Minimizing Tree Automata for Unranked Trees
[Extended Abstract]

Wim Martens[1,⋆] and Joachim Niehren[2]

[1] Hasselt University and Transnational University of Limburg,
Agoralaan, gebouw D, B-3590 Diepenbeek, Belgium
wim.martens@uhasselt.be
[2] INRIA Futurs, LIFL, Mostrare project, Lille, France
http://www.grappa.univ-lille3.fr/mostrare

Abstract. Automata for unranked trees form a foundation for XML schemas, querying and pattern languages. We study the problem of efficiently minimizing such automata. We start with the unranked tree automata (UTAs) that are standard in database theory, assuming bottom-up determinism and that horizontal recursion is represented by deterministic finite automata. We show that minimal UTAs in that class are not unique and that minimization is NP-hard. We then study more recent automata classes that do allow for polynomial time minimization. Among those, we show that bottom-up deterministic stepwise tree automata yield the most succinct representations.

1 Introduction

Finite automata for unranked trees constitute the theoretical basis for XML schema languages [16] and are used in numerous areas of XML-related research, such as path and pattern languages [17, 22] and XML querying [7, 18]. Research on automata minimization therefore contributes to each of those fields.

In the context of XML schema languages, minimized schemas would improve the running time on document validation, or on static tests involving the schemas, such as typechecking of XML transformations [13, 26]. Minimal *deterministic* automata for unranked tree languages play a prominent role in recent approaches to query induction for Web information extraction [3]. The objective is to identify a tree automaton for a previously unknown target language from given examples. Standard algorithms from grammatical inference [1, 8, 19] such as RPNI always induce minimal deterministic automata. The smaller this automaton is, the easier it can be inferred.

In this work we focus on the minimization of *automata for unranked tree languages*, which is a fundamental problem to automata theory and recently attracted some attention [6, 21]. The question is particularly relevant for classes of *deterministic automata*, since minimization can be done both efficiently and leads to unique canonical representatives of regular languages, as is well-known

⋆ Corresponding author

G. Bierman and C. Koch (Eds.): DBPL 2005, LNCS 3774, pp. 232–246, 2005.

for string languages and ranked tree languages. It is also well-known that minimal non-deterministic automata are neither unique, nor efficiently computable [9, 11].

The investigation of efficient minimization of deterministic automata for unranked trees language started quite recently [6, 21]. The deterministic devices considered there, however, differ from the standard deterministic automata in database theory – the bottom-up deterministic unranked tree automata (UTAs) of Brüggemann-Klein, Murata, and Wood [2]. In this paper, we investigate efficient (i.e. PTIME) minimization starting from such UTAs.

The transition relation of UTAs uses regular string languages over the states of the automaton to express horizontal recursion. However, it is not specified how these regular string languages should be represented. In practice, this is usually done by finite automata or regular expressions. If we allow for non-deterministic finite automata in bottom-up deterministic UTAs, then minimization becomes PSPACE-hard. As we are interested in *efficient* minimization, we restrict the finite subautomata in UTAs to be deterministic too. These DFAs impose *left-to-right deterministism* in addition to bottom-up determinism.

In the first part of the paper, we will prove two surprising results for these bottom-up and left-to-right deterministic UTAs. We present a counterexample for the uniqueness of minimal UTAs that represent a given regular language. We then prove that minimization becomes NP-complete and thus unfeasible. Both results are in strong contrast to what is known for bottom-up deterministic automata in the ranked case. Our NP-hardness proof refines the proof techniques from [9, 11], showing NP-hardness of minimization for classes of finite automata with limited amount of non-determinism.

In the second part of the paper, we compare the sizes of minimal automata for known automata classes that allow for efficient minimization. We show that bottom-up deterministic *stepwise tree automata* [4] yield the most succinct representations, both compared to the bottom-up deterministic *parallel UTAs* of [6, 21], as well as with respect to bottom-up deterministic automata over the standard *first-child next-sibling encoding* of regular tree languages (up to inversion). The difference in representation size is quadratic in the first case and exponential in the second case.

Finally we discuss a small minimization result for top-down deterministic tree automata. This notion of top-down determinism is very similar to the notion defined in [6] as it has exactly the same expressive power – but the question of minimizing these automata was not treated.

2 Preliminaries

In this section we provide the necessary background on strings, trees and tree automata.

2.1 Strings

For a finite set S, its *size* $|S|$ is its number of elements. By $\mathbb{N}$ we denote the set of natural numbers. We fix a finite alphabet Σ. When $a \in \Sigma$ we also say that

a is a Σ-*symbol*. A *string* $w = a_1 \cdots a_n$ is a finite sequence of Σ-symbols. We denote the empty string by ε.

We assume familiarity with nondeterministic finite automata (NFAs), deterministic finite automata (DFAs), unambiguous finite automata (UFAs) and regular expressions (REs). Given a fininte automaton or a regular expression A, we sometimes freely identify A with the language $L(A)$ it defines. The *size* of a finite automaton or regular expression is the size of its state set, or its number of symbols respectively. Let $\mathcal{C}$ be a class of representations of regular string languages (that is, NFAs, DFAs, UFAs, or REs). Then the *minimization problem* for $\mathcal{C}$ is defined as follows: Given an $A \in \mathcal{C}$ and an integer m, does there exist an $A' \in \mathcal{C}$ such that A and A' accept the same language and the size of A' is lesser than or equal to m. The *containment* and *equivalence problems* for $\mathcal{C}$ ask, given $A, B \in \mathcal{C}$ whether $L(A) \subseteq L(B)$ or $L(A) = L(B)$ respectively. We recall the following results from formal language theory:

Theorem 1 ([9, 24, 25]).

