Deterministic Automata on Unranked Trees

Julien Cristau[1] Christof Löding[2] Wolfgang Thomas[2]

[1] LIAFA, Université Paris VII, France
[2] RWTH Aachen, Germany

Abstract. We investigate bottom-up and top-down deterministic automata on unranked trees. We show that for an appropriate definition of bottom-up deterministic automata it is possible to minimize the number of states efficiently and to obtain a unique canonical representative of the accepted tree language. For top-down deterministic automata it is well known that they are less expressive than the non-deterministic ones. By generalizing a corresponding proof from the theory of ranked tree automata we show that it is decidable whether a given regular language of unranked trees can be recognized by a top-down deterministic automaton. The standard deterministic top-down model is slightly weaker than the model we use, where at each node the automaton can scan the sequence of the labels of its successors before deciding its next move.

1 Introduction

Finite automata over finite unranked trees are a natural model in classical language theory as well as in the more recent study of XML document type definitions (cf. [Nev02]). In the theory of context-free languages, unranked trees (trees with finite but unbounded branching) arise as derivation trees of grammars in which the right-hand sides are regular expressions rather than single words ([BB02]). The feature of finite but unbounded branching appears also in the tree representation of XML documents.

The generalization of tree automata from the case of ranked label alphabets to the unranked case is simple: A transition, e.g., of a bottom-up automaton is of the form (L, a, q), allowing the automaton to assume state q at an a-labeled node with say n successors if the sequence $q_1 \dots q_n$ of states reached at the roots of the n subtrees of these successors belongs to L. Most core results of tree automata theory (logical closure properties, decidability of non-emptiness, inclusion, and equivalence) are easily transferred to this framework of "unranked tree automata" and "regular sets of unranked trees" (cf. [BWM01,Nev02]).

For certain other results of classical tree automata theory, however, such a transfer is less obvious and does not seem to be covered by existing work. In the present paper we deal with two such questions: the problem of automaton minimization, and the definition and expressive power of top-down automata (automata working from the root to the leaves, more closely following the pattern of XML query processing than the bottom-up version). We confine ourselves to the question of tree language recognition; so we do not address models like the query automata of [NS02] or the transducers of [MSV03].

The minimization problem has be reconsidered for the unranked case because two types of automata are involved: the finite tree automaton $\mathcal{A}$ used for building up run trees (on given input trees), and the finite word automata $\mathcal{B}_L$ accepting the languages L that occur in the $\mathcal{A}$-transitions. The $\mathcal{A}$-states are the input letters to the $\mathcal{B}_L$, and the $\mathcal{B}_L$-states are needed to produce the "next $\mathcal{A}$-state" (in bottom-up mode). It is not clear a priori how and in which order to minimize these automata. Using a natural definition of $\mathcal{B}_L$-automaton (which depends on a label a and produces an $\mathcal{A}$-state as output), we show in Section 3 below that a simultaneous and efficient minimization of $\mathcal{A}$ and the $\mathcal{B}_L$ is possible, moreover resulting in a minimal tree automaton that is unique up to isomorphism.

For the question of deterministic top-down processing of input trees, we start with well known results of [Vir80,GS84] on the ranked case. The generalization to the unranked case requires introducing a finite automaton that proceeds from state q at an a-labeled node deterministically to new states $q_1 \dots q_n$ at the n successor nodes. A natural option is to provide the numbers n and i as input in order to compute q_i. A second option, closer to the idea of XML document processing, is to provide as inputs the sequence $a_1 \dots a_n$ of successor labels and the position i. In both cases, the simple approach to define transitions via a finite table does not suffice, instead one has to introduce appropriate transducers to implement transitions. In Section 4, we present such transducers (in the form of bimachines [Eil74,Ber79]), introduce the corresponding top-down tree automata, and show that for a regular set of unranked trees one can decide whether it is recognizable by either of these top-down tree automata. For the technical presentation we focus on the second option mentioned above. The main point is an appropriate definition of "path language", recording the possible paths of trees in a given tree language; the derived notion of "path-closed" tree language then captures those tree languages that are recognizable deterministically in top-down mode.

The paper starts (in Section 2) with some technical preliminaries, gives in Section 3 the results on minimization, in Section 4 the study of top-down automata, and closes in Section 5 with some pointers to current and future work.

We thank S. Abiteboul, F. Neven, and Th. Schwentick for comments on a preliminary version of this work.

2 Automata on Unranked Trees

In this section we define unranked trees and different models of automata running on such trees. In the following, Σ always denotes a finite alphabet, i.e., a finite set of symbols, $\mathbb{N}$ denotes the set of natural numbers, and $\mathbb{N}_{>0}$ denotes the set of positive natural numbers. For a set X we denote by X^* the set of all finite words over X. The empty word is denoted by ε.

A *tree domain* D is a non-empty, prefix-closed subset of $\mathbb{N}_{>0}^*$ satisfying the following condition: if $xi \in D$ for $x \in \mathbb{N}_{>0}^*$ and $i \in \mathbb{N}_{>0}$, then $xj \in D$ for all j with $1 \leq j \leq i$.

An *unranked tree* t over Σ (simply tree in the following) is a mapping $t : \mathrm{dom}_t \to \Sigma$ with a finite tree domain dom_t. The elements of dom_t are called the *nodes* of t. For $x \in \mathrm{dom}_t$ we call nodes of the form $xi \in \mathrm{dom}_t$ with $i \in \mathbb{N}_{>0}$ the *successors* of x (where xi is the ith successor). As usual, a *leaf* of t is a node without successor. If the root of t is labeled by a, i.e., $t(\varepsilon) = a$, and if the root has k successors at which the subtrees $t_1, \dots, t_k$ are rooted, then we denote this by $t = a(t_1 \cdots t_k)$. The set of all unranked trees over Σ is denoted by $\mathcal{T}_\Sigma$. For $a \in \Sigma$ we denote by $\mathcal{T}^a_\Sigma$ the set of all trees from $\mathcal{T}_\Sigma$ whose root is labeled by a.

