, $\delta(t_i) = S_i$;
- for each $K \in \mathfrak{R}$, the number of distinct trees among $t_1, \dots, t_m$ that reach K, i.e., the cardinality of the set $\{t_i \mid 1 \leq i \leq m \text{ and } \delta(t_i) = K\}$, does not exceed $\bar{d}(K)$. In other words, $\bar{d}(K)$ gives the number of available distinct trees that evaluate to K.

Note that, for each $K \in 2^Q \setminus \mathfrak{R}$, we do not put any restriction on the number of distinct trees among $t_1, \dots, t_m$ that reach K. The sets of $(\theta, \mathfrak{R}, \bar{d})$-suitable words is denoted by $\mathrm{suit}(\theta, \mathfrak{R}, \bar{d})$.

As sibling constraints are based on MSO-formulas, it turns out that the suitability conditions introduced above can be translated into MSO-formulas, which justifies the following lemma.

LEMMA 4. *For each subset transition θ, each $\mathfrak{R} \subseteq 2^Q$, and each $\bar{d}\colon \mathfrak{R} \to \mathbb{N}$, the sets* $\mathrm{suit}(\theta)$ *and* $\mathrm{suit}(\theta, \mathfrak{R}, \bar{d})$ *are regular. In particular, it is decidable whether these sets are empty.*

The bound lemma. As in the deterministic case, the next step is to assert the existence of a certain bound on the number of distinct trees needed for each state set in order to apply a subset transition. Such a bound is given in Lemma 5 below. Consequently, our emptiness algorithm needs to collect, for each state set, only as many distinct trees as this bound.

For every θ-suitable word ξ, let $[\![\xi, \theta]\!] \in \mathbb{N}^{(2^Q)}$ be a mapping assigning to each set of states the number of distinct trees evaluating to this state set that are needed in order to apply θ (with respect to a particular application of θ using ξ). Note that $[\![\xi, \theta]\!]$, as has been remarked in [12], does not merely depend on ξ and θ, but also on a certain application of θ using ξ. That is, whenever we pick a θ-suitable word ξ, we always implicitly refer to such a particular application of θ, which then gives a unique value of $[\![\xi, \theta]\!]$. Note also that each value of $[\![\xi, \theta]\!]$, in general, does not need to exceed $|\xi|$.

LEMMA 5. *There exists some $B \in \mathbb{N}$ such that, for each subset transition θ of $\mathfrak{A}$ and each θ-suitable word ξ, there exists a θ-suitable word ξ' satisfying the following properties:*

1. *For each $R \subseteq Q$, $[\![\xi', \theta]\!](R) \leq B$.*
2. *For each $R \subseteq Q$, $[\![\xi', \theta]\!](R) \leq [\![\xi, \theta]\!](R)$.*
3. *For each $R \subseteq Q$, if R occurs in ξ, then it occurs in ξ' as well.*

In essence, the lemma asserts the existence of a bound B such that for each subset transition θ, if we can apply it using ξ, and if this application needs more than B distinct trees for some state set R, then we can as well apply θ using another word ξ', in place of ξ, such that the latter application needs only at most B distinct trees, for each state set. The second condition says, moreover, that the latter application can be carried out using only the trees which have already been available to the former application of θ. The third condition is merely a technical condition asserting that all state sets occurring in ξ also occur in ξ'. Note that, by the definition of subset transitions, the state sets reached by the application of θ using ξ and using ξ' might be different, in contrast to the corresponding bound lemma in the deterministic case (cf. [12, Lemma 3]).

As in the deterministic case, the bound lemma is established by a brute-force algorithm finding the desired bound iteratively. We start with some initial bound on the number of distinct trees needed for each state set in order to apply a subset transition and try all possible scenarios of the actual number of distinct trees for each state set within this bound, which boils down to checking the sets of suitable words (in each iteration with respect to the corresponding value of the bound) for emptiness; by Lemma 4, the emptiness of these sets is indeed decidable. In fact, our algorithm for finding the bound is a straightforward adjustment of the bound algorithm of the deterministic case: we only need to replace 'state' with 'set of states' and 'transition' with 'subset transition'.

The emptiness algorithm. The main idea of the our emptiness algorithm (Algorithm 1 on page 318) is to collect, for each state set $S \subseteq Q$, a certain number of trees that evaluate to S in T_S; let $\bar{d} \in \mathbb{N}^{(2^Q)}$ be such that $\bar{d}(S)$ keeps track of the cardinality of T_S. We collect trees by iteratively constructing new trees out of the trees we have collected in previous rounds by means of some applicable subset transition. In order to check the applicability of subset transitions, we look for subset transitions for which the set of suitable words has not yet been exhausted (cf. the **if**-condition of Line 6–10). Here, the crucial point is to find some appropriate suitable word ξ, which can effectively be done since the emptiness of $\mathrm{suit}(\theta, \bar{d})$ is decidable, and the algorithm, at any point during its execution, stores only a finite number of trees.