(1) Containment and equivalence of NFAs and REs is PSPACE*-complete;*
(2) Containment and equivalence of UTAs and DFAs is in PTIME*;*
(3) Minimizing NFAs and REs is PSPACE*-complete;*
(4) Minimizing UFAs is NP*-complete;*
(5) Minimizing DFAs is in PTIME*.*

2.2 Unranked Trees

The set of unranked Σ-trees, denoted by $\mathcal{T}$, is the smallest set of strings over Σ and the parenthesis symbols ')' and '(' such that for each $a \in \Sigma$ and $w \in \mathcal{T}^*$, $a(w)$ is in $\mathcal{T}$. So, a tree is either ε (empty) or is of the form $a(t_1 \cdots t_n)$ where each t_i is a tree. The latter denotes the tree where the subtrees $t_1, \ldots, t_n$ are attached to the root labeled a. We write a rather than $a()$. Note that there is no a priori bound on the number of children of a node in a Σ-tree; such trees are therefore *unranked*. In the following, whenever we say tree, we always mean Σ-tree. A *tree language* is a set of trees.

For every tree $t \in \mathcal{T}$, the *set of nodes of* t, denoted by $\mathrm{Dom}(t)$, is the subset of $\mathbb{N}^*$ defined as follows: (i) if $t = \varepsilon$, then $\mathrm{Dom}(t) = \emptyset$; and (ii) if $t = a(t_1 \cdots t_n)$ where each $t_i \in \mathcal{T}$, then $\mathrm{Dom}(t) = \{\varepsilon\} \cup \bigcup_{i=1}^{n} \{iu \mid u \in \mathrm{Dom}(t_i)\}$. For every $u \in \mathrm{Dom}(t)$, we denote by $\mathrm{lab}^t(u)$ the label of u in t.

2.3 Unranked Tree Automata

Definition 2 ([2]). An *unranked tree automaton (UTA)* is a tuple $B = (Q, \Sigma, \delta, F)$, where Q is a finite set of states, $F \subseteq Q$ is the set of final states, and δ is a function $\delta : Q \times \Sigma \to 2^{(Q^*)}$ such that $\delta(q, a)$ is a regular string language over Q for every $a \in \Sigma$ and $q \in Q$.

To simplify notation, we sometimes also write $a(L) \to q$ for $\delta(q, a) = L$. A *run* of B on a tree t is a labeling $\lambda : \mathrm{Dom}(t) \to Q$ such that for every $v \in \mathrm{Dom}(t)$

with n children we have that $\lambda(v1)\cdots\lambda(vn) \in \delta(\lambda(v), \mathrm{lab}^t(v))$. Note that when v has no children, the criterion reduces to $\varepsilon \in \delta(\lambda(v), \mathrm{lab}^t(v))$. A run is *accepting* iff the root is labeled with an accepting state, that is, $\lambda(\varepsilon) \in F$. A tree is *accepted* if there is an accepting run. The set of all accepted trees is denoted by $L(B)$ and is called a *regular tree language*. A UTA is *bottom-up deterministic* if for all $q, q' \in Q$ with $q \neq q'$ and $a \in \Sigma$ we have that $\delta(q, a) \cap \delta(q', a) = \emptyset$.

When defining the *size* of a UTA, we have to fix a representation of the regular languages $\delta(q, a)$. As argued in the introduction, we represent $\delta(q, a)$ by a DFA since non-deterministic representations immediately make the minimization problem intractable (Theorem 1). We denote by DUTA the bottom-up deterministic UTAs where the transitions $\delta(q, a)$ are represented by DFAs. As DFAs are deterministic when reading a string from left to right, we also refer to DUTAs as *bottom-up left-to-right deterministic UTAs* . The *size* of a DUTA $B = (Q, \Sigma, \delta, F)$ is $|Q| + \sum_{(q,a)} |\delta(q, a)|$, where $|\delta(q, a)|$ is the number of states of the DFA accepting $\delta(q, a)$.

We mention the following basic result about DUTAs.

Theorem 3. *Containment and equivalence of DUTAs is in* PTIME.

Proof (Sketch). Given two DUTAs, we can translate them in PTIME into tree automata over a known binary encoding of unranked trees, such as the first-child next-sibling encoding. The canonical way to do this translates a DUTA into an *unambiguous tree automaton over binary trees*. Due to the work of Seidl [23], we can test containment and equivalence of these automata in PTIME. □

To the best of our knowledge, it is not known whether the standard containment test works in PTIME, since complementing a DUTA is *not* trivial (unless the DUTA is *complete*, then one just has to switch final and non-final states).

2.4 Are DUTAs Deterministic?

We raise the question whether the computation of a DUTA is truly deterministic or not. Informally, we assume that a *computation* of a DUTA proceeds in a bottom-up manner, reads every node of a tree only once and remembers only one state of an internal DFA while reading the states that are assigned to the children of a certain node. We show in a small example that under these conditions, the computation of a DUTA in fact still has a very limited form of non-determinism. In the next section we show that this is exactly what makes minimization hard.

