1-bounded TWA Cannot Be Determinized

Mikołaj Bojańczyk[*]

Warsaw University

Abstract. Tree-walking automata are a natural sequential model for recognizing tree languages which is closely connected to XML. Two long standing conjectures say TWA cannot recognize all regular languages and that deterministic TWA are weaker than non-deterministic TWA. We consider a weaker model, 1-TWA, of tree walking automata that can make only one pass through a tree. We show that deterministic 1-TWA are weaker than 1-TWA. We also show a very simple language not recognized by 1-TWA.

1 Introduction

Tree-walking Automata (TWA for short) are a natural sequential model for recognizing tree languages. Originally introduced by Aho and Ullman in 1971 [1], they have of late undergone something of a revival in a research program initiated by Engelfriet et al. [4, 5]. TWA have also been gathering interest thanks to the advent of XML (see [7, 2], or the survey [8]).

A TWA is similar to a finite word automaton. In any given moment it is positioned in a single vertex of the tree assuming one of a finite number of states. Based on this state and the label together with some other information about the current vertex, the automaton can change its state and move to some neighboring vertex.

The exact nature of this other information varies from definition to definition. Kamimura and Slutzki [6] showed that if this does not include the child number of the current vertex, it cannot even search the tree in a systematic way, such as doing a depth-first search. The now standard model assumes a TWA knows whether the current vertex is the root, a leaf, a left son or a right son.

Albeit blessed with a long history, very little is known about TWA. One can easily prove that TWA-recognizable tree languages are regular, i. e. can be recognized by any one of several equivalent models of (branching) tree automata. However, many of the most fundamental questions, as posed in [8], still remain open:

1. Do TWA capture the regular tree languages?
2. Are deterministic TWA as expressive as TWA?

[*] This research project was supported by Komitet Badań Naukowych, grant 4 T11C 042 25 for the years 2003-2006 and the European Community Research Training Network "Games and Automata for Synthesis and Validation"

3. Are TWA closed under complementation?

The only progress made to date regarding these questions was by Neven and Schwentick, who in [9] presented a regular language that cannot be recognized by any one bounded TWA (1-TWA for short), i. e. one that traverses every edge at most once in each direction.

Our main contribution is a proof that one bounded deterministic TWA (1-DTWA) do not recognize the same languages as 1-TWA. The language we use is the set L_1 of $\{\mathbf{a},\mathbf{b},\mathbf{c}\}$-labeled trees in which there exists an occurrence of the letter **b** below some occurrence of the letter **a** (or, equivalently, denoted by the expression //**a**//**b** in XPath). Intuitively, we show that a 1-DTWA cannot keep track of whether or not it is below some **a** vertex.

We are able to show this directly, using monoid combinatorics, only if $\mathcal{A}$ is a depth-first search (DFS) automaton. A DFS automaton is a 1-DTWA that visits vertices according to the lexicographical ordering. Such an automaton can, for instance, recognize the set of $\{\wedge,\vee,0,1\}$-labeled trees that correspond to well-formed boolean expressions evaluating to 1.

After showing that no DFS automaton can recognize L_1, we prove that DFS-like behavior can be forced in an arbitrary 1-DTWA. Thus we obtain:

Theorem 1
No deterministic 1-TWA recognizes L_1, but there is a nondeterministic 1-TWA recognizing L_1.

Next we consider the language L_2 of $\{\mathbf{a},\mathbf{b},\mathbf{c}\}$-labeled trees where below some occurrence of **a** there exist *two* occurrences of **b** which are not on the same branch. Using similar monoid combinatorics as in the case of L_1, we show that L_2 cannot be recognized by any 1-TWA. This is similar to the result of Neven and Schwentick mentioned above.

We think, however, that our example has two relative merits: first, the proof is quite straightforward, and second, our language L_2 has very simple logical properties – for instance it can be defined using a CTL [3] formula. Thus proving our conjecture that L_2 is not recognized by arbitrary tree walking automata would show how severely limited TWA are.

Our language L_1 on the one hand, and the aforementioned language of Neven and Schwentick as well as our L_2 on the other, show that the following inequalities hold (where REG is the class of regular languages):

$$\text{1-DTWA} \subsetneq \text{1-TWA} \subsetneq \text{REG}$$

2 Tree Walking Automata

In the paper we will be dealing with finite, binary, labeled trees. Let A be a finite prefix-closed subset of $\{0,1\}^*$ such that $v\cdot 0 \in A \Leftrightarrow v \cdot 1 \in A$ and let Σ be some finite set. A *Σ-tree* is any function $t : A \to \Sigma$. The *domain* $\mathrm{dom}(t)$ of the tree t is the set A. A *vertex of t* is any element $v \in \mathrm{dom}(t)$. For vertices we use the prefix ordering: $v \leq w$ if $w = v \cdot v'$ for some $v' \in \Sigma^*$. We denote the set of Σ-labeled trees by trees(Σ). Given $t, t' \in$ trees(Σ) and $\sigma \in \Sigma$, let (t, σ, t') denote the tree that has σ in the root, whose left subtree is t and whose right subtree is t'.