In order to guarantee termination, we set a bound on the number of trees we are collecting, i.e., the algorithm terminates as soon as this bound has been reached. Such a bound is provided by Lemma 5, which says that, for each state set S, it suffices to collect up to B trees. We encounter some difficulties, though: as has been noted before, applying a subset transition (say, for a state set S) using a suitable word might lead to a tree that does not evaluate exactly to S but, instead, to some superset S' of S. In order to deal with this, we observe that, as far as the applicability of (subset) transitions is concerned, trees evaluating to S' can be used as a replacement for trees evaluating to S; in this case, we have to keep the trees used for S' and the ones used for S separately in order to maintain the satisfaction of the disequality constraints. Thus, instead of collecting B trees for S and S' each, we can as well collect, for instance, $(2 \cdot B)$ trees for S'.

We exploit this observation in the algorithm by considering, for each state set S, not only T_S, but also the union of all $T_{S'}$ with $S' \supseteq S$, which is denoted by $T_S{\uparrow}$ and which is referred to as a *(tree) collection*. In other words, we put a bound, z, on the cardinality of such tree

Algorithm 1 The emptiness algorithm

```
procedure EMPTY($\mathfrak{A}$)
    compute the bound $B$ according to Lemma 5
    initialize each $T_S$ with $\{a \in \Sigma \mid \delta(a) = S\}$
    $z := (B+1) \cdot 2^{2|Q|}$
    repeat
        if there exist some subset transition $\theta \in \Theta_S^a$, some word $\xi = S_1 \dots S_m \in \mathrm{suit}(\theta, \bar{d})$,
           and some trees $t_1, \dots, t_m$ with $t_i \in T_{S_i}$ such that
           − $T_S\!\uparrow$ is not full, i.e., $|T_S\!\uparrow| < z$,
           − $\theta$ can be applied using $\xi$ and $t_1, \dots, t_m$, and
           − $a(t_1 \dots t_m)$ has not been constructed before, i.e., $a(t_1 \dots t_m) \notin \bigcup_{R \subseteq Q} T_R$
        then add $a(t_1 \dots t_m)$ to $T_R$, where $R = \delta(a(t_1 \dots t_m))$, and update $\bar{d}$
              if $T_S\!\uparrow$ has become full (i.e., $|T_S\!\uparrow| \geq z$) then $z := z - 2^{|Q|}$
    until no new tree can be constructed
    if $T_S \neq \emptyset$ for some $S \subseteq Q$ with $S \cap F \neq \emptyset$ then return '$T(\mathfrak{A}) \neq \emptyset$'
    else return '$T(\mathfrak{A}) = \emptyset$'
end procedure
```

collections; that is, we consider a tree collection $T_S\!\uparrow$ *full* (with respect to z) if $|T_S\!\uparrow| \geq z$. Since there are $2^{|Q|-|S|}$ supersets of S, it suffices, for $T_S\!\uparrow$, to collect $B \cdot 2^{|Q|-|S|}$ trees (i.e., B trees for each superset of S). Furthermore, in order to cope with some technicalities arising from the correctness proof of the algorithm (cf. Lemma 7 below), we initialize z with $(B+1) \cdot 2^{2|Q|}$ and decrease z by $2^{|Q|}$ each time a tree collection turns full.

REMARK 6. *Since the bound z is non-increasing, once a tree collection has been declared full, it stays full until the termination of the algorithm. In particular, z is decreased at most $2^{|Q|}$ times since the decrement only takes place if a tree collection turns full (and there are $2^{|Q|}$ of them). Moreover, upon termination of the algorithm, the value of z is at least $B \cdot 2^{2|Q|}$.*

Therefore, for each tree collection, at most $(B+1) \cdot 2^{2|Q|}$ trees are constructed, and in each iteration of the **repeat**-loop a new tree is constructed. Consequently, this loop is iterated at most $((B+1) \cdot 2^{3|Q|})$-times, so the algorithm eventually terminates.

The algorithm is sound as trees are constructed according to the subset transitions of $\mathfrak{A}$. The completeness of the algorithm follows from Lemma 7 below, which is similar to the completeness lemma of the deterministic case (cf. [12, Lemma 6]).

