Санкт-Петербургский государственный университет

**МАРТЫНОВА Ольга Максимовна**

**Выпускная квалификационная работа**

# Minimal strings and graphs accepted by automata

Уровень образования: магистратура
Направление: 01.04.01 «Математика»
Основная образовательная программа: ВМ.5832.2021 «Современная математика»

Научный руководитель:

Профессор
Факультет математики и компьютерных наук
Санкт-Петербургский государственный университет
Ph.D.
Охотин Александр Сергеевич

Рецензент:

Профессор
Факультет компьютерных наук
Национальный исследовательский университет
«Высшая школа экономики»
к.ф.-м.н.
Вялый Михаил Николаевич

Санкт-Петербург
2023

**Аннотация**

Для разных известных видов автоматов изучается вопрос, насколько большим может быть минимальный принимаемый объект. Максимальная длина кратчайшей строки, принимаемой двухсторонним конечным автоматом, который запоминает направление последнего шага, определяется точно. Для автоматов общего вида доказывается более высокая нижняя оценка. Максимальное число вершин в минимальном принимаемом дереве для недетерминированных древесных автоматов определяется точно. Для древоходных автоматов показывается, что максимальный размер минимального принимаемого дерева — двойной экспоненциальный от числа состояний. Кроме того, доказывается разрешимость задачи пустоты для двух видов автоматов на графах. Задача непустоты для графоходных автоматов, которые ходят по рёбрам графа, оказывается NEXP-полной, а для замощений графов подграфами-звёздами, оказывается NP-полной.

**Abstract**

For different classical types of automata, the question, how large the minimal accepted object can be, is investigated. The maximum length of the shortest string accepted by a two-way finite automaton that remembers the direction of the last move is determined precisely. For two-way automata of the general form, a higher lower bound is proved. The maximum possible number of nodes in the minimal tree accepted by a nondeterministic tree automaton is determined precisely. For tree-walking automata, the maximum size of the minimal accepted tree is proved to be double exponential in the number of states. Finally, this thesis proves the decidability of the emptiness problem for two types of automata on graphs. Non-emptiness problem for graph-walking automata (that move in a graph by following its edges) is proved to be NEXP-complete, and for tilings of graphs by star subgraphs is proved to be NP-complete.

# Contents

**1 Introduction** **3**

**2 Shortest accepted strings for two-way finite automata** **7**
2.1 Two-way finite automata . . . . . 7
2.2 Shortest accepted strings for direction-determinate automata . . . . . 9
2.3 Longer shortest strings for automata of the general form . . . . . 13
2.4 Calculations . . . . . 17

**3 Minimal trees accepted by tree automata and tree-walking automata** **20**
3.1 Tree-walking and tree automata . . . . . 20
3.2 Bounds on the size of the minimal accepted tree for a tree automaton . . . 23
3.3 Transformation of a nondeterministic tree-walking automaton to a nondeterministic tree automaton . . . . . 27
3.4 Upper bound on the size of the smallest tree accepted by a nondeterministic tree-walking automaton . . . . . 33
3.5 Lower bound on the size of the smallest tree accepted by a tree-walking automaton . . . . . 38

**4 Complexity of the emptiness problem for graph-walking automata and for tilings with star subgraphs** **47**
4.1 Graph-walking and star automata . . . . . 48
4.2 The non-emptiness problem for signatures is in NP . . . . . 50
4.3 Reducing a star automaton to a signature . . . . . 56
4.4 Reducing a graph-walking automaton to a signature . . . . . 58
4.5 Computational complexity of emptiness problems . . . . . 63

**5 Conclusion** **73**

# Chapter 1

# Introduction

A natural question about automata and related models of computation is how large can be the minimal object an automaton accepts.

The most well-known automata are finite automata that work on strings. A function mapping the size of such an automaton to the maximum length of the shortest accepted string, with the maximum taken over all automata of that size, is a certain complexity measure for a family of automata.

For one-way finite automata, this measure is trivial: the length of the shortest string accepted by a nondeterministic finite automaton (NFA) with $n$ states is at most $n-1$: this is the length of the shortest path to an accepting state. On the other hand, Ellul et al. [9] proved that the length of shortest strings *not* accepted by an $n$-state NFA is exponential in $n$. Similar questions were studied for other models and some variants of the problem. Chistikov et al. [7] investigated the length of shortest strings in counter automata. The length of shortest strings in formal grammars under intersections with regular languages was studied by Pierre [23], and recently by Shemetova et al. [25]. Alpoge et al. [1] investigated shortest strings in intersections of deterministic one-way finite automata (DFA).

The maximum length of shortest strings for deterministic two-way finite automata (2DFA) has been investigated in two recent papers. First of all, from the well-known proof of the PSPACE-completeness of the emptiness problem for 2DFA by Kozen [16] it is understood that the length of the shortest string accepted by an $n$-state 2DFA can be exponential in $n$. There is also an exponential upper bound on this length, given by transforming a 2DFA to an NFA: the construction by Kapoutsis [15] uses at most $\binom{2n}{n+1} = \Theta(\frac{1}{\sqrt{n}}4^n)$ states, and hence the length of the shortest string is slightly less than $4^n$. Overall, the maximum length of the shortest string is exponential, with the base bounded by 4.

The first attempt to determine the exact base was made by Dobronravov et al. [8], who constructed a family of $n$-state 2DFA with shortest strings of length $\Omega((\sqrt[5]{10})^n) \geqslant \Omega(1.584^n)$. The automata they have actually constructed belong to a special class of 2DFA: the *direction-determinate automata*. These are 2DFA with the set of states split into states accessible only by transitions from the right and states accessible only by transitions from the left: in other words, direction-determinate automata always remember the direction of the last transition in their state.

Later, Krymski and Okhotin [17] extended the method of Dobronravov et al. [8] to produce automata of a more general form, with longer shortest accepted strings. They

constructed a family of non-direction-determinate 2DFA with shortest strings of length $\Omega((\sqrt[4]{7})^n) \geqslant \Omega(1.626^n)$.

These bounds for two-way finite automata are improved in Chapter 2. First, the maximum length of the shortest string accepted by $n$-state direction-determinate 2DFA is determined precisely as $\binom{n}{\lfloor \frac{n}{2} \rfloor} - 1 = \Theta(\frac{1}{\sqrt{n}} 2^n)$. The upper bound on the length of the shortest string immediately follows from the complexity of transforming direction-determinate 2DFA to NFA, see Geffert and Okhotin [12]. A matching lower bound is proved by a direct construction of a family of $n$-state automata.

The second result of Chapter 2 is that not remembering the direction helps to accept longer shortest strings: a family of $n$-state non-direction-determinate automata with shortest strings of length $\frac{3}{4} \cdot 2^n - 1$ is constructed. This is more than what is possible in direction-determinate automata.

Automata that work on strings have well-known generalizations to automata that work on trees with labelled nodes. Deterministic and nondeterministic one-way finite automata (DFA and NFA) become deterministic bottom-up tree automata (DTA) and nondeterministic tree automata (NTA). These are automata that compute states in nodes of a given tree from the leaves to the root. A state in a node is determined (deterministically for DTA and nondeterministically for NTA) by a label of this node and by states already chosen in the children of this node. If an automaton can choose states in all nodes of a tree in this way, then the tree is accepted.

Two-way finite automata are generalized to trees in a different way. They become deterministic and nondeterministic tree-walking automata (DTWA and NTWA). Such an automaton works on a tree by moving along its edges. At every moment, an automaton is in some state and stands at some node. It sees only the label of this node. Depending on its state and on the label of the node, it decides (deterministically or not) to move to one of the neighbouring nodes and changes its state. It can also decide to accept or reject.

It is known that nondeterministic tree automata are equal in power to deterministic bottom-up tree automata, like NFA are equal in power to DFA.

However, there are some differences between automata on strings and on trees. Bojańczyk and Colcombet [5] proved that tree-walking automata are strictly weaker in power than tree automata, and deterministic tree-walking automata are strictly weaker than nondeterminic ones [4].

The size of minimal trees accepted by tree and tree-walking automata has not been studied yet. But it is known, that the emptiness problem for these automata (whether a given automaton accepts at least one tree) is decidable, and complexity classes for the emptiness problem were determined precisely for these models of computation. Veanes [27] proved that the emptiness problem for nondeterministic tree automata is P-complete. It is proved by Bojańczyk [3] that the emptiness problem for both deterministic and nondeterministic tree-walking automata is EXP-complete.

The size of minimal accepted trees for tree and tree-walking automata is investigated in Chapter 3. For nondeterministic tree automata, the maximum possible size of the minimal accepted tree is determined precisely as the last element in the inductively defined sequence of numbers. This element can be estimated from above as $2^{rn}$, where $n$ is the number of states in an automaton, and $r$ is the maximum number of children for a node. If $r \geqslant 2$, then the element can be estimated from below by the $rn$-th Fibonacci number. And the witness automata for the precise lower bound are bottom-up deterministic.

Estimating the maximum size of the minimal trees accepted by tree-walking automata

is more difficult, because tree-walking automata are a generalization of two-way finite automata, and the problem about the shortest accepted strings is much harder for two-way finite automata than for one-way finite automata. The lower and upper bounds obtained for tree-walking automata in Chapter 3 do not coincide, but are of similar order. It is proved that for an $n$-state nondeterministic tree-walking automaton that works on trees with a maximal degree at most $r$, the minimal accepted tree has at most $2^{O(rn \cdot 3.572^n)}$ nodes. The lower bound is proved for deterministic tree-walking automata, and it is $2^{\Omega(r \cdot 1.618^n)}$.

Automata on trees can be generalized further by replacing trees with arbitrary graphs. Then, deterministic tree-walking automata are generalized to graph-walking automata (GWA).

A graph-walking automaton is a model of a robot in a maze. It has finitely many states, and it deterministically walks on graphs with labelled nodes and labelled edge end-points. The automaton decides by which edge to move depending on the label of the current node and on its current state. The automaton can also decide to accept or to reject, and so it defines a graph language: the set of graphs it accepts.

Graph-walking automata were first introduced by Michael Rabin, who stated the conjecture that for each graph-walking automaton, even if it is additionally allowed to use finitely many pebbles, there is a graph that it cannot fully explore. Budach [6] proved this conjecture for graph-walking automata without pebbles. Later Fraigniaud et al. [10] gave an easier proof of this fact. Rollik [24] proved that not only pebbles, but even co-operation of several interacting automata would not help to traverse every graph, thus proving Rabin's conjecture. Kunc and Okhotin [18] showed that every graph-walking automaton can be transformed to an automaton which halts on every input, to an automaton which accepts only at the initial node, and to a reversible automaton, which all accept the same set of graphs. Later Martynova and Okhotin [20] reduced the number of states needed for these transformations, and obtained asymptotically tight lower bounds.

For graph-walking automata, not only the maximum size of the minimal accepted graph has not been studied yet, but it is also not known, whether the emptiness problem for these automata is decidable.

The main result of Chapter 4 is the decidability of the emptiness problem for graph-walking automata and its computational complexity. It is proved that the non-emptiness problem for graph-walking automata is NEXP-complete. Furthermore, Chapter 4 provides an upper bound $m4^{n(k+1)}k^{k4^n-1}$ on the number of nodes in the smallest accepted graph for an $n$-state graph-walking automaton that works on graphs with $k$ labels of edge end-points and with $m$ node labels. This upper bound is nearly of the same order as a lower bound for tree-walking automata.

NFA and nondeterministic tree automata are nondeterministic automata that recognize a given object by *tiling* it with neighbourhoods of states. Tiling models were also considered for graphs. Thomas [26] introduced *graph acceptors*: in this model, a graph is accepted, if it can be covered with tiles (subgraphs) from a fixed finite set, so that each node is in the inner part of some tile, states in overlapping tiles are the same, and some further constraints on the number of occurrences of every tile hold. For this general model, Thomas proved undecidability of the emptiness problem by recognizing the set of rectangular grids and simulating a Turing machine on the grids. Thomas also considered *elementary acceptors*: a special case in which every tile is a star, that is, a node with all its neighbours. For elementary acceptors, Thomas proved that the language of grids

cannot be recognized. However, the decidability of the emptiness problem for elementary acceptors remains open.

Besides the emptiness problem for graph-walking automata, another problem considered in Chapter 4, it is the emptiness problem for *star automata*, that is, for elementary acceptors of Thomas without additional constraints on the number of occurrences of tiles. Star automata are at the same time a special case of the model by Thomas, and a generalization of nondeterministic tree automata to graphs.

It is proved in Chapter 4 that non-emptiness problem for star automata is decidable and NP-complete. Also this chapter gives an upper bound $sn^2k^{kn^2-1}$ on the number of nodes in the smallest accepted graph for an $n$-state star automaton with $s$ different stars, that works on graphs with $k$ labels of edge end-points.

Note that the complexity classes for related problems, such as whether a graph-walking automaton accepts all graphs over its signature (the universality problem), or whether the intersection of languages of two automata is empty, can be inferred from the result for the non-emptiness problem. Indeed, since every graph-walking automaton can be transformed to an automaton that halts on every input, and the transformation given by Kunc and Okhotin [18] can be done in polynomial time, the emptiness problem for graph-walking automata is equivalent to the universality problem. As for the intersection emptiness problem, Martynova and Okhotin [21] obtained a transformation for the intersection of two graph-walking automata, which can be done in polynomial time too. Thus, the universality problem and the intersection emptiness problem for graph-walking automata are both co-NEXP-complete.

The results of Chapter 2 have been accepted to DCFS 2023 conference and are due to appear in conference proceedings [22]. The results of Chapter 4 have been submitted for publication [19].

# Chapter 2

# Shortest accepted strings for two-way finite automata

In this chapter, the length of shortest strings accepted by two-way finite automata is investigated. First, two-way finite automata are formally defined in Section 2.1. Section 2.2 determines the exact maximum length of the shortest string accepted by direction-determinate 2DFA. And then even better lower bound is obtained in Section 2.3 for 2DFA of the general form, proving that it is useful to forget the direction of the last move to have longer shortest accepted strings. In Section 2.4, a few non-direction-determinate 2DFA with a small number of states and long shortest accepted strings are presented: these automata were found by a computer program, and the length of their shortest accepted strings exceeds the theoretical lower bound proved in Section 2.3.

## 2.1 Two-way finite automata

Two-way deterministic finite automata (2DFA) are automata equipped with a finite set of states that move over an input string, scanning one symbol at a time. At every step, an automaton may move to the right or to the left and may change its state. This is a well-known model of computation, but some details of the definition, such as acceptance in the beginning or in the end of a string, may vary; the definition used in this thesis is given below.

**Definition 1.** A *two-way deterministic finite automaton* (2DFA) is a quintuple $A = (\Sigma, Q, q_0, \delta, F)$, in which:

- $\Sigma$ is a finite alphabet, which does not contain two special symbols: the left end-marker ($\vdash$) and the right end-marker ($\dashv$);
- $Q$ is a finite set of states;
- $q_0 \in Q$ is the initial state;
- $\delta\colon Q \times (\Sigma \cup \{\vdash, \dashv\}) \to Q \times \{-1, +1\}$ is a partial transition function;
- $F \subseteq Q$ is the set of accepting states, effective at the right end-marker ($\dashv$).

The transitions on the end-markers must not lead the automaton outside of the string, that is, $\delta(q, \vdash) \notin Q \times \{-1\}$ and $\delta(q, \dashv) \notin Q \times \{+1\}$ for all $q \in Q$. Also, there are no transitions in accepting configurations, that is, $\delta(q, \dashv)$ is undefined for all $q \in F$.

The automaton $A$ works on a string $w = a_1 \dots a_m \in \Sigma^*$ as follows. A string $w$ is written on a tape, delimited by two end-markers: $\vdash a_1 \dots a_m \dashv$. Let $a_0 = \vdash$ and $a_{m+1} = \dashv$.

A *configuration* of the automaton $A$ on a string $w$ is a pair $(q, i)$, where $q \in Q$ is a state, and $i \in \{0, \dots, m+1\}$ is a position of the automaton on the tape (here $i = 0$ corresponds to the left end-marker, and $i = m + 1$ to the right end-marker).

A *computation* of the automaton $A$ on the string $w$ is a sequence of configurations $C_0, \dots, C_N$ constructed by induction. The automaton starts in the initial state at the left end-marker: $C_0 = (q_0, 0)$. Let a configuration $C_j = (q, i)$ be defined already.

- If the transition is defined as $\delta(q, a_i) = (r, d)$, then the automaton changes its state to $r$ and moves in the direction $d$, and so the next configuration is $C_{j+1} = (r, i + d)$.
- If $q \in F$ and $i = m+1$, then the automaton accepts, and the configuration $C_j = C_N$ is the last one.
- If $\delta(q, a_i)$ is undefined and the acceptance condition ($q \in F$, $i = m + 1$) does not hold, then the automaton rejects the string $w$, and $C_j = C_N$ is the last configuration.

A computation can be an infinite sequence of configuration, with $N = \infty$, and then the automaton is said to loop on the string $w$.

The automaton $A$ defines a language $L(A)$: the set of all strings it accepts.

There are also *nondeterministic two-way finite automata* (2NFA). They differ from deterministic ones in having a set of initial states $Q_0$ instead of one initial state $q_0$m and in having a transition function $\delta$ map each pair of a state and a symbol to a set of possible transitions rather than one transition: $\delta \colon Q \times (\Sigma \cup \{\vdash, \dashv\}) \to 2^{Q \times \{-1,+1\}}$. In this case, several computations of an automaton on a string can be possible, and a string is accepted if at least one of them is accepting.

This thesis also uses a subclass of 2DFA, in which one can determine the direction of the previous transition from the current state.

**Definition 2** ([18])**.** A 2DFA is called *direction-determinate*, if there is a partition of the set of states $Q = Q^+ \cup Q^-$, with $Q^+ \cap Q^- = \varnothing$, such that for each transition $\delta(q, a) = (r, +1)$, the state $r$ must belong to $Q^+$, and for each transition $\delta(q, a) = (r, -1)$, the state $r$ is in $Q^-$.

The known upper bounds on the length of the shortest accepted string are different for direction-determinate 2DFA and for 2DFA of the general form. These bounds are inferred from the complexity of transforming two-way automata with $n$ states to one-way NFA: for 2DFA of the general form, as proved by Kapoutsis [15], it is sufficient and in the worst case necessary to use $\binom{2n}{n+1}$ states in a simulating NFA, whereas for direction-determinate 2DFA the simulating 2DFA requires $\binom{n}{\lfloor \frac{n}{2} \rfloor}$ states in the worst case, see Geffert and Okhotin [12]. Since the shortest string in a language cannot be longer than the shortest path to an accepting state in an NFA, the following bounds hold.

**Theorem 1** (Dobronravov et al. [8])**.** *Let $n \geqslant 1$, and let $A$ be a 2DFA with $n$ states, which accepts at least one string. Then the length of the shortest string accepted by $A$ is at most $\binom{2n}{n+1} - 1$. If the automaton $A$ is direction-determinate, then the length of the shortest accepted string does not exceed $\binom{n}{\lfloor \frac{n}{2} \rfloor} - 1$.*

The first result of this chapter is that this upper bound for direction-determinate automata is actually precise.

## 2.2 Shortest accepted strings for direction-determinate automata

In this section, direction-determinate automata with the maximum possible length $\binom{n}{\lfloor \frac{n}{2} \rfloor} - 1$ of shortest accepted strings, where $n$ is the number of states, will be constructed.

Automata are constructed for every $k$ and $\ell$, where $k$ is the number of states reachable by transitions to the right and $\ell$ is the number of states reachable in the left direction. The following theorem shall be proved.

**Theorem 2.** *For every $k \geqslant 2$ and $\ell \geqslant 0$ there exists a direction-determinate 2DFA with the set of states $Q = Q^+ \cup Q^-$, where $|Q^+| = k$ and $|Q^-| = \ell$, such that the length of the shortest string it accepts is $\binom{k+\ell}{\ell+1} - 1$.*

The automaton constructed in the theorem works as follows. While working on its shortest string, it processes every pair of consecutive symbols by moving back and forth between them, thus effectively comparing them to each other. Eventually it moves on to the next pair and processes it in the same way. It cannot come back to the previous pair anymore, because it has no transitions for that.

The automaton's motion between two neighbouring symbols begins when it first arrives from the first symbol to the second in some state from $Q^+$. Then it moves back and forth, alternating between states from $Q^+$ at the second symbol and states from $Q^-$ at the first symbol, and finally leaves the second symbol to the right. Among the states visited by the automaton during this back-and-forth motion, the number of states from $Q^+$ is greater by one than the number of states from $Q^-$. Two such sets of states will be denoted by a pair $(P, R)$, where $P \subseteq Q^-$, $R \subseteq Q^+$ and $|R| = |P| + 1$.

**Proposition 1.** *There are $\binom{k+\ell}{\ell+1}$ different pairs $(P, R)$, such that $P \subseteq Q^-$, $R \subseteq Q^+$ and $|R| = |P| + 1$.*

*Proof.* There are as many pairs $(P, R)$ as pairs $(Q^- \setminus P, R)$, where $|R| = |P| + 1$. The number of pairs of the latter form is equal to the number of subsets of $Q$ of size $\ell + 1$, that is, $\binom{k+\ell}{\ell+1}$. □

Let the sets $Q^+$ and $Q^-$ be linearly ordered. Then one can define an order on the set of pairs $(P, R)$ as follows. In every such pair, let $P = \{p_1, \ldots, p_m\}$, where $p_1 < \ldots < p_m$, and $R = \{r_1, \ldots, r_{m+1}\}$, where $r_1 < \ldots < r_{m+1}$. There is a corresponding sequence to each pair, of the form $r_1$, $-p_1$, $r_2$, $-p_2$, …, $r_m$, $-p_m$, $r_{m+1}$, and different pairs are compared by the lexicographic order on these sequences. In Table 2.1, all pairs $(P, R)$, for $k = 4$ and $\ell = 2$, are given in increasing order, along with the corresponding sequences.

| pairs $(P, R)$ | sequences |
|---|---|
| $\varnothing, \{1\}$ | $(1)$ |
| $\{2'\}, \{1, 2\}$ | $(1, -2', 2)$ |
| $\{2'\}, \{1, 3\}$ | $(1, -2', 3)$ |
| $\{2'\}, \{1, 4\}$ | $(1, -2', 4)$ |
| $\{1'\}, \{1, 2\}$ | $(1, -1', 2)$ |
| $\{1', 2'\}, \{1, 2, 3\}$ | $(1, -1', 2, -2', 3)$ |
| $\{1', 2'\}, \{1, 2, 4\}$ | $(1, -1', 2, -2', 4)$ |
| $\{1'\}, \{1, 3\}$ | $(1, -1', 3)$ |
| $\{1', 2'\}, \{1, 3, 4\}$ | $(1, -1', 3, -2', 4)$ |
| $\{1'\}, \{1, 4\}$ | $(1, -1', 4)$ |
| $\varnothing, \{2\}$ | $(2)$ |
| $\{2'\}, \{2, 3\}$ | $(2, -2', 3)$ |
| $\{2'\}, \{2, 4\}$ | $(2, -2', 4)$ |
| $\{1'\}, \{2, 3\}$ | $(2, -1', 3)$ |
| $\{1', 2'\}, \{2, 3, 4\}$ | $(2, -1', 3, -2', 4)$ |
| $\{1'\}, \{2, 4\}$ | $(2, -1', 4)$ |
| $\varnothing, \{3\}$ | $(3)$ |
| $\{2'\}, \{3, 4\}$ | $(3, -2', 4)$ |
| $\{1'\}, \{3, 4\}$ | $(3, -1', 4)$ |
| $\varnothing, \{4\}$ | $(4)$ |

Table 2.1: All pairs $(P, R)$ for sets of states $Q^+ = \{1, 2, 3, 4\}$ and $Q^- = \{1', 2'\}$.

Let $N = \binom{k+\ell}{\ell+1}$ be the number of pairs. Then all pairs are enumerated in increasing order as $(P^{(1)}, R^{(1)}) < \ldots < (P^{(N)}, R^{(N)})$, where $P^{(i)} = \{p_1^{(i)}, \ldots, p_{m_i}^{(i)}\}$ and $R^{(i)} = \{r_1^{(i)}, \ldots, r_{m_i+1}^{(i)}\}$. In particular, the least pair is $(P^{(1)}, R^{(1)}) = (\varnothing, \{\min Q^+\})$, because the corresponding sequence $(\min Q^+)$ is lexicographically the least. The greatest pair is $(P^{(N)}, R^{(N)}) = (\varnothing, \{\max Q^+\})$.

The desired direction-determinate automaton $A$ with the shortest accepted string of length $N - 1$ is defined over an alphabet $\Sigma = \{a_1, \ldots, a_{N-1}\}$, and the shortest accepted string will be $w = a_1 \ldots a_{N-1}$. The set of states is defined as $Q = Q^+ \cup Q^-$, where $Q^+ = \{1, \ldots, k\}$ and $Q^- = \{1', \ldots, \ell'\}$. The initial state is $q_0 = 1$. The only transition by the left end-marker $(\vdash)$ leads from the initial state to the least state in $R^{(1)}$.

$$\delta(q_0, \vdash) = (r_1^{(1)}, +1) \tag{2.1a}$$

For each symbol $a_i$, transitions are defined in the states $R^{(i)} \cup P^{(i+1)}$. If the automaton is at the symbol $a_i$ in any state from $R^{(i)}$ (except for the greatest state), then it moves to the left in the corresponding state from $P^{(i)}$.

$$\delta(r_j^{(i)}, a_i) = (p_j^{(i)}, -1) \qquad (j \in \{1, \ldots, m_i\}) \tag{2.1b}$$

For the greatest state in $R^{(i)}$, there is no corresponding state in $P^{(i)}$, and so the automaton moves to the right (and this is the only way to move from $Q^+$ to $Q^+$, and hence the only way to advance from the symbol $a_i$ to the next symbol for the first time).

$$\delta(r_{m_i+1}^{(i)}, a_i) = (r_1^{(i+1)}, +1) \tag{2.1c}$$

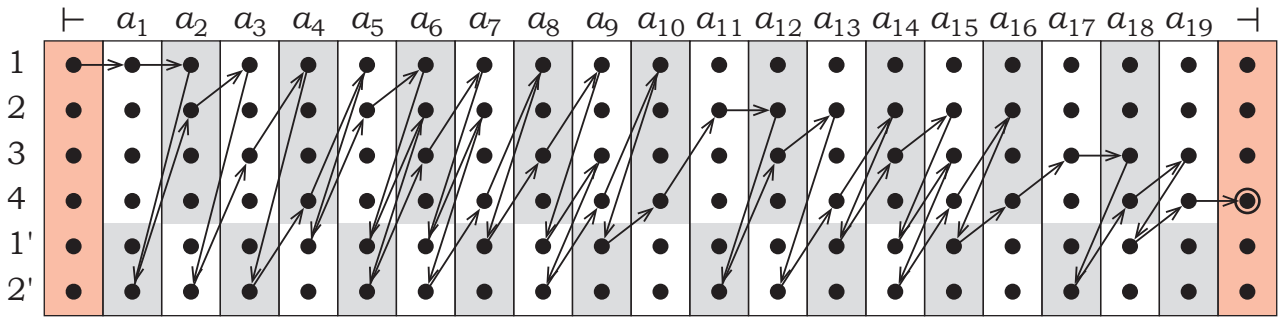


Figure 2.1: The accepting computation of the automaton $A$ on the string $w$, for $k = 4$ and $\ell = 2$.

In each state from $P^{(i)}$, the automaton moves to the right in the next available state from $R^{(i)}$.

$$\delta(p_j^{(i+1)}, a_i) = (r_{j+1}^{(i+1)}, +1) \qquad (j \in \{1, \ldots, m_{i+1}\}) \tag{2.1d}$$

There are no transitions at the right end-marker, and there is one accepting state: $F = \{r_{m_N+1}^{(N)}\}$.

