

Nondeterministic tree-walking automata are not closed under complementation

Olga Martynova* Alexander Okhotin†

September 14, 2025

Abstract

It is proved that the family of tree languages recognized by nondeterministic tree-walking automata is not closed under complementation, solving a problem raised by Bojańczyk and Colcombet (“Tree-walking automata do not recognize all regular languages”, *SIAM J. Comp.* 38 (2008) 658–701). In addition, it is shown that non-deterministic tree-walking automata are stronger than unambiguous tree-walking automata.

1 Introduction

Tree-walking automata, first studied by Aho and Ullman [1], are among the fundamental models in automata theory. A tree-walking automaton walks over a labelled input tree of bounded degree, following the edges; at each moment, an automaton is at some node and is in one of finitely many states, and it uses its transition function to decide which edge to follow and which state to enter. The main questions about the expressive power of tree-walking automata were open for several decades, until the breakthrough results of Bojańczyk and Colcombet [4, 5], who proved that deterministic tree-walking automata (DTWA) are weaker than nondeterministic tree-walking automata (NTWA), which are in turn weaker than bottom-up tree automata.

In these papers, Bojańczyk and Colcombet presented a list of three main problems on tree-walking automata: two of them are the problems they solved, and the third problem is whether the class of tree languages recognized by NTWA is closed under complementation. This paper gives a negative solution to this problem, presenting a tree language recognized by an NTWA, such that its complement cannot be recognized by any NTWA.

Research on tree-walking automata and related models has been conducted in several directions. In particular, tree-walking automata with pebbles were introduced by Engelfriet and Hoogeboom [8], who proved them to be at most as powerful as bottom-up tree automata. Later, Bojańczyk et al. [6] proved strict hierarchies in the number of pebbles for both deterministic and nondeterministic pebble tree-walking automata. They also

*Department of Mathematics and Computer Science, St. Petersburg State University, 7/9 Universitetskaya nab., Saint Petersburg 199034, Russia. E-mail: olga22mart@gmail.com

†Department of Mathematics and Computer Science, St. Petersburg State University, 7/9 Universitetskaya nab., Saint Petersburg 199034, Russia. E-mail: alexander.okhotin@spbu.ru

proved that no number of pebbles can help deterministic automata to simulate NTWA without pebbles. Logical characterizations of pebble tree-walking automata were given by Engelfriet and Hoogeboom [8, 9] and by Neven and Schwentick [15].

For deterministic tree-walking automata, every automaton can be transformed to one that halts on every tree: this was first done by Muscholl et al. [14] using Sipser’s [17] method of traversing the tree of all computations ending in the accepting configuration. This result also implies the closure of DTWA under complementation. Later, Kunc and Okhotin [12] presented a generalized construction applicable to graph-walking automata and producing reversible automata, and also reduced the number of states in the resulting automata from quadratic to linear in the size of the given deterministic automaton.

Much is known about the complexity of decision problems for tree-walking automata. The emptiness and the inclusion problems for both DTWA and NTWA are EXPTIME-complete, see Bojańczyk [2], in contrast to the P-complete emptiness problem for the more powerful bottom-up tree automata, see Veanes [18]. Samuelides and Segoufin [16] determined the complexity of the emptiness and the inclusion problems for k -pebble tree-walking automata: they are k -EXPTIME-complete. Among the recent results, Bojańczyk [3] proved that it is undecidable whether two regular tree languages can be separated by a deterministic tree-walking automaton. For graph-walking automata, both deterministic and nondeterministic, Martynova [13] proved that their non-emptiness problem is NEXPTIME-complete.

The second result of this paper is about *unambiguous tree-walking automata (UTWA)*: these are NTWA, which, for every tree they accept, have a unique accepting computation, while the number of rejecting computations is unrestricted. The unambiguous case has been studied for different models of automata and for different complexity classes, see the survey by Colcombet [7]. Unambiguous automata are an intermediate model between deterministic and nondeterministic ones. In particular, for finite automata on strings, all three types of automata are equal in power, and their relative succinctness has been a subject of much research: see, e.g., the most recent contributions by Indzhev and Kiefer [11] and Göös et al. [10]. On the other hand, for tree-walking automata, DTWA are weaker than NTWA [4], and unambiguous tree-walking automata may theoretically coincide in power with either DTWA or NTWA, or they may be strictly between them. In this paper, we prove that UTWA are weaker than NTWA, while the question of whether they are stronger than DTWA or not remains open.

2 Trees and tree-walking automata

The notion of tree-walking automata is standard, even though it can be presented in various notation. This paper generally adopts the notation used by Bojańczyk and Colcombet [5], with insignificant modifications.

Definition 1. Let Σ be an alphabet of labels. A *binary tree* over Σ is a partial mapping $t: V \rightarrow \Sigma$, where $V \subset \{1, 2\}^*$ is a finite non-empty and prefix-closed set of *nodes*, and t defines the *label* of each node. The empty string $\varepsilon \in V$ is the *root node*, and for each node $v \in V$, if the node $v1$ is in V , it is called the *left child* of v , and v is its *parent*; similarly, if $v2$ is in V , it is the *right child* of v . A node either has both children or none; in the latter case it is called a *leaf*.

The edge between a parent and a child is defined by a function $d: V \times V \rightarrow D$, where $D = \{+1, -1, +2, -2\}$ is the set of direction labels. For every two nodes u and v , such that v is the i -th child of u , let $d(u, v) = +i$ and $d(v, u) = -i$. The function d is undefined on all other pairs.

A node $u \in V$ is said to be *above* a node $v \in V$ if $v = uw$ for some $w \in \{1, 2\}^+$; in this case, v is said to be *below* u . A node u is *to the left* of v if none of them is above the other, and u is lexicographically less than v ; in this case, v is *to the right* of u .

For each node $u \in V$, the *subtree* of t rooted at u is a tree t_u defined by $t_u(v) = t(uv)$ for all $v \in \{1, 2\}^*$ with $t(uv)$ defined.

Some further notation for the trees turns out to be useful. The following function describes the local position of a node in a tree.

Definition 2. Let t be a tree with the set of nodes V . Each node $v \in V$ is assigned a *type*, drawn from the set $\text{Types} = \{\text{root}, 1, 2\} \times \{\text{internal}, \text{leaf}\}$. Define a function $\text{Type}: V \rightarrow \text{Types}$ by $\text{Type}(v) = (x, y)$, where x defines whether v is a left child, a right child or the root, and y specifies if v is a leaf or not.

$$x = \begin{cases} \text{root}, & \text{if } v = \varepsilon \\ 1, & \text{if } v = u1 \\ 2, & \text{if } v = u2 \end{cases} \quad y = \begin{cases} \text{internal}, & \text{if } v1, v2 \in V \\ \text{leaf}, & \text{if } v1, v2 \notin V \end{cases}$$

A (nondeterministic) tree-walking automaton is typically defined as follows. It starts at the root in one of the initial states. At each step it knows the current state and sees the label and the type of the current node. Then, according to the transition function, it nondeterministically decides to proceed to its parent or any of its children, and changes its state. If the automaton ever comes to the root in an accepting state, it accepts.

The definition assumed in this paper follows Bojańczyk and Colcombet [5]. Accordingly, the automaton is invested with the knowledge of the label and the type of the destination node, which are also part of a transition. If the destination node does not have the specified type, that transition cannot be applied. This ability does not give an automaton any extra power, since it could move to the destination node and backwards to collect the same information before making such a transition. This way, transitions become symmetric and can be reversed.

Definition 3. A *nondeterministic tree-walking automaton (NTWA)* is a quintuple $A = (\Sigma, Q, Q_0, \delta, F)$, where

- Σ is a finite alphabet of labels,
- Q is a finite set of states,
- $Q_0 \subseteq Q$ is the set of initial states,
- $\delta \subseteq (Q \times \Sigma \times \text{Types})^2 \times D$ is the transition relation,
- and $F \subseteq Q$ is the set of accepting states.

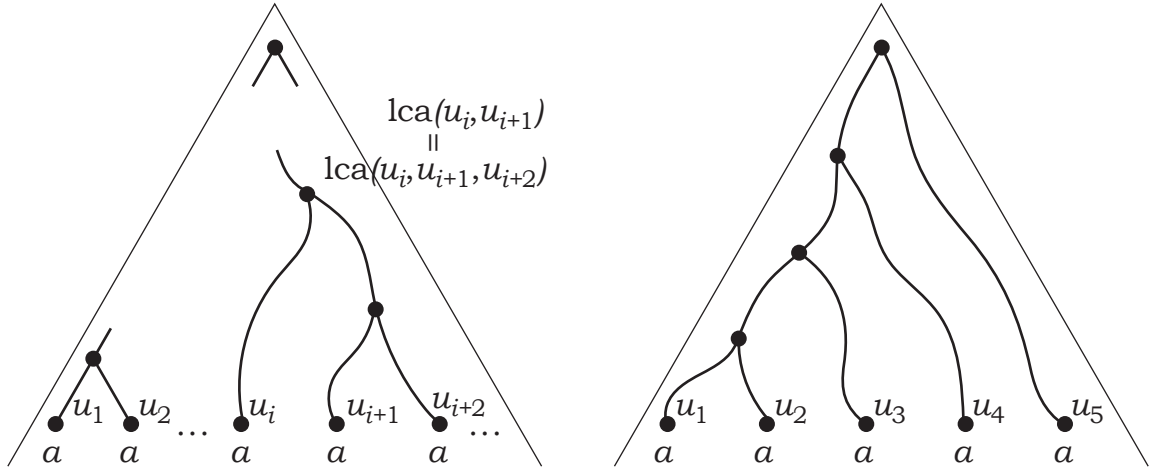


Figure 1: (left) A tree in L ; (right) a tree not in L .

