Tree-Walking Automata Do Not Recognize All Regular Languages

Mikolaj Bojanczyk [*]
Warsaw University and LIAFA, Paris 7
bojan@mimuw.edu.pl

Thomas Colcombet [†]
Warsaw University and IRISA-CNRS
colcombe@irisa.fr

ABSTRACT

Tree-walking automata are a natural sequential model for recognizing tree languages. Every tree language recognized by a tree-walking automaton is regular. In this paper, we present a tree language which is regular but not recognized by any (nondeterministic) tree-walking automaton. This settles a conjecture of Engelfriet, Hoogeboom and Van Best. Moreover, the separating tree language is definable already in first-order logic over a signature containing the left-son, right-son and ancestor relations.

Categories and Subject Descriptors

F.4.3 [**Formal Languages**]: Classes defined by grammars or automata; F.1.1 [**Models of Computation**]: Automata

General Terms

Theory

1. INTRODUCTION

A tree-walking automaton is a natural type of finite automaton working over trees. At every moment of its run, a tree-walking automaton is in a single node of the tree and in one of a finite number of states. It walks around the tree, choosing a neighboring node based on the current state, the label in the current node, and whether this node is a left son, a right son, a leaf, or the root. The tree is accepted if one of the accepting states is reached. Even though tree-walking automata were introduced in the early seventies by Aho and Ullman [1], not much is known about this model.

This situation is different from the "usual" tree automata – branching tree automata – which are a 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 this class enjoys many nice properties of the class of regular word languages.

It is not difficult to prove that every language recognized by a tree-walking automaton is regular. However, until recently most fundamental questions pertaining to tree-walking automata remained unanswered:

1. Is every regular language recognized by a tree-walking automaton?
2. Can tree-walking automata be determinized?
3. Are tree-walking automata closed under complementation?

There has been much related research, which can be roughly grouped in two categories: nondefinability results for weakened models of tree-walking automata [6, 7, 2] and definability results for strengthened models of tree-walking automata [5, 4]. In [3] it was shown that the answer to question 2 is negative. In this paper we show that the answer to question 1 is also negative: we present a regular language that is not recognized by any tree-walking automaton. The techniques used here extend the ones of [3].

2. BASIC DEFINITIONS

The trees in this paper are finite, binary trees labeled by a given finite alphabet Σ. A Σ-*tree* t is a mapping from $N_t \subseteq \{0,1\}^*$ to Σ, where N_t is a finite, non-empty, prefix-closed set such that for any $v \in N_t$, $v0 \in N_t$ iff $v1 \in N_t$. Elements of N_t are called *nodes* of the tree. A set of trees over a given alphabet is called a *tree language*.

With every tree language L we can associate the standard Myhill-Nerode congruence $\simeq_L$, which identifies two trees if they cannot be distinguished by any context. More precisely, $s \simeq_L s'$ holds if for every tree t and every node v of t, either both or none of the trees $t[v := s]$, $t[v := s']$ belong to L. Here $t[v := s]$ is the usual operation of substituting a tree for a node. A tree language L is *regular* if the relation $\simeq_L$ is of finite index. We denote by REG the class of regular tree languages.

We now proceed to define tree-walking automata. Every node v in a tree t has a type. The possible values are Types $= \{r, 0, 1\} \times \{l, i\}$, where r stands for the root, 0 for a left son, 1 for a right son, l for a leaf and i for an internal node (not a leaf). A *direction* is an element of $\{\uparrow, \varepsilon, 0, 1\}$, where informally $\uparrow$ stands for 'parent', ε stands for 'stay', 0 for 'left son' and 1 for 'right son'.

[*]Supported by EC project GAMES and Polish KBN grant No. 4 T11C 042 25.
[†]Partially supported by EC project GAMES.

DEFINITION 1. *A* 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 respectively the sets of* initial *and* accepting *states, and* δ *is the* transition relation *of the form*

$$\delta \subseteq Q \times \text{Types} \times \Sigma \times Q \times \{\uparrow, \varepsilon, 0, 1\}.$$

A *configuration* is a pair of a node and a state. A *run* is a sequence of configurations, where every two consecutive configurations are consistent with the transition relation. A run is *accepting* if it starts and ends in the root of the tree, the first state is in I and the last state is 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. We use TWA to denote the class of tree languages recognized by some tree-walking automaton.

We would like to point out here that reading the type of a node is an essential feature of a tree-walking automaton. Indeed, Kamimura and Slutzki show in [6] that tree-walking automata which do not have access to this information cannot recognize all regular languages, being incapable of even searching a tree in a systematic manner.

One can easily verify that every language recognized by a tree-walking automaton is regular, i.e. TWA $\subseteq$ REG. It has been long open whether this inclusion is strict. Engelfriet conjectured that this is indeed the case [4]. A proof of this conjecture is the subject of the present paper.

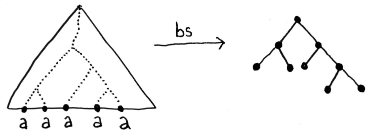
3. THE SEPARATING LANGUAGE

In this section we present a regular language L not accepted by any tree-walking automaton. This language witnesses the strictness of the inequality

$$\text{TWA} \subsetneq \text{REG}.$$

We restrict ourselves to $\{\mathbf{a}, \mathbf{b}\}$-trees. Moreover we require that only leaves can be labeled by $\mathbf{a}$. We sometimes refer to the symbol $\mathbf{b}$ as the *blank symbol.* Trees containing only the blank symbol are called *blank trees.* In a blank tree, only the structure is important.

For a non-blank tree t with $\mathbf{a}$ occurring only in the leaves, we define its "branching structure". Intuitively this is a blank tree whose leaves correspond to $\mathbf{a}$-leaves in the tree t and whose structure mirrors the one in t. Each inner node in the branching structure can be identified with a greatest common ancestor of two $\mathbf{a}$-leaves in the tree t. Formally, a blank tree s is the *branching structure* $bs(t)$ of a tree t if there is an injective mapping h assigning nodes of s to nodes of t such that the lexicographic and prefix orders are preserved and the image under h of the leaves of s is the set of $\mathbf{a}$-labeled leaves in t. The branching structure is uniquely defined. The following drawing illustrates this definition on an example.



Let K be the set of blank trees where all branches are of even length. The language L mentioned at the beginning of this section is $bs^{-1}(K)$, i.e. the set of trees whose branching structure belongs to K. We now state the main result of this paper:

THEOREM 2. *The language* L *is regular but is not recognized by any tree-walking automaton.*

The easy part is showing that L is regular. The Myhill-Nerode congruence for this language has four classes: trees that cannot be a subtree of a tree in L, trees whose branching structure has only branches of even length and trees whose branching structure has only branches of odd length.

One can in fact show a stronger result:

FACT 1. *The language* L *is definable in FO, i.e. first-order logic with the ancestor relation and the left and right successor relations.*

Proof It is easy to show that the function bs can be implemented by an FO-interpretation. Since FO-interpretation preserves FO-definability by inverse image, it remains to show that the language K is FO-definable.