The computation of the automaton on the string $w = a_1 \ldots a_{N-1}$ is illustrated in Figure 2.1. The automaton gradually advances, and moves between every two subsequent symbols, $a_{i-1}$ and $a_i$, according to the sets $P_i$ and $R_i$. Transitions at $a_i$ expect that there is $a_{i-1}$ to the left, whereas transitions at $a_{i-1}$ expect $a_i$ to the right. As long as every symbol is followed by the next symbol in order, these expectations will be fulfilled each time, and the automaton accepts in the end.

**Lemma 1.** *The automaton $A$ accepts the string $w = a_1 \ldots a_{N-1}$.*

*Proof.* It is claimed that the automaton $A$, executed on the string $w$, eventually arrives to each symbol $a_i$ in the state $r_{m_i+1}^{(i)}$. This is proved by induction on $i$.

Base case $i = 1$: the first transition (2.1a) moves the automaton to the state $r_1^{(1)}$. The first pair $(P^{(1)}, R^{(1)})$ is $(\varnothing, \{1\})$, and so $r_1^{(1)} = r_{m_1+1}^{(1)}$.

Induction step. Assume that the automaton comes to the symbol $a_i$ in the state $r_{m_i+1}^{(i)}$. Then it makes a transition (2.1c) to the right in the state $r_1^{(i+1)}$. Then it executes the sequence of transitions (2.1b), (2.1d), defined by the pair $(P_{i+1}, R_{i+1})$, moving back and forth between $a_{i+1}$ and $a_i$, and passing through the states $p_1^{(i+1)}$, $r_2^{(i+1)}$, $p_2^{(i+1)}$, $\ldots r_{m_{i+1}}^{(i+1)}$, $p_{m_{i+1}}^{(i+1)}$, $r_{m_{i+1}+1}^{(i+1)}$. And so it comes to the symbol $a_{i+1}$ in the state $r_{m_{i+1}+1}^{(i+1)}$, as shown in Figure 2.2.

In the end, the automaton comes to the last symbol $a_{N-1}$ in the state $r_{m_{N-1}+1}^{(N-1)}$. Then it makes a transition (2.1c) and moves to the right end-marker in the state $r_1^{(N)}$. And this is the accepting state $r_{m_N+1}^{(N)}$, because the last pair $(P^{(N)}, R^{(N)})$ is $(\varnothing, \{k\})$. Therefore, the string $w$ is accepted. □

It is claimed that the automaton $A$ cannot accept any shorter string. It cannot accept the empty string; if it did, then the first transition would lead to the right end-marker in the state 1, and the automaton would reject, because $k \neq 1$. Next, it will be shown that each accepted string begins with the symbol $a_1$ and ends with the symbol $a_{N-1}$. Finally, it

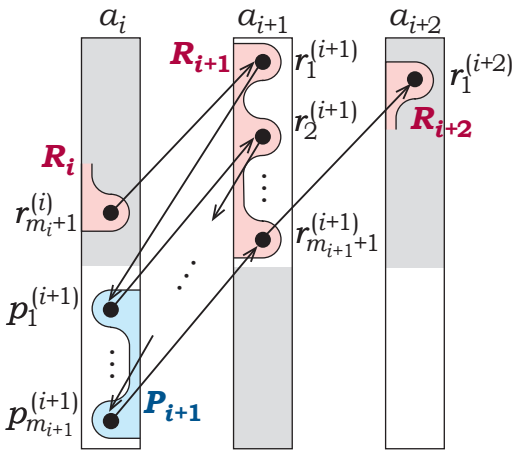


Figure 2.2: The moves of $A$ between two neighbouring symbols of $w$.

will be proved that the automaton cannot skip any number, that is, the number of every next symbol, as compared to the number of the previous symbol, cannot increase by more than 1. If the number decreases or does not change, this would make the string only longer; but in order to reach $a_{N-1}$ from $a_1$ without skipping any number, the automaton would have to move through all symbols of the alphabet, and therefore an accepted string cannot be shorter than $N-1$ symbols.

**Lemma 2.** *Every string accepted by the automaton $A$ begins with the symbol $a_1$.*

*Proof.* Let the automaton $A$ accept some string that starts from some symbol $a_i$. The transition from the initial configuration leads the automaton to the state $r_1^{(1)}$ at the first symbol $a_i$. As $(P^{(1)}, R^{(1)}) = (\varnothing, \{1\})$, the state $r_1^{(1)}$ is 1.

Transitions by the symbol $a_i$ are defined only in states from $R^{(i)} \cup P^{(i+1)}$, and hence $1 \in R^{(i)}$, for otherwise the automaton immediately rejects. If there is at least one more state in $R^{(i)}$, then the transition in the state 1 by $a_i$ moves the automaton to the left. Then the automaton returns to the left end-marker, and then either loops or rejects, because there is only one transition defined there. Therefore, there are no other states in $R^{(i)}$ besides 1, and so, $(P^{(i)}, R^{(i)}) = (\varnothing, \{1\}) = (P^{(1)}, R^{(1)})$, which implies $i = 1$. □

**Lemma 3.** *Every string accepted by the automaton $A$ ends with the symbol $a_{N-1}$.*

*Proof.* Let a string accepted by $A$ end with a symbol $a_i$. To accept, the automaton should move from $a_i$ to the right using the transition (2.1c), and it arrives to the right end-marker in the state $r_1^{(i+1)}$. As the only accepting state is $k$, and the automaton rejects at the right end-marker in all other states, this state must be $r_1^{(i+1)} = k$. Because the state $r_1^{(i+1)}$ is the least in $R^{(i+1)}$, it follows that $R^{(i+1)} = \{k\}$ and $P^{(i+1)} = \varnothing$. Therefore, this is the last pair, and $i = N - 1$. □

**Lemma 4.** *No string accepted by the automaton $A$ may contain any substring of the form $a_i a_j$, where $j > i + 1$.*

*Proof.* The proof is by contradiction. Suppose that $A$ accepts a string that contains a substring $a_i a_j$, with $j > i + 1$. In order to accept, the automaton should eventually reach this symbol $a_j$ for the first time, moving to it from the symbol $a_i$. To make this transition,

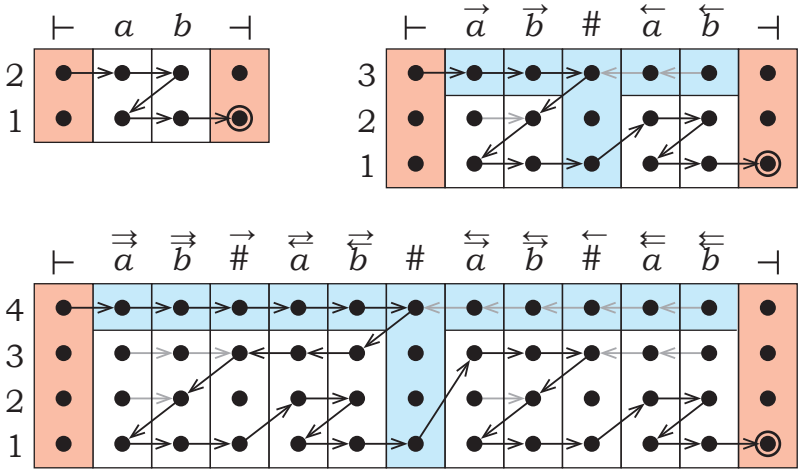


Figure 2.3: Computations of automata $A_2$, $A_3$ and $A_4$ from the proof of Theorem 3 on their shortest strings $w_2$, $w_3$ and $w_4$.

the automaton should be at $a_i$ in some state from $Q^+$ (indeed, if it were in the state from $Q^-$, then it would have been at $a_j$ already at the previous step). Then the automaton must use the transition (2.1c) to move from $a_i$ to $a_j$, and this transition leads to the state $r_1^{(i+1)}$. For the computation to go onward, this state should lie in $R^{(j)}$. Moreover, the state $r_1^{(i+1)}$ should be the least in $R^{(j)}$, for otherwise the pair $(P^{(j)}, R^{(j)})$ would be less than the pair $(P^{(i+1)}, R^{(i+1)})$. Also $r_1^{(i+1)}$ cannot be the only state in $R^{(j)}$: if not, then $(P^{(j)}, R^{(j)})$ would either coincide with or be less than $(P^{(i+1)}, R^{(i+1)})$.

It can be concluded that $r_1^{(i+1)} = r_1^{(j)}$, and the next transition from this state leads to the state $p_1^{(j)}$, moving to the symbol $a_i$. For the automaton to have a transition in the state $p_1^{(j)}$ at $a_i$, this state should belong to $P^{(i+1)}$. In addition, it should be the least among the state in $P^{(i+1)}$, because if there were a lesser state $p$, then the second term in the sequence for $(P^{(i+1)}, R^{(i+1)})$ would be $-p$, and this pair would be greater than $(P^{(j)}, R^{(j)})$. This leads to the equality $p_1^{(j)} = p_1^{(i+1)}$.

By analogous arguments, one can prove that the sequences for $(P^{(j)}, R^{(j)})$ and for $(P^{(i+1)}, R^{(i+1)})$ must coincide and continue infinitely. This is impossible, because the numbers of states increase, and there finitely many of them. □

**Corollary 1** (from Theorem 2)**.** *For every $n \geqslant 2$, there is a direction-determinate 2DFA with $n$ states, such that the length of the shortest string it accepts is $\binom{n}{\lfloor \frac{n}{2} \rfloor} - 1$.*

## 2.3 Longer shortest strings for automata of the general form

The main result of this section is the construction of a family of 2DFA with shortest strings of length $3 \cdot 2^{n-2} - 1$, where $n$ is the number of states in an automaton. This is more than the maximum possible length of shortest strings for direction-determinate automata; in other words, *forgetting the direction is useful.*

**Theorem 3.** *For each $n \geqslant 2$ there exists a 2DFA with $n$ states, such that the shortest string it accepts is of length $3 \cdot 2^{n-2} - 1$.*

*Proof.* The automata and the shortest strings they accept are constructed inductively; for small values of $n$ they are given in Figure 2.3.

For the inductive proof to work, the following set of properties is ensured for every $n$.

**Claim.** *For each $n \geqslant 2$ there exists a 2DFA $A_n = (\Sigma_n, Q_n, \delta_n)$ with no transitions by end-markers, no initial state and no accepting states, with the set of states $Q_n = \{1, \dots, n\}$, and there exists a string $w_n \in \Sigma_n^*$ of length $3 \cdot 2^{n-2} - 1$, such that the following two properties hold.*

1. *If $A_n$ starts at any symbol of $w_n$ in the state $n$, then it eventually leaves this string by a transition from its rightmost symbol to the right in the state 1.*
2. *If for some non-empty string $u$ there exists a position, in which the automaton $A_n$ can start in the state $n$ and eventually leave the string $u$ by a transition from its rightmost symbol to the right in the state 1, then $u$ is at least as long as $w_n$.*

The first observation is that Theorem 3 follows from this claim. Let $n \geqslant 2$, and let $A_n$ and $w_n$ be an automaton and a string that satisfy the conditions in the claim. Then $A_n$ is supplemented with an initial state $n$, a set of accepting states $\{1\}$ and a single transition by the left end-marker: from the state $n$ to the state $n$; no transitions by the right end-marker are defined. The resulting automaton $A'_n$ becomes a valid 2DFA, and it accepts the string $w_n$ as follows: from the initial state at $\vdash$ it moves to the first symbol of $w_n$ in the state $n$, then, by the first point of the claim, the automaton eventually leaves $w_n$ to the right in the state 1, and thus arrives to the right end-marker $\dashv$ in an accepting state.

To see that every string accepted by $A'_n$ is of length at least $|w_n|$, let $u$ be any accepted string. It is not empty, because on the empty string the automaton steps on the right end-marker in the state $n$ and rejects. Then, after the first step the automaton $A'_n$ is at the first symbol of $u$ in the state $n$. It cannot return to $\vdash$, because it has already used the only transition at this label, and if it ever comes back, it will reject or loop. Also the automaton cannot come to $\dashv$ in states other than 1. In order to accept, it must arrive to $\dashv$ in the state 1, and this is the first and the only time when it leaves the string $u$. Then, by the second point of the claim, the length of $u$ cannot be less than the length of $w_n$.

It remains to prove the claim, which is done by induction on $n$.

Base case: $n = 2$.

The automaton $A_2 = (\Sigma_2, Q_2, \delta_2)$ for $n = 2$ is constructed as follows. The alphabet is $\Sigma_2 = \{a, b\}$, and the set of states is $Q_2 = \{1, 2\}$. The transition function is defined by

$$\begin{aligned} \delta_2(2, a) &= (2, +1), \\ \delta_2(2, b) &= (1, -1), \\ \delta_2(1, a) &= (1, +1), \\ \delta_2(1, b) &= (1, +1). \end{aligned}$$

The string $w_2$ is $ab$, and the computation of $A_2$ on $w_2$ is presented in Figure 2.3 (top left). To be precise, computations starting in the state 2 either at $a$ or at $b$ both end by leaving the string to the right in the state 1, as claimed. There are only two shorter non-empty strings: $a$ and $b$. If the automaton starts on the string $a$ in the state 2, then it moves to the right in the state 2; on $b$, it moves to the left in the state 1. In either case,

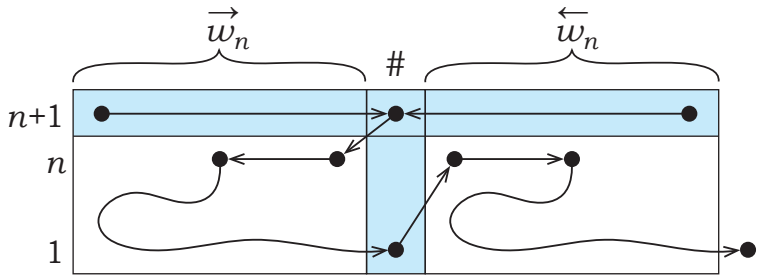


Figure 2.4: Computation of the automaton $A_{n+1}$ on the string $w_{n+1}$.

it does not go to the right in the state 1. Thus, the second point of the claim is satisfied. The length of the string is $|w_2| = 2 = 3 \cdot 2^0 - 1$.

Induction step: $n \to n+1$.

Let an $n$-state 2DFA $A_n = (\Sigma_n, Q_n, \delta_n)$ and a string $w_n \in \Sigma_n^*$ satisfy the claim. The $(n+1)$-state automaton $A_{n+1}$ satisfying the claim is constructed as follows. Let $A_{n+1} = (\Sigma_{n+1}, Q_{n+1}, \delta_{n+1})$.

- Its alphabet is $\Sigma_{n+1} = \overrightarrow{\Sigma_n} \cup \overleftarrow{\Sigma_n} \cup \{\#\}$, where $\overrightarrow{\Sigma_n} = \{\, \overrightarrow{a} \mid a \in \Sigma_n \,\}$ and $\overleftarrow{\Sigma_n} = \{\, \overleftarrow{a} \mid a \in \Sigma_n \,\}$
- The set of states is $Q_{n+1} = Q_n \cup \{n+1\} = \{1, \ldots, n+1\}$.
- The transition function is defined as follows. In the new state $n+1$, the automaton moves by all symbols with arrows in the directions pointed by the arrows.

$$\delta_{n+1}(n+1, \overrightarrow{a}) = (n+1, +1), \qquad \text{for } a \in \Sigma$$
$$\delta_{n+1}(n+1, \overleftarrow{a}) = (n+1, -1), \qquad \text{for } a \in \Sigma$$

  In all old states $1, \ldots, n$, on symbols with arrows, the new automaton works in the same way as the automaton $A_n$ on the corresponding symbols without arrows.

$$\delta_{n+1}(i, \overrightarrow{a}) = \delta_{n+1}(i, \overleftarrow{a}) = \delta_n(i, a), \qquad \text{for } a \in \Sigma \text{ and } i \in \{1, \ldots, n\}$$

  By the new separator symbol #, only two transitions are defined. In the state $n+1$, the automaton moves to the left in the state $n$, thus starting the automaton $A_n$ on the substring to the left.

$$\delta_{n+1}(n+1, \#) = (n, -1)$$

  And if the automaton gets to # in the state 1 (which happens after concluding the simulation of $A_n$ on the substring to the left), then the automaton moves to the right in the state $n$ to start the simulation of $A_n$ also on the substring to the right of the separator #.

$$\delta_{n+1}(1, \#) = (n, +1)$$

  The rest of transitions are undefined.

Note that once the automaton $A_{n+1}$ leaves the state $n+1$, it never returns to it, because there are no transitions to $n+1$ from any other state. Let $h\colon (\overrightarrow{\Sigma_n} \cup \overleftarrow{\Sigma_n})^* \to \Sigma_n^*$ be a string homomorphism which removes the arrow from the top of every symbol, that is, $h(\overrightarrow{a}) = h(\overleftarrow{a}) = a$ for all $a \in \Sigma_n$. The automaton $A_{n+1}$ works in the states $1, \ldots, n$ on symbols from $\overrightarrow{\Sigma_n} \cup \overleftarrow{\Sigma_n}$ as $A_n$ works on the corresponding symbols from $\Sigma_n$. Then, if $h(w) = w_n$ for some $w \in (\overrightarrow{\Sigma_n} \cup \overleftarrow{\Sigma_n})^*$, it follows that the automaton $A_{n+1}$, having started in the state $n$ at any symbol of $w$, eventually leaves the string $w$ by moving to the right in the state 1. Furthermore, if $|w| < |w_n|$ for some string $w \in (\overrightarrow{\Sigma_n} \cup \overleftarrow{\Sigma_n})^*$, then the automaton $A_{n+1}$, having started in the state $n$ at any symbol of $w$, cannot leave the string by moving to the right in the state 1.

The string $w_{n+1}$ is defined as $\overrightarrow{w_n} \# \overleftarrow{w_n}$, where $\overrightarrow{a_1 \ldots a_\ell} = \overrightarrow{a_1} \ldots \overrightarrow{a_\ell}$ and $\overleftarrow{a_1 \ldots a_\ell} = \overleftarrow{a_1} \ldots \overleftarrow{a_\ell}$ for every string $a_1 \ldots a_\ell \in \Sigma_n^*$. The length of $w_{n+1}$ is $|w_{n+1}| = 2|w_n| + 1 = 2(3 \cdot 2^{n-2} - 1) + 1 = 3 \cdot 2^{n-1} - 1$, as desired.

First, it is proved that the automaton $A_{n+1}$ works on the string $w_{n+1}$ as stated in the first point of the claim. Let $A_{n+1}$ start its computation on the string $w_{n+1}$ at any symbol in the state $n+1$, as shown in Figure 2.4. By the symbols in $\overrightarrow{w_n}$, the automaton moves to the right, maintaining the state $n+1$; by the symbols in $\overleftarrow{w_n}$, it moves to the left in $n+1$. Thus, wherever the automaton begins, it eventually arrives to the separator $\#$ in the state $n+1$. Next, the automaton moves to the last symbol of $\overrightarrow{w_n}$ in the state $n$. Since $h(\overrightarrow{w_n}) = w_n$, the automaton $A_{n+1}$ operates on $\overrightarrow{w_n}$ as $A_n$ on $w_n$, and leaves $\overrightarrow{w_n}$ by a transition to the right in the state 1. Then $A_{n+1}$ arrives to the separator $\#$ again, now in the state 1, and moves to the first symbol of $\overleftarrow{w_n}$ in the state $n$. As $h(\overleftarrow{w_n}) = w_n$, the automaton $A_{n+1}$ works as $A_n$ on $w_n$, and leaves $\overleftarrow{w_n}$ (and the whole string $w_{n+1}$) by moving to the right in the state 1.

Turning to the second point of the claim, it should be proved that computations of a certain form are impossible on any strings shorter than $w_{n+1}$. Let $w \in \Sigma_{n+1}^*$ be a string, and let there be a position in $w$, such that the automaton $A_{n+1}$, having started at this position in the state $n+1$, eventually leaves the string $w$ by a transition to the right in the state 1. It is claimed that $|w| \geqslant |w_{n+1}|$.

Consider the computation of $A_{n+1}$ leading out of $w$ to the right in the state 1. It begins in the state $n+1$, and the automaton maintains the state $n+1$ at all symbols except $\#$. In order to reach the state 1, there should be a moment in the computation on $w$ when the automaton arrives at some symbol $\#$ in the state $n+1$. Let $u$ be the prefix of $w$ to the left of this $\#$, and let $v$ be the suffix to the right of this $\#$; note that the substrings $u$ and $v$ may contain more symbols $\#$. It is sufficient to prove that $|u| \geqslant |w_n|$ and $|v| \geqslant |w_n|$.

Consider first the case of the suffix $v$. Let $v_0$ be the longest suffix of $v$ that does not contain the symbol $\#$; then the symbol preceding $v_0$ in $w$ is the separator $\#$, as shown in Figure 2.5. Once the automaton $A_{n+1}$ steps from the last $\#$ in $w$ to the right, it arrives to the first symbol of $v_0$ in the state $n$ (by the unique transition to the right at $\#$). The string $v_0$ cannot be empty, because $n \neq 1$. Once the automaton is inside $v_0$, it cannot return to $\#$ anymore, since it has already used the only transition to the right from $\#$, and cannot use it again without looping. Therefore, the automaton $A_{n+1}$ starts on the string $v_0 \in (\Sigma_{n+1} \setminus \{\#\})^*$ in the state $n$, and, operating as $A_n$, eventually leaves this string to the right in the state 1. Then $|v_0| \geqslant |w_n|$ by the induction hypothesis, and hence $|v| \geqslant |w_n|$.

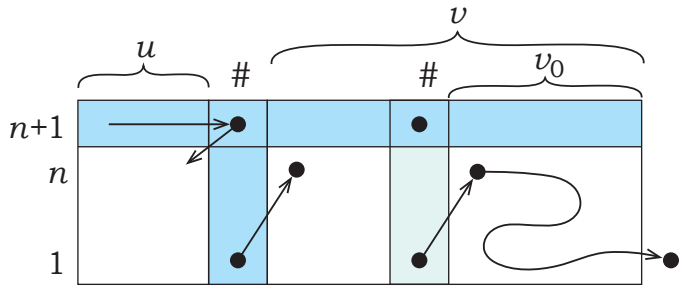


Figure 2.5: The partition $w = u\#v$ and the suffix $v_0$ of $v$.

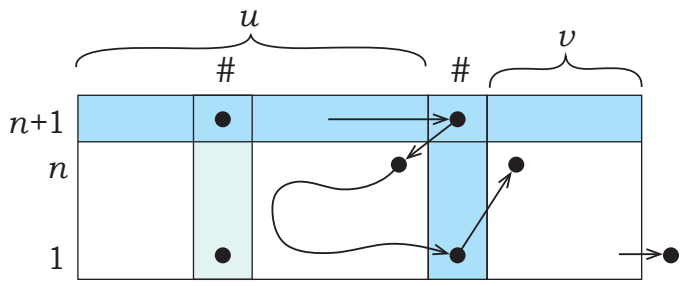


Figure 2.6: The case of computations on $u$ not reaching any separators.

Now consider the prefix $u$. Once the automaton $A_{n+1}$ comes in the state $n+1$ to the separator # between $u$ and $v$, it moves to the last symbol of $u$ in the state $n$. In order to leave the string $u$ to the right and proceed further, it must return to the separator # in the state 1, because there are no transitions by any states $\{2, \ldots, n\}$ at this separator. If there are no symbols # in $u$, or if there are some, but the automaton does not reach them, then the entire computation of $A_{n+1}$ on $u$ takes place on a certain suffix of $u$ that does not contain #, as illustrated in Figure 2.6. This computation follows a computation of $A_n$ on a string from $\Sigma_n^*$. Then, by the induction hypothesis, this suffix is not shorter than $w_n$, and therefore $|u| \geqslant |w_n|$.

The remaining case is when the automaton comes to some symbol # inside the string $u$. Let $u_0$ be the maximal suffix of $u$ not containing any symbols #, as in Figure 2.7. The automaton $A_{n+1}$ visits the separator # to the left of $u_0$, and then immediately moves from this separator back to the first symbol of $u_0$ in the state $n$ (the string $u_0$ is non-empty, because it is followed by #, which has no transitions in the state $n$). Returning back to # to the left of $u_0$ is not an option, since the unique transition by # to the right has been used already. Therefore, the automaton leaves $u_0$ by a transition to the right, and comes to the separator # between $u$ and $v$. In order to continue the computation, it should come there in the state 1. By the induction hypothesis for this computation on $u_0$, the length of $u_0$ is at least $|w_n|$. Then the length of the entire $u$ is also at least $|w_n|$.

This confirms that $|w| = |u| + 1 + |v| \geqslant |w_n| + 1 + |w_n| = |w_{n+1}|$ and completes the proof. □

## 2.4 Calculations

Bounds on the maximum length of shortest strings for small values of the number of states $n$ are given in Table 2.2.

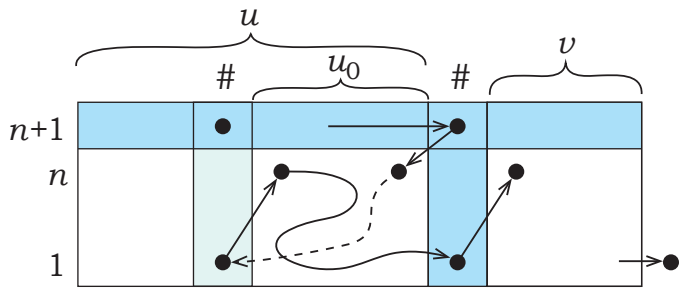


Figure 2.7: The case of computations on $u$ reaching a separator # inside $u$.

| $n$ | direction-determinate 2DFA $\binom{n}{\lfloor n/2 \rfloor} - 1$ | 2DFA of the general form | | |
|---|---|---|---|---|
| | | lower bound $3 \cdot 2^{n-2} - 1$ | computed values | upper bound $\binom{2n}{n+1} - 1$ |
| 2 | 1 | 2 | 2 | 3 |
| 3 | 2 | 5 | 6 | 14 |
| 4 | 5 | 11 | 17 | 55 |
| 5 | 9 | 23 | 32 | 209 |
| 6 | 19 | 47 | | 791 |

Table 2.2: The maximum length of shortest accepted strings for $n$-state 2DFA, for small $n$.

In the table, besides the theoretical bounds, there are also some computed values of the length of shortest strings in some automata. The example for $n = 3$ was obtained by exhaustive search, while the examples for $n = 4$ and $n = 5$ were found by heuristic search. Therefore, the maximum length of the shortest string for 3-state automata is now known precisely, for 4-state automata it is at least 17 and possibly more, and the given length of strings for 5 states is most likely much less than possible. The computations of the automata found for $n = 3$ and $n = 4$ on their shortest strings are presented in Figure 2.8.

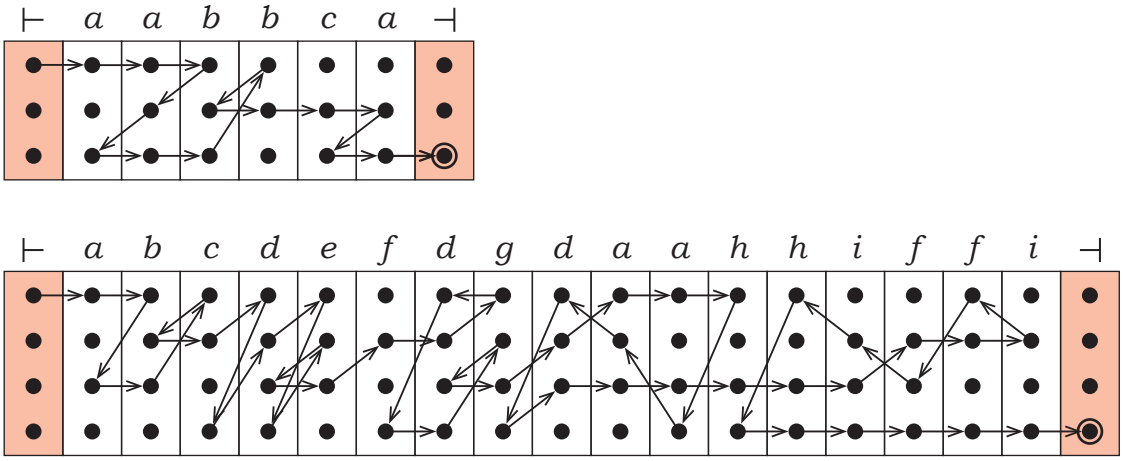


Figure 2.8: Automata found by computer programs, and their shortest strings: (top) 3 states, string of length 6; (bottom) 4 states, string of length 17.

