# Deterministic Top-Down Tree Automata: Past, Present, and Future

Wim Martens[1]
Frank Neven[2]
Thomas Schwentick[1]

[1] University of Dortmund
Germany

[2] Hasselt University and Transnational University of Limburg
School for Information Technology
Belgium

**Abstract**

In strong contrast to their non-deterministic counterparts, deterministic top-down tree automata received little attention in the scientific literature. The aim of this article is to survey recent and less recent results and stipulate new research directions for top-down deterministic tree automata motivated by the advent of the XML data exchange format. In particular, we survey different ranked and unranked top-down tree automata models and discuss expressiveness, closure properties and the complexity of static analysis problems.

## 1 Introduction

The goal of this article is to survey some results concerning deterministic top-down tree automata motivated by purely formal language theoretic reasons (past) and by the advent of the data exchange format XML (present). Finally, we outline some new research directions (future).

**The Past.** Regular tree languages have been studied in depth ever since their introduction in the late sixties [10]. Just as for regular string languages, regular tree languages form a robust class admitting many closure properties and many equivalent formulations, the most prominent one in the form of tree automata. A striking difference with the string case where left-to-right equals right-to-left processing, is that top-down is no longer equivalent to bottom-up. In particular, top-down deterministic tree automata are strictly less expressive than their bottom-up counterparts and consequently form a strict subclass of the regular tree languages. Furthermore, deterministic top-down tree automata do not enjoy many of the important closure properties. For instance, they are neither closed under union nor under complement.

Several variants of deterministic top-down tree automata models have been introduced of which the one defined in [10, 7] is considered to be the standard one: the states assigned to the children of a node depend solely on the label and the state at the current node. We refer to these automata as 'blind' because they cannot see the label of the children when assigning states to them. A natural extension would therefore be to make automata 'sensing' by allowing them to see those labels. The latter model is more expressive than the former and both can be characterized by closure under a subtree exchange property. Using the latter property it becomes very easy to show that the models are neither closed under union nor under complement. The $l$-$r$-determinism for top-dowm tree automata introduced by Nivat and Podelski [17] and defining the homogeneous tree languages is strictly more expressive than blind automata and incomparable to sensing ones. Both blind and sensing tree automata allow for tractable static analysis: emptiness, containment and minimization are in PTIME.

**The Present.** XML, which stands for the eXtensible Markup Language, is a standard defined by W3C [4] for data exchange over the internet. From an abstract viewpoint, XML data or XML documents can be represented by finite labeled unranked trees where unranked means that there is no a priori bound on the number of child nodes a node can have. In a data exchange scenario not every XML document is allowed and the structure of XML documents is usually restricted to adhere to a specified schema. Many schema languages for XML exist of which the most prominent ones are DTD [4], XML Schema [20], and Relax NG [6]. In formal language theoretic terms, every schema defines an unranked tree language. This XML setting motivated Brüggemann-Klein, Murata, and Wood to develop a theory of unranked tree automata, an endeavor already initiated in the late sixties by Thatcher [21]. For deterministic top-down unranked tree automata there is again the difference between the blind and the sensing variant. Furthermore, as nodes can have arbitrarily many children it is natural to consider two variants of sensing automata. The first variant is an online one: given the state and the label of its parent, the state of a child only depends on its label and the labels of its left-siblings. The variant is called online as child states are assigned when processing the child string in one pass from left to right. In contrast, the offline variant first reads the complete child string and only then assigns states to all children. All three models can again be characterized in terms of closure under specific forms of subtree exchange. These properties can be used to show that blind, online, and offline sensing are increasingly more expressive and that neither of the models is closed under union and complement. Interestingly, online sensing top-down tree automata suffice to express all DTDs and XML Schema Definitions. Furthermore, they correspond precisely to the unranked regular tree languages admitting one-pass preorder typing [14]. In this context, typing means the assignment of the correct state to each node. So, online sensing deterministic top-down tree automata capture precisely the schemas which can be validated and typed in a one-pass fashion. A difference with the binary case is that minimization is NP-complete for offline sensing top-down automata, while it is in PTIME for online sensing top-down automata. Minimization for blind automata is in NP but the precise complexity is unknown.

**The Future.** From a theoretical point of view, there is a schema language superior to XML Schema: Relax NG is more expressiveness than XML Schema and it is closed under the Boolean operations. Nevertheless, XML Schema is the language endorsed by W3C and therefore supported by the major database vendors. It constitutes deterministic top-down processing as its basic validation mechanism. As mentioned before, XML Schema lacks the most basic closure properties. From the viewpoint of model management [1] or schema integration, especially the inability to express the union of two schemas is a serious defect. From a formal language theory perspective, Jurvanen, Potthof, and Thomas proposed regular frontier checks as a general extension of deterministic top-down automata [12]. In particular, the acceptance condition is determined by a regular string language $F$ over states added to the model. A tree is then accepted when the string formed by the states assigned to the frontier of the tree is in $F$. Although this formalism is expressive enough to define union and complement it is less convenient as an addition for a schema language. It would therefore be interesting to come up with a convenient top-down deterministic model closed under the Boolean operations. We discuss this and other future directions like optimization and automatic inference problems in the Conclusions.

**Outline.** The article is further organized as follows. In Section 2, we introduce the necessary notation. In Section 3 and 4, we discuss ranked and unranked deterministic top-down models, respectively. Finally, in Section 5, we consider regular frontier checks.

## 2 Preliminaries

### 2.1 An abstract notation for automata

We first explain the generic automata notation that we will use throughout the paper. For a finite set $S$, we denote by $|S|$ its number of elements. By $\Sigma$ we always denote a finite alphabet. We consider different types of data structures built from $\Sigma$ like strings, binary trees, or unranked trees. We write $D_\Sigma$ for the set of all data structures of the given type that can be built from $\Sigma$. For every $d \in D_\Sigma$, we will define a set $\text{Nodes}(d)$, a designated element $\text{root}(d) \in \text{Nodes}(d)$, and a designated set $\text{Frontier}(d) \subseteq \text{Nodes}(d)$. Here, $\text{root}(d)$ will be the root of a tree or the first symbol of a string; $\text{Frontier}(d)$ will be the set of leaves in a tree or the last symbol of a string.

To address automata in a uniform way for the different data structures, we first define them in abstract terms to instantiate them later operating on strings, trees, and unranked trees.

**Definition 2.1.** A *finite automaton* over $\Sigma$ is a tuple

$$A = (\mathrm{States}(A), \mathrm{Alphabet}(A), \mathrm{Rules}(A), \mathrm{Init}(A), \mathrm{Final}(A)),$$

where $\mathrm{States}(A)$ is a finite set of states, $\mathrm{Alphabet}(A) = \Sigma$ is the finite alphabet, $\mathrm{Rules}(A)$ is a finite set of transition rules, $\mathrm{Init}(A) \subseteq \mathrm{States}(A)$ is the set of initial states, and $\mathrm{Final}(A) \subseteq \mathrm{States}(A)$ is the set of final states.

The size of $A$, denoted by $|A|$, is a natural number, which by default will be the number of states of $A$ unless explicitly 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 $r : \mathrm{Nodes}(d) \to \mathrm{States}(A)$. For each kind of automaton, we will define when a run is *accepting*. Then, the language $L(A)$ of an automaton is the set of data structures $d$ that permit an accepting run.We call a finite automaton *unambiguous* if, for every $d$, there exists at most one accepting run of $A$ on $d$.

We consider the following static analysis problems:

- **Emptiness:** Given a finite automaton $A$, is $L(A) = \varnothing$?
- **Containment:** Given two finite automata $A$ and $B$, is $L(A) \subseteq L(B)$?
- **Minimization:** Given a finite automaton $A$ and integer $k$, does there exist an automaton $B$ (of the same class as $A$) such that $L(A) = L(B)$ and $|B| \leq k$?

In the remainder of the paper, we will use the letters $a, b, c, \ldots$ to range over alphabet symbols and we will use $p, q, \ldots$ to range over states.