A *tree with a hole* $t[]$ is a tree t with some distinguished leaf $v \in \text{dom}(t)$. Given another tree s, the tree $t[s]$ obtained from t by substituting the tree s for the leaf v. We distinguish the tree with a hole $[]$, which has the hole in the root. Similarly we define a *tree with two holes* $t[,]$; we assume the first hole is to the left of the second one.

A *tree walking automaton* is a tuple $\langle Q, \Sigma, \delta, q_I, F \rangle$, where Q is the set of *states*, Σ is the *alphabet*, δ is the *transition function*, $q_I \in Q$ the *initial state* and $F \subseteq Q$ the set of *accepting states*. The function δ is of the form:

$$\delta : \{\downarrow, \uparrow_0, \uparrow_1, \perp\} \times Q \times \Sigma \to P(Q \times \{\uparrow, \downarrow_0, \downarrow_1\})$$

We will describe the *run* of $\mathcal{A}$ over some Σ-labeled tree. The automaton starts in the root of the tree assuming the initial state. It then walks around the tree, choosing nondeterministically a direction to move based on the current state, the label in the current vertex and information on what it did in the previous move. This information can assume several values: $\downarrow$ if it entered the current vertex from its parent, $\uparrow_0$ if it entered from a left son, $\uparrow_1$ if it entered from the right son and $\perp$ if there was an error in the previous move.

Based on this information, $\mathcal{A}$ decides whether it wants to move to the ancestor by choosing $\uparrow$; or down into the left or right son by choosing $\downarrow_0$ or $\downarrow_1$ respectively. The previously mentioned error can occur if the automaton tried to move up in the root, or tried to move down in a leaf. In these cases, the error $\perp$ is reported and the automaton does not move. By convention, we assume that in the initial configuration, $\mathcal{A}$ remembers that in the previous move an error occurred. A run is *accepting* if $\mathcal{A}$ enters an accepting state. A tree t is accepted by $\mathcal{A}$ if there is an accepting run over t.

A *deterministic tree walking automaton*, DTWA for short, is defined like a nondeterministic TWA, except that δ returns only one result:

$$\delta : \{\downarrow, \uparrow_0, \uparrow_1, \perp\} \times Q \times \Sigma \to Q \times \{\uparrow, \downarrow_0, \downarrow_1\}$$

We say a TWA (resp. DTWA) is *1-bounded* if it traverses every edge at most once in each direction in every possible run. Let 1-TWA be the class of 1-bounded nondeterministic TWA and let 1-DTWA be the class of 1-bounded DTWA.

Those familiar with TWA will notice that our definition is slightly nonstandard. Normally, a TWA decides what move to make based not on the type $\{\downarrow, \uparrow_0, \uparrow_0, \perp\}$ of the previous move, but on information whether the vertex is: the root, a left son, a right son or a leaf. It is an easy exercise to show that these formalisms are equivalent. However, we choose the history-based approach so that we can talk about runs on a subtree s of a larger tree t without specifying whether s is rooted as a left or right son in t.

2.1 Finite Monoid Equations

Let $\mathrm{Var} = \{x, y, z, \ldots\}$ be an infinite set of *variables*. Given a finite set Σ disjoint with Var, a *Σ-equation* is an expression of the form $w = w'$, where $w, w' \in (\Sigma \cup \mathrm{Var})^*$. A function $\nu : \mathrm{Var} \to \Sigma^*$ is called a *valuation*. Let $\nu^* : (\mathrm{Var} \cup \Sigma)^* \to \Sigma^*$ be the function substituting the $\nu(x)$ for every variable x. Given a homomorphism $h : \Sigma^* \to M$ from the free monoid Σ^* into some finite monoid M, we say ν *satisfies* $(h, w = w')$ if $h(\nu^*(w)) = h(\nu^*(w'))$.

A set E of Σ-equations is *finitely solvable* if for every finite monoid M and every homomorphism $h : \Sigma^* \to M$, there is a valuation ν such that ν satisfies (h, e) for all $e \in E$. A *constrained set of equations* is a set of equations augmented with constraints on the valuation ν. In this paper we consider constraints of the form $\sigma \in \nu(x)$ or $\sigma \notin \nu(x)$ stating what letters $\sigma \in \Sigma$ must or cannot appear in $\nu(x)$.

Example 1. For every alphabet Σ and any satisfiable set of constraints, the equation $x = xx$ is finitely solvable. This follows from the following Lemma:

Lemma 1. *In every finite monoid M there is some $n \in \mathbb{N}$ such that for all elements a of M, $a^n = a^n \cdot a^n$.*

Proof
Since M is finite, $a^j = a^{j+k}$ for some $j, k \leq |M|$. But then $a^m = a^{m+l\cdot k}$ for all $m \geq j, l \geq 0$. In particular $a^{|M|!} = a^{|M|!} \cdot a^{|M|!}$. □

Example 2. The following set of constrained $\{\mathbf{a}, 0, 1\}$-equations is finitely solvable:

$$\begin{aligned} x \cdot 0 \cdot y_0 &= x' \cdot 0 \cdot y_0 \\ x \cdot 1 \cdot y_1 &= x' \cdot 1 \cdot y_1 \\ \mathbf{a} \in \nu(x) &\quad \mathbf{a} \notin \nu(x') \end{aligned}$$