# Chapter 3

# Minimal trees accepted by tree automata and tree-walking automata

This chapter is devoted to estimating the size of minimal trees accepted by tree automata and by tree-walking automata. In Section 3.1, these automata are formally defined. Then the exact maximum size of smallest accepted trees for tree automata is determined in Section 3.2. To get an upper bound for tree-walking automata, these automata are transformed into tree automata, and then the bound on their smallest accepted trees is applied. Bounds on the number of states needed to simulate a tree-walking automaton by a tree automaton are proved in Section 3.3. An even better upper bound on the size of minimal trees accepted by tree-walking automata is obtained in Section 3.4 by a more detailed analysis of the structure of tree automata that simulate tree-walking automata. A lower bound of a similar order on the maximum size of the smallest accepted tree for tree-walking automata is proved in Section 3.5 using the witness direction-determinate 2DFA from the lower bound proof in Chapter 2.

## 3.1 Tree-walking and tree automata

Two-way finite automata can be generalized to trees.

To make a string, it is enough to have an alphabet: a set of symbols. For trees, this is a bit more complicated, because an automaton standing at some node of a tree should know where it can move: how many children this node has, is this node a root or not. All this information needs to be determined from a symbol in a node. Node labels with information on the position of a node in its neighbourhood form a *tree signature*, a generalization of an alphabet.

**Definition 3.** A *tree signature* is a quintuple $S = (\Sigma, \mathrm{rank}, \mathrm{up}, D, (D_a)_{a\in\Sigma})$, where

- $\Sigma$ is a finite set of node labels;
- $\mathrm{rank}\colon \Sigma \to \mathbb{N}_0$ is a function that tells, for each label, how many children must nodes with this label have. If $\mathrm{rank}\, a = 0$, then $a$ is a label of leaf.
- $\mathrm{up}\colon \Sigma \to -\mathbb{N}$ is a partial function that, for each node label, tells the direction upwards to the parent (here the direction down to the $i$-th child is $+i$, and the direction up from the $i$-th child to its parent is $-i$). If $\mathrm{up}\, a$ is undefined, then $a$ is a label for a root.

- The set of directions for a label $a \in \Sigma$ is defined as $D_a = \{+1, \ldots, +\operatorname{rank} a\} \cup \{\operatorname{up} a, \text{ if } a \text{ is not a root label}\}$.
- And the set of all possible directions in the signature is $D = \bigcup_{a \in \Sigma} D_a$.

A tree signature is uniquely defined by its first three components $\Sigma, \operatorname{rank}, \operatorname{up}$, and the additional components $D, (D_a)_{a \in \Sigma}$ are determined from them.

Next, by analogy with strings over an alphabet, one can define trees over a signature. In a string, $+1$ is the direction forward, and $-1$ is the direction backward, whereas in a tree, $+i$ is the direction to the $i$-th child, and $-i$ is the direction from the $i$-th child up to its parent. The following definition effectively says that all information on the position of a node in its neighbourhood written in its label must be true.

**Definition 4.** A *tree* over a signature $S = (\Sigma, \operatorname{rank}, \operatorname{up}, D, (D_a)_{a \in \Sigma})$ is a quadruple $T = (V, v_0, +, \lambda)$, where

- $V$ is a non-empty finite set of nodes,
- $v_0 \in V$ is the root,
- $+\colon V \times D \to V$ is a partial function that, for a node and direction, gives the neighbouring node in that direction. For each $i \in D$, if $v + i$ is defined, then $(v + i) + (-i)$ must be defined and be equal to $v$. Denote $v + (-i)$ by $v - i$.
- $\lambda\colon V \to \Sigma$ is a labelling function that assigns some node label to each node. The root $v_0$ must get a root node label. For a node $v$ labelled with $\lambda(v) = a$, the function $+$ must be defined precisely for directions from the set $D_a = \{\operatorname{up} a, \text{ if } a \text{ is not a root label}\} \cup \{+1, +2, \ldots, +\operatorname{rank} a\}$.

And a tree-walking automaton moves over a tree, choosing where to move at each step, and in which state, according to the current state and the label of the current node (just like a two-way finite automaton moves over a string).

**Definition 5.** A deterministic tree-walking automaton (DTWA) over a signature $S = (\Sigma, \operatorname{rank}, \operatorname{up}, D, (D_a)_{a \in \Sigma})$ is a quadruple $A = (Q, q_0, F, \delta)$, where

- Q is a finite set of states;
- $q_0 \in Q$ is the initial state;
- $F \subseteq Q \times \Sigma$ is a set of acceptance conditions;
- $\delta\colon (Q \times \Sigma) \setminus F \to Q \times D$ is a partial transition function, with $\delta(q, a) \in Q \times D_a$ on its domain.

A tree-walking automaton $A$ works on a tree just like a two-way automaton works on a string.

A *configuration* of a tree-walking automaton $A$ on a tree $T$ is a pair $(q, v)$, where $q \in Q$ is the current state of the automaton, and $v \in V$ is the node where the automaton currently is.

A *computation* of a deterministic tree-walking automaton $A$ on a tree $T$ is a sequence of configurations $C_0, \ldots, C_N$ constructed inductively. The automaton starts in the initial state at the root of the tree: $C_0 = (q_0, v_0)$. For a configuration $C_j = (q, v)$, the next configuration is defined as follows.

- If the transition is defined as $\delta(q, \lambda(v)) = (r, d)$, then the automaton changes its state to $r$ and moves in the direction $d$, and the next configuration is $C_{j+1} = (r, v + d)$.
- If the automaton gets to an accepting pair $(q, \lambda(v)) \in F$, then it accepts, and the configuration $C_j = C_N$ is the last one.
- If $\delta(q, \lambda(v))$ is undefined and the acceptance condition does not hold, then the automaton rejects, and $C_j = C_N$ is the last configuration.

An automaton can accept a tree, reject it or loop (it loops if the computation is infinite, and $N = \infty$).

A tree-walking automaton defines the language $L(A)$ of all trees that it accepts.

A *nondeterministic tree-walking automaton* (NTWA) can be defined as well. It can have several initial states (a set $Q_0$), and several possible transitions in a configuration, defined by a function $\delta\colon (Q \times \Sigma) \setminus F \to 2^{Q \times D}$. There may be multiple computations on a tree, and the tree is accepted if at least one of them is accepting.

The above definitions of tree-walking automata (DTWA, NTWA) are essentially equivalent to the definitions used by Bojańczyk and Colcombet [4, 5] even though there are some differences in the notation: Bojańczyk and Colcombet do not encode the structure of the tree in node labels, and instead provide them to an automaton in the transition function, so that the automaton gets the same information.

Tree-walking automata are a generalization of two-way finite automata. Next, another type of automata on trees, which generalizes one-way nondeterministic finite automata (NFA), will be defined. First, the definition of NFA is given.

**Definition 6.** A *one-way nondeterministic finite automaton* (NFA) is a quintuple $A = (\Sigma, Q, Q_0, \delta, F)$, in which:

- $\Sigma$ is a finite alphabet;
- $Q$ is a finite set of states;
- $Q_0 \in Q$ is the set of initial states;
- $\delta\colon Q \times \Sigma \to 2^Q$ is a transition function;
- $F \subseteq Q$ is the set of accepting states.

A computation of the automaton $A$ on a string $w = a_1 \dots a_m \in \Sigma^*$ is a sequence of states $q_0, q_1, \dots, q_m$, such that $q_0 \in Q_0$ is one of the initial states, and $q_i \in \delta(q_{i-1}, a_i)$ for each $i = 1, \dots, m$. The computation is accepting if $q_m \in F$. A string is accepted if there is at least one accepting computation on it.

A nondeterministic tree automaton chooses a state in each node, so that each triple (the state in a node, the label in this node, the vector of states in its children) satisfies some conditions.

**Definition 7.** *A nondeterministic tree automaton (NTA)* over a signature $S = (\Sigma, \mathrm{rank}, \mathrm{up}, D, (D_a)_{a \in \Sigma})$ is a pair $A = (Q, \delta)$, where

- $Q$ is a finite set of states;

- $\delta \subseteq \bigcup_{a \in \Sigma} Q \times \{a\} \times Q^{\operatorname{rank} a}$ is a transition function that specifies the set of admissible triples of a state $q \in Q$ in a node, its label $a \in \Sigma$ and a vector of states $(q_1, \ldots, q_{\operatorname{rank} a})$ in its children. If $\operatorname{rank} a = 0$, then $\{\, q \mid (q, a, ()) \in \delta \,\}$ is the set of admissible states in a leaf labelled with $a$.

A nondeterministic tree automaton works on a tree like an NFA works on a string. It starts in any state in the root, and then, for the state and the label in a node it nondeterministically chooses the vector of states in the children (which is admissible for this current state and this label of the current node).

There is an alternative outlook: the automaton computes the states from the leaves to the root. Once the states in the children of a node are computed, the automaton nondeterministically chooses the state in this node, so that the triple is admissible.

The first approach (from root to leaves) is called *top-down*, and it directly generalizes the definition of an NFA. The second approach is called *bottom-up*. Both approaches are equivalent, and the following definition of a computation fits both approaches.

**Definition 8.** A computation of a nondeterministic tree automaton $A = (Q, \delta)$ on a tree $T = (V, v_0, +, \lambda)$ is an assignment $(q(v))_{v \in V}$ of states to nodes, such that for each node $v \in V$ the triple at this node is admissible, that is, $(q(v), \lambda(v), (q(v+1), \ldots, q(v + \operatorname{rank} \lambda(v)))) \in \delta$.

A tree automaton accepts a tree if there exists at least one computation on this tree.

In the definition of nondeterministic tree automata, no initial and accepting states are needed, because all restrictions can be defined by removing some triples from $\delta$.

Computations on subtrees can also be defined. In a tree, the root has a root label and no edge upwards. The root of a proper subtree has a non-root label and an edge to its parent. A subtree can be defined independently of the main tree. A *subtree* over a signature is an object that satisfies all conditions in the definition of a tree, except the requirement that its root has a root node label. An edge upwards from the root, if it exists, leads nowhere and is called an *external edge*.

A *computation on a subtree* has the same definition as a computation on a tree: this is a mapping from nodes to states $(q(v))_{v \in V}$ , in which the state in every node satisfies the admissibility condition involving the vector of states in its children and the label of this node.

Languages of trees recognized by nondeterministic tree automata are called *regular tree languages*.

## 3.2 Bounds on the size of the minimal accepted tree for a tree automaton

For a nondeterministic tree automaton, the following bound on the size of the smallest accepted tree is proved.

**Theorem 4.** *Let $S = (\Sigma, \operatorname{rank}, \operatorname{up}, D, (D_a)_{a \in \Sigma})$ be a tree signature, and let $r = \max\{\, \operatorname{rank} a \mid a \in \Sigma \,\}$ be the maximum possible number of children in a node. Let $A = (Q, \delta)$ be a nondeterministic tree automaton over the signature $S$, with $|Q| = n$ states, that accepts at least one tree. Let $m_0, \ldots, m_{rn}$ be a sequence of integers of length*

$rn+1$ *defined as follows:* $m_j = 2^j$ *for all* $j = 0, \ldots, r-1$, *and* $m_j = 1 + \sum_{i=j-r}^{j-1} m_i$ *for all* $j = r, \ldots, rn$.

*Then the minimal tree accepted by* $A$ *has at most* $m_{rn} \leqslant 2^{rn}$ *nodes.*

*Proof.* Under certain conditions, a subtree in an accepted tree can replaced by another subtree, so that the tree is still accepted. Let $T$ be a tree accepted by $A$, let $(q(v))_{v \in V}$ be a computation on this tree, and let $t_1$ be a subtree rooted at some node $v$. If $t_2$ is a subtree that has a computation with the state $q(v)$ in the root of $t_2$, and the direction of the external edge of $t_2$ is the same as in $t_1$, then the subtree $t_1$ in the tree $T$ can be replaced with $t_2$, and the new tree will be accepted as well. Indeed, the computation on the new tree can be obtained by merging the computation on $T$ in the old nodes with the computation on $t_2$ on the nodes of $t_2$.

Let $T$ be a minimal accepted tree, let $(q(v))_{v \in V}$ be an accepting computation on this tree. Then, for every node $v \in V$, the subtree rooted at $v$ must the minimal subtree that has a computation with the state $q(v)$ in the root, and with the same direction of the external edge as in $v$.

For each state $q \in Q$ and for each number of child $i \in \{1, \ldots, r\}$, fix $T_{q,i}$ as one of the minimal subtrees that have an external edge in the direction $-i$, and a computation that produces the state $q$ in the root of the subtree. If there are no such subtrees, then $T_{q,i}$ is undefined. For a subtree $T_{q,i}$, fix a computation $C_{q,i}$ on it that has $q$ in the root of the subtree. Let $f \colon Q \times \{1, \ldots, r\} \to \mathbb{N}$ be a partial function that, for a state $q \in Q$ and a child number $i \in \{1, \ldots, r\}$, gives the number of nodes in the subtree $T_{q,i}$.

Now, all pairs $(q, i)$, on which the function is defined, are sorted in the order of increasing value of $f(q, i)$. Let the values of the function $f$ on these sorted pairs be $m'_0 \leqslant m'_1 \leqslant \ldots \leqslant m'_N$. Here $N$ is less than $rn$, because there are $rn$ distinct pairs $(q, i)$, with $q \in Q$ and $i \in \{1, \ldots, r\}$.

For each pair $(q, i)$, on which $f$ is defined, there is a subtree $T_{q,i}$ with a computation $C_{q,i}$, in which the state $q$ is reached at the root of the subtree, and the external edge from the root goes in the direction $-i$. Let $r'$ be the number of children of the root of $T_{q,i}$, then $0 \leqslant r' \leqslant r$. Let $q_1, \ldots, q_{r'}$ be the states in the children of the root in the computation $C_{q,i}$. Since the subtree $T_{q,i}$ is minimal, the subtrees in the children on the root cannot be replaced with smaller subtrees, and hence have as many nodes as the subtrees $T_{q_1,1}, \ldots, T_{q_{r'},r'}$. Then $f(q, i) = 1 + f(q_1, 1) + \ldots + f(q_{r'}, r')$. The numbers $f(q_1, 1)$, $\ldots$, $f(q_{r'}, r')$ are smaller than $f(q, i)$, and therefore the pairs $(q_1, 1), \ldots, (q_{r'}, r')$ occur in the sorted list of pairs earlier than the pair $(q, i)$; the pairs $(q_j, j)$ are pairwise distinct, because they have different directions $-j$. It can be concluded that every next number in the sequence $m'_0, m'_1, \ldots, m'_N$ is a sum of 1 and at most $r$ numbers previous occurring in the sequence, with pairwise distinct indices.

Each number $m'_j$, with $j = 0, \ldots, N$, is bounded by the sum $r$ preceding numbers in the sequence (if there are at least $r$ of them) plus one, or the sum of all preceding numbers (if there fewer than $r$ of them) plus one. Then, for $j = 0, \ldots, r-1$, the upper bound $m'_j \leqslant 2^j = m_j$ is obtained by induction. Every next number is bounded by $m'_j \leqslant 1 + \sum_{i=j-r}^{j-1} m'_i \leqslant m_j$, for all $j = r, \ldots, N$. The root of a minimal accepted tree $T$ has at most $r$ children with different directions up, and the number of nodes $m'_{N+1}$ in the whole tree does not exceed $1 + m'_{N-r+1} + \ldots + m'_N \leqslant m_{N+1} \leqslant m_{rn}$ (the latter inequality holds because $N + 1 \leqslant rn$).

If, instead of these sums of at most $r$ previous elements of the sequence, one always

summed up *all* previous elements, then powers of two would be obtained. This gives an upper bound on $m_j$ of the form $m_j \leqslant 2^j$, for all $j = 0, \ldots, rn$, and hence, $m_{rn} \leqslant 2^{rn}$. $\square$

The upper bound from Theorem 4 is actually precise: in the next theorem, an example of an automaton is given, which has minimal accepted tree of the same size. Furthermore, the lower bound is given already for a deterministic bottom-up tree automaton. (in this automaton, $\delta$ is a partial function that maps a label $a \in \Sigma$ and a vector of states in the children $(q_1, \ldots, q_{\operatorname{rank} a})$ to a state in a node).

**Theorem 5.** *For every $r \geqslant 1$ there is a signature $S_r$, with maximum rank of a label $r$, and with $2r+1$ node labels, such that for every $n \geqslant 1$ there exists an $n$-state deterministic bottom-up tree automaton $A$ over the signature $S_r$, such that the minimal tree accepted by $A$ has exactly $m_{rn}$ nodes, where $m_j = 2^j$ for $j = 0, \ldots, r-1$, and $m_j = 1 + \sum_{i=j-r}^{j-1} m_i$ for $j = r, \ldots, rn$.*

*Proof.* The goal is to define a signature and an automaton, so that the automaton accepts exactly one tree, and that tree has the desired number of nodes.

Such a deterministic bottom-up tree automaton is defined as $A = (Q, \delta)$, where $Q = \{1, \ldots, n\}$ is the set of states. For each state $q \in Q$ and for each direction $i \in \{1, \ldots, r\}$, it is planned that there is a unique subtree $T_{q,i}$ with an external edge $-i$, on which the (uniquely defined) bottom-up computation ends with the state $q$ in the root.

The signature $S_r = (\Sigma, \operatorname{rank}, \operatorname{up}, D, (D_a)_{a \in \Sigma})$ is defined as follows.

- Node labels are: $\Sigma = \{a_0, \ldots, a_{r-1}\} \cup \{b_{-1}, \ldots, b_{-r}\} \cup \{c\}$.
- Each node label $a_i$, with $i = 0, \ldots, r-1$, has rank $i$. The rest of the node labels have rank $r$.
- Each label $a_i$, with $i = 0, \ldots, r-1$, has direction upward $\operatorname{up} a_i = -(i+1)$. The direction upward for $b_{-i}$, with $i = 1, \ldots, r$, is $\operatorname{up} b_{-i} = -i$. The label $c$ is a root label, its direction upward is undefined.

Let $((q_j, i_j))_{j \in 0, \ldots, rn-1}$ be the ordered set of all pairs $(q, i)$, with $q \in Q$ and $i = 1, \ldots, r$. These pairs are compared first by $q$ in the increasing order, and then by $i$.

For each pair $(q_j, i_j)$, with $j = 0, \ldots, rn-1$, the transition function $\delta$ will have a unique transition involving a node label with upward direction $-i_j$, which produces the state $q_j$. The corresponding subtrees $T_{q_j, i_j}$ with external edge in the direction $-i_j$ and with the automaton producing the state $q_j$ in the root will be constructed along with defining such transitions inductively on $j$, as shown in Figure 3.1.

For $j = 0, \ldots, r-1$, the state is $q_j = 1$ and direction $-i_j$ equals $-(j+1)$. Then the transition is $\delta(1, \ldots, 1, a_j) = 1$ (where the number of arguments 1 is $j$). The corresponding subtree $T_{q_j, i_j} = T_{1, j+1}$ has the label $a_j$ in the root, with upward direction $-(j+1)$, and with attached subtrees $T_{1,1}$, $T_{1,2}$, …, $T_{1,j}$. The computation on each of the latter subtrees produces state 1, and by the above transition the computation on $T_{1,j+1}$ also produces state 1.

For each $j \in \{r, \ldots, rn-1\}$, the corresponding subtree $T_{q_j, i_j}$ is comprised of a root with label $b_{-i_j}$ and with upward direction $-i_j$, and of $r$ attached subtrees. The attached subtrees are the preceding subtrees in the sequence, which have already been constructed by the induction hypothesis: $T_{q_{j-r}, i_{j-r}}$, …, $T_{q_{j-1}, i_{j-1}}$. The sequence $(q_j, i_j)_{j=0}^{rn-1}$ is sorted

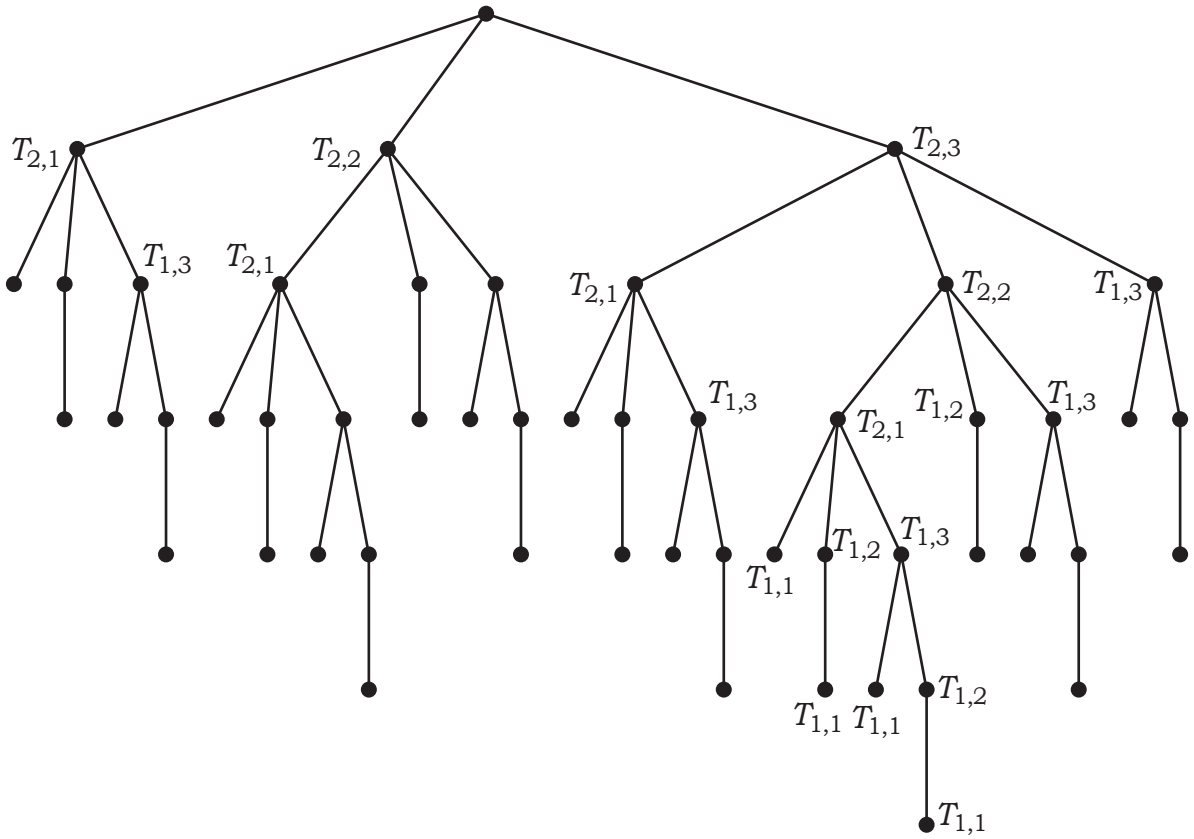


Figure 3.1: Inductive definition of subtrees $T_{q_j,i_j}$, for $n = 2$ and $r = 3$.

lexicographically (first by $q_j$, then by $i_j$), and hence the sequence of directions $(i_j)_{j=0}^{rn-1}$ is periodic with period $r$, that is, $i_j = (j \mod r) + 1$, for all $j \in \{0, \ldots, rn-1\}$. Therefore, the requested previous $r$ subtrees for $T_{q_j,i_j}$ have distinct directions upwards, and can be attached to the new root, because its label $b_{-i_j}$ has all possible directions down. Each subtree has to be attached in its proper direction, and so the subtrees $T_{q_{j-r},i_{j-r}}$, $\ldots$, $T_{q_{j-1},i_{j-1}}$ are attached to the node $b_{-i_j}$ cyclically shifted, starting with the one with upward direction $-1$. Let $s \in \{1, \ldots, r\}$ be the number with $i_{j-s} = 1$. Then a new transition $\delta(q_{j-s}, \ldots, q_{j-1}, q_{j-r}, \ldots, q_{j-s-1}, b_{-i_j}) = q_j$ is defined, and by this transition the state $q_j$ is computed in the root of the subtree $T_{q_j,i_j}$.

Thus, the subtrees $T_{q_j,i_j}$ have been constructed for all $j = 0, \ldots, rn-1$.

**Claim 1.** *For every state $q \in Q$ and for every direction $i \in \{1, \ldots, r\}$, the subtree $T_{q,i}$ is the unique subtree with direction $-i$ upwards, and with the state $q$ computed in its root.*

*Proof.* Let $j$ be the index of the pair $(q, i)$ in the sequence: $(q, i) = (q_j, i_j)$. The proof is by induction on $j$.

Let $T$ be any subtree with upward direction $-i$, on which the automaton computes the state $q$. By the construction of $\delta$, there is a unique transition by a label with upward direction $-i$, by which the state $q$ is computed. Only this transition can be used in the root of $T$. This transition defines the label in root of $T$, the number of its children and the state in each child (the same as for the root of $T_{q_j,i_j}$).

By the construction of the transition function, all pairs (state, direction upwards) for the children of the root of $T_{q_j,i_j}$ are listed in the ordered sequence just before $(q_j, i_j)$:

these are the pairs $\{(q_{j-1}, i_{j-1}), \ldots, (q_{j-t}, i_{j-t})\}$, with $t = \min\{r, j\}$. By the induction hypothesis, the subtrees for these pairs are unique, and therefore the root of $T$ has the same attached subtrees as the root of $T_{q_j,i_j}$. Hence, $T$ and $T_{q_j,i_j}$ coincide. □

**Claim 2.** *Each subtree $T_{q_j,i_j}$, with $j \in \{r, \ldots, rn-1\}$, has exactly $m_j$ nodes.*

*Proof.* Induction on $j$. For $j = 0, \ldots, r-1$, the subtree $T_{q_j,i_j}$ consists of the root and all previous subtrees, and so it has $1 + \sum_{t=0}^{j-1} m_t = 1 + \sum_{t=0}^{j-1} 2^t = 2^j = m_j$ nodes.

Every next subtree $T_{q_j,i_j}$, with $j \in \{r, \ldots, rn-1\}$, consists of its root and $r$ previous subtrees. Hence, it has $1 + \sum_{t=j-r}^{j-1} m_t = m_j$ nodes, as claimed. □

Only a node labelled with $c$ can be the root of the entire tree. For this label, a unique transition $\delta(n, n, \ldots, n, c) = n$ is defined, with $r$ arguments $n$. This transition requires $r$ last subtrees in the sequence, $T_{n,1}, \ldots, T_{n,r}$, to be attached to the root. Therefore, a unique tree is accepted, and it has exactly $m_{rn} = 1 + m_{rn-1} + \ldots + m_{rn-r}$ nodes. □

## 3.3 Transformation of a nondeterministic tree-walking automaton to a nondeterministic tree automaton

It is known that if a 2DFA with $n$ states accepts at least one string, then it accepts some string of length at most $\binom{2n}{n+1} - 1 = O(\frac{1}{\sqrt{n}}4^n)$. This bound is proved by first transforming an $n$-state 2DFA to an NFA with $m = \binom{2n}{n+1}$ states (see Kapoutsis [15]), and then observing that an $m$-state NFA recognizing a non-empty language must accept some string of length at most $m-1$. The currently best lower bound on the maximum length of the shortest accepted string for an $n$-state 2DFA is $\frac{3}{4} \cdot 2^n - 1$, proved in Chapter 2.

For the generalization of 2DFA to trees, the deterministic tree-walking automata (DTWA), the size of the minimal accepted tree has not previously been studied. However, an upper bound can be obtained by extending the method of Kapoutsis to trees. As a generalization of NFA to trees, one can take a nondeterministic tree automaton (NTA). Then, in order to apply the method of Kapoutsis, one should prove an upper bound for the transformation of DTWA to NTA and then apply an upper bound in Theorem 4 on the size of the minimal tree accepted by a nondeterministic tree automaton, proved in Section 3.2.

In this section, bounds on the number of states needed for the transformation of DTWA to NTA are proved. An upper bound for this transformation applies to nondeterministic tree-walking automata.

