Nondeterministic Tree-Walking Automata Are Not Closed Under Complementation

Olga Martynova
Department of Mathematics and Computer Science, St. Petersburg State University, Russia

Alexander Okhotin
Department of Mathematics and Computer Science, St. Petersburg State University, Russia

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 nondeterministic tree-walking automata are stronger than unambiguous tree-walking automata.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory; Theory of computation → Models of computation

Keywords and phrases Finite automata, tree-walking automata, complementation

Digital Object Identifier 10.4230/LIPIcs.ICALP.2025.168

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Funding Research supported by the Russian Science Foundation, project 23-11-00133.

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 considered three main problems on tree-walking automata stated by Neven [15]: 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.

The common idea of the two papers of Bojańczyk and Colcombet [4, 5] is using trees made out of specifically constructed elements, called *patterns*, and then showing that tree-walking automata cannot detect some rotations in the branching structure of such trees. They prove that DTWA cannot detect any rotations at all, whereas NTWA cannot detect some rotations that a bottom-up tree automaton can detect. In our paper, we reuse the tools of Bojańczyk and Colcombet [4, 5], and construct a new separating language defined by a certain condition on the branching structure. Our proof ultimately uses two trees that differ only by a single rotation.

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 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 [16].

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 [18] 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 [19]. Samuelides and Segoufin [17] 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]. 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 $t\colon V \to \Sigma$, 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 $d\colon V \times V \to D$, 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* $\text{Type}\colon V \to \text{Types}$ *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.*

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

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 to 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. 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.*

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 $(r_0, w_0), (r_1, w_1), \ldots, (r_N, w_N)$, where $N \geqslant 0$, $r_i \in Q$ 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 $0 \leqslant i \leqslant N - 1$, the configurations (r_i, w_i) and (r_{i+1}, w_{i+1}) are connected by a transition, that is,*

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

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 at the root.*

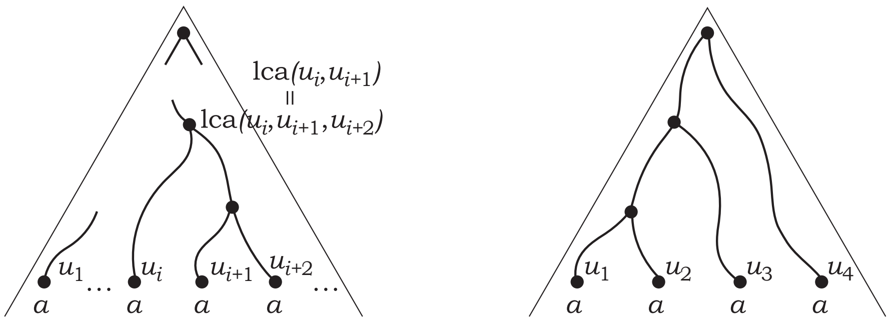

Figure 1 (left) A tree in L; (right) a tree not in L.

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 $u_1, u_2, \ldots$

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

Sketch of a proof of the first part. An NTWA A_L recognizing the language L 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 2 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 $i \bmod 2$ 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 $i \bmod 2$. And now the automaton wants to check that the number of the current leaf is equivalent to i modulo 2. To this end, the automaton starts another depth-first search, now from right to left, initially keeping in memory the number $(i+1) \bmod 2$, and decrementing it by one modulo 2 at each leaf labelled with a. If it finishes the traversal with residue zero in memory, then it accepts.

If the tree is not in L, then the automaton cannot accept regardless of which node it believes to be the least common ancestor (the proof is omitted). ◀

