Available online at www.sciencedirect.com

SCIENCE DIRECT®

Theoretical Computer Science 350 (2006) 164 – 173

Theoretical Computer Science

www.elsevier.com/locate/tcs

Tree-walking automata cannot be determinized

Mikołaj Bojańczyk*,[1], Thomas Colcombet[2]

Institute of Informatics, Warsaw University, Banacha 2, 02-097 Warsaw, Poland

Abstract

Tree-walking automata are a natural sequential model for recognizing languages of finite trees. Such automata walk around the tree and may decide in the end to accept it. It is shown that deterministic tree-walking automata are weaker than nondeterministic tree-walking automata.

Keywords: Tree-walking automata; Deterministic tree-walking automata

0. Introduction

A tree-walking automaton is a natural type of finite automaton over trees. The automaton is a finite memory device that walks around a tree, choosing what move to make according to its current state and the local environment (the label of the current node and whether this node is a left son, a right son, the root or a leaf). The automaton accepts the tree if it ever reaches one of the designated accepting states. Even though tree-walking was introduced in the early seventies by Aho and Ullman [1], very little is known about this model.

This situation is different in the case of the "usual" tree automata—branching tree automata—which are a very well understood object. Both top-down and bottom-up nondeterministic branching tree automata recognize the same class of languages. Languages of this class are called *regular*, the name being so chosen because it enjoys many nice properties of the class of regular word languages. In particular, this class is closed by all the boolean operations. The deterministic variants of branching tree automata are similarly well understood—deterministic bottom-up branching tree automata also recognize all regular tree languages, while deterministic top-down branching tree automata recognize a strict subclass of the class of regular languages.

It is a classical result that every language recognized by a tree-walking automaton is regular. It is also known that those languages are closed by intersection and union. However, most other fundamental questions pertaining to tree-walking automata remain unanswered:

(1) Is every regular language recognized by a tree-walking automaton?
(2) Can tree-walking automata be determinized?
(3) Is the class of languages recognized by tree-walking automata closed under complementation?

* Corresponding author.
E-mail addresses: bojan@mimuw.edu.pl (M. Bojańczyk), colcombet@mimuw.edu.pl (T. Colcombet).

[1] Supported by Polish KBN Grant no. 4 T11C 042 25.

[2] Supported by the European Community Research Training Network GAMES.

0304-3975/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2005.10.031

It is believed that the answers to all three questions above are negative. There has been much related research, which can be roughly grouped into two categories: nondefinability results for weakened models of tree-walking automata [8,3] and definability results for strengthened models of tree-walking automata [7,4,5]. The three questions stated above, however, have remained open.

In this paper we answer question (2): we prove that there exists a language that is recognized by a tree-walking automaton, but by no deterministic one.

1. Tree walking automata and the separating language

In this section we define tree-walking automata, specify our separating language and prove it is recognized by a nondeterministic tree-walking automaton.

1.1. Preliminaries

The trees we deal with in this paper are finite, binary trees labeled by a given finite alphabet Σ. A *Σ-tree* is a mapping from $N \subseteq \{0, 1\}^*$ to Σ, where N is a finite, nonempty, prefix-closed set such that for any $v \in N$, $v0 \in N$ iff $v1 \in N$. Elements of N are called *nodes* of the tree. The maximal elements are called *leaves*, the minimal element ε is called the *root*. Every node v of a tree t has a *type* belonging to Types $= \{o, l, r\} \times \{f, l\}$ where the first component says whether the node is the r*o*ot, a *l*eft son or a *r*ight son, and the second component says whether the node is a *l*eaf or the *f*ather of some other node. A *direction* is an element in Dir $= \{\uparrow, \varepsilon, 0, 1\}$, where $\uparrow$ stands for 'up', ε stands for 'stay', 0 stands for 'down left' and 1 for 'down right'.

Definition 1. A *(nondeterministic) tree-walking automaton* is a tuple $\mathcal{A} = (Q, \Sigma, I, F, \delta)$, where Q is a finite set of *states*, $I, F \subseteq Q$ are the *initial* and *accepting* states, and δ is the *transition relation* of the form

$$\delta \subseteq Q \times \text{Types} \times \Sigma \times Q \times \text{Dir}.$$

In a tree, a *configuration* is a pair of a node and a state. In a given configuration, the automaton looks at its current state, the type of the current node and its label; it then picks—according to δ—a new state along with a direction and moves according to this direction. A *run* is a sequence of configurations, where every two consecutive configurations are consistent with δ in the manner described above. A run is *accepting* if it starts and ends in the root of the tree, the first state being in I and the last state being in F. The automaton $\mathcal{A}$ accepts a tree if it has an accepting run over it. A set of Σ-trees L is *recognized* by $\mathcal{A}$ if $\mathcal{A}$ accepts exactly the trees in L. An automaton is *deterministic* if it has one initial state and the transition relation is a function from $Q \times$ Types $\times\, \Sigma$ to $Q \times$ Dir. We write TWA, DTWA for the classes of languages recognized, respectively, by nondeterministic and deterministic tree-walking automata.