**Theorem 6.** *Let $S = (\Sigma, \mathrm{rank}, \mathrm{up}, D, (D_a)_{a\in\Sigma})$ be a tree signature, and let $A = (Q, Q_0, F, \delta)$ be a nondeterministic tree-walking automaton (NTWA) with $n$ states. Then there exists a nondeterministic tree automaton (NTA) $A' = (Q', \delta')$ over $S$ with $\binom{2n+1}{n+1} - 1 = O(\frac{1}{\sqrt{n}}4^n)$ states, which accepts the same set of trees as $A$.*

*Proof.* The construction is similar to the 2NFA→NFA transformation by Kapoutsis [15]. The states of the tree automaton are:

$$Q' = \{\, (P, R) \mid P, R \subseteq Q,\ (P, R) \neq (Q, Q),\ \text{and either } |R| = |P| + 1, \text{ or } |R| = |P| \,\}$$

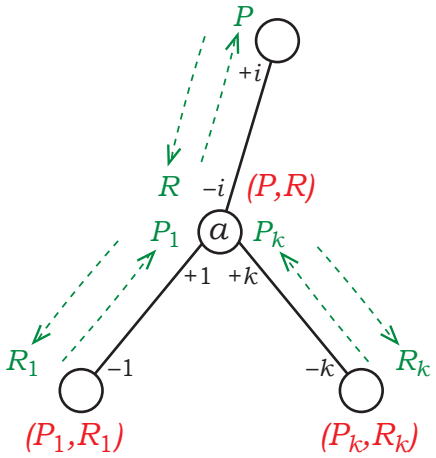


Figure 3.2: Simulation of NTWA by NTA transition $((P, R), a, ((P_1, R_1), \ldots, (P_k, R_k))) \in \delta'$.

First, the number of states is determined. A pair $(P, R) \in Q' \cup \{(Q, Q)\}$ is chosen by selecting any $|R| + |Q \setminus P|$ elements from two columns of height $n$, and letting $R$ be the elements selected from the first column, and $P$ be the elements *not selected* from the second column, Then, for one of the equalities $|R| = |P| + 1$ or $|R| = |P|$ to hold, $|R| + |Q \setminus P|$ must be either $n$ or $n+1$, and accordingly $|Q'| = \binom{2n}{n} + \binom{2n}{n+1} - 1 = \binom{2n+1}{n+1} - 1$, as claimed in the theorem. The growth rate of this binomial coefficient is known: $\binom{2n+1}{n+1} = O(\frac{1}{\sqrt{n}} 4^n)$.

The nondeterministic tree-walking automaton $A$ can be assumed to have a unique initial state. If it has several, then one can choose any $q_0 \in Q_0$ as the initial state, and define transitions in $q_0$ at root node labels to simulate the transitions of other initial states.

The idea is that if the tree automaton comes to the state $(P, R)$ at a node $v$, this means that $A$, in its accepting computation, arrives to $v$ from the parent of $v$ in all states from $R$, and rises from $v$ up to its parent in all states from $P$. In the construction by Kaputsis [15] for the case of 2NFA on strings, the only option is $|R| = |P| + 1$, because a 2NFA accepts only at the right end-marker. But a tree-walking automaton may accept anywhere, and hence, if it does not accept in the subtree of $v$, then $|R| = |P|$.

The transition function $\delta'$ of the tree automaton $A'$ will be defined so that it may encode any accepting computation of $A$ without cycles, and possibly some cycles from the graph of all computations of $A$ that are disjoint with the accepting path.

Note that the missing state $(Q, Q) \notin Q'$ is not needed to encode an accepting computation without cycles, because if the automaton goes up from a node in every state, then, in particular, it goes up in the same state which it has earlier used on the way down. This yields a cycle in the computation.

The transition function $\delta'$ of the automaton $A'$ is defined as follows (see Figure 3.2). Let a node $v$ have label $a \in \Sigma$, with $\operatorname{rank} a = k$ and $\operatorname{up} a = -i$ ($i$ is undefined if a $a$ is a root label). Let the states in its children be $(P_1, R_1), \ldots, (P_k, R_k)$. Can the node $v$ have a given state $(P, R)$? There is a possible transition $((P, R), a, ((P_1, R_1), \ldots, (P_k, R_k))) \in \delta'$ if and only if the following conditions are met.

1. The sets $P_1, \ldots, P_k$ and $R$ must be pairwise disjoint. If $a$ is a root label, then $\{q_0\}$

also must be disjoint with these sets.

2. Let $Q_1 = P_1 \cup \ldots \cup P_k \cup R \cup \{\, q_0 \mid \text{if } a \text{ is a root label} \,\}$ be the set of all states encoded in the node $v$. Let $M = (P \times \{-i\}) \cup (R_1 \times \{+1\}) \cup \ldots \cup (R_k \times \{+k\})$ be the set of all encoded transitions from the node $v$. Let $\widehat{Q}_1 = \{\, q \in Q_1 \mid (q, a) \notin F \,\}$ be all states from $Q_1$ that are not accepting at label $a$.

   Then each of the transitions from $M$ must be executed exactly once, in different states from $\widehat{Q}_1$, and all these states must be used. That is, there should exist a bijection $f\colon \widehat{Q}_1 \to M$, such that $f(q) \in \delta(q, a)$ for all $q \in \widehat{Q}_1$.

3. If $a$ is a root label, then $(P, R)$ must be $(\varnothing, \varnothing)$, because the automaton cannot walk over a non-existent edge upwards.

A nondeterministic tree automaton $A'$ has been defined, and it remains to prove that the automata $A$ and $A'$ accept the same set of trees.

Let the tree-walking automaton $A$ accept a tree $T$. Then there is an accepting computation $C$ of $A$ on the tree $T$ without cycles. Let $C'$ be a computation of the tree automaton $A'$ on the tree $T$, defined as follows: the state at a node $v$ is $q(v) = (P, R)$, where $P$ is the set of all states in which the automaton $A$ in the computation $C$ arrives to the parent of $v$ from $v$, and $R$ is the set of all states in which $A$ in the computation $C$ moves down to $v$ from its parent. Then $C'$ satisfies all conditions in the definition of $\delta'$ in each node $v$, and is indeed a computation, which confirms that $A'$ accepts the tree $T$.

Now let the tree automaton $A'$ accept a tree $T = (V, v_0, +, \lambda)$. It has to be proved that $A$ also accepts it. Let $C'$ be any computation of $A'$ on $T$. Denote the state at every node $v \in V$ in the computation $C'$ by $(P_v, R_v)$.

For every node $v$, denote by $Q_{1,v}$, $\widehat{Q}_{1,v}$ and $M_v$ the sets $Q_1$, $\widehat{Q}_1$ and $M$ in the definition of $\delta'$ for $v$ and the computation $C'$. Fix any bijection $f_v\colon \widehat{Q}_{1,v} \to M_v$ as in the definition of $\delta'$; then $f_v(q) \in \delta(q, \lambda(v))$ for every $q \in \widehat{Q}_{1,v}$.

Denote the set of configurations of $A$ encoded in a node $v$ by $X_v = Q_{1,v} \times \{v\}$ and let $\widehat{X}_v = \widehat{Q}_{1,v} \times \{v\}$ be the subset of non-accepting configurations of $X_v$. The set of all configurations entered by the automaton $A$ after applying transitions from $M_v$ at the node $v$ is denoted by $Y_v$. If $k$ is the number of children of $v$, and $-i$ is the direction upwards (undefined for the root), then $Y_v = (P_v \times \{v-i\}) \cup (R_{v+1} \times \{v+1\}) \cup \ldots \cup (R_{v+k} \times \{v+k\})$. In the new notation, the bijection $f_v$ gives rise to another bijection $g_v\colon \widehat{X}_v \to Y_v$, such that, for every $x \in \widehat{X}_v$, the configuration $g_v(x)$ is one of the possible next configurations of the automaton $A$ for the configuration $x$.

The sets of configurations $X_v$ are pairwise disjoint for different $v \in V$, because each $X_v$ contains only configurations at $v$. It is claimed that the sets $Y_v$ for different $v \in V$ are also disjoint, that is, no configuration $(q, v)$, with $q \in Q$ and $v \in V$, may occur in two such sets. Let $v$ have $k$ children and upward direction $-i$. A configuration with the node $v$ may lie in $Y_u$ only if $u$ is a neighbour of $v$. In the set $Y_{v-i}$, configurations with $v$ are those in $R_v \times \{v\}$; in the sets $Y_{v+1}, \ldots, Y_{v+k}$, the sets of configurations with $v$ are $P_{v+1} \times \{v\}$, …, $P_{v+k} \times \{v\}$. By the condition in the definition of the function $\delta'$, the sets $R_v, P_{v+1}, \ldots, P_{v+k}$ are disjoint, and hence the configuration $(q, v)$ may occur in at most one set $Y_u$ with $u \in V$.

Then, let $X = \bigcup_{v \in V} X_v$ be the set of all encoded configurations, and let the set $\widehat{X}$ contain all non-accepting configurations from $X$. Let $Y = \bigcup_{v \in V} Y_v$ be the set of

all configurations reached by encoded transitions. Since all sets $X_v$, with $v \in V$, are pairwise disjoint, and all set $Y_v$ are pairwise disjoint as well, the bijections $(g_v)_{v \in V}$ can be combined into one big bijection $g\colon \widehat{X} \to Y$. The function $g$ maps each configuration in $\widehat{X}$ to a possible next configuration.

The plan is to take the initial configuration and to apply $g$ iteratively to produce an accepting computation of $A$. For these configurations to form a sequence, it has to be shown that all configurations from $Y$ lie in $X$. A stronger claim is established below: it is also shown that each non-initial encoded configuration is a next configuration to another encoded configuration.

**Claim 3.** $Y = X \setminus \{(q_0, v_0)\}$.

*Proof.* By the definition of $Y_v$, the set $Y$ is of the following form.

$$Y = \bigcup_{v \in V} Y_v = \bigcup_{v \in V} (P_v \times \{v + \operatorname{up} \lambda(v)\}) \cup (R_{v+1} \times \{v+1\}) \cup \ldots \cup (R_{v+\operatorname{rank} \lambda(v)} \times \{v + \operatorname{rank} \lambda(v)\})$$

Next, the pairs are regrouped according to the nodes in the second components: if $\operatorname{up} \lambda(v) = -i$ and $u = v - i$, then the set of pairs $P_v \times \{v - i\}$ equals $P_{u+i} \times \{u\}$ in the new notation. At the same time, the sets of pairs $R_{v+j} \times \{v + j\}$ are represented as $R_u \times \{u\}$, for all $u$. The following equality holds as a result.

$$\bigcup_{v \in V} (P_v \times \{v + \operatorname{up} \lambda(v)\}) \cup (R_{v+1} \times \{v+1\}) \cup \ldots \cup (R_{v+\operatorname{rank} \lambda(v)} \times \{v + \operatorname{rank} \lambda(v)\}) = \\ = \bigcup_{u \in V} \big( (P_{u+1} \cup \ldots \cup P_{u+\operatorname{rank} \lambda(u)} \cup R_u) \times \{u\} \big)$$

(an extra expression $R_{v_0} \times \{v_0\}$ for the root of the tree occurs in the second part; however, it does not affect the resulting value, because $R_{v_0} = \varnothing$ by one of the conditions in the definition of $\delta'$)

This is the union of all $X_u$ without the initial configuration.

$$\bigcup_{u \in V} \big( (P_{u+1} \cup \ldots \cup P_{u+\operatorname{rank} \lambda(u)} \cup R_u) \times \{u\} \big) = \bigcup_{u \in V} \big( X_u \setminus \{(q_0, v_0)\} \big) = X \setminus \{(q_0, v_0)\}$$

□

Since the function $g\colon \widehat{X} \to X \setminus \{(q_0, v_0)\}$ is a bijection, there is exactly one accepting configuration in $X$. For each configuration except the accepting one, $g$ provides one of the successor configurations of the nondeterministic automaton $A$. All these next configurations are distinct, and the initial configuration is not a $g$-successor to any configuration in $X$. Therefore, all configurations in $X$ are split into a path from the initial configuration to the accepting one, and possibly several disconnected cycles. This path from initial to accepting configuration is the desired accepting computation of the automaton $A$ on the tree $T$. □

As in the transformation by Kapoutsis [15] for automata on strings, it will be proved that the number of states used in the above transformation of NTWA to NTA cannot be reduced in the worst case already for deterministic tree-walking automata. The lower bound argument will use signatures with exponentially many node labels, just as in the

Kapoutsis's proof for automata on strings, where an alphabet of exponential size was actually necessary to achieve the precise bound, see Geffert and Okhotin [12].

The upper bound $\binom{2n+1}{n+1}-1$ in Theorem 6 is not much greater than the $\binom{2n}{n+1}$ bound for the 2NFA→NFA transformation by Kapoutsis [15]. It turns out that this small increase is not due to the branching in the trees, but only because tree-walking automata may accept anywhere, whereas 2NFA accept only at the right end-marker. A precise lower bound to Theorem 6 will be proved already in the case of strings, that is, for the transformation of 2NFA accepting anywhere to one-way NFA.

The proof uses the standard lower bound method for one-way NFA: the *fooling sets*.

**Lemma 5** (Birget [2]; Glaister and Shallit [11])**.** *Let $L \subseteq \Sigma^*$ be a language, and let $\{(u_1, v_1), \ldots, (u_N, v_N)\}$ be a set of pairs of strings over $\Sigma$, such that*

- *for each $i = 1, \ldots, N$ the concatenation $u_i v_i$ is in $L$,*
- *for all $i, j = 1, \ldots, N$, with $i \neq j$, at least one of the strings $u_i v_j$ and $u_j v_i$ is not in $L$*

*(such a set of pairs is called a fooling set). Then every NFA recognizing the language $L$ must have at least $N$ states.*

This method is used in the proof of the following lower bound.

**Lemma 6.** *Let $n \geqslant 1$. Then there exists a 2DFA $B_n$ with $n$ states that may accept in the middle of a string using a set $F \subseteq Q \times \Sigma$ of accepting pairs, such that every NFA that recognizes the language $L(B_n)$ must have at least $N = \binom{2n+1}{n+1} - 1$ states.*

*Proof.* The desired automaton is defined as $B_n = (\Sigma, Q, q_0, F, \delta)$, where

- the set of states is $Q = \{1, 2, \ldots, n\}$,
- the alphabet $\Sigma = \{\, f \mid f\colon Q \to (Q \times \{+1, -1\}) \cup \{\text{accept}, \text{reject}\}\,\}$ is the set of all possible transition functions and acceptance conditions on a single symbol,
- the initial state is $q_0 = 1$,
- the set of acceptance conditions $F$ consists of all pairs $(q, f)$, with $q \in Q$ and $f \in \Sigma$, such that $f(q) = \text{accept}$,
- the transition function sets one transition $\delta(1, \vdash) = (1, +1)$ on the end-markers, and all transitions $\delta(q, f) = f(q)$, for $q \in Q$ and $f \in \Sigma$ with $f(q) \in Q \times \{+1, -1\}$.

The proof is by constructing a fooling set of size $N = \binom{2n+1}{n+1}-1$ for the language $L(B_n)$. Let $X = \{\, (P, R) \mid P, R \subseteq Q,\ (P, R) \neq (Q, Q), \text{ and either } |R| = |P| + 1, \text{ or } |R| = |P|\,\}$. There are exactly $N$ pairs of subsets $(P, R)$ in the set $X$ (as shown in the beginning of the proof of Theorem 6.

Let $(P, R)$ be a pair of subsets in $X$, and let $P = \{p_1, \ldots, p_s\}$ and $R = \{r_1, \ldots, r_t\}$, where $p_1 < p_2 < \ldots < p_s$ and $r_1 < r_2 < \ldots < r_t$, and $t = s + 1$ or $t = s$. For each pair $(P, R)$, the corresponding strings $u_{P,R}$ and $v_{P,R}$ are constructed as follows.

Let $i = \min(Q \setminus P)$; since $s \neq n$ due to $(P, R) \neq (Q, Q)$, such a state $i$ exists. Define $u_{P,R} = fg$. Here $f(1) = (i, +1)$, and $f$ rejects in all other states. The function

$g$ maps the states $i, p_1, \ldots, p_{t-1}$ forward to the states $r_1, \ldots, r_t$, that is, $g(i) = (r_1, +1)$, $g(p_1) = (r_2, +1)$, $\ldots$, $g(p_{t-1}) = (r_t, +1)$. If $s = t > 0$, then there is one extra state left in $P$, which is mapped to acceptance: $g(p_s) = \text{accept}$. If $s = t = 0$, then $g(i) = \text{accept}$. The function $g$ rejects in all other states.

Define the matching string as $v_{P,R} = h$, where $h(r_i) = (p_i, -1)$ for each $i = 1, \ldots, s$. If $t = s + 1$, then $h(r_t) = \text{accept}$. In all other states, $h$ rejects.

It remains to prove that $\{ (u_{P,R}, v_{P,R}) \mid (P, R) \in X \}$ is a fooling set.

- For each pair $(P, R) \in X$, the string $u_{P,R}v_{P,R}$ is in $L(B_n)$.

  Indeed, consider the computation of the 2DFA $B_n$ on this string. Let $P = \{p_1, \ldots, p_s\}$ and $R = \{r_1, \ldots, r_t\}$, where $p_1 < p_2 < \ldots < p_s$ and $r_1 < r_2 < \ldots < r_t$. Let $u_{P,R}v_{P,R} = fgh$. The automaton starts at the left end-marker $\vdash$ in the state 1, and moves in the state 1 to the first symbol $f$. From there it moves to the second symbol $g$ in the state $i = \min(Q \setminus P)$. If $s = t = 0$, then the automaton accepts here. Otherwise, it moves forward to $h$ in the state $r_1$. Then, according to the transition functions by the symbols $g$ and $h$, the automaton moves back and forth between these two symbols, assuming states $p_1, r_2, p_2, r_3, p_3, \ldots$. If $s = t$, then the automaton finally accepts at the symbol $g$ in the state $p_s$, and otherwise the automaton accepts at $h$ in the state $r_t$.

- Let $(P, R) \neq (P', R')$ be two different pairs in $X$. It is claimed that the automaton $B_n$ rejects at least one of the strings $u_{P,R}v_{P',R'}$ and $u_{P',R'}v_{P,R}$.

  Consider the computations of the automaton $B_n$ on these strings. Let $P = \{p_1, \ldots, p_s\}$, $R = \{r_1, \ldots, r_t\}$, $P' = \{p'_1, \ldots, p'_{s'}\}$ and $R' = \{r'_1, \ldots, r'_{t'}\}$, where $p_1 < p_2 < \ldots < p_s$, $r_1 < r_2 < \ldots < r_t$, $p'_1 < p'_2 < \ldots < p'_{s'}$ and $r'_1 < r'_2 < \ldots < r'_{t'}$. Let $u_{P,R}v_{P,R} = fgh$ and $u_{P',R'}v_{P',R'} = f'g'h'$. Consider the earliest difference between the computations of $B_n$ on the strings $fgh$ and $f'g'h'$.

  1. The first case is when there exists a number $k$, with $0 \leqslant k \leqslant \min\{s, s'\}$, such that $r_j = r'_j$ and $p_j = p'_j$ for all $j = 1, \ldots, k$, and either $r_{k+1} \neq r'_{k+1}$, or one of $r_{k+1}$ and $r'_{k+1}$ does not exist.

     Assume, without loss of generality, that either $r_{k+1} < r'_{k+1}$, or $r'_{k+1}$ does not exist. Consider the computation of the automaton $B_n$ on the string $u_{P,R}v_{P',R'} = fgh'$. Until the automaton reaches the state $p_k$, it works on $fgh'$ as on $fgh$ or $f'g'h'$. Eventually it comes to $g$ in the last common state $p_k$ (if $k = 0$, then let $p_0 = i = \min(Q \setminus P)$). Since $r_{k+1}$ exists, by the definition of $g$, the automaton moves to $h'$ in the state $r_{k+1} > r_k = r'_k$. If $r'_{k+1}$ does not exist, then the automaton rejects since it has come to a state with a larger number than any states with transitions on $h'$ defined. If $r'_{k+1}$ exists, then $r'_k = r_k < r_{k+1} < r'_{k+1}$, and the state $r_{k+1}$ is enclosed between two neighbouring states with a transition defined, and the automaton rejects.

  2. The second case is when there is a number $k$, with $1 \leqslant k \leqslant \min\{t, t'\}$, such that $p_j = p'_j$ for each $j = 1, \ldots, k-1$, and $r_j = r'_j$ for each $j = 1, \ldots, k$, and either $p_k \neq p'_k$, or one of $p_k$ and $p'_k$ does not exist.

     Without loss of generality, assume that either $p_k > p'_k$, or $p_k$ does not exist. Consider the computation of $B_n$ on the string $u_{P,R}v_{P',R'} = fgh'$. Eventually the automaton arrives at $h'$ in the last common state $r_k = r'_k$. Then it steps

backwards in the state $p'_k$. If the state $p_k$ does not exist, then the automaton comes to $g$ in a state $p'_k > p'_{k-1} = p_{k-1}$, that is, in a state not in $P$. This means that it either rejects, or, if $p'_k = \min(Q \setminus P)$, moves forward in $r_1$ and loops. If $p_k$ exists, then $p_{k-1} = p'_{k-1} < p'_k < p_k$. The automaton is thus at $g$ in a state not in $P$, and hence either rejects or loops.

Therefore, $\{ (u_{P,R}, v_{P,R}) \mid (P,R) \in X \}$ is a fooling set, and every NFA recognizing the language $L(B_n)$ has at least $N = \binom{2n+1}{n+1} - 1$ states. □

Now a precise lower bound on the state complexity of the DTWA→NTA transformation is established by adapting the string languages in Lemma 6 to the formalism of tree signatures. The left end-marker becomes the root of a tree, the right end-marker becomes a leaf, and all symbols of the alphabet become node labels of rank 1. Then 2DFA accepting in the middle are equivalent to DTWA over this signature, whereas NTA are equivalent to NFA operating on strings with end-markers. But NFA with end-markers are easily seen to be equivalent to NFA without end-markers, and so applying Lemma 6 gives the next theorem.

**Theorem 7.** *For every $n \geqslant 1$ there exists a signature $S_n$ and a deterministic tree-walking automaton $A_n$ over it, such that every nondeterministic tree automaton $A'$ recognizing the same set of trees as $A_n$ has at least $N = \binom{2n+1}{n+1} - 1$ states.*

## 3.4 Upper bound on the size of the smallest tree accepted by a nondeterministic tree-walking automaton

The upper bound on the size of minimal accepted tree for tree automata obtained in Section 3.2 can be combined with the upper bound for the transformation of tree-walking automata to tree automata from Section 3.3 to prove an upper bound on the size of minimal accepted tree for tree-walking automata.

**Theorem 8.** *Let $S = (\Sigma, \mathrm{rank}, \mathrm{up}, D, (D_a)_{a\in\Sigma})$ be a tree signature, and let $r = \max\{ \mathrm{rank}\, a \mid a \in \Sigma \}$ be the maximum number of children in a node. Let $A = (Q, Q_0, F, \delta)$ be a nondeterministic tree-walking automaton (NTWA) over $S$ with $|Q| = n$ states that accepts at least one tree. Then the minimal tree accepted by this automaton has at most $2^{r4^n}$ nodes.*

*Proof.* First, a nondeterministic tree automaton $A' = (Q', \delta')$ with $|Q'| = \binom{2n+1}{n+1} - 1$ states that recognizes the same language $L(A)$ is constructed by Theorem 6. The number of states $|Q'|$ is estimated as follows.

$$|Q'| = \binom{2n+1}{n+1} - 1 \leqslant \binom{2n+1}{n} \leqslant \sum_{k=0}^{n} \binom{2n+1}{k} = \frac{1}{2} \sum_{k=0}^{2n+1} \binom{2n+1}{k} = \frac{1}{2} 2^{2n+1} = 4^n.$$

Then, by Theorem 4, the minimal tree accepted by $A'$, and hence by $A$, has at most $2^{r|Q'|} \leqslant 2^{r4^n}$ nodes. □

The constant 4 in the upper bound $2^{r4^n}$ on the size of minimal trees accepted by NTWA is the same constant 4 as in the upper bound $O(\frac{1}{\sqrt{n}}4^n)$ on the length of shortest strings accepted by 2DFA (which comes from the optimal 2DFA to NFA transformation by Kapoutsis [15]). The lower bound on the length of the shortest strings have been improved several times, with the current bound being $\Omega(2^n)$. The constant 4 in the upper bound has not yet been improved.

In the case of strings, the bound $O(\frac{1}{\sqrt{n}}4^n)$ is the number of different pairs of sets of states $(P, R)$, with $|R| = |P| + 1$, in which an $n$-state automaton moves back and forth over an edge. In a minimal accepted string, such pairs of sets on edges never occur twice, for otherwise the substring between two equal pairs can be cut out of the string, and the string will still be accepted. Let pairs $(P, R)$ will $|R| \geqslant \frac{n}{2} + 1$ be called *heavy pairs*, and accordingly edges in the string, over which the automaton moves forward at least $\frac{n}{2} + 1$ times are called *heavy edges*. The rest of the pairs $(P, R)$ are *light pairs*, and the rest of the edges are *light edges*. Then, there cannot be two heavy edges in a row in an accepted string, because if two subsequent pairs $(P_1, R_1)$ and $(P_2, R_2)$ are both heavy, then the automaton comes to the node between the two edges $|R_1| + |P_2|$ times, but $|R_1| + |P_2| \geqslant \frac{n}{2} + 1 + \frac{n}{2} > n$, and hence the automaton comes to the same node more than $n$ times, and loops. However, the number of heavy pairs does not exceed the number of light pairs, and all pairs $(P, R) \in 2^Q \times 2^Q$ with $|R| = |P| + 1$ can be arranged in order so that no two heavy pairs are next to each other. Therefore, this division into light and heavy pairs would not improve the upper bound $O(\frac{1}{\sqrt{n}}4^n)$ on the length of the shortest accepted string for 2DFA.

What happens in the case of trees? At least three edges meet in each node with branching. Now an edge is said to be *heavy* if the automaton with $n$ states moves upward by this edge at least $\frac{n}{3} + 1$ times ($|P| \geqslant \frac{n}{3} + 1$). The rest of the edges are *light*. Then there are at most two heavy edges in each node, and each node with branching has at least one adjacent light edge. The overall number of such light pairs $(P, R)$ is no longer half of all pairs, but substantially lower. Then the idea of dividing into light and heavy pairs will be used to improve constant 4 in the case of trees.

**Theorem 9.** *Let $S = (\Sigma, \mathrm{rank}, \mathrm{up}, D, (D_a)_{a\in\Sigma})$ be a tree signature, and let $r$ be the maximum number of children in a node. Let $A = (Q, Q_0, F, \delta)$ be a nondeterministic tree-walking automaton (NTWA) over $S$, with $|Q| = n$ states, which accepts at least one tree. Then the minimal accepted tree has at most $2^{O(rn\cdot 3.572^n)}$ nodes.*

*Proof.* First a nondeterministic tree automaton $A' = (Q', \delta')$ that recognizes the same language as $A$ is constructed by Theorem 6. The goal is to give an upper bound on the number of nodes in the minimal tree accepted by $A'$.

The proof uses some details in the construction of the automaton $A'$ from the proof of Theorem 6. The states of $A'$ are pairs of subsets of the set of states of $A$.

$$Q' = \{\, (P, R) \mid P, R \subseteq Q,\ (P, R) \neq (Q, Q),\ \text{and either } |R| = |P| + 1, \text{ or } |R| = |P| \,\}$$

Let a node $v$ in some accepted tree have $k$ children, and let some computation of the automaton $A'$ on this tree have state $(P, R)$ at the node $v$, and states $(P_1, R_1), \ldots, (P_k, R_k)$ at the children of $v$. Then, by the construction of function $\delta'$ in the proof of Theorem 6, the sets $R, P_1, \ldots, P_k$ are pairwise disjoint. This means that there cannot be many large sets among these $k+1$ sets. The pairs in $Q'$ are divided into *heavy* and *light* by the number

of states in their first components: a pair $(P, R) \in Q'$ is called *heavy*, if $|P| \geqslant \frac{n}{3} + 1$, and *light* otherwise. Then there cannot be more than two heavy pairs among the states at a node $v$ and at its children.