A *non-deterministic bottom-up tree automaton* ($\uparrow$NTA) $\mathcal{A} = (Q, \Sigma, \Delta, F)$ consists of a finite set Q of states, a finite input alphabet Σ, a finite set $\Delta \subseteq \mathrm{Reg}(Q) \times \Sigma \times Q$ of transitions ($\mathrm{Reg}(Q)$ denotes the set of regular languages over Q), and a set $F \subseteq Q$ of final states.

A *run* of $\mathcal{A}$ on a tree t is a function $\rho : \mathrm{dom}_t \to Q$ with the following property: for each $x \in \mathrm{dom}_t$ with n successors $x1, \dots, xn$ there is a transition $(L, t(x), \rho(x)) \in \Delta$ such that the word $\rho(x1) \cdots \rho(xn)$ is in L. If x is a leaf, this means that there must be a transition $(L, t(x), \rho(x)) \in \Delta$ with $\varepsilon \in L$. If for some run ρ of $\mathcal{A}$ on t the root of ρ is labeled with q, then we write $t \to_\mathcal{A} q$. For $Q' \subseteq Q$ we write $t \to_\mathcal{A} Q'$ if $t \to_\mathcal{A} q$ for some $q \in Q'$. We call ρ *accepting* if $\rho(\varepsilon) \in F$ and say that t is accepted by $\mathcal{A}$ if there is an accepting run of $\mathcal{A}$ on t. The language $T(\mathcal{A})$ accepted by $\mathcal{A}$ is $T(\mathcal{A}) := \{t \in T_\Sigma \mid \mathcal{A} \text{ accepts } t\}$. The *regular languages of unranked trees* are those that can be accepted by $\uparrow$NTAs.

In the definition of $\uparrow$NTA we did not specify how the regular languages used in the transitions are given. First of all, note that it is not necessary to have two transitions (L_1, a, q) and (L_2, a, q) because these can be merged into a single transition $(L_1 \cup L_2, a, q)$. Usually, one then assumes that the transition function is given by a set of regular expressions or non-deterministic finite automata defining for each $q \in Q$ and $a \in \Sigma$ the language $L_{a,q}$ with $(L_{a,q}, a, q) \in \Delta$.

One can also define non-deterministic tree automata that work in a top-down fashion. For this purpose it is sufficient to view the final states as initial states. Thus, for non-deterministic automata it does not make any difference whether we consider top-down or bottom-up automata. In contrast, to obtain a deterministic model with the same expressive power as the corresponding non-deterministic model one has to consider bottom-up automata as introduced in the following. Deterministic top-down automata are treated in Section 4.

The standard definition of deterministic bottom-up tree automata ($\uparrow$DTA) is obtained by imposing a semantic restriction on the set of transitions: it is required that for each letter a and all states q_1, q_2 if there are transitions (L_1, a, q_1) and (L_2, a, q_2), then $L_1 \cap L_2 = \emptyset$. Each $\uparrow$NTA can be transformed into an equivalent $\uparrow$DTA using a standard subset construction [BWM01]. Here, we do not use this semantic approach to define determinism but require a representation of the transition function that syntactically enforces determinism. Besides the advantage of not needing any semantic restrictions, our model is obtained in a natural way when applying the subset construction to $\uparrow$NTAs. Since minimization is often applied to reduce the result of a determinization construction, the choice of this model is a natural one for our purposes.

A ↑DTA $\mathcal{A}$ is given by a tuple $\mathcal{A} = (Q, \Sigma, (\mathcal{D}_a)_{a\in\Sigma}, F)$ with Q, Σ, and F as for ↑NTA, and deterministic finite automata $\mathcal{D}_a$ with output defining the transitions of $\mathcal{A}$. Each of the $\mathcal{D}_a$ (with $a \in \Sigma$) is of the form $\mathcal{D}_a = (S_a, Q, s_a^{\text{in}}, \delta_a, \lambda_a)$ with a finite set S_a of states, input alphabet Q, initial state s_a^{in}, transition function $\delta_a : S_a \times Q \to S_a$, and output function $\lambda_a : S_a \to Q$. As usual, we define $\delta_a^* : S_a \times Q^* \to S_a$ by $\delta_a^*(s, \varepsilon) = s$ and $\delta_a^*(s, uq) = \delta_a(\delta_a^*(s, u), q)$.

Such a ↑DTA can be transformed into the standard representation as follows. For each $q \in Q$ and $a \in \Sigma$ let $L_{a,q} = \{w \in Q^* \mid \lambda_a(\delta_a^*(s_a^{\text{in}}, w)) = q\}$. Then, the set of transitions defined by the family $(\mathcal{D}_a)_{a\in\Sigma}$ consists of all transitions $(L_{a,q}, a, q)$.

Example 1. For $\Sigma = \{\wedge, \vee, 0, 1\}$ we consider trees whose leaves are labeled by 0 or 1, and whose inner nodes are labeled by $\wedge$ or $\vee$. Such trees can be evaluated in a natural way to 0 or 1. Let T be the language of all trees that evaluate to 1. We define a ↑DTA for T with state set $Q = \{q_0, q_1, q_\perp\}$, final states $F = \{q_1\}$, and automata $\mathcal{D}_a$, $a \in \Sigma$, as depicted in Figure 1. An entry $s \mid q$ in the picture means that the output at state s is q, e.g., $\lambda_\wedge(s_\wedge^{\text{in}}) = q_\perp$. For readability we have omitted the transitions leading to rejection. All missing transitions are assumed to lead to a sink state of the respective automaton with output $q_\perp$.