Configurations of the automaton on a tree t with a set of nodes V are pairs (q, v) , with $q \in Q$ and $v \in V$. A *computation* from a configuration (p, u) to a configuration (q, v) is any sequence of the form $(r_0, w_0), (r_1, w_1), \dots, (r_N, w_N)$, where $N \geq 0$, $r_i \in Q$ and $w_i \in V$ for all i , $(r_0, w_0) = (p, u)$, $(r_N, w_N) = (q, v)$, and for every i , with $0 \leq i \leq N - 1$, the configurations (r_i, w_i) and (r_{i+1}, w_{i+1}) are connected by a transition, that is,

$$(r_i, t(w_i), \text{Type}(w_i), r_{i+1}, t(w_{i+1}), \text{Type}(w_{i+1}), d(w_i, w_{i+1})) \in \delta.$$

An *accepting computation* is a computation from an initial configuration (q_0, v_0) , where $q_0 \in Q_0$ is an initial state and $v_0 = \varepsilon$ is the root node, to any accepting configuration of the form (q, v_0) , with $q \in F$. Thus, the automaton accepts only in the root.

A tree t is accepted by the automaton A if there is at least one accepting computation on t . The language recognized by A , denoted by $L(A)$, is the set of all trees the automaton accepts.

3 Separating language

We consider binary trees, with nodes labelled with a or b , where b is the blank symbol. Let all leaves labelled with a in a tree be enumerated from left to right, starting with 1, and denoted by u_1, u_2, \dots

The language L is defined as the set of all trees in which there is a triple of leaves labelled with a , with consecutive numbers $i, i + 1, i + 2$, satisfying the condition $\text{lca}(u_i, u_{i+1}, u_{i+2}) = \text{lca}(u_i, u_{i+1})$, where lca denotes the lowest common ancestor. The form of trees satisfying this condition is illustrated in Figure 1(left), whereas trees that are not in L have the form shown in Figure 1(right). In the figure, blank leaves are omitted along with paths leading to them.

The following theorem is the main result of this paper.

Theorem 1. *There is a nondeterministic tree-walking automaton that recognizes the language L . No nondeterministic tree-walking automaton can recognize the complement of L .*

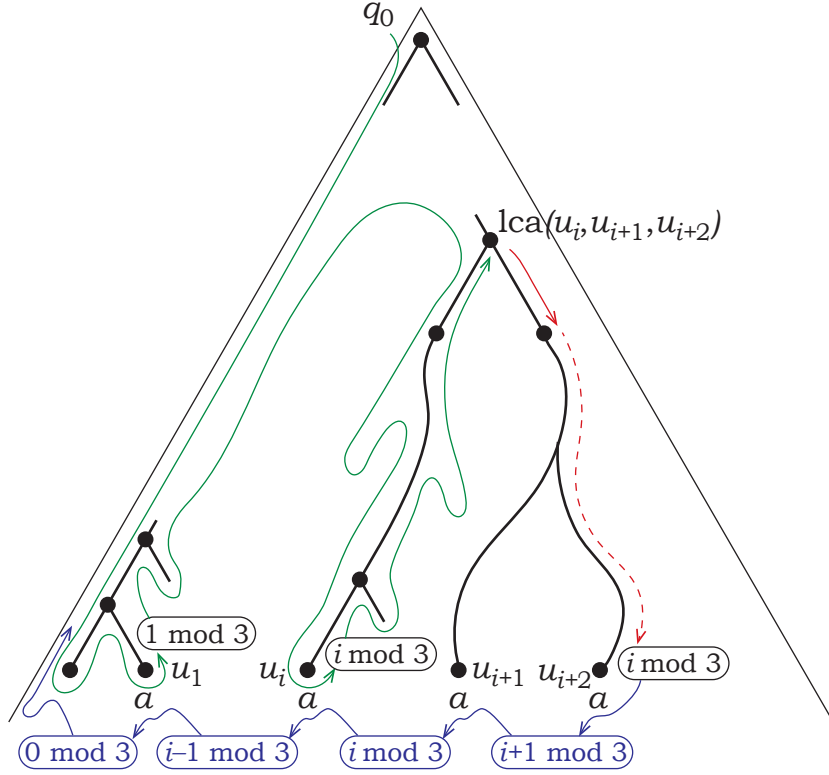


Figure 2: NTWA accepting a tree from L , executing the algorithm from Section 3.

In this section, the first part of the theorem is proved: an NTWA A_L recognizing the language L will be described. This NTWA works using the following algorithm, illustrated in Figure 2. It begins with the traversal of the tree using depth-first search from left to right, in which it counts modulo 3 the number of leaves with label a . At some point, while making a turn (that is, while climbing from a left child to a parent node and then immediately descending into its right child), the automaton nondeterministically decides that this node must be the lowest common ancestor of a suitable triple of leaves (u_i, u_{i+1}, u_{i+2}) , and that the last a -labelled leaf encountered was u_i . At this moment the automaton holds the residue $i \bmod 3$ in its state. Next, the automaton, having descended into the right subtree, nondeterministically guesses the path to the leaf u_{i+2} . It ends its descent in some leaf labelled with a (rejecting otherwise), still remembering the residue $i \bmod 3$. And now the automaton wants to check that the number of the current leaf is equivalent to $i + 2$ modulo 3. To this end, the automaton starts another depth-first search, now from right to left, initially keeping in memory the number $(i + 1) \bmod 3$, and decrementing it by one modulo 3 at each leaf labelled with a . If it finishes the traversal with residue zero in memory, then it accepts.

For every tree in the language L there is an accepting computation of the automaton, in which it just guesses the triple of leaves and their lowest common ancestor correctly, and then guesses the path to u_{i+2} , as described above.

Conversely, assume that the automaton accepts some tree. It should be proved that this tree is in L . At some moment, the automaton assumes that some node v is the lowest common ancestor of three suitable leaves. Let u_i be the last leaf labelled with a visited by the automaton up to this moment. Then, either the leaf u_i is in the left subtree of the

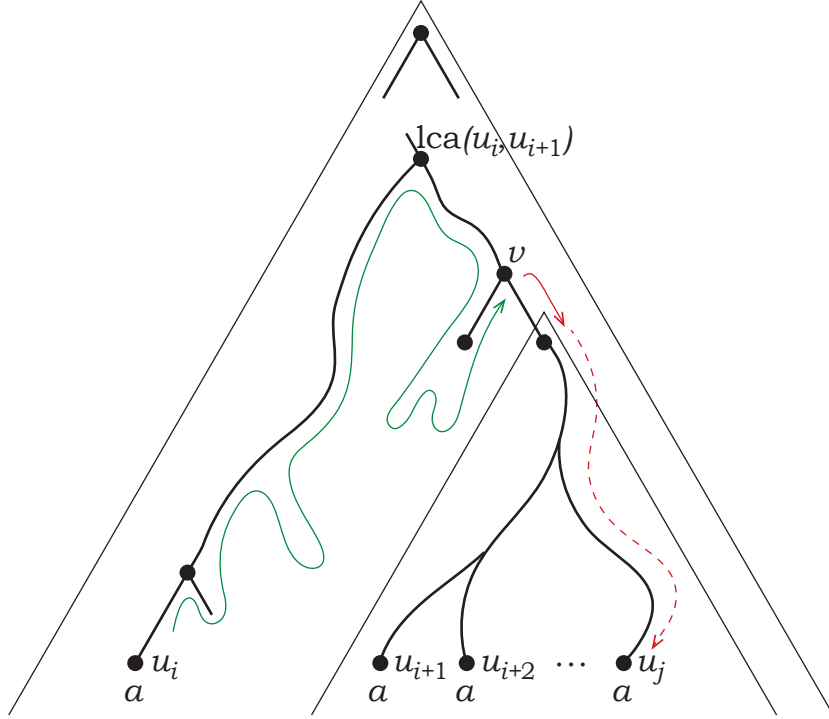


Figure 3: The general form of accepting computation of NTWA A_L .

node v , or u_i is in the left subtree of the lowest common ancestor of u_i and v , whereas v is accordingly in the right subtree of this ancestor (the latter case is illustrated in Figure 3). The automaton continues its computation by descending into the right subtree of v and then nondeterministically chooses an a -labelled leaf in this subtree. This leaf is u_j , with $j > i$, because all leaves with numbers up to i were traversed before visiting v . Therefore, all leaves from u_{i+1} to u_j are in the right subtree of v . Finally, the automaton checks that the number of the leaf it has found has the same residue modulo 3 as $i + 2$. This implies that $j \geq i + 2$, and that both leaves u_{i+1} and u_{i+2} lie in the right subtree of v . Then the lowest common ancestor of u_i and u_{i+1} is precisely the lowest common ancestor of u_i and v , which coincides with the lowest common ancestor of the three nodes u_i , u_{i+1} and u_{i+2} .