For a nondeterministic tree automaton $A'$ of this particular form, in which states are divided into light and heavy pairs, and in which no node with its children may have more than two heavy pairs, one can obtain a better upper bound on the size of the minimal accepted tree than for NTA of a general form. For this purpose, the proof of Theorem 4 shall be elaborated using the additional constraints on the tree automaton.

Let $T = (V, v_0, +, \lambda)$ be a minimal tree accepted by the automaton $A'$, and let $(q(v))_{v \in V}$ be a computation of $A'$ on it. As shown in the proof of Theorem 4, for each internal node $v$, with state $q = q(v)$ and upward direction $-i = \operatorname{up} \lambda(v)$, the subtree rooted at $v$ is minimal in the number of nodes among all possible subtrees with upward direction $-i$ and with a computation that results in the state $q$ in the root. Hence, the tree $T$ and the computation on it can be modified so that for each pair $(q, i)$, all nodes $v$ with the state $q = q(v)$ and upward direction $-i$ have identical subtrees with the same computations on them (such a replacement preserves the validity of the computation and the minimality of the tree). The resulting tree and computation are fixed until the end of the proof. For each state $q$ and upward direction $-i$, denote the subtree used in all nodes with $q$ and $-i$ by $T_{q,i}$ (it is undefined, if $T$ has no such subtrees).

Next, all pairs $(q, i)$, with $q \in Q'$, $i \in \{1, \ldots, r\}$, for which the subtree $T_{q,i}$ is defined, are arranged in the increasing order of the number of nodes in $T_{q,i}$, and are enumerated from $(q_0, i_0)$ to $(q_{k-1}, i_{k-1})$, where $k < r \cdot |Q'|$. Let $m_j$ be the number of nodes in $T_{q_j, i_j}$, for $j \in \{0, \ldots, k-1\}$. Then $m_0 \leqslant m_1 \leqslant \ldots \leqslant m_{k-1}$.

Indices $j \in \{0, \ldots, k-1\}$ corresponding to pairs $(q_j, i_j) \in Q' \times \{1, \ldots, r\}$, in which $q_j \in Q'$ is a light pair, will be called *light indices*, and the rest of them, *heavy indices*.

The number of nodes in the entire tree $T$ is a sum of the sizes of the attached subtrees, plus one for the root. Thus, one more index $k$ representing $T$ is added to the sequence, and it is light, because the state in the root is $(\varnothing, \varnothing)$. The goal is to prove an upper bound on the number $m_k$ of nodes in $T$.

Let $j \in \{0, \ldots, k-1\}$ be an arbitrary index. It has a corresponding pair $(q_j, i_j)$ and a subtree $T_{q_j, i_j}$. Let $v$ be the root of $T_{q_j, i_j}$. There are exactly $m_j$ nodes in the subtree $T_{q_j, i_j}$, and this number is represented as $m_j = 1 + m_{j_1} + \ldots + m_{j_t}$, where $t$ is the number of children of $v$, and $j_1$, …, $j_t$ are indices of pairs corresponding to these children. These indices are pairwise distinct, because the children of $v$ have different directions up. Each $s$-th subtree $T_{q_{j_s}, i_{j_s}}$ has fewer nodes than $T_{q_j, i_j}$, and hence the subtrees of the children of $v$ correspond to earlier indices than $j$. Furthermore, since the computation in the entire tree has state $q_j$ in the node $v$, and states $q_{j_1}$, …, $q_{j_t}$ it its children, there are at most two heavy indices among $j, j_1, \ldots, j_t$.

Then, for every $j = 0, \ldots, k$, if the index $j$ is light, then $m_j$ is bounded by the sum of $m_t$ for all light indices $t < j$, plus $m_{j'}$ for the two last heavy indices before $j$, and plus one for the root of this subtree. If the index $j$ is heavy, then $m_j$ is not greater that the sum of number for all previous light indices, plus the number for the last heavy index before $j$, plus one for the root. For every index $j \in \{0, \ldots, k\}$, let $a_j$ and $b_j$ be the last and the second last heavy indices before $j$. If there are fewer than two heavy indices before $j$, then $a_j$ or $b_j$ may not exist; in this case it is assumed that $m_{a_j}$ or $m_{b_j}$ are equal to 0.

Then the sequence $m_0, \ldots, m_k$ is elementwise bounded by another sequence $x_0, \ldots, x_k$, constructed as follows.

- $x_0 = 1$,
- if $j$ is light, then $x_j = 1 + x_{a_j} + x_{b_j} + \sum_{t<j,\ t \text{ is light}} x_t$,
- if $j$ is heavy, then $x_j = 1 + x_{a_j} + \sum_{t<j,\ t \text{ is light}} x_t$.

If $a_j$ or $b_j$ are undefined, then $x_{a_j}$ or $x_{b_j}$ are equal to zero.

It is sufficient to give an upper bound on the number $x_k$. To this end, several inequalities bounding the growth of the sequence $(x_j)_{j=0,\ldots,k}$ will be proved.

**Claim 4.** *Let $j \in \{1, \ldots, k\}$ be an index. Then $x_j \leqslant 2x_{j-1}$.*

*Proof.* There are two cases.

- The index $j-1$ is light.

  Then $a_j = a_{j-1}$ and $b_j = b_{j-1}$, and regardless of whether $j$ is light or heavy, the following inequality holds.

  $$x_j \leqslant 1 + x_{a_j} + x_{b_j} + \sum_{t<j,\ t \text{ is light}} x_t = 1 + x_{a_{j-1}} + x_{b_{j-1}} + x_{j-1} + \sum_{t<j-1,\ t \text{ is light}} x_t = 2x_{j-1}$$

- The index $j-1$ is heavy.

  In this case, $a_j = j-1$ and $b_j = a_{j-1}$, and the inequality holds.

  $$x_j \leqslant 1 + x_{a_j} + x_{b_j} + \sum_{t<j,\ t \text{ is light}} x_t = 1 + x_{j-1} + x_{a_{j-1}} + \sum_{t<j-1,\ t \text{ is light}} x_t = 2x_{j-1}.$$

□

Thus, the elements of the sequence are at most doubled at each step. But when a chain of subsequent heavy indices occurs in the sequence $(x_j)_{j\in\{0,\ldots,k\}}$, the growth will be proved to be substantially slower.

**Claim 5.** *If indices $i, i+1, \ldots, j \in \{0, \ldots, k\}$, with $i < j$, are all heavy, then $x_j \leqslant (j-i+1)x_i$.*

*Proof.* Note that $\sum_{t<j,\ t \text{ is light}} x_t = \sum_{t<i,\ t \text{ is light}} x_t$. At the same time, $a_j = j-1$, $a_{j-1} = j-2$, $\ldots$, $a_{i+1} = i$. Then $x_j$ can be bounded as follows.

$$x_j = x_{j-1} + 1 + \sum_{t<i,\ t \text{ is light}} x_t = x_{j-2} + 2(1 + \sum_{t<i,\ t \text{ is light}} x_t) = \ldots =$$
$$= x_i + (j-i)(1 + \sum_{t<i,\ t \text{ is light}} x_t) \leqslant (j-i+1)x_i$$

□

According to Claims 4 and 5, the sequence $x_0, \ldots, x_k$ is elementwise bounded by yet another sequence $y_0, \ldots, y_k$, defined as follows.

- $y_0 = 1$,
- if $j$ is light or $j-1$ is light, then $y_j = 2y_{j-1}$,

- if $j$ and $j-1$ are both heavy, then $y_j = (j-i+1)y_i$, where $i$ is the least index, such that all indices from $i$ to $j$ are heavy.

In order to prove an upper bound on $y_k$, let $c_j = \frac{y_j}{y_{j-1}}$, for all $j = 1, \ldots, k$. Then $y_k = \prod_{j=1}^{k} c_j$. The coefficient $c_j$ is 2 for light indices $j$ and for heavy $j$ after light. For a heavy index $j \in \{1, \ldots, k\}$ preceded by exactly $t > 0$ heavy indices, $c_j = \frac{t+1}{t}$. In each chain of heavy indices, the coefficients $c_j$ do not increase, and therefore $y_k$ may only increase if a heavy index is moved from a longer chain to a shorter one. Then, the more there are sequences of heavy indices, and the less is the difference between their lengths, the greater will be $y_k$. In addition, the coefficient at every light index is no less than at any heavy index, and the more light indices are there in the sequence, the greater can be $y_k$.

Let there be exactly $\ell$ light indices, including $k$. Then there are at most $\ell$ chains of heavy indices. The number of heavy indices is bounded by the length of the sequence $k$, which does not exceed $r|Q'| \leqslant r4^n$. The value $y_k$ is the greatest if the heavy indices are distributed between the $\ell$ chains as evenly as possible. Each sequence of heavy indices of length $t$ (unless it starts at index 0) contributes a factor $2\frac{2}{1}\frac{3}{2}\ldots\frac{t}{t-1} = 2t$ to $y_k$; if it starts it 0, then it contributes $\frac{2}{1}\frac{3}{2}\ldots\frac{t}{t-1} = t$.

Then $y_k$ can be bounded as follows.

$$y_k = \prod_{j=1}^{k} c_j = \Big(\prod_{j \text{ is light}} c_j\Big)\Big(\prod_{j \text{ is heavy}} c_j\Big) \leqslant 2^\ell \left(2\left\lceil\frac{r4^n}{\ell}\right\rceil\right)^\ell$$

The first case to consider is when $\ell$ is less than $r$. Then $\ell = \frac{r}{c}$ for some number $c > 1$, and $y_k$ is bounded like this.

$$y_k \leqslant 2^\ell \left(2\left\lceil\frac{r4^n}{\ell}\right\rceil\right)^\ell \leqslant \left(4\left\lceil\frac{r4^n}{\ell}\right\rceil\right)^\ell \leqslant \left(4\frac{r4^n}{\ell} + 4\right)^\ell \leqslant \left(c4^{n+1} + 4\right)^{\frac{r}{c}} \leqslant$$
$$\leqslant \left(c4^{n+2}\right)^{\frac{r}{c}} = 4^{(n+2+\log_4 c)\frac{r}{c}} \leqslant 4^{O(nr)}$$

The theorem holds in this case.

Let $\ell \geqslant r$. Then there is a following bound on $y_k$.

$$y_k \leqslant 2^\ell \left(2\left\lceil\frac{r4^n}{\ell}\right\rceil\right)^\ell \leqslant 2^\ell (2 \cdot 4^n)^\ell \leqslant (4^{n+1})^\ell$$

It remains to bound the number of light indices, and to substitute the result for $\ell$. Light indices correspond to pairs $((P, R), i)$, where $i \in \{1, \ldots, r\}$, and $P$ and $R$ are subsets of $Q$, such that $|R| = |P| + 1$ or $|R| = |P|$, with $|P| \leqslant \lceil\frac{n}{3}\rceil$. Then the number of light indices is not greater than the product of the number of directions by the number of ways to choose $P$ and then $R$. The light index for the root that concludes the sequence has a corresponding pair $(P, R) = (\varnothing, \varnothing)$, which cannot occur inside the tree.

$$\ell \leqslant r \cdot \sum_{t=0}^{\lceil\frac{n}{3}\rceil} \binom{n}{t}\left(\binom{n}{t} + \binom{n}{t+1}\right) \leqslant r \cdot \left(\left\lceil\frac{n}{3}\right\rceil + 1\right)\binom{n}{\lceil\frac{n}{3}\rceil}\binom{n+1}{\lceil\frac{n}{3}\rceil + 1}$$

By Stirling's approximation,

$$\binom{3m}{m} = O\left(\frac{1}{\sqrt{m}}\frac{(3m)^{3m}}{(2m)^{2m}m^m}\right) = O\left(\frac{1}{\sqrt{m}}\left(\frac{27}{4}\right)^m\right)$$

Then the number of light indices is bounded by

$$\ell \leqslant r \cdot \left(\left\lceil \frac{n}{3} \right\rceil + 1\right)\binom{n}{\lceil \frac{n}{3} \rceil}\binom{n+1}{\lceil \frac{n}{3} \rceil + 1} \leqslant O\left(rn\left(\frac{1}{\sqrt{n}} \cdot \left(\frac{27}{4}\right)^{\frac{n}{3}}\right)^2\right) = \\ = O\left(r\left(\frac{9}{\sqrt[3]{16}}\right)^n\right) = O(r \cdot 3.572^n)$$

Therefore, the minimal number of nodes in an accepted tree is at most

$$y_k \leqslant 4^{(n+1)O(r \cdot 3.572^n)} \leqslant 2^{O(rn \cdot 3.572^n)}.$$

□

## 3.5 Lower bound on the size of the smallest tree accepted by a tree-walking automaton

In Section 3.4, an upper bound $2^{O(rn \cdot 3.572^n)}$ on the number of nodes in a minimal tree accepted by a nondeterministic tree-walking automaton was obtained, where $n$ is the number of states, and $r$ is the maximum number of children in a node. The number of nodes in this bound is double exponential in the number of states and exponential in the maximum degree of a node. Can there truly be that many nodes in a minimal accepted tree? The goal of this section is to obtain a lower bound of the form $2^{\Omega(r \cdot c^n)}$, where $c > 1$ is a constant and $r \geqslant 2$. The proof is by constructing a deterministic tree-walking automaton with a minimal accepted tree of this size.

The case of binary trees ($r = 2$) will be considered first. The construction will use two-way finite automata with long shortest accepted strings: an $n$-state 2DFA with shortest accepted strings of length $\frac{3}{4}2^n - 1$ was constructed in Chapter 2, whereas the upper bound is $O(4^n)$.

Unlike strings, trees may branch, and the number of nodes in a tree can be exponential in its height. Can another exponent be added to this bound in the case of binary trees? One can take a 2DFA with $n$ states over an alphabet $\Sigma$ with exponentially long shortest accepted string, and try to construct a deterministic tree-walking automaton with $O(n)$ states that works on binary trees with symbols from $\Sigma$ in its nodes, and checks whether the string on each path from root to leaf is accepted by the 2DFA. Then, in each accepted tree, the depth of every leaf will be exponential in $n$, and the number of nodes in the minimal accepted tree will be double exponential in $n$.

However, not every 2DFA with a long shortest accepted string is suitable for such a construction. A tree-walking automaton will need to continue simulating a computation from the same moment in all paths in the current subtree. To do this, the automaton, after simulating the computation on one branch, will have to roll it back to the branching point before continuing the simulation in another subtree. There is also an issue with the original 2DFA moving far backwards: as the simulating tree-walking automaton goes far up, it will forget from which node it has ascended, as there are many nodes at the same height. Hence, the tree-walking automaton will forget, on which path it currently simulates the 2DFA.

It turns out that there is a suitable 2DFA, which never makes two step backwards in a row, and whose computation can be effectively rolled back in order to continue

the simulation on the next branch. The length of the shortest accepted string for this automaton is only $\sqrt{n}$ times less than the maximum known. This is the 2DFA from Chapter 2 that gives a precise lower bound on the maximum length of a shortest accepted string in the class of automata that remember the direction of the last move in their state.

**Theorem 10** (more precise statement of Theorem 2)**.** *For every $k \geqslant 2$ and $\ell \geqslant 0$ there exists a direction-determinate 2DFA $A_{k,\ell} = (\Sigma, Q, q_0, \delta, F)$ with the set of states $Q = Q^+ \cup Q^-$, where $|Q^+| = k$ and $|Q^-| = \ell$, such that the length of the shortest string it accepts is $\binom{k+\ell}{\ell+1} - 1$, and with the following additional properties:*

- *the initial state $q_0$ is in $Q^+$, the accepting state $q_{acc}$ is unique and is in $Q^+$ as well,*
- *all transitions in states in $Q^-$ lead to states in $Q^+$,*
- *for every symbol $a \in \Sigma$, there exists a unique state $q(a) \in Q^+$, such that the transition from this state by the symbol $a$ moves the head to the right. The unique transition to the right by the left end-marker is from the initial state. Let $q(\vdash) = q_0$ and $q(\dashv) = q_{acc}$.*

The above additional properties were not explicitly stated in Theorem 2, but the actual automata constructed therein have these properties.

In the rest of this section, let $A_{k,\ell}$ be the 2DFA constructed in Theorem 10.

In the next theorem, a tree-walking automaton is constructed based on 2DFA $A_{k,\ell}$ that, given a binary tree, checks, for each path from root to leaf, that the string written along this path is accepted by $A_{k,\ell}$.

**Theorem 11.** *Let $k \geqslant 2$ and $\ell \geqslant 0$, and let the 2DFA $A_{k,\ell} = (\Sigma, Q, q_0, \delta, F)$ with the set of states $Q = Q^+ \cup Q^-$, where $|Q^+| = k$ and $|Q^-| = \ell$, be as in Theorem 10. Let $S = (\Sigma', \mathrm{rank}, \mathrm{up}, D, (D_a)_{a\in\Sigma'})$ be a binary tree signature with node labels $\Sigma' = \{\vdash_1\} \cup \{\, a_i \mid a \in \Sigma \cup \{\dashv\}, i \in \{1,2\}\,\}$. Here $\vdash_1$ is the root label, $\dashv_1$ and $\dashv_2$ are labels for left and right leaves, and $\{\, a_1 \mid a \in \Sigma\,\}$ and $\{\, a_2 \mid a \in \Sigma\,\}$ are labels for left and right children, with symbols from $\Sigma$. Each node except the leaves must have two children.*

*Then there exists a deterministic tree-walking automaton $B$ with $k + 2\ell + 2$ states that accepts a tree if and only if, for each path from root to leaf, the string of symbols from $\Sigma$ written on this path is accepted by $A_{k,\ell}$.*

*Proof.* The tree-walking automaton $B = (Q', q_0', \delta', F')$ over the signature $S$ works as follows. It gets a tree $T$ over a signature $S$ as an input, and it should check that the automaton $A_{k,\ell}$ accepts all strings, written along all paths from root to leaf. To do this, the tree-walking automaton will come to all leaves one by one, from left to right, and verify that the string from the root to the current leaf is accepted by the 2DFA $A_{k,\ell}$.

The computation of the automaton $B$ constructed for the two-way automaton $A_{2,1}$ is illustrated in Figure 3.3. The tree in the figure has the shortest string accepted by $A_{2,1}$ written along each path.

The set of states of $B$ is $Q' = Q^+ \cup Q_1^- \cup Q_2^- \cup \{s_1, s_2\}$, where $Q_1^- = \{\, q_1 \mid q \in Q^-\,\}$ and $Q_2^- = \{\, q_2 \mid q \in Q^-\,\}$. Altogether there are $k + 2\ell + 2$ states in $Q'$.

The states $Q^+ \cup Q_1^- \cup Q_2^-$ will be used to simulate the automaton $A_{k,\ell}$ on the leftmost path from an arbitrary node. The transitions of $B$ on $Q^+ \cup Q_1^- \cup Q_2^-$ should be defined so that the next claim holds.

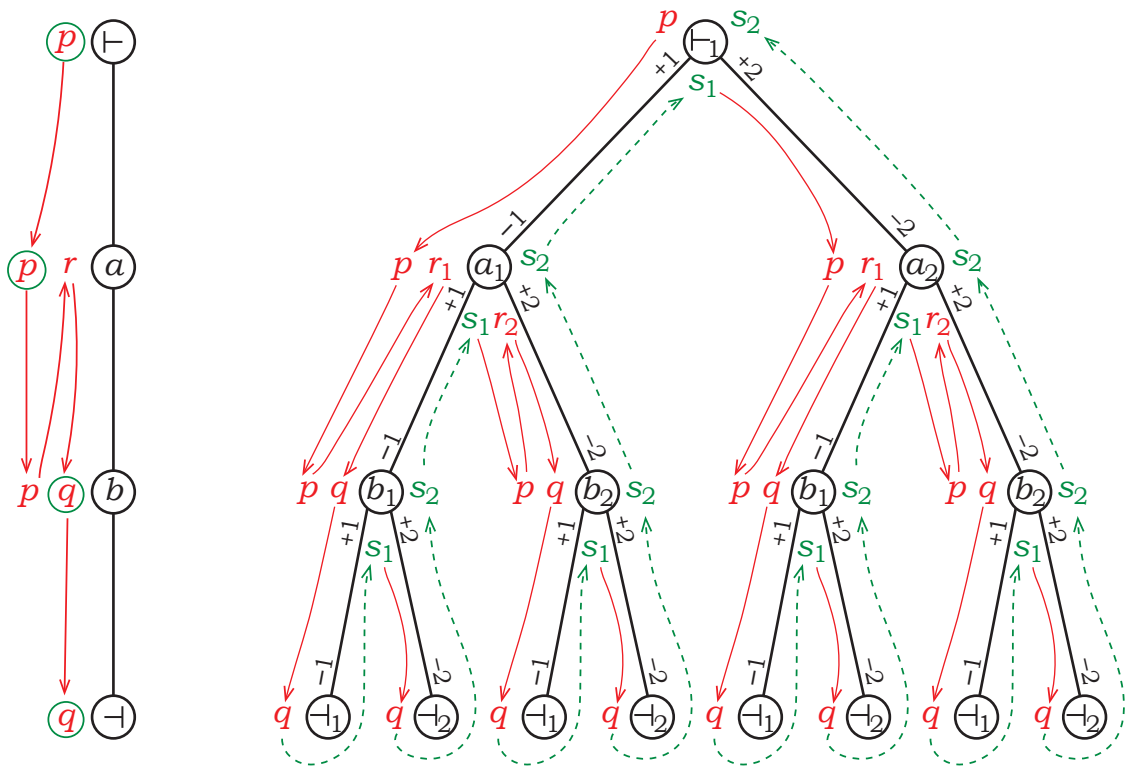


Figure 3.3: (left) 2DFA $A_{2,1}$ works on its shortest accepted string; (right) tree-walking automaton $B$ traverses a tree checking that the string on each path from root to leaf is accepted by $A_{2,1}$.

**Claim 6.** *Let $x$ be the leftmost leaf in the subtree of a node $v$, and let the automaton $A_{k,\ell}$ assume the state $q$ at the node $v$ in its computation on the string from the root to $x$. Then, if the automaton $B$ is started at the node $v$ in the state $q$ (if $q \in Q^+$) or in the state $q_1$ (if $q \in Q^-$), it completes the simulation of $A_{k,\ell}$ on the string on the path from root to $x$: that is, if the automaton $A_{k,\ell}$ rejects or loops on this string, then so does $B$, and if $A_{k,\ell}$ accepts this string, then $B$ comes to the leaf $x$ in the state $q_{acc}$.*

Let $x$ and $v$ be fixed. Since all transitions of $A_{k,\ell}$ in the states from $Q^-$ lead to states in $Q^+$, the automaton $A_{k,\ell}$ cannot make two steps backwards in a row, and hence its simulation along the path from root to $x$ may go up from the node $v$ at most by one edge. If the simulation never goes up from $v$, then the automaton $B$ can work as if on string using states from $Q^+$ and $Q_1^-$ to simulate $A_{k,\ell}$ in states from $Q^+$ and $Q^-$, always moving by left edges and ignoring right edges. All edges on the path from $v$ to $x$ are left (that is, in directions $\pm 1$). But sometimes the automaton $B$ will need to go from $v$ to its parent, and this edge may be right. In this case, the automaton $B$ needs to remember that it has to descend from the parent of $v$ to the right, not to the left as in all other cases. To do this, $B$, as it goes upwards, will remember in the choice of a state from $Q_1^-$ or from $Q_2^-$ whether it has come from the left or from the right child. In order for $B$ to work like this, the transitions used to simulate $A_{k,\ell}$ are defined as follows.

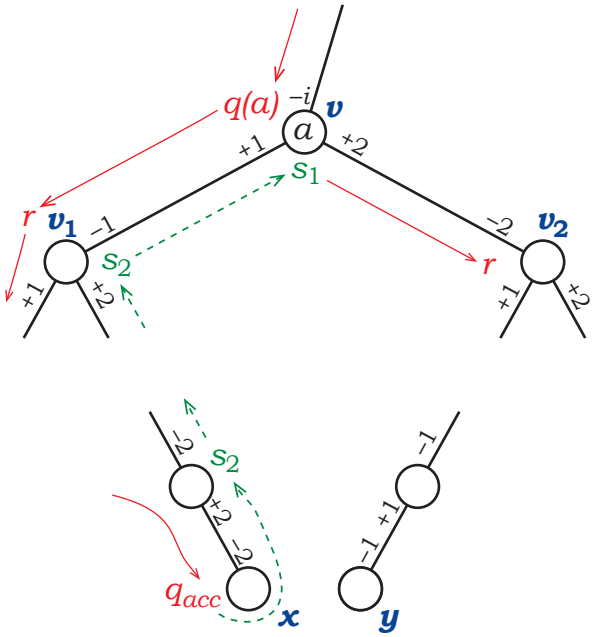


Figure 3.4: The automaton $B$ moves from the state $q_{acc}$ at the leaf $x$ to the computation of $A_{k,\ell}$ on the path from root to the next leaf $y$.

For $q \in Q^+$ and $a \in \Sigma \cup \{\vdash, \dashv\}$:

$$\begin{aligned} \delta'(q, a_j) &= (r, +1), && \text{if } \delta(q, a) = (r, +1), \text{ and } j \in \{1, 2\} \\ \delta'(q, a_j) &= (r_j, -j), && \text{if } \delta(q, a) = (r, -1), \text{ and } j \in \{1, 2\} \end{aligned}$$

For $q \in Q^-$ and $a \in \Sigma \cup \{\vdash\}$:

$$\delta'(q_j, a_i) = (r, +j), \qquad \text{if } \delta(q, a) = (r, +1), \text{ and } i, j \in \{1, 2\}$$

These transitions are enough for $B$ to work as in Claim 6.

Now all remaining transitions of $\delta'$ will be defined, so that the automaton $B$ checks that the strings on all paths from the root to all leaves are accepted by $A_{k,\ell}$.

The initial state of $B$ is $q_0 \in Q^+$. The automaton $B$ starts in this state at the root, and, by Claim 6, it simulates $A_{k,\ell}$ on the leftmost path in the tree. If the string on this path is not accepted by $A_{k,\ell}$, then $B$ rejects or loops. If the string is accepted, then $B$ comes to the leftmost leaf in the state $q_{acc}$.

Let $B$ come to some leaf $x$ in the state $q_{acc}$, as in Figure 3.4. Then it has already checked that the strings on the paths from the root to all leaves to the left of $x$ and to $x$ itself are accepted by $A_{k,\ell}$. Let $x$ be not the rightmost leaf in the whole tree. Then there is the next leaf $y$. Now the automaton $B$ needs to determine whether the string on the path from the root to $y$ is accepted, and if it does, then come to $y$ in the state $q_{acc}$. The automaton $B$ does this as follows. First it goes up, remembering whether the last step was from the left child or from the right child, and does so until it makes the first step up from a left child. To do this, $B$ uses states $s_1$ and $s_2$ and the following transitions.

$$\begin{aligned} \delta'(q_{acc}, \dashv_j) &= (s_j, -j), && \text{where } j \in \{1, 2\} \\ \delta'(s_2, a_j) &= (s_j, -j), && \text{where } a \in \Sigma,\ j \in \{1, 2\} \end{aligned}$$

As a result, the automaton makes zero or more steps up from a right child, and finally one step from the left child, and emerges at a certain node $v$ in the state $s_1$. Let $v_1$ and $v_2$ be the left and the right children of $v$. Then the leaf $x$ is on the rightmost path in the subtree of $v_1$, whereas the leaf $y$ is on the leftmost path in the subtree of $v_2$.