## 2.2 Strings and Trees

By $\mathbb{N}_0$ we denote the set of nonnegative integers and by $\mathbb{N}$ the set of positive integers. We call $a \in \Sigma$ a $\Sigma$-*symbol*. A $\Sigma$-*string* (or simply *string*) $w \in \Sigma^*$ is a finite sequence $a_1 \cdots a_n$ of $\Sigma$-symbols. We denote the empty string by $\varepsilon$.

The set of *positions*, or *nodes*, of a $\Sigma$-string $w$ is $\mathrm{Nodes}(w) = \{1, \ldots, n\}$. The *root* of $w$ is $\mathrm{root}(w) = 1$ and the *frontier* of $w$ is $\mathrm{Frontier}(w) = \{n\}$. The *length* of $w$, denoted by $|w|$, is $n$. The label $a_i$ of node $i$ in $w$ is denoted by $\mathrm{lab}^w(i)$.

A *tree domain* $N$ is a non-empty, prefix-closed subset of $\mathbb{N}^*$ satisfying the following condition: if $ui \in N$ for $u \in \mathbb{N}^*$ and $i \in \mathbb{N}$, then $uj \in N$ for all $j$ with $1 \leq j \leq i$. An *unranked* $\Sigma$-*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 $i \in \mathbb{N}$ the *children* of $u$ (where $ui$ is the $i$th child). The *root* of a tree is $\text{root}(t) = \varepsilon$ and the *frontier* of a tree is its set of nodes with no children, that is, $\text{Frontier}(t) = \{u \mid u1 \notin \text{Nodes}(t)\}$. 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 $t = a(t_1 \cdots t_k)$. 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. The *depth* of a node $i_1 \cdots i_n \in \mathbb{N}^*$ 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 $\Sigma$-trees by $\mathcal{T}_\Sigma$. By $\text{subtree}^t(u)$ we denote the *subtree of $t$ rooted at $u$*. For two $\Sigma$-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 $t_2$. A *tree language* is a set of trees.

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

## 2.3 Finite String Automata

We instantiate our abstract notion of finite automata over strings:

**Definition 2.2.** A *finite string automaton (FSA)* over $\Sigma$ is a finite automaton over $\Sigma$ where $\text{Rules}(A)$ is a finite set of rules of the form $q_1 \xrightarrow{a} q_2$ with $q_1, q_2 \in \text{States}(A)$ and $a \in \text{Alphabet}(A)$.

A *run* of $A$ on a string $w \in \text{Alphabet}(A)^*$ is a mapping $r : \text{Nodes}(w) \to \text{States}(A)$ such that

(i) there exists $q_0 \in \text{Init}(A)$ with $q_0 \xrightarrow{a} r(1)$ in $\text{Rules}(A)$ for $\text{lab}^w(1) = a$; and,

(ii) for every $i = 1, \ldots, |w| - 1$, it holds that $r(i) \xrightarrow{a} r(i+1)$ in $\text{Rules}(A)$ where $\text{lab}^w(i+1) = a$.

A run $r$ is *accepting* if $r(|w|) \in \text{Final}(A)$. An FSA $A$ is *deterministic* if it satisfies the following two conditions, implying that no string permits more than one run by $A$:

(i) $\text{Init}(A)$ 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 DFSA be the class of deterministic finite string automata.

### 2.4 Exchange Properties for Tree Languages

We define several of the exchange properties for tree languages that we use in the following sections to characterize the expressive power of tree automata.

#### 2.4.1 Path-Closed Languages

A well-known characterization of tree languages recognizable by a class of top-down deterministic tree automata is the one of *path closed* languages by Virágh [23]. The *path language of a tree* $t$, denoted $\mathrm{Path}(t)$, is the set of strings

$$\mathrm{lab}(\varepsilon) i_1 \mathrm{lab}(i_1) \quad \cdots \quad i_n \mathrm{lab}(i_1 \cdots i_n),$$

for nodes $i_1, i_1 i_2, \ldots i_1 \cdots i_n$ in $\mathrm{Nodes}(t)$.[1] The *path langauge of a tree language* $L$, denoted $\mathrm{Path}(L)$, then is the union of the path languages of its trees, that is, $\mathrm{Path}(L) = \bigcup_{t \in L} \mathrm{Path}(t)$. The *path closure* of a tree language $L$ is defined as $\text{P-Closure}(L) = \{t \mid \mathrm{Path}(t) \subseteq \mathrm{Path}(L)\}$. Finally, a tree language $L$ is *path-closed* when $\text{P-Closure}(L) \subseteq \mathrm{Path}(L)$.

Nivat and Podelski argued that path-closed languages can also be characterized using the following subtree exchange property [17].[2] A regular tree language $L$ is path-closed if and only if, for every $t \in L$ and every node $u \in \mathrm{Nodes}(t)$,

$$\text{if } t[u \leftarrow a(t_1, \ldots, t_n)] \in L \text{ and } t[u \leftarrow a(s_1, \ldots, s_n)] \in L, \text{ then}$$
$$t[u \leftarrow a(t_1, \ldots, s_i, \ldots, t_n)] \in L \text{ for each } i = 1, \ldots, n.$$

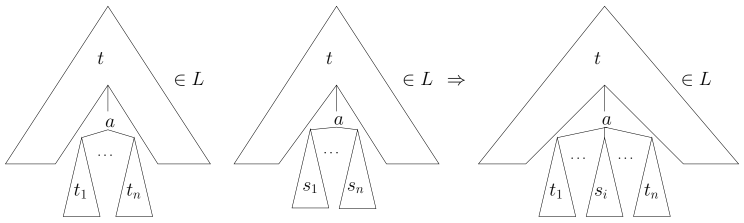
This subtree exchange closure for path-closed languages is illustrated in Figure 1. In the remainder of the article, when we say that a language is path-closed, we will always refer to this closure under the just mentioned exchange property.

#### 2.4.2 Guarded Subtree Exchange

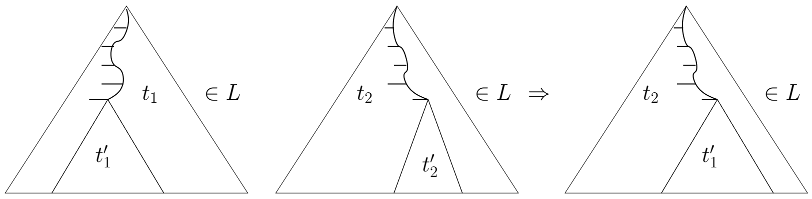
For a node $v = uk$ in a tree $t$ with $k \in \mathbb{N}$, we denote by $\text{l-sib-str}^t(v)$ the string formed by the label of the $v$ and the labels of its left siblings, that is, $\mathrm{lab}^t(u1) \cdots \mathrm{lab}^t(uk)$. By $\text{r-sib-str}^t(v)$ we denote the string formed by $v$ and its right siblings, that is, $\mathrm{lab}^t(uk) \cdots \mathrm{lab}^t(un)$, if $u$ has $n$ children. We define $\text{l-sib-str}^t(\varepsilon) = \text{r-sib-str}^t(\varepsilon) = \mathrm{lab}^t(\varepsilon)$. Let $v = i_1 i_2 \cdots i_\ell$ with

[1] We tacitly assume here that $\Sigma \cap \mathbb{N} = \varnothing$.

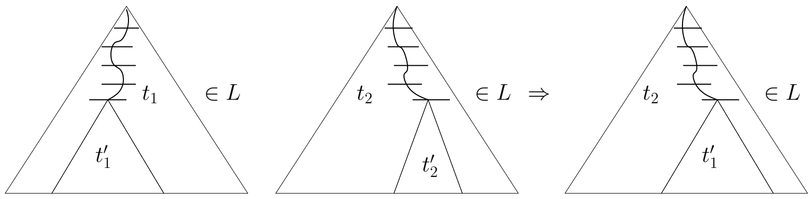
[2] Actually, Nivat and Podelski only considered path-closedness on ranked trees, but it is easy to see that the properties are also equivalent on unranked trees.



(a) Path-closed.



(b) Ancestor-left-sibling-closed.



(c) Spine-closed.

FIGURE 1. Various kinds of subtree exchange properties for tree languages.