Let A be the DUTA with transition function

$$\delta(q_a, a) = \delta(q_b, b) = \varepsilon$$

$$\delta(q_1, r) = \{q_a q_a\} \quad \text{by} \quad \xrightarrow{q_a}\ 1 \xrightarrow{q_a}\ 2 \xrightarrow{}\ (3)$$

$$\delta(q_2, r) = \{q_a q_b\} \quad \text{by} \quad \xrightarrow{q_a}\ 4 \xrightarrow{q_b}\ 5 \xrightarrow{}\ (6)$$

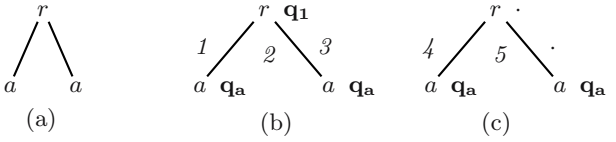

Fig. 1. A tree t, a successful run on t and a partially successful run on t

and final states $\{q_1, q_2\}$. This automaton accepts the language $\{r(aa), r(ab)\}$. When computing a bottom-up run for the tree in Figure 1(a), the state q_a will be assigned to both a-labeled leafs. At that point, it remains to assign a state to the r-labeled root. Here, we have the choice of starting to run the DFA for $\delta(q_1, r)$ or for $\delta(q_2, r)$. In Figure 1(b), we show the run that we obtain by choosing $\delta(q_1, r)$ (in which we also annotated the internal states $1, 2$ and 3 of $\delta(q_1, r)$ in italic), and Figure 1(c) shows the partial run that is obtained when choosing $\delta(q_2, r)$, which cannot be completed to a successful run. So even though there is only one successful run, the computation of the run itself still has a limited choice. Intuitively, this corresponds to an *unambiguous* rather than completely deterministic automata model.

One could argue that this choice in the computation is implementation-dependent. When implementing the automaton A, one could e.g. choose to simulate $\delta(q_1, r)$ and $\delta(q_2, r)$ *in parallel*. But then, one actually obtains a different notion of UTAs, namely the *parallel UTAs* [6, 21] that we study in Section 4.4.

3 Minimizing UTAs

In this section we study the minimization problem on bottom-up left-to-right deterministic UTAs. We show two unexpected negative results: Given a regular tree language L, there does *not* exist an (up to isomorphism) unique minimal DUTA that accepts L. The minimization problem for DUTAs even turns out to be NP-complete.

3.1 Minimal Automata are Not Unique

We show the non-uniqueness by an example. Consider the regular languages L_1, L_2 and L_3 defined by regular expressions $(bbb)^*$, $b(bbbbbb)^*$ and $bb(bbbbbbbbb)^*$ respectively. Note that L_1, L_2 and L_3 are pairwise disjoint, and that the minimal DFAs A_1, A_2 and A_3 accepting L_1, L_2 and L_3 have 3, 6 and 9 states respectively. It is easy to verify that the minimal DFAs B_1 and B_2 accepting $L_1 \cup L_2$ and $L_1 \cup L_3$ have 6 and 9 states respectively. Let $L = L_1 \cup L_2 \cup L_3$ and consider the tree language $T := \{r(a(w)) \mid w \in L\}$.

There exist two minimal DUTAs for T. The first one, $N_1 = (Q_1, \Sigma, \delta_1, F_1)$ has accept state q_0 and transition function

$$\delta_1(q_0, r) = q_1 + q_2 \qquad \delta_1(q_1, a) = B_1 \qquad \delta_1(q_2, a) = A_3 \qquad \delta_1(b, b) = \varepsilon.$$

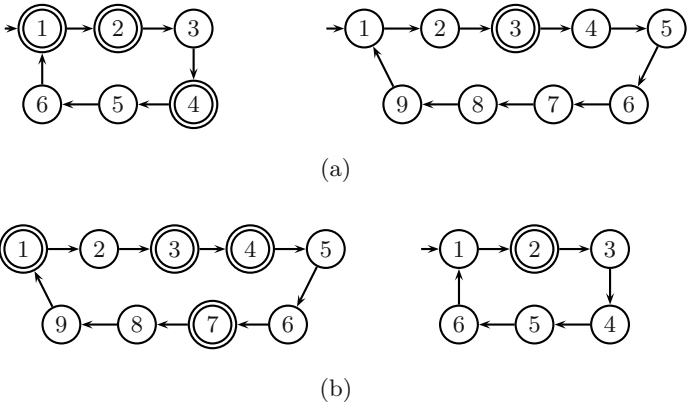

Fig. 2. Figure 2(a) contains the DFAs B_1 and A_3; and Figure 2(b) contains the DFAs B_2 and A_2. All transition arrows read the symbol b.

The size of N_1 is $|Q_1|+|\delta_1(q_0,r)|+|B_1|+|A_3|+|\delta_1(b,b)| = 4+2+6+9+1 = 22$. The DFAs B_1 and A_3 are sketched in Figure 2(a). The other automaton, $N_2 = (Q_2, \Sigma, \delta_2, F_2)$ has accept state q_0 and transition function

$$\delta_2(q_0,r) = q_1 + q_2 \qquad \delta_2(q_1,a) = B_2 \qquad \delta_2(q_2,a) = A_2 \qquad \delta_2(b,b) = \varepsilon.$$

The size of N_2 is $|Q_2|+|\delta_2(q_0,r)|+|B_2|+|A_2|+|\delta_2(b,b)| = 4+2+9+6+1 = 22$. The DFAs B_2 and A_2 are sketched in Figure 2(b).

