Minimizing Tree Automata for Unranked Trees

Wim Martens, Joachim Niehren

On the Minimization of XML Schemas and Tree Automata for Unranked Trees ⋆

Wim Martens [a,*] and Joachim Niehren [b]

[a] *Hasselt University and Transnational University of Limburg, B-3590 Diepenbeek, Belgium*

[b] *INRIA Futurs, Lille, France*

Abstract

Automata for unranked trees form a foundation for XML schemas, querying and pattern languages. We study the problem of efficiently minimizing such automata. First, we study unranked tree automata that are standard in database theory, assuming bottom-up determinism and that horizontal recursion is represented by deterministic finite automata. We show that minimal automata in that class are not unique and that minimization is NP-complete. Second, we 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. Third, we investigate abstractions of XML schema languages. In particular, we show that the class of one-pass preorder typeable schemas allows for polynomial time minimization and unique minimal models.

Key words: minimization, unranked tree automata, XML schema languages

1 Introduction

The concept of unranked regular tree languages lies at the formal basis for many XML schema languages such as DTD, XML Schema, and Relax NG. However, both DTD and XML Schema lack the expressive power to define

⋆ An extended abstract of this paper appeared as reference [19] in the Tenth International Symposium on Database Programming Languages (DBPL 2005).

* Corresponding author.

Email addresses: wim.martens@uhasselt.be (Wim Martens), www.lifl.fr/~niehren (Joachim Niehren).

every unranked regular tree language (see, e.g. [20,18] for more details). This situation is different for Relax NG. Not only is the design of Relax NG based on unranked tree automata theory, validators for Relax NG are also often implemented as tree automata [36].

Tree automata for unranked trees are not only useful in the area of schema languages. They are used as a toolbox in numerous areas of XML-related research such as path and pattern languages [22,28] and XML querying [10,23]. The focus of the present article is on studying the problem of efficiently minimizing such automata.

The problem of minimizing the number of states of a finite unranked tree automaton is particularly relevant for classes of *deterministic automata*, since, for these automata, minimization can be done both efficiently and leads to unique canonical representatives of regular languages, as is well-known for string languages and ranked tree languages. It is also well-known that minimal nondeterministic automata are neither unique, nor efficiently computable [14,16].

Besides being a fundamental problem of theoretical interest, the minimization problem for tree automata or for XML schemas also has its use in practical applications. In the context of XML schema languages, minimized schemas would improve the running time or memory consumption for document validation. For static tests involving schemas, such as typechecking for XML transformations (see, e.g., [17,33]), a schema minimizer can be used as a preprocessor to improve the running time of the typechecker. Minimal *deterministic* automata for unranked tree languages play a prominent role in recent approaches to query induction for Web information extraction [4]. The objective is to identify a tree automaton for a previously unknown target language from given examples. Standard algorithms from grammatical inference [1,11,24] such as RPNI always induce minimal deterministic automata. The smaller this automaton is, the easier it can be inferred.

The investigation of efficient minimization of bottom-up deterministic automata for unranked tree languages started quite recently [9,27]. 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 [3]. In this article, we investigate efficient [1] 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

[1] That is, PTIME, under the assumption that PTIME $\neq$ NP.

minimization becomes PSPACE-hard, because minimization is already PSPACE-hard for the non-deterministic finite automata. As we are interested in *efficient* minimization, we restrict the finite subautomata in UTAs to be deterministic too. These deterministic finite automata (dFAs) impose *left-to-right determinism* in addition to bottom-up determinism.

We prove two unexpected results for these bottom-up and left-to-right deterministic UTAs. First, we present a counterexample for the uniqueness of minimal UTAs that represent a given regular language. Second, we prove that minimization becomes NP-complete. 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 [14,16], showing NP-hardness of minimization for classes of finite automata with a limited amount of nondeterminism.

Even though minimization for bottom-up and left-to-right deterministic UTAs is intractable, there exist automata models for unranked trees that do allow for efficient minimization. Examples of such models are *stepwise tree automata* [5], *parallel tree automata* [9,27], and bottom-up deterministic automata over the standard *first-child next-sibling encoding* of regular tree languages. As each of these models allows for tractable minimization and unique minimal representatives, we compare the models in terms of succinctness. We obtain that stepwise tree automata yield the smallest representations of unranked tree languages. In general, they are quadratically smaller than parallel tree automata and exponentially smaller than tree automata over the first-child next-sibling encoding (up to inversion).

Finally, we investigate models for unranked trees which form a theoretical basis for XML schema languages. In database theory, XML Schema Definitions [30] are abstracted as *single-type extended DTDs* [18,20], which are, from a structural point of view, not very expressive. More expressive is the so-called class of *restrained competition extended DTDs* [18,20], which captures precisely the class of schemas that can be validated and correctly typed in a one-pass preorder manner. We provide a polynomial time minimization algorithm for the latter class and show that this class gives rise to unique minimal models. Moreover, when given an input that satisfies the single-type restriction, the minimization algorithm outputs a minimal single-type model. It therefore also minimizes single-type extended DTDs.

2 Complexity of Minimization

We introduce automata for strings, binary trees and unranked trees, and present an overview over existing and new complexity results for automata

minimization.

2.1 Automata Notation and Problems

We explain the generic notation that we will use throughout the paper. For a finite set S, we denote by $|S|$ its number of elements. Let Σ be a finite alphabet. We consider data structures built from Σ that may be of different types, either strings, binary trees, or unranked trees. We write D_Σ for the set of all data structures of the given type that can be built from Σ. For every $d \in D_\Sigma$, we will define a set $\mathrm{nodes}(d)$ and a designated element $\mathrm{root}(d) \in \mathrm{nodes}(d)$, which will be the root of a tree or the last letter of a string.

We will consider different classes of automata for different data types. An automaton A will always be a tuple containing a finite set $\mathrm{alphabet}(A)$ of alphabet symbols ranged over by a, b, c, a finite set $\mathrm{states}(A)$ which we denote by p, q, and a set $\mathrm{final}(A) \subseteq \mathrm{states}(A)$ of final states. The size $|A|$ of A is a natural number, which will by default be the number states of A:

$$|A| = |\mathrm{states}(A)| \qquad \text{if not stated otherwise.}$$

A run of an automaton A on a data structure $d \in D_{\mathrm{alphabet}(A)}$ will always be defined as some function of type $r : \mathrm{nodes}(d) \to \mathrm{states}(A)$. This allows to infer an evaluation function of type $\mathrm{eval}_A : D_{\mathrm{alphabet}(A)} \to \mathrm{states}(A)$ as follows:

$$\mathrm{eval}_A(d) = \{r(\mathrm{root}(d)) \mid r \text{ is a run of } A \text{ on } d\}.$$

A run r of A on d is *accepting* or *successful* if $r(\mathrm{root}(d)) \in \mathrm{final}(A)$. The language $L(A)$ of an automaton is the set of data structures d that permit a successful run by A:

$$L(A) = \{d \in D_{\mathrm{alphabet}(A)} \mid \text{there exists a successful run } r \text{ of } A \text{ on } d\}.$$

Unless otherwise mentioned, an automaton A is *unambiguous* if it permits exactly one accepting run for every data structure $d \in L(A)$.

The central decision problem of this article is the *minimization problem*, which is parametrized by a class $\mathcal{C}$ of automata. Minimization is closely related to equivalence, inclusion, and universality. We define these problems formally.

MINIMIZATION: Given an automaton $A \in \mathcal{C}$ and a natural number $m \in \mathbb{N}$, does there exist an $A' \in \mathcal{C}$ such that $L(A) = L(A')$ and the size of A' is at most m?
EQUIVALENCE: Given $A, B \in \mathcal{C}$, does $L(A) = L(B)$ hold?
INCLUSION: Given automata $A, B \in \mathcal{C}$, does $L(A) \subseteq L(B)$ hold?
UNIVERSALITY: Given an automaton $A \in \mathcal{C}$, does $D_{\mathrm{alphabet}(A)} \subseteq L(A)$ hold?

We say that an automaton $A \in \mathcal{C}$ is *minimal* if MINIMIZATION is false for A and every $m < |A|$.

The minimization problem for a class $\mathcal{C}$ of automata can be solved by an NP(EQUIVALENCE $(\mathcal{C})$)-algorithm, that is, a nondeterministic polynomial time algorithm with an oracle able to solve the equivalence problem of $\mathcal{C}$. Given A and m it is sufficient to guess another automaton A' with size at most m and to test whether $L(A) = L(A')$.

As we will see in Table 1, it often holds that UNIVERSALITY is easier than MINIMIZATION. This will be useful to prove lower bounds for MINIMIZATION problems.

2.2 Strings and Finite Automata

By $\mathbb{N}$ we denote the set of natural numbers and by $\mathbb{N}_0$ we denote $\mathbb{N} - \{0\}$. By Σ we always denote a finite alphabet. We call $a \in \Sigma$ a Σ-*symbol*. A Σ-*string* (or simply *string*) $w \in \Sigma^*$ is a finite sequence $a_1 \cdots a_n$ of Σ-symbols. We denote the empty string by ε.

The set of *positions*, or *nodes*, of w is $\mathrm{nodes}(w) = \{1, \ldots, n\}$. The *root* of w is $\mathrm{root}(w) = n$. The *length* of w, denoted by $|w|$, is the number of symbols occurring in it. The label a_i of node i in w is denoted by $\mathrm{lab}^w(i)$.

Definition 1 A possibly nondeterministic *finite automaton* (nFA) over Σ is a tuple $A = (\mathrm{states}(A), \mathrm{alphabet}(A), \mathrm{rules}(A), \mathrm{init}(A), \mathrm{final}(A))$, where $\mathrm{alphabet}(A) = \Sigma$, $\mathrm{rules}(A)$ is a finite set of rules of the form $q_1 \xrightarrow{a} q_2$ with $q_1, q_2 \in \mathrm{states}(A)$ and $a \in \mathrm{alphabet}(A)$, and $\mathrm{init}(A) \subseteq \mathrm{states}(A)$.

A finite automaton uses Σ-strings as its data structure. A *run* of A on a string $w \in \mathrm{alphabet}(A)^*$ is a mapping $r : \mathrm{nodes}(w) \to \mathrm{states}(A)$ such that

(i) there exists $q_0 \in \mathrm{init}(A)$ with $q_0 \xrightarrow{a} r(1)$ in $\mathrm{rules}(A)$ for $\mathrm{lab}^w(1) = a$; and,
(ii) for every $i = 1, \ldots, |w| - 1$, it holds that $r(i) \xrightarrow{a} r(i+1)$ in $\mathrm{rules}(A)$ where $\mathrm{lab}^w(i+1) = a$.

We call an nFA A (left-to-right) *deterministic* if it satisfies the following two conditions, implying that no string permits more than one run by A:

(i) $\mathrm{init}(A)$ is a singleton; and,
(ii) for every $q_1 \in \mathrm{states}(A)$ and $a \in \mathrm{alphabet}(A)$, there exists at most one rule $q_2 \in \mathrm{states}(A)$ such that $q_1 \xrightarrow{a} q_2$ is in $\mathrm{rules}(A)$.

We denote by dFA be the class of deterministic, and by uFA the class of

unambiguous finite automata.

2.3 Unranked and Binary Trees

It is common to view XML documents as finite unranked trees with labels from a finite alphabet Σ.

We define these finite unranked trees formally. A *tree domain* N is a non-empty, prefix-closed subset of ${\mathbb{N}_0}^*$ satisfying the following condition: if $ui \in N$ for $u \in {\mathbb{N}_0}^*$ and $i \in \mathbb{N}_0$, then $uj \in N$ for all j with $1 \leq j \leq i$. An *unranked* Σ-*tree* t (which we simply call tree in the following) is a mapping $t : \text{nodes}(t) \to \Sigma$ where $\text{nodes}(t)$ is a finite tree domain. The elements of $\text{nodes}(t)$ are called the *nodes* of t. For $u \in \text{nodes}(t)$, we call nodes of the form $ui \in \text{nodes}(t)$ with $i \in \mathbb{N}_0$ the *children* of u (where ui is the ith child). For a tree t and a node $u \in \text{nodes}(t)$, we denote the label $t(u)$ by $\text{lab}^t(u)$. If the root of t is labeled by a, that is, $\text{lab}^t(\varepsilon) = a$, and if the root has k children at which the subtrees $t_1, \ldots, t_k$ are rooted from left to right, then we denote this by $t = a(t_1 \cdots t_k)$. The *depth* of a node $i_1 \cdots i_n \in (\mathbb{N}_0)^*$ in a tree is $n + 1$. The *depth* of a tree is the maximum of the depths of its nodes. We denote the set of unranked Σ-trees by $\mathcal{T}_\Sigma$. A *tree language* is a set of trees. In the sequel, we adopt the following convention: when we write a tree as $a(t_1 \cdots t_n)$, we tacitly assume that all t_i's are trees.

A *binary alphabet* or *binary signature* is a pair $(\Sigma, \text{rank}_\Sigma)$, where rank_Σ is a function from Σ to $\{0, 2\}$. The set of *binary* Σ-*trees* is the set of Σ-trees inductively defined as follows. When $\text{rank}_\Sigma(a) = 0$, then a is a binary Σ-tree. When $\text{rank}_\Sigma(a) = 2$ and t_1, t_2 are binary Σ-trees, then $a(t_1\ t_2)$ is a binary Σ-tree.

2.4 Traditional Tree Automata for Binary Trees

In this article, we discuss tree automata over binary as well as unranked trees. For clarity, we refer to the former as "traditional tree automata" and to the latter as "unranked tree automata".

Definition 2 A possibly nondeterministic traditional *tree automaton* (nTA) over Σ is a tuple $A = (\text{states}(A), \text{alphabet}(A), \text{rules}(A), \text{init}(A), \text{final}(A))$ where $\Sigma = \text{alphabet}(A)$ is a binary alphabet, $\text{init}(A) \subseteq \text{states}(A)$ is a set of initial states, and $\text{rules}(A)$ is a set of rules of the form

- $a \to q$ with $\text{rank}_{\text{alphabet}(A)}(a) = 0$ and $q \in \text{states}(A)$; or,
- $a(q_1, q_2) \to q$ with $\text{rank}_{\text{alphabet}(A)}(a) = 2$ and $q_1, q_2, q \in \text{states}(A)$.

A traditional tree automaton uses binary Σ-trees as its data structure. It is *(bottom-up) deterministic* if no two of its rules have the same left-hand sides. A *run* of A on a binary Σ-tree t is a mapping $r : \text{nodes}(t) \to \text{states}(A)$ such that

(i) for every leaf node u with label a, $a \to r(u)$ is in rules(A); and,
(ii) for every inner node u with label a, $a(r(u1), r(u2)) \to r(u)$ is in rules(A).

We denote by dTA the class of deterministic traditional tree automata and by uTA the class of unambiguous traditional tree automata.

2.5 Unranked Tree Automata

We recall the definition of unranked tree automata (UTAs) [3] which dates back to the work of Thatcher [34].

Definition 3 An *unranked tree automaton (UTA)* over Σ is a tuple $A = (\text{states}(A), \text{alphabet}(A), \text{rules}(A), \text{final}(A))$, $\text{alphabet}(A) = \Sigma$, and rules($A$) is a set of rules of the form $a(L) \to q$ such that

(i) $a \in \text{alphabet}(A)$;
(ii) $q \in \text{states}(A)$; and
(iii) L is a regular string language over the alphabet states(A).

For every $a \in \text{alphabet}(A)$ and $q \in \text{states}(A)$, there is at most one L such that $a(L) \to q$ is a rule in A. A UTA is *bottom-up deterministic* if, for all rules $a(L_1) \to q_1$ and $a(L_2) \to q_2$ with $q_1 \neq q_2$, we have that $L_1 \cap L_2 = \emptyset$.

An unranked tree automaton uses $\mathcal{T}_\Sigma$ (that is, unranked Σ-trees) as its data structure. A *run* of A on a tree t is a labeling $r : \text{nodes}(t) \to \text{states}(A)$ such that, for every $v \in \text{nodes}(t)$ with n children, there is a rule $a(L) \to r(v)$ in rules(A), where the label of v is a and $r(v1) \cdots r(vn) \in L$. Notice that, when v has no children, the criterion reduces to $\varepsilon \in L$.

As briefly mentioned in the introduction, we need to specify the representations for the internal (horizontal) string languages of UTAs for the minimization problem. Since our definition of *size* of an unranked tree automata will take the states of the finite automata for the internal string languages into account, the minimization problem for unranked tree automata is at least as hard as for the finite automata for the internal string languages. As a consequence, the minimization problem is immediately NP-hard if we choose nFAs or uFAs for this representation (see Table 1). We therefore refine the definition of unambiguousness and (bottom-up) determinism for UTAs as follows.

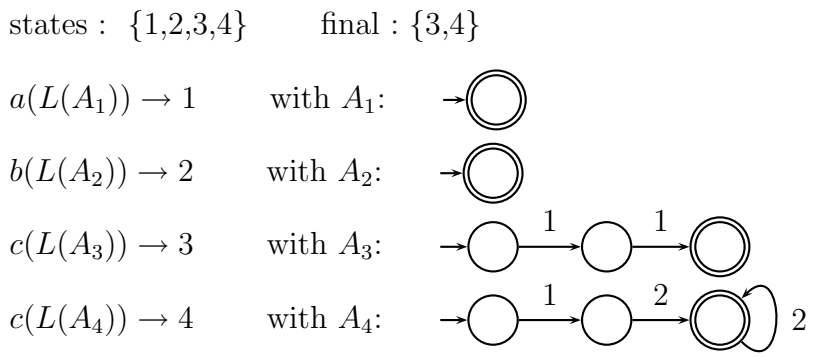


Fig. 1. Example for a dUTA of size 12.

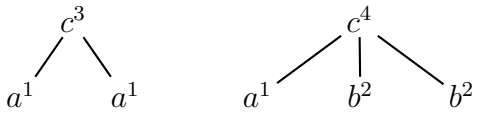


Fig. 2. Two successful runs by the dUTA in Figure 1 annotated to the trees.