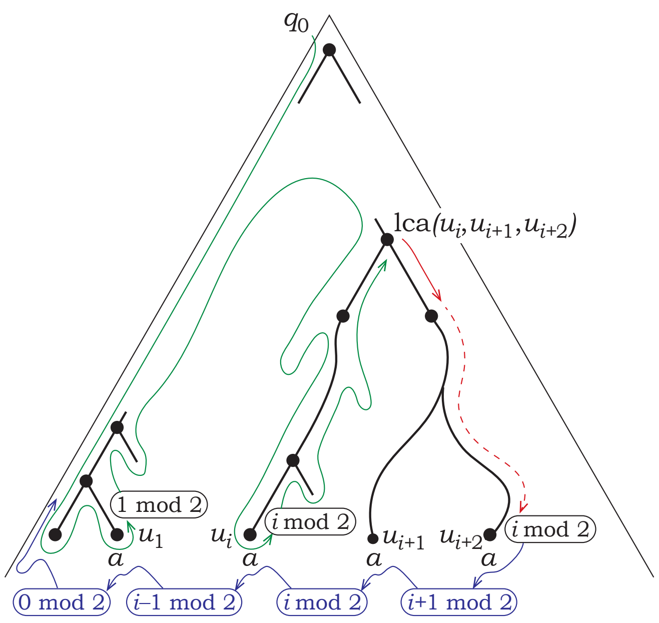

Figure 2 NTWA accepting a tree from L, executing the algorithm from Section 3.

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 tools developed by Bojańczyk and Colcombet [5]. Namely, the desired two trees are constructed out of elements defined in their paper, and we also use 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 5.** *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* $k \geqslant 0$*, 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 6** (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 $i, j \in \{0, 1, \ldots, k\}$ 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 $\delta_\Delta \subseteq Q \times \{0, 1, \ldots, k\} \times Q \times \{0, 1, \ldots, k\}$, 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 $i \in \{0, 1, \ldots, k\}$.

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 7** (Bojańczyk and Colcombet [4, Lemma 9], [5, Lemma 3.1]). *Let A be a tree-walking automaton. Then there are patterns Δ_0, Δ_1, Δ_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 Δ_0, Δ_1, Δ_2 is equivalent to one of Δ_0, Δ_1, Δ_2 (the one of the same rank as Δ).*

In the following, Δ_0, Δ_1, Δ_2 are patterns constructed for the automaton A by Lemma 7. An additional 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].

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 [5, Fact 3.2].

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 unique common node of the two elements Δ_1 (the *junction node*), 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. The existence of such an inner loop is denoted by $p \to_\varepsilon q$. The computation in this definition is allowed to be empty, and hence $p \to_\varepsilon p$ for all p.

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

▶ **Lemma 8** (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 $p \to_\varepsilon q$.*

Every inner loop can be executed at the junction node between any two elements Δ_0, Δ_1, Δ_2 or Δ_a, because attaching Δ_1 to any port always produces an equivalent element by Lemma 7. Then, runs between ports of these elements can be regarded as runs between junction nodes, each located at the junction of two elements Δ_1. Hence, one can consider *runs extended with loops*, which begin with an inner loop at the port of departure, then make a run to the destination port, and finally execute another inner loop there. For brevity, such runs shall be called *transfers*.

▶ **Definition 9** (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 $p \to_\varepsilon p'$, continues with a run of type $(p', i, q', j) \in \delta_\Delta$, and ends with another inner loop $q' \to_\varepsilon q$, 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 10.** *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 11.** *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.*

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]).

$p \circlearrowleft q$	if $(p, 0, q, 0) \in \gamma_{\Delta_0}$	$p \circlearrowleft_a q$	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 $(p, 0, q, 2) \in \gamma_{\Delta_2}$
$p \curvearrowleft q$	if $(p, 2, q, 1) \in \gamma_{\Delta_2}$	$p \curvearrowright q$	if $(p, 1, q, 2) \in \gamma_{\Delta_2}$

Note that the notation $p \diamond q$, with $\diamond \in \{\circlearrowleft, \dots, \curvearrowright\}$, always refers to a transfer from the state p to the state q, regardless of the direction of the arrow. For example, $p \curvearrowleft q$ denotes that there is a transfer from p at port 2 to q at port 1 in the element Δ_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 12.** *Let A be an NTWA with a set of states Q. Let Δ be a pattern of rank k, with $k \geqslant 2$, 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 $1 \leqslant i \leqslant k-1$, or $p \to_\varepsilon q$.*

The proof is omitted due to space constraints.

The following lemma 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.