4 Tools from the paper by Bojańczyk and Colcombet and their further properties

So the language L is recognized by a nondeterministic tree-walking automaton. Now it will be shown that no NTWA recognizes the complement of L . Let A be an NTWA, let Q be its set of states. The plan is to construct two trees, one not in L and the other in L , so that if the automaton A accepts the former tree, then it also accepts the latter tree. Then the automaton A cannot recognize the complement of the language L .

In constructing these trees and proving that the automaton operates on them as desired, we use the tools developed by Bojańczyk and Colcombet [5]. Namely, the desired two trees are constructed out of the elements defined in their paper, and we also use the basic properties of those elements in our proofs. In this section, the required definitions and lemmata by Bojańczyk and Colcombet [5] are presented, along with several new

lemmata addressing some further basic properties of those elements.

Following Bojańczyk and Colcombet [5], we consider trees with designated holes (ports) and call them *patterns*.

Definition 4. A *pattern* is a binary tree with labels $\{a, b, *\}$, in which the labels a and $*$ may only be used in leaves, and all leaves labelled with $*$ are left children. A pattern must have at least two nodes.

The root of the tree is called the *root port* or *port 0*. All leaves labelled with $*$ are *leaf ports*, enumerated from left to right starting with one. The number of leaf ports in a pattern is called its *rank*. A pattern of rank k , with $k \geq 0$, thus has the set of ports $\{0, 1, \dots, k\}$, and it is called a *k-ary pattern*.

Patterns can be attached to each other by substituting one pattern for a leaf labelled with $*$ in another pattern. This operation is called *composition* of patterns, and is denoted by $\Delta[\Delta_1, \dots, \Delta_k]$: here patterns $\Delta_1, \dots, \Delta_k$ are attached to the leaf ports of a k -ary pattern Δ . If patterns are attached not to all leaf ports, then $*$ is written instead of a pattern to be substituted.

Consider how an automaton can move through a pattern if this pattern is a part of some tree.

Definition 5 (Bojańczyk and Colcombet [4, Defn. 3], [5, Defn. 3]). Let A be an NTWA with a set of states Q , and let Δ be a pattern of rank k . Let $p, q \in Q$ be two states and let $i, j \in \{0, 1, \dots, k\}$ be two ports. A computation of A on Δ that begins in state p in port i and ends in state q in port j , without visiting any ports on the way, and treating ports i and j as left non-leaf children labelled with b , is said to be a *run of type (p, i, q, j)* .

The automaton's *transition relation* over Δ is $\delta_\Delta \subseteq Q \times \{0, 1, \dots, k\} \times Q \times \{0, 1, \dots, k\}$, and it contains the types of all runs of A over Δ (and no other quadruples).

Two patterns Δ and Δ' are called *equivalent* (with respect to A) if they are of the same rank and their transition relations coincide: $\delta_\Delta = \delta_{\Delta'}$.

A computation of zero length is considered as a run, that is, $(p, i, p, i) \in \delta_\Delta$, for all $p \in Q$ and $i \in \{0, 1, \dots, k\}$.

The equivalence relation on patterns is defined so that it respects composition: if some pattern Δ is obtained as a composition of other patterns, and if one of those subpatterns is replaced with an equivalent pattern, then the resulting pattern will be equivalent to Δ .

Most trees constructed in the papers by Bojańczyk and Colcombet [4, 5] are obtained by combining several specifically constructed patterns, defined with respect to an automaton A , which have the following remarkable properties.

Lemma 1 (Bojańczyk and Colcombet [4, Lemma 9], [5, Lemma 3.1]). *Let A be a tree-walking automaton. Then there are patterns $\Delta_0, \Delta_1, \Delta_2$ of rank 0, 1 and 2, respectively, which have no labels a , such that every pattern Δ of rank at most 2 obtained as a composition of any number of patterns $\Delta_0, \Delta_1, \Delta_2$ is equivalent to one of $\Delta_0, \Delta_1, \Delta_2$ (the one of the same rank as Δ).*

In the following, $\Delta_0, \Delta_1, \Delta_2$ are patterns constructed for the automaton A by Lemma 1. One more basic pattern is Δ_a , defined as $\Delta_1[\Delta'_a]$, where Δ'_a is a tree with three nodes: the root and the right leaf are labelled with b , whereas the left leaf has a label a , as in the paper by Bojańczyk and Colcombet [5].

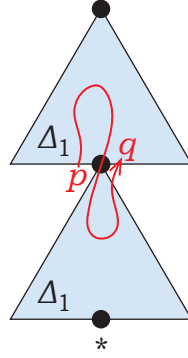


Figure 4: Inner loop $p \rightarrow_\varepsilon q$.

In this paper, the patterns Δ_0 , Δ_1 , Δ_2 and Δ_a , as well as some combinations of a few such patterns, shall be called *elements*, as all trees considered in this paper shall be constructed out of them.

In patterns composed of elements Δ_0 , Δ_1 , Δ_2 , one can attach an element Δ_1 to any port and get an equivalent pattern.

Lemma 2 (Bojańczyk and Colcombet [4, Lemma 8], [5, Fact 3.2]). *Let A be an NTWA. Let a pattern Δ be a composition of any number of elements Δ_0 , Δ_1 , Δ_2 . Then, $\Delta[*], \dots, [*]$ is equivalent to all patterns obtained by attaching Δ_1 to any port: $\Delta_1(\Delta[*], \dots, [*])$, $\Delta[*], \dots, [*], \Delta_1[*], *, \dots, [*]$.*

Consider a computation of the automaton A on a pattern Δ composed of elements Δ_1 and Δ_2 . This computation naturally splits into runs over the constituent elements. The automaton possibly returns to the same ports of some elements multiple times, and this complicates the analysis of such a computation. In order to handle the returns to the same ports, Bojańczyk and Colcombet [5, §3.2] introduced the notion of *inner loop*. An inner loop from a state p to a state q is a computation in the pattern $\Delta_1[\Delta_1[*]]$ starting in the state p at the junction node between two elements Δ_1 , and ending in the state q at the same junction node, without visiting either port of the pattern $\Delta_1[\Delta_1[*]]$ on the way (see Figure 4). The existence of such an inner loop is denoted by $p \rightarrow_\varepsilon q$. The computation in this definition is allowed to be empty, and hence $p \rightarrow_\varepsilon p$ for all p .

Some computations in patterns that return to their point of origin can be shrunk to just inner loops.

Lemma 3 (Bojańczyk and Colcombet [5, Lemma 3.3]). *Let A be an NTWA. Let Δ, Δ' be patterns of nonzero rank obtained as compositions of any number of elements Δ_0 , Δ_1 , Δ_2 . Let the pattern Δ' be attached to the i -th leaf port of Δ . Assume that the automaton A begins at the junction node between Δ and Δ' in some state p , moves over these patterns without visiting their ports except the junction node, and returns to the junction node in some state q . Then $p \rightarrow_\varepsilon q$.*

Every inner loop can be executed at the junction node between any two neighbouring elements Δ_0 , Δ_1 , Δ_2 , Δ_a . Indeed, by Lemma 2 the elements Δ_0 , Δ_1 , Δ_2 are equivalent to themselves extended by attaching Δ_1 to all ports including the root port. As for Δ_a , the pattern $\Delta_1[\Delta_a]$ is equivalent to Δ_a , because $\Delta_1[\Delta_a] = \Delta_1[\Delta_1[\Delta'_a]]$, and $\Delta_1[\Delta_1[*]]$ is equivalent to $\Delta_1[*]$ by Lemma 2.

Then, runs between ports of these elements can be regarded as runs between junction nodes, each containing a pair $\Delta_1[\Delta_1[*]]$ in the vicinity, and hence one can consider *runs extended with loops*, which begin with an inner loop at the port of departure, then make an ordinary run, and finally execute another inner loop at the destination port. For brevity, such runs shall be called *transfers*.

Definition 6 (Bojańczyk and Colcombet [5, Defn. 5]). Let A be an NTWA with a set of states Q . Let Δ be a pattern of rank k , let $p, q \in Q$ and $i, j \in \{0, 1, \dots, k\}$. A *transfer of type* (p, i, q, j) over Δ is a computation that begins with an inner loop $p \rightarrow_\varepsilon p'$, continues with a run of type $(p', i, q', j) \in \delta_\Delta$, and ends with another inner loop $q' \rightarrow_\varepsilon q$, where $p', q' \in Q$ are some states.

The *relation of transfers* over Δ is $\gamma_\Delta \subseteq Q \times \{0, 1, \dots, k\} \times Q \times \{0, 1, \dots, k\}$, and it contains the types of all transfers of A over Δ .

If two patterns Δ and Δ' , composed of Δ_0 , Δ_1 , Δ_2 and Δ_a , are equivalent, then their relations γ_Δ and $\gamma_{\Delta'}$ coincide: indeed, the definition of a transfer depends only on the relation δ , whereas inner loops can be executed at any junction nodes.

A run through a pattern made of elements naturally splits into runs through these elements. Such a partition also can be made for transfers, as follows.

Definition 7. Let a pattern Δ be obtained as a composition of any number of elements Δ_1 and Δ_2 . For every transfer over Δ , its *partition into elementary transfers* over constituent elements Δ_1 and Δ_2 is obtained by first splitting this transfer into inner loops and runs between neighbouring junction nodes, and then attaching every inner loop to the preceding run to obtain a transfer crossing this element (inner loops at the beginning are attached to the following run).