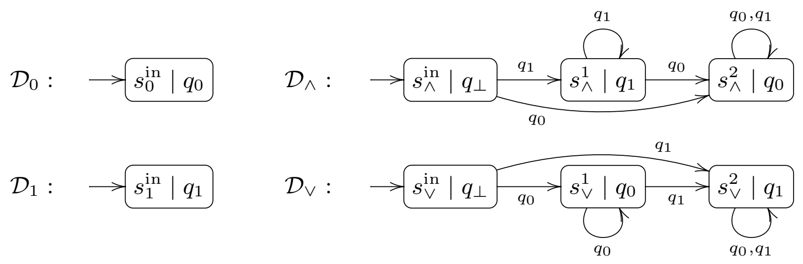

Fig. 1. A ↑DTA recognizing all $\wedge$-$\vee$-trees that evaluate to 1

3 Minimization of Deterministic Bottom-Up Automata

In this section $\mathcal{A}$ always denotes a ↑DTA $\mathcal{A} = (Q, \Sigma, (\mathcal{D}_a)_{a\in\Sigma}, F)$ with $\mathcal{D}_a = (S_a, Q, s_a^{\text{in}}, \delta_a, \lambda_a)$ for each $a \in \Sigma$. Furthermore, we let $S = \bigcup_{a\in\Sigma} S_a$. For complexity considerations we define the size of $\mathcal{A}$ as $|\mathcal{A}| = |Q| \cdot |S|$. This is a reasonable measure since the sizes of the transition functions of the automata $\mathcal{D}_a$ are of order $|Q| \cdot |S_a|$.

For minimization it is necessary to ensure that all states (from Q and S) are reachable. Note that there is an interdependence since a state in Q is reachable if it is the output of some reachable state in S, and a state in S is reachable if it can be reached by some input consisting of reachable states in Q.

Lemma 1. *The set of reachable states of a given $\uparrow$DTA can be computed in linear time.*

Proof. The algorithm maintains a set S' of reachable states from S and a set Q' of reachable states from Q. These sets are initialized to $S' = \{s_a^{\text{in}} \mid a \in \Sigma\}$ and $Q' = \{\lambda_a(s_a^{\text{in}}) \mid a \in \Sigma\}$. Starting from S' the transition graphs of the $\mathcal{D}_a$ are traversed in a breadth-first manner using only the transition labels from Q'. Whenever we encounter a state $s \in S_a$ with $q = \lambda_a(s) \notin Q'$, then q is added to Q' and the targets of the transitions with label q departing from those states in S' that have already been processed by the breadth first search are added to S'. This algorithm traverses each transition of the automata $\mathcal{D}_a$ at most once. and hence can be implemented to run in linear time. $\square$

From now on we assume that all states in $\mathcal{A}$ are reachable. For each $q \in Q$ we define $T_q = \{t \in \mathcal{T}_\Sigma \mid t \to_{\mathcal{A}} q\}$, and for each $a \in \Sigma$ and $s \in S_a$ we define

$$T_s = \{a(t_1 \cdots t_k) \mid \exists q_1, \ldots, q_k \ : \ t_i \in T_{q_i} \text{ and } \delta_a^*(s_a^{\text{in}}, q_1 \cdots q_k) = s\}.$$

If all states are reachable, then these sets are non-empty, and we can fix for each $q \in Q$ some $t_q \in T_q$ and for each $s \in S$ some $t_s \in T_s$.

To prove the existence of a unique minimal $\uparrow$DTA for a regular tree language we introduce two equivalence relations in the spirit of Nerode's congruence for word languages. To this aim we first define two different kinds of concatenations for trees.

The set $\mathcal{T}_{\Sigma,X}$ of pointed trees over Σ contains all trees t from $\mathcal{T}_{\Sigma \cup \{X\}}$ (for a new symbol X) such that exactly one leaf of t is labeled by X. For $t \in \mathcal{T}_{\Sigma,X}$ and $t' \in \mathcal{T}_\Sigma \cup \mathcal{T}_{\Sigma,X}$ we denote by $t \circ t'$ the tree obtained from t by replacing the leaf labeled X by t'. For $T \subseteq \mathcal{T}_\Sigma$ the equivalence relation $\sim_T \subseteq \mathcal{T}_\Sigma \times \mathcal{T}_\Sigma$ is defined by

$$t_1 \sim_T t_2 \qquad \text{iff} \qquad \forall t \in \mathcal{T}_{\Sigma,X} \ : \ t \circ t_1 \in T \Leftrightarrow t \circ t_2 \in T.$$

This relation is called 'top-congruence' in [BWM01]. In the case of ranked trees it is the natural extension of Nerode's congruence from words to trees. However, as already noted in [BWM01], for T being regular in the unranked setting it is not sufficient that $\sim_T$ is of finite index. One also has to impose a condition that ensures the regularity of the 'horizontal languages' that are used in the transition function. For this we need another concatenation operation on trees.

For trees $t = a(t_1 \ldots t_k)$ and $t' = a(t'_1 \cdots t'_\ell)$ let $t \odot t' = a(t_1 \cdots t_k t'_1 \cdots t'_\ell)$. The equivalence relation $\overset{\rightarrow}{\sim}_T$ is defined for all $a \in \Sigma$ and $t_1, t_2 \in \mathcal{T}_\Sigma^a$ by

$$t_1 \overset{\rightarrow}{\sim}_T t_2 \qquad \text{iff} \qquad \forall t \in \mathcal{T}_\Sigma^a \ : \ t_1 \odot t \sim_T t_2 \odot t$$

To simplify notation we write $[t]$ for the $\sim_T$-class of t and $[t]_\to$ for the $\overset{\rightarrow}{\sim}_T$-class of t. If T is accepted by a $\uparrow$DTA $\mathcal{A}$, then $\mathcal{A}$ has to distinguish trees that are not equivalent. This is expressed in the following lemma.