▶ **Lemma 13** (Bojańczyk and Colcombet [5, Prop. 5.6]). *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 $p \searrow q$.*

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, let $n = |Q|$, and assume, without loss of generality, that n is even. 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 3. The pattern Δ_n, called *the small correct pattern*, is a chain of n elements $\Delta_2[*, \Delta_a]$ of rank 1. The other pattern Δ'_{2M}, *the faulty pattern*, is constructed by attaching to the element $\Delta_2[*, \Delta_2[\Delta_a, \Delta_a]]$ two chains of $M-1$ elements $\Delta_2[*, \Delta_a]$ 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 $\mathit{root}[\Delta_1[\Delta_n[\Delta_0]]]$ 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 $\mathit{root}[\Delta_1[\Delta_n[\Delta_0]]]$, then it also accepts the tree $\mathit{root}[\Delta_1[\Delta'_{2M}[\Delta_0]]]$. This will show that the automaton A does not recognize the complement of L, which is enough to prove Theorem 4.

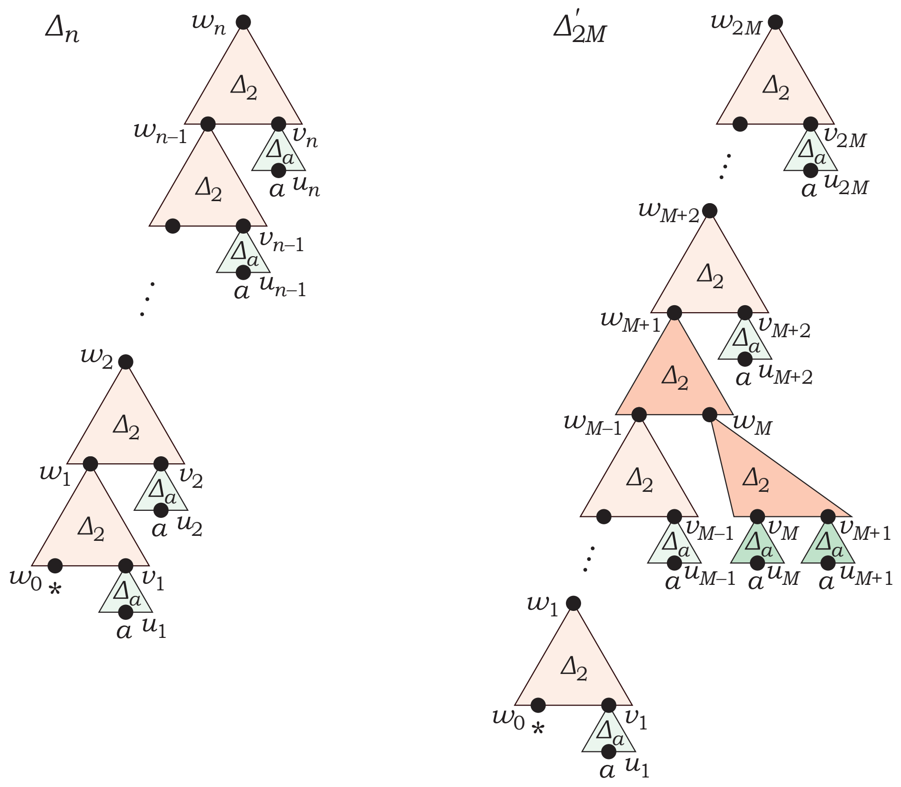

Figure 3 Patterns Δ_n and Δ'_{2M}.

Consider any accepting computation of the automaton A on the tree $\mathit{root}[\Delta_1[\Delta_n[\Delta_0]]]$. 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 $\mathit{root}[\Delta_1[\Delta'_{2M}[\Delta_0]]]$. 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}.

▶ **Main Lemma.** *Let A be an NTWA with the set of states Q of even size $n \geqslant 4$, operating on binary trees with labels $\{a, b\}$, let Δ_0, Δ_1, Δ_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 $\delta_{\Delta_n} \subseteq \delta_{\Delta'_{2M}}$.*

Any run through Δ_n that begins and ends in the same port is exactly reproduced in the faulty pattern Δ'_{2M}, since Δ'_{2M} begins and ends with two subpatterns Δ_n. Since runs going from the root port of Δ_n to its leaf port are symmetric to runs from the leaf port to the root port, it is sufficient to prove the latter case. Accordingly, 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 $(q_{\text{start}}, 1, q_{\text{finish}}, 0)$

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.

