Unranked Tree Automata with Sibling Equalities and Disequalities

Wong Karianto and Christof Löding

Lehrstuhl für Informatik 7, RWTH Aachen, Germany

Abstract. We propose an extension of the tree automata with constraints between direct subtrees (Bogaert and Tison, 1992) to unranked trees. Our approach uses MSO-formulas to capture the possibility of comparing unboundedly many direct subtrees. Our main result is that the nonemptiness problem for the deterministic automata, as in the ranked setting, is decidable. Furthermore, we show that the nondeterministic automata are more expressive than the deterministic ones.

1 Introduction

The notion of unranked trees, i.e., finite (ordered) trees for which there are no constraints on the number of successors of a node, has recently regained interest from the research community, especially due to the application of such trees as models of semi-structured data. As with ranked trees, automata-related and logic-related notions have been developed for unranked trees. In fact, many results that hold for the ranked case have been shown to hold for the unranked case as well. For references, the reader is referred to, e.g., the surveys [16, 13].

A current trend in the theory of unranked tree automata is concerned with the development of logics and automaton models that are more expressive than the framework of finite automata and, at the same time, have good (algorithmic) properties. Such frameworks, in turn, can be useful for developing (logic-based) query languages over unranked trees (with application to query languages for XML documents) with desirable algorithmic properties. A particular approach along this line, for instance, has been to incorporate the notion of numerical constraints in unranked tree automata: in the Presburger automata of Seidl et. al. [17, 18] (cf. also the sheaves automata of Lugiez and Dal Zilio [8]), the applicability of a (bottom-up) transition at a node of an input tree is subject to formulas of Presburger arithmetic (the first-order logic over the natural numbers) over the occurrences of the states to which the children of this node are evaluated to. In another approach, structures (among others, words and trees) with data, i.e. where the nodes carry, besides a label from a finite alphabet, a data value from an infinite set, have been considered; see, e.g., [3, 2] as well as the references therein. In these two papers, logics and automata with equality tests between data values in different nodes are studied, and decidability results for fragments of first-order logics over words and unranked trees with data are shown.

Regarding the latter approach, another possibility of incorporating data is to encode the data value of a node (together with the node's label) as a subtree over

a finite alphabet (e.g., natural numbers can be coded by unary subtrees of the corresponding depth) instead of directly taking values from an infinite alphabet. In this view, equality tests between data values then amount to equality comparisons between subtrees. As a first step toward this approach, in this paper we study a class of (unranked) tree automata that can deal with such comparisons in a restricted way.

In the ranked setting, automata with equality comparisons between subtrees have been studied in the literature; for references, see [7, Chapter 4]. It turns out that tree automata with such constraints, in the most general form where it is allowed to compare arbitrary subtrees, fail to have a decidable nonemptiness problem [15]. This result even carries over to the case where equality tests are only allowed between cousin subtrees, i.e., subtrees of depth at most two [19]. Nevertheless, by imposing appropriate restrictions on the transition structure and/or the equality constraints, it is possible to identify classes of automata with a decidable nonemptiness problem; the class of reduction automata and its variants [9, 6, 11] are a case in point. Another subclass of tree automata with equality constraints has been suggested by Bogaert and Tison [1]: they allow equality (and disequality) constraints only between sibling subtrees (i.e., direct subtrees or subtrees of depth one) and show that this class forms a Boolean algebra and that the nonemptiness problem for this class is decidable.

In this paper, we aim at extending Bogaert and Tison's automaton model to the unranked setting. However, even the definition of such automata is not obvious: with unrankedness, on the one hand, the number of pairs of sibling subtrees to be compared is not a priori bounded and may increase with the size of the input tree. On the other hand, the (possibly unboundedly many) sibling comparisons must be finitely representable in order to define an automaton model properly. Here, we propose using formulas of monadic second-order logic over the state set of the underlying automaton to address the pairs of siblings to be compared. In this way, we meet the two requirements just mentioned: unbounded number of but finitely representable equality tests between sibling subtrees.

The main result of this paper is that the nonemptiness problem for the deterministic unranked tree automata with equality and disequality constraints between siblings we propose is decidable, which we show by adapting Bogaert and Tison's nonemptiness decision procedure. As a remark, using encodings (e.g., the first-child-next-sibling encoding; cf. [16]), a standard way of transferring results from ranked trees to unranked ones, obviously fails for our purposes because the sibling relation must be destroyed by any encoding mapping unranked trees to ranked ones.

Further, regarding the use of subtrees to represent data values mentioned above, we would like to point out that, if we want to test equality only between the data values and ignore the node labels, then we actually do not want to compare whole sibling subtrees. Thus, a next step along this line of study would be to consider automaton models that do not directly compare subtrees but, instead, the output of some preprocessing of the subtrees; see Section 5 for a discussion on this.

Outline of the paper. After fixing our notations in Section 2, in Section 3 we present our automaton model, indicate some closure properties, and show that the nondeterministic automata are more expressive than the deterministic ones. In Section 4 we show our main result, namely that the nonemptiness problem for the deterministic case is decidable. Section 5 indicates some possible variations of our automaton model. We conclude with some remarks on further prospects in Section 6. Due to space limitations, proof details are omitted and can be found in the preliminary version of this paper [12].

Related works. Lugiez [14] proposes automata on multitrees (unranked, unordered trees) with a certain type of constraints among sibling multitrees in the transitions and shows that these automata are closed under Boolean operations, determinizable, and have a decidable nonemptiness problem. The constraints he uses incorporate both numerical (Presburger) constraints and inclusion relations among multisets of (multi)trees. By using Boolean combinations of constraints of the latter kind, it is then possible to impose equality tests among sibling (multi)trees, so his work also extends Bogaert and Tison's. Nevertheless, his approach is not comparable to ours in several respects. In his approach, besides unorderedness, evaluating a constraint in an unbounded (unordered) sequence of (multi)trees is reduced to evaluating the constraint in an (unordered) sequence of multisets of trees whose length is bounded by the number of states of the underlying automaton. Consequently, first, equality tests are imposed between multisets of trees (in our setting: between trees), and second, the number of equality tests depends on the number of states of the automaton instead of the size of the input (multi)tree.

2 Preliminaries

We denote the set of (positive) natural numbers by $\mathbb{N}$ (respectively, $\mathbb{N}_+$). For $k > 0$, the set of k-tuples over these sets are denoted by $\mathbb{N}^k$ and $\mathbb{N}_+^k$, respectively; throughout the paper, such tuples are usually denoted by $\bar{d}, \bar{e}, \ldots$. Further, as usual, these tuples are ordered by comparing them componentwise. Whenever k is clear from the context, we denote by $\widehat{m}$, for $m \in \mathbb{N}$, the k-tuple $(m, \ldots, m)$.

For a set A, we denote the set of all (finite) words over A by A^*. We denote the empty word by ε and write A^+ for $A^* \setminus \{\varepsilon\}$. For a word w over A, we denote its length by $|w|$.

Let A be a finite, nonempty alphabet. A nonempty word w over A defines the word structure $\langle \{1, \ldots, |w|\}, S, <, (\chi_a)_{a \in A} \rangle$ where S and $<$ denote the successor and the order relation, respectively, over the set $\{1, \ldots, |w|\}$ of positions in w, and χ_a, for each $a \in A$, is the set of a-labeled positions in w. To simplify notation, we do not distinguish between a word and its corresponding word structure. The formulas of monadic second-order (MSO) logic over words over A are built up from: first-order variables $x, y, z, \ldots$, which range over positions; monadic second-order variables $X, Y, Z, \ldots$, which range over sets of positions; atomic formulas $x = y$, $x < y$, $S(x, y)$, $X(x)$, and $\chi_a(x)$, for all $a \in A$ and for all variables x, y, X; Boolean connectives; and first-order as well as set quantifiers. We

write $\varphi(x_1, \ldots, x_n, X_1, \ldots, X_m)$ to indicate that the MSO-formula φ may contain free occurrences of the variables $x_1, \ldots, x_n, X_1, \ldots, X_m$. If a word structure w, together with an assignment of positions $\kappa_1, \ldots, \kappa_n$ and of sets $K_1, \ldots, K_m$ of positions in w to the free variables $x_1, \ldots, x_n, X_1, \ldots, X_m$, respectively, satisfies φ, we write $w \models \varphi(\kappa_1, \ldots, \kappa_n, K_1, \ldots, K_m)$.

In the sequel, Σ will always denote 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 $j \in \{1, \ldots, i\}$. A finite unranked tree t over Σ (or simply Σ-labeled tree in the sequel) 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 $i, j \in \{1, \ldots, k\}$. 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 $a(t_1 \cdots t_k)$ to indicate that its root is labeled with a and that it has k successors at which the subtrees $t_1, \ldots, t_k$ are rooted. We denote the set of all Σ-labeled trees by $\mathcal{T}_\Sigma$.

3 Automata with Equality and Disequality Constraints between Siblings on Unranked Trees

In the framework of ranked trees, roughly speaking, a finite tree automaton processes a given input tree by assigning states to the nodes of the tree, say, in a bottom-up fashion, according to its transitions. The automaton model introduced in [1] extends this definition by requiring that the application of a transition on a node of the input tree is subject to some equality and disequality constraints between the direct subtrees of that particular node. For instance, "$1 = 2 \wedge 1 \neq 3$" expresses the property that the first and the second subtree are equal while the first and the third subtree are different from each other. Note that the constraints directly address the subtrees to be compared, which is possible since the number of successors of any node of a ranked tree is bounded.

When moving on to the framework of unranked trees, we encounter the fact that the number of successors of a node, when applying a transition, is no longer a priori bounded by a rank. The usual approach to this phenomenon is to use regular word languages over the set of states in the transitions of a finite tree automaton instead of mere sequences of states (cf. [4]). In this way, we allow the number of successors of a node to be arbitrarily large (but finite) while ensuring a finite representation (by means of, e.g., regular expressions over the set of states) of the automaton model.

The same phenomenon occurs if we now want to add equality and disequality constraints between the direct subtrees of a node (or simply *sibling constraints* for short). As an illustration, if we want to express that "all direct subtrees are equal to one another", then we will have to address unboundedly many pairs of

direct subtrees to be compared; in the framework of ranked trees, say, of rank k, we just need to define the constraint $\bigwedge_{1\leq i,j\leq k}(i=j)$. To cope with this phenomenon, in the sequel, we will use MSO-formulas to address the pairs of direct subtrees to be compared; by doing so, we take into account the unboundedness aspect while ensuring that the constraints we use are finitely representable.

Let A be a finite, nonempty alphabet. An *atomic sibling constraint over* A is given by a pair (φ,η) where $\varphi(x,y)$ is an MSO-formula over words over A that may contain free occurrences of the first-order variables x and y, and η is one of the following four types of usage: $\exists_{\mathsf{EQ}}$, $\exists_{\mathsf{NEQ}}$, $\forall_{\mathsf{EQ}}$, and $\forall_{\mathsf{NEQ}}$. Intuitively, an $\exists_{\mathsf{EQ}}$-constraint ($\exists_{\mathsf{NEQ}}$-constraint) says that "there is a pair of positions that satisfies φ and the subtrees at these positions are equal (or distinct, respectively)", and a $\forall_{\mathsf{EQ}}$-constraint ($\forall_{\mathsf{NEQ}}$-constraint) says that "for each pair of positions that satisfies φ the subtrees at these positions must be equal (or distinct, respectively)". Formally, a nonempty word w over A together with a sequence $t_1\dots t_{|w|}$ of Σ-labeled trees are said to satisfy an atomic sibling constraint (φ,η) if, depending on η, one of the following holds:

- $\eta=\exists_{\mathsf{EQ}}$: there exist $\kappa,\lambda\in\{1,\dots,|w|\}$ such that $w\models\varphi(\kappa,\lambda)$ and $t_\kappa=t_\lambda$.
- $\eta=\exists_{\mathsf{NEQ}}$: there exist $\kappa,\lambda\in\{1,\dots,|w|\}$ such that $w\models\varphi(\kappa,\lambda)$ and $t_\kappa\neq t_\lambda$.
- $\eta=\forall_{\mathsf{EQ}}$: for all $\kappa,\lambda\in\{1,\dots,|w|\}$, if $w\models\varphi(\kappa,\lambda)$, then $t_\kappa=t_\lambda$.
- $\eta=\forall_{\mathsf{NEQ}}$: for all $\kappa,\lambda\in\{1,\dots,|w|\}$, if $w\models\varphi(\kappa,\lambda)$, then $t_\kappa\neq t_\lambda$.