The main idea is that using FO we can check the parity of the depth of a leaf in $(01)^*(\varepsilon + 0)$. We will refer to such a leaf as the *middle leaf* of the tree, and to the corresponding branch as the *middle branch.* An FO-formula can detect the middle leaf by checking that each of its ancestors is either the right son of a left son, the left son of a right son, the left son of the root, or the root itself. The *middle parity* of a tree is defined to be the parity of the depth of the middle leaf; it is FO-definable since the middle node is at even depth if and only if it is a left son. The *middle parity* of a node is defined to be the middle parity of the subtree rooted at this node.

Let M be the set of trees whose middle parity is even, and where the two sons of any internal node have the same middle parity. We claim that $K = M$. According to the previous remarks, this implies that K is FO-definable.

The inclusion $K \subseteq M$ is obvious. For the other direction, let t be a tree outside K. If all leaves in t have the same depth parity, then the middle node is a right son and $t \notin M$. Otherwise, let v be a node in t of maximal depth whose subtree has leaves of both even and odd depth. But then the middle parities of v's sons must be different and $t \notin M$. $\square$

The hard part in the proof of Theorem 2 remains: we need to prove that the language L is not recognized by any tree-walking automaton. The rest of this paper is devoted to proving this result.

3.1 Overview of the proof

The proof is divided into three parts.

In the first part (Section 4), we define patterns. A pattern is a particular type of tree with distinguished nodes, called ports. Patterns are constructed in such a way that the automaton gets lost when traveling from one port to another. For every possible branching structure t, we construct out of patterns a tree $\Delta_t^{\mathbf{a}}$ whose branching structure is t. We then show that in such a tree, a tree-walking automaton is basically limited to doing variations on a depth-first search.

In the second part (Section 5) we reduce the problem to the acceptance of K by a simpler kind of automata, called frontier automata. A frontier automaton can be seen as a version of the tree-walking automaton that works directly on the branching structure. This reduction is based on inspecting the way a tree-walking automaton can behave over a tree built out of patterns.

Finally, in the third part (Sections 6 and 7) we show that frontier automata cannot recognize the language K. The proof principle is to take a big tree in K and perform a transformation on it — a 'rotation' — which cannot be detected by the frontier automata, but yields a tree outside K. This concludes the proof of Theorem 2.

4. PATTERNS

In this section we define patterns, develop a pumping argument for them, and then study its consequences for the automaton.

Patterns are fragments of trees with some holes (called ports) in them. Patterns can be assembled by gluing their ports together. Any automaton naturally induces an equivalence relation on such objects: two patterns are equivalent if in any context, the automaton cannot detect the difference when one pattern is replaced by another. This equivalence relation is the key notion in the study of patterns.

In Section 4.1 we define patterns. We then state Lemma 4, where we introduce three basic patterns, which satisfy useful equivalences. These will be used as building blocks in subsequent constructions.

In Section 4.2, we combine the basic patterns into pattern expansions, i.e. bigger patterns that are locally confusing for the automaton.

In Section 4.3 we study possible behaviors of the automaton over the basic patterns (Proposition 1 and Lemma 10). This study shows what are the possible runs that the automaton can perform inside a pattern expansion.

4.1 Patterns and pattern equivalence

We fix for this section a tree-walking automaton

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

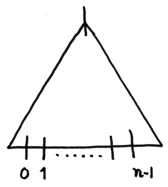


Figure 1: A pattern of arity n

A *pattern* Δ is a $\{\mathbf{b}, *\}$-tree where the symbol $*$ labels only leaves which are left sons (the left son assumption is for technical reasons). See Fig. 1 for an illustration. The i-th $*$-labeled leaf (numbered from left to right, starting from 0) is called the i-th-*port*. *Port* ε stands for the root. The number of $*$ labels is called the *arity* of the pattern. 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 pattern Δ_i for the i-th port. Given a set P of patterns, we denote by $\mathcal{C}(P)$ the least set of patterns which 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 (p, i, q, j) if it is possible for $\mathcal{A}$ to go from state p in port i to state q in port j in Δ. Ports are treated as non-leaf left sons. In particular the port ε is not seen as the root and leaf ports are not seen as leaves by the automaton.

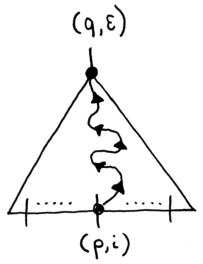


Figure 2: A pattern Δ with (p, i, q, ε) in δ_Δ

From the point of view of the automaton, the relation δ_Δ sums up all important properties of a pattern and we consider two patterns *equivalent* if they induce the same relation. 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. To simplify the definition, we only consider contexts where the root of the pattern corresponds to a left son, and the nodes plugged into the leaf ports are not leaves.

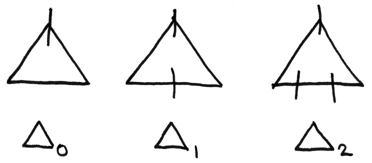


Figure 3: The patterns Δ_0, Δ_1 and Δ_2

The following lemma was shown in [3]:

Lemma 4. *There exist patterns $\Delta_0, \Delta_1, \Delta_2$ – of arities* 0, 1 *and* 2 *respectively – such that any pattern in $\mathcal{C}(\Delta_0, \Delta_1, \Delta_2)$ of arity $i = 0, 1, 2$ is equivalent to Δ_i.*

The patterns Δ_0, Δ_1 and Δ_2 are the key to our proof. In a sense, their construction encapsulates all of the pumping arguments that we will do with respect to the automaton $\mathcal{A}$. For instance, the pattern Δ_1 is equivalent to a composition of any number of copies of Δ_1 patterns. In particular, if the automaton can go from the leaf port of Δ_1 to the root port, then there must be a state that is used twice along the way.

The automaton may do some redundant moves, such as going one step down, and then one step up, without any apparent purpose. It will be convenient to eliminate this obfuscating phenomenon. For this we introduce the inner loop relation:

Definition 5. *The* inner loop $\to_\varepsilon$ *relation over states is the least transitive and reflexive relation such that $p \to_\varepsilon q$ holds whenever $(p, \varepsilon, q, \varepsilon)$ or $(p, 0, q, 0)$ belongs to δ_{Δ_1}. For a pattern Δ, the relation γ_Δ is defined to be the set of tuples (p, i, q, j) such that $p \to_\varepsilon p'$ and $q' \to_\varepsilon q$ for some p', q' satisfying $(p', i, q', j) \in \delta_\Delta$.*

The following lemma shows that we can treat the δ and γ relations interchangeably:

Lemma 6. *Two patterns $\Delta, \Delta' \in \mathcal{C}(\{\Delta_0, \Delta_1, \Delta_2\})$ are equivalent if and only if $\gamma_\Delta = \gamma_{\Delta'}$.*

Proof [Idea] A consequence of Lemma 4 is that all patterns in $\mathcal{C}(\{\Delta_0, \Delta_1, \Delta_2\})$ are equivalent to ones where Δ_1 has been plugged in all the ports. This implies that the γ_Δ relation is obtained in a uniform way from the δ_Δ relation. $\Box$

4.2 Pattern expansions

The *pattern preexpansion* of blank tree t is the pattern obtained by replacing every inner node of t with the pattern Δ_2 and replacing every leaf node with a port $*$. The pattern preexpansion has as many leaf ports as t has leaves. The *pattern expansion* Δ_t of t is obtained plugging a Δ_1 pattern into every port (leaf and root) of the pattern preexpansion (see Fig. 4). With every node v of t we associate a node Δ_v in the pattern Δ_t, this node does not depend on t. A *special node* in a pattern expansion is any node of the form Δ_v.