LEMMA 7. *For each tree $t \in \mathcal{T}_\Sigma$ and each state set $S \subseteq Q$, if $\delta(t) = S$, then $t \in T_S$ (that is, the tree t is eventually constructed by the algorithm), or $T_S\!\uparrow$ has been declared full (for some value of z) upon the termination of the algorithm.*

As with its deterministic-case counterpart, the proof of this lemma goes by an induction on the structure of t. However, it is more involved, in particular for the case $t = a(t_1 \dots t_m)$ with $\delta(t) = S$ but $t \notin T_S$ in the induction step. For this, we have to show that the tree collection $T_S\!\uparrow$ can be declared full by constructing as many trees as necessary. In order to achieve this, we also need to reduce, in the course of the algorithm, the requirement of a tree collection being full, which is done by decreasing the bound z on the collection size.

The complexity of our emptiness algorithm depends on the bound B given by the bound lemma. Unfortunately, we have not yet been able to give an upper bound for B since in the proof of the bound lemma we make use of Dickson's Lemma [9], which guarantees that our algorithm for finding the desired bound indeed terminates, but which does not come with any complexity analysis (recall that the proof of Dickson's Lemma is a non-constructive one). Thus, the complexity of our emptiness algorithm is still an open issue.

4 The Universality Problem

The universality problem for UTACS is the question whether a given UTACS accepts all its input trees. For deterministic UTACS, this problem is decidable since deterministic UTACS are effectively closed under complementation; the decidability of universality then follows from the decidability of emptiness (see [12]). For nondeterministic UTACS, in contrast, it turns out that this problem is undecidable.

THEOREM 8. *The universality problem for nondeterministic UTACS is undecidable.*

In order to show this, we use a reduction from the halting problem for 2-register machines: given a 2-register machine, we construct a UTACS such that the 2-register machine has a halting computation if and only if there exists some unranked tree that is not accepted by the UTACS, which is supposed to be the encoding of the halting computation. Due to space limitations, we will only present a brief sketch of the encoding we use and point out the main difficulties arising in the proof of Theorem 8.

In the core of the reduction is how the computations of a 2-register machine are encoded as unranked trees. As usual, a computation of a 2-register machine is a sequence $\kappa_1 \ldots \kappa_m$ where, for $i = 1, \ldots, m$, $\kappa_i = (p_i, d_i, e_i)$ is a configuration, which records the current control state p_i as well as the contents $d_i, e_i \in \mathbb{N}$ of the registers. Basically, we want to encode such a computation as a word of the form $p_1 \perp a^{d_1} \perp\!\!\!\perp b^{e_1} \$ \ldots \$ p_m \perp a^{d_m} \perp\!\!\!\perp b^{e_m}$, where $a, b, \perp, \perp\!\!\!\perp, \$$ are new symbols. We then consider each symbol of this word as the root of a unary tree (of a certain depth) and connect these trees with a single root, thus obtaining a tree that represents the underlying computation.

This encoding is actually quite similar to the encoding of the solutions of PCP (Post's Correspondence Problem) as data words in [2], which is used to show that the satisfiability problem for the logic $\mathrm{FO}^3(\sim, S)$ over data words is undecidable. In fact, the unary trees mentioned in our encoding above can be seen as data values that are attached to the word encoding of a 2-register-machine computation (see also Section 5 below).

Although the reduction, given the encoding, is fairly standard, we encounter some technical difficulties arising from the restricted quantification patterns in our definition of UTACS constraints (i.e., only $\forall x \forall y$ and $\exists x \exists y$). As an illustration, consider the case of a word encoding of a 2-register machine computation which contains two consecutive configurations, say $\kappa = p \perp a^d \perp\!\!\!\perp b^e$ and $\kappa' = p' \perp a^{d'} \perp\!\!\!\perp b^{e'}$, which do not represent a correct execution of, say, an increment to the first register. This occurs if, for example, $e < e'$ (that is, κ' contains more b's than κ). Intuitively, this can be captured by expressing the following constraints: first, in each of κ and κ', all the unary trees attached to the b-positions are pairwise different; second, for each b-position of κ there exists a b-position in κ' with the same unary tree;

third, there exists a b-position in κ' such that the unary tree attached to it does not occur at the b-positions of κ. Notice the quantification patterns occurring in these constraints: $\forall x \forall y$, $\forall x \exists y$, and $\exists x \forall y$, respectively. The first quantification pattern, $\forall x \forall y$, can easily be handled by UTACS constraints. The third one, $\exists x \forall y$, can be handled by first nondeterministically guessing the position of x and then comparing this position with all positions y using UTACS constraints. The second quantification pattern, $\forall x \exists y$, however, cannot be captured by UTACS constraints, so we have to overcome this difficulty by putting some additional requirements on the unary trees used in the tree encoding of a halting computation.