Definition 4 A *possibly nondeterministic unranked tree automaton* (nUTA) A is a UTA whose rules have the form $a(L(B)) \to q$ and the string language $L(B)$ is represented by an nFA B with alphabet(B) = states(A). A *(horizontally) unambiguous unranked tree automaton (uUTA)* is an unambiguous nUTA whose finite subautomata are unambiguous, that is, all nFAs are uFAs. A *bottom-up (left-to-right) deterministic unranked tree automaton* (dUTA) is an nUTA that

(i) is bottom-up deterministic as a UTA; and
(ii) whose finite subautomata are all deterministic, that is, all nFAs are dFAs.

The *size* of an nUTA A is defined differently than before, as the states of all nFAs for the horizontal languages are also taken into consideration:

$$|A| \quad = \quad |\text{states}(A)| + \sum_{a(L(B))\to q \in \text{rules}(A)} |B|$$

An example for a dUTA with 12 states is given in Figure 1; it accepts the unranked tree language $\{c(w) \mid w \in L(aa \cup ab^+)\}$. Two of its successful runs are drawn in Figure 2.

Everyone agrees that a dFA is indeed a deterministic device. When during a computation the automaton is in a certain state at a certain node, the next state is always uniquely determined. We raise the question whether a dUTA is a fully deterministic representation of unranked tree languages or not. Clearly, every state computed by a run is uniquely determined due to bottom-up determinism. The internal computation inside of horizontal automata is deterministic too since performed by dFAs. However, choice is needed when one has to decide which rule to apply for a given letter. It requires guessing or testing the possibilities. Intuitively, dUTAs represent the internal regular languages

over states by *a disjoint union of dFAs,* which is in fact an unambiguous representation with one non-deterministic step: the choice of the initial state.

The dUTA in the example in Figure 1 has two rules for the letter c. In the first successful run in Figure 2, we have to chose the upper rule, in the second one the lower rule. It is precisely this limited form of non-determinism that will be exploited in our NP-hardness proof for dUTA minimization.

2.6 Result Overview

In Table 1, we collect complexity results about automata minimization and the related problems.

For finite automata, all presented results are very well known, perhaps maybe with the exception for uFAs, for which EQUIVALENCE, INCLUSION, and UNIVERSALITY are in PTIME, while MINIMIZATION is NP-complete. The NLOGSPACE-completeness of dFA EQUIVALENCE, INCLUSION, and UNIVERSALITY follows from a reduction from and to the well-known reachability problem for graphs. To the best of our knowledge, it is not known whether the PTIME upper bounds for dFA MINIMIZATION and uFA EQUIVALENCE, INCLUSION, and UNIVERSALITY are tight.

For traditional tree automata, the situation is well established too. The PTIME lower bound for UNIVERSALITY of dTAs follows from a straightforward reduction from PATH SYSTEMS, which is known to be PTIME-complete [7]. Notice that the same complexities hold for uFAs and uTAs, even though the proofs for the upper bounds become more involved for uTAs. For the EXPTIME-hardness of nTA MINIMIZATION, note that nTA UNIVERSALITY can be LOGSPACE-reduced to MINIMIZATION, since an automaton A with alphabet$(A) = \Sigma$ is universal if and only if *(i)* MINIMIZATION for A and 1 is true, *(ii)* $a \in L(A)$ for every a with $\text{rank}_\Sigma(a) = 0$, and *(iii)* $b(a\ a) \in L(A)$ for every a, b with $\text{rank}_\Sigma(b) = 2$ and $\text{rank}_\Sigma(a) = 0$.

2.6.1 Some New Results

For nUTAs, the EXPTIME-hardness of EQUIVALENCE, INCLUSION, and UNIVERSALITY are immediate from the binary case, since every nTA can be encoded in PTIME into an nUTA. The EXPTIME upper bound carries over from the case of traditional tree automata for binary trees, based on some binary encoding for unranked trees (see, for example, [12]).

The EXPTIME-hardness of MINIMIZATION follows from a reduction from UNIVERSALITY similarly to the case of traditional tree automata: an nUTA A

	EQUIVALENCE, INCLUSION, UNIVERSALITY	MINIMIZATION
dFA	NLOGSPACE	in PTIME [13] NLOGSPACE-hard (from UNIVERSALITY)
uFA	in PTIME [31] NLOGSPACE-hard	in NP (from EQUIVALENCE) NP-hard [14]
nFA	PSPACE [32]	PSPACE [32]
dTA	in PTIME [6] PTIME-hard [7]	in PTIME [6] PTIME-hard (from UNIVERSALITY)
uTA	PTIME [29]	in NP (from EQUIVALENCE) NP-hard (from uFAs)
nTA	EXPTIME [29]	in EXPTIME (from EQUIVALENCE) EXPTIME-hard (from UNIVERSALITY)
dUTA	in PTIME (Theorem 5) PTIME-hard (from dTAs)	in NP (from EQUIVALENCE) **NP-hard (Theorem 14)**
uUTA	in PTIME (Theorem 5) PTIME-hard (from uTAs)	in NP (from EQUIVALENCE) NP-hard (from uFAs)
nUTA	in EXPTIME (from nTAs) EXPTIME-hard (from nTAs)	in EXPTIME (from EQUIVALENCE) EXPTIME-hard (from UNIVERSALITY)

Table 1
Complexity overview for nondeterministic, unambiguous, and bottom-up and/or left-to-right deterministic automata for strings, binary trees, and unranked trees. For a complexity class C, we write "in C" (or "C-hard") to denote that the mentioned problems are in C (or hard for C), respectively.

with alphabet$(A) = \Sigma$ is universal if and only if *(i)* MINIMIZATION for A and $|\Sigma| + 1$ is true, *(ii)* for every $a \in \Sigma$, $a \in L(A)$, and, *(iii)* for every $a, b \in \Sigma$, $a(b) \in L(A)$.

Upper bounds for the INCLUSION problem for dUTAs and uUTAs can be obtained through an encoding to binary trees, as argued in the following theorem.

Theorem 5 INCLUSION *for dUTAs and uUTAs is in* PTIME. MINIMIZATION *for dUTAs and uUTAs is thus in* NP.

PROOF. Given two uUTAs, we can translate them in PTIME into uTAs with

respect to a binary encoding of unranked trees. For dUTAs and with respect to the standard first-child next-sibling encoding of unranked trees, this has been proposed in Lemma 4.24 of [12]. Another proof through the Curried encoding is presented in the present paper (Propositions 24 and 18). Due to the work of Seidl, we can test inclusion of uTAs in PTIME (Theorem 4.3 in [29]). □

Even though the proposed PTIME algorithm seems overly complicated, it is, to the best of our knowledge, not known whether the "standard" inclusion test of dUTAs works in PTIME. The standard test would, given two dUTAs A and B, test whether $L(A)$ has an empty intersection with the complement of $L(B)$. The difficulty of this approach lies in finding a sufficiently small dUTA for the complement of $L(B)$. This is *not* trivial unless B is *complete*, then one simply has to switch final and non-final states. (A UTA B is *complete* when, for every $w \in \text{states}(B)^*$, there exists a rule $a(L) \to q$ such that $w \in L$.)

There remains one further result in Table 1 that we have not discussed so far. This is the NP-hardness result of dUTA minimization, which is the subject of Section 3. Alternative notions of bottom-up determinism for other kinds of automata on unranked trees will be discussed in Section 4. Models for XML schema languages are studied in Section 5.

3 Minimizing dUTAs

In this section we study the minimization problem of dUTAs. We show two unexpected negative results:

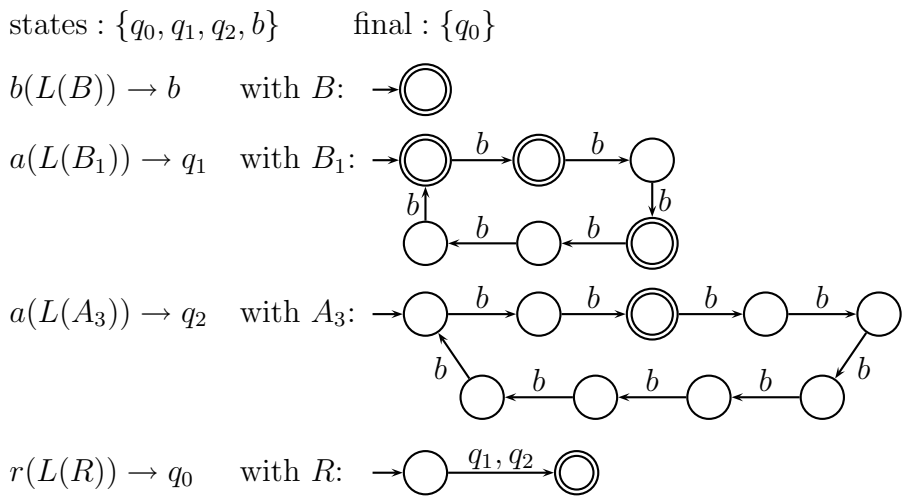
(1) There are regular tree languages for which no unique (up to isomorphism) minimal dUTA exists.
(2) The minimization problem for dUTAs even turns out to be NP-complete.

3.1 Minimal Automata are not Unique

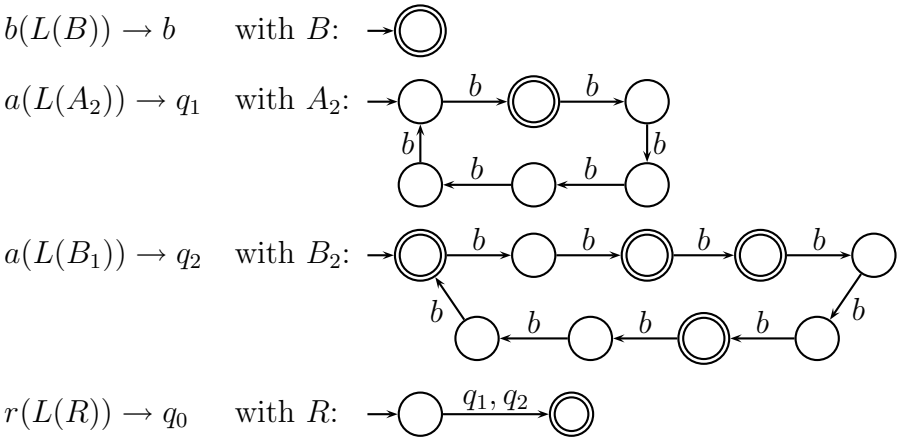
We show the non-uniqueness by means of an example. Consider the regular string languages L_1, L_2, and L_3 defined by the regular expressions

$$(bbb)^*,\ b(bbbbbb)^*, \text{ and } bb(bbbbbbbbb)^*,$$

respectively. Notice 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. The minimal dFAs B_1 and B_2 accepting $L_1 \cup L_2$ and $L_1 \cup L_3$ (which are depicted as parts of Figure 3) have 6 and 9 states, respectively.



(a) The dUTA N_1.



(b) The dUTA N_2.

Fig. 3. Two equivalent minimal dUTAs that are not isomorphic.

Define L to be the language $L_1 \cup L_2 \cup L_3$ and consider the tree language $T = \{r(a(w)) \mid w \in L\}$.

There exist two non-isomorphic minimal dUTAs for T. The first one, N_1, is defined in Figure 3(a). Notice that the size of N_1 is

$$|\text{states}(N_1)| + 1 + |B_1| + |A_3| + 2 = 4 + 1 + 6 + 9 + 2 = 22.$$

The other automaton, N_2, is defined in Figure 3(b). Notice that the size of N_2 is

$$|\text{states}(N_2)| + 1 + |B_2| + |A_2| + 2 = 4 + 1 + 9 + 6 + 2 = 22.$$

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 make use of the following theorem for regular string languages over a one-letter alphabet:

Theorem 6 (e.g., [26]) *A string language L over $\{a\}$ is regular if and only if there are two integers $n_0 \geq 0, k \geq 1$ such that, for any $n \geq n_0$, $a^n \in L$ if and only if $a^{n+k} \in L$. Moreover, when L is regular, the minimal dFA for L contains a cycle with k states.*

We show that no other combination of splitting L into a union of regular languages results in a smaller dUTA accepting T. First, observe that any dUTA N defining T needs at least three states in states(N), since all trees in T have depth three. However, as argued above, the minimal size of such a dUTA with three states is $3 + 1 + 18 + 2 = 24$. The only way to obtain an equivalent dUTA smaller than N_1 and N_2 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 a cycle with 6 states, as D_1 may not accept strings *not* in L (applying Theorem 6 with any $k \leq 6$ would imply that $L(D_1) \not\subseteq L$). 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 $D_2 \neq D_1$, then we can obtain analogously that D_2 has a cycle with 9 states. If $D_1 = D_2$, we obtain analogously that D_1 has at least 18 states. Therefore, the above automata are indeed minimal for T, and as Figure 3 shows, they are clearly not isomorphic as the final states in the internal dFAs are different.

3.2 Minimization is NP*-Complete*

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 refer to this problem as MINIMAL DISJOINT UNION and we define it formally later in this section.

In this section, we show that MINIMAL DISJOINT UNION is NP-complete by a reduction from VERTEX COVER. Actually, MINIMAL DISJOINT UNION is even NP-complete when we are asked to define a regular string language as a small disjoint union of *two* dFAs. The proof for this result is technically the hardest proof in the article. The reduction is technical but interesting in its own right as it shows that minimization is hard for a class of finite string automata with

very little nondeterminism (Lemma 11).

We start by formally defining the decision problems that are of interest to us. Given a graph $G = (V, E)$ such that V is its set of vertices and $E \subseteq V \times V$ is its set of edges, we say that a set of vertices $VC \subseteq V$ is a *vertex cover* of G if, for every edge $(v_1, v_2) \in E$, VC contains v_1, v_2, or both. We can assume without loss of generality that G is an undirected graph which does not contain *self-loops*. That is, G does not contain edges of the form (v_1, v_1), and if $(v_1, v_2) \in E$, then (v_2, v_1) is also in E.

If B and C are finite collections of finite sets, we say that B is a *normal set basis* of C if, for each $c \in C$, there is a pairwise disjoint subcollection B_c of B whose union is c. For $m \in \mathbb{N}_0$, we say that B is a *K-separable normal set basis* of C if B is a normal set basis of C and B can be written as a disjoint union $B_1 \uplus \cdots \uplus B_K$ such that, for each $j = 1, \ldots, K$, the subcollection B_c of B contains at most one element from B_j. The *size* of a collection of finite sets is the sum of the sizes of the finite sets it contains.

We say that a collection C of sets contains *obsolete* symbols if there exist two elements $a \neq b$ such that, for every $c \in C, a \in c \Leftrightarrow b \in c$.

We consider the following decision problems.

VERTEX COVER: Given a pair (G, k) where G is a graph and k is an integer, does there exist a vertex cover of G of size at most k?

NORMAL SET BASIS: Given a pair (C, s) where C is a finite collection of finite sets and s is an integer, does there exist a normal set basis of C containing at most s sets?

K-SEPARABLE NORMAL SET BASIS: Given a pair (C, s) where C is a finite collection of finite sets and s is an integer, does there exist a K-separable normal set basis of C containing at most s sets?

K-MINIMAL DISJOINT UNION: Given a pair (M, ℓ) where M is a dFA and ℓ is an integer, do there exist dFAs $M_1, \ldots, M_K$ such that
(1) $L(M) = L(M_1) \cup \cdots \cup L(M_K)$; and
(2) for every $i \neq j$, $L(M_i) \cap L(M_j) = \emptyset$; and
(3) $\sum_{i=1}^{K} |M_i| \leq \ell$?

We assume that the integers in the input of these decision problems are given in their binary representation. The first two problems are known to be NP-complete [14]. We will show that the last two problems are NP-complete (for $K \geq 2$) as well.

We start by showing that NORMAL SET BASIS and K-SEPARABLE NORMAL SET BASIS are NP-complete for every $K \geq 2$. We revisit a slightly modified reduction which is due to Jiang and Ravikumar [14], as our further results heavily rely on a construction in their proof.

Lemma 7 (Jiang and Ravikumar [14]) NORMAL SET BASIS *is* NP*-complete.*

PROOF. Obviously, NORMAL SET BASIS is in NP. Indeed, given an input (C, s) for NORMAL SET BASIS, the NP algorithm simply guesses a collection B containing at most s sets, guesses the subcollections B_c for each $c \in C$, and verifies whether the sets B_c satisfy the necessary conditions.

We show that NORMAL SET BASIS is NP-hard by a reduction from VERTEX COVER. Given an input (G, k) of VERTEX COVER, where $G = (V, E)$ is a graph and k is an integer, we construct in LOGSPACE an input (C, s) of NORMAL SET BASIS, where C is a finite collection of finite sets and s is an integer. In particular, (C, s) is constructed such that

G has a vertex cover of size at most k if and only if
C has a normal set basis containing at most s sets.

For a technical reason which will become clear later in the article, we assume without loss of generality that $k < |E| - 3$. Notice that, under this restriction, VERTEX COVER is still NP-complete under LOGSPACE reductions. Indeed, if $k \geq |E| - 3$, VERTEX COVER can be solved in LOGSPACE by testing all possibilities of the at most 3 vertices which are not in the vertex cover, and verifying that there does not exist an edge between 2 of these 3 vertices.

Formally, let $V = \{v_1, \ldots, v_n\}$. For each $i = 1, \ldots, n$, define c_i to be the set $\{x_i, y_i\}$ which intuitively corresponds to the node v_i. Let (v_i, v_j) be in E with $i < j$. To each such edge we associate five sets as follows:

$$
\begin{aligned}
c_{ij}^1 &:= \{x_i, a_{ij}, b_{ij}\}, \\
c_{ij}^2 &:= \{y_j, b_{ij}, d_{ij}\}, \\
c_{ij}^3 &:= \{y_i, d_{ij}, e_{ij}\}, \\
c_{ij}^4 &:= \{x_j, e_{ij}, a_{ij}\}, \text{ and} \\
c_{ij}^5 &:= \{a_{ij}, b_{ij}, d_{ij}, e_{ij}\}.
\end{aligned}
$$

Figure 4 contains a graphical representation of the constructed sets c_i, c_j, $c_{ij}^1, \ldots, c_{ij}^5$ for some $(v_i, v_j) \in E$.