We would like to point out here that reading the type of a node is an essential feature of tree-walking automata. Indeed, Kamimura and Slutzki showed in [7] that tree-walking automata that do not have access to this information cannot recognize all regular languages since they are unable of even searching in a tree in a systematic manner. In particular, such weaker tree-walking automata cannot perform depth-first searches.

Example A. The alphabet is $\{a, b\}$. Consider the following tree-walking automaton that accepts exactly the trees with at least one b. The states are p and q. The only initial state is p and the only accepting state is q. The automaton nondeterministically finds a node labeled by b in the tree using the state p and then changes to state q. The transitions of states q then allows the automaton to walk back to the root. Formally, the transitions are:

(p, t, a, p, d) for any $t \in \{o, l, r\} \times \{f\}$ and $d \in \{0, 1\}$;
$(p, t, b, q, \varepsilon)$ for any $t \in$ Types;
$(q, t, x, q, \uparrow)$ for any $t \in \{l, r\} \times \{f, l\}$, $x \in \{a, b\}$.

Example B. We now present a deterministic tree-walking automaton which also recognizes the language of the previous example. To make sure that it skips no b in its search, the automaton performs a depth-first search. The states are p, p', q and r. The only initial state is p and the only accepting state is r. The states p and q are used in a depth-first search

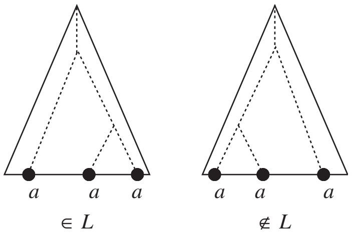

Fig. 1. The two kinds of well-formed trees.

from left to right, and p' is used in this search as an intermediate state. The state r is used to walk back to the root. The transition relation is:

$(p, t, a, p, 0)$ for any $t \in \{o, l, r\} \times \{f\}$;
$(p, t, a, q, \varepsilon)$ for any $t \in \{o, l, r\} \times \{l\}$;
$(q, t, a, p', \uparrow)$ for any $t \in \{l, r\} \times \{f, l\}$;
$(p', t, a, p, 1)$ for any $t \in \{o, l, r\} \times \{f\}$;
$(p, t, b, r, \varepsilon)$ for any $t \in$ Types;
$(r, t, x, r, \uparrow)$ for any $t \in \{l, r\} \times \{f, l\}$ and $x \in \{a, b\}$.

The above examples show that even when tree-walking automata can be determinized, the determinized automaton may need to walk around the tree in a completely different way.

1.2. The separating language L

In this section we specify our separating language L, which witnesses the strictness of the inequality DTWA $\subsetneq$ TWA. We also present a nondeterministic tree-walking automaton which recognizes L. Our proof that no deterministic tree-walking automata can recognize L is more involved and will be spread across the subsequent sections.

The language L involves a very simple kind of trees, which we call *well-formed* trees: $\{B, a\}$-trees that have all nodes labeled by the *blank symbol B* but for three leaves that are labeled by a. The set of well-formed trees can be recognized by a deterministic tree-walking automaton.

There are two possible kinds of well-formed trees: ones where the deepest common ancestor of the two leftmost a's is above the rightmost a; and the other ones. The separating language L is defined to be the set of well-formed trees of the first kind. This definition is illustrated in Fig. 1.

Lemma 2. *The language L is recognized by a nondeterministic tree-walking automaton.*

Proof. We will only give here an informal description of the automaton. This automaton first checks that the tree is well-formed, then goes to the rightmost a. This can be done without using nondeterminism. From this node, it goes toward the root and chooses nondeterministically some node v. It then accepts the tree if there are exactly two a's that are to the right of the leftmost leaf below v. A depth-first search from left to right starting at position v can perform this verification.

One can verify that there exists an accepting run of this automaton if and only if the tree belongs to L. Indeed, when the tree belongs to L the automaton chooses v to be the deepest common ancestor of the two rightmost a's. On the other hand, if a tree is well-formed but does not belong to L, for every ancestor v of the rightmost a, there are either one or three a's to the right of the leftmost leaf below v. □

2. Patterns

In this section, we introduce the key technical concept of patterns and outline how they can be used to show that no deterministic tree-walking automaton recognizes L. A concept similar to our patterns was used in [2] to analyze automata on a two-dimensional tape.

From now on we fix a deterministic tree-walking automaton

$$\mathcal{A} = (Q, \{q_I\}, F, \delta).$$

We will prove that $\mathcal{A}$ does not recognize the language L.