A *sibling constraint over* A is built up from atomic sibling constraints by means of Boolean connectives, and the semantics definition above is extended accordingly. The set of all sibling constraints over A is denoted by CONS_A.

We remark that $\exists_{\mathsf{EQ}}$-constraints and $\forall_{\mathsf{NEQ}}$-constraints are dual with respect to negation. Likewise, $\exists_{\mathsf{NEQ}}$-constraints and $\forall_{\mathsf{EQ}}$-constraints are dual with respect to negation. Hence, it suffices to consider only positive Boolean combinations (i.e., without negation) of atomic sibling constraints.

An *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 final or accepting states; $\Lambda\subseteq\Sigma\times Q$ contains the leaf transitions; and

$$\Delta\subseteq \mathrm{Reg}_+(Q)\times\mathsf{CONS}_Q\times\Sigma\times Q\ ,$$

where $\mathrm{Reg}_+(Q)$ denotes the set of regular subsets of Q^+, is the set of inner-node transitions. Given a Σ-labeled tree t, a run of $\mathfrak{A}$ on t is defined as a Q-labeled tree $\rho\colon \mathrm{dom}_t\to Q$ with the following property: (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 $(L,\alpha,t(u),\rho(u))\in\Delta$ such that the word $\rho(u1)\dots\rho(uk)$ belongs to L and, together with the tree sequence $t|_{u1}\dots t|_{uk}$, satisfies α. In case such a run exists, we write $t\to_{\mathfrak{A}}\rho(\varepsilon)$ or simply $t\to\rho(\varepsilon)$, whenever no confusion might arise, and say that t evaluates to $\rho(\varepsilon)$. The run ρ is said to be accepting if $\rho(\varepsilon)\in F$. The tree t is accepted by $\mathfrak{A}$ if there is an accepting run of $\mathfrak{A}$ on t. The set of trees accepted by $\mathfrak{A}$ is denoted by $T(\mathfrak{A})$.

The UTACS $\mathfrak{A}$ is called *deterministic* if, for each tree $t\in\mathcal{T}_\Sigma$, there exists at most one state q with $t\to q$.

Example 1. The set of well-balanced trees over the alphabet $\{a\}$ can be recognized by a UTACS by taking $Q = F = \{q\}$, $\Lambda = \{(a, q)\}$, and $\Delta = \{(Q^+, \alpha, a, q)\}$ with $\alpha = (\mathsf{true}, \forall_{\mathsf{EQ}})$.

By adapting the standard constructions from the ranked setting (see, for example, [7, 1]), one can show that the class of (nondeterministic) automata with constraints between siblings on unranked trees is closed under union and intersection, and that the class of deterministic automata is closed under complementation. On the other hand, the nondeterministic automata, as opposed to the ranked case, are more powerful than the deterministic ones (see Proposition 2 below); this fact, in turn, raises the question whether the class of nondeterministic UTACS's is closed under complementation.

Proposition 2. *There exists a tree language that is recognizable by a nondeterministic UTACS, but by no deterministic UTACS.*

The tree language that we use to separate the two classes consists of trees of the form depicted in Figure 1. Intuitively, such a tree consists of a root labeled

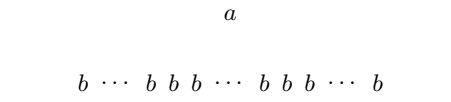

Fig. 1. Trees separating nondeterministic and deterministic UTACS's (the dashed lines represent b-strands)

with a and below it strands of b's. All but two of the b-strands are of the same length, and the two special b-strands themselves are of the same length. With nondeterminism, essentially, we would guess the positions of the latter b-strands and mark them by means of a special state. Then, using this particular state, we can address the appropriate pairs of positions that should be equal and those that should be distinct. With determinism, this is no longer possible; the fact that there are b-strands of arbitrary length prevents the possibility of using a special state to mark the positions of the two special b-strands and thus also of addressing their positions in the constraints.

4 Nonemptiness Problem: the Deterministic Case

In the ranked setting, it has been shown in [1] that the nonemptiness problem for deterministic automata with sibling constraints is decidable, which carries over into nondeterministic automata since the latter can be determinized. The method used there is an adaptation of the standard marking algorithm: one constructs trees that are accepted by the automaton under consideration, in

a bottom-up fashion, by applying the transitions of the automaton. In order to apply a transition, in turn, one needs to find, for each state occurring in the transition, a tree that evaluates to this state. With disequality constraints, however, we may need more than one tree evaluating to a state; for instance, if a transition requires that the first and the second subtree both evaluate to the same state, say p, and that they are distinct, then at least two distinct trees evaluating to p are needed to apply the transition. Of course, if we want to apply a transition to a node, then we need, for each state occurring in the transition, at most only as many distinct trees as the number of successors of this node. Thus, if the number of successors a tree node may have is bounded, then this bound gives an upper bound on the sufficient number of distinct trees needed for each state in order to apply a transition.

Now, our main obstacle to transferring the above nonemptiness decision procedure to the unranked setting indeed lies in the unrankedness aspect: as the number of successors of a tree node is not a priori bounded, we first need to find out how we can bound the sufficient number of distinct trees needed to satisfy a sibling constraint. In Lemma 3 below, we assert the existence of such a bound: for each transition, if this transition is applicable, then as many distinct trees as given by this bound are sufficient in order to apply this transition. Using this bound, we can then devise, based on the corresponding algorithm in the ranked setting, a nonemptiness decision procedure for deterministic UTACS's.

In the remainder of this section, unless stated otherwise, let $\mathfrak{A} = (Q, \Sigma, \Delta, \Lambda, F)$ be a deterministic UTACS and $\tau = (L, \alpha, a, q)$ be a transition of $\mathfrak{A}$.

We call a word $w \in Q^+$ *suitable for* τ if it can be used in an application of τ, thus resulting in a tree that evaluates to q, provided that, for each state occurring in w, there are plenty of (i.e., sufficiently many) distinct trees evaluating to this state. Note that not every word is suitable for τ; for instance, if α requires that the subtrees at the first and the second position of w are equal, then the states at these positions must be the same, too, since $\mathfrak{A}$ is deterministic. Let us denote the set of words that are suitable for τ by S_τ. Now, in order to analyze the applicability of τ, it suffices only to consider words that are suitable for τ, for if a word w is not suitable for τ, then there is no sequence of trees together with which w can both belong to L and satisfy the constraint α. Moreover, in the following exposition we can assume that S_τ is not empty as τ otherwise cannot be applied at all and can thus be removed from Δ.

For a τ-suitable word w, let $[\![w, \tau]\!] \in \mathbb{N}^{|Q|}$ be a tuple of natural numbers that indicates, for each state, the number of distinct trees that are used for a particular application of τ that uses w. Alternatively, $[\![w, \tau]\!]$ can be seen as a mapping $[\![w, \tau]\!] \colon Q \to \mathbb{N}$ where $[\![w, \tau]\!](p)$ is assigned the p-component of $[\![w, \tau]\!]$, for each $p \in Q$. We would like to point out that the value of $[\![w, \tau]\!]$ does not solely depend on w and τ, but also on a certain application of τ by means of w. In order to simplify our presentation, whenever we pick a τ-suitable word w, in the following, we always implicitly refer to such a particular application of τ, which then gives a unique value of $[\![w, \tau]\!]$. We remark that, in general, $[\![w, \tau]\!](p)$, for each $p \in Q$, does not need to exceed $|w|$.

Our aim is to show the existence of a bound N such that for each word w that is suitable for τ, if $[\![w, \tau]\!](p)$ exceeds N, for some $p \in Q$, then we can find another τ-suitable word w' such that $[\![w', \tau]\!]$ is less than or equal to $\widehat{N}$. This is stated in the following lemma, to which we will refer to as the bound lemma.

Lemma 3. *There exists some $N \geq 0$ such that, for each transition τ of $\mathfrak{A}$ and for each word $w \in S_\tau$, there exists a word $w' \in S_\tau$ such that the following holds:*

$$[\![w', \tau]\!] \leq \widehat{N} \tag{1}$$

$$[\![w', \tau]\!] \leq [\![w, \tau]\!] \tag{2}$$

For any $p \in Q$, if $[\![w, \tau]\!](p) > N$, then $[\![w', \tau]\!](p) > 0$. (3)

In essence, the lemma asserts that, if a transition τ can be applied by means of the word w, then we can replace w with another word w' such that, for each state $p \in Q$, the sufficient number of distinct trees evaluating to p that are needed to apply τ by means of w' exceeds neither N nor the corresponding number when w is used instead of w'. The third condition in the bound lemma is needed for technical reasons; it asserts that if a component $p \in Q$ in $[\![w, \tau]\!]$ exceeds N, then it must occur in w'.

Before we sketch our method of finding such a bound, let us first introduce some further notations. We recall that a word $w \in Q^+$ is suitable for τ if, given plenty of distinct trees for each state occurring in w, the transition τ can be applied. Now, given a set $R \subseteq Q$ and a tuple $\bar{d} \in \mathbb{N}^{|R|}$, the word w is said to be *suitable for τ with respect to R and $\bar{d}$* if the transition τ can be applied under the assumption that for each state p occurring in w: (a) there are $\bar{d}(p)$ many distinct trees that evaluate to p, if $p \in R$, and (b) there are plenty of (i.e., sufficiently many) distinct trees that evaluate to p, if $p \notin R$. We denote the set of all words that are suitable for τ with respect to R and $\bar{d}$ by $S_{\tau,R,\bar{d}}$.

Lemma 4. *The sets S_τ and $S_{\tau,R,\bar{d}}$, for all $R \subseteq Q$ and $\bar{d} \in \mathbb{N}^{|R|}$, are regular subsets of Q^+. In particular, the nonemptiness problem for these sets is decidable.*

Proof (sketch). Roughly speaking, a word w belongs to S_τ iff it belongs to L and the constraints in α do not cause conflicts in w; for instance, any pair (κ, λ) of positions in w satisfying a $\forall_{\mathsf{EQ}}$-constraint of τ may not satisfy any $\forall_{\mathsf{NEQ}}$-constraint of τ. In addition, since $\mathfrak{A}$ is supposed to be deterministic, if a pair of positions is declared to have equal subtrees by α, then the Q-labels of those positions must be equal. Since atomic constraints are built from MSO-formulas, we can write an MSO-formula that captures all these requirements, which shows the regularity of S_τ. To show the regularity of $S_{\tau,R,\bar{d}}$, we just need to additionally require that the occurrences of $p \in R$ can be partitioned into $\bar{d}(p)$-many sets of positions such that this partitioning does not cause conflict in w; for instance, if a pair (κ, λ) of positions in w that are labeled with a state from R satisfies a $\forall_{\mathsf{EQ}}$-constraint, then both positions must lie in the same partition. □

Let us now illustrate our method of finding the desired bound for a fixed transition τ by means of a simple example. Suppose $q_1, \ldots, q_4$ are the states of

$\mathfrak{A}$ and q_1 is the target state of τ, and suppose $v \in S_\tau$, say, with $[\![v, \tau]\!] = (2, 4, 5, 1)$. Then, $[\![v, \tau]\!]$ already gives a first approximation of the bound, namely 5. That is, if, for each state, there are five distinct trees that evaluate to this state, then we can apply τ (using v). Now, what happens if there are actually, say, only one tree for q_1 and three distinct trees for q_2? Then, two cases may occur. First, it might be the case that τ cannot be applied at all, i.e., it is not possible to apply τ under these conditions (with only one tree for q_1 and three distinct trees for q_2). Otherwise, second, there is an application of τ under these conditions, for example, using a word v', six distinct trees for q_3, and seven distinct trees for q_4. In the latter case, the bound must then be updated to 7 in order to make certain that we do not miss out any possibility of applying τ.