Then, define

$$C := \{c_i \mid 1 \leq i \leq n\} \cup \{c_{ij}^t \mid (v_i, v_j) \in E, i < j, \text{ and } 1 \leq t \leq 5\}$$

and

$$s := n + 4|E| + k.$$

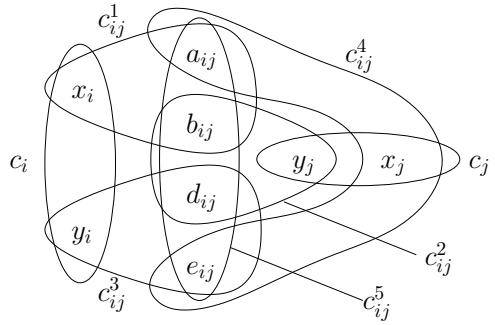


Fig. 4. The constructed sets $c_i, c_j, c^1_{ij}, \ldots, c^5_{ij}$ in the proof of Lemma 7.

Notice that the C contains $n+5|E|$ sets and that C does not contain obsolete symbols. Obviously, C and s can be constructed from G and k in polynomial time.

We show that the given reduction is also correct, that is, that G has a vertex cover of size at most k if and only if C has a normal set basis containing at most s sets.

($\Rightarrow$): Let G have a vertex cover VC of size k. We need to show that C has a normal set basis B containing at most $s = n + 4|E| + k$ sets.

Thereto, we define a collection B of sets as follows. For every $v_i \in V$,

- if $v_i \in VC$, we include both $\{x_i\}$ and $\{y_i\}$ in B;
- otherwise, we include $c_i = \{x_i, y_i\}$ in B.

The number of sets included in B so far is $2k+(n-k) = k+n$. Let $e = (v_i, v_j)$ (where $i < j$) be an arbitrary edge in G. Since VC is a vertex cover, either v_i or v_j (or both) is in VC. When v_i is in VC, we additionally include the sets

$$r^1_{ij} := \{a_{ij}, b_{ij}\}, \qquad r^2_{ij} := \{d_{ij}, e_{ij}\},$$
$$r^3_{ij} := \{y_j, b_{ij}, d_{ij}\}, \text{ and} \quad r^4_{ij} := \{x_j, a_{ij}, e_{ij}\}$$

in B. When v_i is not in VC, we additionally include the sets

$$r^5_{ij} := \{a_{ij}, e_{ij}\}, \qquad r^6_{ij} := \{b_{ij}, d_{ij}\},$$
$$r^7_{ij} := \{x_i, a_{ij}, b_{ij}\}, \text{ and} \quad r^8_{ij} := \{y_i, d_{ij}, e_{ij}\}$$

in B. This completes the definition of B. Notice that, when $v_i \in VC$, c^1_{ij}, c^3_{ij}, and c^5_{ij} can be expressed as a disjoint union of members of B as

$$c^1_{ij} = \{x_i\} \uplus r^1_{ij}, \qquad c^3_{ij} = \{y_i\} \uplus r^2_{ij}, \qquad c^5_{ij} = r^1_{ij} \uplus r^2_{ij}$$

and that $c^2_{ij} = r^3_{ij}$ and $c^4_{ij} = r^4_{ij}$ are members of B. Analogously, when $v_i \notin VC$,

c^2_{ij}, c^4_{ij}, and c^5_{ij} can be expressed as a disjoint union of members of B as

$$c^2_{ij} = \{y_j\} \uplus r^6_{ij}, \qquad c^4_{ij} = \{x_j\} \uplus r^5_{ij}, \qquad c^5_{ij} = r^5_{ij} \uplus r^6_{ij}$$

and $c^1_{ij} = r^7_{ij}$ and $c^3_{ij} = r^8_{ij}$ are members of B. Since the total number of sets included in B for each edge is four, B contains $(k+n)+4|E| = s$ sets. From the foregoing argument it is also obvious that B is a normal set basis of C.

Notice that B is in fact a 2-separable normal set basis for C. Indeed, we can partition B into the sets

$$\begin{aligned} B_1 = \{\{x_i\}, \{x_j, y_j\} \mid v_i \in VC, v_j \notin VC\} \\ \cup \{r^2_{ij}, r^3_{ij} \mid (v_i, v_j) \in E, i < j, v_i \in VC\} \\ \cup \{r^6_{ij}, r^7_{ij} \mid (v_i, v_j) \in E, i < j, v_i \notin VC\} \end{aligned}$$

and

$$\begin{aligned} B_2 = \{\{y_i\} \mid v_i \in VC\} \\ \cup \{r^1_{ij}, r^4_{ij} \mid (v_i, v_j) \in E, i < j, v_i \in VC\} \\ \cup \{r^5_{ij}, r^8_{ij} \mid (v_i, v_j) \in E, i < j, v_i \notin VC\}, \end{aligned}$$

which satisfy the necessary condition.

($\Leftarrow$): Suppose that C has a normal set basis B containing at most $s = n + 4|E| + k$ sets. We can assume without loss of generality that no proper subcollection of B is a normal set basis. We show that G has a vertex cover VC of size at most k. Define $VC = \{v_i \mid \text{both } \{x_i\} \text{ and } \{y_i\} \text{ are in } B\}$. Let k' be the number of elements in VC. The number of sets in B consisting of only x_i and/or y_i is at least $n + k'$. This can be seen from the fact that B must have the subset c_i for all i such that $v_i \notin VC$. Thus, there are $n - k'$ such sets in addition to $2k'$ singleton sets corresponding to i's such that $v_i \in VC$. Let $E' \subseteq E$ be the set of edges covered by VC, that is, $E' = \{(v_i, v_j) \mid v_i \text{ or } v_j \text{ is in } VC\}$. The following observation can easily be shown (by checking all possibilities):

Observation: For any $e \in E'$ at least four sets of B (excluding sets $c_i, c_j, \{x_i\}$, $\{y_i\}, \{x_j\},$, or $\{x_j\}$) are necessary to be a normal set basis for the five sets c^t_{ij}, $t = 1, \ldots, 5$. Further, at least five sets (excluding sets $c_i, c_j, \{x_i\}, \{y_i\}, \{x_j\},$, or $\{x_j\}$) are required to be a normal set basis for them if $e \notin E'$. Notice that, for $e \notin E'$, $\{x_i\}$ and $\{y_i\}$, or $\{x_j\}$ and $\{y_j\}$ are never both in B, by definition of E'.

Now the total number of sets needed to cover C is at least $n + k' + 4|E'| + 5(|E| - |E'|)$, which we know is at most $s = n + 4|E| + k$. Hence, we obtain that $n + k' + 5|E| - |E'| \leq n + 4|E| + k$, which implies that $k' + |E| - |E'| \leq k$. We conclude the proof by showing that there is a vertex cover VC' of size

$|E| - |E'| + k'$. Add one of the end vertices of each edge $e \in E - E'$ to VC. This vertex cover is of size $|E| - |E'| + k' \leq k$. □

The next proposition now follows from the proof of Lemma 7. The intuition behind Proposition 8 is that C has a normal set basis containing s sets if and only if C has a 2-separable normal set basis containing s sets for any input (C, s) in $\mathbf{I}$. Of course, the latter property does not hold for the set of all possible inputs for the normal set basis problem.

Proposition 8 *There exists a set of inputs $\boldsymbol{I}$ for* NORMAL SET BASIS*, such that*

(1) NORMAL SET BASIS *is* NP*-complete for inputs in $\boldsymbol{I}$; and,*
(2) for each $(C, s) \in \boldsymbol{I}$, the following are equivalent:
  (a) C has a normal set basis containing s sets.
  (b) there exists a $K \geq 2$ such that C has a K-separable normal set basis containing s sets.
  (c) for every $K \geq 2$, C has a K-separable normal set basis containing s sets.

Moreover, for each (C, s) in $\boldsymbol{I}$, C contains every set at most once, C does not contain obsolete symbols, and $s < |C| - 3$.

PROOF. The set $\mathbf{I}$ is obtained by applying the reduction in Lemma 7 to inputs (G, k) of VERTEX COVER. In the proof of Lemma 7 we showed that, if G has a vertex cover of size k, then C has a 2-separable normal set basis containing s sets, which implies (a), (b), and (c). Conversely, if (a), (b), or (c) holds, meaning that C has a normal set basis containing s sets (which is allowed to be K-separable for any $K \geq 2$), we have shown that G has a vertex cover of size k.

Moreover, we observed that C was constructed such that it does not contain obsolete symbols. For the size constraint, we have to recall the assumption in Lemma 7, that $k < |E| - 3$. Hence, we obtain that $s = n + 4|E| + k < n + 5|E| - 3 = |C| - 3$. □

Since the proof of Lemma 7 shows that NORMAL SET BASIS is an NP-complete problem for inputs in $\mathbf{I}$, we immediately obtain the following:

Corollary 9 *For every $K \geq 2$,* K-SEPARABLE NORMAL SET BASIS *is* NP*-complete.*

Our next goal is to show a result for MINIMAL DISJOINT UNION which is similar to Proposition 8. However, in order to apply the result immediately to MINIMIZATION for dUTAs later, we need to treat a minor technical issue. (Readers who are only interested in the NP-hardness of K-MINIMAL DISJOINT UNION can safely skip the following definition.) Due to the fact that the internal dFAs of dUTAs do not read alphabet symbols, but states of the tree automaton, we need to take extra care of the languages we define in the reduction for the MINIMAL DISJOINT UNION problem: we will require that the languages do not contain *interchangeable symbols*, a property which we define as follows.

Definition 10 Given a string language L over an alphabet Σ, we say that two symbols $a, b \in \Sigma$, $a \neq b$, are *interchangeable with respect to* L if, for every two Σ-strings u and v, we have that $uav \in L \Leftrightarrow ubv \in L$. We say that L *contains interchangeable symbols* if there exist $a, b \in \Sigma$, $a \neq b$, which are interchangeable with respect to L.

We are now ready to show the following lemma.

Lemma 11 *For every $K \geq 2$, K-*MINIMAL DISJOINT UNION *is* NP*-complete.*

PROOF. The NP upper bound follows from the fact that we can guess a disjoint union of sufficiently small size and verify in PTIME that it is equivalent (see also Section 2.6, where we recall that testing equivalence of unambiguous string automata is in PTIME).

For the lower bound, we reduce from 2-SEPARABLE NORMAL SET BASIS. To this end, let (C, s) be an input of 2-SEPARABLE NORMAL SET BASIS. Hence, C is a collection of n sets and s is an integer. According to Proposition 8, we can assume without loss of generality that $(C, s) \in \mathbf{I}$, that is, C has a 2-separable normal set basis containing s sets if and only if there exists a $K \geq 2$ for which C has a K-separable normal set basis containing s sets. Moreover, we can assume that $s < n - 3$.

We construct in LOGSPACE an input (M, ℓ) of MINIMAL DISJOINT UNION such that

$$C \text{ has a 2-separable normal set basis containing at most } s \text{ sets}$$
$$\text{if and only if}$$
$$\text{there exists a } K \geq 2 \text{ such that } K\text{-MINIMAL DISJOINT UNION is true for } (M, \ell)$$
$$\text{if and only if,}$$
$$\text{for every } K \geq 2, K\text{-MINIMAL DISJOINT UNION is true for } (M, \ell).$$

Intuitively, M accepts the language $\{ca \mid c \in C \text{ and } a \in c\}$, which is a finite language of strings of length two.

We state the following claim, which is needed later in the article but is not important for the proof of the present lemma. We prove the claim after the proof of the present lemma.

Claim 12 *$L(M)$ does not contain interchangeable symbols.*

Formally, let $C = \{c_1, \ldots, c_n\}$ and $c_i = \{a_{i,1}, \ldots, a_{i,n_i}\}$. Then, M is defined over

$$\text{alphabet}(M) = \bigcup_{1 \leq i \leq n} \{c_i, a_{i,1}, \ldots, a_{i,n_i}\}.$$

The state set of M is $\text{states}(M) = \{q_0, q_1, \ldots, q_n, q_f\}$, and the initial and final state sets of M are $\{q_0\}$ and $\{q_f\}$, respectively. The transitions $\text{rules}(M)$ are depicted in Figure 5 and are formally defined as follows:

- for every $i = 1, \ldots, n$, $q_0 \xrightarrow{c_i} q_i$; and
- for every $i = 1, \ldots, n$ and $j = 1, \ldots, n_i$, $q_i \xrightarrow{a_{i,j}} q_f$.

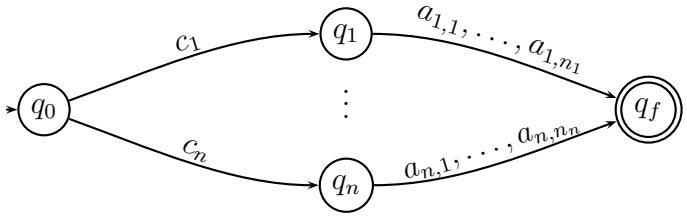


Fig. 5. Illustration of a fragment of the constructed automaton M in the proof of Lemma 11.

Finally, define

$$\ell := s + 4.$$

Obviously, M and ℓ can be constructed from C and s using logarithmic space. Observe that, due to Proposition 8, C contains every set at most once, and hence does not contain $c_i = c_j$ with $i \neq j$. Hence, M is a minimal dFA for $L(M)$.

We now show that,

(a) if C has a 2-separable normal set basis containing at most s sets, then 2-MINIMAL DISJOINT UNION is true for (M, ℓ); and
(b) if there exists a $K \geq 2$ for which K-MINIMAL DISJOINT UNION is true for (M, ℓ), then C has a 2-separable normal set basis containing at most s sets.

This proves the lemma, since a disjoint union of two dFAs can also be seen as a disjoint union of K dFAs where $K - 2$ dFAs have an empty state set.

(a) Assume that C has a 2-separable normal set basis containing s sets. We need to show that there exist two dFAs M_1 and M_2 such that

(1) $L(M) = L(M_1) \cup L(M_2)$; and
(2) $L(M_1) \cap L(M_2) = \emptyset$; and
(3) $|M_1| + |M_2| \leq \ell$,

where $\ell = s + 4$.

Thereto, let $B = \{r_1, \ldots, r_s\}$ be the 2-separable normal set basis of C containing s sets. Also, let B_1 and B_2 be disjoint subcollections of B such that each element of C is either an element of B_1, an element of B_2, or a disjoint union of an element of B_1 and an element of B_2.

To describe M_1 and M_2, we first fix the representation of each set c in C as a disjoint union of at most one set in B_1 and at most one set in B_2. Say that each basic member of B in this representation *belongs to* c.

We define the state sets of M_1 and M_2 as

$$\text{states}(M_1) = \{q_0^1, q_f^1\} \cup \{r_i \in B_1\}$$

and

$$\text{states}(M_2) = \{q_0^2, q_f^2\} \cup \{r_i \in B_2\},$$

respectively. The transition rules of M_1 and M_2 are defined as follows. For every $i = 1, \ldots, n$, $j = 1, \ldots, s$, and $x = 1, 2$, rules(M_x) contains the rules

- $q_0^x \xrightarrow{c_i} r_j$, if $r_j \in B_x$ and r_j belongs to c_i; and
- $r_j \xrightarrow{a} q_f^x$, if $r_j \in B_x$ and $a \in r_j$.

Notice that the sum of the sizes of M_1 and M_2 is $|B| + 4 = s + 4 = \ell$, which fulfills condition (3). By construction, we have that $L(M_1) \cup L(M_2) = L(M)$, which fulfills condition (1).

We argue that M_1 is deterministic (M_2 follows by symmetry). By construction, M_1 has only one start state, and all transitions going to its final state are deterministic. Hence, it remains to show that the transitions of the form $q_0^1 \xrightarrow{c_i} r_j$, are deterministic. Towards a contradiction, assume that M_1 contains transitions of the form $q_0^x \xrightarrow{c_i} r_j$ and $q_0^x \xrightarrow{c_i} r_{j'}$ with $j \neq j'$. But this means that both r_j and $r_{j'}$ belong to c_i, which contradicts the definition of B_1.

We still have to show that $L(M_1) \cap L(M_2)$ is empty. Towards a contradiction, assume that the string $c_i a$ is in $L(M_1) \cap L(M_2)$. Let r_j (respectively, $r_{j'}$) be the state that M_1, (respectively, M_2) reaches after reading c_i. By construction of M_1 and M_2, we have that $j \neq j'$. But this means that both r_j and $r_{j'}$ belong to c_i, and their intersection contains the element a, which contradicts the disjointness condition of the is a normal set basis B.

(b) Assume that $L(M)$ can be accepted by a disjoint union of the dFAs

$M_1, \ldots, M_K$ such that the sum of the sizes of $M_1, \ldots, M_K$ is at most ℓ, and for every $i = 1, \ldots, K$, $L(M_i) \neq \emptyset$. We can assume that every M_i is minimal. We need to show that there exists a 2-separable normal set basis for C containing at most $s = \ell - 4$ sets.

Recall that we assumed that $s < n - 3$. Hence, we have that $\ell = s + 4 < n + 1 = |M| - 1$. As we observed that M is a minimal dFA for $L(M)$, it must be the case that $K \geq 2$.

Let, for every $i = 1, \ldots, K$, q_0^i and q_f^i be the initial and final state of M_i, respectively. Since $M_1, \ldots, M_K$ accept a finite set of strings of length 2, we can divide the union of the state sets of $M_1, \ldots, M_K$ into three sets $\mathcal{Q}_0$, $\mathcal{Q}_1$, and $\mathcal{Q}_2$ such that the only transitions in M_i are from $\mathcal{Q}_0$ to states in $\mathcal{Q}_1$ and from states in $\mathcal{Q}_1$ to states in $\mathcal{Q}_2$. For each state $q \in \mathcal{Q}_1$, define a set $B_q = \{a \mid q \xrightarrow{a} q_f^i \in \text{rules}(M_i), 1 \leq i \leq K\}$.