2.1. Patterns and pattern equivalence

A *pattern* Δ is a $\{B, *\}$-tree where the symbol $*$ is used solely in the leaves. For technical reasons, we require the leaves labeled with $*$ to be left sons. The ith $*$-labeled leaf (numbered from left to right, starting with 0) is called the ith-(*leaf*) *port*. *Port* ε is the root. The *arity* of the pattern is the number of leaf ports. We use Pat^n to denote the set of n-ary patterns.

Given an n-ary pattern Δ and n patterns $\Delta_0, \ldots, \Delta_{n-1}$, the *composition* $\Delta[\Delta_0, \ldots, \Delta_{n-1}]$ is obtained from Δ by simultaneously substituting each Δ_i for the ith port. We may use $*$ instead of some substituted patterns in a composition, the intended meaning being that the corresponding ports remain untouched. When all Δ_i's are $*$ but for Δ_k we simply write $\Delta[\Delta_k/k]$. If Δ is a unary pattern, we write $\Delta \cdot \Delta'$ instead of $\Delta[\Delta']$. Given a set P of patterns, we denote by $\mathcal{C}(P)$ the least set of patterns that contains P and is closed under composition.

Definition 3. The *automaton's transition relation over an n-ary pattern* Δ,

$$\delta_\Delta \subseteq Q \times \{\varepsilon, 0, \ldots, n-1\} \times Q \times \{\varepsilon, 0, \ldots, n-1\},$$

contains a tuple (q, i, r, j) if the automaton can go from state q in port i to state r in port j, without visiting any port along the way. Moreover, the ports are treated as having type (l, f), i.e. non-leaf left sons.

The last clause in the above definition is postulated in order to make composition work. In particular, the port ε is not seen as the root and the leaf ports are not seen as leaves.

From the point of view of the automaton, the transition relation sums up all important properties of the pattern and we consider two patterns equivalent if they induce the same relation. More precisely, for two patterns Δ and Δ' of same arity, we write

$$\Delta \simeq \Delta' \quad \text{iff } \delta_\Delta = \delta_{\Delta'}.$$

The essence of this equivalence is that if one replaces a sub-pattern by an equivalent one, the automaton is unable to see the difference. This is summarized by the following fact:

Fact 4. *The relation $\simeq$ is a congruence with respect to pattern composition.*

2.2. Outline of the proof

In order to prove that $\mathcal{A}$ cannot recognize L, we will produce three patterns: a nullary pattern Δ_0, a unary pattern Δ_1 and a binary pattern Δ_2. We then prove that compositions of these patterns satisfy several desirable properties. In particular, the following equivalence holds:

$$\Delta_2[*, \Delta_2] \simeq \Delta_2[\Delta_2, *]. \tag{1}$$

Having this equivalence, proving that $\mathcal{A}$ does not recognize L becomes a simple matter. Consider a context where a B-labeled tree is attached to the root port, and to each leaf port is attached a tree with exactly one a. If we insert the left pattern from (1) into this context, we obtain a tree in L, and if we insert the right pattern, we obtain a tree outside L. However, since the patterns are equivalent, the automaton L cannot distinguish the two resulting trees and will either accept both or reject both, hence $\mathcal{A}$ does not recognize L.

Since the deterministic automaton $\mathcal{A}$ was chosen arbitrarily, it follows that $L \notin$ DTWA. Together with Lemma 2, we obtain this paper's contribution:

Theorem 5. *The class DTWA is strictly included in the class TWA.*

It remains to construct the patterns Δ_0, Δ_1 and Δ_2, which we do in Section 3; and then study properties of those patterns using the determinism of $\mathcal{A}$, which we do in Section 4. The culmination of this study is Lemma 18, which establishes equivalence (1).

3. Basic patterns

In this section, we define the patterns Δ_0, Δ_1 and Δ_2 and prove basic properties related to their composition, namely Lemmas 8–10.

We will use a result concerning finite semigroups. Recall that a *semigroup* is a set together with an associative binary operation, which we write multiplicatively here. The following can be easily shown using Greene's relations (see [6] for an introductory text); for completeness, however, we provide an elementary proof of the result.

Lemma 6. *For every finite semigroup S and any $u, v \in S$, there exist $u', v' \in S$ such that the elements $U = u \cdot u'$ and $V = v \cdot v'$ satisfy the following equations*:

$$U = U \cdot U = U \cdot V \quad \text{and} \quad V = V \cdot U = V \cdot V.$$

Proof. For $s \in S$, we write Ss for the set of elements of the form $t \cdot s$, with $t \in S$. An element s of S is said *idempotent* if $s \cdot s = s$. If we fix N to be the factorial of $|S|$, a classical result states that for any $s \in S$, s^N is idempotent.

