RWTH Aachen

Department of Computer Science
Technical Report

Unranked Tree Automata with Sibling Equalities and Disequalities

Wong Karianto and Christof Löding

ISSN 0935–3232 · Aachener Informatik Berichte · AIB-2006-13

RWTH Aachen · Department of Computer Science · October 2006

Unranked Tree Automata with Sibling Equalities and Disequalities

Wong Karianto and Christof Löding

Lehrstuhl für Informatik 7, RWTH Aachen, Germany
{karianto,loeding}@informatik.rwth-aachen.de

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. It turns out that the main difficulty is indeed the absence of the rank, as it gives a certain bound on the number of distinct subtrees needed in order to satisfy an equality or disequality constraint. We overcome this difficulty by finding such a bound via a brute-force method.

Keywords: (unranked) tree automata, monadic second-order logic, equality and disequality constraints

1 Introduction

The notion of unranked trees, that is, 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 example, one can show that the regular languages of unranked trees, that is, those that are recognizable by (say, bottom-up) finite automata, coincide with those that are definable in monadic second-order logic. For references on this and other results, the reader is referred to, for example, the surveys [Nev02] and [Lib05] as well as the references therein.

A current trend in the theory of unranked tree automata is concerned with the development of automaton models that are more expressive than the finite automata and, at the same time, still permit good (algorithmic) properties. In particular, an approach has been to incorporate the notion of *constraints* in the definition of unranked tree automata. To illustrate this, let us consider a transition of a bottom-up finite automaton on unranked trees. Such a transition is usually described by a tuple (L, a, q) where L is a regular language over the state set of the automaton, a is a symbol from the input alphabet, and q is a state of the automaton. Intuitively, this transition can be applied at a node if (a) the node is labeled with a, (b) the node has k subtrees that evaluate to the states $q_1, \ldots, q_k$, and the word $q_1 \ldots q_k$ belongs to L. Now, here is the point where constraints come into play; one restricts the applicability this transition by requiring that among the states $q_1, \ldots, q_k$, for instance, a particular state occurs at least half as often as k. Note that such a constraint cannot be captured by the regular language L of the transition above.

Examples of unranked tree automata with numerical constraints are the Presburger automata of Seidl et. al. [SSM03,SSMH04] and, almost equivalently, the sheaves automata of Lugiez and Dal Zilio [DL03]. In these models, the applicability of transitions is subject to numerical constraints expressed in Presburger arithmetic (the first-order logic over the natural numbers). It turns out that these classes of automata share many good properties with the finite automata. In particular, the nonemptiness problem for these automata is decidable, which relies on the decidability of Presburger arithmetic. As an application of the latter result, the authors of these papers develop classes of logic-based query languages over unranked trees; some of which then turn out to enjoy decidable satisfiability.

Apart from numerical constraints, another type of constraint that has been considered in the classical setting of ranked trees deals with the aspect of non-linearity, which, for instance, appears in the (non-regular) language of trees of the form $a(tt)$. In order to capture this aspect, it has been suggested that the applicability of a tree automaton transition is subject to some equality tests between subtrees of the node under consideration; for references, see, for instance, [CDG+97]. However, 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 decidable nonemptiness problem [MS81]. Furthermore, undecidability remains even if the equality tests are imposed only to cousin subtrees, that is, subtrees of depth at most two [Tom92]. Hence, several subclasses of tree automata with equality constraints with decidable nonemptiness problem have been proposed in the literature, such as the *reduction automata* as well as its variants [DCC95,CCC+94,JRV06].

Another subclass of tree automata with equality constraints has been suggested by Bogaert and Tison in [BT92]; they restrict the equality tests to sibling subtrees (subtrees of depth one). For this class of (ranked) tree automata, they are able to show not only that it forms a Boolean algebra, but also that the nonemptiness problem for this class is decidable. Actually, the latter result is shown for the deterministic automata, but the corresponding results holds for the nondeterministic ones as well, as Bogaert and Tison show that the latter can be determinized.

In this paper, we aim at extending the latter results to the unranked setting. The first obstacle we need to deal with stems from the unboundedness aspect: as opposed to the ranked setting, 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. In order to define an automaton model properly the (possibly unboundedly many) equality tests must be finitely representable. 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. Throughout this paper, we will refer to the unranked tree automata with constraints between siblings that we propose as UTACS.

The main result of this paper is that the nonemptiness problem for deterministic UTACS is decidable. We do this by adapting Bogaert and Tison's nonemptiness decision procedure. However, as the termination of this procedure relies on the particular rank of the input alphabet, which is absent from our setting, we need to find an appropriate termination criterion for our decision procedure. More precisely, we develop a brute-force method to find a bound on the number of distinct subtrees that are needed in order to apply a transition.

In contrast to Bogaert and Tison's automata, however, there exists a nondeterministic UTACS that is not equivalent to any deterministic UTACS. Thus, this paper leaves open the question whether the nonemptiness problem for nondeterministic UTACS is decidable.

Related works. The works that serve as a motivation for this paper have been mentioned above.

Lugiez [Lug03] considers unranked, unordered trees as the so-called *multitrees* and proposes multitree automata with constraints among sibling multitrees in the transitions. He shows that these automata, with an appropriate definition of the constraints, are closed under all Boolean operations, are 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.

Structure of the paper. Following this introduction, we fix the notations we are going to use throughout our exposition. In Section 3 we present our automaton model and indicate some closure properties. Then, in Section 4 we show our main result, namely that the emptiness problem for deterministic UTACS is decidable. Section 5 indicates some possible variations of our automaton model. We conclude with some remarks on further prospects in Section 6. Finally, Appendix A and B give some details omitted from the main part of the paper.

2 Preliminaries

We denote the set of natural numbers by $\mathbb{N}$ and the set of positive integers by $\mathbb{N}_+$. For $k > 0$, we denote the set of k-tuples of natural numbers by $\mathbb{N}^k$. As usual, tuples of natural numbers are ordered by comparing them componentwise. For a natural number m, whenever no confusion might arise, $\bar{m}$ denotes the tuple $(m, \ldots, m)$ consisting of k-many m's.

Regular word languages and monadic second-order logic. We follow the presentation in [Tho97].

Let A be a finite, nonempty alphabet. We denote the empty word by ε. The set of all words and the set of all nonempty words over A are denoted by A^* and

A^+, respectively. The set of regular languages over A is denoted by $\mathrm{Reg}(A)$, and the set of regular languages over A that exclude the empty word is denoted by $\mathrm{Reg}_+(A)$. For a word w over A, let $|w|$ denote its length.

A nonempty word w over A defines the word structure $\langle\{1,\ldots,|w|\}, S, <, (\chi_a)_{a\in A}\rangle$, denoted as $\underline{w}$, where S denotes the usual successor relation, $<$ denotes the usual order relation, and χ_a is the set of those positions in w that are labeled with a, for each $a \in A$.

The formulas of monadic second-order (MSO, for short) logic over words over A, which we will simply refer to as MSO-formulas (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 (such as $\wedge$, $\vee$, $\neg$, $\rightarrow$, $\leftrightarrow$) and first-order as well as monadic second-order quantifiers.

For an MSO-formula φ, we will write $\varphi(x_1,\ldots,x_n,X_1,\ldots,X_m)$ to indicate that φ may contain free occurrences of the variables $x_1,\ldots,x_n,X_1,\ldots,X_m$. Then, the semantics of φ is given by a word structure $\underline{w}$ and an assignment of positions $\kappa_1,\ldots,\kappa_n$ and of sets $K_1,\ldots,K_m$ of positions in $\underline{w}$ to the free variables $x_1,\ldots,x_n,X_1,\ldots,X_m$, respectively. If, with this interpretation, φ holds, we write $\underline{w} \models \varphi(\kappa_1,\ldots,\kappa_n,K_1,\ldots,K_m)$.