A transfer is called *simple* if, in the above partition, at most one elementary transfer crosses each element.

This notion of a simple transfer is analogous to a simple path in a tree.

Lemma 4. *Let A be an NTWA, and let Δ be a pattern obtained by a composition of any number of elements Δ_1 and Δ_2 . Then, for every transfer over Δ there is a simple transfer of the same type.*

Proof. **TOPROVE 0** □

The following notation for transfers of the automaton A through elements Δ_0 , Δ_1 , Δ_2 and Δ_a is introduced (see Bojańczyk and Colcombet [5, Fig. 5.1]).

$p \circ q$	if $(p, 0, q, 0) \in \gamma_{\Delta_0}$
$p \circ_a q$	if $(p, 0, q, 0) \in \gamma_{\Delta_a}$
$p \downarrow q$	if $(p, 0, q, 1) \in \gamma_{\Delta_1}$
$p \uparrow q$	if $(p, 1, q, 0) \in \gamma_{\Delta_1}$
$p \nearrow q$	if $(p, 1, q, 0) \in \gamma_{\Delta_2}$
$p \nwarrow q$	if $(p, 2, q, 0) \in \gamma_{\Delta_2}$
$p \swarrow q$	if $(p, 0, q, 1) \in \gamma_{\Delta_2}$
$p \searrow q$	if $(p, 0, q, 2) \in \gamma_{\Delta_2}$
$p \curvearrowright q$	if $(p, 2, q, 1) \in \gamma_{\Delta_2}$
$p \curvearrowleft q$	if $(p, 1, q, 2) \in \gamma_{\Delta_2}$

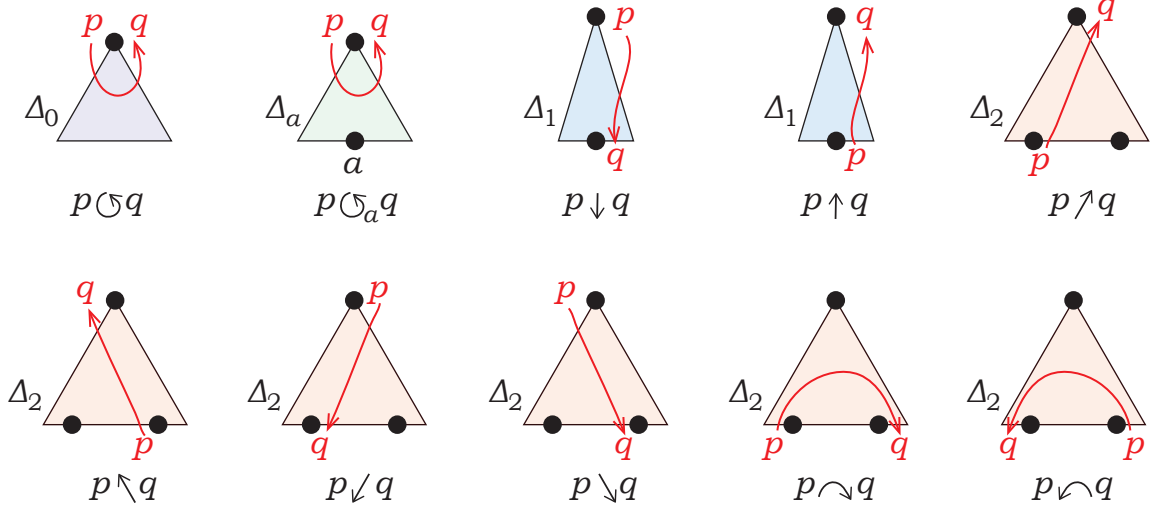


Figure 5: Notation for transfers through elements Δ_0 , Δ_1 , Δ_2 and Δ_a .

This notation is illustrated in Figure 5. Note that the notation $p \diamond q$, with $\diamond \in \{\cup, \dots, \cap\}$, always refers to a transfer from the state p to the state q , regardless of the direction of the arrow. For example, $p \cap q$ denotes that there is a transfer from p at port 2 to q at port 1 in the element Δ_2 .

By Lemma 2, attaching the element Δ_1 to any port of Δ_2 or Δ_1 produces an equivalent element. Therefore, for any transfer continued through an attached element Δ_1 , there is a transfer of the same type through the original element. The following lemma states a few such cases that are used in the paper.

Lemma 5. *Let A be an NTWA with a set of states Q . Then, for all states $p, q, r \in Q$,*

- (a) $p \uparrow q \uparrow r$ implies $p \uparrow r$,
- (b) $p \nearrow q \uparrow r$ implies $p \nearrow r$,
- (c) $p \uparrow q \nearrow r$ implies $p \nearrow r$,
- (d) $p \nwarrow q \uparrow r$ implies $p \nwarrow r$,
- (e) $p \uparrow q \cap r$ implies $p \cap r$.

The converse is also true: if there is a transfer of type (p, i, r, j) in the element Δ_1 or Δ_2 that moves from port i to port j , then a transfer of this type is also possible on an equivalent element obtained by attaching Δ_1 to port i or j . The latter transfer passes through the junction node, and can be split into two transfers.

Lemma 6. *Let A be an NTWA, let Q be its set of states. Then, for all states $p, r \in Q$,*

- (a) $p \nearrow r$ implies $p \nearrow q \uparrow r$, for some $q \in Q$;
- (b) $p \nwarrow r$ implies $p \nwarrow q \uparrow r$, for some $q \in Q$;
- (c) $p \cap r$ implies $p \cap q \downarrow r$, for some $q \in Q$.

Proof. **TOPROVE 1**

□

Another basic lemma says that all transfers between any leaf ports of patterns made of Δ_0 , Δ_1 and Δ_2 can be reproduced on the element Δ_2 .

Lemma 7. *Let A be an NTWA with a set of states Q . Let Δ be a k -ary pattern, with $k \geq 2$, constructed by composition of any number of elements Δ_0 , Δ_1 and Δ_2 . Let $(p, i, q, j) \in \gamma_\Delta$, for some states $p, q \in Q$ and for some port numbers i and j , with $1 \leq i, j \leq k$ and $i \neq j$. Then, if $i < j$, then $p \curvearrowright q$, and if $i > j$, then $p \curvearrowleft q$.*

Proof. **TOPROVE 2** □

The next lemma asserts that, as long as an automaton can move from the left leaf port of Δ_2 to its right leaf port, starting in a state p and ending in a state q , it can either similarly move in every pattern constructed from elements Δ_0 , Δ_1 and Δ_2 from an arbitrary leaf port to the next leaf port in order, or there is an inner loop from state p to state q .

Lemma 8. *Let A be an NTWA with a set of states Q . Let Δ be a pattern of rank k , with $k \geq 2$, made of any number of elements Δ_0 , Δ_1 and Δ_2 . Let $p, q \in Q$ be some states with $p \curvearrowright q$. Then either $(p, i, q, i + 1) \in \gamma_\Delta$ for all i with $1 \leq i \leq k - 1$, or $p \rightarrow_\epsilon q$.*

Proof. **TOPROVE 3** □

A convenient tool used by Bojańczyk and Colcombet [5, Sect. 2.1] is an assumption that a tree-walking automaton is *time-symmetric*, in the sense that for every computation it can make, it can also make a related computation that proceeds in the reverse direction. For a time-symmetric automaton, the number of cases in many proofs can be halved. Furthermore, every nondeterministic tree-walking automaton can be made trivially time-symmetric by adding unreachable states with reversed transitions. This is stated in the following lemma.

Lemma 9. *For every NTWA $B = (\Sigma, Q, Q_0, \delta, F)$ there exists another NTWA $\hat{B} = (\Sigma, \hat{Q}, Q_0, \hat{\delta}, F)$, with $|\hat{Q}|$ even, which recognizes the same set of trees as B , and is time-symmetric, in the sense that there exists a bijection $\tau: \hat{Q} \rightarrow \hat{Q}$, such that for every pattern Δ , for every two ports i, j in the pattern, and for every two states $\hat{p}, \hat{q} \in \hat{Q}$, the following two statements are equivalent.*

1. *There is a run of type (\hat{p}, i, \hat{q}, j) of \hat{B} on Δ .*
2. *There is a run of type $(\tau(\hat{q}), j, \tau(\hat{p}), i)$ of \hat{B} on Δ .*

Furthermore, the same equivalence holds for transfers over Δ .

Proof. **TOPROVE 4** □

The use of time symmetry shall now be illustrated on the following lemma, which states that if the automaton can move through Δ_1 upwards, then it can move upwards through Δ_2 from at least one of the leaf ports using the same states; and the same result holds for downward motion. The proof of the upward case can be found in the paper by Bojańczyk and Colcombet [5], and, for demonstration purposes, the downward case shall now be inferred from the upward case.