Consider now the s_i's and t_i's defined inductively as follows:

$$s_1 := u^N, \quad t_i := (v \cdot s_i)^N \quad \text{and} \quad s_{i+1} := (u \cdot t_i)^N.$$

Clearly, every s_i is of the form $u \cdot u'$ for some u' and every t_i is of the form $v \cdot v'$ for some v'. Moreover, for any $i < j$ we have

$$s_j, t_j \in St_i \cap Ss_i.$$

Since S is finite, for some integers $n < m$ both $s_n = s_m$ and $t_n = t_m$ hold. Let us fix $U = s_n$ and $V = t_n$. By construction, U and V are idempotent, hence we only need to verify $U \cdot V = U$ and $V \cdot U = V$. Let us show $U \cdot V = V$:

$$U \cdot V = s_n \cdot t_n = s_m \cdot t_n = s_m = U.$$

The third equation follows from the fact that s_m belongs to St_n and t_n is idempotent. The proof of the equation $V \cdot U = V$ is similar. □

This lemma will be used in the construction of the patterns Δ_0, Δ_1 and Δ_2. The insightful reader will notice that this construction does not involve the determinism of $\mathcal{A}$.

Let us denote by B_k the full binary tree of depth k where all nodes are labeled by B. As the pattern equivalence relation $\simeq$ is of finite index, there exist m, n such that $B_n \simeq B_m$ and B_n appears at least twice in B_m as a subtree rooted in a left son. Formally, there exists a binary pattern Δ_X such that $\Delta_X[B_n, B_n] = B_m$. Let us fix Δ_0 to be B_n and consider the following patterns:

$$\Delta_u = \Delta_X[*, \Delta_0] \quad \text{and} \quad \Delta_v = \Delta_X[\Delta_0, *].$$

The following fact is proved by an obvious induction on the structure of Δ:

Fact 7. *Every pattern $\Delta \in \mathcal{C}(\{\Delta_u, \Delta_v\})$ satisfies $\Delta \cdot \Delta_0 \simeq \Delta_0$.*

Let S be the semigroup whose elements are patterns in $\mathcal{C}(\{\Delta_u, \Delta_v\})$ and where the multiplication operation is the composition of unary patterns. By Lemma 4 the equivalence $\simeq$ is a congruence over S. Since furthermore S is finite modulo $\simeq$, by Lemma 6 there exists unary patterns $\Delta_{u'}$ and $\Delta_{v'}$ in $\mathcal{C}(\{\Delta_u, \Delta_v\})$ such that the two patterns $\Delta_U = \Delta_u \cdot \Delta_{u'}$ and $\Delta_V = \Delta_v \cdot \Delta_{v'}$ satisfy the following equivalences:

$$\Delta_U \simeq \Delta_U \cdot \Delta_U \simeq \Delta_U \cdot \Delta_V \quad \text{and} \quad \Delta_V \simeq \Delta_V \cdot \Delta_U \simeq \Delta_V \cdot \Delta_V. \tag{2}$$

Let us finally set Δ_1 to be Δ_U and Δ_2 to be $\Delta_1 \cdot \Delta_X[\Delta_{u'} \cdot \Delta_1, \Delta_{v'} \cdot \Delta_1]$. We write $\mathcal{C}_\mathcal{A}$ for the set $\mathcal{C}(\{\Delta_0, \Delta_1, \Delta_2\})$ and $\mathcal{C}^n_\mathcal{A}$ for the set $\mathcal{C}_\mathcal{A} \cap \mathrm{Pat}^n$.

The following lemma shows that plugging a Δ_1 pattern next to any port of a pattern in $\mathcal{C}_\mathcal{A}$ leads to an equivalent pattern.

Lemma 8. *Let Δ be an n-ary pattern in $\mathcal{C}_\mathcal{A}$. All the patterns $\Delta_1 \cdot \Delta$ and $\Delta[\Delta_1/0], \dots, \Delta[\Delta_1/n-1]$ are equivalent to Δ.*

Proof. The case of $\Delta = \Delta_0$ follows from Fact 7. The case of $\Delta = \Delta_1$ follows from (2). The remaining cases follow from (2) and the fact that every pattern of arity at least two has a Δ_1 pattern next to each port. □

The following lemma shows that, from the point of view of the automaton $\mathcal{A}$, all patterns of a given small arity in $\mathcal{C}_\mathcal{A}$ look the same:

Lemma 9. *For $k = 0, 1, 2$ all patterns Δ in $\mathcal{C}^k_\mathcal{A}$ are equivalent to Δ_k.*

Proof. We prove first that $\Delta_2[\Delta_0, *] \simeq \Delta_1$:

$$\begin{aligned}
\Delta_2[\Delta_0, *] &= \Delta_1 \cdot \Delta_X[\Delta_{u'} \cdot \Delta_1 \cdot \Delta_0, \Delta_{v'} \cdot \Delta_1] \\
&\simeq \Delta_1 \cdot \Delta_X[\Delta_0, \Delta_{v'} \cdot \Delta_1] \\
&\simeq \Delta_1 \cdot \Delta_v \cdot \Delta_{v'} \cdot \Delta_1 \\
&= \Delta_U \cdot \Delta_V \cdot \Delta_U \\
&\simeq \Delta_U = \Delta_1.
\end{aligned}$$

Symmetrically, we get $\Delta_2[*, \Delta_0] \simeq \Delta_1$.