Let $h : \{0, 1, \mathbf{a}\}^* \to M$ be an arbitrary homomorphism and let n be the constant from Lemma 1 appropriate to the monoid M. Let $a' = 1 \cdot 0^n$ and $a = 1 \cdot \mathbf{a} \cdot 0^n$. Given a word σ, let $\sigma^{-1} \cdot b$ be the word b' such that $\sigma \cdot b' = b$.We define the valuation ν as follows:

$$\nu(x) = a' \cdot a^n \qquad \nu(x') = a' \qquad \nu(y_0) = 0^{n-1} \cdot a^n \qquad \nu(y_1) = 1^{-1} \cdot a^n$$

We must show that:

$$\begin{aligned} h(a' \cdot a^n \cdot 0^n \cdot a^n) &= h(a' \cdot 0^n \cdot a^n) \\ h(a' \cdot a^n \cdot a^n) &= h(a' \cdot a^n) \end{aligned}$$

These equations are true because, by Lemma 1, $h(a' \cdot 0^n) = h(a')$, $h(a \cdot 0^n) = h(a)$, and $h(a^n) = h(a^n \cdot a^n)$.

3 Separating 1-DTWA from TWA

In this and the following section we will define two languages L_1 and L_2 we conjecture to separate the DTWA from TWA and TWA from regular languages respectively. The first language, L_1 is the following set of $\{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$-trees:

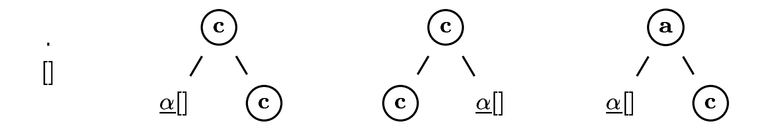
$$L_1 = \{t : \exists x, y \in \mathrm{dom}(t).\ t(x) = \mathbf{a} \wedge t(y) = \mathbf{b} \wedge x < y\}$$

We are only able to prove that L_1 cannot be recognized by any 1-DTWA. The proof is split into two parts. First we show that no deterministic automaton that does a depth-first search of the tree can recognize L_1. Then we generalize this result to arbitrary 1-DTWA.

3.1 A DFS Automaton Cannot Recognize L_1

A DFS automaton is a 1-DTWA that in every run visits v before w iff v is lexicographically before w. Given a DFS automaton $\mathcal{A}$, we will prepare two trees, $t \in L_1$ and $t' \notin L_1$ such that $\mathcal{A}$ cannot distinguish between them. In these trees we distinguish several nodes: $w_1, w_2, w_3 \in \mathrm{dom}(t)$ and $w_1', w_2', w_3' \in \mathrm{dom}(t')$. We then claim that they are corresponding in the sense that they have similar histories – for instance the state assumed by $\mathcal{A}$ the second time it visits w_2 is the same as the state assumed in w_2' in the corresponding visit. Finally, we show that $\mathcal{A}$ cannot distinguish the paths that lead from w_3 to the root in t and from w_3' to the root in t' respectively, thus showing that $\mathcal{A}$ either accepts both trees or rejects both trees.

The trees t and t' will have essentially very little branching – in fact they have only one "real" branching point each, apart from which we use trees obtained via an encoding operation, which given a word $\alpha \in \{0, 1, \mathbf{a}\}^*$, produces a tree with a hole $\underline{\alpha}[]$ (see Fig. 1 for the definition of $\underline{\alpha}[]$).



$$\underline{\epsilon}[] = [] \qquad \underline{0\alpha}[] = (\underline{\alpha}[], \mathbf{c}, \mathbf{c}) \qquad \underline{1\alpha}[] = (\mathbf{c}, \mathbf{c}, \underline{\alpha}[]) \qquad \underline{\mathbf{a}\alpha}[] = (\underline{\alpha}[], \mathbf{a}, \mathbf{c})$$

Fig. 1. The operation assigning $\underline{\alpha}[]$ to α

Let $\alpha, \alpha', \beta, \beta', \gamma_1, \gamma_2 \in \{0, 1, \mathbf{a}\}^*$ be some words which we will specify later by solving certain equations. We only require that α contains the letter $\mathbf{a}$ and α', γ_1 do not. The trees t and t' are defined as follows (see Fig. 2 for a graphic representation):

$$t_1 = \underline{\gamma_1}[(\mathbf{b}, \mathbf{c}, \underline{\gamma_2}[\mathbf{c}])] \qquad t = \underline{\alpha}[(\underline{\beta}[\mathbf{c}], \mathbf{c}, t_1)] \qquad t' = \underline{\alpha'}[(\underline{\beta'}[\mathbf{c}], \mathbf{c}, t_1)]$$

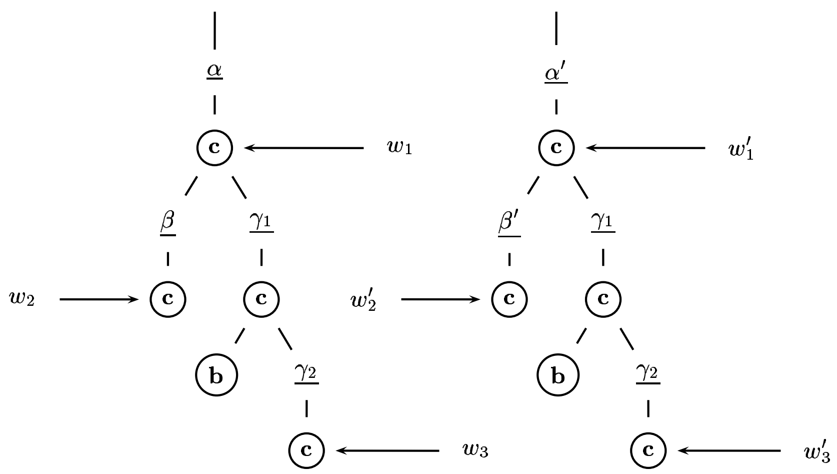


Fig. 2. The trees t and t'