Recapitulating, we proceed for a fixed transition τ as follows:

(i) Start with an initial bound N_τ.
(ii) Check, for all subsets R of Q and all tuples $\bar{d} \in \mathbb{N}^{|R|}$ with $\bar{d} \leq \widehat{N_\tau}$, whether the set $S_{\tau,R,\bar{d}}$ is nonempty (which is possible due to Lemma 4).
(iii) In this case, pick a word $v_{\tau,R,\bar{d}}$, which then gives a new approximation of the desired bound via $[\![v_{\tau,R,\bar{d}}, \tau]\!]$, and update N_τ accordingly.
(iv) Go back to (ii).

Upon termination of this procedure, which turns out to be non-trivial and relies on Dickson's Lemma [10] (see also [5, Lemma 3]), the value of N_τ gives the desired bound with respect to the particular transition τ. For the bound required by Lemma 3, we then take the maximum of the bounds N_τ among all the transitions τ of $\mathfrak{A}$. For more details, in particular, concerning the termination and the correctness of the procedure, the reader is referred to [12, Section 4.3].

We now present an algorithm that, given a deterministic UTACS, decides whether the corresponding tree language is nonempty. In essence, this algorithm is an adaptation of the standard marking algorithm: it consists of a main loop that in each round collects, for each state, a tree resulting from the application of a transition based on the trees collected from the previous rounds. The bound from Lemma 3 gives the sufficient number of distinct trees that we ought to collect for each state; the main loop is iterated until either we cannot construct new trees anymore, or we have collected, for each state, as many trees as the bound. Hence, the algorithm eventually terminates.

Algorithm 5. Given a deterministic UTACS $\mathfrak{A} = (Q, \Sigma, \Lambda, \Delta, F)$, together with the bound N from Lemma 3, the algorithm NONEMPTY($\mathfrak{A}$) decides whether $T(\mathfrak{A}) \neq \emptyset$ holds. The tuple $\bar{d} \in \mathbb{N}^{|Q|}$ keeps track of the number of trees we have collected so far; for each state $q \in Q$, we use T_q to store the trees evaluating to q, so at any time of the computation $\bar{d}(q)$ contains the current value of $|T_q|$.

1: **function** NONEMPTY($\mathfrak{A}$)
2: initialize each T_q with $\{a \in \Sigma \mid (a, q) \in \Lambda\}$
3: **repeat**
4: **if** there exist $\tau = (L, \alpha, a, q) \in \Delta$, $w = q_1 \dots q_m \in S_{\tau,Q,\bar{d}}$, and $t_1 \in T_{q_1}$, $\dots, t_m \in T_{q_m}$ such that w and $t_1 \dots t_m$ satisfy α, and $|T_q| < N$
5: **then** $T_q \leftarrow T_q \cup \{a(t_1 \dots t_m)\}$

6: **until** no new tree can be constructed, or we have $\bar{d} = \widehat{N}$
7: **if** $T_q \neq \emptyset$ for some $q \in F$ **then return** '$T(\mathfrak{A}) \neq \emptyset$'
8: **else return** '$T(\mathfrak{A}) = \emptyset$'

The algorithm is sound since in Line 3–6 trees are constructed according to the transition relation Δ of $\mathfrak{A}$. The completeness of the algorithm (i.e., if $T(\mathfrak{A})$ is nonempty, then its output must be '$T(\mathfrak{A}) \neq \emptyset$') follows immediately from Lemma 6 below, which asserts that, if a tree t evaluates to a state q, then either we will eventually construct it, or we have already had N trees evaluating to q.

Lemma 6. *For any $t \in \mathcal{T}_\Sigma$ and $q \in Q$, if $t \to q$, then $t \in T_q$ or $|T_q| = N$ holds.*

We prove this lemma by an induction on the structure of t (cf. [12, Lemma 14]). The more interesting case is embodied by the induction step, i.e., the case $t = a(t_1 \dots t_m)$ with $t \to q$ and $t_i \to q_i$, for each $i = 1, \dots, m$, where t itself does not belong to T_q. Then, we have to construct N distinct trees evaluating to q out of the trees in $\bigcup_{s \in Q} T_s$. For this, we might have to replace the word $q_1 \dots q_m$ with one that satisfies the requirements of Lemma 3 in order to obtain trees evaluating to q that are actually constructed by Algorithm 5. In particular, the requirement (3) in the bound lemma ensures that we are indeed able to construct N such trees. To sum up, we obtain the main result of this section:

Theorem 7. *The nonemptiness problem for deterministic UTACS's is decidable.*

Throughout our algorithms and proofs, actually, we have just used the regularity of the sets of suitable words without really analyzing the form of the equality and disequality constraints appearing in the transitions at all. Hence, our method will still work if we vary the definition of the constraints, as long as the regularity of the sets of suitable words or, more generally, the decidability of the nonemptiness problem for these sets, is maintained.

As a remark, our method does not work for nondeterministic UTACS's. With determinism, we can assume that, if two trees evaluate to two different states, then these trees must be different as well; this property fails to hold for nondeterministic UTACS's. In fact, this observation has then lead us to define the notion of suitability of words over the set of states with respect to a transition, thereby reducing the analysis of the distinctness among trees to that among states.

5 Restrictions and Extensions of the Model

In this section, we indicate some possible variations of the automaton model we have introduced and discuss how far our results, in particular with respect to the decidability of the nonemptiness problem, are retained.

Sibling constraints without references to states. We recall that the atomic constraints between siblings we have used in the definition of UTACS are given by MSO-formulas over the state set of the underlying UTACS. In other words, the MSO-formulas used as constraints may refer to states when defining the pairs of

sibling subtrees that are supposed to be equal or distinct. In fact, the use of this ability has been demonstrated in Proposition 2 to show that the tree language given there is recognizable by a nondeterministic UTACS.

With this phenomenon in mind, we now prohibit the reference to states in the atomic constraints: the MSO-formulas used as atomic constraints now lack atomic MSO-formulas of the form $\chi_a(x)$. With this definition, we can then show that every nondeterministic restricted UTACS can be transformed into a deterministic one by using the standard subset construction.

Comparing the output of a tree transducer. When applying a transition, given a pair of siblings to be compared, what we do up to now is to check whether the subtrees below these positions are equal or not. A more involved processing is to feed the subtrees into a (deterministic) tree transducer and compare the output trees instead of the subtrees themselves, for example, if we want to compare only data values contained in the trees instead of the whole trees, as indicated in Section 1. Furthermore, if we consider bottom-up tree transducers, we can use MSO-formulas over the state set of the transducer instead of the state set of the underlying UTACS.

However, if we want to apply our method to solve the nonemptiness problem for the resulting automaton model, similar to our remark at the end of Section 4, we must require that if the states assumed by the transducer under consideration after producing two output trees, say, t and t', are different, then t and t' must also be different. Hence, a more restricted model of tree transducers, which are, in some sense, output deterministic, is required.

6 Conclusions

We have extended the tree automaton model defined by Bogaert and Tison in [1] to the case of unranked trees. In the transitions, we use MSO-formulas to address the pairs of positions to be compared. It then turns out that the nondeterministic model, in contrast to the ranked setting, is more expressive than the deterministic one. Our main result is that the nonemptiness problem for the latter model is decidable: we adapt the standard marking algorithm, which collects for each state a sufficient number of distinct trees. For this number, we define an appropriate bound by means of Lemma 3.

As far as the complexity of our nonemptiness decision procedure is concerned, there are two issues that still need to be settled. First, the termination of the bound algorithm is given by Dickson's Lemma, so its time complexity depends on the length of a certain antichain. Second, the complexity of our algorithms depends on the representation of the MSO-formulas that are used as constraints. It is known, however, that the translation from MSO-formulas to automata on words might involve a non-elementary blow-up. Thus, a careful choice of the representation of the sibling constraints is needed in order to analyze the complexity of our algorithms.

In addition to analyzing complexity, other prospective future work includes: (a) studying the nonemptiness problem for nondeterministic UTACS, (b) consid-

ering extensions that involve tree transducers, as indicated in Section 5, (c) considering automaton models that allow determinization, and (d) studying the connection to automata on words and trees with data considered in [3, 2].

References

1. Bogaert, B., Tison, S.: Equality and disequality constraints on direct subterms in tree automata. In Proc. STACS 1992. LNCS 577. Springer (1992) 161–171
2. Bojańczyk, M., David, C., Muscholl, A., Schwentick, T., Segoufin, L.: Two-variable logic on data trees and XML reasoning. In Proc. PODS 2006. ACM Press (2006) 10–19
3. 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) 7–16
4. Brüggemann-Klein, A., Murata, M., Wood, D.: Regular tree and regular hedge languages over unranked alphabets. Research Report HKUST-TCSC-2001-05, HKUST Theoretical Computer Science Center (2001) Available on http://hdl.handle.net/1783.1/738.
5. Burkart, O., Caucal, D., Moller, F., Steffen, B.: Verification on infinite structures. In Handbook of Process Algebra. Elsevier (2001) 545–623
6. 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) 436–449
7. Comon, H., Dauchet, M., Gilleron, R., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: Tree Automata Techniques and Applications. Available on http://www.grappa.univ-lille3.fr/tata (1997) Current release: 1st October 2002.
8. Dal Zilio, S., Lugiez, D.: XML schema, tree logic and sheaves automata. In Proc. RTA 2003. LNCS 2706. Springer (2003) 246–263
9. Dauchet, M., Caron, A.C., Coquidé, J.L.: Automata for reduction properties solving. Journal of Symbolic Computation **20** (1995) 215–233
10. Dickson, L.E.: Finiteness of the odd perfect and primitive abundant numbers with n distinct prime factors. American Journal of Mathematics **35** (1913) 413–422
11. Jacquemard, F., Rusinowitch, M., Vigneron, L.: Tree automata with equality constraints modulo equational theories. In Proc. IJCAR 2006. LNAI 4130. Springer (2006) 557–571
12. Karianto, W., Löding, C.: Unranked tree automata with sibling equalities and disequalities. Technical Report AIB-2006-13, RWTH Aachen (2006)
13. Libkin, L.: Logics for unranked trees: An overview. In Proc. ICALP 2005. LNCS 3580. Springer (2005) 35–50
14. Lugiez, D.: Counting and equality constraints for multitree automata. In Proc. FOSSACS 2003. LNCS 2620. Springer (2003) 328–342
15. Mongy-Steen, J.: Transformation de noyaux reconnaissables d'arbres. Forêts RATEG. PhD thesis, Université de Lille I (1981)
16. Neven, F.: Automata, logic, and XML. In Proc. CSL 2002. LNCS 2471. Springer (2002) 2–26
17. Seidl, H., Schwentick, T., Muscholl, A.: Numerical document queries. In Proc. PODS 2003. ACM Press (2003) 155–166
18. Seidl, H., Schwentick, T., Muscholl, A., Habermehl, P.: Counting in trees for free. In Proc. ICALP 2004. LNCS 3142. Springer (2004) 1136–1149
19. Tommasi, M.: Automates d'arbres avec tests d'égalité entre cousins germains. Technical report, Mémoire de DEA, Université de Lille I (1992)

Appendix

A A Normal Form for Transitions

Let $\mathfrak{A} = (Q, \Sigma, \Lambda, \Delta, F)$ be a deterministic UTACS. In this appendix, we consider some preprocessing of (the constraints of) the transitions of a (deterministic) UTACS, which leads to the following normal form:

Proposition 8. *Each (deterministic) UTACS is equivalent to one where each constraint occurring therein is a conjunction consisting of an* $\forall_{\mathsf{EQ}}$*-constraint* $\theta_{\forall_{\mathsf{EQ}}}$*, an* $\forall_{\mathsf{NEQ}}$*-constraint* $\theta_{\forall_{\mathsf{NEQ}}}$*,* $\exists_{\mathsf{EQ}}$*-constraints* $\varphi_1, \ldots, \varphi_k$*, and* $\exists_{\mathsf{NEQ}}$*-constraints* $\psi_1, \ldots, \psi_\ell$*.*

Proof. First, we show that we can assume, without loss of generality, that each constraint occurring in a transition is a conjunction of atomic constraints. Let α be a sibling constraint over Q; that is, α is a positive Boolean combination of atomic sibling constraints over Q (i.e., of $\forall_{\mathsf{EQ}}$-, $\forall_{\mathsf{NEQ}}$-, $\exists_{\mathsf{EQ}}$-, or $\exists_{\mathsf{NEQ}}$-constraints). Then, α can be transformed into a disjunction of conjunctions of atomic constraints, say $\alpha \equiv \alpha_1 \vee \cdots \vee \alpha_k$, for some $k \geq 1$, where each α_i is a conjunction of atomic constraints. Hence, given a transition (L, α, a, q) of $\mathfrak{A}$, we may split it into k transitions of the form (L, α_i, a, q), for each i with $1 \leq i \leq k$. Note that, although there may be more than one α_i that can be satisfied at once, all of these transitions lead to state q, so determinism is retained.