It is known that the computation on the path from the root to the leaf $x$ is accepting. Could one use this fact to continue this computation from the node $v$ as a computation from the root to the next leaf $y$? Let $a \in \Sigma \cup \{\vdash\}$ be the symbol in $v$. Consider the moment when the automaton $A_{k,\ell}$, simulated on the path from root to $x$, first goes down from $v$ to $v_1$. At this moment, $A_{k,\ell}$ makes a transition forward from a state in $Q^+$ by $a$. For each symbol there is only one transition forward from a state in $Q^+$. Hence, this transition can be made only in the state $q(a)$. The same moment exists in the computation on the path from root to $y$, because the strings to $x$ and to $y$ are the same up to $v$. Thus, in the computation on the string from root to $y$, the automaton $A_{k,\ell}$ comes to the node $v$ in the state $q(a)$, and the next step is descending to $v_2$ in such a state $r$ that $\delta(q(a), a) = (r, +1)$.

In order to turn to this simulation towards the leaf $y$, the automaton $B$, when it comes to $v$ in the state $s_1$ should continue by a transition down to $v_2$ in the state $r \in Q^+$. Then, since $y$ is the leftmost leaf in the subtree of $v_2$, by Claim 6, the automaton $B$ will complete the simulation of $A_{k,\ell}$ on the path from root to $y$. This turn to the next simulation is made by the following transition.

$$\delta'(s_1, a_j) = (r, +2), \text{ if } a_j \in \Sigma' \setminus \{\dashv_1, \dashv_2\}, \ \delta(q(a), a) = (r, +1)$$

Therefore, if the string on the path from root to $y$ is not accepted, then the automaton $B$ rejects or loops. If the string is accepted, then $B$ completes the simulation, arriving to the leaf $y$ in the state $q_{acc}$.

Thus the automaton will move from $y$ to the next leaf, etc., until it either encounters a string on path from root to leaf that is not accepted (in this case $B$ rejects or loops), or until it comes to the rightmost leaf in the state $q_{acc}$.

Let the strings on all paths be accepted. Then the automaton $B$ comes to the righmost leaf in the state $q_{acc}$. Next, the transition by $q_{acc}$ in the leaf leads it up in the state $s_2$, because the leaf is right. The automaton continues ascending from right children in the state $s_2$, until it comes to the root of the tree. If the automaton comes to the root in the state $s_2$, it means that it has already checked all leaves, and in this case $B$ accepts: $F' = \{(s_2, \vdash_1)\}$. □

How can one use Theorem 11 to get a big minimal accepted tree? If the number of states is $n = k + 2\ell + 2$, then, by Theorem 11, one can make an automaton that accepts a binary tree, if every path from root to leaf contains a string accepted by $A_{k,\ell}$, delimited by left and right end-markers. The shortest string accepted by $A_{k,\ell}$ is of length $\binom{k+\ell}{\ell+1} - 1$. With the end-markers attached, the length becomes $\binom{k+\ell}{\ell+1} + 1$. The minimal accepted tree, in which that shortest string is written on every path, is a balanced binary tree with $\binom{k+\ell}{\ell+1} + 1$ levels. At level 0 there is just the root, and every next level has twice as many nodes as the previous level. Overall, the number of nodes in the minimal accepted tree is

$$\sum_{j=0}^{\binom{k+\ell}{\ell+1}} 2^j = 2^{\binom{k+\ell}{\ell+1}+1} - 1.$$

For a given number of states $n$, one should find an optimal partition $n = k + 2\ell + 2$ that maximizes the binomial coefficient $\binom{k+\ell}{\ell+1} = \binom{n-\ell-2}{\ell+1}$.

**Claim 7.** *For each $n \geqslant 3$, the number $\binom{n-\ell-2}{\ell+1}$ is maximal for $\ell = \lfloor \frac{5n-8-\sqrt{5n^2+4}}{10} \rfloor$.*

*Proof.* Denote $C_\ell = \binom{n-\ell-2}{\ell+1}$ for all $\ell \in \{0, \ldots, \lfloor \frac{n-2}{2} \rfloor\}$ (larger values $\ell > \lfloor \frac{n-2}{2} \rfloor$ would give $n-\ell-2 < \ell+1$). If the binomial coefficient $C_\ell$ is maximal, then it should not be less than the neighbouring coefficients $C_{\ell-1}$ and $C_{\ell+1}$, if they exist. Consider the ratio of two subsequent binomial coefficients $\frac{C_\ell}{C_{\ell+1}}$, for all $\ell \in \{0, \ldots, \lfloor \frac{n-2}{2} \rfloor - 1\}$.

$$\frac{C_\ell}{C_{\ell+1}} = \frac{\binom{n-\ell-2}{\ell+1}}{\binom{n-\ell-3}{\ell+2}} = \frac{(n-\ell-2)!(\ell+2)!(n-2\ell-5)!}{(\ell+1)!(n-2\ell-3)!(n-\ell-3)!} = \frac{(\ell+2)(n-\ell-2)}{(n-2\ell-3)(n-2\ell-4)}$$

The equation $\frac{(\ell+2)(n-\ell-2)}{(n-2\ell-3)(n-2\ell-4)} = 1$ is a quadratic equation with unknown $\ell$ and with parameter $n$, its roots are $\ell_1 = \frac{5n-18-\sqrt{5n^2+4}}{10}$ and $\ell_2 = \frac{5n-18+\sqrt{5n^2+4}}{10}$. This equation defines a parabola that opens downwards, hence the ratio $\frac{C_\ell}{C_{\ell+1}}$ is greater than 1 between the roots of the equation, and less than 1 outside. The root $\ell_2$ is greater than $\lfloor \frac{n-2}{2} \rfloor - 1$. Hence, while $\ell$ is less than or equal to $\ell_1$, the ratio $\frac{C_\ell}{C_{\ell+1}}$ is at most 1, and it is profitable to increase $\ell$; after $\ell_1$, the value $C_\ell$ decreases. Therefore, the maximum is reached for $\ell = \lfloor \ell_1 \rfloor + 1 = \lfloor \frac{5n-8-\sqrt{5n^2+4}}{10} \rfloor$. This is a valid value for $\ell$, because $0 \leqslant \lfloor \frac{5n-8-\sqrt{5n^2+4}}{10} \rfloor \leqslant \lfloor \frac{n-2}{2} \rfloor$ for all $n \geqslant 3$. □

The next claim shows the growth rate of the binomial coefficient $\binom{n-\ell-2}{\ell+1}$ for the optimal value of $\ell$.

**Claim 8.** *The binomial coefficient $\binom{n-\ell-2}{\ell+1}$, where $\ell = \ell(n) = \lfloor \frac{5n-8-\sqrt{5n^2+4}}{10} \rfloor$, has asymptotics $\Theta(\frac{1}{\sqrt{n}} C^n)$, where $C = \left(\frac{c^c}{(c-1)^{c-1}}\right)^{\frac{1}{c+1}}$ and $c = \frac{10}{5-\sqrt{5}} - 1$. The stated value of $C$ is within the bounds $1.618 < C < 1.619$.*

*Proof.* First, both arguments of the binomial coefficient $\binom{n-\ell-2}{\ell+1}$ are approximated as $n$ multiplied by a constant.

$$n-\ell-2 = n - \Big\lfloor \frac{5n-8-\sqrt{5n^2+4}}{10} \Big\rfloor - 2 = n\Big(1 - \frac{5-\sqrt{5}}{10}\Big) + \Theta(1) = n \cdot \frac{5+\sqrt{5}}{10} + \Theta(1), \text{ and}$$

$$\ell + 1 = \Big\lfloor \frac{5n-8-\sqrt{5n^2+4}}{10} \Big\rfloor + 1 = n\Big(\frac{5-\sqrt{5}}{10}\Big) + \Theta(1).$$

Let $m = n \cdot \frac{5-\sqrt{5}}{10}$, then $\ell+1 = m + \Theta(1)$, and $n-\ell-2 = n \cdot \frac{5+\sqrt{5}}{10} + \Theta(1) = m \cdot \frac{5+\sqrt{5}}{5-\sqrt{5}} + \Theta(1) = m \cdot (\frac{10}{5-\sqrt{5}} - 1) + \Theta(1) = cm + \Theta(1)$. By Stirling's approximation,

$$\binom{n-\ell-2}{\ell+1} = \binom{cm+\Theta(1)}{m+\Theta(1)} = \Theta\Big(\binom{cm}{m}\Big) = \Theta\Big(\frac{1}{\sqrt{m}} \cdot \frac{(cm)^{cm}}{((c-1)m)^{(c-1)m} \cdot m^m}\Big) =$$
$$= \Theta\Big(\frac{1}{\sqrt{m}} \Big(\frac{c^c}{(c-1)^{c-1}}\Big)^m\Big) = \Theta\Big(\frac{1}{\sqrt{n}} \Big(\frac{c^c}{(c-1)^{c-1}}\Big)^{\frac{5-\sqrt{5}}{10} \cdot n}\Big) = \Theta\Big(\frac{1}{\sqrt{n}} C^n\Big).$$

□

Thus, for binary trees, the following lower bound on the maximum number of nodes in a minimal accepted tree is obtained.

**Theorem 12.** *For every $n \geqslant 1$, there exist a tree signature $S_n$ with all node labels of rank 0 or 2, and an $n$-state deterministic tree-walking automaton $B_n$ over $S_n$, such that the automaton accepts at least one tree, and the minimal accepted tree has at least $2^{\Omega(1.618^n)}$ nodes.*

Now let the maximum number of children in a node be $r \geqslant 4$. How can one use the extra directions to obtain a higher lower bound? This will be done inductively on even $r$, with binary trees as the base case. In the induction step, the directions $\{+1, +2\}$ and $\{+3, \ldots, +r\}$ are treated differently. Theorem 12 gives an automaton $B$ that traverses binary trees and has minimal accepted tree with at least $2^{c \cdot 1.618^n}$ leaves, where $c$ is some constant. The induction hypothesis gives an automaton $B_{r-2}$ that traverses trees with at most $r-2$ children in a node. Trees for $B_{r-2}$ are adapted for the signature of a new automaton $B_r$ by rewriting them in directions $\{+3, \ldots, +r\}$, with anything in directions $\{+1, +2\}$. Then, trees for $B_r$ are trees for $B$, with each leaf replaced with a tree for $B_{r-2}$ adapted as above.

The new automaton $B_r$ operating on such trees acts on the part for $B$ as $B$, moving only in directions $\{\pm 1, \pm 2\}$. When $B_r$ visits a leaf for $B$, it will traverse the attached subtree for $B_{r-2}$ by simulating $B_{r-2}$ and moving only in directions $\{\pm 3, \ldots, \pm r\}$. Then the automaton $B_r$ always knows whether it simulates $B$ on the top part of the tree or $B_{r-2}$ on one of the attached subtrees, by observing whether the direction upwards is in $\{\pm 1, \pm 2\}$ or not.

This allows the number of nodes in the minimal trees to be multiplied by $2^{c \cdot 1.618^n}$ in the transition from $B_{r-2}$ to $B_r$, and ultimately gives a lower bound $2^{\Omega(r \cdot 1.618^n)}$ for trees of rank at most $r$. This is formally proved in the theorem below.

**Theorem 13.** *For all $r \geqslant 2$ and $n \geqslant 1$, there exist a tree signature $S_{n,r}$ with maximum rank $r$ and an $n$-state deterministic tree-walking automaton $B_{n,r}$ over this signature, such that the automaton accepts at least one tree, and the minimal accepted tree has at least $2^{\Omega(r \cdot 1.618^n)}$ nodes.*

*Proof.* The number $r$ is assumed to be even (for odd $r$, the bound for $r-1$ can be used). Also assume that $n \geqslant 4$ (the case of small $n$ will be treated in the end of the proof).

By Theorem 12, there is a sequence of automata $B_{n,2}$ over signatures $S_{n,2}$, such that, for every $n$, the minimal tree accepted by $B_{n,2}$ has at least $2^{c \cdot 1.618^n}$ leaves, where $c$ is a constant. Such automata $B_{n,2}$ are constructed using Theorem 11, for $\ell = \left\lfloor \frac{5n-8-\sqrt{5n^2+4}}{10} \right\rfloor$ and $k = n - 2\ell - 2$. The following additional properties can be inferred from the proof of Theorem 11.

- All states of the automaton $B_{n,2}$ are split into two disjoint sets: states in $P^+$ are entered after moving downwards, and states in $P^-$ are entered after a move upwards.
- The automaton $B_{n,2}$ accepts only at the root.
- If the automaton accepts, then it visits each leaf of the tree in the state $q_{acc} \in P^+$.

Let $n$ be fixed. Automata $B_{n,r}$ with $n$ states are constructed inductively on even $r$, along with their signatures $S_{n,r}$. The states of each $B_{n,r}$ are split into the same sets $P^+$ and $P^-$, as in $B_{n,2}$, and the minimal tree accepted by $B_{n,r}$ must have at least $2^{\frac{r}{2} c \cdot 1.618^n}$ nodes, and $B_{n,r}$ must accept only in the root. The signature $S_{n,r}$ has a unique label for the root, which is not a leaf.

Base case: $r = 2$, the automaton $B_{n,2}$ has all the required properties.

Induction step: $r \to r + 2$. Let the automaton $B_{n,r}$ be constructed, the task is to construct $B_{n,r+2}$.

An intermediate automaton $C_{n,r}$ is constructed first: this is a copy of the automaton $B_{n,r}$, operating on directions $\{\pm 3, \ldots, \pm(r+2)\}$. For this, the signature $S_{n,r}$ is transformed into a new signature $S'_{n,r}$ by shifting it by two directions: two more children are added to each node label, and its direction upwards is increased by two; furthermore, all node labels from the signature $S_{n,2}$ are added to $S'_{n,r}$. Only labels from $S_{n,2}$ have upward directions $-1$ or $-2$. Now every tree over the signature $S'_{n,r}$ is either a tree over $S_{n,2}$, or a copy of a tree over $S_{n,r}$ written in directions $\{\pm 3, \ldots, \pm(r+2)\}$ instead of $\{\pm 1, \ldots, \pm r\}$, and with each direction $+1$ and $+2$ closed by some subtree over $S_{n,2}$.

At all node labels from $S_{n,2}$, the automaton $C_{n,r}$ rejects right away. The automaton $C_{n,r}$ implements the transitions of $B_{n,r}$ by moving in a direction $+(i+2)$ instead of $+i$, and in a direction $-(i+2)$ instead of $-i$, for all $i \in \{1, \ldots, r\}$. Then $C_{n,r}$ accepts only trees obtained from trees accepted by $B_{n,r}$ by rewriting them in directions $\{\pm 3, \ldots, \pm(r+2)\}$. The automaton $C_{n,r}$, like $B_{n,r}$, accepts only in the root; its states are partitioned into $P^+$ and $P^-$, as in $B_{n,r}$; and the minimal tree accepted by $C_{n,r}$ has at least $2^{\frac{r}{2}c \cdot 1.618^n}$ nodes.

The automaton $B_{n,r+2}$ is based on both the automaton $B_{n,2}$, which works in directions $\{\pm 1, \pm 2\}$, and the automaton $C_{n,r}$, working in directions $\{\pm 3, \ldots, \pm(r+2)\}$. The plan is that the automaton $B_{n,r+2}$ recognizes the trees accepted by $B_{n,2}$, in which every leaf is replaced with a tree accepted by $C_{n,r}$.

First, new node labels are added to the signature $S'_{n,r}$. These are all labels of the form $(a, b)$, where $a$ is a leaf label for $B_{n,2}$, and $b$ is a root label for $C_{n,r}$. Each label $(a, b)$ has upward direction up $a$ and the same rank as $b$. All new labels $(a, b)$ have upward direction $-1$ or $-2$, and hence they may not occur on paths from the root that use only directions $\{\pm 3, \ldots, \pm(r+2)\}$. Since the automaton $C_{n,r}$ never moves in directions $\{\pm 1, \pm 2\}$, it cannot reach the nodes with new labels, and the properties of $C_{n,r}$ remain unchanged: accepting at least one tree, acceptance only in the root, partition of states into $P^+$ and $P^-$, and having at least $2^{\frac{r}{2}c \cdot 1.618^n}$ nodes in the minimal accepted tree. The number of nodes in the minimal accepted tree will not decrease, because every new label can be replaced with a leaf without affecting the computation of $C_{n,r}$.

The signature $S_{n,r+2}$ is obtained from $S'_{n,r}$ by removing all root labels except the root for $B_{n,2}$.

Every tree over the signature $S_{n,r+2}$ has a root with a label from $S_{n,2}$, which has 2 children. All node labels in $S_{n,r+2}$ with upward directions $-1$ and $-2$ are either in the signature $S_{n,2}$ or are of the form $(a, b)$. Hence, on every path from the root in the directions $+1, +2$, the first label not in $S_{n,2}$ can only be a label of the form $(a, b)$, where $a$ is a leaf label for $B_{n,2}$, and $b$ is a root label for $C_{n,r}$. The subtree rooted at a node with label $(a, b)$, is always a tree over $S'_{n,r}$, in which its original root $b$ has been replaced with $(a, b)$. Then each tree over the signature $S_{n,r+2}$ is a tree over $S_{n,2}$, in which some leaves are replaced with trees over the signature $S'_{n,r}$.

The automaton $B_{n,r+2}$ is constructed as follows. First, $B_{n,2}$ is taken and augmented with transitions for each new node label $(a, b)$ as if for a leaf labelled with $a$, whereas the old transitions at the leaves are removed. So far, the automaton accepts exactly the trees accepted by $B_{n,2}$, in which each leaf is replaced by some tree over the signature $S'_{n,r}$ (which the automaton never descends into). The accepting computation of $B_{n,r+2}$ follows the accepting computation of $B_{n,2}$ on the upper part of the tree, and hence visits every

former leaf in the state $q_{acc}$, and finally accepts in the root. It remains to check that all trees that were substituted for leaves are accepted by $C_{n,r}$. To do this, the transition from $q_{acc}$ at labels of the form $(a, b)$ is changed: the automaton $B_{n,r+2}$ will make the same transition as $C_{n,r}$ would do at a root labelled with $b$. Then, the automaton $B_{n,r+2}$ simulates $C_{n,r}$, treating the node labelled with $(a, b)$ at the top of the subtree as a root. If the subtree is accepted by $C_{n,r}$, then $B_{n,r+2}$ comes to the root of the subtree labelled with $(a, b)$ in an accepting configuration of $C_{n,r}$. At this moment it makes the same transition as $B_{n,2}$ makes at a leaf $a$ in the state $q_{acc}$, and then continues the simulation of $B_{n,2}$ on the upper part of the tree.

It remains to explain how the automaton $B_{n,r+2}$ simulates two different automata that share the same set of states, and does so without extra states. At each moment, it needs to determine, which of the two automata it simulates: $B_{n,2}$ or $C_{n,r}$. If the label of the current node is not of the form $(a, b)$, this can be understood from the upward direction: if it is $-1$ or $-2$, then $B_{n,2}$ is being simulated, and if the upward direction is one of $\{-3, \dots, -(r+2)\}$, then the simulated automaton is $C_{n,r}$. In a node with label of the form $(a, b)$, the automaton being simulated is determined from the state: if the state is from $P^+$, then $B_{n,r+2}$ has got here from above, and hence this is a simulation of $B_{n,2}$; and if the state is in $P^-$, then the automaton has come here from below, and $C_{n,r}$ is being simulated.

Such an automaton $B_{n,r+2}$ may accept only at the root, has the same partition of states into $P^+$ and $P^-$, and accepts at least one tree, because $B_{n,2}$ and $C_{n,r}$ each accept some trees. Each tree accepted by $B_{n,2}$ has at least $2^{c \cdot 1.618^n}$ leaves, and every tree accepted by $B_{n,r+2}$ has each of these leaves replaced with a tree accepted by $C_{n,r}$. Each tree that $C_{n,r}$ accepts has at least $2^{\frac{r}{2} c \cdot 1.618^n}$ nodes. Therefore, every tree accepted by $B_{n,r+2}$ has at least $2^{c \cdot 1.618^n} \cdot 2^{\frac{r}{2} c \cdot 1.618^n} = 2^{\frac{r+2}{2} c \cdot 1.618^n}$ nodes. The induction step has been proved.

It is left to consider the case of $n \in \{1, 2, 3\}$. By Theorem 5, there is a tree automaton with 1 state, over a signature with maximum rank $r$, such that the minimal tree it accepts has $2^r$ nodes. Then, all node labels, on which the transition of the automaton is undefined, are removed from the signature. Now the automaton accepts all trees and may be replaced with a 1-state tree-walking automaton that accepts immediately. Thus, for small $n$, if the constant in $\Omega$-notation is less than $1.618^{-3}$, then one can take these signatures and an immediately accepting automaton. $\square$

# Chapter 4

# Complexity of the emptiness problem for graph-walking automata and for tilings with star subgraphs

This chapter is about two kinds of automata on graphs. Graph-walking automata walk over an input graph by following its edges. Star automata accept a graph by tiling it with star subgraphs that assign states to nodes.

The basic definitions of automata are given in Section 4.1. Graph-walking automata and star automata are defined over a *signature*, which is an alphabet for graphs. A signature defines finite sets of possible node labels and possible labels of edge end-points (called directions). Also, for each node label, there is a set of directions used in all nodes with this label.

The decidability of the emptiness problem and upper bounds on its complexity are obtained for graph-walking automata and for star automata using generally the same method. A simpler problem called *signature non-emptiness* is considered first: does there exist at least one graph over a given signature? Its decidability is proved in Section 4.2 by reducing it to finding a non-negative integer solution to a certain system of linear Diophantine equations. From this, it is inferred that the non-emptiness problem for signatures can be solved in NP. Furthermore, if a signature is non-empty, that is, if there is at least one graph over this signature, then the number of nodes in the smallest such graph does not exceed $2mr \min\{r^r, k^{2r-2}\}$, where $m$ is the number of node labels in the signature, $2r$ is the number of directions, and $k$ is the maximum degree of a node.

It turns out that both checking non-emptiness of a graph-walking automaton and checking non-emptiness of a star automaton can be reduced to checking non-emptiness of a certain signature, which is constructed for a given automaton.

For star automata, such a reduction is presented in Section 4.3. It gives a proof that the non-emptiness problem for star automata is in NP. Also it gives an upper bound $sn^2k^{kn^2-1}$ on the number of nodes in the smallest accepted graph, where $n$ is the number of states in the star automaton, $s$ is the number of stars, and $k$ is the number of directions in the signature.

In Section 4.4, a graph-walking automaton is reduced to a signature. The reduction proves that its non-emptiness problem is in NEXP, as well as gives an upper bound $m4^{n(k+1)}k^{k4^n-1}$ on the number of nodes in the smallest accepted graph, where $n$ is the number of states, $k$ is the number of directions, and $m$ is the number of node labels.

In Section 4.5, all the above non-emptiness problems are proved to be hard in their complexity classes. NP-hardness of the signature non-emptiness problem is obtained by reducing 3-colourability to this problem. This also gives NP-hardness for non-emptiness of star automata. To prove NEXP-hardness of non-emptiness of graph-walking automata, it is shown that a graph-walking automaton can recognize the set of graphs containing a rectangular grid of exponential size in the number of its states. On this grid, the computation of a nondeterministic Turing machine is then simulated.

## 4.1 Graph-walking and star automata

In this section, graph-walking automata and star automata that generalize automata on strings and trees to graphs are formally defined. All definitions for graph-walking automata are inherited from the paper by Kunc and Okhotin [18]. Star automata are a generalization of nondeterministic tree automata and a variant of *elementary acceptors* by Thomas [26] without constraints on the number of tiles. Star automata are given in a different notation for uniformity with graph-walking automata.

Graph-walking automata are defined over a signature. A signature specifies the sets of labels of nodes and edge end-points in the graphs, and thus defines the set of all labelled graphs that can be used as inputs for a graph-walking automaton.

**Definition 9** ([18])**.** A *signature* $S$ is a quintuple $S = (D, -, \Sigma, \Sigma_0, (D_a)_{a\in\Sigma})$, where:

- $D$ is a finite set of directions, which are labels attached to edge end-points;
- a bijection $-\colon D \to D$ provides an opposite direction, with $-(-d) = d$ for all $d \in D$;
- $\Sigma$ is a finite set of node labels;
- $\Sigma_0 \subseteq \Sigma$ is a subset of possible labels of the initial node;
- $D_a \subseteq D$, for every $a \in \Sigma$, is the set of directions used in nodes labelled with $a$.

Graphs are defined over a signature like strings are defined over an alphabet.

**Definition 10.** A *graph* over a signature $S = (D, -, \Sigma, \Sigma_0, (D_a)_{a\in\Sigma})$ is a quadruple $(V, v_0, +, \lambda)$, where:

- $V$ is a finite set of nodes;
- $v_0 \in V$ is the initial node;
- edges are defined by a partial function $+\colon V \times D \to V$, such that if $v + d$ is defined, then $(v + d) + (-d)$ is defined and equals $v$; also denote $v - d = v + (-d)$;
- node labels are assigned by a total mapping $\lambda\colon V \to \Sigma$, such that
  i. $v + d$ is defined if and only if $d \in D_{\lambda(v)}$, and
  ii. $\lambda(v) \in \Sigma_0$ if and only if $v = v_0$.

The set of all graphs over the signature $S$ is denoted by $L(S)$.

The function + defines the edges of the graph. If $u+d=v$, then the nodes $u$ and $v$ in the graph are connected with an edge with its end-points labelled with directions $d$ (on the side of $u$) and $-d$ (on the side of $v$). Multiple edges and loops are possible: if $v+d=v$ and $d \neq -d$, then it is a loop at the node $v$ with two ends labelled with directions $d$ and $-d$. If $v+d=v$ and $d=-d$, then it is a loop at the node $v$ with one end, labelled with $d$.

A graph-walking automaton is defined similarly to a 2DFA and a DTWA, with an input graph instead of an input string or an input tree.

**Definition 11.** *A (deterministic) graph-walking automaton (GWA) over a signature* $S = (D, -, \Sigma, \Sigma_0, (D_a)_{a\in\Sigma})$ is a quadruple $A = (Q, q_0, F, \delta)$, where

- $Q$ is a finite set of states;
- $q_0 \in Q$ is the initial state;
- $F \subseteq Q \times \Sigma$ is a set of acceptance conditions;
- $\delta\colon (Q \times \Sigma) \setminus F \to Q \times D$ is a partial transition function, with $\delta(q,a) \in Q \times D_a$ for all $q$ and $a$ where $\delta$ is defined.

When an automaton operates on a graph, at every moment it knows its current state and sees only the label of the current node. The transition function gives the new state and the direction to one of the neighbouring nodes, in which the automaton moves. If the current pair of a state and a node label is in $F$, then the automaton accepts. If the pair is not in $F$ and no transition is defined for it, then the automaton rejects. It may also continue walking indefinitely, it this case it is said to loop.

Formally, an automaton's *configuration* on a graph $G = (V, v_0, +, \lambda)$ is a pair $(q, v)$, with $q \in Q$ and $v \in V$. A *computation* of an automaton $A$ on a graph $G$ is the following uniquely defined sequence of configurations. The computation starts in the initial configuration $(q_0, v_0)$. For every configuration $(q, v)$ in the computation, if $\delta(q, \lambda(v))$ is defined and equals $(q', d)$, then the next configuration after $(q, v)$ is $(q', v+d)$. Otherwise, the configuration $(q, v)$ is the last one in the computation; if $(q, \lambda(v)) \in F$, then the automaton *accepts* in the configuration $(q, v)$, otherwise it rejects. If the computation is an infinite sequence, then the automaton is said to *loop*.

A graph-walking automaton $A$ defines the language $L(A)$, this is the set of graphs it accepts.

The methods used in this chapter to prove the decidability of the emptiness problem for graph-walking automata and to determine its computational complexity can also be applied to another related model. These are *star automata*, which are defined as follows.

**Definition 12.** Let $S = (D, -, \Sigma, \Sigma_0, (D_a)_{a\in\Sigma})$ be a signature and let some linear order be fixed on the set of directions $D$. *A star automaton* $A_*$ *over the signature* $S$ is a pair $(Q, T)$, where

- $Q$ is a finite set of states;
- $T$ is a finite set of stars, where a star is a sequence of the form $(a, q, q_1, \ldots, q_{|D_a|})$, where $a$ is a node label, $q$ is used for the state in the current node, $q_1, \ldots, q_{D_a}$ are used for states in the neighbours of the current node in all directions from $D_a$.