First, consider a run $(q_{\text{start}}, 1, q_{\text{finish}}, 0)$ through the small correct pattern Δ_n. Since Δ_n has two ports, w_0 and w_n, and $n-1$ junction nodes, $w_1, \dots, w_{n-1}$, some two of these nodes are visited by the automaton for the first time in the same state. Then, in the original computation, the part between the first visits to these nodes can be repeated periodically to form a computation on Δ_{2M}. This is possible, because $M \equiv \frac{n}{2} \pmod{n!}$. The goal is to take this computation on the correct pattern Δ_{2M} and to reproduce it on the faulty pattern Δ'_{2M}, so that the periodically repeated sequence will pass by the fault, 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; assume that at least one Δ_a is 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 14.** *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 4(left). 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.*

A proper step of type (i, p, q) can be used anywhere in the pattern Δ_{2M}: that is, for all integers x satisfying $1 \leqslant x \leqslant 2M$ and $1 \leqslant x+i \leqslant 2M$, 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, the run through Δ_{2M} constructed above is modified so that its periodic part consists of proper steps.

▷ Claim 15. There is a run of type $(q_{\text{start}}, 1, q_{\text{finish}}, 0)$ on the pattern Δ_{2M} that first comes to some node $v_{i'}$, with $i' \leqslant n$, in some state $\widehat{q}$, 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 $\widehat{q}$, and after that the automaton finishes the run visiting at most n top elements $\Delta_2[*, \Delta_a]$.

Sketch of a proof. Consider the run through Δ_{2M} constructed above, which contains a periodic part. First, the periodically repeated sequence is shifted in order to begin with a visit to the root port of some Δ_a (and accordingly to end with such a visit). This new sequence can be split into segments by the visits to the root ports of any Δ_a. These segments, however, are not necessarily proper steps in the sense of Definition 14; but, using Lemma 11, each of these segments can be replaced with a proper step. Since each iteration of the period was initially made on n consecutive elements $\Delta_2[*, \Delta_a]$, after the shift it fits into $2n$ consecutive elements, and hence each step is of pace between $-(2n-1)$ and $2n-1$. ◁

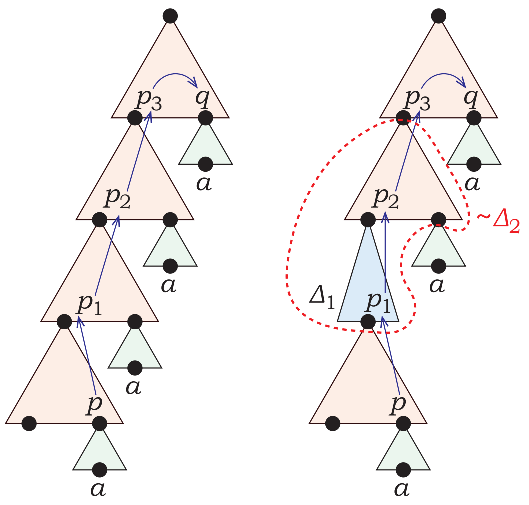

Figure 4 (left) proper step of type (i, p, q), for $i = 3$; (middle) the second element $\Delta_2[*, \Delta_a]$ from the bottom replaced with Δ_1; (right) the resulting proper step of type $(i - 1, p, q)$.

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 16. 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 $1 \leqslant x \leqslant 2M$ and $1 \leqslant x + i \leqslant 2M$, 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. The proof is given only for the case of $i > 0$ (the case of $i < 0$ is proved symmetrically). The cases $i > 1$ and $i = 1$ are considered separately.

- In the case $i > 1$ it will be proved that a proper step of type $(i - 1, p, q)$ exists. Consider the partition of any proper step of type (i, p, q) into transfers through elements Δ_2. Let $p_1, \ldots, p_i$ be the intermediate states in this partition, as in Figure 4(left). Then $p \nwarrow p_1 \nearrow p_2 \nearrow \ldots \nearrow p_i \curvearrowright q$. By Lemma 13, $p_1 \nearrow p_2$ implies $p_1 \uparrow p_2$, that is, if the element $\Delta_2[*, \Delta_a]$ is replaced with the element Δ_1, as in Figure 4(middle), then the automaton can make a computation of the form $p \nwarrow p_1 \uparrow p_2 \nearrow \ldots \nearrow p_i \curvearrowright q$, without noticing the difference between the two patterns. Attaching an element Δ_1 to any port of an element Δ_2 results in an element equivalent to Δ_2 by Lemma 7. Then $p \nwarrow p_1 \nearrow p_3 \ldots \nearrow p_i \curvearrowright q$, and this is a proper step of type $(i-1, p, q)$, see Figure 4(right).