Of course, there are other possibilities to write $L = L_1 \cup L_2 \cup L_3$ as a disjoint union of regular languages. The obvious combinations one can make with A_1, A_2 and A_3 lead to DUTAs of size 26 (using A_1, A_2 and A_3), 28 (using $(A_2 \cup A_3)$ and A_1) and 24 (one automaton for L).

We show that no other combination of splitting L into a union of regular languages will result in a smaller DUTA accepting T. First, observe that any DUTA defining T needs at least three states in its state set Q, since all trees in T have depth three. However, as argued above, the minimum size of such a DUTA with three states is $3+2+18+1 = 24$. The only way to obtain a smaller DUTA is then to define L as a union of DFAs, of which the sum of the number of states is strictly smaller than $9 + 6 = 15$. However, if we write L as a union of DFAs, there must be at least one DFA D_1 that accepts an infinite number of strings in L_2. It is easy to see that D_1 has at least 6 states, as D_1 may not accept strings *not* in L. Analogously, we can argue that there must be at least one DFA D_2 that accepts an infinite number of strings in L_3. If $D_1 = D_2$, it is easy to see that D_1 has at least 18 states. If $D_2 \neq D_1$, then it is easy to see that D_2 has at least 9 states. Therefore, the above automata are indeed minimal for T, and as Figure 2 shows, they are clearly not isomorphic.

3.2 Minimization Is NP-Complete

The minimization problem for DUTAs is defined analogously as the minimization problem for finite automata: Given a DUTA A and an integer m, decide whether

there exists a DUTA B such that $L(B) = L(A)$ and the size of B is lesser than or equal to m.

As Section 3.1 illustrates, the problem of defining a regular string language as a small disjoint union of DFAs lies at the heart of the minimization problem for DUTAs. We call this problem GENERAL MINIMUM DISJOINT UNION and define it formally as follows: Given a DFA M and an integer ℓ, do there exist DFAs $M_1, \ldots, M_n$ such that

(1) $L(M) = L(M_1) \cup \cdots \cup L(M_n)$; and
(2) for every $i \neq j$, $L(M_i) \cap L(M_j) = \emptyset$; and
(3) $\sum_{i=1}^{n} |M_i| \leq \ell$?

It can be shown that GENERAL MINIMUM DISJOINT UNION is NP-complete by a reduction from VERTEX COVER. Actually, GENERAL MINIMUM DISJOINT UNION is even NP-complete when $n = 2$. The proof for this is not straightforward, and technically the hardest proof in the paper, but the reduction is interesting in its own right. The proof can be found in [15]. Although we do not go deeper on this in the paper, variations of the reduction can actually be used to show that the three open problems stated in the conclusion of [11] are NP-complete [12].

Theorem 4. DUTA MINIMIZATION *is* NP*-complete.*

Proof (Sketch). The upper bound follows from Theorem 3. Given a DUTA A and an integer m, the NP algorithm simply guesses an automaton B of size at most m and verfies in PTIME whether it is equivalent to A.

For the lower bound, we do a reduction from GENERAL MINIMUM DISJOINT UNION. Given a DFA $M = (Q_M, \Sigma_M, \delta_M, I_M, F_M)$ and integer ℓ, we have to construct a DUTA A and an integer m such that A has an equivalent DUTA of size m iff M can be written as a disjoint union of DFAs for which the size does not exceed ℓ. Intuitively, we construct A such that it accepts the trees of the form $r(w)$, where the root node is labeled with a special symbol r and the string w is in $L(M)$. For the full proof, we refer to [15]. □

4 Solutions for Efficient Minimization

As we have shown, UTA minimization is unfeasible even when the horizontal languages are represented by DFAs. The problem is raised when using multiple rules for the same label, for recognizing these horizontal regular languages.

Three alternative notions of bottom-up deterministic tree automata for unranked trees were proposed recently, each of them yielding a solution to the problem. First, one can define notions of bottom-up determinism based on translations between unranked and ranked trees. *Stepwise tree automata* [4] are an algebraic notion of automata for unranked trees which also correspond to automata over binary trees by means of such a translation. Alternatively, one can use tree automata that operate on the *standard encoding* of unranked into binary trees (see, e.g. [7]).

Finally, *parallel UTAs (pUTAs)* alter the rule format of UTAs and have been independently proposed in [21] and [6]. All these automata yield notions of bottom-up determinism which lead to unique minimal automata and polynomial time minimization.

4.1 Automata on Binary Trees

We first treat the automata models which can also be defined on ranked trees. We therefore recall the notion of a traditional tree automaton.

Definition 5. A (traditional) *tree automaton (TA)* for a binary signature is a tuple $A = (Q, \Sigma, \delta, (I_a)_{a\in\Sigma}, F)$ where Q is a finite set of states, the signature $\Sigma = \Sigma_0 \uplus \Sigma_2$ consists of a finite set of constants Σ_0 and finite set of binary function symbols Σ_2, $F \subseteq Q$ is the set of final states, $I_a \subseteq Q$ is a set of initial states for every $a \in \Sigma_0$, and δ is a function $\delta : Q \times Q \times \Sigma_2 \to 2^Q$ mapping a pair of states and a function symbol to a set of possible new states.