By assumption on α, α' and γ_1, we have $t \in L_1$ and $t' \notin L_1$. We will show that for every DFS automaton $\mathcal{A}$, we can find words $\alpha, \alpha', \beta, \beta', \gamma_1, \gamma_2$ such that $\mathcal{A}$ cannot distinguish between t and t'.

A *partial state transformation* is any partial function from Q to Q. For every word $\alpha \in \{0, 1, \mathbf{a}\}^*$ we will define two partial state transformations $f^{\uparrow}_{\alpha}$ and $f^{\downarrow}_{\alpha}$. The intuition behind this being that $f^{\uparrow}_{\alpha}$ assigns to a state q the state $f^{\uparrow}_{\alpha}$ that $\mathcal{A}$ will end in if it starts in the hole of $\underline{\alpha}[]$ in state q and moves toward the root continuing the depth-first search. If in this run the automaton does not behave like a DFS automaton – it wants to visit something twice, for instance – $f^{\uparrow}_{\alpha}(q)$ is undefined. A similar intuition goes for $f^{\downarrow}_{\alpha}(q)$, except that we start in the root and end up in the hole this time.

We will only define $f^{\uparrow}_{\mathbf{a}}(q)$, leaving the reader to define $f^{\uparrow}_{0}, f^{\uparrow}_{1}, f^{\downarrow}_{\mathbf{a}}, f^{\downarrow}_{0}, f^{\downarrow}_{1}$ analogously. Let q_1, q_2, q_3, q_4 be states satisfying the following equations:

$$(q_1, \downarrow_1) = \delta(\uparrow_0, \mathbf{a}, q) \qquad (q_2, \downarrow_0) = \delta(\downarrow, \mathbf{c}, q_1)$$
$$(q_3, \uparrow) = \delta(\bot, \mathbf{c}, q_2) \qquad (q_4, \uparrow) = \delta(\uparrow_1, \mathbf{a}, q_3)$$

If such states cannot be found, this means that $\mathcal{A}$ does not behave like a DFS automaton and $f^{\uparrow}_{\mathbf{a}}(q)$ is undefined, otherwise $f^{\uparrow}_{\mathbf{a}}(q) = q_4$. We extend $f^{\uparrow}, f^{\downarrow}$ from letters to arbitrary words in $\{0, 1, \mathbf{a}\}^*$ in the obvious fashion:

$$f^{\uparrow}_{\alpha \cdot \beta} = f^{\uparrow}_{\alpha} \circ f^{\uparrow}_{\beta} \qquad f^{\downarrow}_{\alpha \cdot \beta} = f^{\downarrow}_{\beta} \circ f^{\downarrow}_{\alpha}$$

Consider the run of $\mathcal{A}$ on the tree t. Let q_1 be the state of $\mathcal{A}$ assumed when first arriving in the vertex w_2 (see Fig. 2). Let q_2 be the state assumed when coming back from the left son to w_1; q_3 when first arriving at w_3; and, finally, let q_4 be the state upon coming back to the root upon reading the whole tree.

Analogously we define the states q_1', q_2', q_3', q_4', which describe the run on the tree t'. An easy analysis shows that if $\mathcal{A}$ is a DFS automaton then these states satisfy the following equations:

$$\begin{array}{lll}
q_1 = f^{\downarrow}_{\alpha\cdot 0\cdot\beta}(q_I) & q_1' = f^{\downarrow}_{\alpha'\cdot 0\cdot\beta'}(q_I) & (1) \\
q_2 = f^{\uparrow}_{0\cdot\beta}(g_{\mathbf{c}}(q_1)) & q_2' = f^{\uparrow}_{0\cdot\beta'}(g_{\mathbf{c}}(q_1')) & (2) \\
q_3 = g(q_2) & q_3' = g(q_2') & \\
q_4 = f^{\uparrow}_{\alpha\cdot 1\cdot\gamma_1\cdot 1\cdot\gamma_2}(q_3) & q_4' = f^{\uparrow}_{\alpha'\cdot 1\cdot\gamma_1\cdot 1\cdot\gamma_2}(q_3') & (3)
\end{array}$$

where

- q_I is the initial state of $\mathcal{A}$
- g is the state transformation induced by going down the branch $\underline{\gamma_1}$, into the letter **b**, back up one edge, and down the branch $\underline{\gamma_2}$
- $g_{\mathbf{c}}$ is the state transformation induced by entering a **c** leaf and leaving it