It is well known that MSO-sentences (that is, formulas without free variables) over A exactly define the regular languages of nonempty words over A. For more details, the reader is referred to [Tho97].

Trees and tree languages. We denote the set of words over or sequences of (positive) natural numbers by $\mathbb{N}^*$ (and $\mathbb{N}_+^*$, respectively).

A *tree domain* D is a nonempty subset of $\mathbb{N}_+^*$ such that for each $u \in D$ we have

- $u' \in D$, for each prefix u' of u, and
- if $ui \in D$, for some $i \in \mathbb{N}_+$, then also $uj \in D$, for each $j \in \{1,\ldots,i\}$.

Let Σ be a nonempty, finite (unranked) alphabet. A tree t over Σ (or, for short, Σ-labeled tree) is given by a mapping $t\colon \mathrm{dom}_t \to \Sigma$ where $\mathrm{dom}_t \subseteq \mathbb{N}_+^*$ is a finite tree domain. The elements of dom_t are referred to as the nodes of t. The node ε is called the root of t. A node $u \in \mathrm{dom}_t$ is said to have $k \in \mathbb{N}$ successors if $ui \in \mathrm{dom}_t$, for all i with $1 \geq i \geq k$, and $ui \notin \mathrm{dom}_t$, for all $i > k$. 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 and j with $1 \geq i, j \geq k$. A leaf of t is a node without any successor.

Given a node u of t, the subtree of t at u, denoted as $t|_u$ 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 $v \in \mathrm{dom}_{t|_u}$. Moreover, $t|_u$ is called a direct subtree of t if $|u| = 1$.

In order to simplify our presentation, we sometimes refer to a tree t as $a(t_1\cdots t_k)$ in order to indicate that the root of the tree t is labeled with a and if it has k successors at which the subtrees $t_1,\ldots,t_k$ are rooted; that is, $t_i = t|_i$, for each i with $1 \leq i \leq k$.

A set of Σ-labeled trees is called a tree language over Σ. The tree language containing all Σ-labeled trees is denoted by $\mathcal{T}_\Sigma$.

An ordered sequence of Σ-labeled trees is called a hedge over Σ. The set of all Σ-labeled hedges is denoted by $\mathcal{H}_\Sigma$.

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

In the framework of ranked trees, automata with equality and disequality constraints between direct subtrees, as introduced in [BT92], extend the usual bottom-up tree automata in the following sense. On a given input tree, such an automaton works in a bottom-up fashion, i.e., from the leaves to the root, thereby assigning states to the nodes of the tree according to its transitions. Moreover, the application of a transition on a node of the tree is subject to some equality and disequality constraints between the *direct* subtrees of that particular node. In other words, such a transition is of the form $(q_1, \dots, q_k, \alpha, a, q)$ where $q_1, \dots, q_k, q$ are states of the automaton, a is a symbol of rank k, and α is a Boolean combination of atomic equality and disequality constraints. For example, the constraint $1 = 2 \wedge 2 = 3$ expresses the property that the first, second, and third direct subtrees of the node under consideration are equal to one another.

In this case, an atomic constraint simply addresses the direct subtrees to be compared directly, which is possible since the number of successors of the node under consideration is bounded by the rank k. For instance, we can express the constraint “all direct subtrees are equal to one another” by saying

$$\bigwedge_{1 \leq i,j \leq k} i = j \ .$$

When moving on from ranked tree automata to unranked tree automata, we encounter the fact that the number of successors of a node, while applying a transition, is no longer bounded by some rank. For instance, the length of the sequence of states that have been assigned to the successors of a node may be arbitrarily large. The usual approach to this phenomenon is to replace the sequence of states $q_1, \dots, q_k$ in a transition with a regular word language over the set of states. 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, for example, 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. For example, if we want to express that “all direct subtrees are equal to one another”, then we will have to address infinitely many pairs of direct subtrees to be compared.

In the following, we propose a mechanism to address the pairs of direct subtrees, while taking into account the unboundedness aspect. More precisely, we will use MSO-formulas with two free first-order variables as atomic constraints. Then, we can ensure that equality and disequality constraints built up from such atomic constraints are finitely representable.

Constraints between siblings. Let A be a finite, nonempty alphabet. An atomic sibling constraint over A is given by an MSO-formula $\varphi(x, y)$ over A that may contain free occurrences of the first-order variables x and y. Furthermore, we distinguish four types of usages of atomic constraints: $\exists_{\mathsf{EQ}}$-constraints, $\exists_{\mathsf{NEQ}}$-constraints, $\forall_{\mathsf{EQ}}$-constraints, and $\forall_{\mathsf{NEQ}}$-constraints. 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)". Likewise, 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)". The *sibling constraints over* A are built up from atomic sibling constraints by means of Boolean connectives. The set of all sibling constraints over A is denoted by CONS_A.

Formally, we define the semantics of CONS_A as follows. A nonempty word w over A and a hedge $h = t_1 \dots t_{|w|}$ over an alphabet Σ are said to satisfy an atomic sibling constraint φ if, depending on the type of φ, one of the following holds.

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

This semantics is extended to Boolean combinations of atomic sibling constraints as usual.

As a remark, we observe that the $\exists_{\mathsf{EQ}}$-constraints are dual to the $\forall_{\mathsf{NEQ}}$ constraints with respect to negation, and vice versa. Likewise, the $\exists_{\mathsf{NEQ}}$-constraints are dual to the $\forall_{\mathsf{EQ}}$ constraints, and vice versa. Hence, it suffices to consider only positive (that is, without negation) Boolean combinations of atomic sibling constraints.

Tree automata with constraints between siblings. An *automaton with equality and disequality constraints between siblings on Σ-labeled trees* (*UTACS*) 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
- $\Delta \subseteq \mathrm{Reg}_+(Q) \times \mathsf{CONS}_Q \times \Sigma \times Q$ 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 $\rho\colon \mathrm{dom}_t \to Q$ such that for each node $u \in \mathrm{dom}_t$ the following holds.

- If u is a leaf node, then $(t(u), \rho(u)) \in \Lambda$.
- If u has k successors with the direct subtrees $t_1, \dots, t_k$, respectively, then there exists a transition $(L, \alpha, t(u), \rho(u)) \in \Delta$ such that
  - the word $\rho(u1) \dots \rho(uk) \in L$, and
  - the word $\rho(u1) \dots \rho(uk)$ and the hedge $t_1 \dots t_k$ satisfy α.

In case such a run exists, we write $t \to_{\mathfrak{A}} \rho(\varepsilon)$ or just $t \to \rho(\varepsilon)$, whenever no confusion might arise, and say that the tree 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 $\Delta = \{Q^+, \varphi, a, q\}$ where $\varphi(x, y) = \mathsf{true}$ is a $\forall_{\mathsf{EQ}}$-constraint.

By adapting the standard constructions from the ranked setting (see, for example, [CDG+97,BT92]), 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 are more powerful than the deterministic ones, as opposed to the ranked case, where determinization is possible.

Proposition 2. *There exist a tree language that is recognizable by a nondeterministic UTACS, but not by any deterministic UTACS's.*

The proof of Proposition 2 is quite technical and therefore omitted here; the interested reader is referred to Appendix A. Nevertheless, we give here the tree language that we use to separate the two classes: it is the set of trees of the form depicted in Figure 1. Intuitively, such a tree 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. 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 disequal.

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 particular b-strands and thus also of addressing these two particular positions in the constraints.

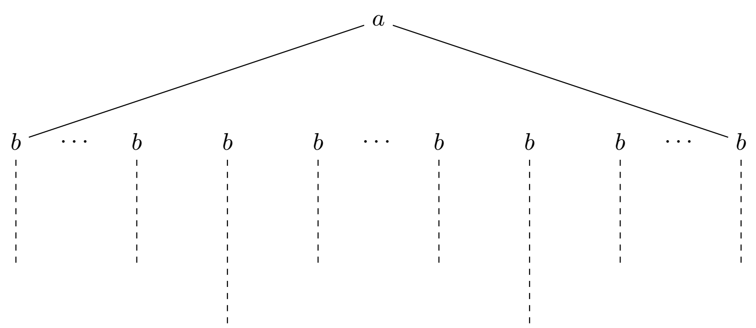

Fig. 1. Trees separating nondeterministic and deterministic UTACS's (the dashed lines represent b-strands)

Question 3. Is the class of nondeterministic UTACS's closed under complementation?