Lemma 10 (Bojańczyk and Colcombet [5, Prop. 5.6]). *Let A be a time-symmetric NTWA. Then, for all states p and q , $p \uparrow q$ if and only if $p \nearrow q$ or $p \nwarrow q$. Similarly, $p \downarrow q$ if and only if $p \swarrow q$ or $p \searrow q$.*

Proof. **TOPROVE 5** □

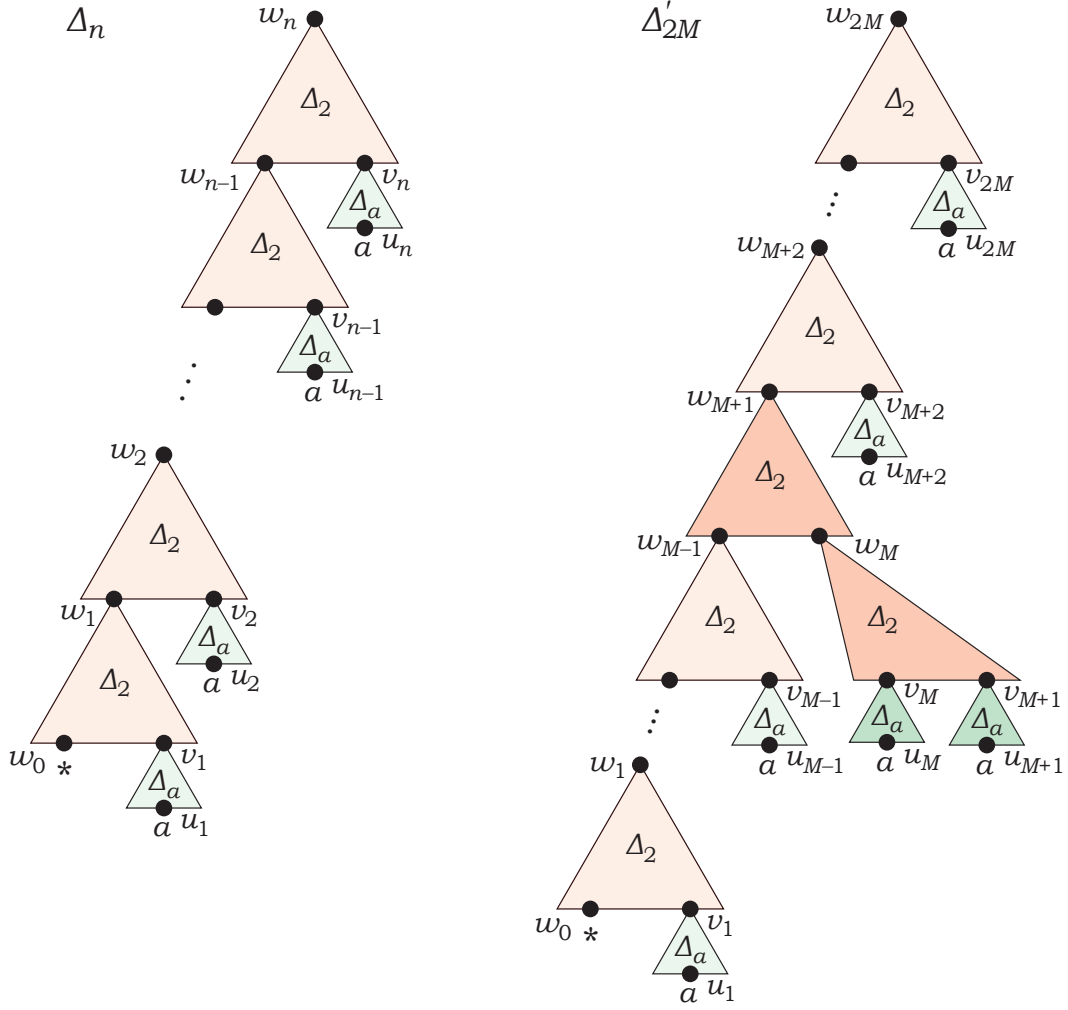


Figure 6: Patterns Δ_n and Δ'_{2M} .

5 Patterns Δ_n and Δ'_{2M}

Let A be an NTWA, it should be proved that it does not recognize the complement of L . Let Q be its set of states and let $n = |Q|$. In view of Lemma 9, it can be safely assumed that the automaton A is time-symmetric, and n is even and is at least 4. This automaton A is fixed for the rest of the paper.

The goal of the proof is to construct two trees: a tree not in L and a modified tree in L , such that if the automaton A accepts the first tree, then it can be lured to accept the second tree. These two trees are constructed in the form of two large patterns of rank 1, which are completed into trees by attaching a root with Δ_1 at the top and Δ_0 at the bottom.

Let M be a number with residue $\frac{n}{2}$ modulo $n!$, such that $M > n^2 + 10n$.

The desired patterns are denoted by Δ_n and Δ'_{2M} and are illustrated in Figure 6. The pattern Δ_n , called *the small correct pattern*, is a chain of n elements $\Delta_2[*]$ of rank 1. The other pattern Δ'_{2M} , *the faulty pattern*, is constructed by attaching to the element $\Delta_2[*]$ two chains of $M - 1$ elements $\Delta_2[*]$ each, one to the leaf port and the other to the root port of the central element. We shall call this central part

the fault, since the resulting tree will be in L due to the different structure of a -leaves in this element.

In both patterns Δ_n and Δ'_{2M} , all leaves labelled with a are enumerated from left to right; denote these leaves by u_1, \dots, u_n in the small correct pattern, and by u_1, \dots, u_{2M} in the faulty pattern. For convenience, the root ports of elements Δ_a containing these leaves are denoted by v_1, \dots, v_n in the small correct pattern and by v_1, \dots, v_{2M} in the faulty pattern, whereas the root ports of the elements $\Delta_2[* , \Delta_a]$, are denoted by w_1, \dots, w_n in the small correct pattern and w_1, \dots, w_{2M} in the faulty pattern. Let the leaf port be w_0 in both patterns. Note that, in the faulty pattern, w_M is the root port of $\Delta_2[\Delta_a, \Delta_a]$, and w_{M+1} is the root port of $\Delta_2[* , \Delta_2[\Delta_a, \Delta_a]]$: these two nodes are enumerated out of order of traversal.

In the small correct pattern Δ_n , all triples of consecutive a -leaves u_i, u_{i+1}, u_{i+2} do not satisfy $\text{lca}(u_i, u_{i+1}, u_{i+2}) = \text{lca}(u_i, u_{i+1})$, whereas in Δ'_{2M} , there is a triple u_{M-1}, u_M, u_{M+1} with $\text{lca}(u_{M-1}, u_M, u_{M+1}) = \text{lca}(u_{M-1}, u_M)$. Therefore, if the small correct pattern Δ_n is replaced with Δ'_{2M} in the tree $\text{root}[\Delta_1[\Delta_n[\Delta_0]]]$ that is not in L , then the resulting tree will be in L . The goal is to prove that if the automaton A accepts the tree $\text{root}[\Delta_1[\Delta_n[\Delta_0]]]$, then it also accepts the tree $\text{root}[\Delta_1[\Delta'_{2M}[\Delta_0]]]$. This will show that the automaton A does not recognize the complement of L , which is enough to prove Theorem 1.

Consider any accepting computation of the automaton A on the tree $\text{root}[\Delta_1[\Delta_n[\Delta_0]]]$. This computation is split into segments between visits to the ports of the small correct pattern Δ_n . Segments that lie outside Δ_n can be exactly replicated in the tree $\text{root}[\Delta_1[\Delta'_{2M}[\Delta_0]]]$. The rest of the computation is formed by runs through the small correct pattern Δ_n , which have to be reproduced on the faulty pattern Δ'_{2M} . That is, for every run of type $(p, i, q, j) \in \delta_{\Delta_n}$ one should prove that $(p, i, q, j) \in \delta_{\Delta'_{2M}}$. Then it is enough to prove the following lemma.

Main Lemma. *Let A be a time-symmetric nondeterministic tree-walking automaton, with the set of states Q of even size $n \geq 4$, operating on binary trees with labels $\{a, b\}$, let $\Delta_0, \Delta_1, \Delta_2$ and Δ_a be the elements constructed for this automaton as in Section 4, and let the patterns Δ_n and Δ'_{2M} be as defined above. Then $\delta_{\Delta_n} \subseteq \delta_{\Delta'_{2M}}$.*

Note that the faulty pattern Δ'_{2M} begins and ends with two subpatterns Δ_n , that is, it can be represented as $\Delta_n[\Delta'_{2M-2n}[\Delta_n[*]]]$. This implies that if the automaton can return to the same port in Δ_n , changing its state from p to q , then it can do the same in Δ'_{2M} . That is, if $(p, i, q, i) \in \delta_{\Delta_n}$, for some states $p, q \in Q$ and some port $i \in \{0, 1\}$, then $(p, i, q, i) \in \delta_{\Delta'_{2M}}$.

Now it remains to prove that for every run through the small correct pattern Δ_n from the leaf port to the root port (or from the root port to the leaf port), there is a run of the same type through the faulty pattern Δ'_{2M} . Since the automaton A is time-symmetric, the proof will be given only for runs from leaf to root, that is, that $(p, 1, q, 0) \in \delta_{\Delta_n}$ implies $(p, 1, q, 0) \in \delta_{\Delta'_{2M}}$. Then, for runs from root to leaf, if $(p, 0, q, 1) \in \delta_{\Delta_n}$, then $(\tau(q), 1, \tau(p), 0) \in \delta_{\Delta_n}$, which is a run from leaf to root, and its existence implies $(\tau(q), 1, \tau(p), 0) \in \delta_{\Delta'_{2M}}$ by the case of an upward run. Finally, by time symmetry again, $(p, 0, q, 1) \in \delta_{\Delta'_{2M}}$.