Obviously, if we can prove that the corresponding $f^{\uparrow}$, $f^{\downarrow}$ functions in the above equations are equal, then we will have proven that $q_4 = q_4'$ and, consequently, that $\mathcal{A}$ does not distinguish between t and t'. Using similar techniques as in Example 2, we can show that the following set of $\{\mathbf{a}, 0, 1\}$-equations is finitely solvable:

$$\begin{array}{rcll}
x\cdot 0\cdot y & = & x'\cdot 0\cdot y' & (e_1) \\
0\cdot y & = & 0\cdot y' & (e_2) \\
x\cdot 1\cdot z_1\cdot 1\cdot z_2 & = & x'\cdot 1\cdot z_1\cdot 1\cdot z_2 & (e_3) \\
\mathbf{a}\in\nu(x) & & \mathbf{a}\notin\nu(x') & \text{constraints}
\end{array}$$

Let M_Q be the monoid whose elements are partial state transformations and the monoid operation is composition. Given an arbitrary monoid M, let $\tilde{M}$ denote the monoid whose operation $\cdot_{\tilde{M}}$ is defined:

$$a\cdot_{\tilde{M}} b = b\cdot_M a$$

The function h associating with each word α the functions $f^{\uparrow}_{\alpha}, f^{\downarrow}_{\alpha}$ is a homomorphism from $\{\mathbf{a}, 0, 1\}^*$ into the monoid $M_Q \times \tilde{M}_Q$. Let ν be a valuation satisfying (h, e_1), (h, e_2) and (h, e_3), which exists by the finite solvability of the above equations. Then the equations (1), (2) and (3) will be satisfied if we set:

$$\alpha = \nu(x) \qquad \alpha' = \nu(x') \qquad \beta = \nu(y) \qquad \beta' = \nu(y') \qquad \gamma_1 = \nu(z_1) \qquad \gamma_2 = \nu(z_2)$$

3.2 From DFS Automata to Arbitrary 1-DTWA

We fix a 1-DTWA $\mathcal{A}$ for this section. We will show that $\mathcal{A}$ cannot recognize L_1 by reducing the problem to DFS automata. The reduction is accomplished by replacing in t and t' every σ-labeled (for $\sigma \in \{\mathbf{a}, \mathbf{c}\}$) inner node with a tree $s_\sigma[,]$, which has the property that if $\mathcal{A}$ first visits the left son in the root of s_σ, then it will first visit the left son in both of the holes.

Lemma 2. *If $L(\mathcal{A}) = L_1$ then $\mathcal{A}$ must visit every node in a tree outside L_1.*

Proof
Assume $\mathcal{A}$ does not visit a node v in the tree $s \notin L_1$. Let s' be some tree in L_1. Since $s[v := s'] \in L_1$, the desired contradiction comes from the fact that $\mathcal{A}$ accepts s iff $\mathcal{A}$ accepts $s[v := s']$. $\square$

We say q is an *entering* state if in some run there is a vertex v such that when $\mathcal{A}$ first visits v, state q is assumed. A state is *clean entering* if one of the runs in question is over a tree outside L_1. It is $\mathbf{c}$-*clean entering* if additionally all vertices betweend v and the root are labeled by $\mathbf{c}$. Let $\sigma \in \{\mathbf{a}, \mathbf{c}\}$. A tree t is (q, σ)-clean if either:

- q is $\mathbf{c}$-clean entering and $(t, \sigma, t) \notin L_1$;
- q is merely clean entering and $(t, \mathbf{a}, t) \notin L_1$

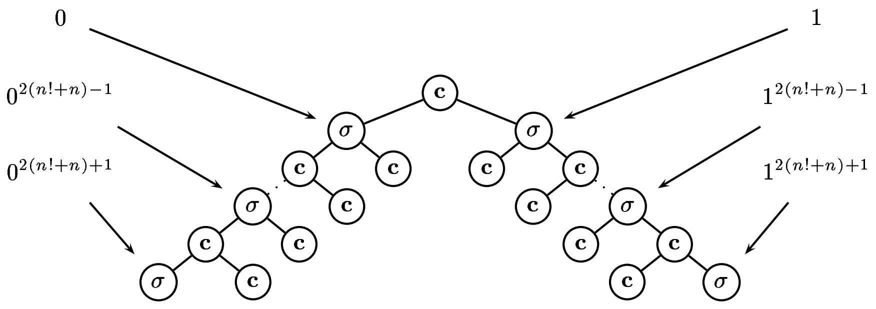


Fig. 3. The tree s_σ

We are now going to define for $\sigma \in \{\mathbf{a}, \mathbf{c}\}$ a tree s_σ that will allow us to simulate DFS automaton behavior in an arbitrary 1-DTWA. Let $n = |Q|$. We define the tree s_σ as follows (see Fig. 3):

$$\begin{aligned} \mathrm{dom}(s_\sigma) = &\quad \{0^i : i \le 2(n! + n)\} \cup \{1^i : i \le 2(n! + n)\} \cup \\ &\cup \{0^i \cdot 1 : i < 2(n! + n)\} \cup \{1^i \cdot 0 : i < 2(n! + n)\} \\ s_\sigma(v) = &\quad \begin{cases} \sigma & \text{if } v = 0^{2i+1} \text{ or } v = 1^{2i+1} \text{ for some } i \\ \mathbf{c} & \text{otherwise} \end{cases} \end{aligned}$$

Let $s_\sigma[,]$ be a tree with two holes which is obtained from s_σ by placing the first hole instead of the vertex $0^{2n!}$ and placing the second hole instead of the vertex $1^{2n!}$.