4 Nonemptiness Problem: the Deterministic Case

In the ranked setting, it has been shown in [BT92] that the nonemptiness problem for deterministic automata with sibling constraints is decidable, which carries over into nondeterministic automata since the latter can be determinized. Also note that the restriction to comparing only sibling subtrees is essential in order to have the decidability of the nonemptiness problems: the nonemptiness problem for automata with constraints between arbitrary subtrees or between cousin subtrees has been shown to be undecidable (see [CDG$^+$97] and the references therein).

The method used in [BT92] is an adaptation of the standard marking algorithm: one needs to find, for each state q of the given automaton, a tree that evaluates to this state. In order to satisfy the disequality constraints, however, one may need to find more than one of such trees. In the ranked case the termination of the nonemptiness procedure then relies on the maximal arity of the given tree automaton as this gives the maximal number of distinct trees needed for each state.

Herein lies our main obstacle to a nonemptiness decision procedure in the unranked setting: as the arity of a tree node is unbounded, it is not quite clear how to bound the number of distinct trees needed in order to satisfy a disequality constraint. In other words, if we want to transfer the marking algorithm to the unranked setting, we are facing two questions, namely what kind of bound we need and how we can find this bound.

The following subsections deal with these questions. In the first subsection, we state our main lemma: we assert that for each transition there is a bound on the number of distinct trees needed in order to apply this transition. We present a procedure to find these bounds and proof this lemma in the third subsection; before this, in the second subsection we will present some preparatory definitions and statements. Finally, we present the nonemptiness decision procedure for deterministic UTACS's in the last subsection.

4.1 The Bound Lemma

Let $\mathfrak{A} = (Q, \Sigma, \Delta, \Lambda, F)$ be a deterministic UTACS, and let $\tau = (L, \alpha, a, q)$ be a transition of $\mathfrak{A}$. A word $w \in Q^+$ is said to be *suitable for* τ if τ can be applied, thus resulting in a tree that evaluates to q, provided that, for each state occurring in w, sufficiently many trees evaluating to this state are given, In this case, 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.[1]

Remark 4. Note that, in general, $[\![w, \tau]\!]$ is not unique and that $[\![w, \tau]\!](p)$, for each $p \in Q$, does not need to exceed $|w|$.

Consequently, 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 hedge 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, the

[1] 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 $p \in Q$.

set of words in L that are suitable for τ, is nonempty as τ otherwise cannot be applied at all and can thus be removed from Δ.

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' that does not exceed N. This is stated in the following lemma, to which we will refer to as the bound lemma.

Lemma 5. *Let $\mathfrak{A}$ be a deterministic UTACS. 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*

(5a) $[\![w',\tau]\!] \leq \bar{N}$,
(5b) $[\![w',\tau]\!] \leq [\![w,\tau]\!]$, *and*
(5c) for any $p \in Q$, if $[\![w,\tau]\!](p) > N$, then $[\![w',\tau]\!](p) > 0$.

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 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 reason; it asserts that if a component $p \in Q$ in $[\![w,\tau]\!]$ exceeds N, then it must occur in w'. The proof of the bound lemma will be given below.

4.2 Suitable Words: Regularity and Restrictions

Let $\mathfrak{A} = (Q, \Sigma, \Lambda, \Delta, F)$ be a deterministic UTACS. 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.

Let us first consider some preprocessing of (the constraints of) the transitions of a (deterministic) UTACS, which leads to the normal form from Lemma 6 below.

Conjunctions 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 q, so determinism is retained.

Merging for-all constraints. 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 hedge $t_1 \cdots t_{|w|} \in \mathcal{H}_\Sigma$, we have

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

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.

Lemma 6. *Each UTACS is equivalent to one where each constraint occurring in $\mathfrak{A}$ 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 $\psi_1, \ldots, \psi_\ell$.*

Next, let us fix a transition $\tau = (L, \alpha, a, q)$ of $\mathfrak{A}$. We show that for each transition in $\mathfrak{A}$, the set of suitable words for this transition is a regular subset of Q^+.

Lemma 7. *The set $S_\tau \subseteq Q^+$ is regular.*

Proof. By Lemma 6, 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 if and only if, first, it belongs to L and, second, the constraints in α do not cause conflicts in w; for instance, any pair (κ, λ) of positions in w satisfying $\theta_{\forall_{\mathsf{EQ}}}$ may not satisfy $\theta_{\forall_{\mathsf{NEQ}}}$. Moreover, 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 if and only if all of the following requirements are met:

1. The word w belongs to L.
2. There exist some pairs of positions in $\underline{w}$, say, (x_1, y_1), …, (x_k, y_k), (x'_1, y'_1), …, (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

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

3. For each pair (x, y) of positions in $\underline{w}$, the formulas
$$\theta_{\forall_{\mathsf{EQ}}}(x,y) \to \neg(\theta_{\forall_{\mathsf{NEQ}}}(x,y) \lor \theta_{\forall_{\mathsf{NEQ}}}(y,x)) \land \bigwedge_{p\in Q}(\chi_p(x) \leftrightarrow \chi_p(y))$$
and
$$\theta_{\forall_{\mathsf{NEQ}}}(x,y) \to \neg(\theta_{\forall_{\mathsf{EQ}}}(x,y) \lor \theta_{\forall_{\mathsf{EQ}}}(y,x)) \land \neg(x = y)$$
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. □

Let R be a subset of Q, and let $\bar{d} \in \mathbb{N}^{|R|}$ be a tuple of natural numbers. In order to simplify our presentation, let us fix $d_p = \bar{d}(p)$, for each $p \in R$.

We recall that a word w is suitable for τ if, given sufficiently many trees for each state occurring in w, the transition τ can be applied. Now, 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:

- there are d_p many distinct trees that evaluate to p, if $p \in R$, and
- there are 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 $S_{\tau,R,\bar{d}}$.

Remark 8. If $\bar{e} \in \mathbb{N}^{|R|}$ is a tuple of natural numbers with $\bar{d} \leq \bar{e}$, then we have $S_{\tau,R,\bar{d}} \subseteq S_{\tau,R,\bar{e}}$.

Moreover, by adapting the proof of Lemma 7, we can show that the latter set $S_{\tau,R,\bar{d}}$ is regular, too.

Lemma 9. *The set $S_{\tau,R,\bar{d}} \subseteq Q^+$ is regular.*

Proof. Again, by Lemma 6, 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 $\bar{d}$ if the occurrences of $p \in R$ can be partitioned into 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 $\bar{d}$ if and only if all of the following requirements are met:

1. The word w belongs to L.
2. There exists a family of sets of positions is $\underline{w}$, 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:
$$\forall x \quad \bigwedge_{p\in R}\left[\chi_p(x) \to \bigvee_{j_p=1}^{d_p}\left(C^p_{j_p}(x) \land \bigwedge_{k_p\in\{1,\ldots,d_p\}\setminus\{j_p\}} \neg C^p_{k_p}(x)\right)\right]$$
$$\land \bigwedge_{p\in R}\bigwedge_{j_p=1}^{d_p}\left[C^p_{j_p}(x) \to \chi_p(x)\right]$$

(b) there exist some pairs of positions in $\underline{w}$, say, $(x_1, y_1), \ldots, (x_k, y_k), (x'_1, y'_1), \ldots, (x'_\ell, y'_\ell)$, 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 $j = 1, \ldots, \ell$,

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

     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 $\underline{w}$, the formulas

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

   and

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

   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. □

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 8 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 they are empty or not, which will be exploited in the following subsections.