A graph $G$ is accepted by the star automaton $A_*$, if there is a choice of states $(q(v))_{v\in V}$ in all nodes such that the following condition holds for each node $v \in V$. Let $a$ be the label of the node $v$, let $d_1, \ldots, d_{D_a}$ be the directions from $D_a$ listed in the order. Then, the star in the node $v$ is the sequence $s(v) = (a, q(v), q(v + d_1), \ldots, q(v + d_{|D_a|}))$. And every such star should belong to the set of automaton's stars $T$. Such a sequence $(q(v))_{v\in V}$ is called *a computation* of the star automaton $A_*$ on the graph $G$. There can be several computations.

## 4.2 The non-emptiness problem for signatures is in NP

In this section, the decidability of the non-emptiness problem for signatures is proved; more precisely, an NP-algorithm that solves this problem is constructed. Furthermore, for non-empty signatures, an upper bound on the number of nodes in the minimal graph over a given signature is obtained.

It turns out that to prove that a signature is non-empty it is not necessary to find an actual graph. It is sufficient to find only a collection of nodes without the edge structure of the graph; such a collection is described by a vector with every coordinate giving the number of nodes with a certain label. A vector can be turned into a graph if it satisfies a few conditions.

**Definition 13.** Let $S = (D, -, \Sigma, \Sigma_0, (D_a)_{a\in\Sigma})$ be a signature. A vector of non-negative integers $(x_a)_{a\in\Sigma}$, where $x_a$ is the number of nodes with the label $a$, is called *balanced*, if it satisfies the following two balance conditions:

1. an initial node exists and is unique: $\sum_{a_0\in\Sigma_0} x_{a_0} = 1$,
2. for each direction $d \in D$, such that $d \neq -d$, all nodes together need the same number of edges by $d$ and by $-d$:

$$\sum_{a\in\Sigma:\ d\in D_a} x_a = \sum_{a\in\Sigma:\ -d\in D_a} x_a.$$

The next lemma shows that every balanced vector gives rise to a graph, and hence one can work with balanced vectors instead of graphs.

**Lemma 7.** *Let $S = (D, -, \Sigma, \Sigma_0, (D_a)_{a\in\Sigma})$ be a signature. Let $x_a$, for each node label $a \in \Sigma$, be a non-negative integer.*

*A graph over the signature $S$ with exactly $x_a$ nodes labelled with $a$, for all $a \in \Sigma$, exists if and only if the vector $(x_a)_{a\in\Sigma}$ is balanced.*

*Furthermore, there is an algorithm that, given a signature $S$ and a balanced vector $(x_a)_{a\in\Sigma}$, constructs a graph over $S$ with exactly $x_a$ nodes with each label $a \in \Sigma$, and does so in time linear in the sum of sizes of the signature and of the constructed graph.*

*Proof.* For every graph $G$ over $S$, let $(x_a)_{a\in\Sigma}$ be the vector of quantities of nodes for all labels. It is claimed that the vector $(x_a)_{a\in\Sigma}$ is balanced. The first balance condition holds, because every graph has exactly one initial node. Now to the second condition. Let $d \in D$ be one of the directions, with $d \neq -d$. Then, every edge $v + d = u$ in the graph links the

two edge end-points: in the direction $d$ at the node $v$, and in the direction $-d$ at the node $u$. Thus, the total number $\sum_{a\in\Sigma:\ d\in D_a} x_a$ of edge end-points labelled with $d$ in the graph equals the number $\sum_{a\in\Sigma:\ -d\in D_a} x_a$ of edge end-points labelled with $-d$, and the second balance condition holds.

Conversely, let $(x_a)_{a\in\Sigma}$ be a balanced vector. A graph $G = (V, v_0, +, \lambda)$ with exactly $x_a$ nodes for each node label $a$ is constructed by the following algorithm.

- First, the set of nodes $V$ and the labelling function $\lambda$ are defined: for each node label $a \in \Sigma$ in the signature, $x_a$ new nodes labelled with $a$ are added to the set $V$.
- The initial node is the node with a label from the set $\Sigma_0$, the first balance condition states that such a node exists and is unique.
- Now the edges shall be defined so, that each node $v$ labelled with $a$ will have edges exactly in the directions from $D_a$. For each direction $d \in D$, let $I_d$ be the set of all nodes $v$ with $d \in D_{\lambda(v)}$.

  For such directions $d \in D$, that $d = -d$, the algorithm makes loops: for every node $v \in I_d$ it adds a loop $v + d = v$.

  For each pair of opposite directions $d \neq -d$, the algorithm takes nodes from $I_d$ and $I_{-d}$, and links them with $(d, -d)$-edges. By the second balance condition, $|I_d| = |I_{-d}|$, thus, every node gets all the edges it needs.

□

Now, to check whether a signature is non-empty, that is, whether there is at least one graph over this signature, one can just check whether there is at least one balanced vector for this signature.

For a signature $S = (D, -, \Sigma, \Sigma_0, (D_a)_{a\in\Sigma})$, balanced vectors $(x_a)_{a\in\Sigma}$ with the minimal possible sum of coordinates will be called *minimal balanced vectors*.

How large could be the sum of the coordinates of a minimal balanced vector? The next theorem gives an upper bound on this sum, that is, on the minimal number of nodes in the graph over a signature.

**Theorem 14.** *Let $S = (D, -, \Sigma, \Sigma_0, (D_a)_{a\in\Sigma})$ be a non-empty signature, and assume that $|D| \geqslant 2$ and that $D_a$ is non-empty for all $a \in \Sigma$. Let $r = \frac{1}{2}|D|$, $m = |\Sigma|$ and $k = \max_{a\in\Sigma} |D_a|$.*

*Then, there is a graph over the signature $S$ with at most $2mr \min\{r^r, k^{2r-2}\}$ nodes.*

Note that the bound $2mrk^{2r-2}$ can be useful for signatures with many directions, but with a small maximum degree of nodes. Later on, such signatures will be produced by the reductions of the emptiness problems for graph-walking automata and for star automata to the emptiness problem for signatures.

First, the conditions and the claims of Theorem 14 are reformulated in the language of linear algebra.

By Lemma 7, to prove Theorem 14 it is sufficient to prove that there is such a balanced vector $(x_a)_{a\in\Sigma}$ that $\sum_{a\in\Sigma} x_a \leqslant 2mr \min\{r^r, k^{2r-2}\}$.

Let $n$ be the number of pairs of opposite directions $\{d, -d\}$, with $d, -d \in D$ and $d \neq -d$, in the signature $S$. It is convenient to rewrite $n$ linear equations in the second

balance condition as one vector equation. Let $\{d_1, -d_1, \ldots, d_n, -d_n\}$ be all such directions in $D$ that $d \neq -d$, here the directions $d_i$ and $-d_i$ are opposite, for $i = 1, \ldots, n$.

For each node label $a \in \Sigma$, the contribution of one node labelled with $a$ to the balance of directions in a graph is given by a column vector $v_a$ of height $n$. The $i$-th element of the vector $v_a$, for $i \in \{1, \ldots, n\}$, is defined as follows:

$$v_{a,i} = \begin{cases} 1 & \text{if } d_i \in D_a, -d_i \notin D_a \\ -1 & \text{if } -d_i \in D_a, d_i \notin D_a \\ 0 & \text{if } d_i \notin D_a, -d_i \notin D_a \text{ or } d_i \in D_a, -d_i \in D_a \end{cases} \tag{4.1}$$

Thus, the $i$-th element of the vector $v_a$ is the contribution of an $a$-labelled node to the difference of the number of directions $d_i$ and $-d_i$ in a graph.

Then, the second balance condition for the vector of quantities of labels $(x_a)_{a\in\Sigma}$ can be written in the following form:

$$\sum_{a\in\Sigma} x_a v_a = 0$$

If $n = 0$, then all directions are of the form $d = -d$, and one initial node with the loops is a correct graph. Let $n \geqslant 1$. As $n \leqslant r$, it is sufficient to prove an upper bound $2mn \min\{n^n, k^{2n-2}\}$. Let $(x_a)_{a\in\Sigma}$ be a balanced vector with the minimal possible sum of the coordinates. Among the initial node labels, only one has a non-zero coefficient. Fix this initial label $a_0$ and let the vector $-v_{a_0}$ be denoted by $b$. Then, the coefficients for other initial labels are zeros and $\sum_{a\in(\Sigma\setminus\Sigma_0)} x_a = \left(\sum_{a\in\Sigma} x_a\right) - 1$. Then, to prove the theorem, it is sufficient to find such a non-negative integer solution $(x_a)_{a\in(\Sigma\setminus\Sigma_0)}$ to the equation $\sum_{a\in(\Sigma\setminus\Sigma_0)} x_a v_a = b$, that $\sum_{a\in(\Sigma\setminus\Sigma_0)} x_a \leqslant 2mn \min\{n^n, k^{2n-2}\} - 1$.

Some vectors $v_a$ for different non-initial labels can coincide. Let $v_1, \ldots, v_\ell$ be all vectors from the set $\{\, v_a \mid a \in (\Sigma \setminus \Sigma_0) \,\}$ without repetitions and without a zero vector. Note that $\ell < m$. Then, it is sufficient to find a non-negative integer solution $(x_i)_{i=1}^{\ell}$ to the equation $\sum_{i=1}^{\ell} x_i v_i = b$, with $\sum_{i=1}^{\ell} x_i \leqslant 2\ell n \min\{n^n, k^{2n-2}\}$.

What is known about vectors $v_1, \ldots, v_\ell$? These are column vectors of height $n$, with all elements in $\{0, 1, -1\}$. Each vector has at most $k$ non-zero elements, since each node label $a \in \Sigma$ has at most $k$ directions in $D_a$. To apply the methods of linear algebra, these vectors are considered over the field of real numbers: $v_1, \ldots, v_\ell \in \mathbb{R}^n$. Therefore, Theorem 14 is reduced to the following lemma.

**Lemma 8.** *Let $v_1, \ldots, v_\ell \in \{0, 1, -1\}^n$ be distinct non-zero column vectors of height $n$, where $n \geqslant 1$; let $b \in \{0, 1, -1\}^n$ be a column vector. Let $k$ be the maximum number of non-zero elements in the vector. Then, if the linear equation $\sum_{i=1}^{\ell} x_i v_i = b$ has at least one non-negative integer solution, then there exists such a non-negative integer solution $(x_i)_{i=1}^{\ell}$, that $\sum_{i=1}^{\ell} x_i \leqslant 2\ell n \min\{n^n, k^{2n-2}\}$.*

The proof of Lemma 8 will use a classical bound on matrix determinants, proved by Hadamard [13].

**Theorem A** (Hadamard [13])**.** Let $n \geqslant 1$ be an integer and let $A$ be an $n \times n$ matrix, with all elements real and not exceeding 1 in absolute value. Then, $|\det A| \leqslant n^{\frac{n}{2}}$.

Hadamard also proved that if $n$ is a power of 2, then the bound $n^{\frac{n}{2}}$ is achieved on some matrices.

Also, I will use the following trivial upper bound for matrices with a small number of non-zeros in columns.

**Claim 9.** *Let $n \geqslant 1$ be an integer and let $A$ be an $n \times n$ matrix, with all elements real and not exceeding 1 in absolute value. Let $k \geqslant 1$ be such an integer, that each column in the matrix $A$, maybe except one, has at most $k$ non-zero elements. Then $|\det A| \leqslant k^{n-1}$.*

The upper bounds on determinants are used to estimate the coefficients in linear equations.

**Lemma 9.** *Let $n \geqslant 1$ be an integer, let $v_1, \ldots, v_t \in \{0, 1, -1\}^n$, with $t \geqslant 1$, be column vectors of height $n$, which are linearly independent in $\mathbb{R}^n$. Let $k$ be the maximum number of non-zero elements in a vector, and let $N = \min\{n^{\frac{n}{2}}, k^{n-1}\}$. Let some vector $u \in \mathbb{R}^n$, with the maximum absolute value of its elements $c$, be represented as a linear combination: $\alpha_1 v_1 + \ldots + \alpha_t v_t = u$.*

*Then, $|\alpha_i| \leqslant cN$, for all $i \in \{1, \ldots, t\}$. Furthermore, if all elements in the vector $u$ are integers, then all coefficients $\alpha_i$, for $i \in \{1, \ldots, t\}$, are rational, and after multiplying the equation by their least common denominator one obtains the equation $\beta_1 v_1 + \ldots + \beta_t v_t + \beta_{t+1} u = 0$, with all coefficients $\beta_i$, for $i \in \{1, \ldots, t+1\}$, integer and not exceeding $cN$ in absolute value.*

*Proof.* If $u$ is a zero vector, then all coefficients in the linear combination are zeros. Now let $u$ be not a zero vector. The vectors $v_1, \ldots, v_t$ are linearly independent, so the system of equations $x_1 v_1 + \ldots + x_t v_t = u$ has at most one solution. Thus, the solution $(\alpha_1, \ldots, \alpha_t)$ is unique. To solve this system of equations using Cramer's rule, one needs the matrix of coefficients $V = (v_1, \ldots, v_t)$ to be square.

Since the vectors $v_1, \ldots, v_t$ are linearly independent, $t \leqslant n$. First, consider the case of $t < n$. The matrix $(v_1, \ldots, v_t, u)$ has the column rank $t$, because the columns $v_1, \ldots, v_t$ are linearly independent, and the column $u$ is their linear combination. It is known that the column rank equals the row rank, so there are $t$ linearly independent rows in the matrix $(V, u)$, all other rows are their linear combinations. That is, in the system of equations $x_1 v_1 + \ldots + x_t v_t = u$, all equations are linear combinations of some $t$ linearly independent equations. Taking only these $t$ linearly independent equations one obtains a system $x_1 v'_1 + \ldots + x_t v'_t = u'$, with all vectors of height $t$. The set of solutions has not changed, so $(\alpha_1, \ldots, \alpha_t)$ remains the only solution. Let $V' = (v'_1, \ldots, v'_t)$ be the matrix of coefficients of the new system of equations, it is a non-degenerate square matrix. If $t = n$, then the matrix $V$ is already square and non-degenerate; in this case let $V' = V$, $u' = u$.

Now the new system of equations can be solved by Cramer's rule. Let $V'_i = (v'_1, \ldots, v'_{i-1}, u', v'_{i+1}, \ldots, v'_t)$ be the matrix, obtained from $V'$ by replacing of the $i$-th column with the column vector $u'$, for each $i = 1, \ldots, t$. Then, Cramer's rule claims that the unique solution to the system is $\alpha_i = \frac{\det V'_i}{\det V'}$, for $i = 1, \ldots, t$.

Now one needs to estimate the determinants of the matrices $V'$ and $V'_i$, for $i = 1, \ldots, t$. The matrix $V'$ has all its elements in $\{0, 1, -1\}$. Also, each column of $V'$ has at most $k$ non-zeros. So Theorem A and Claim 9 give $|\det V'| \leqslant \min\{n^{\frac{n}{2}}, k^{n-1}\} = N$. Since all elements of $V'$ are integers and the matrix is non-degenerate, $\det V'$ is a non-zero integer. Now consider the matrix $V'_i$, for some $i = 1, \ldots, t$. Let $V''_i$ be the matrix obtained from $V'_i$ by dividing the $i$-th column, which equals $u$, by $c$. Then, all elements of $V''_i$ are not greater than 1 in absolute value. And each column has at most $k$ non-zero elements, maybe, except the $i$-th column. Then, by Theorem A and by Claim 9, the determinant of $V''_i$ is estimated as follows: $|\det V''_i| \leqslant \min\{n^{\frac{n}{2}}, k^{n-1}\} = N$. Thus, the determinant of the matrix $V'_i$, which has one column multiplied by $c$, is bounded like this: $|\det V'_i| \leqslant cN$.

So, $|\alpha_i| = |\frac{\det V_i'}{\det V'}| \leqslant |\det V_i'| \leqslant cN$, for all $i = 1, \ldots, t$. If all elements of the vector $u$ are integers, then all the determinants $\det V_i'$ are integers as well. Then all $\alpha_i$, for $i = 1, \ldots, t$, are rational. And after muliplying the equation by their least common denominator, which is not greater than $|\det V'|$ in absolute value, one gets all new coefficients $\beta_i$, for $i = 1, \ldots, t+1$, not greater in absolute value than $\max\{|\det V_1'|, \ldots, |\det V_t'|, |\det V'|\} \leqslant cN$. □

Now it is time to prove the lemma, to which Theorem 14 has been reduced.

*Proof of Lemma 8.* Let $N = \min\{n^{\frac{n}{2}}, k^{n-1}\}$ be the minimum of the upper bounds from Theorem A and from Claim 9 on determinants of $n \times n$ matrices with real elements not exceeding 1 in absolute value, and with at most $k$ non-zero elements in each column, maybe, except one.

Let $(x_1, \ldots, x_\ell)$ be a non-negative integer solution to the system of linear equations $\sum_{i=1}^{\ell} x_i v_i = b$, with the minimum sum $\sum_{i=1}^{\ell} x_i$, and among these, with the minimum number of coordinates greater than $N$. The goal is to prove, that $\sum_{i=1}^{\ell} x_i \leqslant 2\ell n \min\{n^n, k^{2n-2}\}$.

**Step 1** is to prove that all vectors $v_i$, for $i = 1, \ldots, \ell$, with $x_i > N$, are linearly independent over the field $\mathbb{R}$.

For the sake of a contradiction, suppose that these vectors are linearly dependent. Then a linear dependence involving the least number of vectors is chosen. The vectors $v_1, \ldots, v_\ell$ are rearranged, so that the vectors from the dependence go in the beginning: let $v_1, \ldots, v_{t+1}$ be the vectors from this minimal linear dependence. It is known that $t \geqslant 2$, because all vectors $v_1, \ldots, v_\ell$ are distinct and there is no zero vector among them.

The vectors $v_1, \ldots, v_t$ are linearly independent, whereas $v_1, \ldots, v_{t+1}$ are linearly dependent. Then, the vector $v_{t+1}$ is uniquely represented as a linear combination of the others: $v_{t+1} = \alpha_1 v_1 + \ldots + \alpha_t v_t$, where $\alpha_1, \ldots, \alpha_t \in \mathbb{R}$.

The vector $v_{t+1}$ has all its elements integer and the maximum absolute value of its elements is 1; the vectors $v_1, \ldots, v_t$ satisfy all conditions of Lemma 9. Thus, by Lemma 9, all coefficients $\alpha_1, \ldots, \alpha_t$ are rational, and after multiplying the linear combination by their least common denominator one gets the new linear combination $\beta_1 v_1 + \ldots + \beta_{t+1} v_{t+1} = 0$, with all coefficients integer and not exceeding $N$ in absolute value.

Since the chosen linear dependence has the minimal number of vectors, $\beta_i \neq 0$, for all $i = 1, \ldots, t+1$. If $\sum_{i=1}^{t+1} \beta_i < 0$, then the dependence $\beta_1 v_1 + \ldots + \beta_{t+1} v_{t+1} = 0$ can be multiplied by $-1$, so one can assume, that $\sum_{i=1}^{t+1} \beta_i \geqslant 0$.

Consider the case when $\sum_{i=1}^{t+1} \beta_i > 0$. Then, let $(y_1, \ldots, y_\ell)$ be a vector defined by $y_i = x_i - \beta_i$, for $i = 1, \ldots, t+1$, and $y_i = x_i$, for $i = t+2, \ldots, \ell$. Then, $\sum_{i=1}^{\ell} y_i v_i = b$, that is, $(y_1, \ldots, y_\ell)$ is another solution to the system of equations. All $y_i$ are non-negative integers, because $x_1, \ldots, x_{t+1}$ are greater than $N$, and $\beta_1, \ldots, \beta_{t+1}$ are integer and not greater than $N$ in absolute value. And, $\sum_{i=1}^{\ell} y_i < \sum_{i=1}^{\ell} x_i$. This contradicts the minimality of the sum of the coordinates in the solution $(x_1, \ldots, x_\ell)$.

Now let $\sum_{i=1}^{t+1} \beta_i = 0$. Then one can similarly subtract $(\beta_1, \ldots, \beta_{t+1})$ from $(x_1, \ldots, x_{t+1})$ several times until some coefficient among the first $t+1$ becomes not greater than $N$. Such subtractions will not break the equation, will not make any coordinate negative, will not change the sum of the coordinates in the solution, but will decrease the number of coordinates which are greater than $N$. This contradicts the minimality of the number of such coordinates among the solutions with the minimal sum of the coordinates.

Step 1 is done. Now it is known that all vectors among $v_1, \ldots, v_\ell$ which have the corresponding coefficients in the solution $(x_1, \ldots, x_\ell)$ greater than $N$ are linearly independent. Let these vectors be put first, so that they are $v_1, \ldots, v_t$.

**Step 2** is to prove that $x$ is the desired solution, that is, that $\sum_{i=1}^{\ell} x_i \leqslant 2\ell n \min\{n^n, k^{2n-2}\}$.

The sum to be estimated is: $\sum_{i=1}^{\ell} x_i = \sum_{i=1}^{t} x_i + \sum_{i=t+1}^{\ell} x_i$. The second sum is bounded by $\sum_{i=t+1}^{\ell} x_i \leqslant (\ell - t)N$, as it has all coefficients not greater than $N$. If the first sum is non-empty ($t > 0$), then the first $t$ variables are bounded as follows. The system of equations is rewritten in the following way: $x_1 v_1 + \ldots + x_t v_t = b - (x_{t+1} v_{t+1} + \ldots + x_\ell v_\ell)$. Here the vectors $v_1, \ldots, v_t$ are linearly independent, whereas the sum on the right-hand side is a column vector of height $n$, with all elements not greater than $\ell N$ in absolute value (if $t > 0$, then $\sum_{i=t+1}^{\ell} x_i < \ell N$). By applying Lemma 9, with $u = b - (x_{t+1} v_{t+1} + \ldots + x_\ell v_\ell)$, one obtains $|x_i| \leqslant \ell N^2$, for all $i = 1, \ldots, t$. As $t \leqslant n$,

$$\sum_{i=1}^{\ell} x_i = \sum_{i=1}^{t} x_i + \sum_{i=t+1}^{\ell} x_i \leqslant t\ell N^2 + (\ell - t)N \leqslant n\ell N^2 + \ell N \leqslant$$

$$\leqslant 2\ell n N^2 = 2\ell n \min\{n^n, k^{2n-2}\}.$$

□

Theorem 14, which has just been proved, gives the upper bound $2mr \min\{r^r, k^{2r-2}\}$ on the number of nodes in the minimal graph over a non-empty signature, which depends on its parameters: on the number of node labels $m = |\Sigma|$, on the number of directions $2r = |D|$ and on the maximum possible degree of a node $k = \max\{\, |D_a| \mid a \in \Sigma \,\}$. This bound, and also Lemma 7, that allows one to work with balanced vectors instead of graphs, help to construct an NP-algorithm, that solves the non-emptiness problem for signatures.

**Theorem 15.** *There is an NP-algorithm that takes a signature as an input and determines whether there is at least one graph over this signature or not.*

*Proof.* The size of an input $S = (D, -, \Sigma, \Sigma_0, (D_a)_{a \in \Sigma})$ is not less than $|\Sigma| + |D|$. In the degenerate case of $|D| \leqslant 1$, it is sufficient to check for one-node graphs. Any initial labels $a_0$ with $D_{a_0}$ empty form correct graphs; any such non-initial labels can be omitted.

With the trivial cases removed, by Theorem 14, if a graph over the signature $S$ exists, then there is a graph with at most exponentially many nodes in $|D|$ and $|\Sigma|$. Then, by Theorem 14 and by Lemma 7, the signature is non-empty if and only if there exists a balanced vector $(x_a)_{a \in \Sigma}$, with the sum of coordinates not greater than this exponential upper bound.

Thus, the nondeterministic algorithm guesses a vector $(x_a)_{a \in \Sigma}$, with sum of the coordinates not greater than exponential, and writes it down in polynomial time. It remains to check whether the guessed vector is balanced: that is, whether only one label among the initial node labels has a non-zero coefficient, and whether for each pair of opposite directions $(d, -d) \in D$, with $d \neq -d$, the following equation holds:

$$\sum_{a \in \Sigma:\ d \in D_a} x_a = \sum_{a \in \Sigma:\ -d \in D_a} x_a.$$

This can all be checked in polynomial time, because the number of terms in these sums is polynomial, and each term is not greater than exponential.

If the algorithm guessed the vector, which is balanced, then the signature is non-empty and the algorithm answers "yes". Otherwise, it answers "no". □

In fact, the non-emptiness problem for signatures is NP-complete, this is shown later in Section 4.5.

## 4.3 Reducing a star automaton to a signature

This section proves the decidability of the emptiness problem for star automata. An NP-algorithm is constructed, which, for a given star automaton, determines whether it accepts at least one graph. Moreover, an upper bound on the number of nodes in the smallest accepted graph is proved in this section.

It turns out that the emptiness problem for star automata can be reduced in polynomial time to the emptiness problem for signatures, which was proved to be in NP.

**Theorem 16.** *There exists a polynomial-time algorithm that takes as an input a signature $S = (D, -, \Sigma, \Sigma_0, (D_a)_{a\in\Sigma})$ with $k$ directions and a star automaton $A_* = (Q, T)$ over $S$ with $n$ states and $s$ stars, and computes a signature $S' = (D', -, \Sigma', \Sigma'_0, (D'_{a'})_{a'\in\Sigma'})$ with $kn^2$ directions and with $s$ node labels, with the following property. There exists a bijective function $f$ that maps a graph $G$ over $S$ and a computation $C = (q(v))_{v\in V}$ of the automaton $A_*$ on this graph to a graph $G' = f(G, C)$ over the signature $S'$, which has the same set of nodes and the same edge structure as the graph $G$ (the only difference between $G$ and $G'$ is in node labels and in directions).*

*Proof.* Node labels and directions of the new signature $S'$ will contain information on old node labels and directions, and also some additional information that encodes the computation of the star automaton $A_*$ on a graph. More precisely, node labels will additionally encode stars in nodes that appear in the computation, whereas directions will encode the states of the star automaton at the two ends of an edge.

The new signature $S'$ is constructed as follows.

- Node labels are all the stars of the automaton $A_*$, that is, $\Sigma' = T$.
- Initial node labels are all the stars of $A_*$, in which the first component is an initial node label from the old signature, that is, $\Sigma'_0 = \{\, (a, q, q_1, \ldots, q_{|D_a|}) \in T \mid a \in \Sigma_0 \,\}$.
- The set of directions is $D' = D \times Q \times Q$, where the direction $(d, q_1, q_2)$ means that in the old graph the direction $d$ was here, and in the encoded computation the state at the current node is $q_1$ and the state at the opposite end of the edge is $q_2$.
- The relation of the opposite direction is: $-(d, q_1, q_2) = (-d, q_2, q_1)$, for all $(d, q_1, q_2) \in D'$.
- For each star $t = (a, q, q_1, \ldots, q_{|D_a|}) \in \Sigma'$, where $d_1, \ldots, d_{|D_a|}$ are ordered directions from $D_a$, the set of directions for the node label $t$ is defined by $D'_t = \{\, (d_i, q, q_i) \mid i = 1, \ldots, |D_a| \,\}$.