Lemma 3. *Let q be a clean entering state, $\sigma \in \{\mathbf{a}, \mathbf{c}\}$, and let $t, t' \in trees(\{\mathbf{a}, \mathbf{b}, \mathbf{c}\})$ be such that $t(\varepsilon) = t'(\varepsilon) = \mathbf{c}$ and t is (q, σ)-clean. If $\mathcal{A}$ first visits the left son in the root of s_σ then in the run on $s_\sigma[t, t']$, $\mathcal{A}$ first visits the left son in the vertices $0^{2n!}$ and $1^{2n!}$.*

Proof
Let q be a clean entering state and assume that $\delta(\downarrow, \mathbf{c}, q) = (q', \downarrow_0)$, i. e. $\mathcal{A}$ first goes into the left subtree. Consider now the run ρ on s_σ that starts in this state q. First note that, in accordance with Lemma 2, this run must visit every vertex of the tree.

For $i \in \{0, \ldots, n! + n\}$, let q^i be the state assumed in 0^{2i} after going up from 0^{2i+1}. By a counting argument, for some $j \leq n$, $q^{n!+j} = q^{n!}$. But then $q^{n!} = q^{n!-j} = \cdots = q^0$. Since $\mathcal{A}$ must visit all the tree and the left subtree of the root ε was visited first, $\delta(\uparrow_0, q^{n!}, \mathbf{c}) = \delta(\uparrow_0, q^0, \mathbf{c}) = (q', \downarrow_1)$ for some $q' \in Q$. But this means that the left son of $0^{2n!}$ was visited before the right son. By a similar reasoning, we show that the left son of $1^{2n!}$ was visited before the right son.

Finally, to finish the proof of the Lemma, we notice that the decision to first visit the left subtree in $0^{2n!}$ is made *before* visiting the actual subtree. This means that the result stays the same no matter what tree we substitute under the vertex $0^{2n!}$, as long as this tree has $\mathbf{c}$ in the root. A similar argument can be made for $1^{2n!}$, however this time we must use the assumption that the tree substituted under $0^{2n!}$ was (q, σ)-clean, since otherwise we could not have assumed in the proof that $\mathcal{A}$ needs to visit the whole tree. $\square$

Lemma 4. *$\mathcal{A}$ does not recognize L_1.*

Proof

Without loss of generality, we assume that in the root and initial state q_I, $\mathcal{A}$ first visits the left subtree. For every tree $s \in trees\{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$ that has inner nodes labeled only by $\mathbf{a}, \mathbf{c}$, we define a tree $\tilde{s} \in \text{trees}(\{\mathbf{a}, \mathbf{b}, \mathbf{c}\})$ by induction. If $s = (s_0, \sigma, s_1)$, then $\tilde{s} = s_\sigma[\tilde{s}_0, \tilde{s}_1]$; otherwise $\text{dom}(s) = \{\varepsilon\}$ and then $\tilde{s} = s$. Given $v \in \text{dom}(s)$, let $\tilde{v} \in \text{dom}(\tilde{s})$ be the vertex whose subtree is $\widetilde{s|_v}$. We will prove that $\mathcal{A}$ does not distinguish between $\tilde{t}$ and $\tilde{t'}$ – where the trees t and t' are the ones discussed in the previous section.

For $s \in \{t, t'\}$, we call a vertex v *s-main*, if $v = \tilde{w}$ for some $w \in \text{dom}(s)$. Since $\tilde{t'} \notin L_1$, the assumptions of Lemma 3 apply and $\mathcal{A}$ behaves like a DFS automaton in its entire run over $\tilde{t'}$, i. e. first visits the left son before the right son in every t'-main vertex. By a similar reasoning, in the run on $\tilde{t}$, $\mathcal{A}$ behaves like a DFS automaton before entering the right son of $\tilde{w_1}$. By the results on DFS automata, the states assumed by $\mathcal{A}$ before entering the right son of $\tilde{w_1}$ and before entering the right son of $\tilde{w'_1}$ are the same. Since the right subtrees of $\tilde{w_1}$ and $\tilde{w'_1}$ are the same, the vertices $\tilde{w_3}$ and $\tilde{w'_3}$ are reached in the same state.

In order not to violate the 1-bounded condition, $\mathcal{A}$ must now go back to the root without visiting the left son of any main vertex. Note that here we use the fact that *no* run of a 1-bounded TWA can traverse an edge twice in the same direction. Thus, by the results on DFS automata, $\mathcal{A}$ does not distinguish between $\tilde{t}$ and $\tilde{t'}$. $\square$

Theorem 1
No deterministic 1-TWA recognizes L_1, but there is a nondeterministic 1-TWA recognizing L_1.

Proof
This follows immediately from Lemma 4 and the fact L_1 can be easily recognized by a 1-TWA that guesses some position labeled by **a** and then some deeper position labeled by **b**. □

In fact, we conjecture:

Conjecture 1. No deterministic TWA recognizes L_1. Consequently, deterministic TWA recognize fewer languages than their nondeterministic counterparts.