4.3 Finding the Bound

Before we present the procedure for finding the bounds, let us illustrate the method we are going to pursue by means of a simple example.

Example 10. Let $\mathfrak{A} = (Q, \Sigma, \Lambda, \Delta, F)$ be a deterministic UTACS with $Q = \{q_1, q_2, q_3, q_4\}$, and let $\tau = (L, \alpha, a, q_1)$ be a transition of $\mathfrak{A}$.

Suppose that S_τ is not empty, so let $v \in S_\tau$, say, with $[\![v, \tau]\!] = (2, 6, 1, 5)$. Then, $[\![v, \tau]\!]$ already gives a first approximation of the bound N, namely 6. In other words, if, for each state, there are six distinct trees that evaluate to this state, then we can apply τ. Now, what happens if there are actually only one tree for q_1 and three trees for q_2? Then, either (a) τ is indeed no longer applicable, or (b) τ is still applicable, but we now need, say, 17 and 11 trees for q_3 and q_4, respectively. In the latter case, thus, the bound must be updated to 17.

Example 10 illustrates our method in finding the bound:

1. start with an initial bound N;
2. check, for all subsets R of Q and all tuples $\bar{d} \in \mathbb{N}^{|R|}$ with $\bar{d} \leq \bar{N}$, whether the set of τ-suitable words with respect to R and $\bar{d}$ is empty or not;
3. in the latter case, update N accordingly and go back to 2.

Note that the bound N might get larger and larger during this process, so it needs a careful implementation in order to guarantee termination.

Let $\mathfrak{A} = (Q, \Sigma, \Delta, \Lambda, F)$ be a deterministic UTACS. In the following, 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 now 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 11. The input consists of 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 output is a natural number, which is supposed to be the bound 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: **end if**
5: get a word $u_{\tau,M}$ from $S_{\tau,M}$
6: $n \leftarrow |u_{\tau,M}|$ // see Remark 4
7: $I \leftarrow (\mathbb{N}_{\leq n})^m$ // I contains the m-tuples to be checked
8: **for all** $R \subseteq Q$ **do**
9: $I_R^+ \leftarrow \emptyset$ // I_R^+ and I_R^- contain the tuples $\bar{e}$ from $I\restriction_R$ that
10: $I_R^- \leftarrow \emptyset$ // have proven successful (i.e., $S_{\tau,R,\bar{e},M} \neq \emptyset$)
11: // and unsuccessful (i.e., $S_{\tau,R,\bar{e},M} \neq \emptyset$), respectively
12: **end for**
13: **while** $I \neq \emptyset$ **do**
14: get a tuple $\bar{z}$ from I and remove it from I
15: **for all** $R \subseteq Q$ **do**
16: $\bar{d} \leftarrow \bar{z}\restriction_R$
17: **if** there is no $\bar{e} \leq \bar{d}$ with $\bar{e} \in I_R^+$ **then**
18: // neither $\bar{d}$ nor any $\bar{e} \leq \bar{d}$ has proven successful
19: **if** there is no $\bar{e} \geq \bar{d}$ with $\bar{e} \in I_R^-$ **then**
20: // neither $\bar{d}$ nor any $\bar{e} \geq \bar{d}$ has proven unsuccessful
21: **if** $S_{\tau,R,\bar{d},M} = \emptyset$ **then**
22: $I_R^- \leftarrow I_R^- \cup \{\bar{d}\}$ // $\bar{d}$ has proven unsuccessful
23: **else**
24: $I_R^+ \leftarrow I_R^+ \cup \{\bar{d}\}$ // $\bar{d}$ has proven successful
25: get a word $v_{\tau,R,\bar{d},M}$ from $S_{\tau,R,\bar{d},M}$
$(*)$ 26: $n' \leftarrow \max\{n, |v_{\tau,R,\bar{d},M}|\}$ // update n and
27: $I \leftarrow I \cup ((\mathbb{N}_{\leq n'})^m \setminus (\mathbb{N}_{\leq n})^m)$ // put the new tuples
28: $n \leftarrow n'$ // into I
29: **end if**
30: **end if**
31: **end if**
32: **end for**
33: **end while**
34: **return** n
35: **end function**

Note, first, that the choice of $u_{\tau,M}$ and $v_{\tau,R,\bar{d},M}$ in Line 5 and 25, respectively, is not unique and, second, that the time complexity of the algorithm depends on this choice.

Lemma 12. *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 13–33 of the algorithm is executed infinitely often.

Line 14 ensures that after we have fetched a tuple from I, we also remove it from I, and Line 27 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 27, so this line and, more generally, Line 24–28 (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 15 for which $(*)$ is executed infinitely often. Let us now restrict our attention to these particular iterations of $(*)$. Let $\bar{z}_1, \bar{z}_2, \bar{z}_3, \dots \in \mathbb{N}^m$ 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 14 when $(*)$ is executed for the i-th time with the choice of R in Line 15. 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 [Dic13] (see also [BCMS01, 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 17 is violated. Thus, $(*)$ is not executed, a contradiction. □

Proof of Lemma 5. We now prove the bound lemma. Let $\mathfrak{A} = (Q, \Sigma, \Delta, \Lambda, F)$ be a deterministic UTACS. To begin with, let us define the bound N as

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

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 $[\![w, \tau]\!] \leq \bar{N}$, then we are done: we just need to take w as w' and verify that all three requirements of the lemma are 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 11 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 5. 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 2: we assume that we have the states $q_0, \dots, q_9$ 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 (5b) of Lemma 5. 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 11, 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 11, 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 5.

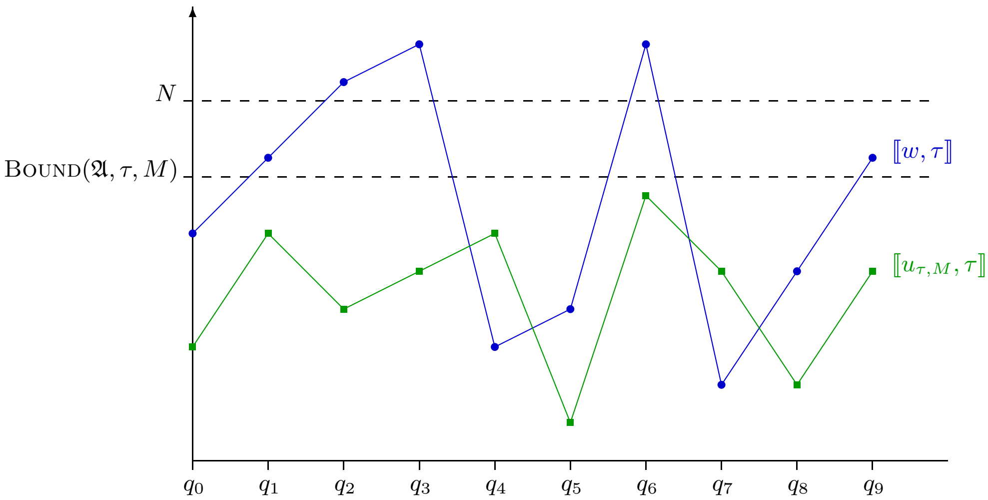

Fig. 2. The case $[\![u_{\tau,M}, \tau]\!] \not\leq [\![w, \tau]\!]$ in the proof of Lemma 5: the horizontal and the vertical axis represent the states and the number of distinct trees needed for the states, respectively. The connecting lines between points are only meant for readability.

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

$$[\![u_{\tau,M}, \tau]\!](s) > [\![w, \tau]\!](s) \ . \tag{1}$$

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 (1) 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 25. 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'. □

4.4 The Nonemptiness Decision Procedure

We are now ready to describe an algorithm that, given a deterministic UTACS $\mathfrak{A}$, decides whether the language accepted by $\mathfrak{A}$ is empty or not. In essence, this algorithm is an adaptation of the standard marking algorithm (see, for example, [CDG+97]): it consists of a main loop that in each round, for each state, collects a tree resulting from applying a transition based on the trees collected from previous rounds. The bound resulting from Lemma 5 gives the maximal number of distinct trees for each state we ought to collect; 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. Thus, the algorithm eventually terminates.