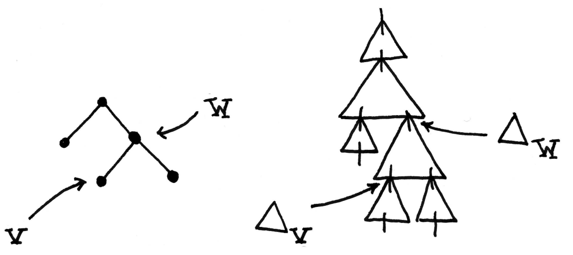


Figure 4: A pattern expansion

Given a blank tree t, the tree Δ_t^{a} is obtained by plugging an a-labeled node into each port of Δ_t. One can easily verify that the branching structure of Δ_t^{a} is t. If the tree walking automaton were to accept the language L, it would have to accept every tree Δ_t^{a} for $t \in K$ and reject every tree Δ_t^{a} for $t \notin K$. We will show later that this is impossible, due to the way tree-walking automata get lost in pattern expansions.

The following lemma shows that the γ_{Δ_2} relation describes the way our fixed tree walking automaton can move across pattern expansions:

LEMMA 7. *Let t be a blank tree along with two nodes $v \cdot a, v \cdot b$, with $v \in \{0,1\}^*$ and $a \neq b \in \{\varepsilon, 0, 1\}$. The following are equivalent for any two states p and q:*

- *The automaton can go in Δ_t from state p in the node $\Delta_{v \cdot a}$ to state q in the node $\Delta_{v \cdot b}$ without visiting any ports or other special nodes, and not visiting $\Delta_{v \cdot b}$ before $\Delta_{v \cdot a}$;*
- *(p, a, q, b) belongs to γ_{Δ_2}.*

LEMMA 8. *Let v be a node in a blank tree t. If the automaton can loop in Δ_t from state p in node Δ_v to state q in node Δ_v without visiting any ports, then $p \to_\varepsilon q$ holds.*

The above two lemmas show that runs of the automaton between special nodes in pattern expansions can be assumed to have a very particular form. Take for instance a blank tree t and two nodes $v < w$. If there is a run that goes from Δ_v to Δ_w then, by Lemmas 7 and 8, there is a run that does this by doing a series of steps of the form $(p, \varepsilon, q, 0), (p, \varepsilon, q, 1) \in \gamma_{\Delta_2}$. A similar characterization holds when v and w are incomparable: the automaton first goes directly from Δ_v in the up direction, then does one of the steps $(p, 0, q, 1), (p, 1, q, 0) \in \gamma_{\Delta_2}$ and then goes directly down to Δ_w. This type of reasoning will be used in the reduction of Theorem 2 to a study of frontier automata.

4.3 A characterization of moves over Δ_1

In this section, we analyze the relations γ_{Δ_0}, γ_{Δ_1} and γ_{Δ_2}. We present a classification of the possible ways the automaton can go in Δ_1 from the leaf port to the root port. This classification will be used to formulate two key properties of frontier automata: Provisos 1 and 2.

From now, instead of the γ_{Δ_0}, γ_{Δ_1} and γ_{Δ_2} relations, we will be using the more graphical notation depicted in Fig. 5.

$$\begin{array}{lll lll}
 & & & p \nwarrow q & \text{if} & (p, 1, q, \varepsilon) \in \gamma_{\Delta_2} \\
p \circlearrowleft q & \text{if} & (p, 0, q, 0) \in \gamma_{\Delta_0} & p \nearrow q & \text{if} & (p, 0, q, \varepsilon) \in \gamma_{\Delta_2} \\
 & & & p \searrow q & \text{if} & (p, \varepsilon, q, 1) \in \gamma_{\Delta_2} \\
p \uparrow q & \text{if} & (p, 0, q, \varepsilon) \in \gamma_{\Delta_1} & p \swarrow q & \text{if} & (p, \varepsilon, q, 0) \in \gamma_{\Delta_2} \\
p \downarrow q & \text{if} & (p, \varepsilon, q, 0) \in \gamma_{\Delta_1} & p \curvearrowleft q & \text{if} & (p, 1, q, 0) \in \gamma_{\Delta_2} \\
 & & & p \curvearrowright q & \text{if} & (p, 0, q, 1) \in \gamma_{\Delta_2}
\end{array}$$

$$\begin{array}{ll}
p \Lsh q & \text{if } p \nwarrow q \text{ and not } p \nearrow q \\
p \Rsh q & \text{if } p \nearrow q \text{ and not } p \nwarrow q \\
p \Rdsh q & \text{if } p \swarrow q \text{ and not } p \searrow q \\
p \Ldsh q & \text{if } p \searrow q \text{ and not } p \swarrow q
\end{array}$$

Figure 5: Graphical notation for $\gamma_{\Delta_0}, \gamma_{\Delta_1}, \gamma_{\Delta_2}$

The following proposition is key to our understanding of the way tree-walking automata move across expansions:

PROPOSITION 1. *If $p \uparrow q$ holds then either:*

1. *For some state r, $p \uparrow r \nearrow r \nwarrow r \uparrow q$ holds; or*
2. *For some state r, $p \Lsh r \Lsh r \Lsh q$ holds; or*
3. *For some state r, $p \Rsh r \Rsh r \Rsh q$ holds; or*
4. *For some states r_1, r_2, r, one of the below holds:*
   (a) *$p \uparrow r_1 \Lsh r_1 \uparrow q$ and $p \uparrow r_2 \Rsh r_2 \uparrow q$; or*
   (b) *$p \uparrow r_1 \Lsh r_1 \nearrow r_2 \Lsh r_2 \uparrow q$; or*
   (c) *$p \uparrow r_1 \Rsh r_1 \nwarrow r_2 \Rsh r_2 \uparrow q$.*

We do not go into the proof of this proposition, which is long and complicated. A symmetric proposition holds for $\downarrow$.

The point of characterizing $\uparrow$ and $\downarrow$ is that these are the most basic types of move the automaton can make in a pattern expansion. Indeed, by Lemma 7, in order to move from one special node to another, the automaton needs to traverse the Δ_2 pattern. Since the pattern Δ_2 can be seen as having Δ_1 plugged in each of its ports, each such traversal must employ one of the moves $\uparrow$ or $\downarrow$. But then we can use Proposition 1 in order to uncover other possible moves of the automaton.

Proposition 1 becomes really useful when used in conjunction with Lemma 10, which relates it with depth-first searches. We now proceed to define the concept of a depth-first search (DFS) over patterns.