Some states q_{start} and q_{finish} with $(q_{\text{start}}, 1, q_{\text{finish}}, 0) \in \delta_{\Delta_n}$ are fixed for the rest of the proof. And it should be proved that $(q_{\text{start}}, 1, q_{\text{finish}}, 0) \in \delta_{\Delta'_{2M}}$.

6 The correct pattern Δ_{2M} and a run of type $(q_{\text{start}}, 1, q_{\text{finish}}, 0)$ through it

The ultimate goal is to construct a run of type $(q_{\text{start}}, 1, q_{\text{finish}}, 0)$ through the faulty pattern Δ'_{2M} , which is much larger than the small correct pattern Δ_n . In this section, a run of this type is constructed for another pattern: *the correct pattern* Δ_{2M} , which is an inflated version of Δ_n . It is composed of $2M$ elements $\Delta_2[* , \Delta_a]$ forming a chain. In the pattern Δ_{2M} , there are nodes u_i, v_i, w_i , with $i \in \{1, \dots, 2M\}$, and w_0 , which are defined analogously to the nodes of Δ_n .

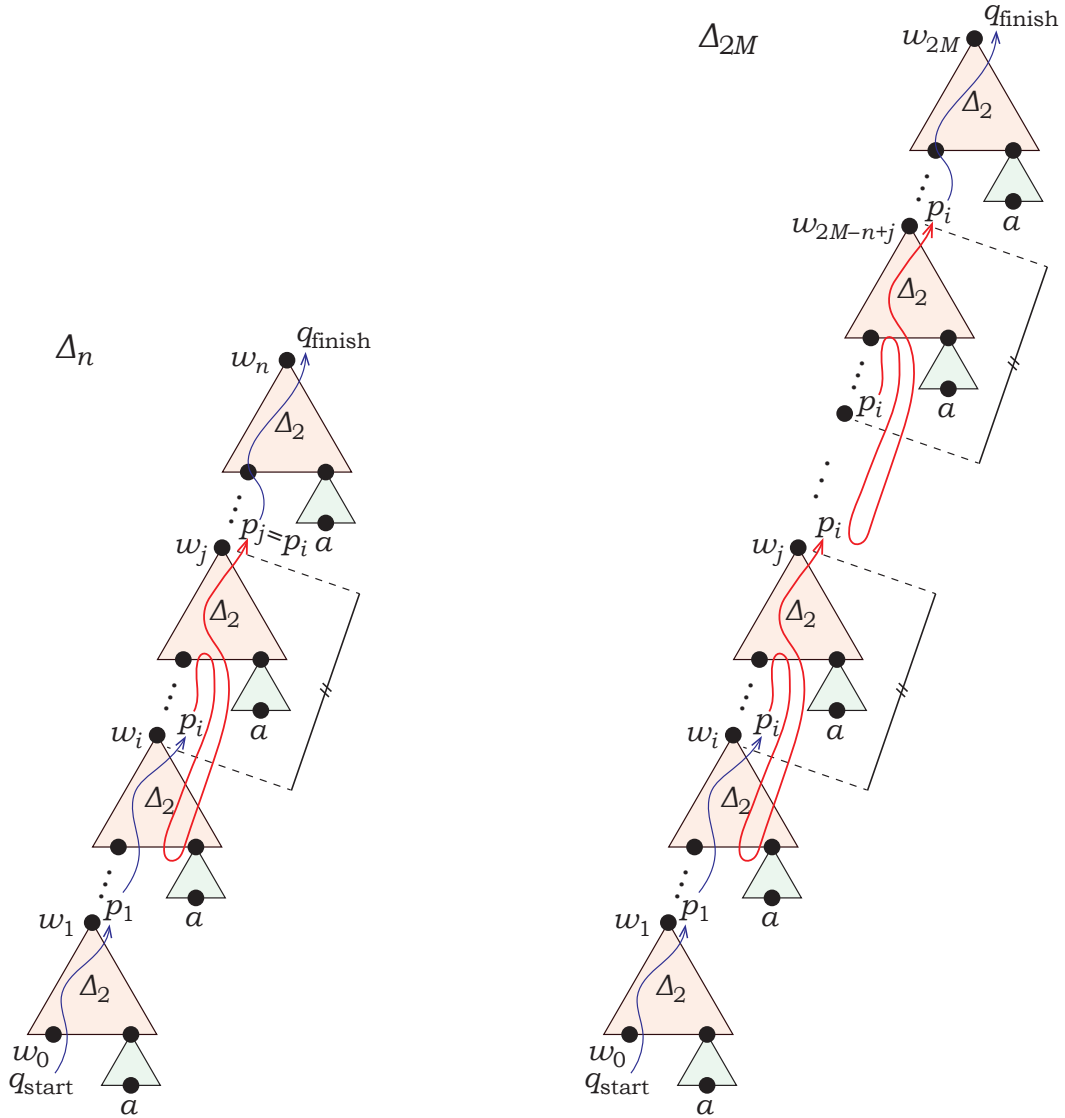


Figure 7: (left) A run of type $(q_{\text{start}}, 1, q_{\text{finish}}, 0)$ through Δ_n , with its part from (p_i, w_i) to (p_i, w_j) to be used as a period; (right) A run of the same type through Δ_{2M} , with multiple iterations of the period.

The first claim is that there is a run of type $(q_{\text{start}}, 1, q_{\text{finish}}, 0)$ through the correct pattern Δ_{2M} , which behaves periodically in the middle of the pattern, and which is obtained by taking a run through the small correct pattern Δ_n and repeating its part

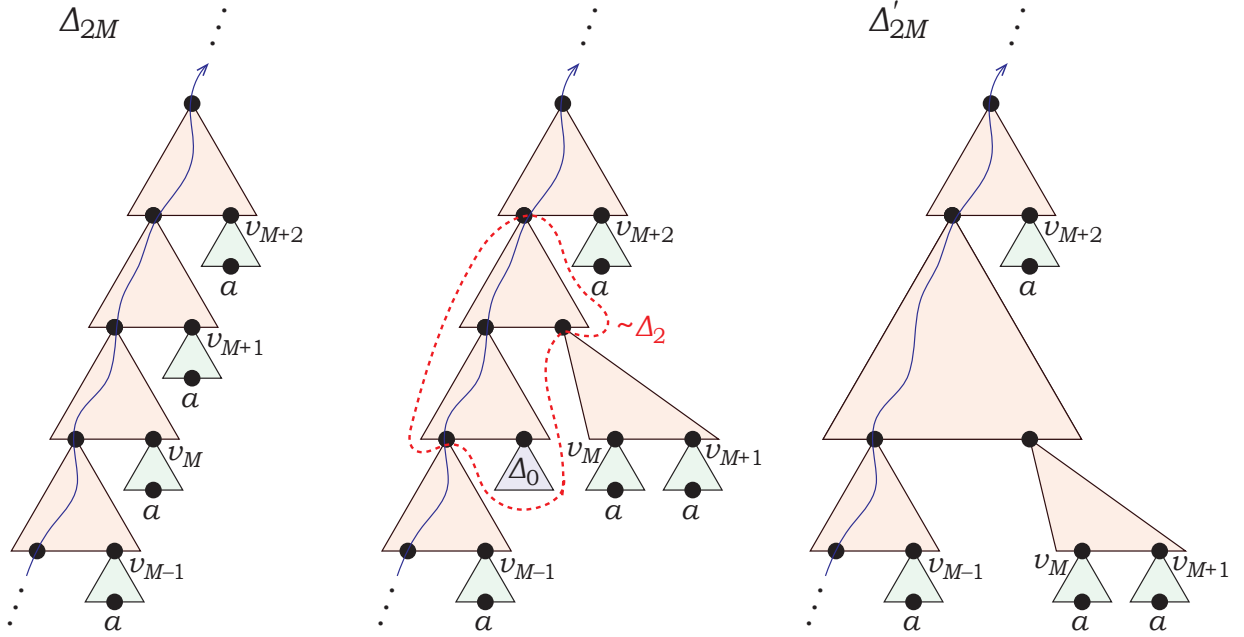


Figure 8: (left) Computation on Δ_{2M} never visiting any Δ_a ; (middle) Two of Δ_a replaced with Δ_0 and $\Delta_2[\Delta_a, \Delta_a]$; (right) An element replaced with an equivalent one, resulting with Δ'_{2M} .

periodically, as illustrated in Figure 7.

Claim 1. *There is a run of type $(q_{start}, 1, q_{finish}, 0)$ through the correct pattern Δ_{2M} , such that, for some i with $0 \leq i \leq n-1$, after the first visit to the node w_i the automaton starts periodically repeating a certain sequence of transitions, with the prefix before periodic part and the first iteration together confined to the bottom n elements $\Delta_2[* , \Delta_a]$. This periodic behaviour leads it to a node with number greater than $2M - n$, and the last iteration together with the suffix after the periodic part are confined to the top n elements $\Delta_2[* , \Delta_a]$ of Δ_{2M} .*

Proof. **TOPROVE 6** □

The goal is to take the computation on the correct pattern Δ_{2M} from Claim 1, and to reproduce it on the faulty pattern Δ'_{2M} , so that the periodically repeated sequence will pass by the fault, that is, the element $\Delta_2[* , \Delta_2[\Delta_a, \Delta_a]]$, paying no attention to the difference.