$i_1, i_2, \ldots, i_\ell \in \mathbb{N}$. Let # and $\triangledown$ be two symbols not in $\Sigma$. By anc-l-sib-str$^t(v)$ we denote the ancestor-left-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. By spine$^t(v)$ we denote the ancestor-sibling-string

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

formed by concatenating the left-sibling-strings and right-sibling strings of all ancestors of $v$, starting from the root.

We say that a tree language $L$ is *ancestor-left-sibling-closed*[3] if whenever for two trees $t_1, t_2 \in L$ with nodes $u_1 \in \text{Nodes}(t_1)$ and $u_2 \in \text{Nodes}(t_2)$,

[3] This property was called "closure under ancestor-sibling-guarded subtree exchange" in [14].

$\text{anc-l-sib-str}^{t_1}(u_1) = \text{anc-l-sib-str}^{t_2}(u_2)$ implies $t_1[u_1 \leftarrow \text{subtree}^{t_2}(u_2)] \in L$. We say that $L$ is *spine-closed* if $\text{spine}^{t_1}(u_1) = \text{spine}^{t_2}(u_2)$ implies $t_1[u_1 \leftarrow \text{subtree}^{t_2}(u_2)] \in L$. The latter notions are illustrated in Figure 1.

## 3 Top-Down Automata on Binary Trees

As we consider in this section automata over binary trees, we take $\Sigma$ as a binary alphabet. We define two flavors of top-down determinism. The first is the traditional one, such as defined, for example, by Gecseg and Steinby [9] and in the on-line textbook TATA [7]. In brief, the label of the current symbol and the current state uniquely determine the states assigned to the children of the current symbol (Definition 3.1). The second notion of top-down determinism is slightly more expressive. Here, the states assigned to the children of the current node are determined by the current node's label, the state assigned to the current node, *and the labels of the children* (Definition 3.2). The latter notion of top-down determinism is reminiscent to the notion of "l-r-determinism" studied by Nivat and Podelski [17], and similar notions of top-down determinism on unranked trees have been studied by Cristau, Löding, and Thomas [8] and by Martens [13]. We refer to the first kind of automata as blind and to the second as sensing.

**Definition 3.1.** A *blind top-down finite tree automaton (BTA)* is a finite automaton $A$ such that $\text{Rules}(A)$ is a set of rules

$$(q, a) \rightarrow (q_1, q_2) \text{ or } (q, a) \rightarrow \varepsilon.$$

A *run* of $A$ on a binary $\Sigma$-tree $t$ is a mapping $r : \text{Nodes}(t) \rightarrow \text{States}(A)$ such that

(i) $r(\varepsilon) \in \text{Init}(A)$;

(ii) for each leaf node $u$ with label $a$, $(r(u), a) \rightarrow \varepsilon$ is in $\text{Rules}(A)$; and

(iii) for each non-leaf node $u$ with label $a$, $(r(u), a) \rightarrow (r(u1), r(u2))$ is in $\text{Rules}(A)$.

If a run exists, it is *accepting.* We say that a BTA is *(top-down) deterministic* if $\text{Init}(A)$ is a singleton and no two of its rules have the same left-hand sides.

**Definition 3.2.** A *sensing top-down finite tree automaton (STA)* is a finite automaton $A$ such that $\text{Rules}(A)$ is a set rules of the form

$$a \rightarrow q \quad \text{ or } \quad q(a_1, a_2) \rightarrow (q_1, q_2).$$

For an STA $A$, we have that $\text{Init}(A) = \{q \mid a \rightarrow q \in \text{Rules}(A)\}$. A *run* of $A$ on a binary $\Sigma$-tree $t$ is a mapping $r : \text{Nodes}(t) \rightarrow \text{States}(A)$ such that

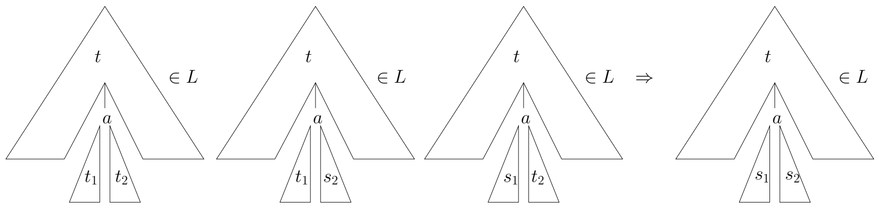


FIGURE 2. Closure property for homogeneous tree languages.

(i) if $r(\varepsilon) = q$ and $\text{lab}(\varepsilon) = a$ then there is a rule $a \to q \in \text{Rules}(A)$, and

(ii) for each non-frontier node $u$, if $r(u) = q$, $\text{lab}(u1) = a_1$, and $\text{lab}(u2) = a_2$, then there is a rule $q(a_1, a_2) \to (r(u1), r(u2))$ in $\text{Rules}(A)$.

The run is *accepting* if, for each leaf node $u$, $r(u) \in \text{Final}(A)$. We say that an STA is *deterministic* if no two of its rules have the same left-hand sides.

## 3.1 Relative Expressive Power

It is well-known that top-down automata cannot recognize all regular tree languages. In this section, we compare several forms of top-down determinism that have been investigated with respect to their expressive power.

### 3.1.1 Homogeneous Languages

Nivat and Podelski defined a notion of top-down determinism that they called *l-r-determinism*. This form of determinism will not be treated very deeply in this article, as it does not correspond to the order in which one would like to process trees in an XML context. We use their characterization in terms of closure under subtree exchange to formally argue this. Nivat and Podelski define a BTA $A$ to be l-r-deterministic if whenever $(q, a) \to (q_1, q_2)$ and $(q, a) \to (q_1', q_2')$ is in $\text{Rules}(A)$ then

- $q_1 \neq q_1'$ implies that $L(A[q_2]) \cup L(A[q_2']) = \varnothing$ and
- $q_2 \neq q_2'$ implies that $L(A[q_1]) \cup L(A[q_1']) = \varnothing$.

Here, for $q = q_1, q_2, q_1', q_2'$, $A[q]$ denotes automaton $A$ in which $\text{Init}(A) = \{q\}$. We will, however, focus on a characterization of the languages accepted by l-r-deterministic tree automata which is, for our purpose, more workable.

A regular tree language $L$ is *homogeneous* if, whenever $t[u \leftarrow a(t_1, t_2)] \in L$, $t[u \leftarrow a(s_1, t_2)] \in L$, and $t[u \leftarrow a(t_1, s_2)] \in L$, then also $t[u \leftarrow a(s_1, s_2)] \in L$. This closure under subtree exchange is illustrated in Figure 2.

### 3.1.2 The Characterization

We characterize the expressiveness of the tree automata models by the closure properties introduced in Section 2.4.

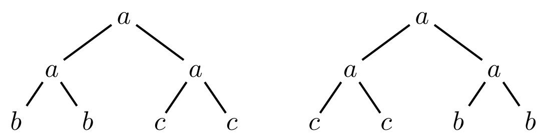


FIGURE 3. A homogeneous language that is not spine-closed.

**Theorem 3.3 (Characterization Theorem).** A regular tree language $L$ is recognizable by

(1) a deterministic BTA if and only if $L$ is path-closed.

(2) an l-r-deterministic tree automaton if and only if $L$ is homogeneous.

(3) a deterministic STA if and only if $L$ is spine-closed.

Theorem 3.3(1) is known from, e.g., Virágh [23] and from Gecseg and Steinby [10]. Theorem 3.3(2) is Theorem 2 in the work by Nivat and Podelski [17]. Finally, Theorem 3.3(3) is proved by Cristau, Löding, and Thomas [8] and by Martens [13] for more general unranked tree automata with this form of top-down determinism. It should be noted that Cristau et al. did not explicitly use a subtree exchange property for spine-closedness but an equivalent closure property that considers the spine language of a tree (as in the original definition of path-closedness).

**Corollary 3.4.** (1) l-r-deterministic tree automata are strictly more expressive than deterministic BTAs.

(2) Deterministic STAs are strictly more expressive than deterministic BTAs.