A *run* of A on a tree t is a labeling $\lambda : \mathrm{Dom}(t) \to Q$ such that *(i)* for every leaf node u, $\lambda(u) \in I_{\mathrm{lab}(u)}$; and *(ii)* for all inner nodes u, $\lambda(u) \in \delta(\lambda(u1), \lambda(u2), \mathrm{lab}(u))$. A run is *accepting* iff the root is labeled with an accepting state, that is, $\lambda(\varepsilon) \in F$. A tree is *accepted* if there is an accepting run on t.

A binary tree automaton is *(bottom-up) deterministic* if for all $q, q' \in Q$ and $a \in \Sigma$, $\delta(q, q', a)$ contains at most one element. To simplify notation for TAs, we sometimes also write $a \to q$ and $a(q, q') \to p$ to say that $q \in I_a$ and $p \in \delta(q, q', a)$ respectively.

The *size* of a binary tree automaton $A = (Q, \Sigma, \delta, (I_a)_{a\in\Sigma}, F)$ is the number of elements in its state set Q. We denote the size of A by $|A|$.

4.2 Stepwise Tree Automata

Stepwise tree automata are an algebraic version of automata for unranked trees [4]. For the algebraic perspective, we refer to that paper.

A *stepwise tree automaton* over an unranked signature Σ is a (traditional) tree automaton over the binary signature $\Sigma \uplus \{@\}$ where all labels in Σ serve as constants and @ is a binary function symbol.

One of the nicest features of stepwise tree automata is that they are traditional tree automata, but can also run over *unranked trees*. Indeed, stepwisetree au-

initial states	$I_a = \{5\}$	$I_b = \{4\}$
rules	$@(5,4) \to 6$	$@(5,5) \to 6$
	$@(6,4) \to 6$	$@(6,5) \to 6$
final states	$\{5, 6\}$	

Fig. 3. A deterministic stepwise tree automaton for $a((a|b)^*)$, equivalent to the pUTA in Figure 5

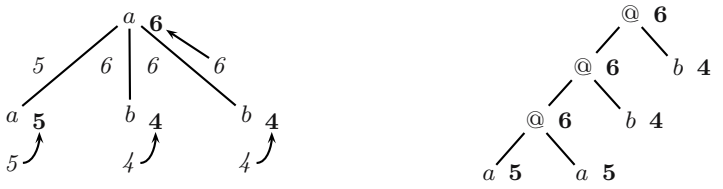

Fig. 4. Runs of the stepwise automaton from Figure 3. To the left, on the unranked tree and to the right, on its Curried binary encoding.

tomata can be understood as traditional tree automata that operate on Curried binary encodings of unranked trees. Currying the unranked tree

$$\text{plus}(4, 5, \text{plus}(6, 7, 8))$$

for instance yields plus@4@5@(plus@6@7@8) which (if we assume left associativity) is the binary tree

$$@\big(@\big(@(\text{plus}, 4), 5\big), @\big(@\big(@(\text{plus}, 6), 7\big), 8\big)\big),$$

but in infix notation. Figure 4 shows this correspondence. The italic states in the left tree correspond exactly to the bold states in the run on its Curried encoding, which is the tree on the right. On the unranked tree, the italic states can be seen as the explicit computation of the tree automaton. For every node, the state in bold is simply a copy of the rightmost italic state below. In Section 4.4, we show a run of an equivalent *parallel UTA* in Figure 5. There, the correspondence between the italic and the bold states is given by the *output function* of the automaton.

Myhill-Nerode Property. The Myhill-Nerode theorem yields an up to isomorphism unique representation for minimal deterministic automata for regular languages. The states of the minimal automaton correspond to the classes of the congruence induced by the language.

The Myhill-Nerode theorem holds generally for algebraic automata notions (see e.g. [5]) and thus for finite automata, standard tree automata [10, 27], and stepwise tree automata [4]. Myhill-Nerode inspired theorems for automata on unranked trees were shown for UTAs (Theorem G in [2]) and for *parallel UTAs* [6], which we treat in Section 4.4. Remarkably, in the former case the theorem does not lead to minimal automata and in the latter case the theorem uses two quite particular equivalence relations instead of a single canonical congruence, as in the ranked case. In this section, we formulate the Myhill-Nerode theorem, such that it holds both for traditional tree automata and stepwise tree automata interpreted over unranked trees.

In both cases, a *context* C is a function mapping trees to trees. In the case of binary trees over Σ, a context can be represented by a binary tree over the binary signature $\Sigma \uplus \{\bullet\}$ that contains a single occurence of the hole marker $\bullet$ at

a leaf node. Context application $C(t)$ to a tree t replaces the hole marker in C by t. A *context* C for an unranked tree over Σ is a tree over the unranked signature $\Sigma \uplus \{\bullet\}$ that contains a single occurence of the hole marker, but this time possibly labeling an internal node. Given a context C and a tree $t = a(t_1 \cdots t_n)$, we define context application $C(t)$ inductively as follows:

- $\bullet(t'_1, \cdots, t'_m)(a(t_1, \cdots, t_n)) = a(t_1, \cdots, t_n, t'_1, \cdots, t'_m)$
- $a(t'_1, \cdots, t'_i, \cdots, t'_m)(t) = a(t'_1, \cdots, t'_i(t), \cdots, t'_m)$ where t'_i contains the $\bullet$.