- In the case of $i = 1$, the goal of the proof is to show, for each number x, that either this x satisfies Condition I in the claim, or Condition II holds in general. Let x be any integer with $1 \leqslant x \leqslant 2M - 1$. It will be proved that either the automaton can move on the faulty pattern Δ'_{2M} as stated, or there exists a proper step of type $(0, p, q)$.
  A proper step of type $(1, p, q)$ splits into two transfers, that is, $p \nwarrow p_1 \curvearrowright q$ for some state p_1. Then, Lemma 13 asserts that $p \uparrow p_1 \curvearrowright q$, and since attaching Δ_1 produces an equivalent element, $p \curvearrowright q$. A new pattern Δ of rank $2M + 1$ is obtained out of the faulty pattern Δ'_{2M} by removing all elements Δ_a. Then the nodes v_x and v_{x+1} are consecutive ports of the pattern Δ. Since $p \curvearrowright q$, by Lemma 12, either $(p, j, q, j+1) \in \gamma_\Delta$ for all $j \in \{1, \ldots, 2M\}$, or $p \to_\varepsilon q$. In the former case, the automaton can move from configuration (p, v_x) to configuration (q, v_{x+1}) on the faulty pattern Δ'_{2M}, and the number x satisfies Condition I. And in the latter case there is a proper step of type $(0, p, q)$, and Condition II holds. ◀

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 17. 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 $x \geqslant 1$ be an integer bounded as $x + i \leqslant 2M$. 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 $1 \leqslant y < M$ and $M + 1 < z \leqslant 2M$, the automaton can move on Δ'_{2M} from configuration (p, v_y) to configuration (q, v_z).

The proof is by a longer case analysis; due to space constraints, only one case is presented.

Sketch of a proof for the case $x < M$, $x + i = M$. Let $p \nwarrow p_1 \nearrow p_2 \nearrow \ldots \nearrow p_i \curvearrowright q$, for some intermediate states $p_1, \ldots, p_i \in Q$ in the partition of some proper step of type (i, p, q) into transfers through elements Δ_2. Then $p_i \curvearrowright q$ implies $p_i \curvearrowright t \downarrow q$, for some intermediate state t. Next, by Lemma 13, there is at least one of the transfers $t \swarrow q$ and $t \searrow q$. If $t \swarrow q$, then the sequence of transfers $p \nwarrow p_1 \nearrow p_2 \nearrow \ldots \nearrow p_i \curvearrowright t \swarrow q$ on the faulty pattern Δ'_{2M} leads from configuration (p, v_x) to configuration (q, v_M), and Condition I holds. And if $t \searrow q$, then, similarly, $p \nwarrow p_1 \nearrow p_2 \nearrow \ldots \nearrow p_i \curvearrowright t \searrow q$, and the automaton moves on Δ'_{2M} from (p, v_x) to (q, v_{M+1}); this is Condition II. ◁

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 4 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 15, 16 and 17.

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 15; it 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 defined in Claim 15: it can move up to configuration $(\widehat{q}, v_{i'})$, and it can finish the computation from configuration $(\widehat{q}, v_{i''})$. It remains to prove that the automaton A can also move from configuration $(\widehat{q}, v_{i'})$ to configuration $(\widehat{q}, v_{i''})$ on the faulty pattern Δ'_{2M}.

If any proper step satisfying Condition III from Claim 17 is used in the periodically repeated part of the computation, then this proper step can be stretched to almost the entire faulty pattern Δ'_{2M}, and it can be used to skip the fault, so that the automaton A moves on the faulty pattern Δ'_{2M} from configuration $(\widehat{q}, v_{i'})$ to configuration $(\widehat{q}, v_{i''})$. 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 $\widehat{q}$. 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 $\widehat{q}$; what is important is that it comes in this state to *some* node far beyond the fault.

$\triangleright$ Claim 18. The automaton A operating on the faulty pattern Δ'_{2M} may move from configuration $(\widehat{q}, v_{i'})$ to some configuration $(\widehat{q}, v_j)$, where j is a number with $M + 2n + 1 < j < M + 8n$.