Now, given a conjunction of atomic constraints, we show that we can merge all $\forall_{\mathsf{EQ}}$-constraints and all $\forall_{\mathsf{NEQ}}$-constraints to one $\forall_{\mathsf{EQ}}$-constraint and one $\forall_{\mathsf{NEQ}}$-constraint, respectively. Let α be a conjunction of atomic sibling constraints over Q, and let φ and ψ be two $\forall_{\mathsf{EQ}}$-constraints therein. That is, $\varphi(x, y)$ and $\psi(x, y)$ are both MSO-formulas with two free variables. Then, for any word $w \in Q^+$ and any sequence $t_1 \ldots t_{|w|}$ of Σ-labeled trees, we have

$$
\begin{aligned}
& w \text{ and } t_1 \ldots t_{|w|} \text{ satisfy } \varphi \wedge \psi, \\
\text{iff} \quad & \text{for all } \kappa, \lambda \in \{1, \ldots, |w|\}, \text{ if } w \models \varphi(\kappa, \lambda), \text{ then } t_\kappa = t_\lambda, \\
& \qquad\qquad \text{and if } w \models \psi(\kappa, \lambda), \text{ then } t_\kappa = t_\lambda, \\
\text{iff} \quad & \text{for all } \kappa, \lambda \in \{1, \ldots, |w|\}, \text{ if } w \models \varphi(\kappa, \lambda) \vee \psi(\kappa, \lambda), \text{ then } t_\kappa = t_\lambda.
\end{aligned}
$$

Hence, we may replace $\varphi \wedge \psi$ with one single $\forall_{\mathsf{EQ}}$-constraint defined by $\theta(x, y) = \varphi(x, y) \vee \psi(x, y)$. Similarly, we may replace a conjunction of two $\forall_{\mathsf{NEQ}}$-constraints with one single $\forall_{\mathsf{NEQ}}$-constraint. □

B Determinism versus Nondeterminism

We present a tree language that is recognizable by a nondeterministic UTACS but not by any deterministic UTACS, thus justifying the claim of Proposition 2.

Let $\Sigma = \{a, b\}$, and let T be a tree language over Σ containing trees with the following properties. A tree t over Σ belongs to T iff

– $\mathrm{dom}_t \subseteq \{\varepsilon\} \cup \mathbb{N}\{1\}^*$,
– $t(\varepsilon) = a$ and $t(z) = b$, for any $z \in \mathrm{dom}_t \setminus \{\varepsilon\}$, and
– there exist $x, y \in \mathrm{dom}_t$ such that
  - $x, y \in \mathbb{N}$ and $1 < x < y < \max\{n \in \mathbb{N} \mid n \in \mathrm{dom}_t\}$ (i.e., y is not the most-right child of the root),
  - $t|_x = t|_y$, and
  - for all $z, z' \in \mathbb{N} \setminus \{x, y\}$ we have $t|_z = t|_{z'}$ and $t|_z \neq t|_x$.

Figure 2 illustrates a tree belonging to T. Intuitively, such a tree t consists of a root labeled with a and below it strands of b's. All but two of the b-strands are of the same length, and the two special b-strands themselves are of the same length.

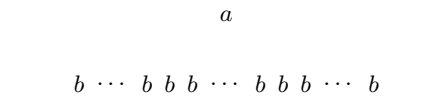

Fig. 2. A tree belonging to T (the dashed lines represent b-strands)

Proposition 9. *The tree language T can be recognized by a nondeterministic UTACS.*

Proof. The main idea for constructing a nondeterministic UTACS for T is to use a special state to mark the positions of the two special b-strands nondeterministically and to impose the appropriate constraints accordingly: the subtrees under the positions marked with the special state must be equal; all other subtrees must be equal among themselves; and the former and the latter trees must be different.

Formally, we define a nondeterministic UTACS with:

– the states q, p, and the final state q_{fin};
– the leaf transition (b, q);
– the inner-node transitions (q, true, b, q), (q, true, b, p), and $(q^+pq^+pq^+, \varphi \wedge \psi \wedge \theta, a, q_{\mathrm{fin}})$ with
  - $\exists_{\mathsf{EQ}}$-constraint $\varphi(x, y) = (1 < x \neq y < \max) \wedge \chi_p(x) \wedge \chi_p(y)$,
  - $\forall_{\mathsf{EQ}}$-constraint $\psi(x, y) = \chi_q(x) \wedge \chi_q(y)$, and
  - $\forall_{\mathsf{NEQ}}$-constraint $\theta(x, y) = (\chi_p(x) \wedge \chi_q(y)) \vee (\chi_q(x) \wedge \chi_p(y))$.

□

Proposition 10. *The tree language T cannot be recognized by any deterministic UTACS.*

Proof (sketch). Toward a contradiction, let us assume that a deterministic UTACS, say $\mathfrak{A} = (Q, \Sigma, \Lambda, \Delta, F)$, recognizes T. In order to simplify notation, let us refer to the set of transitions leading to some final state as Δ_{fin}. Furthermore, without loss of generality, we can assume that each constraint occurring in $\mathfrak{A}$ is a conjunction consisting of one $\forall_{\mathsf{EQ}}$-constraint, one $\forall_{\mathsf{NEQ}}$-constraint, several $\exists_{\mathsf{EQ}}$-constraints, and several $\exists_{\mathsf{NEQ}}$-constraints (see Proposition 8 in Appendix A). In order to simplify notation, we refer to the set of transitions leading to some final state as Δ_{fin}.

The idea of the proof is to construct a tree from T that cannot be accepted by $\mathfrak{A}$. For this, we will start from a tree belonging to T and then modify it step by step, thereby preventing the applicability of $\mathfrak{A}$'s transitions that lead to a final state.

First of all, since Q is finite, there is some state p of Q such that infinitely many b-strands of different length are evaluated to p. Let us pick three of them, say, s, t, and u. Now, for all integers $n_1, n_2, n_3 \geq 1$, we will consider the trees of the form

$$a(\underbrace{s \cdots s}_{n_1\text{-times}}\ t\ \underbrace{s \cdots s}_{n_2\text{-times}}\ t\ \underbrace{s \cdots s}_{n_3\text{-times}})\ . \tag{4}$$

By the definition of T, these trees belong to T and must hence be accepted by $\mathfrak{A}$. Thus, by the choice of s and t, for every $n_1, n_2, n_3 \geq 1$ a transition $\tau \in \Delta_{\text{fin}}$ with $p^{n_1+n_2+n_3+2} \in S_\tau$ must exist. Let us refer to the word $p^{n_1+n_2+n_3+2}$ as $w(n_1, n_2, n_3)$ and to the tree of the form (4) as $t(n_1, n_2, n_3)$.

Moreover, since the values of $n_1, n_2, n_3 \geq 1$ are arbitrary and, on the other hand, Δ_{fin} is finite, there must be a transition τ therein such that $|S_\tau \cap p^+|$ is infinite. At this point, we can forget about those transitions that lack this property as they cannot be applied to the trees of the form (4) anymore if n_1, n_2, n_3 are sufficiently large.

Avoiding $\forall_{\mathsf{NEQ}}$-constraints. Our next step is to abandon the possibilities of using transitions with $\forall_{\mathsf{NEQ}}$-constraints in order to accept trees of the form (4). More precisely, we want to choose the values of n_1, n_2, and n_3 such that for each transition with a $\forall_{\mathsf{NEQ}}$-constraint θ either

(a) θ is trivially fulfilled as there is no pair (κ, λ) of positions in $w(n_1, n_2, n_3)$ with $w(n_1, n_2, n_3) \models \theta(\kappa, \lambda)$, or
(b) θ is not fulfilled as there is some pair (κ, λ) of positions in $w(n_1, n_2, n_3)$ with $w(n_1, n_2, n_3) \models \theta(\kappa, \lambda)$ and $t(n_1, n_2, n_3)|_\kappa = t(n_1, n_2, n_3)|_\lambda$, which means that the underlying transition cannot be applied in order to accept $t(n_1, n_2, n_3)$.

Consider a transition $\tau \in \Delta_{\text{fin}}$ with a $\forall_{\mathsf{NEQ}}$-constraint θ. Let $P_{\tau,\theta} = \{p^n \in S_\tau \mid n \in \mathbb{N}$ and there are some positions κ, λ in p^n such that $p^n \models \theta(\kappa, \lambda)\}$.

If $P_{\tau,\theta}$ is finite, then we choose n_1, n_2, n_3 to be greater than $\max\{n \in \mathbb{N} \mid p^n \in P_{\tau,\theta}\}$, so the case (a) above occurs.

Otherwise, consider a word $p^n \in P_{\tau,\theta}$ with the respective positions κ and λ. Without loss of generality, let us assume $\kappa < \lambda$. Informally, the occurrences of κ

and λ can be illustrated as markers in the underlying word p^n, as (5) shows:

$$\begin{array}{ccccccccc} & & & \kappa & & \lambda & & & \\ p & \cdots & p & \cdots & p & \cdots & p & & \\ & & & \bullet & & \circ & & & \end{array} \tag{5}$$

Now, as S_τ is regular and θ is an MSO-formula, $P_{\tau,\theta}$ is regular. Thus, the standard pumping lemma for regular languages applies in the following sense: if n is sufficiently large, then there exists some 'period' $m \geq 1$ such that p^n can be decomposed into

$$\begin{array}{ccccccccc} & & \kappa & & \lambda & & & & \\ p & \cdots & p & \cdots & p & \cdots & p^m & \cdots & p \\ & & \bullet & & \circ & & & & \end{array} \tag{6}$$

or

$$\begin{array}{ccccccccc} & & & & \kappa & & \lambda & & \\ p & \cdots & p^m & \cdots & p & \cdots & p & \cdots & p \\ & & & & \bullet & & \circ & & \end{array} \tag{7}$$

or

$$\begin{array}{ccccccccc} & & \kappa & & & & \lambda & & \\ p & \cdots & p & \cdots & p^m & \cdots & p & \cdots & p \\ & & \bullet & & & & \circ & & \end{array} \tag{8}$$

and 'pumping' p^m always results in a word belonging to $P_{\tau,\theta}$ (with the respective markers κ' and λ'). Moreover, we can do this sufficiently often such that the resulting word can be decomposed into $p^{n_1}pp^{n_2}pp^{n_3}$ such that κ' and λ'

- both lie in the p^{n_1}-part (this corresponds to (6)), or
- both lie in the p^{n_3}-part (this corresponds to (7)), or
- lie in the p^{n_1}-part and p^{n_3}-part, respectively (this corresponds to (8)).