DEFINITION 9. *A pair* $(q,\bar{q})$ *is a* left-to-right DFS *if*

$$q \swarrow q\ , \qquad q \circlearrowleft \bar{q}\ , \qquad \bar{q} \nwarrow \bar{q}\ , \quad \textit{and} \quad \bar{q} \curvearrowright q\ .$$

A pair of states $(q,\bar{q})$ *is a* right-to-left DFS *if*

$$q \searrow q\ , \qquad q \circlearrowleft \bar{q}\ , \qquad \bar{q} \nearrow \bar{q}\ , \quad \textit{and} \quad \bar{q} \curvearrowleft q\ .$$

Assume for instance that $(q,\bar{q})$ is a left-to-right depth-first search. In this case, the automaton can go in any expansion Δ_t from state $\bar{q}$ in a node Δ_v to state q in any node Δ_w, as long as w is lexicographically after v and there is no port of Δ_t lexicographically between the nodes.

LEMMA 10. *If* $\bar{q} \Rsh \bar{q}$ *holds, then* $(q,\bar{q})$ *is a left-to-right DFS for some state* q. *If* $\bar{q} \Lsh \bar{q}$ *holds, then* $(q,\bar{q})$ *is a right-to-left DFS for some state* q.

A symmetric lemma holds for $↳$ and $↲$, except that the roles of the states q and $\bar{q}$ are reversed. When put together, Proposition 1 and Lemma 10 give us some idea of how a tree-walking automaton can move upwards within a pattern expansion: it may either get completely lost (by allowing a move from any node to any of its ancestors, case 1 in Proposition 1), allow a depth-first search in some fixed direction and nothing else (cases 2 and 3), or, finally, do some depth-first searches coupled with moves in opposing directions (case 4).

5. FRONTIER AUTOMATA

In this section we introduce a new kind of automaton, called a frontier automaton. We then state Proposition 2, which reduces Theorem 2 to proving that the language K (the set of blank trees with all leaves at even depth) is not accepted by any positive boolean combination of frontier automata.

Informally speaking, a frontier automaton is a tree-tree walking automaton that jumps from one leaf to another: sequences of steps of the automaton which do not encounter the root or an $\mathbf{a}$-labeled leaf are now considered as "atomic". We now proceed with a formal definition of a frontier automaton.

A *step* is an element of $\{\varepsilon,0,1\} \times \{\varepsilon,0,1\}$. Given a step (a,b) and nodes v,w, we write $v \rightarrow_{(a,b)} w$ if $v = ua$ and $w = ub$ for some u. For instance we have $010 \rightarrow_{(0,1)} 011$.

A *relative path* (or simply a path) is a sequence of steps. A path $\pi = s_0 \ldots s_n$ goes from a node v to a node w, written $v \rightarrow_\pi w$, if for some (actually unique) sequence of nodes $v = v_0 \ldots v_n = w$, $v_i \rightarrow_{s_i} v_{i+1}$ holds for all $i < n$.

For $k \in \mathbb{N} \cup \{\infty\}$, a *k-mix* of a in b is defined to be the set of words over $\{a,b\}$ that have at most k a's. A *mix* of $\{a,b\}$ is a k-mix of a in b or a k-mix of b in a, for some k. A *move* is a finite union sets of the form $\{\varepsilon\}$, $U(0,1)D$ or $U(1,0)D$, where U and D are some mixes of $\{(0,\varepsilon),(1,\varepsilon)\}$ and $\{(\varepsilon,0),(\varepsilon,1)\}$ respectively. A move of the form $U(0,1)D$ is called a *right move*, a move of the form $U(1,0)D$ is called a *left move*. Given a move M and two nodes v,w, we write vMw if there is a path π in M such that $v \rightarrow_\pi w$ holds.

We assume that all moves satisfy a certain *additional property* that will be defined in Provisos 1 and 2 in Section 7.

DEFINITION 11. *A* frontier automaton *is a tuple* $\mathcal{A} = (Q, q_I, I, q_F, F, \delta)$, *where* Q *is a finite set of states,* $q_I, q_F \in Q$ *are called respectively the* initial *and* final *states,* I, F *are mixes over* $\{0,1\}$ *called the respectively the* initial *and* final *positions and the* transition function δ *assigns to each pair of states a move.*

A frontier automaton runs over leafs in a blank tree. A *configuration* of the automaton is a pair (p,u) where p is a state of the automaton and u is a leaf of the tree. We write $(p,u) \rightarrow^{\mathcal{A}} (q,v)$ if the automaton can go from configuration (p,u) to configuration (q,v) using one move, i.e if uMv holds for $M = \delta(p,q)$. A *run* in a tree is a sequence of configurations where the automaton can go from every configuration to the next one in one move. We write $(p,u) \Rightarrow_t^{\mathcal{A}} (q,v)$ if a run exists that begins in (p,u) and ends in (q,v). When the automaton $\mathcal{A}$ is clear from the context, we skip the superscript $\mathcal{A}$ from the notation $\rightarrow^{\mathcal{A}}$ and $\Rightarrow_t^{\mathcal{A}}$. A tree is *accepted* by the automaton if there is a run that starts in a leaf belonging to I with state q_I and ends in some leaf belonging to F with state q_F.

The following proposition reduces the membership of L in TWA to the question whether frontier automata can recognize the language K. It will allow us to work directly on K. In particular, all trees considered afterward will be blank trees.

PROPOSITION 2. *If a tree-walking automaton recognizes* L, *then* K *is a positive boolean combination of languages recognized by frontier automata.*

The proof of this statement follows by inspecting runs of a tree-walking automaton over pattern expansions. We omit here the details and only give a sketch of the construction.

As noted before, a tree-walking automaton that recognizes L must be able separate the sets

$$\{\Delta_t^{\mathbf{a}} : t \in K\} \qquad \text{and} \qquad \{\Delta_t^{\mathbf{a}} : t \notin K\}\ .$$

However, over trees of the form $\Delta_t^{\mathbf{a}}$, a tree-walking automaton moves in a special way, which is dependent on the relations γ_{Δ_0}, γ_{Δ_1} and γ_{Δ_2}. Recall from Section 4.2 that when going from one special node to another (incomparable) one in a pattern expansion, a tree-walking automaton may be assumed to use a run of the form: first go up to the greatest common ancestor, then do a $(1,0)$ or $(0,1)$ move in Δ_2, and then descend into the destination node. This accounts for the fact that moves in a frontier automaton are of the form $U(0,1)D$ or $U(1,0)D$. The fact that U and D are mixes, and the additional Provisos 1 and 2 are inferred from the characterizations of $\uparrow$ and $\downarrow$ presented in Section 4.3. The positive boolean combination is obtained by decomposing the run of the tree-walking automaton into subruns that do not visit the root.