The lemma is then proved by induction on the structure of the pattern Δ. This pattern has Δ_0, Δ_1 or Δ_2 at root, and the subpatterns have arity at most two. By induction, each subpattern is equivalent to one of Δ_0, Δ_1 or Δ_2. The conclusion of the lemma follows by one of the two equivalences we just proved, or by Lemma 8. □

3.1. Application to run analysis

We now use Lemmas 8 and 9 to exhibit some more properties of the patterns.

The following result shows that entering and exiting a pattern by the same port do not really depend on the pattern. This result, apart from its importance in the remainder of the proof, is also a good illustration of proof techniques used afterward.

Lemma 10. *For all patterns $\Delta \in \mathcal{C}_\mathcal{A}$ of nonzero arity and all states q, r:*

$$\begin{aligned}
&(q, \varepsilon, r, \varepsilon) \in \delta_\Delta \quad \textit{iff } (q, \varepsilon, r, \varepsilon) \in \delta_{\Delta_1}, \\
&(q, i, r, i) \in \delta_\Delta \quad \textit{iff } (q, 0, r, 0) \in \delta_{\Delta_1} \textit{ for } i \in \{0, \dots, n-1\}.
\end{aligned}$$

Proof. Let Δ be a pattern of arity $n \geqslant 1$ and suppose $(q, \varepsilon, r, \varepsilon)$ belongs to δ_Δ. The corresponding run does not visit any leaf ports and can therefore still be used no matter what is plugged into them. In particular, the same run can still be used in the pattern

$$\Delta[\overbrace{\Delta_0, \dots, \Delta_0}^{n-1 \text{ times}}, *].$$

Since this pattern is equivalent to Δ_1 by Lemma 9, we conclude that $(q, \varepsilon, r, \varepsilon)$ belongs to δ_{Δ_1}.

Let us now suppose that $(q, \varepsilon, r, \varepsilon)$ belongs to δ_{Δ_1}. By the same reasoning as above, the corresponding run can still be used in $\delta_{\Delta_1} \cdot \Delta$. By Lemma 8, this pattern is equivalent to Δ. Therefore $(q, \varepsilon, r, \varepsilon)$ belongs to δ_Δ.

The case of leaf ports is shown analogously. Let i be a leaf port of the pattern Δ and let (q, i, r, i) belong to δ_Δ. By Lemma 8, we have

$$\Delta_1 \simeq \Delta_i[\overbrace{\Delta_0, \dots, \Delta_0}^{i-1 \text{ times}}, *, \overbrace{\Delta_0, \dots, \Delta_0}^{n-i \text{ times}}].$$

Since the run that went from (q, i) to (r, i) in Δ will also work in the above pattern, we obtain that $(q, 0, r, 0)$ belongs to δ_{Δ_1}. The reverse implication is proved analogously using the equivalence $\Delta \simeq \Delta[\Delta_1/i]$. □

4. Removing oscillation

From now on, we will be using the fact that the automaton $\mathcal{A}$ is deterministic. A consequence of the determinism of $\mathcal{A}$ is that for any pattern Δ of arity n, the relation δ_Δ is a partial function

$$\delta_\Delta : Q \times \{\varepsilon, 0, \dots, n-1\} \to Q \times \{\varepsilon, 0, \dots, n-1\}.$$

This function may be partial even if the original transition function was not, since the automaton can be trapped in a loop inside the pattern. From now on we use a functional notation for δ relations.

One sort of behavior that makes notation more cumbersome is what we call oscillation. This is when the automaton comes back to the same port in the pattern from which it entered. To suppress such behavior, we replace in this section the δ transition relation with a new one, called the γ function.

Consider a unary pattern of the form $\Delta_1 \cdot \Delta_1$, with the nodes $v < w$ corresponding to the leaf ports of the two Δ_1 patterns. For a state q, consider the unique maximal run of $\mathcal{A}$ which starts in (q, v) and visits none of the ports of $\Delta_1 \cdot \Delta_1$. If this run visits v a finite number of times, the *ε-successor* $s_\varepsilon(q)$ of q is defined to be the state in which v is last visited. If the run loops around v, the ε-successor is undefined.

Notice that the *ε-successor* is defined in terms of Δ_1 patterns, but thanks to Lemma 10, we could have used any patterns of non-zero arity in place of both Δ_1 patterns. The ε-successor function describes the loops possible at the junction node between two patterns of non-zero arity.