In either case, we have $t(n_1, n_2, n_3)|_{\kappa'} = t(n_1, n_2, n_3)|_{\lambda'}$ and $w(n_1, n_2, n_3) \models \theta(\kappa', \lambda')$. Hence, θ is not satisfied, so τ cannot be applied to accept $t(n_1, n_2, n_3)$.

If there is another transition τ' with a $\forall_{\mathsf{NEQ}}$-constraint, then we proceed as above: we start with a word $p^{n'}$ that 'avoids' τ with the respective positions κ' and λ', and we obtain a pumping period m'. Now, in order to avoid the applicability of the previous transition τ and the current transition τ', pumping must be done with respect to the least common multiple of m and m' as the period.

We proceed as above until each transition with a $\forall_{\mathsf{NEQ}}$-constraints falls into one of the cases (a) or (b). Note that the fact that we only consider p-labeled suitable words is crucial in order to avoid interference between the pumping processes above.

Satisfying $\exists_{\mathsf{EQ}}$-constraints and $\exists_{\mathsf{NEQ}}$-constraints. Suppose now that n_1, n_2, n_3 such that $t(n_1, n_2, n_3)$ can only be accepted by using some transition from Δ_{fin} such that either α contains no $\forall_{\mathsf{NEQ}}$-constraint, or α's $\forall_{\mathsf{NEQ}}$-constraint is trivially satisfied. Thus, let us fix such a transition $\tau = (L, \alpha, a, q) \in \Delta_{\text{fin}}$ and let us assume

$$\alpha = \theta \wedge \bigwedge_{i=1}^{k} \varphi_i \wedge \bigwedge_{j=1}^{\ell} \psi_\ell \tag{9}$$

where θ is a $\forall_{\mathsf{EQ}}$-constraint, $\varphi_1, \dots \varphi_k$ are $\exists_{\mathsf{EQ}}$-constraints, and $\psi_1, \dots, \psi_\ell$ are $\exists_{\mathsf{NEQ}}$-constraints. Moreover, the applicability of τ implies that each of these constraints is satisfied.

Let us first consider the $\exists_{\mathsf{EQ}}$-constraint φ_1. As φ_1 is satisfied, there exist some $\kappa_1, \lambda_1 \in \{1, \dots, n_1 + n_2 + n_3 + 2\}$ such that $w(n_1, n_2, n_3) \models \varphi_1(\kappa_1, \lambda_1)$ and $t(n_1, n_2, n_3)|_{\kappa_1} = t(n_1, n_2, n_3)|_{\lambda_1}$. Now, following the same argumentation as with avoiding $\forall_{\mathsf{NEQ}}$-constraints, we can replace n_1, n_2, n_3 with appropriate n_1^1, n_2^1, n_3^1 such that κ_1' and λ_1'

- both lie at the very beginning of the $p^{n_1^1}$-part, or
- both lie at the very end of the $p^{n_3^1}$-part, or
- lie at the very beginning of the $p^{n_1^1}$-part and at the very end of the $p^{n_3^1}$-part, respectively.

Note that the terms 'very beginning' and 'very end' are meant to be relative to the length of the resulting word $w(n_1^1, n_2^1, n_3^1)$. In either case, we have $t(n_1^1, n_2^1, n_3^1)|_{\kappa_1'} = t(n_1^1, n_2^1, n_3^1)|_{\lambda_1'}$ and $w(n_1^1, n_2^1, n_3^1) \models \theta(\kappa_1', \lambda_1')$. Hence, φ_1 is still satisfied.

Starting with n_1^1, n_2^1, n_3^1, we next consider φ_2 and obtain n_1^2, n_2^2, n_3^2. Repeating this procedure for all the remaining $i = 3, \dots, k$, we finally obtain n_1^k, n_2^k, n_3^k such that all $\exists_{\mathsf{EQ}}$-constraints are satisfied, and moreover, all pairs of positions that are used to satisfy these constraints occur only at the very beginning or at the very end of the word $w(n_1^k, n_2^k, n_3^k)$.

Let us now consider the $\exists_{\mathsf{NEQ}}$-constraint ψ_1. As ψ_1 is satisfied, there exist some μ_1, ν_1 such that $w(n_1^k, n_2^k, n_3^k) \models \psi_1(\mu_1, \nu_1)$ and $t(n_1^k, n_2^k, n_3^k)|_{\mu_1} \neq t(n_1^k, n_2^k, n_3^k)|_{\nu_1}$. By the choice of $t(n_1^k, n_2^k, n_3^k)$ as a tree of the form (4), either one of the subtrees at positions μ_1 and ν_1 must be one of the subtrees t, and the other one must be s. Without loss of generality, let us assume that μ_1 is the position of the left subtree t and that the subtree at ν_1 is s, as illustrated below:

$$\begin{array}{cccccccccc} & & & & \mu_1 & & & & \nu_1 & \\ s & \cdots & s & t & s & \cdots & s & t & s & \cdots & s \end{array} \tag{10}$$

Using a similar pumping argument as before, we can replace n_1^k, n_2^k, n_3^k with m_1^1, m_2^1, m_3^1 such that in the resulting p-labeled word there is a pair of positions μ_1' and ν_1' such that

- the subtree at position μ_1' is the left subtree t, and

– the subtree at position ν_1' is s, and ν_1' is either very far from μ_1'

$$\begin{array}{cccccccccccccc} & & & & \mu_1' & & & & & & & & \nu_1' & \\ s & \cdots\cdots & s & t & s & \cdots\cdots & s & t & s & \cdots\cdots & s \end{array}$$

or very near to μ_1':

$$\begin{array}{cccccccccccc} & & & & \mu_1' & \nu_1' & & & & & & \\ s & \cdots\cdots & s & t & s & \cdots\cdots & s & t & s & \cdots\cdots & s \end{array}$$

Here, the terms 'far' and 'near' are meant to be relative to the length of the resulting p-labeled word.

In either case, we have $t(m_1^1, m_2^1, m_3^1)|_{\mu_1'} \neq t(m_1^1, m_2^1, m_3^1)|_{\nu_1'}$ and $w(m_1^1, m_2^1, m_3^1) \models \psi_1(\mu_1', \nu_1')$. Thus, ψ_1 is satisfied.

As with the $\exists_{\mathsf{EQ}}$-constraints, we repeat this procedure for all the remaining $\exists_{\mathsf{NEQ}}$-constraints.

Finally, we obtain m_1^k, m_2^k, m_3^k such that all $\exists_{\mathsf{EQ}}$-constraints and all $\exists_{\mathsf{NEQ}}$-constraints are satisfied, and moreover, all positions that are used to satisfy these constraints either lie

– at the very beginning of $w(m_1^k, m_2^k, m_3^k)$, or
– at the very end of $w(m_1^k, m_2^k, m_3^k)$, or
– very near to $m_1^k + 1$ (the position of the left subtree t), or
– very near to $m_2^k + 2$ (the position of the right subtree t).

Again, the terms 'very beginning', 'very end', and 'near' are meant to be relative to the length of the resulting word $w(m_1^k, m_2^k, m_3^k)$. This word and the corresponding positions are illustrated in Figure 3.

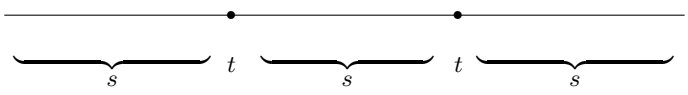

Fig. 3. An illustration of the word $w(m_1^k, m_2^k, m_3^k)$ (and of $w(m_1, m_2, m_3)$); the gray areas roughly mark the positions that are used to satisfy the $\exists_{\mathsf{EQ}}$-constraints and $\exists_{\mathsf{NEQ}}$-constraints; the bullets represent the positions where t occurs.

Handling the $\forall_{\mathsf{EQ}}$-constraint. Before we turn to the $\forall_{\mathsf{EQ}}$-constraint θ of (9), we observe that the area between the second and the third gray area in Figure 3 can be made arbitrarily large by replacing m_1^k, m_2^k, m_3^k with the appropriate m_1, m_2, m_3. This is due to the fact that there are only finitely many $\exists_{\mathsf{EQ}}$-constraints and $\exists_{\mathsf{NEQ}}$-constraints. Therefore, in the following consideration we

will assume that m_1, m_2, m_3 are chosen in such away that the gap between the second and the third gray area is as large as we will need it to be.

Let us now consider the $\forall_{\mathsf{EQ}}$-constraint θ of (9). Further, let κ and λ be the position of the subtrees t in $w(m_1, m_2, m_3)$.

If $w(m_1, m_2, m_3) \not\models \theta(\kappa, \lambda)$, then we can replace the subtree at position λ with u without affecting the satisfaction of θ nor of the other constraints:

- the $\exists_{\mathsf{EQ}}$-constraints are still satisfied since the positions that are used to satisfy them do not include λ;
- the $\exists_{\mathsf{NEQ}}$-constraints are still satisfied since the satisfaction of these constraints relies upon the fact that the tree at position λ, namely t, is different from the trees positions other than κ, namely s, and this is still the case with u at position λ.

As a result, the tree

$$a(\underbrace{s \cdots s}_{m_1\text{-times}}\ t\ \underbrace{s \cdots s}_{m_2\text{-times}}\ u\ \underbrace{s \cdots s}_{m_3\text{-times}})\ ,$$

which does not belong to T, will be accepted by $\mathfrak{A}$ as well, a contradiction.

If $w(m_1, m_2, m_3) \models \theta(\kappa, \lambda)$, then, as with $\forall_{\mathsf{NEQ}}$-constraints, we can consider the positions of κ and λ in $w(m_1, m_2, m_3)$ to be labeled with some markers, as depicted in Figure 4. Again, as $P_{\tau,\theta}$ is regular, and if the gap between the second

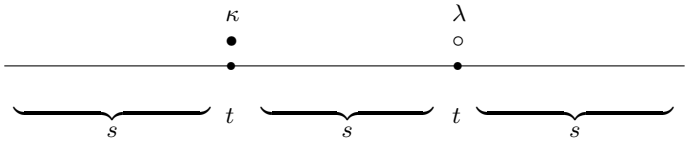

Fig. 4. The word $w(m_1, m_2, m_3)$ with the marked positions κ and λ

and the third gray area is sufficiently large, then we can find a position μ within this gap, such that $w(m_1, m_2, m_3) \models \theta(\kappa, \mu)$ or $w(m_1, m_2, m_3) \models \theta(\mu, \lambda)$ holds. In either case, the subtree at position μ must then be t, a contradiction. □

C Proof of the Bound Lemma

In this appendix, we give a detailed proof of Lemma 3. Throughout this appendix, let $\mathfrak{A} = (Q, \Sigma, \Delta, \Lambda, F)$ be a deterministic UTACS, and unless stated otherwise, let $\tau = (L, \alpha, a, q)$ be a transition of $\mathfrak{A}$.

C.1 Suitable Words: Regularity and Restrictions

Let us recall a remark on τ-suitable words that we have made in Section 4:

Remark 11. For any word $w \in Q^+$, in general, $[\![w, \tau]\!](p)$, for each $p \in Q$, does not need to exceed $|w|$.

In this subsection, we show that, for each transition of $\mathfrak{A}$, the set of words that are suitable for it is regular. Moreover, if we impose some further restrictions on this set, which we will introduce later on in this subsection, the resulting sets are still regular.