As $K \geq 2$, we have that $B = \{B_q \mid q \in \mathcal{Q}_1\}$ contains at most $\ell - 4$ sets. We show that the collection B is also a normal set basis of C.

By definition of $L(M)$, we have that every $c \in C$ is the union of $B_c := \{B_q \mid q_0^i \xrightarrow{c} q \in \text{rules}(M_i)\}$. It remains to show that B_c is also a disjoint subcollection of B. When B_c contains only one set, there is nothing to prove. Towards a contradiction, assume that B_c contains two different sets B_{q_1} and B_{q_2} such that $a \in B_{q_1} \cap B_{q_2}$. As every M_i is deterministic, we have that $q_1 \in \text{states}(M_{i_1})$ and $q_2 \in \text{states}(M_{i_2})$ with $i_1 \neq i_2$. But this means that $ca \in L(M_{i_1}) \cap L(M_{i_2})$, which contradicts that $M_1, \ldots, M_K$ is a disjoint union.

Hence, B is a normal set basis of C. As $(C, s) \in \mathbf{I}$, we have that C has a 2-separable normal set basis of size $s = \ell - 4$ by Proposition 8. □

It remains to prove Claim 12.

Proof of Claim 12: *$L(M)$ does not contain interchangeable symbols.*

PROOF. Recall that M accepts a language $\{ca \mid c \in C \text{ and } a \in c\}$ of strings of length 2, for a collection of sets C. We denote by E the set $\{a \mid c \in C \text{ and } a \in c\}$ of elements of sets in C.

By definition of $L(M)$, we have that the alphabet C that we use for the letters of the first position is disjoint from the alphabet E that we use for the letters of the second position. Hence, symbols from C are never interchangeable with symbols from E.

We prove the remaining cases by contraposition:

- Suppose that c_1 and c_2 are different elements from C and that c_1 and c_2 are interchangeable. By definition of $L(M)$, this means that c_1 and c_2 contain precisely the same elements, which contradicts that they are different elements from C.
- Suppose that a_1 and a_2 are different elements from E and that a_1 and a_2 are interchangeable. By definition of $L(M)$ this means that a_1 is contained in precisely the same sets as a_2. But this means that C contains obsolete symbols, which contradicts that we chose (C, s) in a set $\mathbf{I}$ satisfying the conditions in Proposition 8. □

The following proposition is the counterpart of Proposition 8 for the MINIMAL DISJOINT UNION problem.

Proposition 13 *There exists a set of inputs $\boldsymbol{J}$ for* MINIMAL DISJOINT UNION*, such that*

*(1) for each $K \geq 2$, K-*MINIMAL DISJOINT UNION *is* NP*-complete for inputs in $\boldsymbol{J}$; and,*
(2) for each $(M, \ell) \in \boldsymbol{J}$, the following are equivalent:
  *(a) there exists a $K \geq 2$ such that (M, ℓ) has a solution for K-*MINIMAL DISJOINT UNION*;*
  *(b) for every $K \geq 2$, (M, ℓ) has a solution for K-*MINIMAL DISJOINT UNION*.*

Moreover, $L(M)$ does not contain interchangeable symbols and $\ell < |M| - 1$.

PROOF. The set $\mathbf{J}$ is obtained by applying the reduction in Lemma 11 to inputs $\mathbf{I}$ of NORMAL SET BASIS in Proposition 8. Let (M, ℓ) in $\mathbf{J}$ be obtained by applying the reduction in Lemma 11 to some (C, s) in $\mathbf{I}$. In the proof of Lemma 11 we showed that,

(a) if C has a 2-separable normal set basis containing at most s sets, then 2-MINIMAL DISJOINT UNION is true for (M, ℓ); and
(b) if there exists a $K \geq 2$ for which K-MINIMAL DISJOINT UNION is true for (M, ℓ), then C has a 2-separable normal set basis containing at most s sets.

Since a 2-MINIMAL DISJOINT UNION is also a K-MINIMAL DISJOINT UNION for every $K > 2$, in which $K - 2$ dFAs have an empty state set, the equivalence between (2)(a) and (2)(b) immediately follows.

Since 2-SEPARABLE NORMAL SET BASIS is NP-complete for inputs in $\mathbf{I}$, 2-MINIMAL DISJOINT UNION is NP-complete for inputs in $\mathbf{J}$. Due to the equivalence of (2)(a) and (2)(b), we also have that (1) holds.

It is shown in Claim 12 that $L(M)$ does not contain interchangeable symbols. The size constraint is obtained by observing that, in the proof of Lemma 11, we assumed that $s < n - 3$, which implied that $\ell < |M| - 1$. □

We are now ready to prove the main result of the present section.

Theorem 14 MINIMIZATION *for dUTAs is* NP*-complete.*

PROOF. The upper bound follows from Theorem 5. Given a dUTA A and an integer m, the NP algorithm guesses an automaton B of size at most m and verifies in PTIME whether it is equivalent to A.

For the lower bound, we reduce from 2-MINIMAL DISJOINT UNION. Given a dFA M and integer ℓ, we construct a dUTA A and an integer m such that A has an equivalent dUTA of size m if and only if M can be written as a disjoint union of two dFAs for which the sum of their sizes 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 $r \notin \text{alphabet}(M)$ and the string w is in $L(M)$.

According to Proposition 13, we can assume without loss of generality that $(M, \ell) \in \mathbf{J}$, which implies that $\ell < |M| - 1$ and that $L(M)$ does not contain interchangeable symbols.

We define A formally as follows. The set $\text{alphabet}(A)$ is $\{r\} \uplus \text{alphabet}(M)$. We define $\text{states}(A)$ as $\{q_r\} \uplus \text{alphabet}(M)$, and $\text{final}(A) = \{q_r\}$. For every $a \in \text{alphabet}(M)$, we include the rule $a(\{\varepsilon\}) \to a$. We also include the rule $r(L(M)) \to q_r$. Finally, let $m = 2 + 2|\text{alphabet}(M)| + \ell$. Obviously, A and m can be constructed from (M, ℓ) using logarithmic space. We now show that

K-MINIMAL DISJOINT UNION is true for (M, ℓ) for any $K \geq 2$
if and only if $L(A)$ can be accepted by a dUTA of size m.

($\Rightarrow$) Suppose that K-MINIMAL DISJOINT UNION is true for (M, ℓ) for any $K \geq 2$. According to Proposition 13, there exist dFAs M_1 and M_2 such that

(1) $L(M) = L(M_1) \cup L(M_2)$; and
(2) $L(M_1) \cap L(M_2) = \emptyset$; and
(3) $|M_1| + |M_2| \leq \ell$.

We construct a dUTA B as follows: $\text{states}(B)$ consists of $\text{alphabet}(M) \uplus \{r_1, r_2\}$ and $\text{final}(B) = \{r_1, r_2\}$. The transition rules of B are defined to be

- $r(L(M_1)) \to r_1$;

- $r(L(M_2)) \to r_2$; and
- $a(\{\varepsilon\}) \to a$ for every $a \in \text{alphabet}(M)$.

Obviously, $L(B) = L(A)$. The size of B is

$$\begin{aligned}|B| &= |M_1| + |M_2| + |\text{states}(B)| + \sum_{a(\{\varepsilon\})\to a \in \text{rules}(B)} 1 \\ &= \ell + |\text{alphabet}(M)| + 2 + |\text{alphabet}(M)| \\ &= 2 + 2|\text{alphabet}(M)| + \ell \\ &= m\end{aligned}$$

($\Leftarrow$) Suppose that there exists a dUTA B for $L(A)$ of size at most $m = 2 + 2|\text{alphabet}(M)| + \ell$. We state the following claims (which we prove later):

Claim 15 *B has at least $|alphabet(M)|$ non-accepting states.*

As B is bottom-up deterministic and only accepts trees of depth two, Claim 15 induces a bijection ϕ between states of B and alphabet(M)-symbols: for every state $q \in Q_B$, $\phi(q)$ is the unique symbol $a \in \text{alphabet}(M)$ such that $a(\{\varepsilon\}) \to q$ is a rule in rules(B). We also denote by ϕ the homomorphic bijective extension of ϕ to string languages.

Claim 16 *B has at least two final states.*

Let $r_1, \ldots, r_x$ be the accepting states of B, where $x > 1$. Let, for every $i = 1, \ldots, x$, M_i' be the minimal dFA such that $r(L(M_i')) \to r_i$ is in rules(B). It is easy to see that, from each M_i', a dFA M_i'' can be constructed which is of the same size and accepts $\phi(L(M_i'))$. Moreover, since B is bottom-up deterministic, the languages $L(M_i')$ are pairwise disjoint. As ϕ is bijective, the languages $\phi(L(M_i'))$ are also pairwise disjoint. The total size of $\sum_{i=1}^{x} |M_i''|$ is $m - 2|\text{alphabet}(M)| - x \leq \ell$. Hence, 2-MINIMAL DISJOINT UNION for (M, ℓ) is true. According to Proposition 13, we also have that K-MINIMAL DISJOINT UNION is true for (M, ℓ) for every $K \geq 2$. □

It remains to prove Claims 15 and 16.

Proof of Claim 15: *B has at least $|alphabet(M)|$ non-accepting states*

PROOF. First observe that $L(B)$ contains only trees of depth two. We say that B *assigns* a state $q \in \text{states}(B)$ to a label $a \in \text{alphabet}(M)$ if $a(\{\varepsilon\}) \to q$ is a rule in rules(B).

We first argue that B assigns only non-accepting states to labels in alphabet(M). Indeed, should B assign an accepting state to some $a \in$ alphabet(M), then the tree a, which has depth one, should be in $L(B)$, which is a contradiction.

We now show that B needs at least |alphabet(M)| *different* non-accepting states to assign to the leaves. Towards a contradiction, suppose that B uses lesser than |alphabet(M)| non-accepting states. As B is bottom-up deterministic, there exist two alphabet symbols a and b to which B assigns the same state q in every successful run of B. However, by definition of $L(B)$ this means that, for every two alphabet(M)-strings u and v, $uav \in L(M) \Leftrightarrow ubv \in L(M)$. This contradicts that $L(M)$ does not contain interchangeable symbols, which was shown in Proposition 13. □

Proof of Claim 16: *B has at least two final states.*

PROOF. We recall that $|A| = 1 + 2|\text{alphabet}(M)| + |M|$ and $|B| \leq 2 + 2|\text{alphabet}(M)| + \ell$. Since we chose $(M, \ell) \in \mathbf{J}$, and hence, $\ell < |M| - 1$, we have $|B| < |A|$. Towards a contradiction, suppose that B has only one accepting state q_f. Then B has exactly one transition rule of the form $r(L(M')) \to q_f$, where M' is a dFA accepting $\phi^{-1}(L(M))$. However, as M' accepts a language isomorphic to $L(M)$, and M is a minimal automaton, the size of M' is at least $|M|$. But this means that the size of B is at least $1 + 2|\text{alphabet}(M)| + |M|$, which is a contradiction. □

4 Solutions for Efficient Minimization

As we have shown, UTA minimization is NP-complete even when the internal regular string 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. They contribute different notions of automata and bottom-up determinism for unranked trees, which lead to unique minimal automata and polynomial time minimization. However, as we will see in this section, they do not lead to minimal automata of the same size.

First, *stepwise tree automata* [5] are an algebraic notion of automata for unranked trees which also correspond to automata over binary trees by means

of a binary encoding. Second, *parallel UTAs (PUTAs)* alter the rule format of UTAs and have been independently proposed in [9] and [27]. Third, one can use tree automata that operate on the standard *first-child next-sibling encoding* of unranked into binary trees (see, for example, [10]).

4.1 Stepwise Tree Automata

Stepwise tree automata have been introduced as an algebraic notion of automata for unranked trees [5]. In this section, we show that regular unranked tree languages are recognized by unique minimal deterministic stepwise tree automata, and formulate the corresponding Myhill-Nerode property for unranked tree languages.

From a UTA point of view, the main difference between UTAs and stepwise tree automata is that stepwise tree automata no longer use different state sets for the internal nFAs and for assigning to the nodes of a tree in its run: all these sets are merged into one set.

Definition 17 A possibly nondeterministic *stepwise tree automaton (nSTA)* over Σ is a tuple $A = (\text{states}(A), \text{alphabet}(A), \text{rules}(A), (\text{init}_a(A))_{a \in \text{alphabet}(A)}, \text{final}(A))$, where

- $\text{alphabet}(A) = \Sigma$; and,
- for every $a \in \text{alphabet}(A)$, $(\text{states}(A), \text{states}(A), \text{rules}(A), \text{init}_a(A), \text{final}(A))$ is a finite automaton accepting strings over $\text{states}(A)$.

We denote the latter finite automaton by $A[\text{init}_a]$.

A run of a stepwise tree automaton A on an unranked tree t is a function $r : \text{nodes}(t) \to \text{states}(A)$ such that, for every node $\nu \in \text{nodes}(t)$ with n children $\nu 1, \ldots, \nu n$, it holds that

$$r(\nu) \in \text{eval}_{A[\text{init}_{\text{lab}^t(\nu)}]}(\ r(\nu 1)\ \cdots\ r(\nu n)\ ).$$

That is, the state of a node ν is computed by running $A[\text{init}_{\text{lab}^t(\nu)}]$, that is, the nFA with initial states determined by the label of ν, on the sequence of states assigned to ν's children.

A *(bottom-up) deterministic stepwise tree automaton (dSTA)* A is a stepwise tree automaton for which every finite automaton $A[\text{init}_a]$ is a dFA. A uSTA is an unambiguous stepwise tree automaton for which every finite automaton $A[\text{init}_a]$ is a uFA. An example of a dSTA is given in Figure 6.

In the present perspective on stepwise tree automata it is not very clear that nSTAs can be determinized without altering the language of unranked trees

Fig. 6. An stepwise tree automaton over $\{a, b\}$ recognizing $\{a(w) \mid w \in L(ab^*)\}$ and one of its successful runs. Initial states for a are pointed to by arrows labeled by a. The state 3 of the root is obtained by running the automaton with initial states for a on the string 122.

they recognize, and that every regular language of unranked trees is recognized by a unique minimal deterministic stepwise tree automaton (up to isomorphism).

Thereto, we observe in the following section that stepwise tree automata are in fact traditional tree automata over a binary encoding of unranked trees. In order to differentiate between the unranked tree language and the binary tree language a stepwise automaton defines, we write $L^u(A)$ for the language of unranked trees recognized by A.

4.1.1 Curried Binary Encoding

We can identify stepwise tree automata with traditional tree automata that operate on Curried binary encodings of unranked trees. While Definition 17 provides the clearest way to present STAs in examples, the present characterization is often more convenient in proofs. It allows to carry over results directly from the theory of traditional tree automata.

We consider the binary alphabet $\Sigma_@ = \Sigma \uplus \{@\}$ in which all labels in Σ have rank zero and @ has rank two. The idea of the Curried encoding is to identify an unranked tree with a lambda term. The tree $a(b\ c\ d)$, for instance, designates the application of function a to the arguments b, c, d. Its Curried encoding $(((a@b)@c)@d)$ applies function a to the same arguments but stepwise one-by-one. Formally, we define the Curried encoding $\mathrm{curry}(t)$ of an unranked tree t as follows:

(i) $\mathrm{curry}(a) = a$;
(ii) $\mathrm{curry}(a(t_1 \cdots t_n)) = @(\mathrm{curry}(a(t_1 \cdots t_{n-1}))\ \mathrm{curry}(t_n))$

An STA A over Σ can be identified with a traditional tree automaton for binary trees over $\Sigma_@$, whose states are those of A. Hence, A has the same size when viewed as an STA or as a traditional tree automaton. We identify the

states : $\{1, 2, 3\}$

rules : $\{a \to 1, \quad @(1,1) \to 3,$

$b \to 2, \quad @(3,2) \to 3\ \}$

final : $\{3\}$

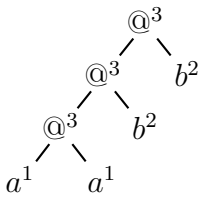


Fig. 7. The STA of Figure 6 with unranked language $\{a(w) \mid w \in L(ab^*)\}$ as standard tree automaton on Curried encodings.

rules as follows:

$$q_1 \xrightarrow{q_2} q \ \text{ is identified with } \ @(q_1, q_2) \to q$$

$$q \in \text{init}_a(A) \ \text{ is identified with } \ a \to q$$

The binary tree language $L^b(A)$ of a stepwise tree automaton A over Σ is the language recognized by the corresponding traditional tree automaton for binary trees over $\Sigma_@$.

The following proposition shows the connection between runs of an STA on unranked trees and their binary encodings.

Proposition 18 *For every STA A, curry$(L^u(A)) = L^b(A)$. Furthermore, A is a dSTA (respectively, uSTA) if and only if it is a dTA (respectively, uTA) as a traditional tree automaton on binary trees.*

PROOF. Let eval^u_A and eval^b_A be the evaluators defined by A on unranked and binary trees respectively, and $\text{eval}_{A[\text{init}_a]}$ the evaluators on strings of states. We show that $\text{eval}^u_A(t) = \text{eval}^b_A(\text{curry}(t))$ for all unranked trees t over Σ. The proof is by induction on the structure of unranked trees.

For the base case, let $t = a$. Then we have $q \in \text{eval}^u_A(t)$ if and only if $q \in \text{eval}_{A[\text{init}_a]}(\text{root}(t))$, if and only if $q \in \text{init}_a(A)$, if and only if $a \to q \in \text{rules}(A)$, if and only if $q \in \text{eval}^b_A(t) = \text{eval}^b_A(\text{curry}(t))$. For the inductive case, we assume $t = a(t_1 \cdots t_n)$. It then holds that $q \in \text{eval}^u_A(t)$ if and only if $q \in \text{eval}_{A[\text{init}_a]}(\text{eval}^u_A(t_1) \cdots \text{eval}^u_A(t_n))$. By induction, this is equivalent to $q \in \text{eval}_{A[\text{init}_a]}(\text{eval}^b_A(\text{curry}(t_1)) \cdots \text{eval}^b_A(\text{curry}(t_n)))$, which holds if and only if $q \in \text{eval}^b_A(@(\cdots @(a\ \text{curry}(t_1)) \cdots)\ \text{curry}(t_n))$ given the correspondence of the automaton rules. By definition of the Curried encoding the latter is equivalent to $q \in \text{eval}^b_A(\text{curry}(a(t_1 \cdots t_n)))$. □

As a consequence, we can determinize every stepwise tree automaton seen as a traditional tree automaton, without changing its language of unranked trees.