Sketch of a proof. The idea is to take each proper step used in the periodic part of the computation on the correct pattern, and to reproduce it in the computation on the faulty pattern. When this is impossible, that proper step will be modified: a proper step downward will be shrunk using Claim 16, and an upward proper step will be stretched by Claim 17. This way, the sequence of modified proper steps on the faulty pattern Δ'_{2M} will lead the automaton upward faster than the original sequence of proper steps on the correct pattern. Finally, once the modified sequence passes by the fault, the desired configuration $(\widehat{q}, v_j)$ can be obtained by just finishing the current iteration. $\triangleleft$

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 $\widehat{q}$ 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 $\widehat{q}$, 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 $\widehat{q}$ on Δ_{2M}.

Proof of the Main Lemma. If it is possible to pass by the fault with zero shift, then this is all, the Main Lemma is proved. Assume that this is impossible. Then, some proper step of some type (i, p, q) from the periodic part of the computation on the correct pattern cannot be reproduced on the faulty pattern. In this case, Condition I in Claim 16 does not hold for this step. Then the claim asserts that there is a proper step of type $(\widetilde{i}, p, q)$, where $|\widetilde{i} - i| = 1$ and $|\widetilde{i}| = |i| - 1$. Let a single iteration of the periodically repeated sequence move the automaton up by s elements $\Delta_2[*, \Delta_a]$; and from Claim 15 it is known that $1 \leqslant s \leqslant n$. Furthermore, $s \geqslant 2$, because otherwise all nodes from $v_{i'}$ to $v_{i''}$ would be visited in the state $\widehat{q}$, and the automaton would pass by the fault without a shift. Then, if one uses the shrunken proper step of type $(\widetilde{i}, p, q)$ in the iteration instead of the original proper step of type (i, p, q), then the resulting iteration moves the automaton upwards either by $s - 1$ or by $s + 1$ elements $\Delta_2[*, \Delta_a]$.

The idea is to use this iteration with a shrunken proper step several times to compensate for the shift. By Claim 18, the automaton can move on the faulty pattern from $(\widehat{q}, v_{i'})$ to $(\widehat{q}, v_j)$, with $M + 2n + 1 < j < M + 8n$. In the computation on the correct pattern the automaton comes to nodes with numbers i', $i' + s$, $i' + 2s$, …, i'' in the state $\widehat{q}$. And to get back on the track of this computation, the automaton should compensate for the wrong residue of the number j modulo s. This can be done by applying at most $s - 1$ iterations with the shrunken proper step. It remains to prove that the automaton cannot reach the fault and cannot move upward too far while compensating for the wrong residue modulo s.

Since the original iteration has pace at least 2 and involves at most $2n$ consecutive elements $\Delta_2[*, \Delta_a]$, the iteration with the shrunken proper step has positive pace and involves at most $2n + 1$ consecutive elements $\Delta_2[*, \Delta_a]$. Thus, the automaton cannot reach the fault by applying such iterations starting in the node v_j, because $j > M + 2n + 1$.

Now consider how far up the automaton can move while correcting the shift. It applies at most $s - 1$ iterations with the shrunken proper step; such iterations have pace at most $s + 1$. Therefore, applying these iterations leads the automaton to a node with number at most $j + (s-1)(s+1)$, and $j + (s-1)(s+1) < j + s^2 < M + 8n + n^2$, which is less than i'', because $n^2 + 10n < M$ and $2M - n < i''$. And since each iteration involves at most $2n+1$ consecutive elements $\Delta_2[*, \Delta_a]$, during these iterations the automaton can visit nodes with numbers at most $j + (s-1)(s+1) + 2n$, which satisfies $j + (s-1)(s+1) + 2n < M + 10n + n^2 < 2M$.

Therefore, the automaton A, operating on the faulty pattern Δ'_{2M}, can return to the track of the computation on the correct pattern, and come to the root in the state q_{finish}. This proves the Main Lemma, as well as the entire Theorem 4. ◀

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 4, it is recognized by an NTWA A_L.