Lemma 12. *The set $S_\tau \subseteq Q^+$ is regular.*[1]

Proof. By Proposition 8, we can assume that α is a conjunction of an $\forall_{\mathsf{EQ}}$-constraint $\theta_{\forall_{\mathsf{EQ}}}$, an $\forall_{\mathsf{NEQ}}$-constraint $\theta_{\forall_{\mathsf{NEQ}}}$, $\exists_{\mathsf{EQ}}$-constraints $\varphi_1, \ldots, \varphi_k$, and $\exists_{\mathsf{NEQ}}$-constraints $\psi_1, \ldots, \psi_\ell$.

Roughly speaking, a word $w \in Q^+$ is suitable for τ iff it belongs to L and the constraints in α do not cause conflicts in w; for instance, if a pair (κ, λ) of positions in w satisfies $\theta_{\forall_{\mathsf{EQ}}}$, then it should not satisfy $\theta_{\forall_{\mathsf{NEQ}}}$. Furthermore, since $\mathfrak{A}$ is supposed to be deterministic, if a pair of positions is declared to have equal subtrees by α, then the Q-labels of those positions must be equal.

More precisely, a word $w \in Q^+$ is suitable for τ iff all of the following requirements are met:

1. The word w belongs to L.
2. There exist some pairs of positions in w, say, (x_1, y_1), $\ldots$, (x_k, y_k), (x'_1, y'_1), $\ldots$, (x'_ℓ, y'_ℓ), such that
   - the sets $\{(x_1, y_1), \ldots, (x_k, y_k)\}$ and $\{(x'_1, y'_1), \ldots, (x'_\ell, y'_\ell)\}$ do not overlap, that is, for each $i = 1, \ldots, k$ and $j = 1, \ldots, \ell$,

     $$\{x_i, y_i\} \neq \{x'_j, y'_j\} \ ,$$

     which stands as an abbreviation for

     $$\neg(x_i = x'_j \wedge y_i = y'_j) \wedge \neg(x_i = y'_j \wedge y_i = x'_j) \ ,$$

     is satisfied;
   - for each $i = 1, \ldots, k$, the pair (x_i, y_i) satisfies

     $$\varphi_i(x_i, y_i) \wedge \neg(\theta_{\forall_{\mathsf{NEQ}}}(x_i, y_i) \vee \theta_{\forall_{\mathsf{NEQ}}}(y_i, x_i)) \wedge \bigwedge_{p \in Q} (\chi_p(x_i) \leftrightarrow \chi_p(y_i)) \ ;$$

   - for each $j = 1, \ldots, \ell$, the pair (x'_j, y'_j) satisfies

     $$\psi_j(x'_j, y'_j) \wedge \neg(\theta_{\forall_{\mathsf{EQ}}}(x'_j, y'_j) \vee \theta_{\forall_{\mathsf{EQ}}}(y'_j, x'_j)) \wedge \neg(x'_j = y'_j) \ .$$

[1] This lemma appears as a part of Lemma 4 in the body of this paper.

3. For each pair (x, y) of positions in w, the formulas

$$\theta_{\forall_{\mathsf{EQ}}}(x,y) \to \neg(\theta_{\forall_{\mathsf{NEQ}}}(x,y) \vee \theta_{\forall_{\mathsf{NEQ}}}(y,x)) \wedge \bigwedge_{p\in Q} (\chi_p(x) \leftrightarrow \chi_p(y))$$

and

$$\theta_{\forall_{\mathsf{NEQ}}}(x,y) \to \neg(\theta_{\forall_{\mathsf{EQ}}}(x,y) \vee \theta_{\forall_{\mathsf{EQ}}}(y,x)) \wedge \neg(x = y)$$

are satisfied.

It is not difficult to write an MSO-sentence that captures all these requirements, so we conclude that S_τ is indeed regular. □

By adapting the proof of Lemma 12, we can show that the set $S_{\tau,R,\bar{d}}$, for all $R \subseteq Q$ and $\bar{d} \in \mathbb{N}^{|R|}$, is regular, too.

Lemma 13. *The set $S_{\tau,R,\bar{d}} \subseteq Q^+$ is regular.*[2]

Proof. Again, by Proposition 8, we can assume that α is a conjunction of an $\forall_{\mathsf{EQ}}$-constraint $\theta_{\forall_{\mathsf{EQ}}}$, an $\forall_{\mathsf{NEQ}}$-constraint $\theta_{\forall_{\mathsf{NEQ}}}$, $\exists_{\mathsf{EQ}}$-constraints $\varphi_1, \ldots, \varphi_k$, and $\exists_{\mathsf{NEQ}}$-constraints $\psi_1, \ldots, \psi_\ell$.

Here, we need a finer analysis of the suitable words. A word $w \in S_\tau$ respects R and $\bar{d}$ iff the occurrences of $p \in R$ can be partitioned into $\bar{d}(p)$-many sets of positions, and moreover, this partitioning may not cause conflict in w. As an illustration, if a pair (κ, λ) of positions in w satisfy $\theta_{\forall_{\mathsf{EQ}}}$, and if they are labeled with a state from R, then both positions must lie in the same partition.

In other words, a word $w \in Q^+$ is suitable for τ with respect to R and $\bar{d}$ iff all of the following requirements are met (to simplify our presentation, we write d_p instead of $\bar{d}(p)$ below):

1. The word w belongs to L.
2. There exists a family of sets of positions in w, say $(C^p_{j_p})_{p\in R, j_p=1,\ldots,d_p}$, such that:
   (a) for each $p \in R$, the sets $C^p_1, \ldots, C^p_{d_p}$ define a partitioning on the set of positions that are labeled with p:

$$\forall x \quad \bigwedge_{p\in R} \left[\chi_p(x) \to \bigvee_{j_p=1}^{d_p} \left(C^p_{j_p}(x) \wedge \bigwedge_{k_p \in \{1,\ldots,d_p\}\setminus\{j_p\}} \neg C^p_{k_p}(x) \right) \right]$$
$$\wedge \bigwedge_{p\in R} \bigwedge_{j_p=1}^{d_p} \left[C^p_{j_p}(x) \to \chi_p(x) \right]$$

   (b) there exist some pairs of positions in w, say, (x_1, y_1), $\ldots$, (x_k, y_k), (x'_1, y'_1), $\ldots$, (x'_ℓ, y'_ℓ), such that

[2] This lemma appears as a part of Lemma 4 in the body of this paper.

– the sets $\{(x_1, y_1), \ldots, (x_k, y_k)\}$ and $\{(x'_1, y'_1), \ldots, (x'_\ell, y'_\ell)\}$ do not overlap, that is, for each $i = 1, \ldots, k$ and $j = 1, \ldots, \ell$,

$$\{x_i, y_i\} \neq \{x'_j, y'_j\}$$

is satisfied;

– for each $i = 1, \ldots, k$, the pair (x_i, y_i) satisfies

$$\begin{aligned}\varphi_i(x_i, y_i) \wedge \neg(&\theta_{\forall_{\mathsf{NEQ}}}(x_i, y_i) \vee \theta_{\forall_{\mathsf{NEQ}}}(y_i, x_i)) \\ &\wedge \bigwedge_{p \in Q \setminus R} (\chi_p(x_i) \leftrightarrow \chi_p(y_i)) \\ &\wedge \bigwedge_{p \in R} \bigwedge_{j_p=1}^{d_p} (C^p_{j_p}(x_i) \leftrightarrow C^p_{j_p}(y_i)) \ ;\end{aligned}$$

– for each $j = 1, \ldots, \ell$, the pair (x'_j, y'_j) satisfies

$$\begin{aligned}\psi_j(x'_j, y'_j) \wedge \neg(&\theta_{\forall_{\mathsf{EQ}}}(x'_j, y'_j) \vee \theta_{\forall_{\mathsf{EQ}}}(y'_j, x'_j)) \\ &\wedge \neg(x'_j = y'_j) \\ &\wedge \bigwedge_{p \in R} \bigwedge_{j_p=1}^{d_p} \neg(C^p_{j_p}(x'_j) \wedge C^p_{j_p}(y'_j)) \ ;\end{aligned}$$

(c) for each pair (x, y) of positions in w, the formulas

$$\begin{aligned}\theta_{\forall_{\mathsf{EQ}}}(x, y) \rightarrow \neg(&\theta_{\forall_{\mathsf{NEQ}}}(x, y) \vee \theta_{\forall_{\mathsf{NEQ}}}(y, x)) \\ &\wedge \bigwedge_{p \in Q \setminus R} (\chi_p(x) \leftrightarrow \chi_p(y)) \\ &\wedge \bigwedge_{p \in R} \bigwedge_{j_p=1}^{d_p} (C^p_{j_p}(x) \leftrightarrow C^p_{j_p}(y))\end{aligned}$$

and

$$\begin{aligned}\theta_{\forall_{\mathsf{NEQ}}}(x, y) \rightarrow \neg(&\theta_{\forall_{\mathsf{EQ}}}(x, y) \vee \theta_{\forall_{\mathsf{EQ}}}(y, x)) \\ &\wedge \neg(x = y) \\ &\wedge \bigwedge_{p \in R} \bigwedge_{j_p=1}^{d_p} \neg(C^p_{j_p}(x) \wedge C^p_{j_p}(y))\end{aligned}$$

are satisfied.

It is now not difficult to write an MSO-sentence that captures all these requirements, so we conclude that $S_{\tau,R,\bar{d}}$ is indeed regular. □

Remark 14. If $\bar{e} \in \mathbb{N}^{|R|}$ is a tuple of natural numbers with $\bar{d} \leq \bar{e}$, then we have $S_{\tau,R,\bar{d}} \subseteq S_{\tau,R,\bar{e}}$.

In order to deal with the third condition of Lemma 3, we introduce a further restriction of the sets of suitable words. Let M be a subset of Q. We denote by $S_{\tau,M}$ and $S_{\tau,R,\bar{d},M}$ the restriction of both sets, respectively, to those words in which each state in M occurs at least once. As the former sets are regular, the latter sets also are. Also, Remark 14 still holds for the restriction to M: given a tuple $\bar{e} \in \mathbb{N}^{|R|}$ with $\bar{d} \leq \bar{e}$, we have $S_{\tau,R,\bar{d},M} \subseteq S_{\tau,R,\bar{e},M}$.

Note that all the sets of suitable words we have introduced so far have been shown to be regular. Hence, it is decidable whether these sets are empty or not.

C.2 Finding the Bound

In this subsection, we present an algorithm for finding the bound

- for a fixed transition $\tau = (L, \alpha, a, q) \in \Delta$ of $\mathfrak{A}$, for which we can assume, without loss of generality, that S_τ is not empty (otherwise, we can remove it from $\mathfrak{A}$ without affecting the language recognized by $\mathfrak{A}$);
- for a fixed nonempty subset M of Q in order to incorporate the third condition of the bound lemma.

Later, we will take the maximum of all the bounds over all transitions of $\mathfrak{A}$ and all nonempty subsets of Q as the bound for $\mathfrak{A}$.

Let us fix the notation we are going to use in the algorithm. We refer to $|Q|$, the number of states of $\mathfrak{A}$, as m. Let $n \geq 0$ be a natural number. We denote by $\mathbb{N}_{\leq n}$ the set of natural numbers that are less than or equal to n, and we refer to the set of m-tuples built from $\mathbb{N}_{\leq n}$ as $(\mathbb{N}_{\leq n})^m$. Given a set $R \subseteq Q$ and a tuple $\bar{z} \in \mathbb{N}^m$, we denote by $\bar{z}{\restriction_R}$ the restriction of $\bar{z}$ with respect to R, that is, $\bar{z}{\restriction_R}\colon R \to \mathbb{N}$ is an $|R|$-tuple with $\bar{z}{\restriction_R}(p) = \bar{z}(p)$, for all $p \in R$, and $\bar{z}{\restriction_R}(p)$ is undefined for all $p \in Q \setminus R$. Likewise, given a set $I \subseteq (\mathbb{N}_{\leq n})^m$ of m-tuples, the restriction of I with respect to R is defined as $I{\restriction_R} = \{\bar{z}{\restriction_R} \mid \bar{z} \in I\}$.