(3) Deterministic STAs and l-r-deterministic tree automata are incomparable w.r.t. expressive power.

*Proof.* (1) It is easy to see that every path-closed language is homogeneous. Furthermore, the language $\{a(b,b), a(c,c)\}$ is homogeneous but not path-closed.

(2) It is easy to see that every path-closed language is also spine-closed. Furthermore, the language $\{a(b,b), a(c,c)\}$ is spine-closed but is not path-closed.

(3) The language $\{a\big(a(b,b), a(c,c)\big), a\big(a(c,c), a(b,b)\big)\}$ is homogeneous but not spine-closed (see also Figure 3). The language $\{a(b,b), a(b,c), a(c,b)\}$ is spine-closed but not homogeneous. Q.E.D.

### 3.1.3 L-R-Determinism Versus Top-Down State Assignment

Figure 3 depicts a finite language $L$ which is homogeneous but not spine-closed. So, $L$ can be recognized by an l-r-deterministic tree automaton but not by a deterministic STA.

One easily obtains infinite languages with this property. Indeed, let $L_b$ and $L_c$ be the set of trees in which every internal node is labeled $a$ and every leaf is labeled $b$ and $c$, respectively. The language $L_{bc}$ now consists of all trees $a(t_b, t_c)$ and $a(t_c, t_b)$ for which $t_b \in L_b$ and $t_c \in L_c$. Clearly, $L_{bc}$ is homogeneous.

We now want to argue informally that, for any tree automaton $A$ recognizing $L_{bc}$, the state that $A$ assigns to each of the two children of the root in an accepting run cannot be determined without looking *arbitrarily deep* into at least one subtree of the root. In other words, this means that there is at least one child $u$ of the root such that $A$ needs to investigate the subtree rooted at $u$ before assigning a state to $u$. This is something what is not commonly associated with "top-down determinism".

Let $A$ be a tree automaton that recognizes the language $L_{bc}$. Let $n$ be an arbitrarily large natural number and let $a(t_b, t_c)$ be a tree in $L_{bc}$ such that every path from root to leaf in $t_b$ and $t_c$ has length at least $n + 1$. This way, $t_b$ and $t_c$ are identical up to depth $n$. Towards a contradiction, suppose that $A$ does not investigate $t_b$ or $t_c$ arbitrarily deep, i.e., not up to depth $n$, before assigning a state to the root of $t_b$ (the argument for $t_c$ is the same). More formally, assume that the state $A$ assigns to the root of $t_b$ is functionally determined by the structure of $t_b$ and $t_c$ up to depth at most $n - 1$. Let $r_1$ be an accepting run of $A$ on $a(t_b, t_c)$ and let $r_2$ be an accepting run of $A$ on $a(t_c, t_b)$. As $A$ does not investigate $t_b$ or $t_c$ arbitrarily deep, $r_1$ assigns the sames state to the root of $t_b$ in $a(t_b, t_c)$ as $r_2$ assigns to the root of $t_c$ in $a(t_c, t_b)$. As $A$ is a tree automaton, it is now easy to see that $a(t_c, t_c)$ is also in $L(A)$, with the accepting run that behaves as $r_2$ on the left copy of $t_c$ and as $r_1$ on the right copy of $t_c$. This contradicts that $A$ accepts $L_{bc}$.

Therefore, our focus in the remainder of the article will be on deterministic BTAs and deterministic STAs, rather than l-r-deterministic tree automata.

## 3.2 Closure Properties

The characterization theorem can easily be used to show that deterministic top-down tree automata are not closed under complement and union.

**Theorem 3.5.** (1) Deterministic BTAs and deterministic STAs are closed under intersection.

(2) Deterministic BTAs and deterministic STAs are not closed under complement or union.

*Proof.* (1) This follows immediately from the standard product construction for tree automata. One merely has to observe that the intersection construction preserves the determinism constraint for BTAs and STAs.
(2) These results can be proved quite directly from the characterizations in Theorem 3.3. Indeed, let $L_b$ (resp., $L_c$) be the tree language over alphabet $\{a, b, c\}$ in which every internal node (i.e., with two children) is labeled $a$ and every leaf is labeled $b$ (resp., $c$). The languages $L_b$ and $L_c$ are easily seen to be recognizable by deterministic BTAs.

On the other hand, the union $L_b \cup L_c$, the set of all trees in which every internal node is labeled $a$ and either all leaves are labeled $b$ or all leaves are labeled $c$ is *not* spine-closed. Hence, $L_b \cup L_c$ is not recognizable by a deterministic STA, which means that deterministic BTAs and deterministic STAs are not closed under union. From closure under intersection and non-closure under union we can readily conclude non-closure under complement.

Q.E.D.

## 3.3 Static Analysis

In this section, we will prove the following Theorem:

**Theorem 3.6.** (1) Emptiness is in PTIME for BTAs and STAs.

(2) Containment is in PTIME for deterministic BTAs and deterministic STAs.

(3) Minimization is in PTIME for deterministic BTAs and deterministic STAs.

*Proof.* (1) It is well-known that emptiness is in PTIME for (non-deterministic bottom-up) tree automata in general [7]. Therefore, emptiness is also in PTIME for deterministic BTAs and deterministic STAs.
(2) It is easy to see that deterministic BTAs and deterministic STAs and intersections thereof are in fact unambiguous tree automata. The result now follows from the work by Seidl, who proved that equivalence of unambiguous tree automata is in PTIME [18].
(3) For deterministic BTAs, this follows from the work by Gecseg and Steinby [9]. Although their work does not explicitly concern complexity, they prove that minimization for deterministic BTAs can be polynomially reduced to equivalence/containment for deterministic BTAs. As containment for deterministic BTAs is in PTIME by part (2), we also have that minimization is in PTIME.

To explain their algorithm, we start by discussing a minor optimization matter for tree automata. For an automaton $A$ and $q \in \text{States}(A)$ we denote by $A[q]$ the language accepted by $A$ when $\text{Init}(A) = \{q\}$.[4] We say that $q$ is *reachable* in $A$ if one of the following holds:

[4] If $A$ is an STA, we require in addition that every rule $a \to p$ is replaced by $a \to q$.

(1) *Reduce* $A$, that is,

(a) remove all states $q$ from $A$ for which $L(A[q]) = \varnothing$; and then

(b) remove all states $q$ from $A$ which are not reachable from $\text{Init}(A)$.

(2) Test, for each $p \neq q$ in $\text{States}(A)$, whether $L(A[p]) = L(A[q])$.
If $L(A[p]) = L(A[q])$, then

(a) replace all occurrences of $p$ in the definition of $A$ by $q$ and

(b) remove $p$ from $A$.

FIGURE 4. The Minimization Algorithm.

- $q \in \text{Init}(A)$ or
- $p$ is reachable and there is a rule of the form $(p, a) \to (q_1, q_2)$ or $p(a_1, a_2) \to (q_1, q_2)$ in $\text{Rules}(A)$, where $q = q_1$ or $q = q_2$.

We now say that $A$ is *reduced* if, every state $q$ is *useful*, that is, $q$ is reachable and $L(A[q]) \neq \varnothing$. Algorithmically, one would convert a tree automaton into a reduced tree automaton by first removing all the states $q$ for which $L(A[q]) = \varnothing$ and then removing all the states that are not reachable. The order in which these two steps are performed is important, as the other order does not necessarily produce a reduced automaton.

The following observation states that a state is useful if and only if it can be used in some accepting run of the automaton.

**Observation 3.7.** Let $A$ be a tree automaton and $q \in \text{States}(A)$. Then, $q$ is useful if and only if there exists a tree $t \in L(A)$, an accepting run $r$ of $A$ on $t$, and a node $u \in \text{Nodes}(t)$ such that $r(u) = q$.

The algorithm of Gecseg and Steinby is now informally presented in Figure 4.

Interestingly, for deterministic STAs, it seems that one can likewise use the algorithm of Figure 4 for minimization. It only has to be shown that, given a deterministic STA, the algorithm returns a minimal deterministic STA. Thereto, let $A_{\min}$ be the automaton obtained by applying the above minimization algorithm on a deterministic STA $A$. Formally, we need to prove that

(a) $A_{\min}$ is a deterministic STA;

(b) $L(A_{\min}) = L(A)$; and that

(c) the number of states of $A_{\min}$ is indeed minimal.