▶ **Theorem 19.** *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. Let B be any NTWA recognizing the language L. It should be proved that the automaton B is not unambiguous. Let $n \geqslant 4$ be the number of its states. Let Δ_n and Δ'_{2M}, both of rank 1, be the patterns defined for the automaton B as in Section 5. By the Main Lemma, $\delta_{\Delta_n} \subseteq \delta_{\Delta'_{2M}}$.

Consider the following four trees.

$$\begin{aligned}
T_{0,0} &= root[\Delta_1[\Delta_n[\Delta_n[\Delta_0]]]],\\
T_{0,1} &= root[\Delta_1[\Delta_n[\Delta'_{2M}[\Delta_0]]]],\\
T_{1,0} &= root[\Delta_1[\Delta'_{2M}[\Delta_n[\Delta_0]]]],\\
T_{1,1} &= root[\Delta_1[\Delta'_{2M}[\Delta'_{2M}[\Delta_0]]]].
\end{aligned}$$

Of these, the three trees $T_{0,1}, T_{1,0}, T_{1,1}$ are in the language L, and hence accepted by the automaton B. The tree $T_{0,0}$ is not in L, and must be rejected by B.

Let $C_{0,1}$ be an accepting computation on the tree $T_{0,1}$. By replacing the small correct pattern Δ_n in the tree $T_{0,1}$ with Δ'_{2M}, and by using the inclusion $\delta_{\Delta_n} \subseteq \delta_{\Delta'_{2M}}$, the computation $C_{0,1}$ is transformed into an accepting computation $C'_{0,1}$ on the tree $T_{1,1}$, such that for every run through the upper pattern Δ'_{2M} made in $C'_{0,1}$ there is a run of the same type through the small correct pattern Δ_n (as it was made in $C_{0,1}$).

Similarly, an accepting computation $C_{1,0}$ on the tree $T_{1,0}$ is transformed into an accepting computation $C'_{1,0}$ on the tree $T_{1,1}$. All runs through the lower pattern Δ'_{2M} made in $C'_{1,0}$ can be reproduced on Δ_n.

Suppose that the accepting computations $C'_{0,1}$ and $C'_{1,0}$ on the tree $T_{1,1}$ are the same. Then the computation $C'_{0,1}$, while passing through each pattern Δ'_{2M}, makes only such runs that can be executed on Δ_n. Replacing both patterns Δ'_{2M} with Δ_n, one obtains an accepting computation of the automaton B on the tree $T_{0,0}$, which is not in L. This is a contradiction, and therefore the computations $C'_{0,1}$ and $C'_{1,0}$ are distinct, and thus the automaton B is not unambiguous. ◀

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

Proof. The automaton moves its pebble in the order of depth-first tree traversal. When it puts the pebble at some node v, it proceeds with checking the following two conditions: first, that the left subtree of v contains at least one a-labelled leaf, and secondly, that the right subtree of v contains at least two a-labelled leaves. With the pebble in place, both searches can be done deterministically. If both conditions hold for some node v, then the automaton reports that the tree is in L. Otherwise, it moves the pebble to the next position and begins the next check.

If no node in the tree has both conditions satisfied at the same time, the automaton eventually completes moving its pebble around and reports that the tree is not in L. ◀

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

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 determinized whereas UGWA cannot.

References

1 Alfred V. Aho and Jeffrey D. Ullman. Translations on a context-free grammar. *Inf. Control.*, 19(5):439–475, 1971. doi:10.1016/S0019-9958(71)90706-6.

2 Mikołaj Bojańczyk. Tree-walking automata. In Carlos Martín-Vide, Friedrich Otto, and Henning Fernau, editors, *Language and Automata Theory and Applications, Second International Conference, LATA 2008, Tarragona, Spain, March 13-19, 2008. Revised Papers*, volume 5196 of *Lecture Notes in Computer Science*, pages 1–2. Springer, 2008. doi:10.1007/978-3-540-88282-4_1.

3 Mikołaj Bojańczyk. It is undecidable if two regular tree languages can be separated by a deterministic tree-walking automaton. *Fundam. Informaticae*, 154(1-4):37–46, 2017. doi:10.3233/FI-2017-1551.

4 Mikołaj Bojańczyk and Thomas Colcombet. Tree-walking automata cannot be determinized. *Theor. Comput. Sci.*, 350(2-3):164–173, 2006. doi:10.1016/j.tcs.2005.10.031.