Algorithm 15. Given a deterministic UTACS $\mathfrak{A} = (Q, \Sigma, \Delta, \Lambda, F)$, a transition τ of $\mathfrak{A}$, and a set $M \subseteq Q$ of states, the algorithm returns a natural number that is supposed to be the bound of $\mathfrak{A}$ with respect to τ and M.

1: **function** BOUND($\mathfrak{A}, \tau, M$)
2: $m \leftarrow |Q|$
3: **if** $S_{\tau,M} = \emptyset$ **then return** 0
4: get a word $u_{\tau,M}$ from $S_{\tau,M}$
5: $n \leftarrow |u_{\tau,M}|$ // see Remark 11
6: $I \leftarrow (\mathbb{N}_{\leq n})^m$ // I contains the m-tuples to be checked
7: **for all** $R \subseteq Q$ **do**
8: $I_R^+ \leftarrow \emptyset$ // I_R^+ and I_R^- contain the tuples $\bar{e}$ from $I{\restriction_R}$ that
9: $I_R^- \leftarrow \emptyset$ // have proven successful (i.e., $S_{\tau,R,\bar{e},M} \neq \emptyset$)
// and unsuccessful (i.e., $S_{\tau,R,\bar{e},M} = \emptyset$), respectively
10: **while** $I \neq \emptyset$ **do**
11: get a tuple $\bar{z}$ from I and remove it from I
12: **for all** $R \subseteq Q$ **do**

```
13:          d̄ ← z̄↾_R
14:          if there is no ē ≤ d̄ with ē ∈ I_R^+ then
                                  // neither d̄ nor any ē ≤ d̄ has proven successful
15:              if there is no ē ≥ d̄ with ē ∈ I_R^- then
                                  // neither d̄ nor any ē ≥ d̄ has proven unsuccessful
16:                  if S_{τ,R,d̄,M} = ∅ then
17:                      I_R^- ← I_R^- ∪ {d̄}          // d̄ has proven unsuccessful
18:                  else
19:                      I_R^+ ← I_R^+ ∪ {d̄}          // d̄ has proven successful
20:                      get a word v_{τ,R,d̄,M} from S_{τ,R,d̄,M}
(*)                  n' ← max{n, |v_{τ,R,d̄,M}|}       // update n and
21:                  I ← I ∪ ((ℕ_{≤n'})^m \ (ℕ_{≤n})^m)   // put the new tuples
22:                  n ← n'                           // into I
23:
24:      return n
```

Note, first, that the choice of $u_{\tau,M}$ and $v_{\tau,R,\bar{d},M}$ in Line 4 and 20, respectively, is not unique and, second, that the time complexity of the algorithm depends on this choice.

Lemma 16. *For each $\mathfrak{A}$, τ, and M, the algorithm* Bound$(\mathfrak{A}, \tau, M)$ *terminates.*

Proof. Toward a contradiction, suppose that the computation of Bound$(\mathfrak{A}, \tau, M)$ does not terminate. This means that the **while**-loop in Line 10–23 of the algorithm is executed infinitely often.

Line 11 ensures that after we have fetched a tuple from I, we also remove it from I, and Line 22 ensures that only new tuples are put into I. Consequently, in every execution of the **while**-loop we will always get a new tuple to consider.

The fact that the **while**-loop is executed infinitely often implies that new tuples are added to I infinitely often; otherwise, I would eventually be empty and the computation would eventually terminate. Putting new tuples into I is done in Line 22, so this line and, more generally, Line 19–23 (the part marked with $(*)$) are executed infinitely often.

Furthermore, as there are only finitely many subsets of Q, there is one particular subset $R \subseteq Q$ chosen in Line 12 for which $(*)$ is executed infinitely often. Let us now restrict our attention to these particular iterations of $(*)$. Let $\bar{z}_1, \bar{z}_2, \bar{z}_3, \cdots \in \mathbb{N}^m$ be the (infinitely many) tuples from I where, for each $i \geq 1$, the tuple $\bar{z}_i$ corresponds to the tuple that is fetched in Line 11 when $(*)$ is executed for the i-th time with the choice of R in Line 12. Note that for each $i \geq 1$ the i-th iteration puts $\bar{z}_i{\restriction}_R$ into I_R^+. Now, by Dickson's Lemma [10] (see also [5, Lemma 3]) there exist some i and j with $i < j$ such that $\bar{z}_i \leq \bar{z}_j$, which means, in particular, that $\bar{z}_i{\restriction}_R \leq \bar{z}_j{\restriction}_R$. In the j-th iteration, however, $\bar{z}_i{\restriction}_R$ belongs to I_R^+, so the condition of the **if**-statement in Line 14 is violated. Thus, $(*)$ is not executed, a contradiction. □

Proof of Lemma 3. We now give a detailed proof of the bound lemma. Let $\mathfrak{A} = (Q, \Sigma, \Delta, \Lambda, F)$ be a deterministic UTACS. To begin with, let us define the

bound N to be

$$\max\{\textsc{Bound}(\mathfrak{A}, \tau, M) \mid \tau \in \Delta, M \subseteq Q\} \ .$$

Let τ be a transition of $\mathfrak{A}$ and w be a word in S_τ. Our task is now to find a word $w' \in S_\tau$ that meets the three requirements given in the lemma. In particular, if τ can be applied by using w, then this must also hold for w'.

If $[\![w, \tau]\!] \leq \widehat{N}$, then we are done: we just need to take w as w', and all three requirements of the lemma are trivially met.

Otherwise, there exist some states for which the corresponding values of $[\![w, \tau]\!]$ exceed N. Let $M \subseteq Q$ be the (nonempty) set of these states; that is,

$$M = \{p \in Q \mid [\![\tau, w]\!](p) > N\} \ .$$

We now consider the computation of $\textsc{Bound}(\mathfrak{A}, \tau, M)$, that is, the computation of Algorithm 15 with $\mathfrak{A}$, τ, and M as its inputs.

By the definition of suitable sets, the word w belongs to the set $S_{\tau,M}$, so this set is not empty. In particular, we obtain the word $u_{\tau,M}$ from Line 4. If $[\![u_{\tau,M}, \tau]\!] \leq [\![w, \tau]\!]$, then we are done by taking this word as w'; it is not difficult to verify that the word $w' = u_{\tau,M}$ satisfies the requirements of the lemma.

Before handling the case $[\![u_{\tau,M}, \tau]\!] \not\leq [\![w, \tau]\!]$, let us first illustrate this situation by means of Figure 5: we assume that we have the states $q_0, \ldots, q_9$ and the tuples $[\![u_{\tau,M}, \tau]\!] = (3, 6, 4, 5, 6, 1, 7, 5, 2, 5)$ and $[\![w, \tau]\!] = (6, 8, 10, 11, 3, 4, 11, 2, 5, 8)$. The states for which $[\![w, \tau]\!]$ exceeds N are q_2, q_3, q_6, so these states constitute the set M. The case $[\![u_{\tau,M}, \tau]\!] \not\leq [\![w, \tau]\!]$ occurs since, for instance, $[\![u_{\tau,M}, \tau]\!](q_4) > [\![w, \tau]\!](q_4)$. Actually, this means that the applicability of τ by means of w, which requires 3 distinct trees evaluating to q_4, does not imply the applicability of τ by means of $u_{\tau,M}$, which requires 6 of such trees. Consequently, we cannot use $u_{\tau,M}$ as a replacement for w; technically speaking, the word $u_{\tau,M}$ violates the requirement (2) of Lemma 3. Thus, our goal is now to find another word for which this implication holds (within the computation of $\textsc{Bound}(\mathfrak{A}, \tau, M)$). For this implication, the tuple $[\![w, \tau]\!]$ provides us with the necessary restrictions on the number of distinct trees; we distinguish two kinds of states:

- For q_0, we have $[\![w, \tau]\!](q_0) \leq \textsc{Bound}(\mathfrak{A}, \tau, M)$. This means that, if we take a word v resulting from the computation of $\textsc{Bound}(\mathfrak{A}, \tau, M)$, it might be the case that $[\![v, \tau]\!](q_0) > [\![w, \tau]\!](q_0)$, in which case the applicability of τ by means of w does not imply the applicability of τ by means of v, similar to $u_{\tau,M}$ above. Hence, we should only consider those words v for which $[\![v, \tau]\!](q_0) \leq [\![w, \tau]\!](q_0)$. Similarly, this argumentation applies to the states q_4, q_5, q_7, and q_8.
- For q_1, we have $[\![w, \tau]\!](q_1) > \textsc{Bound}(\mathfrak{A}, \tau, M)$. The q_1-component of a word v resulting from the computation of $\textsc{Bound}(\mathfrak{A}, \tau, M)$, by the design of Algorithm 15, never exceeds $\textsc{Bound}(\mathfrak{A}, \tau, M)$, so, with respect to q_1, the applicability of τ by means of w always implies the applicability of τ by means of v. Hence, we do not need to impose any restriction on the q_1-component of v. Similarly, this argumentation applies to the states q_1, q_2, q_3, q_6, and q_9.

To sum up, we have to look for a word v in the computation of $\textsc{Bound}(\mathfrak{A}, \tau, M)$ under the restriction that the q_0-, q_4-, q_5-, q_7-, and q_8-component of $[\![v, \tau]\!]$ do not exceed 6, 3, 4, 2, and 5, respectively. Using the notation of Algorithm 15, this restriction is represented by the set $R = \{q_0, q_4, q_5, q_7, q_8\}$ and the tuple $\bar{d} = (6, 3, 4, 2, 5)$. Since w belongs to $S_{\tau,R,\bar{d},M}$, this set is not empty. Hence, there exists some $\bar{e} \leq \bar{d}$ that has proven successful in the computation of $\textsc{Bound}(\mathfrak{A}, \tau, M)$, thereby giving a word that satisfies all the requirements of Lemma 3.

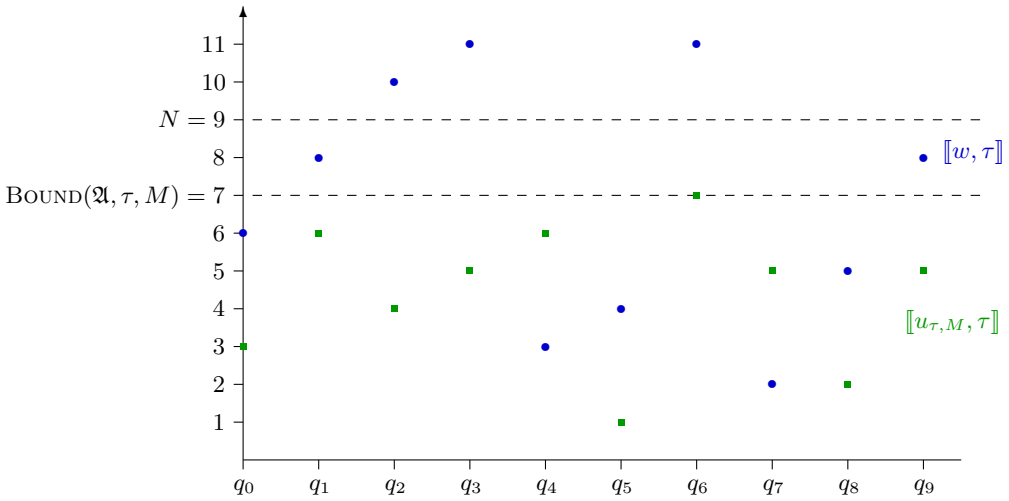

Fig. 5. The case $[\![u_{\tau,M}, \tau]\!] \not\leq [\![w, \tau]\!]$ in the proof of Lemma 3: the horizontal and the vertical axis represent the states and the number of distinct trees needed for the states, respectively. The connecting lines between points are only meant for readability.