4 Separating 1-TWA from REG

Consider the regular language $L_2 \subseteq \text{trees}(\{\mathbf{a}, \mathbf{b}, \mathbf{c}\})$ consisting of trees where there exist two incomparable occurrences of **b** below some occurrence of **a**.

Conjecture 2. No TWA recognizes L_2

From a logical point of view, this language is rather simple, as testified by:

– In XPath, L_2 is the set of trees satisfying the query //**a**[//[*[1]//**b**][*[2]//**b**]];
– An obvious TWA that uses conjunction can recognize L_2;
– In CTL over ordered trees, L_2 is the set of trees satisfying (0 and 1 stand for child number tests) $\mathrm{EF}[\mathbf{a} \wedge \mathrm{EX}(0 \wedge \mathrm{EF}\mathbf{b}) \wedge \mathrm{EX}(1 \wedge \mathrm{EF}\mathbf{b})]$

Consequently, proving Conjecture 2 would show that TWA do not subsume the above formalisms. Unfortunately, we can only prove that no 1-TWA can recognize L_2:

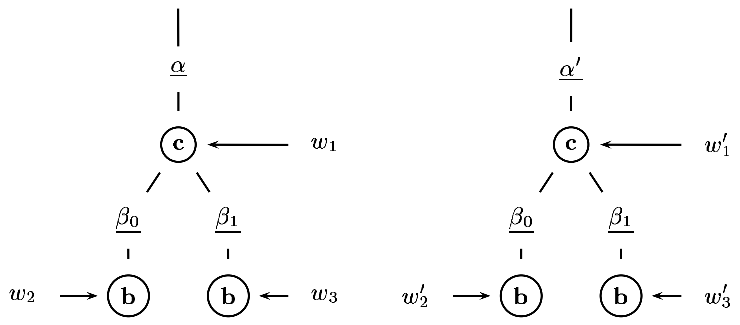


Fig. 4. The trees t and t'

Lemma 5. *No 1-TWA recognizes L_2.*

Proof (sketch)
Let $\mathcal{A}$ be some 1-TWA. We take some words $\alpha, \alpha', \beta_0, \beta_1 \in \{\mathbf{a}, 0, 1\}^*$ such that $\mathbf{a} \in \alpha$, $\mathbf{a} \notin \alpha'$ and define the trees t, t' as in Fig. 4. By assumption on α and α', we have $t \in L_2$ and $t \notin L_2$. We will prove that – given a suitable choice of $\alpha, \alpha', \beta_0, \beta_1$ – if $\mathcal{A}$ accepts t then $\mathcal{A}$ accepts t'.

Since we are now dealing with nondeterministic automata, the state transformations will be replaced by arbitrary relations, which associate with a state all the possible states reachable from it via a particular tree. Let N_Q be the monoid whose elements are binary relations over Q and whose operation is the standard composition of relations:

$$(q, q') \in R \circ S \Leftrightarrow \exists q''.(q, q'') \in R \wedge (q'', q') \in S$$

With each word $\gamma \in \{\mathbf{a}, 0, 1\}^*$ we associate two relations $R_\gamma^\uparrow$ and $R_\gamma^\downarrow$. Intuitively, $(q, q') \in R_\gamma^\uparrow$ if there is a run that starts in q in the hole of $\underline{\gamma}[]$ and ends up in q' in the root and a similar definition holds for $R_\gamma^\downarrow$. Since the function associating these relations with a word is a homomorphism into the monoid $N_Q \times \tilde{N_Q}$, we can finish the proof by solving the following constrained equations:

$$\begin{aligned} x \cdot 0 \cdot y_0 &= x' \cdot 0 \cdot y_0 \\ x \cdot 1 \cdot y_1 &= x' \cdot 1 \cdot y_1 \\ \mathbf{a} \in \nu(x) &\quad \mathbf{a} \notin \nu(x') \end{aligned}$$

These are, of course, the equations from Example 2, and we know that they are finitely solvable. This means that there are words $\alpha, \alpha', \beta, \beta'$ such that:

$$\begin{array}{llll} \text{(A)} & R^\uparrow_{\alpha\cdot 0\cdot \beta_0} = R^\uparrow_{\alpha'\cdot 0\cdot \beta_0} & R^\downarrow_{\alpha\cdot 0\cdot \beta_0} = R^\downarrow_{\alpha'\cdot 0\cdot \beta_0} & \text{(B)} \\ \text{(C)} & R^\uparrow_{\alpha\cdot 1\cdot \beta_1} = R^\uparrow_{\alpha'\cdot 1\cdot \beta_1} & R^\downarrow_{\alpha\cdot 1\cdot \beta_1} = R^\downarrow_{\alpha'\cdot 1\cdot \beta_1} & \text{(D)} \end{array}$$

We use these words in the trees t and t'. Consider an accepting run of $\mathcal{A}$ on the tree t. Obviously, it must visit both w_2 and w_3. Assume, without loss of generality, that w_2 is reached before w_3, in state q. Since in this run the right son of w_1 was not visited, there is also a run over the tree $\underline{\alpha \cdot 0 \cdot \beta_1}[\mathbf{b}]$ that assumes state q in the $\mathbf{b}$ node. By equation (B), there is a run over the tree $\underline{\alpha' \cdot 0 \cdot \beta_1}[\mathbf{b}]$ that also assumes state q in the $\mathbf{b}$ node.