To show (a), observe that, in step (1) of the algorithm, we only remove states. Hence, no non-determinism is introduced in step (1). In step (2), non-determinism can be introduced by overwriting occurrences of $p$ with $q$. However, the following observation, which is easy to show by contraposition, proves that this non-determinism is removed further on in the algorithm.

**Observation 3.8.** Let $p$ and $q$ be two states such that $L(A[p]) = L(A[q])$ and let $p(a_1, a_2) \to (p_1, p_2)$ and $q(a_1, a_2) \to (q_1, q_2)$ be two transition rules of $A$. Then $L(A[p_1]) = L(A[q_1])$ and $L(A[p_2]) = L(A[q_2])$.

To show (b), observe that, in step (1), we only remove states that cannot be used in a successful run of $A$ (Observation 3.7). Hence, this does not alter the language accepted by $A$. In step (2), we replace states $p$ in $A$ with states $q$ that define the same language. The following observation is easy to prove:

**Observation 3.9.** Let $p$ and $q$ be two states such that $L(A[p]) = L(A[q])$. Let $A'$ be obtained from $A$ by replacing all occurrences of $p$ in the definition of $A$ by $q$, and by removing $q$. Then $L(A) = L(A')$.

It remains to show (c), which is a bit more involved. First, we introduce the following concept. We say that a finite tree automaton $A$ over $\Sigma$ *has spine-based runs* if there is a (partial) function

$$f : (\Sigma \cup \{\#, \triangledown\})^* \to \text{States}(A)$$

such that, for each tree $t \in L(A)$, for each node $v \in \text{Nodes}(t)$, and for each accepting run $r$ of $A$ on $t$, we have that

$$r(v) = f(\text{spine}^t(v)).$$

**Observation 3.10.** Every deterministic STA has spine-based runs.

*Proof.* Let $A$ be a deterministic STA. We assume w.l.o.g. that $A$ is reduced. We define the function $f : (\Sigma \cup \{\#, \triangledown\})^* \to \text{States}(A)$ inductively as follows: for each $a \in \Sigma$, $f(a\triangledown a) = q$, for the unique $q$ such that $a \to q$ is a rule in $A$. Further, for every string $w_0\#w_1a\triangledown aw_2$ with $w_0 \in (\Sigma \cup \{\#, \triangledown\})^*$, $w_1, w_2 \in \Sigma \cup \{\varepsilon\}$, and $a \in \Sigma$, we define $f(w_0\#w_1a\triangledown aw_2) = q$ where $f(w_0) = p$ and $q$ is the unique state such that the following holds. If $w_1 = \varepsilon$, $q$ is the unique state such that $p(a, w_2) \to (q, q') \in \text{Rules}(A)$, and if $w_2 = \varepsilon$, then $q$ is the unique state such that $p(w_1, a) \to (q', q) \in \text{Rules}(A)$. As $A$ is a reduced deterministic STA, $f$ is well-defined and induces a spine-based run. Q.E.D.

**Observation 3.11.** Let $A_1$ and $A_2$ be equivalent deterministic STAs and let $t \in L(A_1) = L(A_2)$. Let $r_1$ and $r_2$ be the unique runs of $A_1$ and $A_2$ on $t$, respectively, and let $u$ be a node in $t$. Then $L(A_1[r_1(u)]) = L(A_2[r_2(u)])$.

*Proof.* Let $p$ and $q$ be $r_1(u)$ and $r_2(u)$, respectively. If $|L(A_1[p])| = |L(A_2[q])| = 1$, the proof is trivial. We show that $L(A_1[p]) \subseteq L(A_2[q])$. The other inclusion follows by symmetry.

Towards a contradiction, assume that there exists a tree $t_0 \in L(A_1[p]) - L(A_2[q])$. As $A_1$ is reduced, there exists a tree $T_0$ in $L(A_1)$, such that

- $t_0$ is a subtree of $T_0$ at some node $v$; and,
- $r'_1(v) = p$, where $r'_1$ is the unique run of $A_1$ on $T_0$.

As $r_1(u) = p = r'_1(v)$, the tree $t_3 = t[u \leftarrow t_0]$ is also in $L(A_1)$. As $A_1$ and $A_2$ are equivalent, $t_3$ is also in $L(A_2)$. Notice that $u$ has the same spine in $t$ and in $t_3 = t[u \leftarrow t_0]$. By Observation 3.10, $A_2$ has spine-based runs, which implies that $r'_2(u) = q$ for the unique run $r'_2$ of $A_2$ on $t_3$. Therefore, $t_0 \in L(A_2[q])$, which leads to the desired contradiction. Q.E.D.

The next observation states that every equivalent minimal deterministic STA is equally large as $A_{\min}$.

**Observation 3.12.** If $A_0$ is a minimal deterministic STA for $L(A_{\min})$, then $|A_0| = |A_{\min}|$.

*Proof.* As $A_0$ is minimal, we know that $A_0$ is reduced and that $|A_0| \leq |A_{\min}|$. As $A_{\min}$ is the output of the minimization algorithm, $A_{\min}$ is reduced as well.

We only have to prove that $|A_{\min}| \leq |A_0|$. Towards a contradiction, assume that $|\text{States}(A_{\min}))| > |\text{States}(A_0)|$. For every state $q \in \text{States}(A_{\min})$, let $t^q_{\min} \in L(A_{\min})$ be a tree and $u^q_{\min} \in \text{Nodes}(t^q_{\min})$ such that $r^q_{\min}(u^q_{\min}) = q$ for the unique accepting run $r^q_{\min}$ of $A_{\min}$ on $t^q_{\min}$. Moreover, let, for every such $t^q_{\min}$, $r^q_0$ be the unique accepting run $r^q_0$ of $A_0$ on $t^q_{\min}$.

According to the Pigeon Hole Principle, there exist two states $p \neq q \in \text{States}(A_{\min})$ such that $r^p_0(u^p_{\min}) = r^q_0(u^q_{\min}) = p_0$, for some $p_0 \in \text{States}(A_0)$. From Observation 3.11, it now follows that $L(A_{\min}[p]) = L(A_0[p_0]) = L(A_{\min}[q])$. This contradicts that $A_{\min}$ is the output of the minimization algorithm, as there still exist two states for which step (2) must be performed. Q.E.D.

This concludes the proof of Theorem 3.6(3). Q.E.D.

## 4 Top-Down Automata on Unranked Trees

The definition of unranked tree automata dates back to the work of Thatcher [21]. Unranked tree automata use $\mathcal{T}_\Sigma$ (that is, unranked $\Sigma$-trees) as their data structure. For convenience, we sometimes abbreviate "unranked tree automaton" by UTA in this section. We start by defining blind top-down deterministic unranked tree automata, which generalize the determinism in

BTAs to unranked trees. Blind top-down deterministic unranked automata are, e.g., defined in [5] under the name of *top-down deterministic* automata.

**Definition 4.1.** A *blind top-down deterministic unranked tree automaton (BUTA)* over $\Sigma$ is a finite automaton $A$ over $\Sigma$ in which Rules($A$) is a set of rules of the form

$$a \to p \qquad \text{or} \qquad (q, a) \to B$$

such that $\text{Init}(A) = \{p \mid a \to p \in \text{Rules}(A)\}$ is a singleton and $B$ is a deterministic FSA over States($A$) with the property that, for each $i \in \mathbb{N}$, $L(B)$ contains at most one string of length $i$. Furthermore, for each $q \in$ States($A$) and $a \in$ Alphabet($A$), Rules($A$) contains at most one rule of the form $(q, a) \to B$.

A *run* of $A$ on a tree $t$ is a labeling $r : \text{Nodes}(t) \to \text{States}(A)$ such that

- if $\text{lab}(\varepsilon) = a$ and $r(\varepsilon) = q$ then $a \to q \in \text{Rules}(A)$ and,
- for every node $u \in \text{Nodes}(t)$ such that $\text{lab}(u) = a$, $r(u) = q$, and $u$ has $n$ children, there is a rule $(q, a) \to B$ such that $B$ accepts $r(u1) \cdots r(un)$.