Lemma 2. *If $\mathcal{A}$ accepts the language T, then $T_q \subseteq [t_q]$ for each $q \in Q$ and $T_s \subseteq [t_s]_\to$ for each $a \in \Sigma$ and $s \in S_a$.*

The above lemma implies that $\overset{\rightarrow}{\sim}_T$ is of finite index if T is regular. On the other hand, if $\overset{\rightarrow}{\sim}_T$ is of finite index, then this ensures the regularity of what is called 'local views' in [BWM01] and hence T is regular. In the following we show that the equivalence classes of $\sim_T$ and $\overset{\rightarrow}{\sim}_T$ can be used to define a canonical minimal $\uparrow$DTA $\mathcal{A}_T$ for T. The equivalence classes of $\sim_T$ correspond to the states of the tree automaton, and the equivalence classes of $\overset{\rightarrow}{\sim}_T$ restricted to $\mathcal{T}_\Sigma^a$ correspond to the states of the automaton defining the transitions for label a.

For the definition of $\mathcal{A}_T$ we need the following lemma stating that $\overset{\rightarrow}{\sim}_T$ refines $\sim_T$ and that $\overset{\rightarrow}{\sim}_T$ is a right-congruence (w.r.t. $\odot$). The proof of this lemma is straightforward.

Lemma 3. *(a) If $t_1 \overset{\rightarrow}{\sim}_T t_2$, then $t_1 \sim_T t_2$ for all $t_1, t_2 \in \mathcal{T}_\Sigma$.*
(b) If $t_1 \overset{\rightarrow}{\sim}_T t_2$ and $t'_1 \sim_T t'_2$, then $t_1 \odot a(t'_1) \overset{\rightarrow}{\sim}_T t_2 \odot a(t'_2)$ for all $t_1, t_2 \in \mathcal{T}_\Sigma^a$ and all $t'_1, t'_2 \in \mathcal{T}_\Sigma$.

The assignment of $\overset{\rightarrow}{\sim}_T$-classes to $\sim_T$-classes that is induced by (a) corresponds to the mappings λ_a that assign to each state from S_a a state from Q.

The following theorem states the existence of a unique (up to isomorphism) $\uparrow$DTA for every regular language T of unranked trees. The notion of homomorphism that we use in the statement of the theorem is the natural one: a *homomorphism* from $\mathcal{A}_1 = (Q_1, \Sigma, (\mathcal{D}_a^1)_{a\in\Sigma}, F_1)$ to $\mathcal{A}_2 = (Q_2, \Sigma, (\mathcal{D}_a^2)_{a\in\Sigma}, F_2)$ maps the states from Q_1 to states from Q_2 while respecting final and non-final states, and maps for each $a \in \Sigma$ the set S_a^1 to S_a^2 (S_a^i denotes the state set of $\mathcal{D}_a^i$) while respecting the initial state, the transition function, and the output function.

Theorem 1. *For every regular $T \subseteq \mathcal{T}_\Sigma$ there is a unique minimal $\uparrow$DTA $\mathcal{A}_T$ and for each $\uparrow$DTA $\mathcal{A}$ recognizing T there is a surjective homomorphism from $\mathcal{A}$ to $\mathcal{A}_T$.*

Proof. Define $\mathcal{A}_T = (Q_T, \Sigma, (\mathcal{D}_a^T)_{a\in\Sigma}, F_T)$ by $Q_T = \mathcal{T}_\Sigma/{\sim_T}$, $F = \{[t] \mid t \in T\}$, and $\mathcal{D}_a^T = (S_a^T, Q_T, [a]_\rightarrow, \delta_a^T, \lambda_a^T)$ with $S_a^T = \mathcal{T}_\Sigma^a/{\overset{\rightarrow}{\sim}_T}$, $\delta_a^T([t]_\rightarrow, [t']) = [t \odot a(t')]_\rightarrow$ for $t \in \mathcal{T}_\Sigma^a$ and $t' \in \mathcal{T}_\Sigma$, and $\lambda_a^T([t]_\rightarrow) = [t]$. Using Lemma 3 one can easily show that $t \rightarrow_{\mathcal{A}_T} [t]$ and hence $T(\mathcal{A}_T) = T$. Furthermore, if $\mathcal{A}$ is some $\uparrow$DTA for T, then it is not difficult to see that mapping each state $q \in Q$ to $[t_q]$ and each $s \in S$ to $[t_s]_\rightarrow$ defines a surjective homomorphism from $\mathcal{A}$ to $\mathcal{A}_T$. $\square$

We now give an algorithm that computes this minimal $\uparrow$DTA $\mathcal{A}_T$ starting from any automaton $\mathcal{A}$ for T. This minimization procedure is an extension of the classical minimization procedure for finite automata (cf. [HU79]). We define equivalence relations on the state sets Q and S that correspond to the relations $\sim_T$ and $\overset{\rightarrow}{\sim}_T$ and obtain the minimal automaton by merging equivalent states. For $q_1, q_2 \in Q$ let $q_1 \sim_\mathcal{A} q_2$ iff $(t \circ t_{q_1} \rightarrow_\mathcal{A} F \Leftrightarrow t \circ t_{q_2} \rightarrow_\mathcal{A} F)$ for all $t \in \mathcal{T}_{\Sigma,X}$. For $a \in \Sigma$ and $s_1, s_2 \in S_a$ let $s_1 \sim_\mathcal{A} s_2$ iff $\lambda_a(\delta_a^*(s_1, u)) \sim_\mathcal{A} \lambda_a(\delta_a^*(s_2, u))$ for all $u \in Q^*$. The following lemma states that it is indeed possible to group equivalent states into a single state.

Lemma 4. *If $q_1 \sim_\mathcal{A} q_2$ for $q_1, q_2 \in Q$ and $s_1 \sim_\mathcal{A} s_2$ for $s_1, s_2 \in S_a$, then $\delta_a(s_1, q_1) \sim_\mathcal{A} \delta_a(s_2, q_2)$.*

For $q \in Q$ and $s \in S$ we denote by $[q]$ and $[s]$ the $\sim_{\mathcal{A}}$-class of q and s, respectively. The reduced automaton $A_\sim$ is defined as $\mathcal{A}_\sim = (Q/{\sim_{\mathcal{A}}}, \Sigma, (\mathcal{D}_a^\sim)_{a\in\Sigma}, F/{\sim_{\mathcal{A}}})$ with $\mathcal{D}_a^\sim = (S_a/{\sim_{\mathcal{A}}}, Q/{\sim_{\mathcal{A}}}, [s_a^{\text{in}}], \delta_a^\sim, \lambda_a^\sim)$, $\delta_a^\sim([s],[q]) = [\delta_a(s,q)]$, and $\lambda_a^\sim([s]) = [\lambda_a(s)]$. Lemma 4 ensures that the definitions of δ_a and λ_a do not depend on the chosen representatives of the equivalence classes.