6. THE ROTATION

By Proposition 2, in order to show Theorem 2, it is enough to show that we can trick any positive boolean combination of frontier automata. This is done as follows. We start with a balanced binary blank tree T of large even depth. Clearly T belongs to K, therefore it is accepted by frontier automata $\mathcal{A}_1,\ldots,\mathcal{A}_n$ that make the positive boolean combination in question true. We then find a node u_0 in T and perform a rotation at that node. Rotation is the operation depicted in Fig. 6; it moves the subtrees rooted in $u_0 00, u_0 01$ and $u_0 1$ to the new positions $u_0 0, u_0 10$ and $u_0 11$. One can easily see that the resulting tree T' is not in K. We will, however, show that all the frontier automata $\mathcal{A}_1,\ldots,\mathcal{A}_n$ must also accept T':

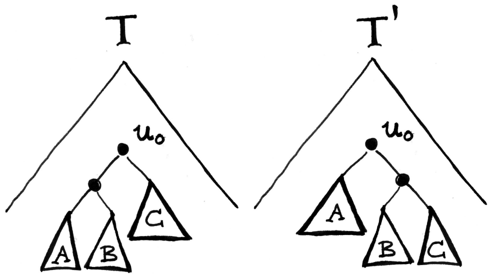


Figure 6: Rotating at node u_0

Proposition 3. *The tree T' is accepted by all the automata $\mathcal{A}_1, \ldots, \mathcal{A}_n$.*

This shows that the boolean combination in question could not have recognized K, thereby proving Theorem 2. In the rest of this section we describe how to properly choose the node u_0. Then, in Section 7, we prove Proposition 3.

We consider two blank trees s, t equivalent if we can replace s by t in any context and none of the frontier automata $\mathcal{A}_1, \ldots, \mathcal{A}_n$ can tell the difference (in terms of accepting). A tree t is *fractal* if it contains a subtree equivalent to itself. All complete binary trees of large enough depth are fractal.

Given a frontier automaton, we say that one state is *reachable* from another if they can be connected by a sequence of nonempty moves. A *strongly connected component* (simply a *component* from now) is a maximal set of pairwise reachable states. With each of the frontier automata $\mathcal{A}_i$ that accept T we associate an accepting run ω_i. We say a run ω_i *changes components* below a node w if it contains two successive configurations (u,p), (v,q) such that p and q are in different components and w is an ancestor of either u or v.

We require the node u_0 to satisfy the following constraints:

1. For $|u| \leq 2$, the subtree of T rooted in u_0u is fractal;
2. The node u_0 is below the node 01010101;
3. The runs ω_i do not change components below u_0;
4. The first and last leaves visited by each run ω_i are not below u_0.

Since the number of times a run can change components is bounded by the number of these components in the automaton, one can find a node satisfying the above properties, provided that the tree T is sufficiently big.

We say that a component Γ of an automaton $\mathcal{A}$ cannot *detect the rotation* if for every two leaves v, w in T – or, equivalently, in T' – not below the node u_0, the following holds for all states p, q in the component Γ:

$$(p, v) \Rightarrow_T^{\mathcal{A}} (q, w) \qquad \text{implies} \qquad (p, v) \Rightarrow_{T'}^{\mathcal{A}} (q, w) \ .$$

Since none of the runs ω_i change components below the node u_0, in order to prove Proposition 3 it is enough to show that no component of the automata $\mathcal{A}_1, \ldots, \mathcal{A}_n$ can detect the rotation. The rest of this paper is devoted to showing this.

$$\begin{aligned}
\mathit{Stay} &= \varepsilon \\
\_\sqcap_{\to} &= (1,\varepsilon)^*(0,1)(\varepsilon,0)^* \\
{}_{\leftarrow}\sqcap\_ &= (0,\varepsilon)^*(1,0)(\varepsilon,1)^* \\
\sqcap_{\to} &= ((0,\varepsilon)+(1,\varepsilon))^*(0,1)(\varepsilon,0)^* \\
{}_{\leftarrow}\sqcap &= ((0,\varepsilon)+(1,\varepsilon))^*(1,0)(\varepsilon,1)^* \\
\_\sqcap\!\downarrow &= (1,\varepsilon)^*(0,1)((\varepsilon,0)+(\varepsilon,1))^* \\
\downarrow\!\sqcap\_ &= (0,\varepsilon)^*(1,0)((\varepsilon,0)+(\varepsilon,1))^* \\
\sqcap_{\leftarrow} &= ((0,\varepsilon)+(1,\varepsilon))^*(0,1)(\varepsilon,1)^* \\
{}_{\to}\sqcap &= ((0,\varepsilon)+(1,\varepsilon))^*(1,0)(\varepsilon,0)^* \\
\underline{\sqcap}\!\downarrow &= (0,\varepsilon)^*(0,1)((\varepsilon,0)+(\varepsilon,1))^* \\
\downarrow\!\underline{\sqcap} &= (1,\varepsilon)^*(0,1)((\varepsilon,0)+(\varepsilon,1))^*
\end{aligned}$$

Figure 7: Elementary moves

7. FRONTIER AUTOMATA CANNOT DETECT THE ROTATION

Before we proceed to show that frontier automata cannot detect the rotation, we need to define the additional properties (Provisos 1 and 2) of frontier automata that were announced in Section 5. The first proviso will allow us to perform a case analysis in the proof of Proposition 3. Generally speaking, it says that each move can either be decomposed as a union of the moves in Figure 7, or it contains what we call 'a shift'. The second, more technical, proviso will be used to solve the components with a shift.

The basic moves of the decomposition are listed in Figure 7. The move *Stay* corresponds to the automaton staying in the same leaf. The moves $\_\sqcap_{\to}$ and ${}_{\leftarrow}\sqcap\_$ correspond to jumping respectively to the next and the previous leaf. The other moves are slightly more complex.

We denote by $\#_t(v)$ the number of the leaves in the tree t that are lexicographically before v. We denote by $\#_t(u,v)$ the offset from u to v within t, i.e. $\#_t(v) - \#_t(u)$.

Definition 12. *A* move offset *of a move M is an integer i such that uMv holds for any two leaves u and v that satisfy $\#_t(u,v) = i$ in some tree t. We write* $\mathrm{moff}(p,q)$ *for the set of move offsets of $\delta(p,q)$. We say that a move from p to q contains a* shift *if* $\mathrm{moff}(p,q)$ *contains two successive integers.*

For instance, any move containing $\_\sqcap_{\to}$ has 1 in its move offset. We now state the properties satisfied by all moves in frontier automata that were announced in Section 5 but not defined or used up till now:

Proviso 1. *For any two states p and q the set* $\mathrm{moff}(p,q)$ *contains one of* $\{-2,-1,0,1,2\}$ *and*

- *either is a union of some of the moves Stay, $\_\sqcap_{\to}$, ${}_{\leftarrow}\sqcap\_$, $\sqcap_{\leftarrow}$, $\sqcap_{\to}$, ${}_{\to}\sqcap$, $\_\sqcap\!\downarrow$, ${}_{\leftarrow}\sqcap$, $\underline{\sqcap}\!\downarrow$, $\downarrow\!\sqcap\_$, $\downarrow\!\underline{\sqcap}$ (see Fig. 7);*
- *or contains a shift.*

Proviso 2. *Let (p,u), (q,v) be two configurations in a tree t such that $(p,u) \to (q,v)$. If $\#_t(u,v) > \max(\mathrm{moff}(p,q))$ then $\delta(p,q)$ contains a right move not contained in $\_\sqcap_{\to}$. Symmetrically, if $\#_t(u,v) < \min(\mathrm{moff}(p,q))$ then $\delta(p,q)$ contains a left move not contained in ${}_{\leftarrow}\sqcap\_$.*

Using the first proviso, we divide all components into two categories: components with a shift, i.e. those where $\text{moff}(p,q)$ contains a shift for some states p, q; and components without a shift. Proposition 3 is then proved in the two following sections for each of the two categories. The second proviso is used in the section on components with a shift.