Notice that, in the second bullet, the criterion that $u$ is a leaf reduces to $\varepsilon \in L(B)$. Therefore, each run that satisfies the above conditions is *accepting.*

Notice that the regular languages defined by the above $B$s are very restricted. Indeed, as pointed out in [16], Shallit [19] has shown that such regular languages are finite unions of regular expressions of the form $xy^*z$ where $x, y, z \in \Sigma^*$.

Just as in the ranked case, blind top-down determinism is the most widely accepted form of top-down determinism. However, in a context such as XML, blind top-down determinism is not very useful as its expressiveness is very limited. We therefore also investigate 'sensing' extensions that can read labels of child nodes before assigning them states.

The following definition is the generalization of determinism for STAs. In a similar effort to generalize determinism for STAs to unranked trees, Cristau et al. [8] and Martens [13] define models with the same expressive power as this one.

**Definition 4.2.** An *offline sensing top-down deterministic unranked tree automaton (offline SUTA)* is a finite automaton $A$ in which Rules($A$) is a set of rules of the form

$$a \to p \qquad \text{or} \qquad q \to B_q,$$

where the automata $B_q$ are FSAs over $\Sigma$ and use the states of $A$ as their state set. That is, $\text{States}(B_q) = \text{States}(A)$. Furthermore, all the $B_q$ have same the final states and the same transition rules, that is, for all $q_1, q_2 \in \text{States}(A)$, $\text{Final}(B_{q_1}) = \text{Final}((B_{q_2})$ and $\text{Rules}(B_{q_1}) = \text{Rules}(B_{q_2})$. In short, the only difference between the automata $B_q$ is their choice in initial states.[5] Furthermore,

- for each $a \in \text{Alphabet}(A)$ there is at most one rule of the form $a \to p$,
- for each $q \in \text{States}(A)$, there is at most one rule $q \to B_q$, and
- for each rule $q \to B_q$, $B_q$ is an *unambiguous FSA*.

We define $\text{Init}(A)$ to be $\{p \mid a \to p \in \text{Rules}(A)\}$ and we require that $\text{Init}(A) \subseteq \text{Final}(B_q)$, for each state $q$.

A *run* $r$ of $A$ on a tree $t$ is a labeling $r : \text{Nodes}(t) \to \text{States}(A)$ such that

- if $\text{lab}(\varepsilon) = a$ and $r(\varepsilon) = q$ then $a \to q \in \text{Rules}(A)$ and,
- for every node $u \in \text{Nodes}(t)$ such that $\text{lab}(u) = a$, $r(u) = q$, and $u$ has $n$ children, there is a rule $q \to B_q$ such that $r(u1) \cdots r(un)$ is an accepting run of $B_q$ on $\text{lab}(u1) \cdots \text{lab}(un)$.

As with BUTAs, the criterion that $u$ is a leaf reduces to $\varepsilon \in L(B)$ in the second bullet. Therefore, each run that satisfies the above conditions is *accepting.*

The restriction to unambiguous FSAs actually ensures that the complete child string can be read prior to the assignment of states. We note that the above mentioned work [8, 13], where "sensing top-down determinism" is simply called "top-down determinism", employs slightly more involved but equivalent definitions in terms of expressive power.

In Section 4.2, we will see that, in contrast to the ranked case, offline sensing top-down determinism is in fact too powerful for efficient static analysis. In particular, minimization will turn out to be NP-hard for offline sensing deterministic automata. We therefore discuss *online sensing*, an intermediate form of top-down determinism which is also known under the name of *restrained competition* for *extended DTDs*.[6] This restriction will turn out to be more expressive than blind top-down determinism, while retaining the desirable complexities for static analysis.

[5] A similar sharing of states is used in *stepwise tree automata*, which were used for defining a clean notion of *bottom-up determinism* for unranked tree automata [15].

[6] Extended DTDs or EDTDs are a grammar-based alternative to tree automata which have been investigated in the context of XML schema languages [13, 14].

**Definition 4.3.** An *online sensing top-down deterministic unranked tree automaton (online SUTA)* is an offline SUTA with the difference that, for each rule $q \to B_q$, $B_q$ is a *deterministic FSA*.

The restriction to deterministic FSAs ensures that states have to be assigned to child nodes when processing them from left to right.

## 4.1 Relative Expressive Power

Again, we characterize the expressiveness of the formalisms in terms of subtree exchange properties.

**Theorem 4.4.** An (unranked) regular tree language $L$ is recognizable by

1. a BUTA if and only if $L$ is path-closed.
2. an online SUTA if and only if $L$ is ancestor-sibling-closed.
3. an offline SUTA if and only if $L$ is spine-closed.

The proof of Theorem 4.4(1) is analogous to the ranked case. Theorem 4.4(2) and Theorem 4.4(3) are proved by Martens et al. [13, 14].

The next corollary then immediately follows:

**Corollary 4.5.**

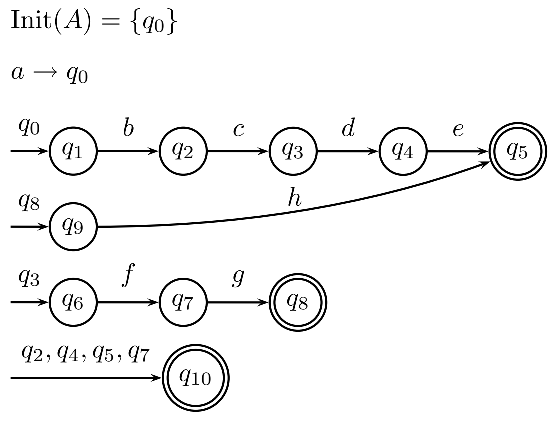
1. BUTAs are strictly less expressive than online SUTAs.
2. Online SUTAs are strictly less expressive than offline SUTAs.

## 4.2 Static Analysis

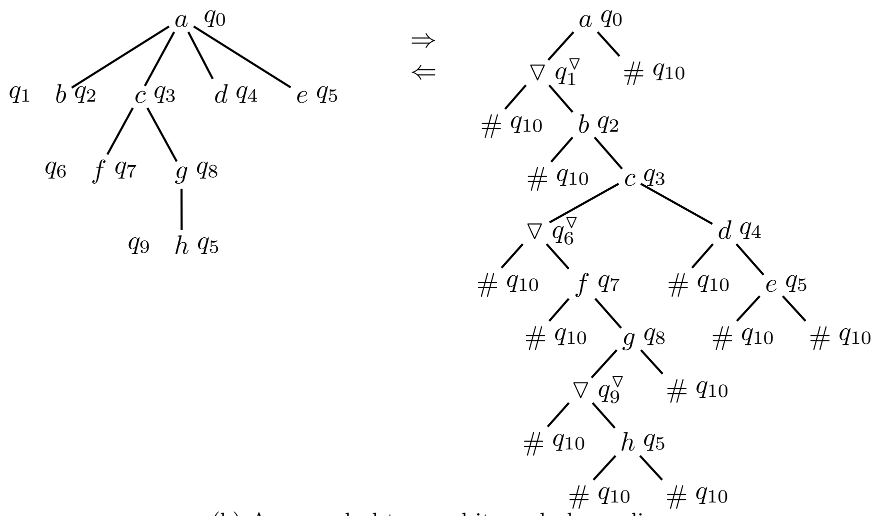
**Theorem 4.6.**

1. Emptiness is in PTIME for BUTAs, online SUTAs and offline SUTAs.
2. Containment is in PTIME for BUTAs, online SUTAs and offline SUTAs.
3. Minimization is in PTIME for online SUTAs.
4. Minimization is NP-complete for offline SUTAs.

*Proof.* (1) This follows from the result that emptiness is in PTIME for non-deterministic unranked tree automata. (See, e.g., [13].)
(2) This follows from PTIME containment for *unambiguous* (ranked) tree automata [18]. For example, when translating an offline SUTA to a ranked tree automaton through the well-known *first-child next-sibling encoding*, one obtains an unambiguous ranked tree automaton. Containment of the unranked tree automata can then be decided by testing containment for the unambiguous ranked automata.