Theorem 19 MINIMIZATION *for dSTAs is in* PTIME. *Moreover, every regular unranked tree language is recognized by an up to isomorphism unique minimal dSTA (up to isomorphism).*

PROOF. It is well-known that every regular unranked tree language can be recognized by a stepwise tree automaton. A proof will follow by conversion of nUTAs to *parallel UTAs* (see Section 4.2) to STAs (Proposition 24). Stepwise tree automata can be determinized as traditional tree automata without changing the unranked tree language. The minimal dSTA for a language of unranked trees T is the minimal deterministic tree automaton for the binary tree language curry(T). This follows from Proposition 18. It can be computed by the usual algorithm for minimizing traditional tree automata. □

4.1.2 Myhill-Nerode Property

Myhill and Nerode characterized regular languages in terms of congruences induced by the language, proved the existence of minimal deterministic automata for regular languages, and characterized such automata in terms of the congruence.

The Myhill-Nerode property holds generally for algebraic notions of automata (see, for example, [8]) and thus for finite automata over strings, traditional tree automata [15,35], and stepwise tree automata [5]. A Myhill-Nerode inspired theorem for UTAs was shown in Theorem G in [3]. Remarkably, this theorem does not lead to minimal automata. Another Myhill-Nerode inspired theorem for tree automata for unranked trees was shown by Thomas et al. [9], which we treat in Section 4.2.

In this section, we formulate the Myhill-Nerode theorem for stepwise tree automata on unranked trees, by translating the Myhill-Nerode theorem for traditional tree automata for binary trees via Currying. Our main motivation for discussing the Myhill-Nerode theorem is that the present version has the advantages of the two other Myhill-Nerode inspired theorems, while not sharing their disadvantages: *(i)* it leads to unique minimal deterministic automata, which can be computed in PTIME, *(ii)* it uses a single, natural congruence relation, and *(iii)* it allows to carry over the minimization algorithm directly from traditional tree automata. Moreover, we show later that it leads to the smallest minimal deterministic automata, when compared to the *parallel UTAs* of [9,27] and to traditional tree automata over the standard first-child next-sibling encoding (Sections 4.2.1 and 4.3.2).

A *binary context* C is a function mapping binary trees to binary trees. A context can be represented by a *pointed binary tree*, that is, a binary tree over the signature $\Sigma \uplus \{\bullet\}$ that contains a single occurrence of the symbol "$\bullet$"

which we call the *hole marker*. The hole marker is always at a leaf. *Context application* $C[t]$ of context C to a binary tree t replaces the hole marker in C by t.

An *unranked context* C is a tree over the unranked signature $\Sigma \uplus \{\bullet\}$ that contains a single occurrence of the hole marker, but this time possibly labeling an internal node. Given an unranked context C and an unranked tree $t = a(t_1 \cdots t_n)$, we define *context application* $C[t]$ inductively as follows:

(i) $\bullet(t'_1 \cdots t'_m)[a(t_1 \cdots t_n)] = a(t_1 \cdots t_n t'_1 \cdots t'_m)$; and
(ii) $a(t'_1 \cdots t'_i \cdots t'_m)[t] = a(t'_1 \cdots t'_i[t] \cdots t'_m)$ where t'_i contains the $\bullet$.

We claim that the unranked contexts and context applications that we defined are precisely the Curried versions of the binary contexts.

Lemma 20 *If C is an unranked context and t is an unranked tree, we have that $curry(C[t]) = curry(C)[curry(t)]$.*

The proof is by straightforward induction on the structure of contexts.

The following definitions are parametric, in that they apply to unranked trees as well as to binary trees. A *congruence* on trees is an equivalence relation $\equiv$ such that,

$$\text{for every context } C, \text{ if } t_1 \equiv t_2 \text{ then } C[t_1] \equiv C[t_2].$$

We refer to the number of equivalence classes of an equivalence relation as the *index* of the equivalence relation. An equivalence relation is of *finite index* when there are only a finite number of equivalence classes. Given a tree language T, we define the congruence $\equiv_T$ induced by T through:

$$t_1 \equiv_T t_2 \text{ if and only if for every context } C\text{: } C[t_1] \in T \Leftrightarrow C[t_2] \in T.$$

Given these definitions, the Myhill-Nerode theorem directly generalizes from ranked to unranked tree languages.

Theorem 21 (Myhill-Nerode) *For any binary or unranked tree language T it holds that T is a regular tree language if and only if its congruence $\equiv_T$ has finite index. Furthermore, there exists an (up to isomorphism) unique minimal bottom-up deterministic (stepwise) tree automaton for all regular languages T. The size of this automaton is equal to the index of $\equiv_T$.*

The proof of this theorem is immediate from the binary case [15], Proposition 18, and Lemma 20.

Parallel UTAs are automata for unranked trees which have been independently proposed in [9] and [27] for efficient minimization. In this section, we compare parallel UTAs and stepwise tree automata with respect to the size of minimal deterministic automata and their Myhill-Nerode theorems.

The idea of parallel UTAs is to start with an nUTA and to merge all its nFAs for the same alphabet symbol into one nFA. When each such nFA is a dFA, this should solve the main reason for why efficient minimization fails for dUTAs. In order to distinguish final states of different nFAs after the merge, an explicit output function needs to be added.

Definition 22 A possibly nondeterministic *parallel UTA (nPUTA)* over Σ is a tuple $A = (\text{states}(A), \text{alphabet}(A), (A_a)_{a \in \text{alphabet}(A)}, o)$ where $\text{alphabet}(A) = \Sigma$, every A_a is an nFA, and o is an output function of type $o : \cup_{a \in \Sigma} \text{final}(A_a) \to \text{states}(A)$.

A *run* r of an nPUTA A on an unranked tree t over $\text{alphabet}(A)$ is a function $r : \text{nodes}(t) \to \cup_{a \in \text{alphabet}(A)} \text{states}(A_a)$ such that, for every $\nu \in \text{nodes}(t)$ with n children $\nu 1, \ldots, \nu n$,

$$r(\nu) \in \text{eval}_{A_{\text{lab}^t(\nu)}}(o(r(\nu 1)) \cdots o(r(\nu n))).$$

A run r on t is successful if $o(r(\text{root}(t))) \in \text{final}(A)$.

A *(bottom-up) deterministic parallel UTA (dPUTA)* is a parallel UTA for which every A_a is a dFA. A uPUTA is an unambiguous parallel UTA for which every A_a is a uFA. An example for the minimal dPUTA for the language $\{a(w) \mid w \in L(ab^*)\}$ is given in Figure 8.

The *size* of an nPUTA A is defined to be

$$|\text{states}(A)| + \sum_{a \in \text{alphabet}(A)} |\text{states}(A_a)|.$$

Although not explicitly stated in [9], we note that it is assumed that the state sets of the automata A_a in PUTAs are disjoint. The latter can be concluded from Theorem 26.

Theorem 23 ([9] and [27]) MINIMIZATION *for dPUTAs is in* PTIME. *Furthermore, every regular unranked tree language is recognized by a unique minimal dPUTA (up to isomorphism).*

It is instructive to convert nUTAs into nPUTAs. Let A be an nUTA for which

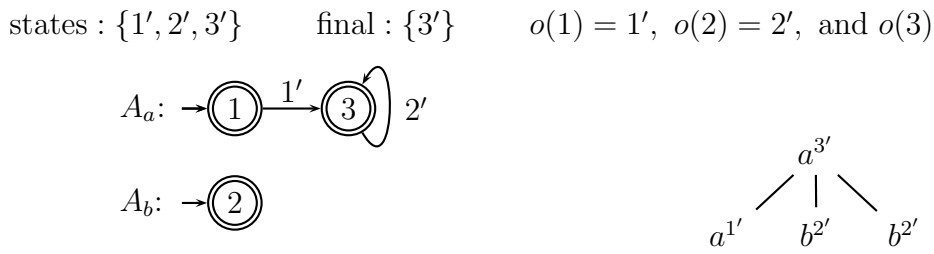


Fig. 8. A dPUTA for $\{a(w) \mid w \in L(ab^*)\}$ and one of its runs. The corresponding stepwise tree automaton is given in Figure 7.

we assume that the state sets of the automata B with $a(L(B)) \to q \in \text{rules}(A)$ are pairwise disjoint. We define an nPUTA PA with the same states and final states such that

- for every $a \in \text{alphabet}(A)$, $PA_a := \cup_{a(L(B))\to q\in\text{rules}(A)} B$; and,
- for every $a(L(B)) \to q \in \text{rules}(A)$, $o(\text{final}(B)) := q$.

The automata PA_a are obtained by unifying all horizontal nFAs for letter a. That is, the state set of PA_a is $\uplus_{a(L(B))\to q\in\text{rules}(A)}\text{states}(B)$ and the rules of PA_a are $\uplus_{a(L(B))\to q\in\text{rules}(A)}\text{rules}(B)$. This transformation preserves unambiguity but not determinism, that is, dUTAs are mapped to uPUTAs. The reason why determinism fails is that the union of dFAs with disjoint languages is an unambiguous, but not necessarily deterministic representation of regular string languages (it may have multiple initial states).

4.2.1 Size Comparison with Stepwise Tree Automata

Every PUTA can be translated into a stepwise tree automaton with fewer or equally many states, such that determinism is preserved. The idea is to unify all nFAs of an PUTA into a single nFA.

Given an nPUTA A, we define an nSTA $\text{step}(A)$ that recognizes the same language. We replace $q \in \text{states}(A)$ by all possible values in $\cup_{a\in\text{alphabet}(A)}\ o^{-1}(q)$, so that the following states remain:

$$\text{states}(\text{step}(A)) := \uplus_{a\in\text{alphabet}(A)}\ \text{states}(A_a)$$
$$\text{final}(\text{step}(A)) := o^{-1}(\text{final}(A))$$

The rules of $\text{step}(A)$ are then given by the following two inference schemata:

$$\frac{q_1 \xrightarrow{p} q_2 \in \text{rules}(A_a) \qquad q \in o^{-1}(p)}{q_1 \xrightarrow{q} q_2 \in \text{rules}(\text{step}(A))} \qquad \frac{q \in \text{init}(A_a) \qquad a \in \text{alphabet}(A)}{q \in \text{init}_a(\text{step}(A))}$$

The stepwise tree automaton in Figure 7, for instance, is the translation of the

PUTA in Figure 8. The main difference is that the nSTA shares the states of all nFAs of the PUTA. As we will see, this kind of sharing allows an automaton to be more succinct in some cases.

In general, the translation preserves runs, successful runs, unambiguity, tree languages, determinism, and the number of states. By composing the two above automata conversions, we obtain:

Proposition 24 *Every dUTA or uUTA can be translated in* PTIME *to an equivalent uPUTA or uSTA with equally many states.*

The latter translation allows us to compare the sizes of minimal deterministic automata for unranked trees:

Theorem 25 *Given a regular tree language T, the minimal dSTA A for T is always smaller or equal in size than the minimal dPUTA. Moreover, the size of the minimal dPUTA for T is in $\mathcal{O}(|\mathit{alphabet}(A)| \cdot |A|)$.*

PROOF. It remains to show the $\mathcal{O}(|\text{alphabet}(A)| \cdot |A|)$ bound of the size increase. We show that any deterministic stepwise tree automaton A can be translated to an equivalent dPUTA PA of size $\mathcal{O}(|\text{alphabet}(A)| \cdot |A|)$. For every $a \in \text{alphabet}(A)$, let A_a be the dFA defined by $\text{states}(A_a) := \{q_a \mid q \in \text{states}(A[\text{init}_a])\}$, $\text{final}(A_a) := \{q_a \mid q \in \text{final}(A[\text{init}_a])\}$, $\text{init}(A_a) := \{q_a \mid q \in \mathit{init}_a\}$, and $\text{rules}(A_a) := \{q_a \xrightarrow{b} q'_a \mid q \xrightarrow{b} q' \in \text{rules}(A)\}$. Then, simply set $PA_a := A_a$ for every $a \in \text{alphabet}(A)$, $\text{states}(PA) := \uplus_{a \in \text{alphabet}(A)} \text{states}(A_a)$, and $o(q) := q$ for every $q \in \text{final}(A)$. □

The minimal stepwise automata can indeed be quadratically smaller than minimal PUTAs, which we will show in Proposition 27, based on the Myhill-Nerode property. This shows that the conversion in the proof of Theorem 25 is optimal.

4.2.2 Myhill-Nerode Property

Cristau, Löding, and Thomas [9] prove a Myhill-Nerode inspired property for dPUTAs. The goal of this section is to compare this Myhill-Nerode property with the previous one for stepwise tree automata. Moreover, the Myhill-Nerode property for dPUTAs allows us to compare the size of minimal dPUTAs with minimal stepwise automata.

A *pointed tree* C over Σ is an unranked context over Σ such that the unique node in C that is labeled by "$\bullet$" is a leaf. For a tree language T, the equivalence

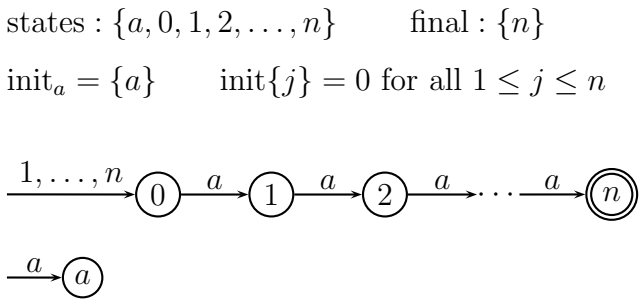


Fig. 9. Deterministic STA for the language T_n of Proposition 27.

relation $\sim_T$ is defined as

$$t_1 \sim_T t_2 \text{ if and only if for every pointed tree } C\text{: } C[t_1] \in T \Leftrightarrow C[t_2] \in T.$$

For two trees $t = a(t_1 \cdots t_k)$ and $t' = a(t'_1 \cdots t'_\ell)$, define

$$t \odot t := a(t_1 \cdots t_k t'_1 \cdots t'_\ell).$$

For $a \in \Sigma$, let $\mathcal{T}^a_\Sigma$ denote the set of Σ-trees which have a as their root label. Then, the equivalence relation $\vec{\sim}_T$ is defined for all $t_1, t_2 \in \mathcal{T}^a_\Sigma$ by

$$t_1 \vec{\sim}_T t_2 \text{ if and only if } \forall t \in \mathcal{T}^a_\Sigma : t_1 \odot t \sim_T t_2 \odot t.$$

Theorem 26 (Theorem 1 in [9], rephrased) *For every regular tree language T, the size of the minimal dPUTA accepting T is $S_T + \sum_{a\in\Sigma} S^a_T$, where*

- *S_T denotes the number of equivalence classes of the relation $\sim_T$; and,*
- *for each $a \in \Sigma$, S^a_T denotes the number of equivalence classes of the relation $\vec{\sim}_T$ in the set $\mathcal{T}^a_\Sigma$.*

Theorem 26 admits us to formally prove that minimal dSTAs are indeed quadratically smaller than minimal dPUTAs in general.

Proposition 27 *There exists a family of unranked regular tree languages $(T_n)_{n\in\mathbb{N}}$ for which the minimal dSTA is quadratically smaller than the minimal dPUTA.*

PROOF. Let Σ_n be the alphabet $\{1, \ldots, n, a\}$ and define the languages

$$T_n := \{j(\underbrace{a \cdots a}_{n}) \mid 1 \leq j \leq n\}.$$

Figure 9 shows a dSTA of size $\mathcal{O}(n)$ accepting T_n.

We show that the minimal dPUTA for T_n has at least n^2 states. Intuitively, the minimal dPUTA for T_n needs n different finite string automata (one for

each i) with n states each (to accept a language consisting of a single string of length n).

Formally, we argue that the equivalence relation $\vec{\sim}_{T_n}$ induces at least n^2 different equivalence classes, which proves the proposition, according to Theorem 26. Thereto, suppose that $t_1 = i(a^k)$ and $t_2 = j(a^\ell)$ are two trees with $i, j, k, \ell \in \{1, \ldots, n\}$.

- If $i \neq j$, then t_1 and t_2 are clearly in different equivalent classes, because the relation $\vec{\sim}_{T_n}$ is only defined between trees with the same root.
- If $i = j$ and $k \neq \ell$, then let t be the tree $i(a^{n-k})$. Then we have that $t \odot t_1 \in T_n$ while $t \odot t_2 \notin T_n$. Taking the context $C = \bullet$, this implies that $t \odot t_1 \not\sim_T t \odot t_2$ since $C[t \odot t_1] \in T_n$ while $C[t \odot t_2] \notin T_n$. Hence, t_1 and t_2 are in different equivalence classes.

It follows that the relation $\vec{\sim}_{T_n}$ induces at least n^2 different equivalence classes. □

4.3 Standard Binary Encoding

Another approach towards efficient minimization for automata representing unranked tree languages is to use the *first-child next-sibling* encoding [10,21,33].

The first-child next-sibling encoding $\mathrm{fcns}(t)$ of some unranked tree t over Σ is a binary tree over the signature $\Sigma_\perp = \Sigma \uplus \{\perp\}$, where the first-child relation is associated with the first position, and the next-sibling relation with the second position.

The idea of using the first-child next-sibling encoding for minimization, is to represent a regular language of unranked trees T by a minimal dTA for the language of their binary encodings $\mathrm{fcns}(T)$, as with stepwise tree automata that recognize the binary tree language $\mathrm{curry}(T)$.