7.1 Components with a shift

In this section we fix a component Γ with a shift and prove that it cannot detect the rotation. In order to do this, we extend the definition of move offsets to *run offsets*, where more than one move can be used. A run offset between state p and state q is defined as a move offset (Definition 12), except that: 1) uMv is replaced by $(p,u) \Rightarrow_t (q,v)$; and 2) the leaves u and v are required to have at least n leaves both to their left and right (n being the number of states in the component). The set of run offsets between states p and q is denoted $\text{roff}(p,q)$. Note that all leaves in T that are below 01 (in particular below u_0) satisfy assumption 2).

A pair of states (p,q) is a *right-teleport* if $\text{roff}(p,q)$ contains all but a finite number of positive integers. The pair (p,q) is a *left-teleport* if $\text{roff}(p,q)$ contains all but a finite number of negative integers.

LEMMA 13. *If a component Γ contains a shift, either all pairs of states from Γ are right-teleports, or all are left-teleports.*

Proof We have the following facts for any states p, q, r:

$$\text{moff}(p,q) \subseteq \text{roff}(p,q) \quad \text{and} \quad \text{roff}(p,q) + \text{roff}(q,r) \subseteq \text{roff}(p,r) .$$

Using this, connectedness of the component Γ, and Proviso 1, we obtain that $\text{roff}(p,q)$ is nonempty for any $p,q \in \Gamma$. We deduce from this that for any state p in Γ, the set $\text{roff}(p,p)$ is closed under addition and contains two consecutive values. It follows by some simple arithmetic that (p,p) is a teleport. This extends to any pair of states in Γ. $\square$

Let us assume without loss of generality that all pairs of states in Γ are left-teleports, i.e. all sets $\text{moff}(p,q)$ contain almost all negative integers. Let $d \in \mathbb{N}$ be the greatest number such that $-d$ does not belong to some set $\text{moff}(p,q)$ for $p,q \in \Gamma$. By inspecting the proof of Lemma 13, one can see that d is quadratic in the size of $|\Gamma|$. The number d has the property that whenever $v, w \geq 01$ are two leaves of T or T' such that $\#_T(w,v) < -d$, the automaton can go from (p,w) to (q,v), regardless of the states $p,q \in \Gamma$.

We now proceed to show that the component Γ cannot detect the rotation, i.e. that the implication

$$(p,v) \Rightarrow_T^A (q,w) \qquad \text{implies} \qquad (p,v) \Rightarrow_{T'}^A (q,w)$$

holds for any two nodes v, w not below u_0, and any two states p, q of the component Γ.

The difficult case is when v is one one side of the subtree of u_0 and w is on the other side. If v is to the right and w to the left of u_0, then we are done, since the number of leaves in T' separating v and w is at least d. The difficult case is when v is on the left side of the subtree of u_0 and w is on the right side (see Fig. 8), and moreover no pair of states from Γ is a right teleport. In this case a special trick is needed that uses Proviso 2 and the assumption on u_0 being below 01010101.

By assumption we have a run going from (p,v) to (q,w) in T. Since w is sufficiently far to the right of v, this run

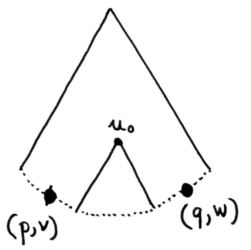


Figure 8: The run from (p,v) to (q,w)

must use a step of the form

$$(r,u) \to (r',u') \qquad \text{with} \qquad \#_t(u,u') > \max(\text{moff}(u,u')) ,$$

because otherwise there would be a state $s \in \Gamma$ with $\text{moff}(s,s)$ containing a positive integer, contradicting the fact that Γ contains no right teleport. By Proviso 2 we infer that $\delta(r,r')$ contains a right move M not contained in $\underrightarrow{\nabla}$.

Our objective is to use this move M in order to "jump" above u_0. For this, we use the following lemma.

LEMMA 14. *There exist in T two leaves u and u' below 01 such that uMu' and, moreover, for any leaf v below u_0,*

$$\#_T(u,v) < -d \qquad \textit{and} \qquad \#_T(v,u') < -d .$$

Proof Let $U(0,1)D$ be one of the components of the move M, with U, D being mixes of $\{(0,\varepsilon),(1,\varepsilon)\}$ and $\{(\varepsilon,0),(\varepsilon,1)\}$ respectively. By the definition of mixes, a right move not contained in $\underrightarrow{\nabla}$ must contain one of the following languages:

$$(i,\varepsilon)^*(1-i,\varepsilon)(0,1)(\varepsilon,j)^* \qquad \text{for } i,j = 0,1$$
$$(i,\varepsilon)^*(0,1)(\varepsilon,j)(\varepsilon,1-j)^* \qquad \text{for } i,j = 0,1$$
$$(0,\varepsilon)^*(0,1)(\varepsilon,0)^*, \;\; (0,\varepsilon)^*(0,1)(\varepsilon,1)^*, \text{ or } (1,\varepsilon)^*(0,1)(\varepsilon,1)^*.$$

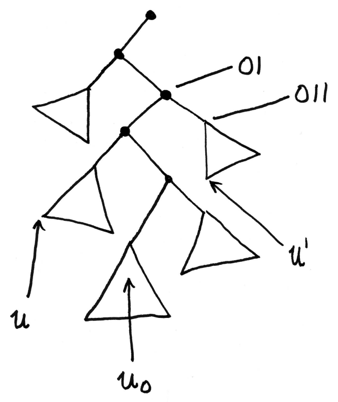


Figure 9: The move from u to u'

The lemma is proved by case analysis as to which one of the above is contained in M. We only do the case of the set $(0,\varepsilon)^*(0,1)(\varepsilon,0)^*$, the other ones are similar. Let u be the leftmost node below 01 and let u' be the leftmost node below 011 (see Fig. 9). Clearly both nodes are below 01. One can easily verify that uMu' holds. Finally, let v be any node below u_0, in particular below 01010101. Since all nodes of the subtree 0100 – and there are more than d of them – are between u and v, we obtain $\#_T(u,v) < -d$. Similarly, all nodes of the subtree 01011 are between v and u', therefore we also have $\#_T(v,u') < -d$. $\square$

We are now ready to show that the component Γ cannot detect the rotation. If the run corresponding to $(p,v) \Rightarrow_T^A (q,w)$ never visits a leaf below u_0, then we can use the same run on T' and we are done. Otherwise, we use left teleports to construct a new run in T' that goes from (p,v) to (q,w). In order to do so we use the following fact:

FACT 2. *There are configurations (p',v') and (q',w') below u_0 such that $(p,v) \Rightarrow_{T'} (p',v')$ and $(q',w') \Rightarrow_{T'} (q,w)$.*

Proof Let M be a move that goes in T from a node v_1 not below u_0 to a node v_2 below u_0. The part of M that goes down in a tree is constructed using mixes of the set $\{(\varepsilon,0),(\varepsilon,1)\}$. By the structure of mixes, one can show that M can also go in T' from v_1 to v', where v' is either the leftmost or the rightmost node below u_0. Using this property, we obtain (p',v') from the statement of the lemma by looking at the first configuration in the run $(p,v) \Rightarrow_T (q,w)$ that corresponds to a leaf below u_0.