Theorem 2. *If $\mathcal{A}$ is an automaton for the language T, then $\mathcal{A}_T$ and $\mathcal{A}_\sim$ are isomorphic.*

Proof. From the definitions of $\sim_{\mathcal{A}}$, $\sim_T$, and $\overset{\rightarrow}{\sim}_T$ one can easily deduce that $q_1 \sim_{\mathcal{A}} q_2$ iff $[t_{q_1}] = [t_{q_2}]$, and $s_1 \sim_{\mathcal{A}} s_2$ iff $[t_{s_1}]_\to = [t_{s_2}]_\to$. This implies that in the reduced automaton every state $[q]$ can be identified with $[t_q]$ and each state $[s]$ can be identified with $[t_s]_\to$. ☐

Hence, to compute the unique minimal automaton for T it suffices to compute the relation $\sim_{\mathcal{A}}$. The algorithm shown in Figure 2 marks all pairs of states that are not in the relation $\sim_{\mathcal{A}}$.

```
INPUT: $\uparrow$DTA $\mathcal{A} = (Q, \Sigma, (\mathcal{D}_a)_{a\in\Sigma}, F)$ with $D_a = (S_a, Q, s_a^{\text{in}}, \delta_a, \lambda_a)$

1. Mark each pair $(q_1, q_2) \in Q^2$ with $q_1 \in F \Leftrightarrow q_2 \notin F$.
2. repeat
3.    For each $a \in \Sigma$ mark $(s_1, s_2) \in S_a^2$ if $(\lambda_a(s_1), \lambda_a(s_2))$ is marked.
4.    For each $a \in \Sigma$ and $q \in Q$ mark $(s_1, s_2) \in S_a^2$ if $(\delta_a(s_1, q), \delta_a(s_2, q))$ is marked.
5.    For each $a \in \Sigma$ and $s \in S_a$ mark $(q_1, q_2) \in Q^2$ if $(\delta_a(s, q_1), \delta_a(s, q_2))$ is marked.
6. until no new pairs are marked

OUTPUT: $R = \{(q_1, q_2) \in Q^2 \mid (q_1, q_2)$ not marked$\}$
            $\cup \{(s_1, s_2) \in S_a^2 \mid a \in \Sigma$ and $(s_1, s_2)$ not marked$\}$
```

Fig. 2. Algorithm Equivalent-States

Theorem 3. *The algorithm* Equivalent-States *from Figure 2 computes for input $\mathcal{A}$ the relation $\sim_{\mathcal{A}}$.*

Proof. We first show that all pairs marked by the algorithm are non-equivalent. For the pairs marked in lines 1,3, and 4 this is a direct consequence of the definition of $\sim_{\mathcal{A}}$. For pairs (q_1, q_2) marked in line 5 the claim follows from Lemma 4 applied to q_1, q_2, and $s = s_1 = s_2$.

To show that all pairs of non-equivalent states are marked, we look at the minimal 'size' of a witness that separates the two states. For the states from Q these witnesses are pointed trees. The size we are interested in is the depth of the leaf labeled by X, i.e., for $t \in \mathcal{T}_{\Sigma,X}$ we define $|t|_X$ to be the depth of X in t. For $q_1, q_2 \in Q$ and $n \in \mathbb{N}$ we define $q_1 \sim_n q_2$ iff $(t \circ t_{q_1} \to_{\mathcal{A}} F \Leftrightarrow t \circ t_{q_2} \to_{\mathcal{A}} F)$ for all $t \in \mathcal{T}_{\Sigma,X}$ with $|t|_X \leq n$. For $s_1, s_2 \in S_a$ we let $s_1 \sim_n s_2$ iff $\lambda_a(\delta_a^*(s_1, u)) \sim_n \lambda_a(\delta_a^*(s_2, u))$ for all $u \in Q^*$. Using this definition one can show that a pair of states is marked in the ith iteration of the loop iff i is the minimal number such that the states are not in the relation $\sim_i$. For this we assume that each of the lines 3–5 is executed as long as there are pairs that can be marked in the respective line. $\square$

For readability we have used the technique of marking pairs of non-equivalent states to compute the relation $\sim_{\mathcal{A}}$. A concrete implementation of the algorithm should rather use the technique of refining an equivalence relation represented by its equivalence classes. This technique is used to improve the complexity of minimization of finite automata on words [Hop71] and can also be applied to the minimization of automata on finite ranked trees (cf. [CDG+97]). Using this technique one can obtain an algorithm running in quadratic time. A more detailed analysis on whether this bound can be improved is still to be done.

Theorem 4. *Given a* $\uparrow$*DTA* $\mathcal{A}$ *one can compute in quadratic time the minimal* $\uparrow$*DTA that is equivalent to* $\mathcal{A}$*.*

4 Deterministic Top-Down Automata

As in the case of ranked trees, deterministic automata that work in a top-down fashion are not as expressive as non-deterministic ones. In this section we introduce such a model for unranked trees and show that it is decidable whether a given regular tree language can be accepted by a deterministic top-down automaton. When directly adapting the definition of top-down deterministic automata on ranked trees, one obtains a model that, depending on its current state, the current label, and the number of successors of the current node, decides which states it sends to the successors. Here, we have decided to make the model a bit more expressive by allowing for a transition to take into account not only the number of successors but also their labeling. All the results from this section can be adapted in a straightforward way to the weaker model as described above.

To define the transitions as just mentioned we use a certain kind of transducer to convert the sequence of labels of the successors of a node into a sequence of states of the tree automaton. Since this transducer should have information on the whole successor sequence before deciding which state to put at a certain successor we use the formalism of bimachines (cf. [Eil74,Ber79]).