In order to distinguish the faulty pattern from the correct pattern, the periodically repeated sequence should visit some of the elements Δ_a attached from the right. In the next claim it is shown that if none of the elements Δ_a are visited, then the computation can be reproduced on the faulty pattern, and in this case the proof of the Main Lemma will be completed.

Claim 2. *Consider some run of type $(q_{start}, 1, q_{finish}, 0)$ through the correct pattern Δ_{2M} , as constructed in Claim 1. Then, if its periodic part never visits any element Δ_a , there is a run of type $(q_{start}, 1, q_{finish}, 0)$ through the faulty pattern Δ'_{2M} .*

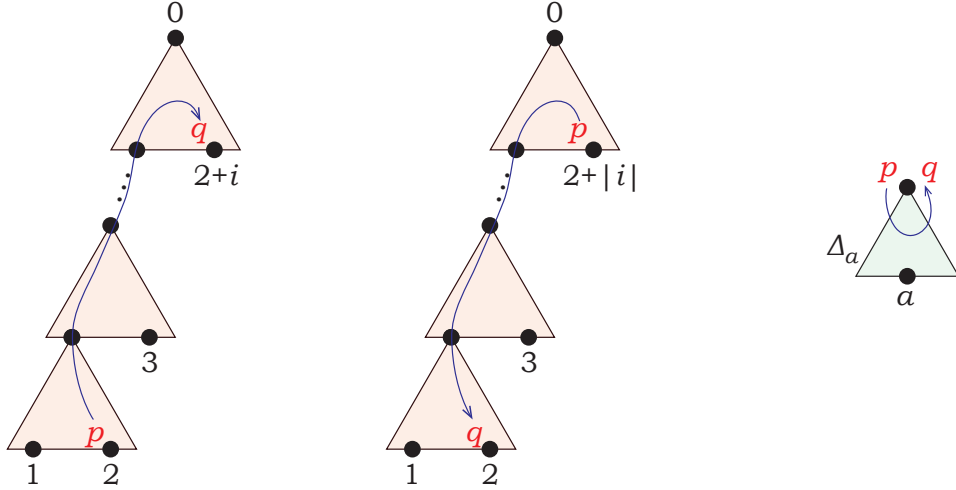


Figure 9: A proper step of type (i, p, q) : (left) $i > 0$; (middle) $i < 0$; (right) $i = 0$.

Proof. **TOPROVE 7** □

If no elements Δ_a are visited in the periodic part of the computation, then the proof of the Main Lemma is completed by Claim 2. From now on, it is assumed that some elements Δ_a are visited in the periodic part. The plan is to represent this periodic part as a sequence of segments between visits to Δ_a . It will be proved that such segments can be executed by simple transfers. These segments shall be called *proper steps*.

Definition 8. Let i be an integer such that $i \neq 0$, and let $p, q \in Q$ be two states. Consider the pattern obtained by attaching $|i| + 1$ elements Δ_2 into a chain, with every next element attached to the left leaf port of the previous one, as illustrated in Figure 9. Then a *proper step of type (i, p, q)* is a simple transfer through this pattern, of type $(p, 2, q, 2+i)$ if $i > 0$, or of type $(p, 2+|i|, q, 2)$ if $i < 0$.

For $i = 0$ a proper step of type $(0, p, q)$ is a transfer of type $(p, 0, q, 0)$ on Δ_a .

The number i is called the *pace* of the proper step.

This definition can be conveniently reformulated in the notation for transfers over elements Δ_2 .

Claim 3. Let $p, q \in Q$ be two states, and let i be a non-zero integer. If $i > 0$, then a proper step of type (i, p, q) exists if and only if there are intermediate states p_1, \dots, p_i , such that $p \nwarrow p_1 \nearrow p_2 \nearrow \dots \nearrow p_i \curvearrowright q$. If $i < 0$, then a proper step of type (i, p, q) exists if and only if $p \curvearrowleft p_1 \swarrow p_2 \swarrow \dots \swarrow p_i \searrow q$, for some intermediate states p_1, \dots, p_i .

Proof. **TOPROVE 8** □

A proper step of type (i, p, q) can be used anywhere in the pattern Δ_{2M} : that is, for all integers x satisfying $1 \leq x \leq 2M$ and $1 \leq x+i \leq 2M$, there is a sequence of transfers on elements Δ_2 , that moves the automaton on the pattern Δ_{2M} from the configuration (p, v_x) to the configuration (q, v_{x+i}) without visiting root ports of any Δ_a on the way, and never traversing any Δ_2 twice.

Next, a run through Δ_{2M} constructed in Claim 1 is modified so that its periodic part consists of proper steps.

Claim 4. *There is a run of type $(q_{start}, 1, q_{finish}, 0)$ on the pattern Δ_{2M} that first comes to some node $v_{i'}$, with $i' \leq n$, in some state \hat{q} , having visited at most n bottom elements $\Delta_2[* , \Delta_a]$ of Δ_{2M} by that time. Next, the computation periodically repeats some sequence of proper steps, with each proper step having pace strictly between $-2n$ and $2n$. Every iteration of the period has pace at most n , that is, it shifts the automaton by at most n elements $\Delta_2[* , \Delta_a]$. Furthermore, every iteration involves at most $2n$ consecutive elements $\Delta_2[* , \Delta_a]$. The periodic part of the computation leads the automaton to some node $v_{i''}$, with $i'' > 2M - n$, in the same state \hat{q} , and after that the automaton finishes the run visiting at most n top elements $\Delta_2[* , \Delta_a]$.*

Proof. **TOPROVE 9** □

The next question is how proper steps of different types pass by the *fault*, that is, the element $\Delta_2[* , \Delta_2[\Delta_a, \Delta_a]]$, which distinguishes the faulty pattern Δ'_{2M} from the correct pattern Δ_{2M} .

7 Shrinking and stretching proper steps to bypass the fault

In this section it will be shown that proper steps that pay attention to the fault in the faulty pattern Δ'_{2M} can be shrunk, or sometimes stretched, so that they bypass the fault.

The possibility of shrinking is established in the following form: if the automaton can move forward or backward by i elements $\Delta_2[* , \Delta_a]$ on the correct pattern Δ_{2M} , then it can either do the same on the faulty pattern, or it can move by $i - 1$ elements on the correct pattern.

Claim 5. *Assume that a proper step of type (i, p, q) exists for $-2n < i < 2n$, $i \neq 0$ and $p, q \in Q$. Then at least one of two conditions holds:*

- I. for every integer x , such that $1 \leq x \leq 2M$ and $1 \leq x + i \leq 2M$, the automaton A can move on the faulty pattern Δ'_{2M} from configuration (p, v_x) to configuration (q, v_{x+i}) ;*
- II. there is a proper step of type $(i - 1, p, q)$ for $i > 0$, and of type $(i + 1, p, q)$ for $i < 0$.*

Proof. **TOPROVE 10** □

The next claim is that sometimes proper steps can be not only shrunk, but also stretched: if the automaton can shift by i elements upwards on the correct pattern Δ_{2M} , then on the faulty pattern Δ'_{2M} the automaton can either move upwards by the same distance without noticing the fault, or it can move one element further, or it can jump from anywhere to anywhere.

Claim 6. *Assume that there is a proper step of type (i, p, q) , for some integer i with $0 < i < 2n$, and for some two states $p, q \in Q$. Let $x \geq 1$ be an integer bounded as $x + i \leq 2M$. Then at least one of the following three conditions holds:*

- I. the automaton A can move on the faulty pattern Δ'_{2M} from configuration (p, v_x) to configuration (q, v_{x+i}) ;*

II. the automaton can move on Δ'_{2M} from configuration (p, v_x) to configuration (q, v_{x+i+1}) ;

III. for all y, z , such that $1 \leq y < M$ and $M + 1 < z \leq 2M$, the automaton can move on Δ'_{2M} from configuration (p, v_y) to configuration (q, v_z) .

Proof. **TOPROVE 11** □

8 How to move through the faulty pattern Δ'_{2M} without noticing the fault

In this section, the proof of the Main Lemma will be completed, along with the whole proof of Theorem 1 stating that the nondeterministic tree-walking automata are not closed under complementation. It remains to prove that $(q_{\text{start}}, 1, q_{\text{finish}}, 0) \in \delta_{\Delta'_{2M}}$, using Claims 4, 5 and 6.

The desired computation from state q_{start} in the leaf port of the faulty pattern Δ'_{2M} to state q_{finish} in its root port is constructed as follows. Consider the computation on the correct pattern Δ_{2M} from Claim 4: first, the automaton moves from q_{start} in the leaf port to the node $v_{i'}$, for some $i' \leq n$, in some state \hat{q} , having visited only the bottom n elements $\Delta_2[* , \Delta_a]$. Next, the automaton A repeats periodically some sequence of proper steps, each with pace strictly between $-2n$ and $2n$. Each iteration of the repeated sequence is contained in some $2n$ consecutive elements $\Delta_2[* , \Delta_a]$. Eventually the automaton comes to some node $v_{i''}$, for some $i'' > 2M - n$, in the same state \hat{q} . And then it finishes its computation in the root port in the state q_{finish} , while visiting only the top n elements $\Delta_2[* , \Delta_a]$. The computation of the automaton A on the correct pattern Δ_{2M} described above is fixed for the rest of this section. And now the goal is to modify this computation to reproduce it on the faulty pattern Δ'_{2M} .