Algorithm 13. The input is a deterministic UTACS $\mathfrak{A} = (Q, \Sigma, \Lambda, \Delta, F)$, together with the bound N from Lemma 5. The output is 'yes', if $T(\mathfrak{A})$ is not empty, or 'no', otherwise. For each state $q \in Q$, we use T_q to store the trees evaluating to q that we have collected so far. In addition, we will use $\bar{d} \in \mathbb{N}^{|Q|}$ to keep track of the number of trees we have collected so far for each state; that is, at any time of the computation, $\bar{d}(q)$ contains the current value of $|T_q|$, for each $q \in Q$.

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 some transition $\tau = (L, \alpha, a, q) \in \Delta$, some τ-suitable
5: word $w = q_1 \dots q_m \in S_{\tau,Q,\bar{d}}$, and some trees $t_1 \in T_{q_1}, \dots, t_m \in T_{q_m}$
6: such that w and $t_1 \dots t_m$ satisfy α, and $|T_q| < N$
7: **then**
8: $T_q \leftarrow T_q \cup \{a(t_1 \dots t_m)\}$
9: **end if**
10: **until** no new tree can be constructed, or we have $\bar{d} = \bar{N}$
11: **if** there exists some $q \in F$ such that $T_q \neq \emptyset$ **then**
12: **return** 'yes'
13: **else**
14: **return** 'no'
15: **end if**
16: **end function**

Note that we do not explicitly give a detailed implementation of the algorithm, especially concerning the implementation of the main loop above (Line 3–10). A more detailed implementation of the algorithm can be found in Appendix B.

As explained above, the termination of Algorithm 13 is guaranteed by the existence of the bound from Lemma 5.

Algorithm 13 is sound: there (in particular, in Line 3–10) trees are constructed according to the transition relation Δ of $\mathfrak{A}$. Thus, if the output of the algorithm is 'yes,' then we have indeed constructed a tree that is accepted by $\mathfrak{A}$.

The completeness of Algorithm 13 (that is, if $T(\mathfrak{A})$ is nonempty, then the output of the algorithm must be 'yes') follows from the following lemma, which asserts that, if a tree t evaluates to a state q, then either we will eventually construct it, or we already have N trees evaluating to q.

Lemma 14. *For any $t \in \mathcal{T}_\Sigma$ and any $q \in Q$, if $t \to_{\mathfrak{A}} q$, then we have $t \in T_q$ or $|T_q| = N$.*

Proof. 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 let ρ be the run of $\mathfrak{A}$ on t. In particular, we have $\rho(\varepsilon) = q$. To simplify our presentation, we will refer to $\rho(i)$ as q_i (that is, $t_i \to q_i$), for each $i = 1, \dots, m$, and 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 belongs to L, and w and $t_1 \dots t_m$ satisfy α. A more precise statement about w concerns its suitability with respect to τ: we have $w \in S_{\tau,Q,\bar{d}}$, where $\bar{d}(s)$ contains the number of occurrences of s among $q_1, \dots, q_m$, for each $s \in Q$.

If T_q contains N trees, then the assertion of the lemma trivially holds. Hence, in the sequel, let us assume

$$|T_q| < N \ . \tag{2}$$

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 3–10 of Algorithm 13 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 consider the set $R \subseteq Q$ defined as

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

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) $[\![u, \tau]\!] \le \bar{N}$,
(ii) $[\![u, \tau]\!] \le [\![w, \tau]\!]$, which means that any state $s \in Q$ occurring in u also occurs in w, so the set T_s is not empty,
(iii) R is not empty, and
(iv) at least one state of R, say p, occurs in u.

We distinguish two cases. First, if $[\![w, \tau]\!] \le \bar{N}$, then we take w as u. The conditions (i), (ii), (iii), and (iv) then follow immediately from our assumptions. Second, if $[\![w, \tau]\!] \not\le \bar{N}$, this means that there exists some state $p_0 \in Q$ such that $[\![w, \tau]\!](p_0) > N$. By Lemma 5, there exists some τ-suitable word w' fulfilling the requirements given therein, namely:

(5a) $[\![w', \tau]\!] \le \bar{N}$,
(5b) $[\![w', \tau]\!] \le [\![w, \tau]\!]$, and
(5c) for any $p_0 \in Q$ with $[\![w, \tau]\!](p_0) > N$, we have $[\![w', \tau]\!](p_0) > 0$.

We now take w' as u. The conditions (i) and (ii) follow from (5a) and (5b), respectively. Since $[\![w, \tau]\!](p_0) > N$, the state p_0 belongs to R, so the condition (iii) is satisfied. Further, by (5c), the state p_0 occurs in w', so the condition (iv) 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) and (iv), 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,[2] which is greater than or equal to N. Hence, we can construct at least N trees belonging to T_q, which contradicts (2). $\square$

Let us summarize the main result of this section:

Theorem 15. *The nonemptiness problem for deterministic UTACS is decidable.*

[2] $\binom{N}{[\![u,\tau]\!](p)}$ 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.

We conclude this section with two remarks on the method we have presented.

Remark 16. Throughout the algorithms above, actually, we just use the fact that, for each transition, the set of words suitable for this transition is regular, which in turn implies that the nonemptiness problem for this set is decidable. In fact, we do not analyze the form of the equality and disequality constraints appearing in the transitions at all. Hence, our method would 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.

Remark 17. The method we have presented does not work for nondeterministic UTACS. With determinism, we can assume that if two trees evaluate to two different states, then these trees must be different as well. First, 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. Second, this assumption is needed in the construction of trees in the proof of Lemma 14.

With nondeterminism, in contrast, two trees evaluating to distinct states might still be equal. Thus, a more involved definition of suitability is required; for instance, one might compare sets of states instead of single states. However, it is still not clear whether, with this new definition, our method carries over into nondeterministic automata.

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 pair of siblings that are supposed to be equal or disequal. The use of this ability has been demonstrated in Proposition 2 when we proved that the nondeterministic UTACS are more expressive than the deterministic ones.

With this phenomenon in mind, we now prohibit the reference to states in the atomic constraints: the MSO-formulas used as atomic constraints lack atomic MSO-formulas of the form $\chi_a(x)$. We denote the set of all sibling constraints built from such atomic constraints by CONS and define its semantics as before. Accordingly, the resulting UTACS's are of the form $(Q, \Sigma, \Lambda, \Delta, F)$ where Q, Σ, Λ, and F are as usual, and Δ is a subset of $\mathrm{Reg}_+(Q) \times \mathsf{CONS} \times \Sigma \times Q$. Runs, acceptance, and determinism are defined as usual.

With this definition, we can show that every nondeterministic restricted UTACS can be transformed into a deterministic one by using the standard subset construction (see, for instance, [CDG$^+$97, Chapter 1 and 4]).

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 under these positions are equal or not. A more involved processing is to feed the subtrees into a (deterministic) tree transducer and consider the output trees instead of the subtrees themselves. For instance, if we want to define an automaton accepting the set of well-balanced trees over a two-letter alphabet (as opposed to Example 1), then we would like to forget the actual labeling of the nodes and just take the structure of the trees into consideration. 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. This kind of extension, currently, is still at an early stage of discussion; in the sequel, we just point out the essential definitions and the difficulties we encounter without going into detail.