Consider first the case when, in this second run, the right son of the vertex corresponding to w_1' is visited. However, in the run over t, starting in state q and vertex w_2, the automaton goes up and then into the right son of w_1. This first case would then give us a run over $\underline{\alpha' \cdot 0 \cdot \beta_1}[\mathbf{b}]$ that violates the 1-bounded condition. Here we use the fact that *no* run of a 1-bounded TWA can traverse an edge twice in the same direction. Thus the second case must hold, i. e., in the run over $\underline{\alpha' \cdot 0 \cdot \beta_1}[\mathbf{b}]$ the right son of the vertex corresponding to w_1' is *not* visited. But this shows that $\mathcal{A}$ can enter w_2' in state q.

In a similar manner we can copy the rest of the accepting run onto the tree t', which shows that $\mathcal{A}$ cannot accept L_2. □

5 Concluding Remarks and Further Work

In this paper we have proved that 1-DTWA recognize fewer languages than 1-TWA, which in turn do not recognize all the regular languages. The result about 1-DTWA is new and may be a first step toward proving the long standing conjecture that DTWA do not recognize all languages recognized by TWA. The second result is an improvement on a language of Neven and Schwentick, which also separates 1-TWA from the regular languages.

The proofs of both our results require solving certain kinds of monoid equations. Apart from giving answers to the conjectures 1 and 2, a good topic for further work is the question:

Is the finite solvability of (constrained) monoid equations decidable?

Apart from an independent interest, an answer to this question might be relevant to further work on the expressive power of TWA. It seems plausible that an attempt to continue the line of attack presented here might require solving numerous and more complicated equations. An automatic procedure might then come in handy; moreover techniques used in such a procedure might shed some insight on the TWA.

Acknowledgments: I would like to take this opportunity to thank my supervisor Igor Walukiewicz for his help in my research. Without his suggestions and our countless discussions, I could not have written this paper.

References

1. A. V. Aho and J. D. Ullman. Translations on a context-free grammar. *Information and Control*, 19:439–475, 1971.
2. A. Brügemann-Klein and D. Wood. Caterpillars. a context specification technique. *Markup Languages*, 2(1):81–106, 2000.
3. E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Logics of Programs: Workshop*, volume 131 of *LNCS*, pages 52–71, 1981.
4. J. Engelfriet and H. J. Hoogeboom. Tree-walking pebble automata. In G. Paum J. Karhumaki, H. Maurer and G. Rozenberg, editors, *Jewels are forever, contributions to Theoretical Computer Science in honor of Arto Salomaa*, pages 72–83. Springer-Verlag, 1999.
5. J. Engelfriet, H. J. Hoogeboom, and J.-P. van Best. Trips on trees. *Acta Cybernetica*, 14:51–64, 1999.
6. T. Kamimura and G. Slutzki. Parallel two-way automata on directed ordered acyclic graphs. *Information and Control*, 49(1):10–51, 1981.
7. T. Milo, D. Suciu, and V. Vianu. Type-checking for XML transformers. In *Proceedgins of the Nineteenth ACM Symposium on Principles of Database Systems*, pages 11–22, 2000.
8. F. Neven. Automata, logic, and XML. In Julian C. Bradfield, editor, *Computer Science Logic, 16th International Workshop, 11th Annual Conference of the EACSL*, volume 2471 of *LNCS*, pages 2–26, 2002.
9. F. Neven and T. Schwentick. On the power of tree-walking automata. In *Automata, Languages and Programming, 27th International Colloquium, ICALP 2000*, volume 1853 of *LNCS*, 2000.

Appendix

Lemma 6. *The following set of* $\{\mathbf{a}, 0, 1\}$*-equations is finitely solvable:*

$$\begin{gathered} x \cdot 0 \cdot y = x' \cdot 0 \cdot y' \\ 0 \cdot y = 0 \cdot y' \\ x \cdot 1 \cdot z_1 \cdot 1 \cdot z_2 = x' \cdot 1 \cdot z_1 \cdot 1 \cdot z_2 \\ \mathbf{a} \in \nu(x) \qquad \mathbf{a} \notin \nu(x') \end{gathered}$$

Proof
Let $h : \{0, 1, \mathbf{a}\}^* \to M$ be an arbitrary homomorphism and let n be the constant from Lemma 1 appropriate to the monoid M. Let:

$$a = 1^n \qquad b = (0b)^n \qquad c = (b^n \mathbf{a} b^n)^n$$

We will show that the following valuation ν is a solution:

$$\begin{aligned} \nu(x) &= b \cdot c & \nu(x') &= b \\ \nu(y) &= 0^{-1} \cdot c & \nu(y') &= 0^{-1} \cdot c \cdot c \\ \nu(z_1) &= 1^{-1} \cdot a & \nu(z_2) &= 1^{-1} \cdot a \cdot c \end{aligned}$$

We must show that:

$$\begin{aligned} h(b \cdot c \cdot c) &= h(b \cdot c \cdot c) \\ h(c) &= h(c \cdot c) \\ h(b \cdot c \cdot a \cdot a \cdot c) &= h(b \cdot a \cdot a \cdot c) \end{aligned}$$

A simple verification employing Lemma 1 shows that these equations are true.
□