A *bimachine* is of the form $\mathcal{B} = (\Sigma, \Gamma, \overrightarrow{\mathcal{B}}, \overleftarrow{\mathcal{B}}, f)$, where Σ is the input alphabet, Γ is the output alphabet, $\overrightarrow{\mathcal{B}} = (\overrightarrow{S}, \Sigma, s_0^{\rightarrow}, \overrightarrow{\delta})$ and $\overleftarrow{\mathcal{B}} = (\overleftarrow{S}, \Sigma, s_0^{\leftarrow}, \overleftarrow{\delta})$ are deterministic finite automata over Σ (without final states), and $f : \overrightarrow{S} \times \Sigma \times \overleftarrow{S} \rightarrow \Gamma$ is the output function.

Given a word $u \in \Sigma^*$ consisting of k letters $u = a_1 \cdots a_k$, $\mathcal{B}$ produces an output $v = b_1 \cdots b_k$ over Γ that is defined as follows. Let $s_0^{\rightarrow} s_1^{\rightarrow} \cdots s_k^{\rightarrow}$ be the run of $\overrightarrow{\mathcal{B}}$ on $a_1 \cdots a_k$, and let $s_0^{\leftarrow} s_1^{\leftarrow} \cdots s_k^{\leftarrow}$ be the run of $\overleftarrow{\mathcal{B}}$ on the reversed input $a_k \cdots a_1$. Then the ith output letter b_i is given by $b_i = f(s_i^{\rightarrow}, a_i, s_{k-i+1}^{\leftarrow})$. This definition is illustrated in Figure 3.

We denote the function computed by $\mathcal{B}$ by $f_{\mathcal{B}}$. One should note that using bimachines we remain inside the domain of regular languages in the sense that

$$\begin{tikzcd}[row sep=small, column sep=small]
\overrightarrow{\mathcal{B}} : & \overrightarrow{s_0} & & \overrightarrow{s_1} & & \overrightarrow{s_2} & \cdots & \overrightarrow{s_{k-1}} & & \overrightarrow{s_k} & \\
& & a_1 \arrow[ur, dash] \arrow[dl, dash] & & a_2 \arrow[ur, dash] \arrow[dl, dash] & & \cdots & & a_k \arrow[ur, dash] \arrow[dl, dash] & & \\
& \overleftarrow{s_k} & {} \arrow[d] & \overleftarrow{s_{k-1}} & {} \arrow[d] & \overleftarrow{s_{k-2}} & \cdots & \overleftarrow{s_1} & {} \arrow[d] & \overleftarrow{s_0} & : \overleftarrow{\mathcal{B}} \\
& & b_1 & & b_2 & & & \cdots & b_k & &
\end{tikzcd}$$

Fig. 3. Computation of a bimachine

$f_{\mathcal{B}}(L)$ for a regular language L is again regular. For further results on bimachines see, e.g., [Ber79] or [Eil74].

A *deterministic top-down tree automaton* ($\downarrow$DTA) uses such bimachines for its transitions. It is of the form $\mathcal{A} = (Q, \Sigma, f_{\text{in}}, (\mathcal{B}_q)_{q \in Q}, F)$, where Q is finite set of states, Σ is the input alphabet, $f_{\text{in}} : \Sigma \to Q$ is the initial function, $F \subseteq Q$ is a set of final states, and each $\mathcal{B}_q$ is a bimachine with input alphabet Σ and output alphabet Q.

A run of $\mathcal{A}$ on a tree t is mapping $\rho : \text{dom}_t \to Q$ such that $\rho(\varepsilon) = f_{\text{in}}(t(\varepsilon))$ and for each node x of t with $n > 0$ successors $x1, \dots, xn$ we have $\rho(x1) \cdots \rho(xn) = f_{\mathcal{B}_{\rho(x)}}(t(x1) \cdots t(xn))$. Note that for each t there is exactly one run of $\mathcal{A}$ on t. The run ρ is accepting if each leaf is labeled with a final state. The language accepted by $\mathcal{A}$ consists of all trees t such that the run of $\mathcal{A}$ on t is accepting.

It is not difficult to see that not all regular tree languages can be recognized by $\downarrow$DTAs. Consider for example the language $T_{cd} = \{a(a(c)a(d)), a(a(d)a(c))\}$. Every $\downarrow$DTA recognizing the two trees from T_{cd} will also recognize the trees $a(a(c)a(c))$ and $a(a(d)a(d))$.

We show that it is decidable whether a given regular tree language can be recognized by a $\downarrow$DTA. The proof follows the same lines as for ranked trees using the notions of path language and path closure ([Vir80,GS84]). When a $\downarrow$DTA descends a tree, then on each path it can only see the sequence of labels of this path and on each level the sequence of labels of the siblings. For example, the information known to a $\downarrow$DTA on the leftmost path in the first tree from T_{cd} can be coded as $a \rhd \triangledown a \lhd a \rhd \triangledown \lhd c$. The letters outside the segments $\rhd \cdots \lhd$ code the sequence of labels on the path, and the letters between the pairs $\rhd$, $\lhd$ show the labels of the siblings with the position corresponding to the node of the considered path marked by $\triangledown$. This idea leads to a corresponding concept of path language. Note that this generalizes the standard concept of path language over ranked alphabets where a path code has the form $a_1 i_1 a_2 i_2 \cdots i_{\ell-1} a_\ell$ indicating that successively the successors $i_1, i_2, \dots$ are taken. In our setting, the rank of a_j is captured by the length of the subsequent segment $\rhd \cdots \lhd$ and i_j by the position of $\triangledown$ in this segment.