Formally, let us assume that $[\![u_{\tau,M}, \tau]\!] \not\leq [\![w, \tau]\!]$, that is, there exists some state $s \in Q$ such that

$$[\![u_{\tau,M}, \tau]\!](s) > [\![w, \tau]\!](s) \ . \tag{11}$$

We now want to look for another word v for which the applicability of τ by means of w implies the applicability of τ by means v by considering subsets of $S_{\tau,M}$ with respect to some appropriate restrictions R and $\bar{d}$. As explained above, for each state $p \in Q$, we distinguish two cases:

- If $[\![w, \tau]\!](p) \leq \textsc{Bound}(\mathfrak{A}, \tau, M)$, then, among the words resulting from the computation of $\textsc{Bound}(\mathfrak{A}, \tau, M)$, we should only consider those words whose p-component does not exceed $[\![w, \tau]\!](p)$.
- If $[\![w, \tau]\!](p) > \textsc{Bound}(\mathfrak{A}, \tau, M)$, then the p-component of the words resulting from the computation of $\textsc{Bound}(\mathfrak{A}, \tau, M)$ will never exceed $[\![w, \tau]\!](p)$; thus, intuitively, we can assume that there are sufficiently many distinct trees evaluating to p.

Hence, we define the set R as

$$\{p \in Q \mid [\![w, \tau]\!](p) \leq \textsc{Bound}(\mathfrak{A}, \tau, M)\}$$

and the tuple $\bar{d} \in \mathbb{N}^{|R|}$ by setting

$$\bar{d}(p) = [\![w, \tau]\!](p)$$

for all $p \in R$. Note that, due to (11) and because $\textsc{Bound}(\mathfrak{A}, \tau, M) \geq [\![u_{\tau,M}, \tau]\!](s)$, the set R is not empty. Furthermore, by the definition of R and $\bar{d}$, the word w belongs to $S_{\tau,R,\bar{d},M}$, so this set is not empty either. Consequently, there exists some $\bar{e} \leq \bar{d}$ that has proven successful in the computation of $\textsc{Bound}(\mathfrak{A}, \tau, M)$. Without loss of generality, let us assume that $\bar{d}$ itself has proven successful. That is, at some point of the computation of $\textsc{Bound}(\mathfrak{A}, \tau, M)$ the part marked with $(*)$ is executed, thereby giving a word $v_{\tau,R,\bar{d},M}$ from Line 20. Now, it is not difficult to verify that the latter word indeed satisfies the requirements of the lemma, so we can take it as the desired word w'. □

D Details of the Nonemptiness Decision Procedure for Deterministic UTACS's

In this appendix, first, we show Lemma 6, from which the completeness of Algorithm 5 follows. Then, we present some more details of Algorithm 5, in particular, concerning a possible implementation of the main loop (Line 3–6 of Algorithm 5).

Proof of Lemma 6. The proof is by induction on the structure of t. If t only consists of a leaf, then the claim holds, as T_q is initialized by means of the leaf transitions.

Let us now consider $t = a(t_1 \dots t_m)$ with $t \to q$ and $t_i \to q_i$, for each $i = 1, \dots, m$, and let us refer to the word $q_1 \dots q_m$ as w. By the definition of runs, there exists some transition $\tau = (L, \alpha, a, q) \in \Delta$ such that $w \in L$, and w and $t_1 \dots t_m$ satisfy α. Our aim is now to show that $t \in T_q$ holds or that we can construct N distinct trees evaluating to q out of trees we have already constructed in the algorithm, which means $|T_q| = N$.

By the induction hypothesis, we have $t_i \in T_{q_i}$ or $|T_{q_i}| = N$. If $t_i \in T_{q_i}$, for all $i = 1, \dots, m$, then we will eventually construct t according to Line 3–6 of Algorithm 5 by means of τ, w, and $t_1, \dots, t_m$, so we have $t \in T_q$.

Otherwise, we aim to be able to construct N distinct trees evaluating to q out of the trees in $\bigcup_{s \in Q} T_s$ by using the transition τ and some appropriate τ-suitable word u. Let us first consider the set $R \subseteq Q$ defined as

$$R = \{r \in Q \mid \text{there exists some } 1 \leq i \leq m \text{ such that } t_i \to r \text{ and } t_i \notin T_r\} \ .$$

Roughly speaking, R contains the states r for which at least one of the t_i's evaluating to r does not belong to T_r; by the induction hypothesis, T_r thus contains N trees. For the word u, we want to take such a τ-suitable word that

(i) $[\![u, \tau]\!] \leq \widehat{N}$,
(ii) $[\![u, \tau]\!] \leq [\![w, \tau]\!]$, which means that for each state $s \in Q$, if it occurs in u, then it also does in w, which in turn implies that T_s is not empty,
(iii) R is not empty, and at least one state $p \in R$ occurs in u.

We distinguish two cases. First, if $[\![w, \tau]\!] \leq \widehat{N}$, then we take w as u. The conditions (i), (ii), and (iii) then follow immediately from our assumptions. Second, if $[\![w, \tau]\!] \not\leq \widehat{N}$, this means that there exists some state $p_0 \in Q$ such that $[\![w, \tau]\!](p_0) > N$. By Lemma 3, there exists some τ-suitable word w' fulfilling the requirements given therein, namely (1), (2), (3). We now take w' as u. The conditions (i) and (ii) follow from (1) and (2), respectively. Since $[\![w, \tau]\!](p_0) > N$, the state p_0 belongs to R, so R is not empty, and further, by (3), the state p_0 occurs in w'; thus, the condition (iii) is satisfied.

Let us now turn to our goal, namely to construct trees t' evaluating to 'q by using the transition τ and the τ-suitable word u chosen as above. For the occurrences of $r \in Q \setminus R$ in u, we use the trees among $t_1, \ldots, t_m$ that evaluate to r; note that this is possible due to the condition (ii), and also note that these trees, by the definition of R, belong to $\bigcup_{s \in Q} T_s$. For the occurrences of $r \in R$, we use the trees from T_r. This is, again, possible, due to the condition (i). As a result, we obtain a tree t' evaluating to q that is constructed out of the trees in $\bigcup_{s \in Q} T_s$, that is, out of the trees that we have already constructed in the algorithm. Hence, we will eventually put t' into T_q.

Now, let us consider the number of possibilities of constructing such a tree t'. By the conditions (iii), we can fix one particular state $p \in R$ occurring in u. Since $|T_p| = N$, there are at least $\binom{N}{[\![u,\tau]\!](p)} \cdot ([\![u, \tau]\!](p))!$ possibilities of constructing such a tree,[3] which is greater than or equal to N. Hence, we can construct at least N trees belonging to T_q, so we have $|T_q| = N$. □

A More Detailed Nonemptiness Decision Procedure. Let us first explain how the algorithm below works and, additionally, fix the notations we are going to use throughout the algorithm. Let $\mathfrak{A} = (Q, \Sigma, \Lambda, \Delta, F)$ be a deterministic UTACS. For the sake of simplicity, let us assume that $Q = (q_1, \ldots, q_r)$. For a state q_i, we use T_i to store the trees evaluating to q_i that we have encountered during the execution of the algorithm. Thus, we initialize each T_i by means of the leaf transitions Λ. The main loop of the algorithm, essentially, consists of two blocks:

– The first block iterates the procedure UPDATE, which works as follows. Let us assume that the trees we have constructed so far are stored in $T_1, \ldots, T_r$ and $T'_1, \ldots, T'_r$; the latter sets are used to store the trees most recently constructed (that is, in the last iteration of this block or of the whole main loop) and should be disjoint to the former ones. Then, the existence of $T'_1, \ldots, T'_r$ enables the possibility of constructing new trees out of the trees in $T_1, \ldots, T_r$. To illustrate this, consider a tree $t = a(t_1 \ldots t_m)$, say, in T_i, where $t_j \in T_{i_j}$.

[3] $\binom{N}{[\![u,\tau]\!](p)}$ represents the number of possibilities to choose $[\![u, \tau]\!](p)$ out of N trees from T_p, while $([\![u, \tau]\!](p))!$ gives the number of permutations of the so-chosen trees.

Consider some t_k among $t_1, \ldots, t_m$. Now, if there exists some tree $t' \in T'_{i_k}$, then we can replace every occurrence of t_k in t with t' and, under the assumption that T_{i_k} and T'_{i_k} are disjoint, obtain a new tree which also evaluates to q_i.

– If we cannot construct new trees in the first block any more, then, in the second block, we construct one by looking for a fresh suitable word for some transition. For this, we keep track of the τ-suitable words that we have used so far in W_τ, for each transition τ.

Finally, we will use the tuples $\bar{d}, \bar{e} \in \mathbb{N}^{|Q|}$ to determine whether a new iteration of the main loop is needed.

Algorithm 17. The input is a deterministic UTACS $\mathfrak{A}$ as described above, together with the bound N from Lemma 3. The output is 'yes,' if $T(\mathfrak{A})$ is not empty, or 'no,' otherwise.

function NONEMPTY($\mathfrak{A}$)
  initialize each T_i with $\{a \in \Sigma \mid (a, p) \in \Lambda\}$
  initialize each T'_i with $\emptyset$
  initialize each W_τ with $\emptyset$
  initialize $\bar{d}$ and $\bar{e}$ with $(|T_1|, \ldots, |T_r|)$
  repeat
    while $\bigcup_{i=1}^{r} T'_i \neq \emptyset$ **do**
    $\bar{d}, \bar{e} \leftarrow (|T_1 \cup T'_1|, \ldots, |T_r \cup T'_r|)$
    if there exists some $\tau = (L, \alpha, a, q_i) \in \Delta$ such that
      $S_{\tau,Q,\bar{d}} \setminus W_\tau \neq \emptyset$ and $|T_i| < N$
    then
    choose some $w = q_{i_1} \ldots q_{i_m}$ from $S_{\tau,Q,\bar{d}} \setminus W_\tau$
    $W_\tau \leftarrow W_\tau \cup \{w\}$
    $T_{\text{new}} \leftarrow \{a(t_1 \ldots t_m) \mid t_j \in T_{i_j}$, for each $j = 1, \ldots, m$, and
      w and $t_1 \ldots t_m$ satisfy $\alpha\}$
    set T'_i to be a subset of T_{new} such that $|T_i \cup T'_i| \leq N$
    $\bar{e} \leftarrow (|T_1 \cup T'_1|, \ldots, |T_r \cup T'_r|)$
  until $\bar{d} = \bar{e}$
  if there exists some $i \in \{1, \ldots, r\}$ such that $q_i \in F$ and $T_i \neq \emptyset$ **then**
    return 'yes'
  else
    return 'no'

procedure UPDATE($T_1, \ldots, T_r, T'_1, \ldots, T'_r, (W_\tau)_{\tau \in \Delta}$)
  initialize each $T_{i,\text{new}}$ with $\emptyset$
  for $i = 1, \ldots, r$ **do**
    for all $t = a(t_1 \ldots t_m) \in T_i \setminus \Sigma$ with $t_1 \in T_{i_1}, \ldots, t_m \in T_{i_m}$ **do**
      let $w = q_{i_1} \ldots q_{i_m}$
      for all $\tau = (L, \alpha, a, q_i) \in \Delta$ **do**
        if $w \in W_\tau$, and w and $t_1 \ldots t_m$ satisfy α **then**
          $T_{i,\text{new}} \leftarrow T_{i_{\text{new}}} \cup \{t' \mid t$ results from t by replacing at least

30: one t_k with $t'_k \in T'_{i_k}$ such that
31: w and $t'_1 \dots t'_m$ satisfy $\alpha\}$
32: **for** $i = 1, \dots, r$ **do**
33: $T_i \leftarrow T_i \cup T'_i$ // merge T_i and T'_i
34: set T'_i to be a subset of $T_{i,\text{new}}$ such that $|T_i \cup T'_i| \leq N$