(a) Automaton $A$ accepting the tree in Figure 5(b).



(b) An unranked tree and its ranked encoding.

Figure 5. Encoding of unranked to binary trees (and back) that links deterministic STAs to online SUTAs. Letters $a, \ldots, h$ represent alphabet symbols and $\{q_0, \ldots, q_{10}\}$ represent states of an accepting run.

(3) We can reduce to Theorem 3.6(3) by means of the unranked-versus-ranked encoding enc and decoding dec illustrated in Figure 5. We explain intuitively how a run of an online SUTA $A$ for $L$ translates to a run of a deterministic STA enc$(A)$ for enc$(L)$. We assume w.l.o.g. that $A$ is reduced. Assignment of initial states to the root of the trees is the same for both automata. Furthermore, the transition rules translate as follows. For each $q \in \text{States}(A)$ and $a \in \text{Alphabet}(A)$, Rules(enc$(A)$) contains

- $a \to q$ if $a \to q$ in Rules$(A)$;
- $q(\triangledown, \#) \to (p^{\triangledown}, q_{\text{leaf}})$ if $\text{Init}(B_q) = \{p\}$ and $q \in \text{Final}(B_q)$;
- $q(\triangledown, a) \to (p^{\triangledown}, q')$ if $\text{Init}(B_q) = \{p\}$ and $q \xrightarrow{a} q' \in \text{Rules}(B_q)$;
- $q(\#, a) \to (q_{\text{leaf}}, q')$ if $B_q$ accepts $\varepsilon$ and $q \xrightarrow{a} q' \in \text{Rules}(B_q)$;
- $q^{\triangledown}(\#, a) \to (q_{\text{leaf}}, q')$ if $q \xrightarrow{a} q' \in \text{Rules}(B_q)$; and
- $q(\#, \#) \to (q_{\text{leaf}}, q_{\text{leaf}})$ if $B_q$ accepts $\varepsilon$ and $q$ is a final state in $B_q$.

Here, $q_{\text{leaf}}$ is a new state not occurring in States$(A)$. The states $q^{\triangledown}$ are copies of states $q$ in $A$ that can only be assigned to the $\triangledown$-labeled nodes in the encoding. The encoded automaton always assigns $q_{\text{leaf}}$ to leaf symbols. Hence, $\text{Final}(\text{enc}(A)) = q_{\text{leaf}}$. Figure 5 illustrates an automaton $A$, an accepting run of $A$ on a tree $t$, and an accepting run of enc$(A)$ on enc$(t)$.

It is easy to see that this encoding preserves determinism. The decoding, however, would not preserve determinism in general, as the initial states Init$(B_q)$ might not be unique. It can be shown, however, that if $L$ is ancestor-sibling closed, the decoding of a minimal deterministic STA for enc$(L)$ is always deterministic.

In order to give the relation between the minimal sizes of $A$ and enc$(A)$, we need a few parameters. We call a state $q$ of $A$ a *sink state*, when no rules of the form $q \to B$ occur in $A$ and no $B$ has a rule $q \xrightarrow{a} q'$ for some $a$. For example, the state $q_{10}$ in Figure 5 is such a sink state. We define $\text{sink}(A) = 0$ if $A$ has such a sink state and $\text{sink}(A) = 1$ otherwise. Furthermore, let trans-init$(A)$ be the number of states $p$ such that $\{p\} = \text{Init}(B_q)$ for some $q$ and $p$ has an incoming transition.

**Observation 4.7.** There exists an online SUTA of size $k$ for $L(A)$ if and only if there exists a deterministic STA of size $k + \text{sink}(A) + \text{trans-init}(A)$ for $L(\text{enc}(A))$.

The reasons for the difference in sizes concerning the sink state and the trans-init states are as follows. If $A$ contains a sink state $q$, then enc$(A)$ could use this sink state instead of $q_{\text{leaf}}$ to label all the #-leaves in the

encoding. Furthermore, in the encoding, each $\triangledown$ node is labeled by a copy $q^{\triangledown}$ of a state $q$, which introduces extra states for $\text{enc}(A)$. However, if $q$ contains an incoming transition in $A$ (and $A$ is reduced), then both $q$ and $q^{\triangledown}$ appear in the minimal automaton for $L(\text{enc}(A))$.
(4) We first argue that minimization for offline SUTAs is in NP. To this end, observe that, given an offline SUTA $A$ and an integer $k$, an NP algorithm can guess an offline SUTA $B$ of size at most $k$ and test in PTIME (according to Theorem 4.6(2)) whether $A$ and $B$ define the same language.

For the NP lower bound, we reduce from the minimization problem for unambiguous FSAs, which is shown to be NP-complete by Jiang and Ravikumar [11]. Observe that, in the proof of Jiang and Ravikumar (Theorem 3.1 in [11]), it is shown that minimization is already NP-hard for unambiguous FSAs that *only accept strings of length two.* As FSAs that only accept strings of length two have a *sink state*, i.e., a state with no outgoing transitions, this simplifies our reduction.

Thereto, let $U$ be an unambiguous FSA that only accepts strings of length two and let $k$ be an integer. We construct an offline SUTA $A$ and an integer $\ell$ such that there exists an equivalent unambiguous FSA for $L(U)$ of size at most $k$ if and only if there exists an offline SUTA for $L(A)$ of size at most $\ell$.

Let $r$ be a symbol not occurring in $\text{Alphabet}(U)$. Intuitively, $A$ will accept the trees $r(w)$ such that $w \in L(U)$. We define $\text{States}(A) = \text{States}(U) \uplus \{q_0\}$, $\text{Alphabet}(A) = \text{Alphabet}(U) \uplus \{r\}$, and the rules of $A$ are defined as

- $r \to q_0$,
- $(q_0, r) \to U$, and
- $(q, a) \to E$, for every $q \in \text{States}(U)$ and $a \in \text{Alphabet}(U)$,

where $E$ is the UFA with $\text{States}(E) = \{q_f\}$ and $L(E) = \{\varepsilon\}$. Here, $q_f$ is a state in $\text{Final}(U)$ which is reachable in $U$ from an initial state of $U$. Finally, $\ell = k + 1$.

We need to argue that the reduction is correct. It is easy to see that $A$ accepts $\{r(w) \mid w \in L(U)\}$.

We need to prove that there is an unambiguous FSA for $L(U)$ of size at most $k$ if and only if there is an offline SUTA for $L(A)$ of size at most $\ell$. From left to right, let $U'$ be an unambiguous FSA of size at most $k$ for $L(U)$. Then, $A'$, constructed from $U'$ in the same way as $A$ is constructed from $U$ is an offline SUTA for $L(A)$ of size at most $\ell$. From right to left, let $A'$ be an offline SUTA for $L(A)$ of size at most $\ell$. W.l.o.g., we can assume that $A'$ is reduced. As $A'$ is an offline SUTA, $A'$ has a unique state $q_0$ which is used in the rule $r \to q_0$. Now consider the transition rule of $q_0$, i.e., $q_0 \to U''$ in $\text{Rules}(A)$. Clearly, $U''$ accepts $L(U)$. As $A'$ only accepts trees of depth two, we have that $q_0$ has no incoming or outgoing transitions in the definition of $U''$. (Otherwise, as $A'$ is reduced, trees can be constructed that are also in $L(A')$ and have depth larger than two, contradicting that $L(A') = L(A)$.) Therefore, the unambiguous FSA $U'$, obtained from $U''$ by removing state $q_0$ also recognizes $L(U)$ and has size at most $k$. Q.E.D.

A similar result as Theorem 4.6(3) was also proved in the context of extended DTDs in [15]. To the best of our knowledge, the precise complexity of minimization for BUTAs is still unknown. It is in NP, as testing equivalence between BUTAs is in PTIME.

### 4.3 Closure Properties

The same closure properties hold for the deterministic unranked tree automata as for the ranked tree automata we defined. The witness languages for non-closure are analogous to the ones in Section 3.2.