A *congruence* on trees is an equivalence relation $\equiv$ that satisfies for every context C: if $t_1 \equiv t_2$ then $C(t_1) \equiv C(t_2)$. It is of *finite index* when there are only a finite number of equivalence classes. Given a tree language L, we define the congruence $\equiv_L$ induced by L through:

$$t_1 \equiv_L t_2 \text{ iff for every context } C\text{: } C(t_1) \in L \Leftrightarrow C(t_2) \in L.$$

Theorem 6 (Myhill-Nerode). *For any ranked or unranked tree language L it holds that L is a regular tree language iff its congruence $\equiv_L$ has finite index. Furthermore, there exists an (up to isomorphism) unique minimal bottom-up deterministic (stepwise) tree automaton for all regular languages L. The size of this automaton is equal to the index of $\equiv_L$.*

The proof of this theorem is immediate from the binary case [10]. It follows from the observation that the contexts we define for unranked trees are obtained by translating the contexts over binary trees through the inverse of the Curried encoding. We note that this theorem was partially proven in [2] (Theorem G). However, we feel that the present proof is simpler (as the theorem immediately carries over from the ranked case) and also leads to minimal automata, which is not the case in [2].

4.3 Standard Binary Encoding

Analogously as with stepwise tree automata, a tree automaton over the standard *first-child next-sibling* encoding of a tree can be seen as working directly over the unranked tree (see, e.g. [7]). In this encoding, an unranked tree is simply viewed as a binary tree over the first-child and next-sibling relation. Whenever a Σ-symbol has no left or no right child, a special symbol $\perp$ is inserted.

As in Section 4.2, a Myhill-Nerode theorem for unranked trees can also be obtained using the inverse of the standard first-child next-sibling encoding. However, in Section 5.1, we argue that the latter leads to exponentially more equivalence classes, and hence, exponentially larger minimal automata.

4.4 Parallel UTAs

The problem with UTAs bottom-up left-to-right determinism is that it may force the interpreter of the automaton to choose between several rules for the same label. The obvious solution is to run all automata for the same label in parallel, and thus unify them. This idea leads to the notion of parallel UTAs.

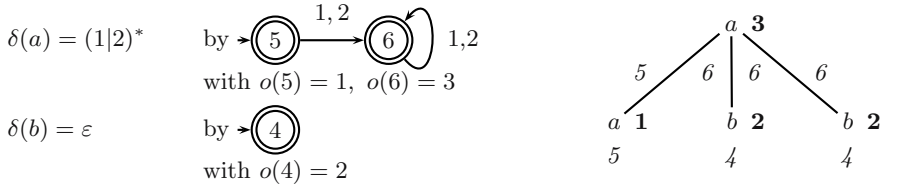

Fig. 5. A deterministic pUTA $(Q, \Sigma, \delta, \{1, 3\}, o)$ for $a((a|b)^*)$, equivalent to the stepwise tree automaton in Figure 3, and its run on $a(abb)$, annotated in bold to the right of the alphabet symbols. The runs of the internal DFAs are annotated in italic.

A *parallel UTA (pUTA)* is a tuple $A = (Q, \Sigma, \delta, F, o)$ which consists of a finite set Q of states, a collection of horizontal regular languages $\delta(a) \subseteq Q^*$ represented by a finite automaton for each $a \in \Sigma$, a set of final states $F \subseteq Q$, and a collection of output functions $o(a)$ for all $a \in \Sigma$ that maps final states of the finite automaton recognizing $\delta(a)$ to states in Q.

Let $(Q_a, Q, \delta_a, I_a, F_a)$ be the finite automaton recognizing $\delta(a)$ for every $a \in \Sigma$. A run of A on a tree t is a labeling function $\lambda : \mathrm{Dom}(t) \to Q$ that satisfies for all nodes $u \in \mathrm{Dom}(t)$ with n children that $\lambda(u) = o(a)(\delta_a^*(\lambda(u1) \cdots \lambda(un)))$, where δ_a^* is the homomorphic extension of δ_a to strings in Q^*. An example for a pUTA is illustrated in Figure 5.

We call a pUTA *deterministic* if all subautomata for recognizing horizontal languages $\delta(a)$ are DFAs. The class of deterministic pUTAs has unique minimal automata and allows for efficient minimization [6, 21]. They can recognize all regular languages, as shown by the following transformations.

Every deterministic pUTA A can be transformed into an equivalent DUTA (Q, Σ, δ', F) so that for every $a \in \Sigma$ and $q \in Q$:

$$\delta'(q, a) = (Q_a, Q, \delta_a, I_a, o(a)^{-1}(q))$$

Conversely, any DUTA can be converted into an equivalent deterministic pUTA but possibly at the cost of an exponential blow-up. The first step is to unify the horizontal subautomata for the language $\cup_{q \in Q} \delta(q, a)$ for every label a, by constructing the product automaton (which can cause the exponential blow-up). The output function $o(a)$ maps every tuple in the product with a final state of the automaton recognizing $\delta(q, a)$ to q.

5 Size Comparison

5.1 Stepwise vs Standard Binary Encoding

Let $[\![t]\!]$ denote the standard binary *first-child next-sibling* encoding of an unranked Σ-tree t over $\Sigma \uplus \{\bot\}$. Let $\overline{t}$ denote the tree obtained from t by reversing for every node its list of children. For instance, if $t = a(b, c, d(e, f))$, then $\overline{t} = a(d(f, e)c, b)$. We extend these notations in the obvious way to tree languages.