5 UTACS and Data Languages

In this section, we are interested in a connection between languages of data words and tree automata with equality constraints, which is established by encoding data words as trees of a certain form. With this encoding, then, data equality amounts to equality between trees.

For the ease of exposition, we consider data words over Σ and $\mathbb{N}_+$, where Σ is a finite alphabet. A data word $w = w_1 \dots w_m$ is a finite sequence of pairs of the form $w_i = (a_i, d_i) \in \Sigma \times \mathbb{N}_+$. We encode such a data word as a tree of the form $\top\ (\ a_1\ \underline{d_1}\ a_2\ \underline{d_2}\ \dots\ a_m\ \underline{d_m}\ )$ where $\top$ is a new symbol and $\underline{d_i}$, for each $i = 1, \dots, m$, encodes the data value d_i (i.e., a positive integer) as a unary tree over $\{\bullet\}$ of depth d_i. Consequently, we can use UTACS accepting trees of this form to define languages of data words. For such a language of data words, in particular, the emptiness problem then amounts to the emptiness problem for UTACS, which, by Theorem 1, is decidable.

Following the notations of [16], more specifically, we consider a fragment of the logic $\mathrm{MSO}(\sim, <, S)$ over data words (with $\sim$ being the data-equality predicate; i.e., $x \sim y$ holds if the data value at position x is equal to the one at position y), which contains positive Boolean combinations of formulas of the form $\exists X_1 \dots \exists X_n . \Big(\ \theta(X_1, \dots, X_n) \wedge \alpha(X_1, \dots, X_n)\ \Big)$, where θ is an $\mathrm{MSO}(<, S)$-formula (i.e., without $\sim$) over Σ with free occurrences of the set variables $X_1, \dots, X_n$, and α is a positive Boolean combination of formulas of the forms:

$$\forall x \forall y . \left[\ \varphi(X_1, \dots, X_n, x, y) \rightarrow x \sim y\ \right] \qquad \exists x \exists y . \left[\ \varphi(X_1, \dots, X_n, x, y) \wedge x \sim y\ \right]$$
$$\forall x \forall y . \left[\ \varphi(X_1, \dots, X_n, x, y) \rightarrow x \not\sim y\ \right] \qquad \exists x \exists y . \left[\ \varphi(X_1, \dots, X_n, x, y) \wedge x \not\sim y\ \right]$$

where φ, in turn, is an $\mathrm{MSO}(<, S)$-formula (i.e., without $\sim$) over Σ with the free variables $X_1, \dots, X_n$ and x, y. Note that these kinds of formulas correspond to the types of constraints we have used in defining the transitions of UTACS. In fact, for every formula of the logic over data words described above, we can construct a UTACS recognizing the set of trees encoding the data words defined by the formula. This allows us to derive from Theorem 1 that the satisfiability problem for this logic is decidable. Furthermore, as remarked in Section 4, in the proof of Theorem 8 we actually encode computations of 2-register machines as data words. Consequently, the validity problem for this logic (i.e., the problem of determining, for a given formula, whether all data words satisfy this formula) is undecidable.

In comparison with the logic $\mathrm{FO}^2(\sim, <, S)$ (first-order logic with two variables) over data words, which is considered in [2], the logic described above seems to be weaker because of the restricted use of data comparisons. In particular, for the languages of data words defined by our logic, the projection w.r.t. the finite alphabet always yields a regular language; this can be shown by using an analysis related to the notion of suitability used in Section 3. For instance, the language of data words (over the label alphabet $\{a, b\}$) satisfying "every two positions labeled with a carry different data values, and for each position labeled with a there exists a position labeled with b with the same data value" cannot be defined in our logic since the projection of this language w.r.t. $\{a, b\}$ yields a language which is not regular, but it can be defined in $\mathrm{FO}^2(\sim, <, S)$ (see [2]). On the other hand, formulas of our logic may use more than just two variables, while still being decidable. This allows us to define, for instance, the language of data words satisfying "between every two a-positions carrying the same data value there exists a b-position" by saying that every two a-positions with no b-position in between must carry different data values, that is, $\forall x \forall y. \left[\, x < y \wedge a(x) \wedge a(y) \wedge \neg(\exists z. x < z < y \wedge b(z)) \rightarrow x \not\sim y \,\right]$. To the best of our knowledge, it is still an open question whether this language can be expressed in $\mathrm{FO}^2(\sim, <, S)$ (see [16]).