4.3.1 Inversion

The goal of this section is to compare the size of the dTAs for $\mathrm{fcns}(T)$ and $\mathrm{curry}(T)$ for regular languages of unranked trees T. Figure 10 illustrates these two binary encodings and two others at the example of the unranked tree $t = a(bcd)$.

The first important difference between $\mathrm{fcns}(T)$ and $\mathrm{curry}(T)$ is that lists of children dare inverted. When traversing $\mathrm{fcns}(t)$ bottom-up, the list $(b\ c\ d)$ of a's children is encountered in inverted order $(d\ c\ b)$, while it occurs in the

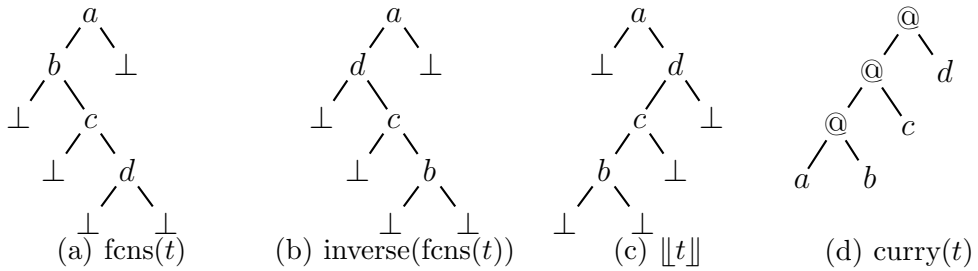


Fig. 10. Four binary encodings of the unranked tree $t = a(b\ c\ d)$: first-child next-t-sibling fcns(t), inverted first-child next-sibling inverse(fcns(t)), previous-sibling last-child $\llbracket t \rrbracket$, and the Curried encoding curry(t).

original order in curry(t).

It is well known for dFAs that language inversion leads to an exponential blow-up of their minimal size. As a consequence, there is in general an exponential blow-up between the minimal dTAs for fcns(T) and curry(T) in both directions. For instance, for the tree languages $T_n = \{c(w) \in \mathcal{T}_\Sigma \mid w \in (a+b)^n a(a+b)^*\}$, where $n \in \mathbb{N}$, the minimal dTA over the curry-encoding is exponentially smaller than the minimal dTA over the fcns-encoding. For the translation in the other direction, the exponential blow-up occurs for the tree languages $T'_n = \{c(w) \in \mathcal{T}_\Sigma \mid w \in (a+b)^* a(a+b)^n\}$.

We wish to ignore such succinctness differences due to inversion. Our goal thus becomes to compare the inverted first-child next-sibling encoding with Currying. Finally, the previous-sibling last-child encoding $\llbracket . \rrbracket$ is equal to the inverted fcns-encoding, except that first and second positions are exchanged for all nodes. This is a minor difference which does not affect the succinctness of minimal bottom-up deterministic automata. The inverted fcns-encoding and the previous-sibling last-child encoding $\llbracket . \rrbracket$ are illustrated in Figure 10. We wish to turn to the previous-sibling last-child encoding since it facilitates constructions later in the article.

The main difference that remains between the previous-sibling last-child encoding $\llbracket t \rrbracket$ and curry(t) in the above example, is that t's root's label a is located at the root of $\llbracket t \rrbracket$, while it is found in the leftmost leaf of curry(t). In bottom-up processing, one sees leaves first, so the Curried encoding should have advantages for minimization.

4.3.2 Size Comparison to Stepwise Tree Automata

We show that minimal deterministic STAs for languages T of unranked trees are at most quadratically larger than dTAs for the previous-sibling last-child encoding $\llbracket T \rrbracket$. On the other hand, they can be exponentially smaller.

Let us define the previous-sibling last-child encoding $\llbracket t \rrbracket$ of some unranked

tree t of Σ more formally. It is a binary tree over the signature $\Sigma_\perp = \Sigma \uplus \{\perp\}$, where the previous-sibling relation is associated with the first position and the last-child relation with the second position:

$$\begin{aligned} [\![a(t_1 \cdots t_n)]\!] &:= [\![\langle a(t_1 \cdots t_n)\rangle]\!] \\ [\![\langle t_1 \cdots t_n a(s_1 \cdots s_m)\rangle]\!] &:= a([\![\langle t_1 \cdots t_n\rangle]\!][\![\langle s_1 \cdots s_m\rangle]\!]) \\ [\![\langle\rangle]\!] &:= \perp \end{aligned}$$

In order to relate the language $[\![L]\!]$ to curry(L), we define a tree transformation "shift" that transforms a tree $[\![t]\!]$ to curry(t). Intuitively, the transformation processes the tree $[\![t]\!]$ in a top-down manner and moves the labels of parents (in the unranked tree) downwards. On the example in Figure 10(c), it would move the a downwards to obtain the tree in Figure 10(d). Formally, the transformation is defined as follows:

$$\begin{aligned} \text{shift}(a(\perp\ t)) &:= \text{shift}_a(t) \\ \text{shift}_a(b(t_1\ t_2)) &:= @(\text{shift}_a(t_1)\ \text{shift}_b(t_2)) \\ \text{shift}_a(\perp) &:= a \end{aligned}$$

The following simple equality will be useful in the proofs to come:

$$\text{shift}([\![a(t_1 \cdots t_n)]\!] = \text{shift}_a([\![\langle t_1 \cdots t_n\rangle]\!]) \qquad (\dagger)$$

It holds by definition of the encoding $[\![.]\!]$ and the shift transformation.

Proposition 28 *For every unranked tree t over Σ, shift$([\![t]\!])$ = curry(t).*

PROOF. The proof is by induction on the structure of unranked trees. The base case $t = a$ is simple: $\text{shift}([\![a]\!]) = \text{shift}_a(\perp) = a = \text{curry}(a)$.

In the induction, we have $t = a(t_1 \cdots t_n b(s_1 \cdots s_m))$, where n and m can be zero, so we can apply equation $(\dagger)$ and the definitions of $[\![.]\!]$ and shift_a:

$$\begin{aligned} \text{shift}([\![t]\!]) &= \text{shift}_a([\![\langle t_1 \cdots t_n b(s_1 \cdots s_m)\rangle]\!]) \\ &= \text{shift}_a(b([\![\langle t_1 \cdots t_n\rangle]\!]\ [\![\langle s_1 \cdots s_m\rangle]\!]) \\ &= @(\text{shift}_a([\![\langle t_1 \cdots t_n\rangle]\!])\ \text{shift}_b([\![\langle s_1 \cdots s_m\rangle]\!])) \end{aligned}$$

$$S1 \quad \frac{\bot \to p \in \text{rules}_A \qquad a \in \Sigma}{a \to p[a] \in \text{rules}(\text{step}(A))}$$

$$S2 \quad \frac{b(p_1, p_2) \to p \in \text{rules}_A \qquad a \in \Sigma}{@(p_1[a], p_2[b]) \to p[a] \in \text{rules}(\text{step}(A))}$$

Fig. 11. Converting automata A for previous-sibling last-child encodings of unranked trees into stepwise tree automata $\text{step}(A)$.

We are now in the position to apply the induction hypothesis, and to conclude

$$\begin{aligned}\text{shift}(\llbracket t \rrbracket) &= @(\text{curry}(a(t_1 \cdots t_n))\ \text{curry}(b(s_1 \cdots s_m))) \\ &= \text{curry}(a(t_1 \cdots t_n b(s_1 \cdots s_m))) \\ &= \text{curry}(t)\end{aligned}$$

by the definition of the Curried encoding. □

Our next goal is to encode nTAs over $\Sigma_\bot$ into nTAs over $\Sigma_@$ that recognize the shifted language. The size should grow no more than quadratically and bottom-up determinism should be preserved.

The idea of the automata conversion is to memorize node labels that have been shifted down. In bottom-up processing, these labels will be seen earlier than needed, so we simply memorize them in the state when moving upwards. Given a traditional tree automaton A over $\Sigma_\bot$ we define an automaton $\text{step}(A)$ over $\Sigma_@$ such that:

$$\text{states}(\text{step}(A)) = \text{states}(A) \times \Sigma$$

We write $p[a]$ for pairs (p, a) where $p \in \text{states}(A)$ and $a \in \Sigma$. Note that $|\text{step}(A)| = |\Sigma| \cdot |A|$, so the size increases at most by a factor of $|\Sigma|$. The rules of $\text{step}(A)$ are produced by the inference rules in Figure 11. It remains to define the final states of $\text{step}(A)$. Thereto, for ease of notation, we extend the definition of eval_A to binary trees in which the leaves can be labeled with states. In particular, we define $\text{eval}_A(p) := \{p\}$ for every $p \in \text{states}(A)$ and $\text{eval}_A(a(t_1 t_2)) := \{q \mid q_1 \in \text{eval}_A(t_1), q_2 \in \text{eval}_A(t_2), \text{ and } a(p_1, p_2) \to p \in \text{rules}(A)\}$ for binary $(\text{states}(A) \cup \text{alphabet}(A))$-trees t_1, t_2 in which the labels from $\text{states}(A)$ only occur at leaves. Then, the final states of $\text{step}(A)$ are defined as

$$\text{final}(\text{step}(A)) := \{p[a] \mid \text{eval}_A(a(\bot\ p)) \cap \text{final}(A) \neq \emptyset\}.$$

We illustrate the conversion in Figure 12. It presents a run of some automaton

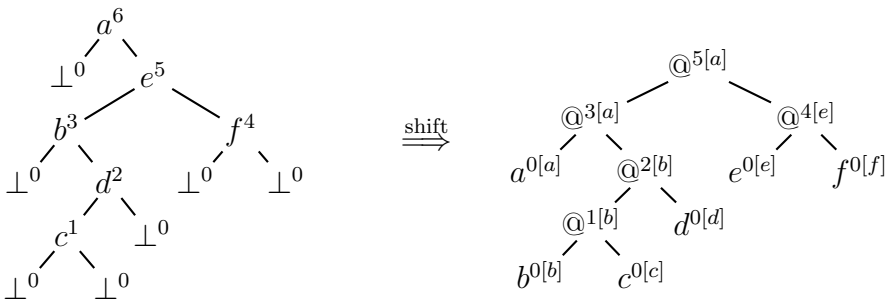


Fig. 12. A run of some tree automaton A on the previous-sibling last-child encoding of the unranked tree $a(b(c\ d)\ e(f))$ and the corresponding run of step(A) on the Curried encoding.

A on the previous-sibling last-child encoding of the unranked tree $a(b(cd)\ e(f))$ and the corresponding run of step(A) on the Curried encoding.

Lemma 29 *Let t be a binary tree over $\Sigma_\bot$ and A a traditional tree automaton over $\Sigma_\bot$. It then holds for all $p \in states(A)$ and $a \in \Sigma$ that*

$$p \in eval_A(t) \text{ if and only if } p[a] \in eval_{step(A)}(shift_a(t)).$$

PROOF. By induction on the structure of t. If $t = \bot$ then the lemma follows from the definition of shift_a and inference rule S1:

$$\begin{aligned} p \in \text{eval}_A(t) &\text{ if and only if } \bot \to p \in \text{rules}(A) \\ &\text{if and only if } a \to p[a] \in \text{rules}(\text{step}(A)) \\ &\text{if and only if } p[a] \in \text{eval}_{\text{step}(A)}(a) \\ &\text{if and only if } p[a] \in \text{eval}_{\text{step}(A)}(\text{shift}_a(t)) \end{aligned}$$

Otherwise, $t = b(t_1 t_2)$ for some $b \in \Sigma$ and binary trees t_1, t_2 over $\Sigma_\bot$. For the one direction, we assume $p \in \text{eval}_A(t)$. Hence, there exists $b(p_1, p_2) \to p \in \text{rules}(A)$ such that $p_1 \in \text{eval}_A(t_1)$ and $p_2 \in \text{eval}_A(t_2)$. The induction hypothesis applied to t_1 and t_2 yields that $p_1[a] \in \text{eval}_{\text{step}(A)}(\text{shift}_a(t_1))$ and $p_2[b] \in \text{eval}_{\text{step}(A)}(\text{shift}_b(t_2))$. Since $b(p_1, p_2) \to p \in \text{rules}(A)$, we can apply inference rule S2 of the construction of step(A) in Figure 11 to obtain

$$p[a] \in \text{eval}_{\text{step}(A)}(@(\text{shift}_a(t_1)\ \text{shift}_b(t_2))).$$

This is equivalent to $p[a] \in \text{eval}_{\text{step}(A)}(\text{shift}_a(t))$, as required.

For the other direction, let $p[a] \in \text{eval}_{\text{step}(A)}(\text{shift}_a(t))$. There exists a rule of step(A) by which to infer $p[a]$. Since $\text{shift}_a(t) = \text{shift}_a(b(t_1 t_2)) = @(\text{shift}_a(t_1)\ \text{shift}_b(t_2))$, it must be inferred form S2 and have the form

$$@(p_1[a], p_2[b]) \to p[a],$$

for some $b(p_1, p_2) \to p \in \text{rules}(A)$ for which $p_1[a] \in \text{eval}_{\text{step}(A)}(\text{shift}_a(t_1))$ and $p_2[b] \in \text{eval}_{\text{step}(A)}(\text{shift}_b(t_2))$. The induction hypothesis applied to t_1 and t_2 yields $p_1 \in \text{eval}_A(t_1)$ and $p_2 \in \text{eval}_A(t_2)$. Thus, $p \in \text{eval}_A(b(t_1, t_2))$, that is, $p \in \text{eval}_A(t)$, as required. □

Proposition 30 *For every nTA A over $\Sigma_\perp$ accepting the previous-sibling last-child encoding of some unranked tree language,*

$$L(\mathit{step}(A)) = \mathit{shift}(L(A)).$$

PROOF. Let $s \in \text{shift}(L(A))$ be a binary tree over $\Sigma \cup \{@\}$. There is some tree $t \in L(A)$ over $\Sigma_\perp$ such that $s = \text{shift}(t)$. By definition of the shift function, we have $t = a(\perp\ t_2)$ for some $a \in \Sigma$ and tree t_2 with $\text{shift}(t) = \text{shift}_a(t_2)$. Furthermore, there exists a $p \in \text{final}(A) \cap \text{eval}_A(t)$. Let p_2 be such that $p \in \text{eval}_A(a(\perp\ p_2))$. Note that $p_2[a] \in \text{final}(\text{step}(A))$. Lemma 29 proves $p[a] \in \text{eval}_{\text{step}(A)}(\text{shift}_a(t_2))$. Thus $s = \text{shift}_a(t_2) \in L(\text{step}(A))$.

For the converse, let $s \in L(\text{step}(A))$. The shift function is one-to-one and onto, so there exists some tree t such that $s = \text{shift}(t)$. It remains to show that $t \in L(A)$. By definition of the shift function, t has the form $a(\perp\ t_2)$ and $s = \text{shift}(t) = \text{shift}_a(t_2)$. There exists $p[a] \in \text{final}(\text{step}(A))$ such that $p[a] \in \text{eval}_{\text{step}(A)}(\text{shift}_a(t_2))$. By Lemma 29, it follows that $p \in \text{eval}_A(t_2)$. By definition of $\text{final}(\text{step}(A))$ it holds that $\text{eval}_A(a(\perp\ p)) \cap \text{final}(A) \neq \emptyset$. Thus, $t = a(\perp\ t_2) \in L(A)$ so that $s = \text{shift}(t) \in \text{shift}(L(A))$. □

Theorem 31 *For every regular language T of unranked trees over Σ, the size of the minimal dTA the previous-sibling last-child encoding $[\![T]\!]$ is at most $|\Sigma|$ times smaller than the minimal deterministic stepwise tree automaton for T.*

PROOF. Let A be the minimal deterministic automaton recognizing $[\![T]\!]$. The automaton $\text{step}(A)$ is deterministic and a factor of $|\Sigma|$ larger than A and recognizes $\text{curry}(T)$:

$$\begin{aligned} L(\text{step}(A)) &= \text{shift}(L(A)) && \text{by Proposition 30} \\ &= \text{shift}([\![T]\!]) \\ &= \text{curry}(T) && \text{by Proposition 28} \end{aligned}$$

By Proposition 18, the minimal dSTA recognizing T is thus smaller or equal in size to $\text{step}(A)$, that is, at most a factor of $|\Sigma|$ larger than A. □

We give two examples relating minimal dTAs with respect to the previous-sibling last-child encoding to minimal dSTAs. The first example proves that

the quadratic construction of Theorem 31 is optimal. The second one illustrates that minimal dSTAs can be exponentially smaller than minimal dTAs over the previous-sibling last-child encodings.

Proposition 32 *There exists an infinite class of languages $(T_n)_{n\in\mathbb{N}}$ such that, for every T_n, the minimal dSTA for T_n is quadratically larger than than the minimal dTA for $[\![T_n]\!]$.*

PROOF. For every $n \in \mathbb{N}$, we define a tree language T_n such that the minimal dSTA for T_n is quadratically larger than the minimal tree automaton accepting $[\![T_n]\!]$. Indeed, consider, for every $n \in \mathbb{N}$, the regular tree language $T_n = \{b_i(b_i(\underbrace{a \cdots a}_{n})) \mid 1 \leq i \leq n\}$ over the alphabet $\Sigma_n = \{b_1, \ldots, b_n, a\}$.