We say that a state q is an *upward* state if it appears *after* $\mathcal{A}$ has traversed a pattern Δ_1 in the up direction, i.e. $\delta_{\Delta_1}(r, 0) = (q, \varepsilon)$ holds for some state r. Similarly, we say that q is a *downward* state if $\delta_{\Delta_1}(r, \varepsilon) = (q, 0)$ holds for some state r. We use Q_U and Q_D to denote the sets of upward and downward states, respectively.

We now introduce a new type of transition function that we will use instead of the δ functions. This new function is meant to eliminate oscillation of the automaton. This step can be interpreted as the construction of a non-oscillating version of the original automaton $\mathcal{A}$. For $\Delta \in \mathcal{C}^n_\mathcal{A}$, the partial function

$$\gamma_\Delta : (Q_\mathrm{D} \times \{\varepsilon\}) \cup (Q_\mathrm{U} \times \{0, \dots, n-1\}) \to (Q_\mathrm{U} \times \{\varepsilon\}) \cup (Q_\mathrm{D} \times \{0, \dots, n-1\})$$

is defined by $\gamma_\Delta(q, i) = \delta_\Delta(s_\varepsilon(q), i)$. From now on, we simplify slightly the notation by using γ_0, γ_1 and γ_2 for γ_{Δ_0}, γ_{Δ_1} and γ_{Δ_2}, respectively.

We remark here, somewhat ahead of time, that the function γ_Δ turns out to be completely defined for any pattern $\Delta \in \mathcal{C}_\mathcal{A}$. This is because—thanks to the choice of the function's domain—we can be sure that the automaton does not loop. The formal proof is provided by Lemma 14.

Lemma 11. *For $q \in Q_\mathrm{D}$, $\gamma_1(q, \varepsilon) = (q, 0)$. For $q \in Q_\mathrm{U}$, $\gamma_1(q, 0) = (q, \varepsilon)$.*

Proof. Let $q \in Q_\mathrm{D}$. By definition, there is some r such that $\delta_{\Delta_1}(r, \varepsilon) = (q, 0)$. Consider the pattern $\Delta_1 \cdot \Delta_1$, with v labeling the interface between the two Δ_1 patterns and a run on this pattern which starts in (r, ε). The first time the node v is passed, state q is assumed, since $\delta_{\Delta_1}(r, \varepsilon) = (q, 0)$. The last time v is passed, state $s_\varepsilon(q)$ is assumed, by definition of s_ε. The first time the leaf port is reached, state q is assumed, since $\delta_{\Delta_1 \cdot \Delta_1}(r, \varepsilon) = \delta_{\Delta_1}(r, \varepsilon) = (q, 0)$. Hence $\gamma_1(q, \varepsilon) = \delta_{\Delta_1}(s_\varepsilon(q), \varepsilon) = (q, 0)$.

The proof for $q \in Q_\mathrm{U}$ is obtained by swapping the roles of ports ε and 0. □

The next lemma shows that γ admits no oscillation:

Lemma 12. *For any $\Delta \in \mathcal{C}_{\mathcal{A}}$ of arity $n \geqslant 1$, states q, r and any port i,*

$$\text{if } \gamma_\Delta(q, i) = (r, j) \quad \text{then } i \neq j.$$

Proof. We only treat the case where q is a downward state and $i = \varepsilon$. Let p be the ε-successor of q. Were the statement in the lemma false, we would have $j = \varepsilon$. This would mean that $\delta_\Delta(p, \varepsilon) = (r, \varepsilon)$. By Lemma 10, $\delta_{\Delta_1}(p, \varepsilon) = (r, \varepsilon)$ holds, a contradiction with the definition of p as an ε-successor. □

Here follows a simple description of the behavior of downward states in a Δ_2 pattern.

Lemma 13. *For a downward state q, $\gamma_2(q, \varepsilon)$ is either $(q, 0)$ or $(q, 1)$.*

Proof. Let $\gamma_2(q, \varepsilon) = (r, i)$. By Lemma 12, i is either 0 or 1. Without loss of generality we assume that $i = 0$. Since $\Delta_1 \simeq \Delta_2[*, \Delta_0]$, we obtain that $\gamma_1(q, \varepsilon) = (r, 0)$. This, together with Lemma 11, shows that $r = q$. □

Lemma 14. *The function γ_Δ is completely defined for patterns $\Delta \in \mathcal{C}_{\mathcal{A}}$.*

Proof. Consider a pattern Δ of nonzero arity n and assume that $\gamma_\Delta(q, i)$ is undefined for some port i. Let us consider first the case when i is a leaf port. If we plug all the other leaf ports of Δ with a Δ_0 pattern, we get a pattern equivalent to Δ_1. The same run that either looped or blocked in Δ would do the same in the new pattern. But this implies that $\gamma_1(q, 0)$ is undefined, a contradiction with Lemma 11. The case when i is the root port is proved the same way.