6 Conclusions

We have shown that the emptiness problem for nondeterministic unranked tree automata with sibling equality and disequality constraints is decidable by extending the method we have proposed previously for the deterministic case. However, the precise complexity (both lower and upper bound) of the problem is still missing. One possible approach towards an upper bound for our method is to provide a different proof for the bound lemma which avoids the use of Dickson's Lemma.

We believe that the connection between our automaton model and languages of data words deserves further studies. Here, it might be worthwhile to study the precise relation of the logic over data words emerging from our automaton model to the existing formalisms for data languages and also to study its complexity.

We remark that our method of deciding emptiness for UTACS, actually, does not rely on the fact that trees are compared with respect to equality. In fact, our method still works if we consider, for instance, automata with sibling constraints with respect to structural equality (two trees are said to be structurally equal if they share the same set of nodes). Thus, we would like to study (in the unranked setting) automata with constraints regarding more general types of relations between trees than just equality, like, e.g., relations defined by tree transducers. This is actually related to some recent works on (ranked) tree automata with constraints. In the automaton models of [5, 6], constraints are posed not directly on the input subtrees, but, instead, on some output trees, called memories, which are produced during a bottom-up run of the automata. Similarly, in [11], one defines a function assigning to each tree a certain size and poses (numerical) constraints with respect to this size function.

Finally, it would be interesting to compare our automaton model with the automata defined in [10], where it is allowed to compare subtrees which are not necessarily siblings but may be remotely located in the tree.

Acknowledgements. We thank Luc Segoufin for many valuable discussions. We also thank the anonymous referees for their numerous comments and suggestions; due to space

limitations, however, we were not able to include all these suggestions.

References

[1] Bogaert, B., Tison, S.: Equality and disequality constraints on direct subterms in tree automata. In *Proc. STACS 1992. LNCS 577*. Springer (1992)

[2] Bojańczyk, M., Muscholl, A., Schwentick, T., Segoufin, L., David, C.: Two-variable logic on words with data. In *Proc. LICS 2006*. IEEE Computer Society (2006)

[3] Caron, A.C., Comon, H., Coquidé, J.L., Dauchet, M., Jacquemard, F.: Pumping, cleaning and symbolic constraints solving. In *Proc. ICALP 1994. LNCS 820*. Springer (1994)

[4] Comon, H., Dauchet, M., Gilleron, R., Löding, C., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: *Tree Automata Techniques and Applications*. Available on `http://www.grappa.univ-lille3.fr/tata` (2007) Released on 12 October 2007.

[5] Comon, H., Cortier, V.: Tree automata with one memory, set constraints and cryptographic protocols. *Theoretical Computer Science* **331** (2005)

[6] Comon-Lundh, H., Jacquemard, F., Perrin, N.: Visibly tree automata with memory and constraints. *Logical Methods in Computer Science* **4** (2008)

[7] Dal Zilio, S., Lugiez, D.: XML schema, tree logic and sheaves automata. In *Proc. RTA 2003. LNCS 2706*. Springer (2003)

[8] Dauchet, M., Caron, A.C., Coquidé, J.L.: Automata for reduction properties solving. *Journal of Symbolic Computation* **20** (1995)

[9] Dickson, L.E.: Finiteness of the odd perfect and primitive abundant numbers with n distinct prime factors. *American Journal of Mathematics* **35** (1913)

[10] Filiot, E., Talbot, J.M., Tison, S.: Tree automata with global constraints. In *Proc. DLT 2008. LNCS 5257*. Springer (2008)

[11] Habermehl, P., Iosif, R., Vojnar, T.: Automata-based verification of programs with tree updates. In *Proc. TACAS 2006. LNCS 3920*. Springer (2006)

[12] Karianto, W., Löding, C.: Unranked tree automata with sibling equalities and disequalities. In *Proc. ICALP 2007. LNCS 4596*. Springer (2007) Full version appeared as Technical Report AIB-2006-13, RWTH Aachen University.

[13] Mongy-Steen, J.: *Transformation de noyaux reconnaissables d'arbres. Forêts RATEG*. PhD thesis, Université de Lille I (1981)

[14] Neven, F.: Automata, logic, and XML. In *Proc. CSL 2002. LNCS 2471*. Springer (2002)

[15] Schwentick, T.: Automata for XML – a survey. *J. Comput. Syst. Sci.* **73** (2007)

[16] Segoufin, L.: Automata and logics for words and trees over an infinite alphabet. In *Proc. CSL 2006. LNCS 4207*. Springer (2006)

[17] Seidl, H., Schwentick, T., Muscholl, A.: Counting in trees. In *Logic and Automata: History and Perspectives*. Amsterdam University Press (2008)

This work is licensed under the Creative Commons Attribution-NonCommercial-No Derivative Works 3.0 License.