The following dTA A_n with alphabet$(A) = \Sigma_n \cup \{\bot\}$ recognizes $[\![T_n]\!]$, has $2n+2$ states $\{a_1, \ldots, a_n, b_1, \ldots, b_n, \bot, \mathrm{ok}\}$ where ok is the only final state, and the following rules:

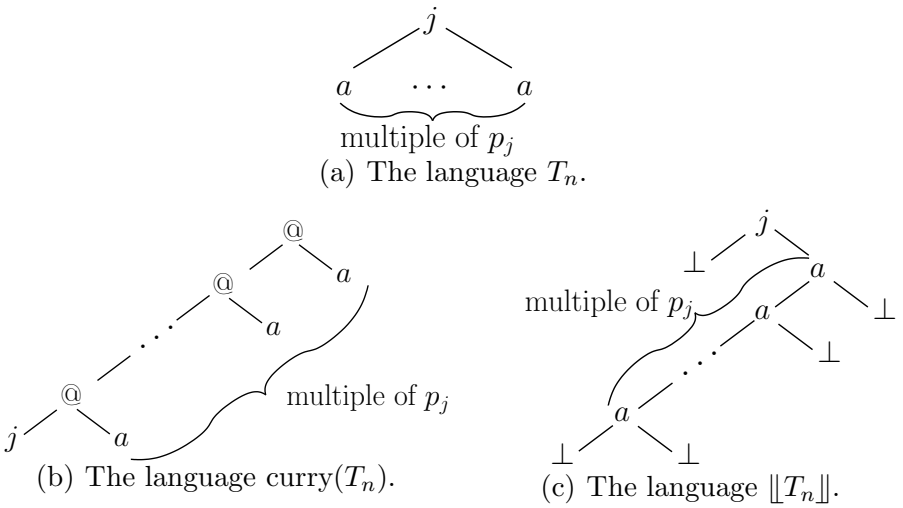
- $\bot \to \bot$;
- $a(\bot\ \bot) \to a_1$;
- $a(a_k\ \bot) \to a_{k+1}$, for every $1 \leq k < n$;
- $b_i(\bot\ a_n) \to b_i$, for every $1 \leq i \leq n$; and,
- $b_i(\bot\ b_i) \to \mathrm{ok}$.

We show that the minimal stepwise automaton for T_n has size at least $n^2 + n + 2$. To this end, we apply the Myhill-Nerode Theorem 21 for stepwise tree automata. We show that index of $\equiv_{T_n}$ is at least $n^2 + n + 2$. It is easy to see that the sets T_n, $\{a\}$, and $\{b_i\}$ for $i = 1, \ldots, n$ form $n + 2$ equivalence classes of $\equiv_{T_n}$. Furthermore, consider the trees $t_{i_1,j_1} = b_{i_1}(a^{j_1})$ and $t_{i_2,j_2} = b_{i_2}(a^{j_2})$ for $1 \leq i_1, i_2, j_1, j_2 \leq n$. Suppose that $i_1 \neq i_2$ or $j_1 \neq j_2$, and consider the context $C = b_{i_1}(\bullet(a^{n-j_1}))$. Then we have that $C[t_{i_1,j_1}] \in T_n$, while $C[t_{i_2,j_2}] \notin T_n$. Hence, each t_{i_1,j_1} and t_{i_2,j_2} are in different equivalence classes when $i_1 \neq i_2$ or $j_1 \neq j_2$, which implies that that the index of $\equiv_{T_n}$ is at least $n^2 + n + 2$. □

The translation from minimal dSTAs to minimal dTA for the previous-sibling last-child encoding of its language can be worse than quadratic, that is, exponential.

Proposition 33 *There exists an infinite class of languages $(T_n)_{n\in\mathbb{N}}$ such that for every T_n, the minimal dSTA for T_n is exponentially smaller than the minimal dTA for the encoding $[\![T_n]\!]$.*

PROOF. The proof is based on the fact that the smallest dFA for the union of an arbitrary number of dFAs can be exponentially larger than the sum



(a) The language T_n.

(b) The language curry(T_n).

(c) The language $[\![T_n]\!]$.

Fig. 13. Illustration of the languages used in the proof of Proposition 33.

of their sizes (see, for example, [26]). Indeed, let A_j to be the minimal dFA accepting $(a^{p_j})^*$, where p_j denotes the j-th prime number. Then, the minimal size of the dFA for $(a^{p_1})^* \cup \cdots \cup (a^{p_n})^*$ is $\prod_{j=1,\ldots,n} p_j$, which is exponentially larger than $\sum_{j=1,\ldots,n} p_j$ when n is arbitrary. The proposition now holds for the tree languages T_n with alphabet $\{1, \ldots, n, a\}$:

$$T_n := \bigcup_{j=1,\ldots,n} \{j(w) \mid w \in L(A_j)\}.$$

We first show that, for every $n \in \mathbb{N}$, there exists a dSTA for T_n of size $\sum_{j=1,\ldots,n} p_j$. Let B_n be the minimal dFA with alphabet$(B_n) = \{1, \ldots, n, a\}$ that accepts the string language $\cup_{j=1}^{n} j(a^{p_j})^*$. The size of B_n is $1 + \sum_{j=1,\ldots,n} p_j$. It can be turned into a stepwise automaton A for T_n by removing the initial state of B_n, adding the state $a = \text{init}_a(A)$, and setting setting $\text{init}_j(A) = \text{eval}_{B_n}(j)$, resulting in a size of $1 + \sum_{j=1,\ldots,n} p_j$.

We show that the minimal dTA for $[\![T_n]\!]$ has size at least $2 + \prod_{j=1,\ldots,n} p_j$ by showing that the index of $\equiv_{[\![T_n]\!]}$ is at least that large. We only consider equivalence classes that contain a tree t for which there exists a context C such that $C[t] \in T_n$. One equivalence class of $\equiv_{[\![T_n]\!]}$ consists precisely of the trees in $[\![T_n]\!]$. Notice that these trees always have some j as their root symbol. A second equivalence class consists of the singleton $\{\bot\}$. The remaining N equivalence classes consist of trees that have their root labeled with a. These equivalence classes are isomorphic to the equivalence classes induced by the minimal dFA for $(a^{p_1})^* \cup \cdots \cup (a^{p_n})^*$. Indeed, let ϕ be the function that maps every binary tree of the form $a(a(\cdots a(\bot\ \bot)\cdots \bot)\ \bot)$ (with k occurrences of a) to the string a^k. Then, ϕ is an isomorphism. It is easy to see that a set of trees S is an equivalence class of $\equiv_{[\![T_n]\!]}$ if and only if $\phi(S)$ is an equivalence class of $\equiv_{(a^{p_1})^* \cup \cdots \cup (a^{p_n})^*}$. Hence, $N = \prod_{j=1,\ldots,n} p_j$. □

5 Models for XML Schema Languages

We now focus on abstractions for XML schema languages. In the literature, XML schema languages are usually abstracted as *extended DTDs* [25] instead of tree automata. [2] We will follow this convention. In particular, we will treat extended DTDs with the *single-type* and the *restrained competition* restrictions, which correspond to the expressive power of XML Schema [30] and 1-pass preorder typeable schemas [18], respectively. As remarked by Cristau et al., restrained competition extended DTDs can be seen as a restricted version of the top-down deterministic tree automata studied in their paper [9].

We recall the notion of a DTD, which is the most widely used XML schema language:

Definition 34 A *DTD* over Σ is a triple

$$d = (\text{alphabet}(d), \text{rules}(d), \text{start}(d)),$$

where $\text{alphabet}(d) = \Sigma$. For every $a \in \text{alphabet}(d)$, $\text{rules}(d)$ contains precisely one rule of the form $a \to D_a$, where D_a is a dFA over $\text{alphabet}(d)$, and $\text{start}(d) \in \text{alphabet}(d)$ is the start symbol. A tree t *is valid with respect to* d (or *satisfies* d) if its root is labeled with $\text{start}(d)$ and, for every node with label a and sequence $a_1 \cdots a_n$ of labels of its children, there is a rule $a \to D_a$ in $\text{rules}(d)$ such that $a_1 \cdots a_n \in L(D_a)$.

We define the *size* $|d|$ of a DTD d to be $\sum_{a \in \text{alphabet}(d)} |D_a|$. By $L(d)$ we denote the set of trees that satisfy d. By $d[\text{start} = a]$ we denote the DTD d in which the start symbol is replaced by a.

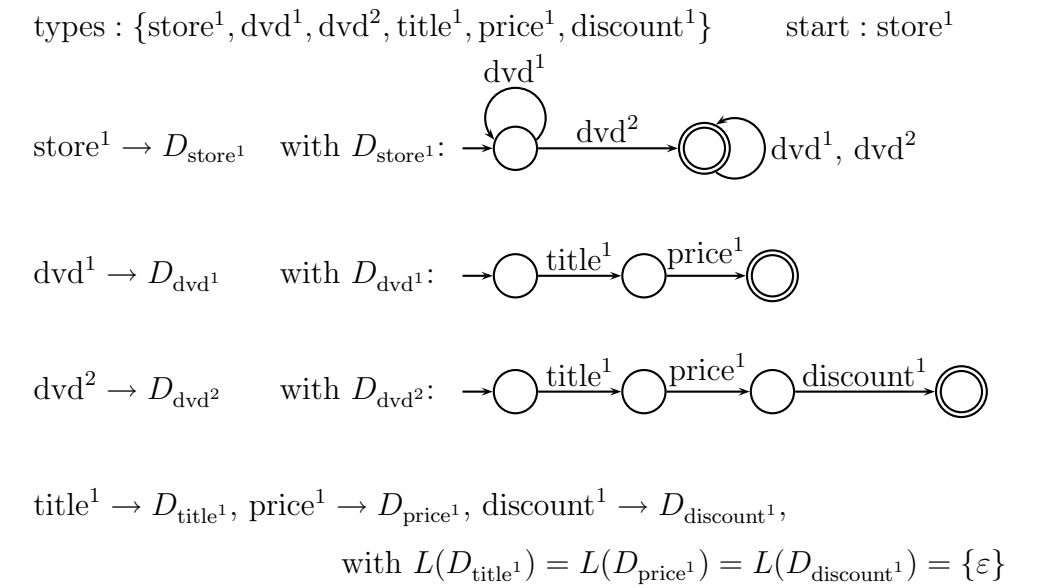
Given a DTD d, we say that the symbol $a \in \text{alphabet}(d)$ is *reachable* in d when either *(i)* $a = \text{start}(d)$, or *(ii)* b is reachable, $b \to D_b$ is a rule in d, and there exist strings $w_1, w_2 \in \text{alphabet}(d)^*$ such that $w_1 a w_2 \in L(D_b)$. We say that d is *reduced* if, for every symbol $a \in \text{alphabet}(d)$, a is reachable and $L(d[\text{start} = a]) \neq \emptyset$. Notice that, when d is reduced, for every $a \in \text{alphabet}(d)$, there exists a tree $t \in L(d)$ such that a is a label in t.

Definition 35 ([2,25]) An *extended DTD* (EDTD) over Σ is a quadruple

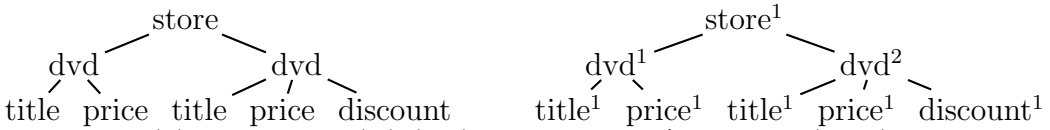
$$E = (\text{alphabet}(E), \text{types}(E), \text{dtd}(E), \text{start}(E), \text{name}_E),$$

where $\text{alphabet}(E) = \Sigma$, $\text{types}(E)$ is an alphabet of *types*, $\text{dtd}(E)$ is a DTD over $\text{types}(E)$, $\text{start}(E) \in \text{types}(E)$ is the start symbol of $\text{dtd}(E)$, and name_E

[2] Papakonstantinou and Vianu used the term *specialized DTD*, as types *specialize* tags. We prefer the term *extended DTD* as it expresses more clearly that the power of the schemas is amplified.



(a) An EDTD E defining a store document.



(b) A tree $t \in L(E)$ (left) and a typing t' of E on t (right).

Fig. 14. An example of an EDTD defining a schema for a store with DVDs.

is a mapping from types(E) to alphabet(E). We extend the function name_E in the homomorphic way to strings and trees over types(E).

A tree t is *valid with respect to* E (or satisfies E) if $t = \text{name}_E(t')$ for some tree $t' \in L(\text{dtd}(E))$. Again, we denote by $L(E)$ the set of trees satisfying E. For a symbol $s \in \text{types}(E)$, we denote by $E[\text{start} = s]$ the extended DTD E where the start symbol start(E) is replaced by s.

For ease of exposition, we always take $\text{types}(E) = \{a^i \mid 1 \leq i \leq k_a, a \in \text{alphabet}(E), i \in \mathbb{N}\}$ for some $k_a \in \mathbb{N}$, and we set $\text{name}_E(a^i) = a$. We refer to the label a^i of a node in t' as its *type*. If $t \in L(E)$ and $t' \in \text{dtd}(E)$ with $\text{name}_E(t') = t$, we also say that t' is a *typing of* E *on* t. The *size* $|E|$ of an extended DTD E is $|\text{types}(E)| + |\text{dtd}(E)|$. We say that E is *reduced* if dtd(E) is reduced.

Example 36 Figure 14(a) contains an EDTD E which defines a store that sells two types of DVDs: dvd^1 defines ordinary DVDs, while dvd^2 defines DVDs on sale, which is reflected by an extra "discount"-child. The rule for store^1 specifies that there should be at least one DVD on sale. Figure 14(b) shows a tree defined by the EDTD, together with its typing.

Remark 37 It is well-known that EDTDs can be identified with nUTAs.

In particular, given an EDTD E, one can obtain an nUTA A equivalent to E by setting alphabet(A) = alphabet(E), states(A) = types(E), final(A) = start(E), and by including the rule $a(L(D_{a^i})) \to a^i$ in rules(A) for every rule $a^i \to D_{a^i} \in$ rules(dtd(E)). It is easy to see that $L(A) = L(E)$ and that A can be constructed from E in linear time. In this perspective, a typing of E corresponds to a run of A.

We formally define single-type and restrained competition EDTDs as follows.

Definition 38 Let E be an EDTD. We say that a regular language L over alphabet types(E) is *single-type* if, for every two strings $w_1 a^i v_1$ and $w_2 a^j v_2$ in L, we have that $i = j$. We say that L *restrains competition* if, for every two strings $wa^i v_1$ and $wa^j v_2$ in L, we have that $i = j$.

An EDTD E is *single-type* (respectively, *restrained competition*) if every regular language defined by dFAs D_{a^i} in the definition of dtd(E) is single-type (respectively, restrains competition).

Example 39 The EDTD E of Figure 14(a) is not single-type or restrained competition. If we replace D_{store} with a dFA defining the language $\text{dvd}^2(\text{dvd}^1)^*$, then the EDTD is restrained competition. If we replace D_{store} with a dFA defining the language $(\text{dvd}^1)^*$, then the EDTD is single-type.

The goal of this section is to prove the following theorem:

Theorem 40

(1) MINIMIZATION *for restrained competition EDTDs is in* PTIME.
(2) Minimal restrained competition EDTDs are unique up to isomorphism.
(3) MINIMIZATION *for single-type EDTDs is in* PTIME.
(4) Minimal single-type EDTDs are unique up to isomorphism.

We first give the minimization algorithm for restrained competition EDTDs and prove Theorem 40(1) and (2) in a series of lemmas. We then observe that the obtained results also carry over to single-type EDTDs.

Let E be a restrained competition EDTD. We assume without loss of generality that each dFA D_{a^i} in E is minimal. The following algorithm minimizes E, i.e., computes an equivalent minimal restrained competition EDTD.

(1) *Reduce* E, that is,
   (a) remove all symbols a^i from types(E) for which $L(E[\text{start} = a^i]) = \emptyset$, remove the corresponding rules $a^i \to D_{a^i}$ from dtd(E), and remove the corresponding transitions of the form $q_1 \xrightarrow{a^i} q_2$ in every dFA in dtd(E); and,
   (b) remove all symbols a^i from types(E) which are not reachable in dtd(E),

remove the corresponding rules $a^i \to D_{a^i}$, and remove the corresponding transitions of the form $q_1 \xrightarrow{a^i} q_2$ in every dFA in dtd(E).

(2) Test, for each a^i and a^j in types(E) with $i < j$, whether $L(E[\text{start} = a^i]) = L(E[\text{start} = a^j])$. If this is so, then
   (a) replace all occurrences of a^j in the definition of dtd(E) by a^i. That is, for every $b^k \in \text{types}(E)$, replace every transition rule $q_1 \xrightarrow{a^j} q_2$ in rules(D_{b^k}) by $q_1 \xrightarrow{a^i} q_2$.
   (b) remove the rule $a^j \to D_{a^j}$ from dtd(E); and,
   (c) remove a^j from types(E).

(3) For each rule $a^i \to D_{a^i}$ in dtd(E), minimize the dFA D_{a^i}.

We argue that the algorithm can be executed in polynomial time. Step (1) can be performed in polynomial time by a polynomial number of emptiness and reachability tests of DTDs. Testing whether a DTD defines an empty language is known to be in PTIME because of the correspondence with EDTDs and nUTAs as explained in Remark 37, and testing emptiness of nUTAs is known to be in PTIME (see, e.g., [17]). Testing whether a symbol is reachable is in NLOGSPACE, by a straightforward reduction to graph reachability. Step (2) is in polynomial time since testing inclusion of restrained competition EDTDs is in PTIME (Theorem 10.4 in [18]). For an alternative, less direct proof that inclusion of restrained competition EDTDs is in PTIME, one can also observe that the polynomial time conversion of a restrained competition EDTD to an nUTA in Remark 37 gives rise to a uUTA. According to Theorem 5, we can test equivalence between uUTAs in PTIME. To show that step (3) can be carried out in polynomial time, we need to argue that, for each rule $a^i \to D_{a^i}$, the automaton D_{a^i} is still deterministic, as we replaced some of its transitions in step (2)(a). Thereto, take, for an arbitrary $b^k \in \text{types}(E)$, the dFA D_{b^k} before execution of step (2)(a). Since $L(D_{b^k})$ restrains competition and D_{b^k} is a minimal dFA, we have that D_{b^k} does not contain any transitions of the form $q_1 \xrightarrow{a^i} q_2$ and $q_1 \xrightarrow{a^j} q_3$ with $q_2 \neq q_3$ or $i \neq j$. Therefore, replacing all occurrences of a^j in the definition of D_{b^k} by a^i preserves the determinism in D_{b^k} and the restrained competition property of $L(D_{b^k})$. Consequently, in step (3), we still have that each automaton D_{a^i} is deterministic. Since minimizing dFAs is in polynomial time, step (3) can also be carried out in polynomial time.