A symmetric argument is used for (q',w'), this time using the last configuration in the run that corresponds to a leaf below u_0. $\square$

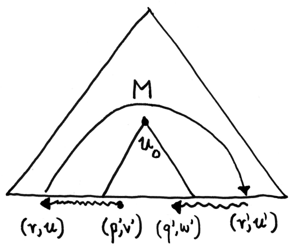


Figure 10: The run in $(p',v') \Rightarrow_{T'} (q',w')$

The run in T' is constructed as follows. First we go from (p,v) to (p',v') (using Fact 2). Then we go to the configuration (r,u) from Lemma 14; this can be done by a left teleport, since u is at least d leaves to the left of v'. From (r,u), we use the move M to go to the configuration (r',u') from Lemma 14. Then we use the left teleport to go to (q',w'), from where we may safely go to (q,w) by Fact 2. See Fig. 10 for an illustration of this run. We have therefore constructed a run in T' that goes from (p,v) to (q,w), thus proving that the component Γ cannot detect the rotation.

7.2 Components without a shift

In this section we consider a component without shifts. According to Proviso 1 the only nonempty moves in the component are: *Stay*, $\sqcup_{\to}$, ${}_{\leftarrow}\sqcup$, $\sqcap_{\leftarrow}$, $\sqcap_{\to}$, $\sqcap_{\downarrow}$, $\sqcup_{\downarrow}$, ${}_{\leftarrow}\sqcap$, ${}_{\to}\sqcap$, ${}_{\downarrow}\sqcup$ and ${}_{\downarrow}\sqcap$. Among these moves, some – called "adjacency moves" – have an important property which prevents them from detecting the rotation. We now proceed to define adjacency moves and then to show how they can be used to simulate other moves.

Let v,w be a pair of nodes such that w is to the right of v. These nodes can be uniquely decomposed as

$$v = u \cdot 0 \cdot 1^i \cdot \bar{v} \qquad w = u \cdot 1 \cdot 0^j \cdot \bar{w}$$

with u being the longest common prefix of v and w and i,j being maximal. We say two pairs of nodes (v,w) and (v',w') are *right adjacency similar* if their corresponding decompositions satisfy $\bar{v} = \bar{v'}$ and $\bar{w} = \bar{w'}$. Two pairs are *adjacency similar* if either (w,v) and (w',v') or (v,w) and (v',w') are right adjacency similar.

DEFINITION 15. *An* adjacency move *is a move M such that for every two adjacency similar pairs of leaves (v,w) and (v',w') we have vMw iff $v'Mw'$.*

LEMMA 16. *Stay*, $\sqcup_{\to}$, ${}_{\leftarrow}\sqcup$, $\sqcap_{\to}$, $\sqcup_{\downarrow}$, ${}_{\leftarrow}\sqcap$, *and* ${}_{\downarrow}\sqcup$ *are adjacency moves.*

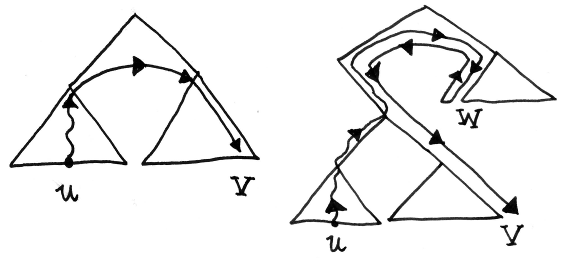


Figure 11: Removing non-adjacency moves

We will now eliminate the moves $\sqcap_{\leftarrow}$, $\sqcap_{\downarrow}$, ${}_{\to}\sqcap$ and ${}_{\downarrow}\sqcap$, which are not adjacency moves. This is done by simulating them with a sequence of adjacency moves. The following lemma treats the case of $\sqcap_{\leftarrow}$ (see Fig. 11 for an illustration of the proof; the other cases are similar):

LEMMA 17. *For a tree t and u,v leaves below 0, $u \sqcap_{\leftarrow} v$ iff there exists another leaf w such that $u \sqcap_{\to} w$ and $w \sqcup_{\to} v$.*

Using this lemma (and the analogous results for $\sqcap_{\downarrow}$, ${}_{\to}\sqcap$ and ${}_{\downarrow}\sqcap$) we can assume without loss of generality that the components without shifts only contain adjacency moves. The following lemma thus shows that a component without shifts cannot detect the rotation, thereby finishing the proof of Theorem 2.

LEMMA 18. *Let p,q be two states of a component containing only adjacency moves. For any nodes u and v not below u_0, if $(p,v) \Rightarrow_T (q,w)$ then $(p,v) \Rightarrow_{T'} (q,w)$.*

Proof Since T and T' are equal over nodes not below u_0, it is enough to establish the lemma for runs where all positions but the initial and final one are below u_0. In other words, the first move of the run is used to enter the subtree rooted in u_0, the last move is used to exit it, and in between all moves are below u_0.

Let f be the mapping that assigns to a leaf in T a leaf in T' with the same number of leaves to the left (i.e. where $\#_T(v) = \#_{T'}(f(v))$). This mapping is a bijection. Let V_1, V_2 and V_3 be the sets of leaves of T respectively below $u_0 00$, $u_0 01$ and $u_0 1$. Let W_1, W_2, W_3 be the sets of leaves of T' respectively below $u_0 0, u_0 10$ and $u_0 11$ (see Fig. 12 for an illustration). One can easily check that $f(V_i) = W_i$ for $i = 1,2,3$. We say two leaves $v \in V_i$ and $w \in V_j$ are *neighbors* if $|i-j| \le 1$. If v,w are neighbors, then the pairs (v,w) and $(f(v),f(w))$ are adjacency similar. In particular, whenever the automaton can go from v to w in one step, then it can

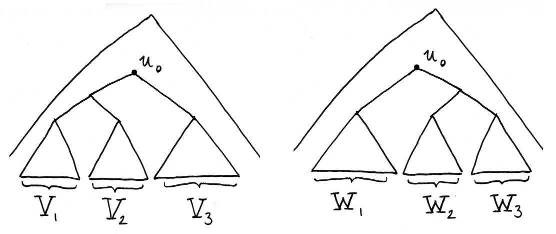


Figure 12: The trees T and T'

do this also from $f(v)$ to $f(w)$. Therefore, if a run only does moves between neighbor nodes then it can be mapped using f into a valid run in the tree T'.