## 5 Regular Frontier Checks

In this section, we revisit the notion of regular frontier checks as a theoretical tool to close top-down deterministic languages under Boolean operations. We apply regular frontier checks to *unranked* automata.

To this end, we assume that the *frontier* of a tree is no longer an unordered set, but ordered from left to right. That is, we assume the lexicographical ordering $<$ on Frontier($t$).

**Definition 5.1.** A *top-down deterministic unranked tree automaton with regular frontier check (FC-UTA)* over alphabet $\Sigma$ is a (blind, online sensing, or offline sensing) top-down deterministic unranked tree automaton $A$ over alphabet $\Sigma$, together with a regular language $F$ over alphabet $\Sigma \times \text{States}(A)$.

A *run* of $A$ on a tree $t$ is defined precisely the same for blind, online sensing, or offline sensing unranked automata, respectively. A run $r$ is *accepting* if $(\text{lab}(u_1), r(u_1)) \cdots (\text{lab}(u_n), r(u_n)) \in F$, where $\text{Frontier}(t) = \{u_1, \ldots, u_n\}$ with $u_1 < \cdots < u_n$.

On ranked trees, top-down tree automata with frontier checks are known to be closed under union, intersection, and complement [12]. On unranked trees, these results can be obtained analogously. Moreover, in order to obtain this closure, one does not even need arbitrary regular languages. Indeed, it is sufficient to consider locally threshold testable languages [22] with diameter $k = 1$.

Hence, FC-UTAs could be one candidate for closing schema languages for XML under the Boolean operations, thereby resolving the issues in model management or schema integration.

## 6 Conclusions and Discussion

We presented an overview of top-down determinism in ranked and unranked tree automata, and explored several connections between them. As many connections were to be expected, we start the conclusions with a discrepancy. This discrepancy is observed between the (ranked) deterministic sensing tree automata (STAs) and the (unranked) deterministic offline sensing tree automata (offline SUTAs). Although they are closely related — they have, e.g., the same expressive power on binary trees and their way of assigning states to nodes in a top-down fashion is quite similar — we have shown that *optimization*, i.e., state minimization, is easy for one class but hard for the other.[7] Indeed, whereas state minimization is in PTIME for STAs, it is NP-complete for offline SUTAs. When inspecting the NP-hardness proof, the difference becomes even more striking: it already holds for offline SUTAs recognizing binary trees.

It thus follows that the determinism in offline SUTAs is actually not a very suitable notion for "top-down determinism" on unranked trees. Similarly as has been argued for the "standard" notion of *bottom-up determinism* on unranked trees [15], determinism in offline SUTAs corresponds more closely to *unambiguousness* rather than true determinism.[8]

On the positive side, the determinism in *online SUTAs* seems to be more suitable. Online SUTAs have been investigated in the context of XML schema languages under the name of *restrained competition EDTDs* and are already attributed to have desirable static analysis properties, while being more expressive than the core of XML Schema [14]. It is even decidable (EXPTIME-complete) for a bottom-up (non)-deterministic unranked tree automaton, whether there exists an equivalent deterministic online SUTA. The latter is referred to as the simplification problem.

In conclusion, only the determinism notion in online SUTAs is known to be truly top-down deterministic on unranked trees. Determinism in BUTAs, as defined by Brüggemann-Klein et al. [5] as the straightforward extension of the "standard" top-down determinism for ranked trees [7], is a bit different. In spite of the close connection to the well-behaved top-down determinism on ranked trees, minimizing deterministic BUTAs is not completely trivial and the precise complexity is still unknown. From an XML point of view, however, this notion of determinism might be less interesting. It assigns states to nodes, only based on the number of their siblings, which makes them rather poor in expressive power. When one would, for instance, want to allow an automaton to *read the label of a node* before assigning it a state,

[7] If PTIME $\neq$ NP.

[8] Of course, this is because our definition of determinism in offline SUTAs use unambiguous automata. However, we feel that similar problems will arise when investigating minimization for the equally expressive models presented in [8, 13].

which seems to be the case in XML schema languages for example, the determinism in online SUTAs would be the obvious candidate.

W.r.t. future research several natural directions emerge:

1. Top-down determinism and closure properties. As previously mentioned, the lack of closure under union is quite unnatural for an XML schema language. This leads to the following natural questions: (1) What are the possible additions to the deterministic top-down automaton model that closes them under the Boolean operations?; (2) What is the best way to approximate a Boolean combination of deterministic top-down tree automata?; and, (3) What are the properties of the class consisting of the Boolean closure of deterministic top-down tree automata (BC-TA)?

2. Optimization problems. Minimization is of course a very important problem. Can FC-UTAs or BC-TAs be efficiently minimized? Furthermore, what is the complexity of the simplification problem (as defined above) for the various models?

3. In practice not many XML schemas are available and some of those are syntactically incorrect, which leads to the problem of automatically inferring them from a set of XML documents. As the latter reduces to learning in the limit from positive data of deterministic top-down tree automata, it would be interesting to pinpoint classes which can be learned in this manner. Bex et al. addressed the problem of inferring subclasses of DTDs and XSDs [2, 3].

## References

[1] P..A. Bernstein and S. Melnik. Model management 2.0: manipulating richer mappings. In *ACM SIGMOD International Conference on Management of Data*, pages 1–12, 2007.

[2] G. J. Bex, F. Neven, T. Schwentick, and K. Tuyls. Inference of concise DTDs from XML data. In *VLDB*, pages 115–126, 2006.

[3] G. J. Bex, F. Neven, and S. Vansummeren. Inferring XML Schema Definitions from XML data.. In *VLDB*, 2007.

[4] T. Bray, J. Paoli, C.M. Sperberg-McQueen, E. Maler, and F. Yergeau. Extensible markup language (XML) 1.0 (fourth edition). Technical report, World Wide Web Consortium, 2006. http://www.w3.org/TR/xml.

[5] 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.

[6] J. Clark and M. Murata. Relax NG specification. Technical report, OASIS, 2001. http://relaxng.org/spec-20011203.html.

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

[8] J. Cristau, C. Löding, and W. Thomas. Deterministic automata on unranked trees. In *15th International Symposium on Fundamentals of Computation Theory*, pages 68–79, 2005.

[9] F. Gécseg and M. Steinby. Minimal ascending tree automata. *Acta Cybernetica*, 4(1):37–44, 1978.

[10] F. Gécseg and M. Steinby. *Tree Automata*. Akademia Kiadó, Budapest, 1984.

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

[12] E. Jurvanen, A. Potthoff, and W. Thomas. Tree languages recognizable by regular frontier check. In *Developments in Language Theory*, pages 3–17, 1993.

[13] W. Martens. *Static Analysis of XML Transformation- and Schema Languages*. PhD thesis, Hasselt University, 2006.

[14] W. Martens, F. Neven, T. Schwentick, and G. J. Bex. Expressiveness and complexity of XML Schema. *ACM Transactions on Database Systems*, 31(3):770–813, 2006.

[15] W. Martens and J. Niehren. On the minimization of XML schemas and tree automata for unranked trees. *Journal of Computer and System Sciences*, 73(4):550–583, 2007.

[16] Frank Neven and Thomas Schwentick. Query automata over finite trees. *Theor. Comput. Sci.*, 275(1-2):633–674, 2002.

[17] M. Nivat and A. Podelski. Minimal ascending and descending tree automata. *SIAM Journal on Computing*, 26(1):39–58, 1997.

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

[19] Jeffrey Shallit. Numeration systems, linear recurrences, and regular sets (extended abstract). In Werner Kuich, editor, *ICALP*, volume 623 of *Lecture Notes in Computer Science*, pages 89–100. Springer, 1992.

[20] C.M. Sperberg-McQueen and H. Thompson. XML Schema. Technical report, World Wide Web Consortium, 2007. http://www.w3.org/XML/Schema.

[21] 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.

[22] W. Thomas. Classifying regular events in symbolic logic. *Journal of Computer and System Sciences*, 25(3):360–376, 1982.

[23] J. Virágh. Deterministic ascending tree automata I. *Acta Cybernetica*, 5:33–44, 1980.