Let $\mathfrak{M}$ be a deterministic bottom-up tree transducer with the state set P. For simplicity, let Σ be both the input and the output alphabet of $\mathfrak{M}$, and let us refer to the transduction defined by $\mathfrak{M}$ as $\mathfrak{M}\colon \mathcal{T}_\Sigma \to \mathcal{T}_\Sigma$. An atomic sibling constraint consists of an MSO-formula $\varphi(x, y)$ over P. As usual, we use atomic constraints to form the set of all sibling constraints, which we will refer to as $\mathsf{CONS}_{\mathfrak{M}}$. Accordingly, an inner-node transition of a UTACS $\mathfrak{A} = (Q, \Sigma, \Lambda, \Delta, F)$ is then given by a tuple (L, α, a, q). Given a node u at which the tree $t = a(t_1 \dots t_k)$ with $t_1 \to q_1, \dots, t_k \to q_k$ is rooted, this transition can be applied if the following holds:

- The word $q_1 \dots q_k$ belongs to L.
- For each $i = 1, \dots, k$, let p_i be the state assumed by $\mathfrak{M}$ after producing the output $\mathfrak{M}(t_i)$. Then, the word $p_1 \dots p_k$ together with the hedge $\mathfrak{M}(t_1) \dots \mathfrak{M}(t_k)$ should satisfy the constraint α.

The notion of determinism is defined as usual. Note also that the restriction to deterministic transducers is needed in order to fix the trees to be compared.

It turns out that our method of solving the nonemptiness problem for deterministic UTACS cannot be applied in this setting. For our method, similar to our observation in Remark 17, we require that if the states assumed by the transducer after producing two output trees, say, t and t', are different, then t and t' must also be different. However, this assumption does not hold for deterministic transducers in general, so a more restricted model of tree transducers is required.

6 Conclusions

We have extended the tree automaton model defined by Bogaert and Tison in [BT92] 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, we have defined an appropriate bound for this number.

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 an antichain (the set I_R^+ in the bound algorithm). 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.

Other prospective future work includes: (a) developing a nonemptiness decision procedure that directly works on nondeterministic UTACS, (b) considering extensions that involve tree transducers, and (c) considering automaton models that allow determinization.

References

[BCMS01] Olaf Burkart, Didier Caucal, Faron Moller, and Bernhard Steffen. Verification on infinite structures. In *Handbook of Process Algebra*, chapter 9, pages 545–623. Elsevier, 2001.

[BT92] Bruno Bogaert and Sophie Tison. Equality and disequality constraintss on direct subterms in tree automata. In *Proceedings of STACS 1992*, pages 161–171, 1992.

[CCC+94] Anne-Cécile Caron, Hubert Comon, Jean-Luc Coquidé, Max Dauchet, and Florent Jacquemard. Pumping, cleaning and symbolic constraints solving. In *Proceedings of ICALP 1994*, volume 820 of *LNCS*, pages 436–449. Springer, 1994.

[CDG+97] Hubert Comon, Max Dauchet, Rémi Gilleron, Florent Jacquemard, Denis Lugiez, Sophie Tison, and Marc Tommasi. *Tree Automata Techniques and Applications*. Unpublished electronic book, available on `http://www.grappa.univ-lille3.fr/tata`, 1997. Current version released on 1st October 2002.

[DCC95] Max Dauchet, Anne-Cécile Caron, and Jean-Luc Coquidé. Automata for reduction properties solving. *Journal of Symbolic Computation*, 20(2):215–233, 1995.

[Dic13] Leonard Eugene Dickson. Finiteness of the odd perfect and primitive abundant numbers with n distinct prime factors. *American Journal of Mathematics*, 35(4):413–422, 1913.

[DL03] Silvano Dal Zilio and Denis Lugiez. XML schema, tree logic and sheaves automata. In *Proceedings of RTA 2003*, volume 2706 of *LNCS*, pages 246–263. Springer, June 2003.

[JRV06] Florent Jacquemard, Michael Rusinowitch, and Laurent Vigneron. Tree automata with equality constraints modulo equational theories. In *Proceedings of IJCAR 2006*, volume 4130 of *LNAI*, pages 557–571. Springer, 2006.

[Lib05] Leonid Libkin. Logics for unranked trees: An overview. In *Proceedings of ICALP 2005*, volume 3580 of *LNCS*, pages 35–50. Springer, 2005.

[Lug03] Denis Lugiez. Counting and equality constraints for multitree automata. In *Proceedings of FOSSACS 2003*, volume 2620 of *LNCS*, pages 328–342. Springer, 2003.

[MS81] Jocelyne Mongy-Steen. *Transformation de noyaux reconnaissables d'arbres. Forêts RATEG*. PhD thesis, Université de Lille I, 1981.

[Nev02] Frank Neven. Automata, logic, and xml. In *Proceedings of CSL 2002*, volume 2471 of *LNCS*, pages 2–26. Springer, 2002.

[SSM03] Helmut Seidl, Thomas Schwentick, and Anca Muscholl. Numerical document queries. In *Proceedings of PODS 2003*, pages 155–166. ACM Press, 2003.

[SSMH04] Helmut Seidl, Thomas Schwentick, Anca Muscholl, and Peter Habermehl. Counting in trees for free. In *Proceedings of ICALP 2004*, volume 3142 of *LNCS*, pages 1136–1149. Springer, 2004.

[Tho97] Wolfgang Thomas. Languages, automata, and logic. In *Handbook of Formal Languages, Volume 3*, chapter 7, pages 389–455. Springer, 1997.

[Tom92] Marc Tommasi. Automates d'arbres avec tests d'égalité entre cousins germains. Technical report, Mémoire de DEA, Université de Lille I, 1992.

A Determinism versus Nondeterminism

We present a tree language that can be recognized by nondeterministic UTACS's but not by any deterministic ones.

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 $z \in \mathrm{dom}_t \setminus \{\varepsilon\}$.
- 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 3 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.

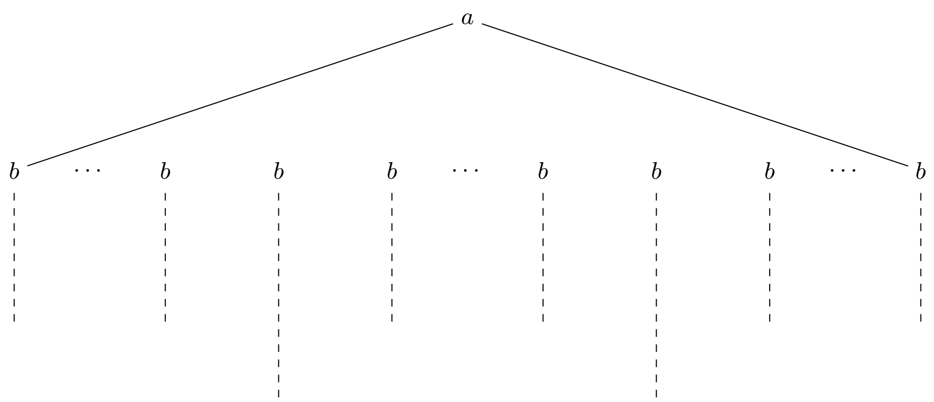

Fig. 3. A tree belonging to T (the dashed lines represent b-strands)

Proposition 18. *The tree language T can be recognized by a nondeterministic UTACS.*

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 $\varphi(x,y) := (1 < x \neq y < \max) \wedge P_p(x) \wedge P_p(y)$,
- $\forall_{\mathsf{EQ}}$-constraint $\psi(x,y) := P_q(x) \wedge P_q(y)$ is a $\forall_{\mathsf{EQ}}$, and
- $\forall_{\mathsf{NEQ}}$-constraint $\theta(x,y) := (P_p(x) \wedge P_q(y)) \vee (P_q(x) \wedge P_p(y))$.