The alphabet we use for path languages is $\Sigma_{\text{path}} = \Sigma \cup \{\rhd, \lhd, \triangledown\}$. The *path language* $\pi(t)$ of a tree t is defined inductively as $\pi(a) = a$ for each $a \in \Sigma$ and $\pi(t) = \bigcup_{i=1}^{k} \{a \rhd a_1 \cdots a_{i-1} \triangledown a_{i+1} \cdots a_k \lhd w \mid w \in \pi(a_i(t_i))\}$ for $t = a(a_1(t_1) \cdots a_k(t_k))$. In this definition we allow that t_i is empty. In this case $a_i(t_i) = a_i$. The path language of $T \subseteq \mathcal{T}_\Sigma$ is $\pi(T) = \bigcup_{t \in T} \pi(t)$. The *path closure* of T is $\mathrm{cl}(T) = \{t \in \mathcal{T}_\Sigma \mid \pi(t) \subseteq \pi(T)\}$. A language with $T = \mathrm{cl}(T)$ is called *path closed*. In the following we show that the $\downarrow$DTA-recognizable languages are exactly the path closed regular tree languages. The proof goes through a sequence of lemmas.

Lemma 5. *If $T \subseteq \mathcal{T}_\Sigma$ is regular, then $\pi(T)$ is a regular language of words.*

Proof. Let $\mathcal{A}$ be a $\uparrow$NTA for T with one transition $(L_{a,q}, a, q)$ for each pair of state q and letter a. We assume that for each state q of $\mathcal{A}$ there is a tree t with $t \rightarrow_\mathcal{A} q$. Note that by applying a straightforward procedure for identifying reachable states, we can restrict to this case.

One can easily define a non-deterministic finite automaton $\mathcal{C}$ accepting $\pi(T)$. This automaton, on reading the first symbol of the input word w, remembers this first symbol a and guesses a final state q of $\mathcal{A}$ such that $t \rightarrow_\mathcal{A} q$ for some t with $w \in \pi(t)$. So, after this first step, $\mathcal{C}$ is in state (q, a).

The next part of the input is of the form $\rhd a_1 \cdots a_{i-1} \triangledown a_{i+1} \cdots a_k \lhd a_i$ (if it is not of this form the input is rejected). The automaton $\mathcal{C}$ guesses a sequence $q_1 \cdots q_k$ such that $q_1 \cdots q_k \in L_{a,q}$. For this purpose, on reaching the gap $\triangledown$, it guesses a_i and verifies this guess after reading $\lhd$. Furthermore, it simulates an automaton for $L_{a,q}$ to verify that the guessed sequence is indeed in $L_{a,q}$. On passing the gap $\triangledown$ it remembers the state q_i and then moves to a state (q_i, a_i) after having read $\lhd a_i$. The final states of $\mathcal{C}$ are those pairs (q, a) for which $\varepsilon \in L_{a,q}$. □

Lemma 6. *If $T \subseteq \mathcal{T}_\Sigma$ is regular, then $\mathrm{cl}(T)$ is recognizable by a $\downarrow$DTA.*

Proof. If T is regular, then $\pi(T)$ is a regular word language by Lemma 5. Let $\mathcal{C} = (Q, \Sigma_{\mathrm{path}}, q_0, \delta, F)$ be a deterministic finite automaton for $\pi(T)$. We briefly sketch how to construct a $\downarrow$DTA $\mathcal{A} = (Q, \Sigma, f_{\mathrm{in}}, (\mathcal{B}_q)_{q \in Q}, F)$ for T that simulates $\mathcal{C}$ on every path. Note that $\mathcal{A}$ has the same set of states as $\mathcal{C}$ and the same set of final states as $\mathcal{C}$. The initial function is defined by $f_{\mathrm{in}}(a) = \delta(q_0, a)$.

For every state $q \in Q$ the behavior of $\mathcal{B}_q$ is as follows. When processing the word $a_1 \cdots a_k$ the machine $\mathcal{B}_q$ should output the sequence $q_1 \cdots q_k$ where q_i is the state reached by $\mathcal{C}$ when reading the word $\rhd a_1 \cdots a_{i-1} \triangledown a_{i+1} \cdots a_k \lhd a_i$ starting from q. To realize this idea $\mathcal{B}_q$ has for each i (1) to compute the behavior of $\mathcal{C}$ on $\rhd a_1 \cdots a_{i-1} \triangledown$ starting from q, and (2) the behavior of $\mathcal{C}$ on $a_{i+1} \cdots a_k \lhd$ starting from any state. For this purpose we define $\overrightarrow{\mathcal{B}}_q$ by $\overrightarrow{S}_q = Q \times Q$, with initial state $(\delta(q, \rhd), \delta(q, \rhd))$ and $\delta_q^{\rightarrow}((s_1, s_2), a) = (\delta(s_1, a), \delta(s_1, \triangledown))$. In this way at letter i we have access to the information described in (1).

To define the machine $\overleftarrow{\mathcal{B}}_q$ we denote by Q^Q the set of mappings from Q to Q and by Id_Q the identity mapping on Q. We let $\overleftarrow{S}_q = \{\mathrm{Id}_Q\} \cup (Q^Q \times \Sigma)$ with Id_Q as initial state. Furthermore, we let $\delta_q^{\leftarrow}(\mathrm{Id}_Q, a) = (h_\lhd, a)$ with $h_\lhd : Q \rightarrow Q$ defined by $h_\lhd(q) = \delta(q, \lhd)$, and for $h \in Q^Q$ and $a, a' \in \Sigma$ we let $\delta_q^{\leftarrow}((h, a), a') = (h', a')$ with $h' : Q \rightarrow Q$ defined by $h'(q) = h(\delta(q, a))$. This provides the information described in (2). These two informations are then combined by the output function f_q of $\mathcal{B}_q$ as follows: $f_q((q_1, q_2), a, (h, b)) = \delta(h(q_2), a)$. □

Lemma 7. *If $T \subseteq \mathcal{T}_\Sigma$ is recognizable by a $\downarrow$DTA, then T is path closed.*

Proof. Let $\mathcal{A} = (Q, \Sigma, f_{\text{in}}, (\mathcal{B}_q)_{q\in Q}, F)$ be a $\downarrow$DTA recognizing T. Note that it is sufficient to show $\text{cl}(T) \subseteq T$ since the other inclusion always holds.