To be able to compare stepwise tree automata to tree automata over the first-child next-sibling encoding, we need to study one of the two over these reversed trees. The reason is that deterministic stepwise tree automata read the children of a node from *left to right* whereas deterministic tree automata over the standard encoding read them from *right to left*, which leads to an exponential blow-up of the minimal size in both directions. The witness tree languages for this claim are based on the languages $L((a+b)^n a(a+b)^*)_{n\in\mathbb{N}}$, which cause an exponential blow-up for DFAs which read strings from right to left. Here we take the standard encoding over the reversed trees.

Lemma 7. *For all unranked trees t_1, t_2, if $[\![\overline{t_1}]\!] \equiv_{[\![\overline{L}]\!]} [\![\overline{t_2}]\!]$ then $t_1 \equiv_L t_2$.*

Proof (Sketch). This follows from the definitions of the encoding and the contexts: any context for a tree in the standard encoding can be obtained by translating a context in the Curried encoding. □

Proposition 8. *The minimal bottom-up deterministic stepwise tree automaton for an unranked regular language L is never larger than the minimal bottom-up deterministic tree automaton for the inverted standard encoding $[\![\overline{L}]\!]$.*

Proof. Due to the Myhill-Nerode Theorem 6 it is sufficient to compare the indexes of the congruences $\equiv_L$ and $\equiv_{[\![\overline{L}]\!]}$. By Lemma 7, if two trees are in different equivalence classes of $\equiv_L$ then their encodings will be different equivalent classes of $\equiv_{[\![\overline{L}]\!]}$, i.e., the index of L smaller or equal than the index of $[\![\overline{L}]\!]$. □

Proposition 9. *There exists an infinite class of languages $(L_i)_{i\in\mathbb{N}}$ such that for every L_i, the minimal bottom-up deterministic stepwise tree automaton for L_i is exponentially more succinct than the minimal bottom-up deterministic tree automaton for the encoding $[\![\overline{L_i}]\!]$.*

Proof (Sketch). The proof is based on the fact that the smallest DFA for the union of DFAs $A_1, \ldots, A_n$ can be exponentially larger than the sum of the sizes $|A_1| + \cdots + |A_n|$. □

5.2 Stepwise vs Parallel UTAs

We mention the following proposition without proof:

Proposition 10. *Minimal deterministic stepwise tree automata are always smaller or equal than minimal deterministic parallel UTAs for the same language.*

It is easy to see that translating a stepwise automaton to a pUTA gives at most a quadratic blow-up. This upper bound is also tight:

Proposition 11. *There exists an infinite class of languages $(L_i)_{i\in\mathbb{N}}$ such that for every L_i, the minimal bottom-up deterministic stepwise tree automaton is quadratically more succinct than the minimal deterministic pUTA.*

Proof. The lemma holds for $L_n = \{a_1(a^n), \ldots, a_n(a^n)\}$. pUTAs need n different automata of size n to accept the string a^n, so their minimal size is $\mathcal{O}(n^2)$. Stepwise automata can share the state sets of these, so their minimal size is $\mathcal{O}(n)$. □

6 Top-Down Deterministic UTAs

We briefly discuss a minimization result for top-down deterministic UTAs. According to the definition of Brüggemann-Klein, Murata and Wood, a UTA $A = (Q, \Sigma, \delta, F)$ is top-down deterministic if for all $q \in Q$, $a \in \Sigma$, and $n \geq 0$, $\delta(q, a)$ contains at most one string of length n [2]. We show that a more expressive form of top-down deteriminism still allows for *(i)* a PTIME minimization algorithm and *(ii)* uniqueness up to isomorphism of the minimal automaton. This notion of top-down determinism not only allows to take into account the number of siblings but also their labeling. It is very similar to the notion defined by Cristau, Löding and Thomas [6].

To define the notion of top-down determinism, we assume that there is a function $f : Q \to \Sigma$ that associates to each state the unique alphabet symbol it can be assigned to in a run of the automaton. The idea from this function stems from *specialized DTDs* [20], which are always provided by such a function. The results in this section therefore directly carry over onto specialized DTDs. We extend this function f in the obvious way to strings over Q.

The main motivation of this section lies in XML schema languages. Indeed, the proposed notion of top-down determinism is strictly more powerful than the notions of *single-type* and *restrained competition* specialized DTDs [16], which correspond to the expressive power of XML Schema [28] and 1-pass preorder typing [14] respectively. It is not hard to see that the proposed minimization algorithm preserves the single-type and restrained competition properties and hence, as a corollary, minimization of single-type and restrained competition specialized DTDs (in which the internal regular languages are represented by DFAs) is also in PTIME.

We call a UTA $A = (Q, \Sigma, \delta, F)$ *top-down deterministic* if every language defined by a DFA D representing $\delta(q, a)$ has the following property: if w and w' in $L(D)$ and $f(w) = f(w')$ then $w = w'$.

Theorem 12. *Every top-down deterministic UTA can be minimized in* PTIME. *This minimal top-down deterministic UTA is unique up to isomorphism.*

We briefly sketch the minimization algorithm. Let $A = (Q, \Sigma, \delta, F)$ be top-down deterministic with mapping $f : Q \to \Sigma$. Given a state $q \in Q$, we denote by $L(A, q)$ the language accepted by $(Q, \Sigma, \delta, \{q\})$. The following algorithm minimizes A:

(1) *Trim* A, that is, remove all unreachable states from Q, and remove all $q \in Q$ for which $L(A, q) = \emptyset$, and their corresponding transitions.