Proposition 19. *The tree language T cannot be recognized by any deterministic UTACS.*

We will now sketch a proof of this proposition. 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. 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

$$a(\underbrace{s \cdots s}_{n_1\text{-times}}\ t\ \underbrace{s \cdots s}_{n_2\text{-times}}\ t\ \underbrace{s \cdots s}_{n_3\text{-times}}) \ . \tag{3}$$

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 (3) 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 (3) 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 (3). 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 $\underline{w(n_1, n_2, n_3)}$ with $\underline{w(n_1, n_2, n_3)} \models \theta(\kappa, \lambda)$, or
(b) θ is not fulfilled as there is some pair (κ, λ) of positions in $\underline{w(n_1, n_2, n_3)}$ with $\underline{w(n_1, n_2, n_3)} \models \theta(\kappa, \lambda)$ 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 $\underline{p^n}$ such that $\underline{p^n} \models \theta(\kappa, \lambda)\}$.

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 (4) shows:

$$\begin{array}{ccccccc} & & \kappa & & \lambda & & \\ p & \cdots & p & \cdots & p & \cdots & p \\ & & \bullet & & \circ & & \end{array} \tag{4}$$

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

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

or

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

or

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

and 'pumping' p^m always results in a word 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 (5)), or
- both lie in the p^{n_3}-part (this corresponds to (6)), or
- lie in the p^{n_1}-part and p^{n_3}-part, respectively (this corresponds to (7)).

In either case, we have $t(n_1, n_2, n_3)_{\kappa'} = t(n_1, n_2, n_3)_{\lambda'}$ and $\underline{w(n_1, n_2, n_3)} \models \theta(\kappa', \lambda')$. 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

$$\alpha = \theta \wedge \bigwedge_{i=1}^{k} \varphi_i \wedge \bigwedge_{j=1}^{\ell} \psi_\ell \tag{8}$$

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 $\underline{w(n_1, n_2, n_3)} \models \varphi_1(\kappa_1, \lambda_1)$ 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 $\underline{w(n_1^1, n_2^1, n_3^1)} \models \theta(\kappa_1', \lambda_1')$. 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 $\underline{w(n_1^k, n_2^k, n_3^k)} \models \psi_1(\mu_1, \nu_1)$ 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 (3), 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:

$$\begin{array}{ccccccccccc} & & & \mu_1 & & & & \nu_1 & & & \\ s & \cdots & s & t & s & \cdots & s & t & s & \cdots & s \end{array} \tag{9}$$

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'

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

or very near to μ_1'.

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

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 $\underline{w(m_1^1, m_2^1, m_3^1)} \models \psi_1(\mu_1', \nu_1')$. 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 4.

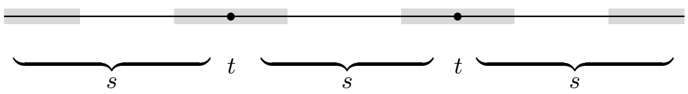

Fig. 4. An illustration of the word $w(m_1^k, m_2^k, m_3^k)$ (and of $w(m_1, m_2, m_3)$); the grey areas roughly mark the positions that are used to satisfy the $\exists_{\mathsf{EQ}}$-constraints and $\exists_{\mathsf{NEQ}}$-constraints; the bullets represent the positions where t occurs.

Handling the $\forall_{\mathsf{EQ}}$*-constraint.* Before we turn to the $\forall_{\mathsf{EQ}}$-constraint θ of (8), we observe that the area between the second and the third gray area in Figure 4 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 (8). Further, let κ and λ be the position of the subtrees t in $w(m_1, m_2, m_3)$.

If $\underline{w(m_1, m_2, m_3)} \not\models \theta(\kappa, \lambda)$, 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 $\underline{w(m_1, m_2, m_3)} \models \theta(\kappa, \lambda)$, 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 5. Again, as $P_{\tau,\theta}$ is regular, and if the gap between the second and the third gray area is sufficiently large, then we can find a position μ within this gap, such that $\underline{w(m_1, m_2, m_3)} \models \theta(\kappa, \mu)$ or $\underline{w(m_1, m_2, m_3)} \models \theta(\mu, \lambda)$ holds. In either case, the subtree at position μ must then be t, a contradiction.

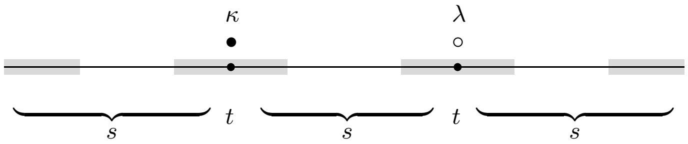

Fig. 5. The word $w(m_1, m_2, m_3)$ with the marked positions κ and λ

This concludes the proof of Proposition 19.

B A More Detailed Nonemptiness Decision Procedure

We present some details of Algorithm 13. In particular, we explain a possible implementation of the main loop (Line3–10).

Let us first explain how the algorithm 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 $t_j \in T_{i_j}$. 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 20. The input is a deterministic UTACS $\mathfrak{A}$ as described above, together with the bound N from Lemma 5. 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**
    end while
    $\bar{d}, \bar{e} \leftarrow (|T_1 \cup T_1'|, \ldots, |T_r \cup T_r'|)$
    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 $|T_i \cup T_i'| \leq N$
    end if
    $\bar{e} \leftarrow (|T_1 \cup T_1'|, \ldots, |T_r \cup T_r'|)$
  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'
  end if
end function

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
            one t_k with $t_k' \in T_{i_k}'$ such that
            w and $t_1' \ldots t_m'$ satisfy $\alpha\}$
        end if
      end for

        end for
    end for
    for $i = 1, \ldots, r$ **do**
        $T_i \leftarrow T_i \cup T_i'$ // merge T_i and T_i'
        set T_i' to be a subset of $T_{i,\mathrm{new}}$ such that $|T_i \cup T_i'| \leq N$
    end for
end procedure

Aachener Informatik-Berichte

This list contains all technical reports published during the past five years. A complete list of reports dating back to 1987 is available from http://aib.informatik.rwth-aachen.de/. To obtain copies consult the above URL or send your request to: Informatik-Bibliothek, RWTH Aachen, Ahornstr. 55, 52056 Aachen, Email: `biblio@informatik.rwth-aachen.de`

2001-01 *	Jahresbericht 2000
2001-02	Benedikt Bollig, Martin Leucker: Deciding LTL over Mazurkiewicz Traces
2001-03	Thierry Cachat: The power of one-letter rational languages
2001-04	Benedikt Bollig, Martin Leucker, Michael Weber: Local Parallel Model Checking for the Alternation Free mu-Calculus
2001-05	Benedikt Bollig, Martin Leucker, Thomas Noll: Regular MSC Languages
2001-06	Achim Blumensath: Prefix-Recognisable Graphs and Monadic Second-Order Logic
2001-07	Martin Grohe, Stefan Wöhrle: An Existential Locality Theorem
2001-08	Mareike Schoop, James Taylor (eds.): Proceedings of the Sixth International Workshop on the Language-Action Perspective on Communication Modelling
2001-09	Thomas Arts, Jürgen Giesl: A collection of examples for termination of term rewriting using dependency pairs
2001-10	Achim Blumensath: Axiomatising Tree-interpretable Structures
2001-11	Klaus Indermark, Thomas Noll (eds.): Kolloquium Programmiersprachen und Grundlagen der Programmierung
2002-01 *	Jahresbericht 2001
2002-02	Jürgen Giesl, Aart Middeldorp: Transformation Techniques for Context-Sensitive Rewrite Systems
2002-03	Benedikt Bollig, Martin Leucker, Thomas Noll: Generalised Regular MSC Languages
2002-04	Jürgen Giesl, Aart Middeldorp: Innermost Termination of Context-Sensitive Rewriting
2002-05	Horst Lichter, Thomas von der Maßen, Thomas Weiler: Modelling Requirements and Architectures for Software Product Lines
2002-06	Henry N. Adorna: 3-Party Message Complexity is Better than 2-Party Ones for Proving Lower Bounds on the Size of Minimal Nondeterministic Finite Automata
2002-07	Jörg Dahmen: Invariant Image Object Recognition using Gaussian Mixture Densities
2002-08	Markus Mohnen: An Open Framework for Data-Flow Analysis in Java
2002-09	Markus Mohnen: Interfaces with Default Implementations in Java
2002-10	Martin Leucker: Logics for Mazurkiewicz traces
2002-11	Jürgen Giesl, Hans Zantema: Liveness in Rewriting
2003-01 *	Jahresbericht 2002
2003-02	Jürgen Giesl, René Thiemann: Size-Change Termination for Term Rewriting
2003-03	Jürgen Giesl, Deepak Kapur: Deciding Inductive Validity of Equations