For $q \in Q$ and $a \in \Sigma$ let $\mathcal{A}_{a\to q} = (Q, \Sigma, f_{\text{in}}^{a\to q}, (\mathcal{B}_q)_{q\in Q}, F)$ with $f_{\text{in}}^{a\to q}(a) = q$ and $f_{\text{in}}^{a\to q}(b) = f_{\text{in}}(b)$ for $b \in \Sigma \setminus \{a\}$, and let $T_{a\to q}$ denote the language accepted by $\mathcal{A}_{a\to q}$. By induction on the height of t one shows that $\pi(t) \subseteq \pi(T_{a\to q})$ implies that $t \in T_{a\to q}$ for each $q \in Q$, $a \in \Sigma$, and $t \in \mathcal{T}_\Sigma^a$. Since $T_{a\to f_{\text{in}}(a)} = T$ we obtain the desired result that $t \in \text{cl}(T)$ (i.e., $\pi(t) \subseteq \pi(T)$) implies that $t \in T$. The details of the induction are left to the reader. $\square$

As an immediate consequence of the previous lemmas we get the following theorem.

Theorem 5. *Let $T \subseteq \mathcal{T}_\Sigma$ be regular. Then T is $\downarrow$DTA recognizable if and only if T is path closed.*

To obtain the desired decidability result we simply note that the construction from Lemma 6 is effective. Furthermore, language inclusion is decidable in exponential time for finite automata over unranked trees. Most easily, this can be seen by using an encoding of unranked trees by ranked trees [Suc02] and then using algorithms for ranked tree automata (cf. [CDG$^+$97]).

Theorem 6. *Given a regular language $T \subseteq \mathcal{T}_\Sigma$ it is decidable whether T can be recognized by a $\downarrow$DTA. Furthermore, such a $\downarrow$DTA can be effectively constructed.*

Let us address two restricted models of deterministic top-down tree automata that are as well natural, mutually incompatible in expressive power, and lead to completely analogous results for suitable adaptions of the notion of path language.

The first model was already indicated in the introduction. Precisely as for the case of deterministic top-down automata over ranked trees, one requires the automaton to assume states at the successor nodes of a tree node x solely on the basis of the label a at x, the state q assumed there, and the rank of a, i.e., the number n of successors, but independent of the labels of the successor nodes. This leads to a special model of bimachine where the input sequence is a word $\bullet^n$ rather than a label sequence $a_1 \dots a_n$. Accordingly, in the definition of path language we have to use between pairs $\triangleright, \triangleleft$ just the symbol $\bullet$ instead of the label letters, besides $\triangledown$ of course.

The second model is based on a left-to-right scanning process of successor labels; so a standard sequential machine (cf. [Eil74,Ber79]) is used to produce the successor states, hence without reference to the rank (the number n of the respective successor nodes altogether). In this case, our coding of paths has to be modified by canceling the segments between a symbol $\triangledown$ and the respective next $\triangleleft$ in order to obtain results analogous Theorems 5 and 6 above.

A similar concept has been used in [MNS05] in the context of typing streaming XML documents in a single pass. Recognizability by the second model from above (using sequential machines for labeling the successors), corresponds to definability by specialized DTDs with ancestor-sibling-based types in [MNS05].

5 Conclusion

In this paper we have extended the list of properties that carry over from tree automata for ranked trees to the unranked setting. We have shown that, using appropriate definitions, it is possible to minimize bottom-up deterministic tree automata over unranked trees in quadratic time. This minimization yields unique representatives for regular languages of unranked trees that can, e.g., be used to speed up equivalence tests. We have also transferred the characterization of deterministic top-down tree languages in terms of path languages and path closure from the ranked to the unranked case. This characterization can be used to decide for a given regular language of unranked trees whether it is top-down deterministic. A refinement of the minimization algorithm and a detailed analysis of the complexity, as well as the problem of minimizing deterministic top-down automata are subject of current and future research.

References

[BB02] J. Berstel and L. Boasson. Formal properties of XML grammars and languages. *Acta Informatica*, 38:649–671, 2002.

[Ber79] J. Berstel. *Transductions and Context-Free Languages.* Teubner, 1979.

[BWM01] A. Brüggemann-Klein, D. Wood, and M. Murata. Regular tree and regular hedge languages over unranked alphabets: Version 1. unfinished technical report, April 2001. `http://citeseer.ist.psu.edu/451005.html`.

[CDG$^+$97] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. *Tree Automata Techniques and Applications.* unpublished electronic book, 1997. `http://www.grappa.univ-lille3.fr/tata`.

[Eil74] S. Eilenberg. *Automata, languages, and machines*, volume A. Academic Press, 1974.

[GS84] F. Gécseg and M. Steinby. *Tree automata.* Akadémiai Kiadò, Budapest, 1984.

[Hop71] J. E. Hopcroft. An $n\log n$ algorithm for minimizing states in a finite automaton. *Theory of Machines and Computations*, pages 189–196, 1971.

[HU79] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison Wesley, 1979.

[MNS05] W. Martens, F. Neven, and Th. Schwentick. Which XML schemas admit 1-pass preorder typing? In *ICDT*, volume 3363 of *LNCS*, pages 68–82. Springer, 2005.

[MSV03] T. Milo, D. Suciu, and V. Vianu. Typechecking for XML transformers. *Journal of Computer and System Sciences*, 66(1):66–97, 2003.

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

[NS02] F. Neven and Th. Schwentick. Query automata on finite trees. *Theoretical Computer Science*, 275:633–674, 2002.

[Suc02] D. Suciu. Typechecking for semistructured data. In *Database Programming Languages, 8th International Workshop, DBPL 2001*, volume 2397 of *LNCS*, pages 1–20. Springer, 2002.

[Vir80] J. Virágh. Deterministic ascending tree automata I. *Acta Cybernet.*, 5:33–42, 1980.