For the case of $\Delta = \Delta_0$, let us assume for a moment that when entering from (q, ε), the automaton gets lost in the pattern. But then, by the equivalences $\Delta_1 \simeq \Delta_2[*, \Delta_0] \simeq \Delta_2[\Delta_0, *]$ and by Lemma 13, $\mathcal{A}$ would also get lost in Δ_1 when entering from (q, ε), a contradiction with Lemma 11. □

Finally, the following lemma shows that in order to establish the equivalence of two patterns, it is enough to study the γ functions.

Lemma 15. *Two patterns in $\mathcal{C}_{\mathcal{A}}$ are equivalent if and only if they have the same γ functions.*

Proof. The left to right implication follows straight from the definition of the function γ, which is defined in terms of the δ function.

For the right to left implication, take two patterns Δ, Δ' with the same γ functions and consider a possible argument (q, i) of the functions δ_Δ, $\delta_{\Delta'}$. We only do the case of $i = \varepsilon$, the case for leaf ports being analogous. Let q be a downward state. If $\delta_{\Delta_1}(q, \varepsilon)$ is either undefined or (r, ε), then the same happens in δ_Δ and $\delta_{\Delta'}$, since the patterns Δ, $\Delta_1 \cdot \Delta$ and Δ', $\Delta_1 \cdot \Delta'$ are all equivalent. Otherwise, q is its own ε-successor and in this case

$$\delta_\Delta(q, \varepsilon) = \delta_\Delta(s_\varepsilon(q), \varepsilon) = \gamma_\Delta(q, \varepsilon) = \gamma_{\Delta'}(q, \varepsilon) = \delta_{\Delta'}(s_\varepsilon(q), \varepsilon) = \delta_{\Delta'}(q, \varepsilon). \qquad \square$$

From now on, the γ function is used instead of the δ function.

5. Depth-first search

In this last part we show that—from the point of view of the γ function—the automaton can only do depth-first searches over patterns in $\mathcal{C}_{\mathcal{A}}$. Using this characterization, we prove the main technical lemma of this paper, Lemma 18. Due to the domain of γ we need to consider two cases: downward states in the root port and upward states in the leaf ports.

We already have a good description of the behavior of downward states when entering in the root port. This was the subject of Lemma 13.

The behavior of upward states is more involved. When starting in a leaf port, an upward state may go in the direction of the root, but it may also try to visit some other leaf port (a case that has no equivalence for downward states). The following definition, along with Lemma 17, gives a classification of the possible behaviors of upward states.

Definition 16. Let q be a downward state and r an upward state. We say the pair (q, r) has *left to right depth-first search behavior* if

$$\gamma_2(q, \varepsilon) = (q, 0), \quad \gamma_0(q, \varepsilon) = (r, \varepsilon), \quad \gamma_2(r, 0) = (q, 1) \quad \text{and} \quad \gamma_2(r, 1) = (r, \varepsilon).$$

A *right to left depth-first search behavior* is defined symmetrically by swapping ports 0 and 1. An upward state r has *ascending behavior* if

$$\gamma_2(r, 0) = \gamma_2(r, 1) = (r, \varepsilon).$$

The following lemma shows that Definition 16 is exhaustive.

Lemma 17. *An upward state r either has ascending behavior or there exists a downward state q such that the pair (q, r) has depth-first search behavior (either left to right or right to left).*

Proof. Let r be an upward state. The proof is by cases as to which one of the equalities $\gamma_2(r, 0) = (r, \varepsilon)$ or $\gamma_2(r, 1) = (r, \varepsilon)$ hold. If both hold, r has ascending behavior. We show that if the first does not hold, then for some downward state q, the pair (q, r) has left to right depth-first search behavior. Symmetrically, if the second equality does not hold, there is some downward state q such that the pair (q, r) has right to left depth-first search behavior.

We only do the case where $\gamma_2(r, 0) \neq (r, \varepsilon)$. By Lemma 12 we have

$$\gamma_2(r, 0) = (q, 1) \tag{3}$$

for some downward state q. Let p be the upward state such that

$$\gamma_0(q, \varepsilon) = (p, \varepsilon). \tag{4}$$

Since $\Delta_1 \simeq \Delta_2[*, \Delta_0]$ and $\gamma_1(r, 0) = (r, \varepsilon)$ (Lemma 11), we obtain that

$$\gamma_2(p, 1) = (r, \varepsilon), \tag{5}$$

and hence $\gamma_1(p, 0) = (r, \varepsilon)$. Since $\gamma_1(p, 0) = (p, \varepsilon)$ holds by Lemma 11, we obtain $r = p$.