We will transform the run from (p, v) to (q, w) into one that also goes from (p, v) to (q, w), but where all moves are done between neighbor leaves (we call this property (*)). According to the previous remark, this is sufficient to conclude the proof of the lemma.

We will do a case analysis regarding the way the automata entered and exited the subtree rooted in u_0. According to the definitions of the possible adjacency moves, there are only three ways of entering the subtree rooted in u_0: by going to the leftmost leaf below u_0 (using one of _⊓↳ or ⊓↳), to the rightmost one (using one of ↲⊓_ or ↲⊓) or anywhere (using one of _⊓↓ or ↓⊓_). Similarly, there are three ways to exit from this subtree. All this results in nine possibilities. We will treat here only two cases, the others are similar.

- Let us consider first the case where the automaton enters in the leftmost node v of V_1 and leaves by the same node. We will show that the whole run could happen in V_1.

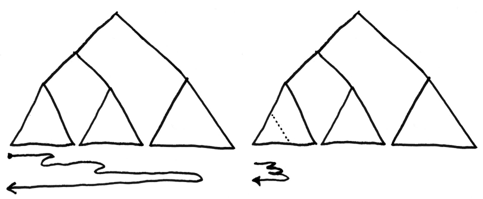
  

  Figure 13: Moving the run to V_1

  Since the subtree below u_0 was fractal, it contains a proper subtree equivalent to itself. Since all subtrees of T are complete binary trees, we may well assume that there is a node u on the leftmost branch below u_0, such that the subtree rooted in u_0u is equivalent to the subtree in u_0. By equivalence, the run that went from the leftmost node below u_0 back to this leftmost node can be assumed to visit only nodes below u_0u (see Fig. 13). But such a run satisfies the property (*).

  All other cases can be solved similarly, except for two: when the automaton enters in the leftmost leaf below u_0 and leaves in the rightmost one, and when the automaton enters in the rightmost leaf below u_0 and leaves in the leftmost one. The first of these is treated in the next item, the second is symmetric.

- Consider a run that begins in the leftmost node of V_1 and ends in the rightmost node of V_3. We are going to construct a similar run satisfying (*). In order to do this, we will use the following extra property: once a position in V_3 is encountered, no position in V_1 is visited anymore. This property is shown in Lemma 19.

  If the run already has property (*), then the problem is over. Otherwise there is some moment in the run where two consecutive configurations are not neighboring. Since after visiting V_3 we never come back to V_1, this means that the first configuration is in V_1 and the second in V_3. In particular, all of the rest of the run satisfies (*). We decompose the run as $\alpha\beta$ with $\alpha = (p_0, v_0) \dots (p_k, v_k)$ and $\beta = (q_0, w_0) \dots (q_n, w_n)$, where the subruns α and β have property (*), $v_k \in V_1$ and $w_0 \in V_3$. The only way to go from a position in V_1 to a position in V_3 is by using the move ⊓↳. This means that w_0 is the leftmost leaf in V_3. However, if we want to use the move ⊓↳ from (p_k, v_k) and satisfy the property (*), the only place we can go to is the leftmost leaf of V_2.

  In order to complete the proof, we will construct a new run γ that satisfies property (*) and goes from state q_0 in the leftmost leaf of V_2 to state p_k in some leaf w' of V_2. The sequence $\alpha\gamma\beta$ is then a valid run satisfying (*), since the move ⊓↳ does not care about the position of the leaf w' within V_2.

  Informally speaking, the run γ uses the run β and fractality to go to some configuration (q_n, w''), with w'' being a node in V_2 with sufficiently many leaves of V_2 to the left and right. Then, using the fact that Γ is a component, we can use $\mathrm{moff}(q_n, p_k)$ to go from (q_n, w'') to (p_k, w'). We omit the details of this construction.

□

LEMMA 19. *Any run in T that begins and ends in V_3 can be modified into one that does not visit V_1.*

Proof Let A be the subtree of T rooted in u_00 (equivalently in u_01). Since the subtree rooted in u_0 was large enough, the tree A is fractal, i.e. has a proper subtree B that is equivalent to A. Since A is a complete binary tree, we may well assume that this subtree is rooted in a node u of A on the rightmost branch. Let S be the tree obtained from T by substituting A for the node u_0u. This tree is equivalent to T (see Fig. 14). Let W be the leaves of S below the node u_0u. Let g be the unique bijection

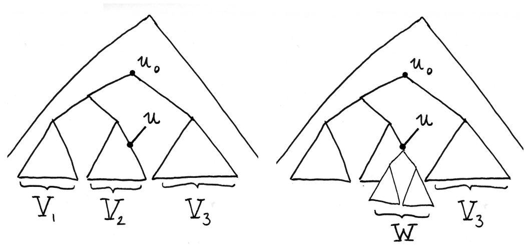


Figure 14: The trees T and S

$$g : V_1 \cup V_2 \cup V_3 \to W \cup V_3$$

that preserves the left-to-right ordering of leaves. One can verify that for any pair of nodes from the domain of g, the image of the pair is adjacency similar to it. Since the component of the automaton in question only has adjacency moves, this means that if the automaton can go from v to w in T, then it can also go from $g(v)$ to $g(w)$ in S. In particular, if there is a run ρ from V_3 back to V_3 in T, then there is a run ρ' with the same starting and ending point in S that only visits $W \cup V_3$. However, since the leaves W in S correspond to A, which is equivalent to the subtree B of u_0u in T, we can replace the W part of ρ' with one over B, thereby obtaining a run over T that does not visit V_1. $\square$

Acknowledgments

We would like to thank the anonymous referee who carefully checked and commented the complete proof of the result.

8. REFERENCES

[1] A. V. Aho and J. D. Ullman. Translations on a Context-Free Grammar. *Information and Control*, 19, pp. 439–475 (1971).

[2] M. Bojańczyk. 1-Bounded TWA Cannot Be Determinized. *Foundations of Software Technology and Theoretical Computer Science*, LNCS 2914, pp. 62–73 (2003)

[3] M. Bojańczyk and T. Colcombet. Tree-Walking Automata Cannot Be Determinized. *International Colloquium on Automata, Languages and Programming*, LNCS 3142, pp. 246–256 (2004)

[4] J. Engelfriet and H. Hoogeboom and J. Van Best. Trips on Trees. *Acta Cybernetica*, 14:1, pp. 51–64 (1999)

[5] J. Engelfriet and H. J. Hoogeboom. Tree-Walking Pebble Automata. *Jewels Are Forever, Contributions to Theoretical Computer Science in Honor of Arto Salomaa*, Springer-Verlag, pp. 72–83 (1999)

[6] , T. Kamimura and G. Slutzki. Parallel Two-Way Automata on Directed Ordered Acyclic Graphs. *Information and Control*, 49:1, pp. 10–51 (1981)

[7] F. Neven and T. Schwentick. On the Power of Tree-Walking Automata. *International Colloquium on Automata, Languages and Programming*, LNCS 1853 (2000)