(2) Test, for each q_i and q_j in Q, $i \neq j$, whether $L(A, q_i) = L(A, q_j)$. If $L(A, q_i) = L(A, q_j)$, then replace all occurrences of q_j in the definition of δ by q_i, remove the transition $\delta(q_j, f(q_j))$, and remove q_j from Q.
(3) For each $q \in Q$, minimize the DFA representing $\delta(q, f(q))$.

7 Conclusions

We have shown that the minimization problem for DUTAs (bottom-up deterministic UTAs in which the languages in the transition function are represented by DFAs) is NP-complete. The reason behind this hardness result is that these DUTAs are not truly deterministic. Indeed, DUTAs still allow to represent regular languages over states by a *disjoint union of DFAs*, as exemplified in Section 3.1. Furthermore, the canonical translations of DUTAs over the known ranked encodings result in *unambiguous* rather than deterministic binary tree automata.

A second contibution of the paper is a comparison between several notions of determinism for unranked tree automata. We compared three different notions: parallel UTAs, which were defined independently in [6] and [21], stepwise tree automata [4], and ranked tree automata over the *first-child next-sibling* encoding. In general, the stepwise tree automata provide the smallest minimal automata. Moreover, since they have a direct connection to traditional ranked tree automata through an encoding based on currying, a PTIME minimization algorithm and a Myhill-Nerode theorem is immediate.

Acknowledgments

We thank Frank Neven and Thomas Schwentick for helpful discussions and comments on a previous version of the paper.

References

1. D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, 1987.
2. A. Brüggemann-Klein, M. Murata, and D. Wood. Regular tree and regular hedge languages over unranked alphabets: Version 1, april 3, 2001. Technical Report HKUST-TCSC-2001-0, The Hongkong University of Science and Technology, 2001.
3. J. Carme, A. Lemay, and J. Niehren. Learning node selecting tree transducers from completely annotated examples. In *ICGI 2004*, pages 91–102, 2004.
4. J. Carme, J. Niehren, and M. Tommasi. Querying unranked trees with stepwise tree automata. In *RTA 2004*, pages 105–118, 2004.
5. B. Courcelle. On recognizable sets and tree automata. In *Resolution of equations in algebraic structures*, pages 93–126, 1989.
6. J. Cristau, C. Löding, and W. Thomas. Deterministic automata on unranked trees. In *FCT 2005*, 2005. To Appear.
7. M. Frick, M. Grohe, and C. Koch. Query evaluation on compressed trees (extended abstract). In *LICS 2003*, pages 188–197, 2003.

8. E.M. Gold. Complexity of automaton identification from given data. *Inform. Control*, 37:302–320, 1978.
9. T. Jiang and B. Ravikumar. Minimal NFA problems are hard. *SIAM Journal on Computing*, 22(6):1117–1141, 1993.
10. D. Kozen. On the Myhill-Nerode theorem for trees *Bulletin of the European Association for Theoretical Computer Science*, 147:170–173, 1992.
11. A. Malcher. Minimizing finite automata is computationally hard. *Theoretical Computer Science*, 327(3):375–390, 2004.
12. W. Martens. On minimizing finite automata with very little non-determinism. Manuscript, 2005.
13. W. Martens and F. Neven. Frontiers of tractability for typechecking simple XML transformations. In *PODS 2004*, pages 23–34, 2004.
14. W. Martens, F. Neven, and T. Schwentick. Which XML schemas admit 1-pass preorder typing? In *ICDT 2005*, pages 68–82, 2005.
15. W. Martens and J. Niehren. Minimizing Tree Automata for Unranked Trees. Full Version. http://www.uhasselt.be/wim.martens/pubs.html
16. M. Murata, D. Lee, M. Mani, and K. Kawaguchi. Taxonomy of XML schema languages using formal language theory. *ACM Transaction on Internet Technology*, 5(4), 2005. To Appear.
17. F. Neven and T. Schwentick. Expressive and efficient pattern languages for tree-structured data. In *PODS 2000*, pages 145–156, 2000.
18. F. Neven and T. Schwentick. Query automata on finite trees. *Theoretical Computer Science*, 275:633–674, 2002.
19. J. Oncina and P. Garcia. Inferring regular languages in polynomial update time. In *Pattern Recognition and Image Analysis*, pages 49–61, 1992.
20. Y. Papakonstantinou and V. Vianu. DTD inference for views of XML data. In *PODS 2000*, pages 35–46. ACM Press, 2000.
21. S. Raeymaekers and M. Bruynooghe. Minimization of finite unranked tree automata. Manuscript, 2004.
22. T. Schwentick. XPath query containment. *Sigmod Record*, 33(2):101–109, 2004.
23. H. Seidl. Deciding equivalence of finite tree automata. *SIAM Journal on Computing*, 19(3):424–437, 1990.
24. R. E. Stearns and H. B. Hunt III. On the equivalence and containment problems for unambiguous regular expressions, regular grammars and finite automata. *SIAM Journal on Computing*, 14(3):598–611, 1985.
25. L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time: Preliminary report. In *STOC 1973*, pages 1–9, 1973.
26. D. Suciu. Typechecking for semistructured data. In *DBPL 2001*, pages 1–20, 2001.
27. J.W. Thatcher and J.B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical Systems Theory*, 2(1):57–81, 1968.
28. World Wide Web Consortium. XML Schema http://www.w3.org/XML/Schema