Since q is a downward state, then by Lemma 13 the value of $\gamma_2(q, \varepsilon)$ must be either $(q, 0)$ or $(q, 1)$. But the second case cannot hold, since together with Eqs. (4) and (5) this would mean that there is a loop $\gamma_{\Delta_2[*,\Delta_0]}(q, \varepsilon) = (r, \varepsilon)$, a contradiction with Lemma 12. This means that

$$\gamma_2(q, \varepsilon) = (q, 0). \tag{6}$$

The equality $p = r$ and Eqs. (3)–(6) show that (q, r) has left to right depth-first search behavior. □

Now that we know exactly how the automaton behaves for upward and downward states, we obtain the following as consequence of Lemmas 13 and 17:

Lemma 18. *The patterns $\Delta = \Delta_2[\Delta_2, *]$ and $\Delta' = \Delta_2[*, \Delta_2]$ are equivalent.*

Proof. We will show that the two patterns have the same γ functions.

For q a downward state, by Lemma 13, we have

$$\gamma_\Delta(q, \varepsilon) = \gamma_{\Delta'}(q, \varepsilon) = (q, 0) \quad \text{or} \quad \gamma_\Delta(q, \varepsilon) = \gamma_{\Delta'}(q, \varepsilon) = (q, 2).$$

Let r be an upward state and let i be a leaf port. By Lemma 17, the state either has ascending behavior or depth-first search behavior. If it has ascending behavior, then

$$\gamma_\Delta(r, i) = \gamma_{\Delta'}(r, i) = (r, \varepsilon).$$

If the pair (q, r) has left to right depth-first search behavior, then

$$\gamma_\Delta(r, i) = \gamma_{\Delta'}(r, i) = \begin{cases} (q, i+1) & \text{for } i \in \{0, 1\}, \\ (r, \varepsilon) & \text{if } i = 2. \end{cases}$$

The case of right to left depth-first search behavior is analogous. □

Notice that the above lemma implies that any two patterns of same arity in $\mathcal{C}_\mathcal{A}$ are equivalent. This is because each two patterns of the same arity can be rewritten into one another using the equivalences from Lemmas 9 and 18.

6. Conclusion

We have established that tree-walking automata cannot be determinized. This result also implies that deterministic tree-walking automata do not recognize all regular tree languages. Both results are new.

The proof was obtained by showing that a deterministic tree-walking automaton is unable to inspect the branching structure of a tree. It would be nice if variants of this approach could be applied to other languages. One interesting example is the majority election language that has been conjectured in [8] to separate tree-walking automata from branching automata. This language is constructed of ternary trees whose leaves are labeled either by 0 or by 1. A tree belongs to this language if it consists of a single leaf labeled by 1, or if at least two of its three subtrees belong to the language. We do not know whether our technique can be applied to show that this language is not accepted by any deterministic tree-walking automaton.

An important tool used by our proof is the definition of the patterns Δ_0, Δ_1 and Δ_2. These patterns are an alternative to the classical pumping argument. As we already mentioned, some of the properties concerning these patterns are still valid in the nondeterministic case. This suggests that the patterns may be useful in the natural continuation of this work: showing that nondeterministic tree-walking automata are weaker than branching tree automata.

References

[1] A.V. Aho, J.D. Ullman, Translations on a context-free grammar, Inform. and Control 19 (5) (1971) 439–475.

[2] M. Blum, C. Hewitt, Automata on a 2-dimensional tape, in: Symp. Switching and Automata Theory, 1967, pp. 155–160.

[3] M. Bojańczyk, 1-bounded TWA cannot be determinized, in: 23rd Conf. Foundations of Software Technology and Theoretical Computer Science, FSTTCS'03, Lecture Notes in Computer Science, Vol. 2914, Springer, Berlin, 2003, pp. 62, 73.

[4] J. Engelfriet, H.J. Hoogeboom, Tree-walking pebble automata, in: G. Paum, J. Karhumaki, H. Maurer, G. Rozenberg (Eds.), Jewels Are Forever, Contributions to Theoretical Computer Science in Honor of Arto Salomaa, Springer, Berlin, 1999, pp. 72–83.

[5] J. Engelfriet, H.J. Hoogeboom, J.-P. van Best, Trips on trees, Acta Cybernetica 14 (1999) 51–64.

[6] J.M. Howie, Fundamentals of Semigroup Theory, Oxford University Press, Oxford, 1996.

[7] T. Kamimura, G. Slutzki, Parallel two-way automata on directed ordered acyclic graphs, Inform. and Control 49 (1) (1981) 10–51.

[8] F. Neven, T. Schwentick, On the power of tree-walking automata, in: 27th Internat. Colloq. Automata, Languages and Programming, ICALP'00, Lecture Notes in Computer Science, Vol. 1853, 2000.