Let $E_{\min}$ be the EDTD obtained by applying the above minimization algorithm on a restrained competition EDTD E. We will show that $E_{\min}$ is the minimal restrained competition EDTD for $L(E)$. More formally, we need that

(a) $E_{\min}$ is restrained competition;
(b) $L(E_{\min}) = L(E)$; and that
(c) every minimal restrained competition EDTD E_0 for $L(E)$ is isomorphic to $E_{\min}$.

We already argued above that (a) holds. It can be shown that (b) holds by a straightforward structural induction on the trees defined by dtd(E) and by using the fact that, in step (2)(a) of the algorithm, we have only replaced types a^i by types a^j that define the same set of Σ-trees. The proof of (c), however, is more complicated; we proceed with showing (c) in a series of lemmas.

We start with some terminology. Let t be a tree and v be a node in t. The *ancestor-sibling-string* of v is the string formed by the ancestors of v and all their left siblings. More formally, for a node $v = uk$ in a Σ-tree t with $k \in \mathbb{N}_0$, we denote by l-sib-str$^t(v)$ the string formed by the label of the v and the labels of its left siblings, that is, $\text{lab}^t(u1) \cdots \text{lab}^t(uk)$. Let $v = i_1 i_2 \cdots i_\ell$ with $i_1, i_2, \ldots, i_\ell \in \mathbb{N}_0$. By anc-sib-str$^t(v)$ we denote the ancestor-sibling-string

$$\text{l-sib-str}^t(\varepsilon)\#\text{l-sib-str}^t(i_1)\#\cdots\#\text{l-sib-str}^t(i_1 i_2 \cdots i_\ell)$$

formed by concatenating the left-sibling-strings of all ancestors of v starting from the root. We assume that the special marker "#" does not occur in Σ.

For two Σ-trees t_1 and t_2, and a node $u \in \text{nodes}(t_1)$, we denote by $t_1[u \leftarrow t_2]$ the tree obtained from t_1 by replacing its subtree rooted at u by t_2.

Definition 41 We say that an EDTD E over Σ *has ancestor-sibling-based typings* if there is a (partial) function

$$f : (\Sigma \cup \{\#\})^* \to \text{types}(E)$$

such that, for each tree $t \in L(E)$ and typing t' of E on t, we have that, for each node $v \in \text{nodes}(t')$,

$$\text{lab}^{t'}(v) = f(\text{anc-sib-str}^t(v)).$$

Notice that, if E has ancestor-sibling-based typings, there is a unique typing of E on t for each $t \in L(E)$.

We start by proving the following basic property of restrained competition EDTDs:

Lemma 42 *Every restrained competition EDTD has ancestor-sibling-based typings.*

PROOF. Let E be a restrained competition EDTD over Σ. Notice that a language L over types(E) is restrained competition if and only if, for every two strings $w_1 a^i v_1$ and $w_2 a^j v_2$ in L, if $\text{name}_E(w_1) = \text{name}_E(w_2)$ then $i = j$.

We assume w.l.o.g. that E is reduced. We define the function $f : (\Sigma \cup \{\#\}) \to \text{types}(E)$ inductively as follows : $f(\text{name}_E(\text{start}(E))) = \text{start}(E)$. Further,

for every string $w_0\#wa$ with $w_0 \in (\Sigma \cup \#)^*$, $w \in \Sigma^*$, and $a \in \Sigma$, we define $f(w_0\#wa) = a^i$ where $f(w_0) = b^j$ and a^i is the unique type such that $w_1 a^i v_1 \in L(D_{b^j})$ with $\text{name}_E(w_1) = w$. As E is restrained competition, f is well-defined and induces a unique typing. □

Lemma 43 *Let E_1 and E_2 be reduced, equivalent restrained competition EDTDs and let $t \in L(E_1) = L(E_2)$. Let t'_1 and t'_2 be the unique typings of E_1 and E_2 on t, respectively, and let u be a node in t. Then $L(E_1[start = lab^{t'_1}(u)]) = L(E_2[start = lab^{t'_2}(u)])$.*

PROOF. Let a^i and a^j be the label of u in t'_1 and t'_2, respectively.

If $|L(E_1[\text{start} = a^i])| = |L(E_2[\text{start} = a^j])| = 1$, the proof is trivial. We show that $L(E_1[\text{start} = a^i]) \subseteq L(E_2[\text{start} = a^j])$. The other inclusion follows by symmetry.

Towards a contradiction, assume that there exists a tree $t_0 \in L(E_1[\text{start} = a^i]) - L(E_2[\text{start} = a^j])$. As E_1 is reduced, there exists a tree T_0 in $L(E_1)$, such that

- t_0 is a subtree of T_0 at some node v; and,
- $\text{lab}^{T'_0}(v) = a^i$, where T'_0 is the unique typing of E_1 on T_0.

As $\text{lab}^{t'_1}(u) = a^i = \text{lab}^{T'_0}(v)$, the tree $t_3 = t[u \leftarrow t_0]$ is also in $L(E_1)$. As E_1 and E_2 are equivalent, t_3 is also in $L(E_2)$. Notice that u has the same ancestor-sibling-string in t and in $t_3 = t[u \leftarrow t_0]$. By Lemma 42, E_2 has ancestor-sibling-based typings, which implies that $\text{lab}^{t'_3}(u) = a^j$ for the unique typing t'_3 of E_2 on t_3. Therefore, $t_0 \in L(E_2[\text{start} = a^j])$, which leads to the desired contradiction. □

Let $E_{\min}$ be an EDTD which is obtained by applying the above minimization algorithm to an EDTD E over Σ. The next lemma states that every equivalent minimal restrained competition EDTD has an equal number of types as $E_{\min}$.

Lemma 44 *Let E_0 be a minimal restrained competition EDTD for $L(E_{min})$. Then, for every $a \in \Sigma$, we have that*

$$|\{a^i \in types(E_0) \mid name_{E_0}(a^i) = a\}| = |\{a^j \in types(E_{min}) \mid name_{E_{min}}(a^j) = a\}|.$$

PROOF. Fix an $a \in \Sigma$ and denote the sets $\{a^i \in \text{types}(E_0) \mid \text{name}_{E_0}(a^i) = a\}$ and $\{a^j \in \text{types}(E_{\min}) \mid \text{name}_{E_{\min}}(a^j) = a\}$ by $\text{Types}_0(a)$ and $\text{Types}_{\min}(a)$ respectively. We first show that $|\text{Types}_0(a)|$ cannot be larger than $|\text{Types}_{\min}(a)|$. Towards a contradiction, assume that $|\text{Types}_0(a)| > |\text{Types}_{\min}(a)|$. For every

$a^i \in \text{Types}_0(a)$, let t_i be an arbitrary tree such that a^i is a label of some node u_i the unique typing t'_{i,E_0} of E_0 on t_i. Also, let $t'_{i,E_{\min}}$ be the unique typing of $E_{\min}$ on t_i (hence, u_i is labeled with some element of $\text{Types}_{\min}(a)$ in $t'_{i,E_{\min}}$).

We now have $|\text{Types}_0(a)|$ typings $t'_{i,E_{\min}}$. Since $|\text{Types}_0(a)| > |\text{Types}_{\min}(a)|$ there must exist two different indices j and k such that

there exists an $a^\ell \in \text{Types}_{\min}(a)$ such that
the label of u_j in $t'_{j,E_{\min}}$ and the label of u_k in $t'_{k,E_{\min}}$ is a^ℓ.

From Lemma 43, it now follows that $L(E_0[\text{start} = a^j]) = L(E_{\min}[\text{start} = a^\ell]) = L(E_0[\text{start} = a^k])$. Therefore, replacing every a^k with a^j in E_0 results in an equivalent, strictly smaller restrained competition EDTD than E_0. This contradicts that E_0 is minimal.

The other direction can be proved completely analogously, with the roles of E_0 and $E_{\min}$ interchanged. Now the contradiction is that $E_{\min}$ cannot be the output of the minimization algorithm, as there still exist a^j and a^k in $\text{types}(E_{\min})$ for which $L(E_{\min}[\text{start} = a^j]) = L(E_{\min}[\text{start} = a^k])$. □

We argue that, for every minimal restrained competition EDTD E_0 accepting $L(E_{\min})$, there exists a bijection I from $\text{types}(E_{\min})$ to $\text{types}(E_0)$ such that $I(a^i)$ is the unique $a^j \in \text{types}(E_0)$ for which $L(E_0[\text{start} = a^i]) = L(E_{\min}[\text{start} = a^j])$. Due to Lemma 44, we know that every minimal restrained competition EDTD for $L(E_{\min})$ has the same number of types for each alphabet symbol. Hence, we only need to show that I is surjective, that is, for every $a^i \in \text{types}(E_0)$, there exists an $a^j \in \text{types}(E_{\min})$ for which $L(E_0[\text{start} = a^i]) = L(E_{\min}[\text{start} = a^j])$. The latter is immediate from Lemma 43.

Let b^k be an arbitrary symbol in $\text{types}(E_{\min})$. Let L_{b^k} and $L_{I(b^k)}$ denote the languages defined by the dFAs in the rules $b^k \to D_{b^k}$ in $\text{rules}(E_{\min})$ and $I(b^k) \to D_{I(b^k)}$ in $\text{rules}(E_0)$, respectively. Then, we have that $L_{b^k} = \mathbf{I}^{-1}(L_{I(b^k)})$ (where we denoted by $\mathbf{I}$ the homomorphic bijective extension of I to string languages). As minimal dFAs for a given regular language are unique up to isomorphisms, we have the following lemma:

Lemma 45 *Every minimal restrained competition EDTD E_0 for $L(E_{min})$ is isomorphic to E_{min}.*

The next lemma is immediate from the observation that, given a single-type EDTD, the minimization algorithm also returns a single-type EDTD. This is due to the fact that, in step (2)(a), the algorithm only overwrites *all* occurrences of a certain type with another type from the schema.

Lemma 46 MINIMIZATION *for single-type EDTDs is in* PTIME. *Moreover, minimal single-type EDTDs are unique up to isomorphism.*

Hence, Theorem 40 now follows from Lemma 45 and Lemma 46.

6 Conclusions

We have shown that the minimization problem is NP-complete for bottom-up deterministic unranked tree automata (UTAs) in which the string languages in the transition function are represented by dFAs (dUTAs). The source of this complexity is a minor amount of non-determinism that is still present in the manner how dUTAs are represented. Indeed, dUTAs still allow to represent regular languages over states by a *disjoint union of dFAs,* as exemplified in Section 3.1.

This raises the question of what a good notion for bottom-up determinism is for unranked tree automata. Therefore, we compare several notions of determinism for unranked tree automata in a second part of the article: deterministic parallel UTAs, which are defined independently in [9] and [27], deterministic stepwise tree automata [5], and deterministic ranked tree automata over the *first-child next-sibling* encoding. Among these three candidates, we feel that deterministic stepwise tree automata provide the most suited notion of bottom-up determinism for unranked tree languages. We base this on the following observations:

(1) In general, the deterministic stepwise tree automata provide the smallest minimal automata: they are generally quadratically smaller than deterministic parallel UTAs and exponentially smaller than deterministic ranked tree automata over the first-child next-sibling encoding (up to inversion).
(2) Stepwise tree automata have a direct connection to ranked tree automata through an encoding which is based on currying. This encoding allows to use the same (PTIME) minimization algorithm for tree automata over unranked trees than for traditional tree automata over binary trees. Moreover, a Myhill-Nerode theorem for unranked tree languages is immediate.
(3) The Myhill-Nerode theorem for deterministic stepwise tree automata uses a single, natural congruence relation and leads to (unique) minimal automata. To the best of our knowledge, none of the Myhill-Nerode inspired theorems for unranked tree languages that have been proven in the past (e.g. in [3,9]) fulfill both of these conditions.

In spite of the quadratical difference in minimal size, deterministic stepwise automata and deterministic parallel UTAs are very closely related. Essentially,

the differences between parallel UTAs and stepwise automata are that

(1) parallel UTAs use an *output function* to relate states of the internal DFAs to the states of the tree automaton; and
(2) parallel UTAs require the state sets of the internal DFAs to be *disjoint*.

While the first difference only has a minor effect on the size of minimal deterministic parallel UTAs, it is the second difference that causes them to be quadratically larger than stepwise automata.

In a third part of the paper, we investigated the minimization problem for single-type and restrained competition extended DTDs, which are abstractions of XML Schema and one-pass preorder typeable schemas, respectively. We showed that such extended DTDs can be minimized in polynomial time, and that a language has a unique minimal canonical model. Moreover, as the minimization algorithm preserves the single-type property of its input extended DTD, we also obtain that the above results hold for single-type extended DTDs.

Acknowledgments

We would like to thank Frank Neven and Thomas Schwentick for helpful discussions and comments on a previous version of the article. Frank Neven's comments have lead to significant improvements of the readability of the proof in Section 3.2. We also thank Mario Vöhl for suggestions that helped to improve Section 5.

References

[1] D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, November 1987.

[2] A. Balmin, Y. Papakonstantinou, and V. Vianu. Incremental validation of XML documents. *ACM Transactions on Database Systems*, 29(4):710–751, 2004.

[3] 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.

[4] J. Carme, A. Lemay, and J. Niehren. Learning node selecting tree transducers from completely annotated examples. In *International Colloquium on Grammatical Inference (ICGI 2004)*, pages 91–102, 2004.

[5] J. Carme, J. Niehren, and M. Tommasi. Querying unranked trees with stepwise tree automata. In *International Conference on Rewriting Techniques and Applications (RTA 2004)*, pages 105–118, 2004.

[6] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on http://www.grappa.univ-lille3.fr/tata, 2001.

[7] S.A. Cook. An observation on time-storage trade-off. *Journal of Computer and System Sciences*, 9(3):308–316, 1974.

[8] B. Courcelle. On recognizable sets and tree automata. In *Resolution of equations in algebraic structures*, pages 93–126, 1989.

[9] J. Cristau, C. Löding, and W. Thomas. Deterministic automata on unranked trees. In *Proceedings 15th International Symposium on Fundamentals of Computation Theory (FCT 2005)*, pages 68–72, 2005.

[10] M. Frick, M. Grohe, and C. Koch. Query evaluation on compressed trees (extended abstract). In *18th IEEE Symposium on Logic in Computer Science (LICS 2003)*, pages 188–197, 2003.

[11] E.M. Gold. Complexity of automaton identification from given data. *Inform. Control*, 37:302–320, 1978.

[12] G. Gottlob, C. Koch, R. Pichler, and L. Segoufin. The complexity of XPath query evaluation and XML typing. *Journal of the ACM*, 52(2):284–335, 2005.

[13] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 2001.

[14] T. Jiang and B. Ravikumar. Minimal NFA problems are hard. *SIAM Journal on Computing*, 22(6):1117–1141, 1993.

[15] D. Kozen. On the Myhill-Nerode theorem for trees. *Bulletin of the European Association for Theoretical Computer Science*, 147:170–173, 1992.

[16] A. Malcher. Minimizing finite automata is computationally hard. *Theoretical Computer Science*, 327(3):375–390, 2004.

[17] W. Martens and F. Neven. On the complexity of typechecking top-down XML transformations. *Theoretical Computer Science*, 336(1):153–180, 2005.

[18] W. Martens, F. Neven, T. Schwentick, and G. J. Bex. Expressiveness and complexity of XML Schema. *ACM Transactions on Database Systems*, 31(3), 2006. To appear.

[19] W. Martens and J. Niehren. Minimizing tree automata for unranked trees [extended abstract]. In *Proceedings of the Tenth International Symposium on Database Programming Languages (DBPL 2005)*, pages 233–247, 2005.

[20] M. Murata, D. Lee, M. Mani, and K. Kawaguchi. Taxonomy of XML schema languages using formal language theory. *ACM Transactions on Internet Technology*, 5(4):1–45, 2005.

[21] F. Neven. Automata theory for XML researchers. *SIGMOD Record*, 31(3), 2002.

[22] F. Neven and T. Schwentick. Expressive and efficient pattern languages for tree-structured data. In *Proceedings of the 19th Symposium on Principles of Database Systems (PODS 2000)*, pages 145–156, 2000.

[23] F. Neven and T. Schwentick. Query automata on finite trees. *Theoretical Computer Science*, 275:633–674, 2002.

[24] J. Oncina and P. Garcia. Inferring regular languages in polynomial update time. In *Pattern Recognition and Image Analysis*, pages 49–61, 1992.

[25] Y. Papakonstantinou and V. Vianu. DTD inference for views of XML data. In *Proceedings of the 19th Symposium on Principles of Database Systems (PODS 2000)*, pages 35–46. ACM Press, 2000.

[26] G. Pighizzini and J. Shallit. Unary language operations, state complexity and Jacobsthal's function. *International Journal of Foundations of Computer Science*, 13(1):145–159, 2002.

[27] S. Raeymaekers and M. Bruynooghe. Minimization of finite unranked tree automata. Manuscript, 2004.

[28] T. Schwentick. XPath query containment. *Sigmod Record*, 33(2):101–109, 2004.

[29] H. Seidl. Deciding equivalence of finite tree automata. *SIAM Journal on Computing*, 19(3):424–437, 1990.

[30] C.M. Sperberg-McQueen and H. Thompson. XML Schema. http://www.w3.org/XML/Schema, 2005.

[31] 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.

[32] L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time: Preliminary report. In *Fifth ACM Symposium on Theory of Computing (STOC 1973)*, pages 1–9. ACM, 1973.

[33] D. Suciu. Typechecking for semistructured data. In *Proceedings of the 8th Workshop on Data Bases and Programming Languages (DBPL 2001)*, pages 1–20, 2001.

[34] J. W. Thatcher. Characterizing derivation trees of context-free grammars through a generalization of automata theory. *Journal of Computer and System Sciences*, 1:317–322, 1967.

[35] 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.

[36] E. van der Vlist. *Relax NG*. O'Reilly, 2003.