2003-04 Jürgen Giesl, René Thiemann, Peter Schneider-Kamp, Stephan Falke: Improving Dependency Pairs
2003-05 Christof Löding, Philipp Rohde: Solving the Sabotage Game is PSPACE-hard
2003-06 Franz Josef Och: Statistical Machine Translation: From Single-Word Models to Alignment Templates
2003-07 Horst Lichter, Thomas von der Maßen, Alexander Nyßen, Thomas Weiler: Vergleich von Ansätzen zur Feature Modellierung bei der Softwareproduktlinienentwicklung
2003-08 Jürgen Giesl, René Thiemann, Peter Schneider-Kamp, Stephan Falke: Mechanizing Dependency Pairs
2004-01 * Fachgruppe Informatik: Jahresbericht 2003
2004-02 Benedikt Bollig, Martin Leucker: Message-Passing Automata are expressively equivalent to EMSO logic
2004-03 Delia Kesner, Femke van Raamsdonk, Joe Wells (eds.): HOR 2004 – 2nd International Workshop on Higher-Order Rewriting
2004-04 Slim Abdennadher, Christophe Ringeissen (eds.): RULE 04 – Fifth International Workshop on Rule-Based Programming
2004-05 Herbert Kuchen (ed.): WFLP 04 – 13th International Workshop on Functional and (Constraint) Logic Programming
2004-06 Sergio Antoy, Yoshihito Toyama (eds.): WRS 04 – 4th International Workshop on Reduction Strategies in Rewriting and Programming
2004-07 Michael Codish, Aart Middeldorp (eds.): WST 04 – 7th International Workshop on Termination
2004-08 Klaus Indermark, Thomas Noll: Algebraic Correctness Proofs for Compiling Recursive Function Definitions with Strictness Information
2004-09 Joachim Kneis, Daniel Mölle, Stefan Richter, Peter Rossmanith: Parameterized Power Domination Complexity
2004-10 Zinaida Benenson, Felix C. Gärtner, Dogan Kesdogan: Secure Multi-Party Computation with Security Modules
2005-01 * Fachgruppe Informatik: Jahresbericht 2004
2005-02 Maximillian Dornseif, Felix C. Gärtner, Thorsten Holz, Martin Mink: An Offensive Approach to Teaching Information Security: "Aachen Summer School Applied IT Security"
2005-03 Jürgen Giesl, René Thiemann, Peter Schneider-Kamp: Proving and Disproving Termination of Higher-Order Functions
2005-04 Daniel Mölle, Stefan Richter, Peter Rossmanith: A Faster Algorithm for the Steiner Tree Problem
2005-05 Fabien Pouget, Thorsten Holz: A Pointillist Approach for Comparing Honeypots
2005-06 Simon Fischer, Berthold Vöcking: Adaptive Routing with Stale Information
2005-07 Felix C. Freiling, Thorsten Holz, Georg Wicherski: Botnet Tracking: Exploring a Root-Cause Methodology to Prevent Distributed Denial-of-Service Attacks
2005-08 Joachim Kneis, Peter Rossmanith: A New Satisfiability Algorithm With Applications To Max-Cut

2005-09 Klaus Kursawe, Felix C. Freiling: Byzantine Fault Tolerance on General Hybrid Adversary Structures

2005-10 Benedikt Bollig: Automata and Logics for Message Sequence Charts

2005-11 Simon Fischer, Berthold Vöcking: A Counterexample to the Fully Mixed Nash Equilibrium Conjecture

2005-12 Neeraj Mittal, Felix Freiling, S. Venkatesan, Lucia Draque Penso: Efficient Reductions for Wait-Free Termination Detection in Faulty Distributed Systems

2005-13 Carole Delporte-Gallet, Hugues Fauconnier, Felix C. Freiling: Revisiting Failure Detection and Consensus in Omission Failure Environments

2005-14 Felix C. Freiling, Sukumar Ghosh: Code Stabilization

2005-15 Uwe Naumann: The Complexity of Derivative Computation

2005-16 Uwe Naumann: Syntax-Directed Derivative Code (Part I: Tangent-Linear Code)

2005-17 Uwe Naumann: Syntax-directed Derivative Code (Part II: Intraprocedural Adjoint Code)

2005-18 Thomas von der Maßen, Klaus Müller, John MacGregor, Eva Geisberger, Jörg Dörr, Frank Houdek, Harbhajan Singh, Holger Wußmann, Hans-Veit Bacher, Barbara Paech: Einsatz von Features im Software-Entwicklungsprozess - Abschlußbericht des GI-Arbeitskreises "Features"

2005-19 Uwe Naumann, Andre Vehreschild: Tangent-Linear Code by Augmented LL-Parsers

2005-20 Felix C. Freiling, Martin Mink: Bericht über den Workshop zur Ausbildung im Bereich IT-Sicherheit Hochschulausbildung, berufliche Weiterbildung, Zertifizierung von Ausbildungsangeboten am 11. und 12. August 2005 in Köln organisiert von RWTH Aachen in Kooperation mit BITKOM, BSI, DLR und Gesellschaft fuer Informatik (GI) e.V.

2005-21 Thomas Noll, Stefan Rieger: Optimization of Straight-Line Code Revisited

2005-22 Felix Freiling, Maurice Herlihy, Lucia Draque Penso: Optimal Randomized Fair Exchange with Secret Shared Coins

2005-23 Heiner Ackermann, Alantha Newman, Heiko Röglin, Berthold Vöcking: Decision Making Based on Approximate and Smoothed Pareto Curves

2005-24 Alexander Becher, Zinaida Benenson, Maximillian Dornseif: Tampering with Motes: Real-World Physical Attacks on Wireless Sensor Networks

2006-01 * Fachgruppe Informatik: Jahresbericht 2005

2006-03 Michael Maier, Uwe Naumann: Intraprocedural Adjoint Code Generated by the Differentiation-Enabled NAGWare Fortran Compiler

2006-04 Ebadollah Varnik, Uwe Naumann, Andrew Lyons: Toward Low Static Memory Jacobian Accumulation

2006-05 Uwe Naumann, Jean Utke, Patrick Heimbach, Chris Hill, Derya Ozyurt, Carl Wunsch, Mike Fagan, Nathan Tallent, Michelle Strout: Adjoint Code by Source Transformation with OpenAD/F

2006-06 Joachim Kneis, Daniel Mölle, Stefan Richter, Peter Rossmanith: Divide-and-Color

2006-07 Thomas Colcombet, Christof Löding: Transforming structures by set interpretations

2006-08 Uwe Naumann, Yuxiao Hu: Optimal Vertex Elimination in Single-Expression-Use Graphs

2006-09 Tingting Han, Joost-Pieter Katoen: Counterexamples in Probabilistic Model Checking

2006-10 Mesut Günes, Alexander Zimmermann, Martin Wenig, Jan Ritzerfeld, Ulrich Meis: From Simulations to Testbeds - Architecture of the Hybrid MCG-Mesh Testbed

2006-11 Bastian Schlich, Michael Rohrbach, Michael Weber, Stefan Kowalewski: Model Checking Software for Microcontrollers

2006-12 Benedikt Bollig, Joost-Pieter Katoen, Carsten Kern, Martin Leucker: Replaying Play in and Play out: Synthesis of Design Models from Scenarios by Learning

* These reports are only available as a printed version.

Please contact `biblio@informatik.rwth-aachen.de` to obtain copies.