Since $M > n$, the faulty pattern Δ'_{2M} begins and ends with n elements $\Delta_2[* , \Delta_a]$, just like the correct pattern Δ_{2M} . Therefore, the automaton A , working on Δ'_{2M} , can repeat the first and the last parts of the original computation: it can move up to configuration $(\hat{q}, v_{i'})$, and it can finish the computation from configuration $(\hat{q}, v_{i''})$. It remains to prove that the automaton A can also move from configuration $(\hat{q}, v_{i'})$ to configuration $(\hat{q}, v_{i''})$ on the faulty pattern Δ'_{2M} .

The next claim considers the simple case when one of the proper steps in the periodically repeated sequence can be stretched to almost the entire faulty pattern Δ'_{2M} .

Claim 7. *Assume that one of the proper steps in the periodically repeated sequence has type (i, p, q) , with $0 < i < 2n$ and $p, q \in Q$, and satisfies Condition III from Claim 6; that is, for all y and z , with $1 \leq y < M$ and $M + 1 < z \leq 2M$, the automaton A can move on the faulty pattern Δ'_{2M} from configuration (p, v_y) to configuration (q, v_z) . Then A can move on Δ'_{2M} from configuration $(\hat{q}, v_{i'})$ to configuration $(\hat{q}, v_{i''})$.*

Proof. **TOPROVE 12** □

Thus, if a proper step satisfying Condition III from Claim 6 is used in the periodically repeated part of the computation, then the automaton A can move on the faulty pattern Δ'_{2M} from configuration $(\hat{q}, v_{i'})$ to configuration $(\hat{q}, v_{i''})$, and the proof of the Main Lemma

in this case is completed. In the rest of the proof, it is assumed that there are no such proper steps in the periodically repeated sequence.

The next claim is that if the automaton A works on the faulty pattern Δ'_{2M} , then it may bypass the fault and get to some node v_j after the fault in the state \hat{q} . It will come not necessarily to one of the nodes to which the original computation on the correct pattern Δ_{2M} arrives in the state \hat{q} ; what is important is that it comes in this state to *some* node far beyond the fault.

Claim 8. *The automaton A operating on the faulty pattern Δ'_{2M} may move from configuration (\hat{q}, v_i) to some configuration (\hat{q}, v_j) , where j is a number with $M + 2n + 1 < j < M + 8n$.*

Proof. **TOPROVE 13** □

Now everything is prepared for the final step of the proof of the Main Lemma. It has been proved that on the faulty pattern Δ'_{2M} the automaton passes by the fault and arrives in the state \hat{q} to some node v_j , with $M + 2n + 1 < j < M + 8n$. The original computation on the correct pattern Δ_{2M} regularly visits nodes in this region in the state \hat{q} , but the node v_j need not be one of those nodes. The idea is to continue the computation on Δ'_{2M} from v_j , so that it gets back on the track of the periodic computation on Δ_{2M} . For that, the automaton should compensate for the *shift* of v_j relative to the nearest node visited in the state \hat{q} on Δ_{2M} . If the shift is zero, then this is all, the Main Lemma is proved. And if coming with zero shift is impossible, then there is a proper step in the periodically repeated sequence which can be shrunk, and the shift will be compensated by using this shrunk proper step in several iterations of the period.

Proof. **TOPROVE 14** □

9 UTWA are weaker than NTWA

An *unambiguous tree-walking automaton* (UTWA) is a nondeterministic tree-walking automaton that has at most one accepting computation on each input tree. In this section, it is proved that UTWA are strictly weaker than NTWA.

The separating language L , defined in Section 3, is the same as in the rest of this paper. By Theorem 1, it is recognized by an NTWA A_L .

Theorem 2. *The class of tree languages recognized by unambiguous tree-walking automata is strictly smaller than the class of languages recognized by nondeterministic tree-walking automata. In particular, no UTWA recognizes the language L .*

Proof. **TOPROVE 15** □

10 A deterministic one-pebble tree-walking automaton recognizing L

It is worth noting that the language L used in this paper can be recognized by a deterministic tree-walking automaton with one pebble (see Bojańczyk et al. [6] for a precise definition of this model).

Theorem 3. *There is a deterministic tree-walking automaton with one pebble recognizing the language L .*

Proof. **TOPROVE 16** □

Since the family of one-pebble deterministic tree-walking automata is closed under complementation [14], the complement of the language L is recognized by such an automaton as well.

Thus, the complement of L is a tree language recognized by a one-pebble deterministic tree-walking automaton, but not by any NTWA.

11 Conclusion

Besides proving the non-closure of nondeterministic tree-walking automata (NTWA) under complementation, this paper also provides partially different proofs of the original results by Bojańczyk and Colcombet [4, 5]. First, since deterministic tree-walking automata (DTWA) are closed under complementation, the result of this paper implies that NTWA cannot be determinized [4]. Secondly, bottom-up tree-walking automata are closed under complementation as well, and hence they are stronger than NTWA [5].

The second result of this paper, that is, that unambiguous tree-walking automata (UTWA) are weaker than nondeterministic ones, leaves a few related questions to investigate. Most importantly, it remains unknown whether UTWA are any more powerful than DTWA. If these families turn out to be different, then another question will arise, whether UTWA are closed under complementation. Furthermore, the same questions can be asked about *unambiguous graph-walking automata* (UGWA): if they can be determinized, then so can be UTWA, but it could also be possible that UTWA can be determined whereas UGWA cannot.

Acknowledgement

This work was supported by the Russian Science Foundation, project 23-11-00133.

References

- [1] A. V. Aho, J. D. Ullman, “Translations on a context free grammar”, *Information and Control*, 19:5 (1971), 439–475.
- [2] M. Bojańczyk, “Tree-walking automata”, *Language and Automata Theory and Applications* (LATA 2008, Tarragona, Spain, March 13–19, 2008), LNCS 5196, 1–2. Extended version available at <https://www.mimuw.edu.pl/~bojan/upload/conflataBojanczyk08.pdf>.
- [3] M. Bojańczyk, “It is undecidable if two regular tree languages can be separated by a deterministic tree-walking automaton”, *Fundamenta Informaticae*, 154:1–4 (2017), 37–46.

- [4] M. Bojańczyk, T. Colcombet, “Tree-walking automata cannot be determinized”, *Theoretical Computer Science*, 350:2–3 (2006), 164–173.
- [5] M. Bojańczyk, T. Colcombet, “Tree-walking automata do not recognize all regular languages”, *SIAM Journal on Computing*, 38:2 (2008), 658–701.
- [6] M. Bojańczyk, M. Samuelides, T. Schwentick, L. Segoufin, “Expressive power of pebble automata”, *Automata, Languages and Programming, 33rd International Colloquium* (ICALP 2006, Venice, Italy, 9–16 July 2006), vol. 1, LNCS 4051, 157–168.
- [7] T. Colcombet, “Unambiguity in automata theory”, *Descriptional Complexity of Formal Systems* (DCFS 2015, Waterloo, Ontario, Canada, June 25–27, 2015), LNCS 9118, 3–18.
- [8] J. Engelfriet, H. J. Hoogeboom, “Tree-walking pebble automata”, *Jewels are Forever, Contributions on Theoretical Computer Science in Honor of Arto Salomaa*, 1999, 72–83.
- [9] J. Engelfriet, H. J. Hoogeboom, “Automata with nested pebbles capture first-order logic with transitive closure”, *Logical Methods in Computer Science* 3:2–3 (2007), 1–27.
- [10] M. Göös, S. Kiefer, W. Yuan, “Lower bounds for unambiguous automata via communication complexity”, *49th International Colloquium on Automata, Languages, and Programming* (ICALP 2022, July 4–8, 2022, Paris, France), LIPIcs 229, 126:1–126:13.
- [11] E. Indzhev, S. Kiefer, “On complementing unambiguous automata and graphs with many cliques and cocliques”, *Information Processing Letters*, 177 (2022), article 106270.
- [12] M. Kunc, A. Okhotin, “Reversibility of computations in graph-walking automata”, *Information and Computation*, 275 (2020), article 104631.
- [13] O. Martynova, “Complexity of the emptiness problem for graph-walking automata and for tilings with star subgraphs”, *Information and Computation*, 296 (2024), article 105127.
- [14] A. Muscholl, M. Samuelides, L. Segoufin, “Complementing deterministic tree-walking automata”, *Information Processing Letters*, 99:1 (2006), 33–39.
- [15] F. Neven, T. Schwentick, “On the power of tree-walking automata”, *Information and Computation*, 183:1 (2003), 86–103.
- [16] M. Samuelides, L. Segoufin, “Complexity of pebble tree-walking automata”, *FCT* 2007, 458–469.
- [17] M. Sipser, “Halting space-bounded computations”, *Theoretical Computer Science*, 10:3 (1980), 335–338.
- [18] M. Veanes, “On computational complexity of basic decision problems of finite tree automata”, Technical Report 133, Uppsala University, Computing Science Department, 1997.