5 Mikołaj Bojańczyk and Thomas Colcombet. Tree-walking automata do not recognize all regular languages. *SIAM J. Comput.*, 38(2):658–701, 2008. doi:10.1137/050645427.

6 Mikołaj Bojańczyk, Mathias Samuelides, Thomas Schwentick, and Luc Segoufin. Expressive power of pebble automata. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *Automata, Languages and Programming, 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part I*, volume 4051 of *Lecture Notes in Computer Science*, pages 157–168. Springer, 2006. doi:10.1007/11786986_15.

7 Thomas Colcombet. Unambiguity in automata theory. In Jeffrey O. Shallit and Alexander Okhotin, editors, *Descriptional Complexity of Formal Systems - 17th International Workshop, DCFS 2015, Waterloo, ON, Canada, June 25-27, 2015. Proceedings*, volume 9118 of *Lecture Notes in Computer Science*, pages 3–18. Springer, 2015. doi:10.1007/978-3-319-19225-3_1.

8 Joost Engelfriet and Hendrik Jan Hoogeboom. Tree-walking pebble automata. In Juhani Karhumäki, Hermann A. Maurer, Gheorghe Păun, and Grzegorz Rozenberg, editors, *Jewels are Forever, Contributions on Theoretical Computer Science in Honor of Arto Salomaa*, pages 72–83. Springer, 1999. doi:10.1007/978-3-642-60207-8_7.

9 Joost Engelfriet and Hendrik Jan Hoogeboom. Automata with nested pebbles capture first-order logic with transitive closure. *Log. Methods Comput. Sci.*, 3(2), 2007. doi:10.2168/LMCS-3(2:3)2007.

10 Mika Göös, Stefan Kiefer, and Weiqiang Yuan. Lower bounds for unambiguous automata via communication complexity. In Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, volume 229 of *LIPIcs*, pages 126:1–126:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.ICALP.2022.126.

11 Emil Indzhev and Stefan Kiefer. On complementing unambiguous automata and graphs with many cliques and cocliques. *Inf. Process. Lett.*, 177:106270, 2022. doi:10.1016/J.IPL.2022.106270.

12 Michal Kunc and Alexander Okhotin. Reversibility of computations in graph-walking automata. *Inf. Comput.*, 275:104631, 2020. doi:10.1016/j.ic.2020.104631.

13 Olga Martynova. Complexity of the emptiness problem for graph-walking automata and for tilings with star subgraphs. *Inf. Comput.*, 296:105127, 2024. doi:10.1016/J.IC.2023.105127.

14 Anca Muscholl, Mathias Samuelides, and Luc Segoufin. Complementing deterministic tree-walking automata. *Inf. Process. Lett.*, 99(1):33–39, 2006. doi:10.1016/j.ipl.2005.09.017.

15 Frank Neven. Automata, logic, and XML. In Julian C. Bradfield, editor, *Computer Science Logic, 16th International Workshop, CSL 2002, 11th Annual Conference of the EACSL, Edinburgh, Scotland, UK, September 22-25, 2002, Proceedings*, volume 2471 of *Lecture Notes in Computer Science*, pages 2–26. Springer, 2002. doi:10.1007/3-540-45793-3_2.

16 Frank Neven and Thomas Schwentick. On the power of tree-walking automata. *Inf. Comput.*, 183(1):86–103, 2003. doi:10.1016/S0890-5401(03)00013-0.

17 Mathias Samuelides and Luc Segoufin. Complexity of pebble tree-walking automata. In Erzsébet Csuhaj-Varjú and Zoltán Ésik, editors, *Fundamentals of Computation Theory, 16th International Symposium, FCT 2007, Budapest, Hungary, August 27-30, 2007, Proceedings*, volume 4639 of *Lecture Notes in Computer Science*, pages 458–469. Springer, 2007. doi:10.1007/978-3-540-74240-1_40.

18 Michael Sipser. Halting space-bounded computations. *Theor. Comput. Sci.*, 10:335–338, 1980. doi:10.1016/0304-3975(80)90053-5.

19 Margus Veanes. On computational complexity of basic decision problems of finite tree automata. UPMAIL Technical Report 133, Computing Science Department, Uppsala University, January 1997.