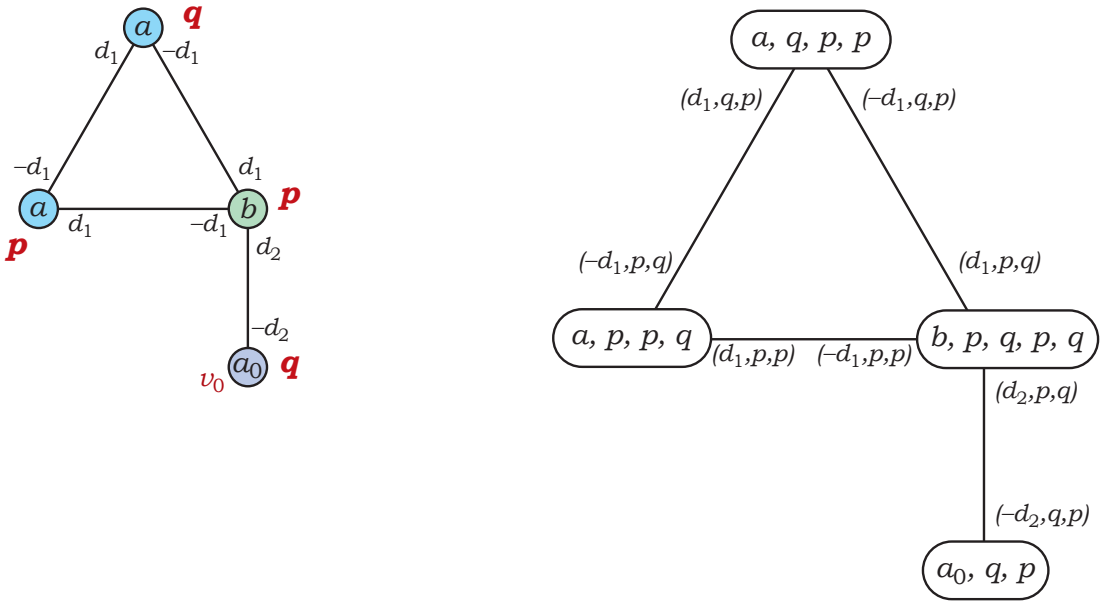


Figure 4.1: Left: computation of a star automaton $A_*$ on a graph $G$. Right: augmented graph $G'$ that encodes both $G$ and this computation.

Such a signature $S'$ can be computed from $S$ and $A_*$ in polynomial time. There are exactly $kn^2$ directions and exactly $s$ node labels in the signature $S'$.

It will be proved now, that there is a one-to-one correspondence between graphs over $S'$ and pairs $(G, C)$ of a graph over $S$ and a computation of $A_*$ on this graph. An example of such a correspondence is shown in Figure 4.1. For a star automaton $A_*$ with stars $(a, q, p, p)$, $(a, p, p, q)$, $(b, p, q, p, q)$, $(a_0, q, p)$, its computation on a graph $G$ is given on the left. On the right, there is a graph $G'$ that encodes stars in node labels and states at the two ends of an edge in directions.

Let $G = (V, v_0, +, \lambda)$ be a graph over $S$, and let $C = (q(v))_{v \in V}$ be a computation of the star automaton $A_*$ on this graph. Then the graph $f(G, C) = G' = (V', v'_0, +, \lambda')$ over the signature $S'$ that encodes the graph $G$ and the computation $C$ is constructed as follows.

- The set of nodes and the initial node are the same: $V' = V$, $v'_0 = v_0$.
- The edges in the graph $G'$ connect the same nodes as in $G$, but all the directions are augmented with the states at the ends of an edge. If $v + d = u$ in the graph $G$, then $v + (d, q(v), q(u)) = u$ in the graph $G'$, and these are all edges in $G'$. Then, the ends of each edge are labelled with opposite directions.
- The node labels in $G'$ are stars in nodes. For each node $v \in V$ with some label $\lambda(v) = a$, the node label in the graph $G'$ is $\lambda'(v) = (a, q(v), q(v+d_1), \ldots, q(v+d_{|D_a|}))$, where $d_1, \ldots, d_{|D_a|}$ are ordered directions from $D_a$. Then, $\lambda'(v) \in T = \Sigma'$, because $(q(v))_{v \in V}$ is a computation. And the directions in $G'$, used at the node $v$, are all the directions from $D_{\lambda'(v)}$. And only the initial node has an initial label.

This transformation maps different pairs $(G, C)$ to different graphs $G'$, because no information is lost. Conversely, for each graph $G'$ over the signature $S'$ there is a unique corresponding pre-image $(G, C)$, where $G$ is obtained by dropping some information from

all labels, and node labels explicitly give states and stars in a computation. Each edge in $G'$ checks that the states at the nodes it connects are consistent with the stars. □

Now the results proved for signatures in the previous section will be transferred to star automata.

**Corollary 2.** *The non-emptiness problem for star automata, that is, whether a given star automaton accepts at least one graph or not, can be solved in NP.*

*Proof.* By Theorem 16, for a star automaton $A_*$ that works over some signature $S$, one can construct in polynomial time such a signature $S'$ of polynomial size, that graphs over $S'$ are bijectively mapped to the computations of $A_*$ on graphs over $S$.

A graph is accepted by the star automaton $A_*$ if there exists at least one computation of $A_*$ on it. Thus, to check whether the star automaton is non-empty, one can just check whether the signature $S'$ is non-empty. By Theorem 15, the latter can be done in nondeterministic polynomial time. □

The upper bound on the number of nodes in the minimal graph over a signature (Theorem 14) can be transferred to star automata as well.

**Corollary 3.** *Let $S = (D, -, \Sigma, \Sigma_0, (D_a)_{a\in\Sigma})$ be a signature with $k \geqslant 2$ directions, and with $|D_a| \geqslant 1$ for all $a \in \Sigma$. Let $A_* = (Q, T)$ be a star automaton with $n$ states and with $s$ stars over this signature. If $A_*$ accepts at least one graph, then the accepted graph with the minimal number of nodes has at most $sn^2k^{kn^2-1}$ nodes.*

*Proof.* Let $A_*$ accept at least one graph. The signature $S'$ is constructed from the signature $S$ and from the star automaton $A_*$ by Theorem 16. The graphs over $S'$ correspond to the computations of $A_*$ on graphs over $S$ with the same number of nodes.

Then, the number of nodes in the minimal accepted graph for $A_*$ equals the number of nodes in the minimal graph over the signature $S'$. This signature has $kn^2$ directions and $s$ node labels, the maximum degree of a node does not exceed $k$ (because the function $f$ from Theorem 16 does not change the edge structure of a graph). Then, Theorem 14 gives the following upper bound on the number of nodes in the minimal graph: $2s\frac{1}{2}kn^2 \min\{(\frac{1}{2}kn^2)^{\frac{1}{2}kn^2}, k^{kn^2-2}\}$. It can be bounded by a simpler expression:

$$2s\frac{1}{2}kn^2 \min\left\{(\frac{1}{2}kn^2)^{\frac{1}{2}kn^2}, k^{kn^2-2}\right\} \leqslant skn^2k^{kn^2-2} = sn^2k^{kn^2-1}.$$

□

## 4.4 Reducing a graph-walking automaton to a signature

In Section 4.3, the emptiness problem for star automata was reduced to the emptiness problem for signatures. In this section such a reduction is made for the emptiness problem for graph-walking automata.

Note that whereas a computation of a star automaton is a way to choose states in nodes, and the graph is accepted by a star automaton if there is at least one computation

on this graph, graph-walking automata are different. In a graph-walking automaton, the computation on a graph is a sequence of configurations $(q, v)$ of the automaton on a graph, where $q$ is the current state, and $v$ is the node which the automaton visits at the moment. This sequence in defined uniquely for each graph. The graph is accepted if the computation is accepting, that is, ends with an accepting configuration.

One way to reduce a graph-walking automaton to a signature is to simulate it by a star automaton. The next theorem shows that if some set of graphs is recognized by a graph-walking automaton, then this set of graphs can be defined by some star automaton. There is an analogous result for trees: star automata on trees are *nondeterministic tree automata*, graph-walking automata on trees are *deterministic tree-walking automata*, and, as noted by Bojańczyk and Colcombet [4], the inclusion of the class of languages defined even by nondeterministic tree-walking automata into the class defined by tree automata is a folklore result.

**Theorem 17.** *For every $n$-state graph-walking automaton $A = (Q, q_0, F, \delta)$ over some signature $S = (D, -, \Sigma, \Sigma_0, (D_a)_{a\in\Sigma})$ with $k$ directions and $m$ node labels, there exists a star automaton $A_* = (P, T)$ with $(k + 1)^n$ states and at most $m(k + 1)^{n(k+1)}$ stars, defined over the same signature $S$, which accepts exactly the same graphs as $A$. The star automaton $A_*$ has size exponential in the size of $A$ and is constructed in exponential time.*

This theorem is given without a proof, because the next theorem gives a direct reduction of a graph-walking automaton to a signature that provides a better upper bound on the number of nodes in the minimal accepted graph.

**Theorem 18.** *There exists an algorithm that takes as an input some $n$-state graph-walking automaton $A = (Q, q_0, F, \delta)$ over some signature $S = (D, -, \Sigma, \Sigma_0, (D_a)_{a\in\Sigma})$ with $k$ directions and $m$ node labels, and computes such a signature $S' = (D', -, \Sigma', \Sigma'_0, (D'_{a'})_{a'\in\Sigma'})$ with $k4^n$ directions and with not more than $m4^{nk}$ node labels, that the following condition holds.*

*There exist two functions $f$ and $g$. The function $f\colon L(A) \to L(S')$ injectively maps graphs over $S$, accepted by the automaton $A$, to graphs over $S'$, and the function $g\colon L(S') \to L(A)$ is a surjection, such that $g(f(G)) = G$. If $G' = f(G)$ or $G = g(G')$, then the graphs $G$ and $G'$ have the same sets of nodes and the same edge structure, only node labels and directions are different.*

*The size of the resulting signature is exponential in the size of the input, and the algorithm works in time exponential in the size of the input.*

*Proof.* New node labels and directions of the signature $S'$ encode node labels and directions of the signature $S$ and some additional information about the behavior of the automaton $A$ in the vicinity of the node or edge end-point.

The new directions are $D' = D \times 2^Q \times 2^Q = \{\, (d, Q_{in}, Q_{out}) \mid d \in D;\ Q_{in}, Q_{out} \subseteq Q \,\}$. Every new direction $(d, Q_{in}, Q_{out})$ is an old direction $d$ with two sets of states attached: $Q_{in}$ encodes the states in which the automaton came in its computation on a graph to the current edge end-point moving in the direction $-d$, whereas $Q_{out}$ consists of states, in which the automaton comes to the opposite end of the edge, moving in the direction $d$.

The opposite direction is $-(d, Q_{in}, Q_{out}) = (-d, Q_{out}, Q_{in})$, for each $(d, Q_{in}, Q_{out}) \in D'$.

Each node label in $S'$ contains an old node label and all information about the new directions in the node. But not every combination of new directions at a node makes a

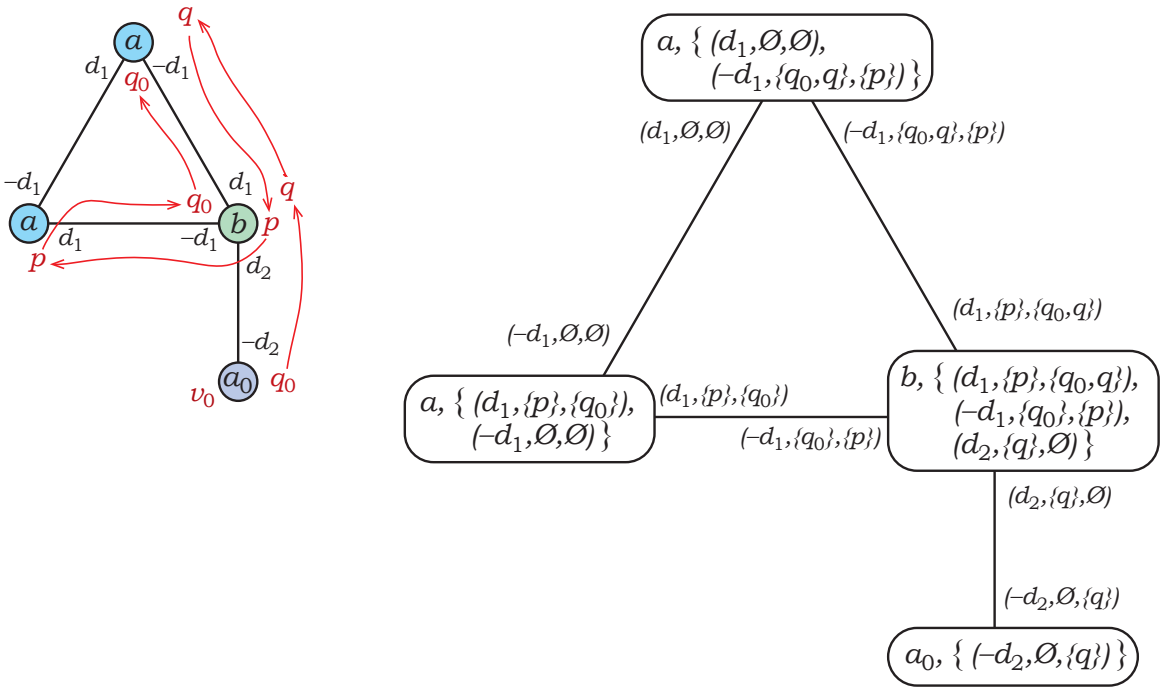


Figure 4.2: Left: the accepting computation of some graph-walking automaton $A$ on some graph $G$ over the signature $S$. Right: the graph $G'$ over the signature $S'$ that encodes the graph and the accepting computation.

new label. The goal is to ensure that each graph over $S'$ encodes a graph over $S$ that is accepted by $A$, along with an accepting computation of $A$ on this graph. For this, some combinations that cannot appear in accepting computations of the automaton $A$ will be left out.

The set of node labels $\Sigma'$ is a subset of

$$\widehat{\Sigma}' = \{\, (a, E) \mid a \in \Sigma, E = \{(d, Q_{in,d}, Q_{out,d})\}_{d \in D_a},\ \text{where } Q_{in,d}, Q_{out,d} \subseteq Q \text{ for all } d \in D_a \,\}.$$

It will be specified later, which elements of the set $\widehat{\Sigma}'$ are in $\Sigma'$ and which are not.

The set of directions of a new node label $(a, E)$ is $D'_{(a,E)} = E$. The label $(a, E)$ is initial if and only if the label $a$ is initial. Note that for each node label $a \in \Sigma$, there is only one direction $(d, Q_{in,d}, Q_{out,d})$ with the first component $d$ in the set $E$, for each direction $d \in D_a$.

Figure 4.2 gives an example of how a graph $G$ over $S$ accepted by the automaton $A$ can be converted to a graph $G'$ over the signature $S'$ by adding to each direction the information on the states in which the automaton crosses the edge, and by adding to each node label the information contained in all new directions at the node.

To complete the definition of the signature $S'$, it remains to say, which pairs $(a, E)$ from the set $\widehat{\Sigma}'$ are in the set $\Sigma'$, that is, are node labels of $S'$. Some pairs $(a, E)$, which represent situations that cannot occur in any accepting computations of $A$, will be left out, and leaving them out will ensure that every graph over the signature $S'$ encodes some graph $G$ and an accepting computation of $A$ on $G$.

A pair $(a, E)$ is in $\Sigma'$ if and only if the following conditions hold.

1. The sets $Q_{in,d}$ and $Q_{in,e}$ cannot intersect for directions $d \neq e$, where $d, e \in D_a$. If the label $a$ is initial, then for each $d \in D_a$ it is prohibited to have $q_0 \in Q_{in,d}$.

Indeed, the automaton $A$ cannot come to the node in the state $q$ twice in the accepting computation, otherwise it will repeat a configuration and loop. By similar reasons the automaton cannot return to the initial node in the state $q_0$ in the accepting computation.

Denote by $Q_{in}$ the set of all states in which the automaton $A$ visits the node, according to the information in the node label $(a, E)$. If $a \notin \Sigma_0$, then $Q_{in} = (\bigcup_{d \in D_a} Q_{in,d})$, if $a \in \Sigma_0$, then $Q_{in} = (\bigcup_{d \in D_a} Q_{in,d}) \cup \{q_0\}$.

2. For each state $q_1 \in Q_{in}$, either a transition $\delta(q_1, a)$ or acceptance $(q_1, a) \in F$ should be defined. If the transition $\delta(q_1, a) = (q_2, d)$ for some $q_2$ and $d$ is defined, then this transition should be encoded, that is $q_2 \in Q_{out,d}$ should hold.

   Indeed, if the automaton $A$ in the accepting computation visits some node in the state $q_1$, then it either accepts, or makes a transition, it cannot reject.

3. For each $d \in D_a$ and for each state $q_2 \in Q_{out,d}$, there must be a way to move from the current node in the state $q_2$ in the direction $d$. That is, there should exist a state $q_1 \in Q_{in}$, with $\delta(q_1, a) = (q_2, d)$.

4. For every two distinct states $p_1, q_1 \in Q_1$, with the transitions at the label $a$ defined, the transitions should be distinct: $\delta(p_1, a) \neq \delta(q_1, a)$.

   Indeed, the automaton in the accepting computation cannot come to the same configuration twice, otherwise it loops.

The signature $S'$ has $k4^n$ directions. There are at most $m4^{kn}$ node labels, as in a label $(a, E)$ there are $m$ ways to choose an old label $a$, and $4^n$ ways to choose sets $Q_{in,d}$ and $Q_{out,d}$ for each direction $d \in D_a$.

All the directions with their opposite directions, and all the labels from $\widehat{\Sigma}'$ with their sets of directions can be written down in time linear in their length, that is, exponential in the length of the input. Checking whether a label $(a, E) \in \widehat{\Sigma}'$ satisfies all conditions, can be done in linear time in the length of the label.

The signature $S'$ has been constructed, and it remains to prove the correspondence between the graphs over $S$ accepted by the automaton $A$, and all the graphs over $S'$, and to construct the functions $f$ and $g$ which define this correspondence.

Let the automaton $A$ accept some graph $G = (V, v_0, +, \lambda)$ over the signature $S$. Then, the graph $f(G) = G' = (V', v_0', +, \lambda')$ over the signature $S'$ is constructed as follows.

- The set of nodes and the initial node are the same: $V' = V$, $v_0' = v_0$.

- The edges in $G'$ are the same as in $G$, but with additional information encoded in the directions. Let some edge $e$ with directions $(d, -d)$ connect the nodes $u$ and $v$ in the graph $G$, that is, $u + d = v$ in $G$. Let $Q_{in} \subseteq Q$ be a set of states in which the automaton $A$ in its computation comes to the node $u$ from the node $v$ by the edge $e$, let $Q_{out} \subseteq Q$ be a set of states, in which the automaton $A$ arrives to the node $v$ from the node $u$ by the edge $e$. Then, the corresponding edge in $G'$ is defined by $u + (d, Q_{in}, Q_{out}) = v$ and $v + (-d, Q_{out}, Q_{in}) = u$. These are all edges in $G'$.

- The node labels in $G'$ are the node labels from $G$, but with added information on the new directions. Let a node $v$ in $G$ have label $a$, and accordingly edges in directions

from $D_a$. These directions in the graph $G'$ are augmented with the information about the automaton's moves, forming the set $E$ of new directions. Then the node label of the node $v$ in the graph $G'$ is $(a, E)$. The label $(a, E)$ is in $\Sigma'$, because it encodes the moves of the automaton in the accepting computation (and only labels encoding situations impossible in accepting computations were not included in $\Sigma'$). The node $v$ has edges in directions from $E = D'_{(a,E)}$. And only the initial node has an initial label, because the new labels' being initial depends only on the component $a$ of $(a, E)$.

Now it remains to check, that each graph over $S'$ corresponds to some graph over $S$ that is accepted by $A$.

What is the general form of a graph $G'$ over $S'$? In the first components of directions and node labels, it encodes some graph $g(G') = G$ over $S$ (and this is a definition of $g$). Then $g(f(G)) = G$ by definition. The other components of directions and node labels encode some information about moves of the automaton. It will be shown that all moves from the computation of the automaton $A$ on a graph $G$ must be encoded, and that looping or rejecting cannot be encoded. Then, for each graph $G'$ over $S'$, the corresponding graph $G = g(G')$ must be accepted by the automaton $A$. Note that, besides all moves from the accepting computation, the graph $G'$ may additionally encode some cycles of transitions that do not intersect with the accepting computation. So an accepted graph $G$ over $S$ may have several pre-images $G'$, such that $g(G') = G$.

It remains to prove that each graph $G'$ over the signature $S'$ must encode all moves the automaton $A$ makes in its computation on the graph $G = g(G')$, and possibly some moves not in this computation, and that the computation of $A$ on $G$ must be accepting.

Fix a graph $G'$ over the signature $S'$, let $G = g(G')$, and let $C = C_0, C_1, \ldots, C_N$ be the computation of the automaton $A$ on the graph $G$, where $C_N$ is the last configuration, or $N = \infty$ if the automaton loops. It should be proved that $C$ is accepting and is encoded in $G'$.

This is proved by induction on $i$ that either the configuration $C_i$ is accepting, or the next configuration $C_{i+1}$ exists and it is different from all previous configurations, and the move from configuration $C_i$ to $C_{i+1}$ is encoded in $G'$.

Let $i \in \{0, 1, 2, \ldots, N\}$, and let the claim be proved for all $j < i$.

Denote the $i$-th configuration by $(q, v)$. Let $(a, E)$ be the label of the node $v$ in $G'$. Then, one can define $Q_{in}$ for the label $(a, E)$ as in the conditions on $\Sigma'$. If $i = 0$, then $a \in \Sigma_0$ and $q = q_0 \in Q_{in}$. Otherwise, the move from $C_{i-1}$ to $C_i$ is encoded in $G'$, and $q \in Q_{in}$ as well. Then, by the second condition, as $q \in Q_{in}$, either $(q, a) \in F$, or $\delta(q, a) = (r, d)$, for some $r \in Q$, $d \in D$, and the transition is encoded as $r \in Q_{out,d}$. In the latter case $r$ will be in $Q_{in,-d}$ for the node $v + d$. It remains to check that $C_i$ is different from all previous configurations. If $i = 0$, then this is true. Now, let $(p, u)$ be the previous configuration, with $\delta(p, \lambda(u)) = (q, d)$ and with $u + d = v$. Then $q \in Q_{in,-d}$ for the label $(a, E)$ of the node $v$. The first condition gives that the automaton could not have entered the node $v$ in the state $q$ from another direction earlier in the computation, and that $(q, v)$ cannot be the initial configuration. And if the previous direction is the same, then the 4-th condition prohibits entering $(q, v)$ earlier from a previous state other than $p$. Then, only $(p, u)$ can be the previous configuration for $(q, v)$, and, by the induction hypothesis, $(p, u)$ is unique in $C_0, \ldots, C_{i-1}$. Then, $(q, v)$ is unique in $C_0, \ldots, C_i$.

Thus, the computation of $A$ on $G$ is encoded in $G'$, this computation cannot loop,

cannot reject, so it is accepting. □

Using Theorem 18 that reduces graph-walking automata to signatures, one can solve the non-emptiness problem for graph-walking automata in nondeterministic exponential time.

**Corollary 4.** *The problem of whether a given graph-walking automaton accepts at least one graph is in NEXP.*

*Proof.* First, the algorithm from Theorem 18 is applied to a given signature $S$ and to a given graph-walking automaton $A$ over this signature, and it constructs a signature $S'$, such that there exist functions $f\colon L(A) \to L(S')$ and $g\colon L(S') \to L(A)$. Then, $L(A)$ is non-empty if and only if $L(S')$ is non-empty. The size of the signature $S'$ is exponential in the size of $S$ and $A$, and this signature is constructed in exponential time. Checking whether $L(S')$ is non-empty can be done in nondeterministic polynomial time in the size of $S'$, that is, in nondeterministic exponential time in the sum of sizes of $S$ and $A$. □

Actually, the non-emptiness problem for graph-walking automata is NEXP-complete, that will be proved in Section 4.5.

An upper bound on the number of nodes in the minimal graph accepted by a graph-walking automaton can be derived from the analogous bound for signatures.

**Corollary 5.** *Let $S = (D, -, \Sigma, \Sigma_0, (D_a)_{a\in\Sigma})$ be a signature with $k \geqslant 2$ directions, with $m$ node labels, and with $|D_a| \geqslant 1$ for each $a \in \Sigma$. Let $A = (Q, q_0, F, \delta)$ be a graph-walking automaton over $S$ with $n$ states. Then, if $A$ accepts at least one graph, then the number of nodes in the smallest accepted graph is at most $m4^{n(k+1)}k^{k4^n-1}$.*

*Proof.* Let $A$ accept at least one graph. By Theorem 18, there is a signature $S'$, and functions $f\colon L(A) \to L(S')$ and $g\colon L(S') \to L(A)$ that do not change the number of nodes in a graph. So the minimal number of nodes for graphs over $S$ accepted by $A$ is equal to the minimal number of nodes in graphs over $S'$.

The signature $S'$ has $k4^n$ directions, at most $m4^{kn}$ node labels, and the maximum degree of a node at most $k$. The latter is because $f$ preserves edge structure of graphs. Then, by Theorem 14, the minimal graph over the signature $S'$ has the number of nodes at most

$$m4^{nk}k4^n \min\left\{\left(\frac{1}{2}k4^n\right)^{\frac{k4^n}{2}}, k^{k4^n-2}\right\} \leqslant m4^{nk}k4^n k^{k4^n-2} = m4^{n(k+1)}k^{k4^n-1}.$$

□

# 4.5 Computational complexity of emptiness problems

It has been proved that the non-emptiness problems for signatures and for star automata are both in NP, and that the non-emptiness problem for graph-walking automata is in NEXP. In this section, all these problems are proved to be complete in their complexity classes.

NP-hardness of the non-emptiness problem for signatures is proved by a reduction of graph 3-colourability to this problem.

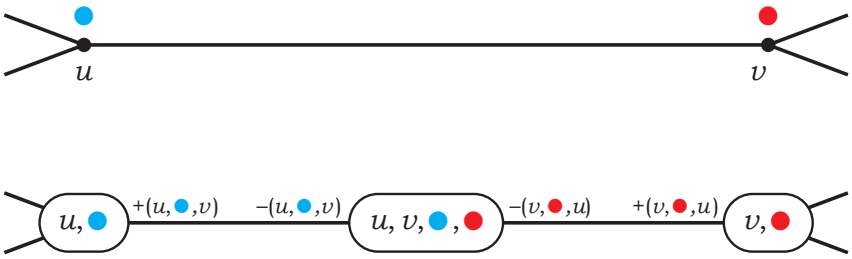


Figure 4.3: From 3-colourability to signature non-emptiness: mapping a graph with a colouring to a graph over a signature.

**Theorem 19.** *The problem of whether there is at least one graph over a given signature is NP-hard.*

*Proof.* The 3-colourability problem for a connected graph $G = (V, E)$ is to check whether its nodes can be coloured in $\{1, 2, 3\}$, so that every edge connects differently coloured nodes.

For an input graph $G$, one should construct such a signature $S_G$ in polynomial time, that there exists a graph over $S_G$ if and only if the graph $G$ can be coloured correctly.

The signature $S_G$ will be constructed so, that graphs over it correspond to correct colourings of the graph $G = (V, E)$.

Nodes of $G$ can have any of the three colours, and for each node and for each colour there is a corresponding node label. Furthermore, for every edge with two distinct colours on its ends, there is a separate node label representing this edge with these colours, that is, an unordered pair of two coloured nodes.

$$\Sigma = \{\ (v,i)\ \mid v \in V,\ i = 1,2,3\ \} \cup \{\ \{(u,i),\ (v,j)\}\ \mid (u,v) \in E,\ i,j \in \{1,2,3\}, i \neq j\ \}.$$

The condition of the colouring to be correct is checked by not having labels of the form $\{(u,i),\ (v,i)\}$, representing edges with the same colour at both ends.

Fix any node $v_0 \in V$, and let all labels $(v_0, i)$, with $i = 1, 2, 3$, be initial.

The set of directions is organized so that for every edge $(u, v)$ in the graph $G$, node labels $(u, i)$ and $(v, j)$, which correspond to the nodes $u$ and $v$ in the graph $G$, would require a connection through an intermediate node that corresponds to the edge $(u, v)$ in $G$, and which gathers information on the colours of the nodes $u$ and $v$.

$$D = \{\ \pm(u,i,v)\ \mid u,v \in V,\ (u,v) \in E,\ i = 1,2,3\ \}.$$

The opposite direction to $+(u, i, v)$ is given by $-(u, i, v)$, for all $u, v \in V$ with $(u, v) \in E$, and for all $i = 1, 2, 3$.

Each node of a graph over this signature which represents one of the nodes of $G$ should be connected with the nodes representing all the edges coming out of this node.

$$D_{(u,i)} = \{\ +(u,i,v)\ \mid v \in V,\ (u,v) \in E\ \},\ \text{for all } u \in V,\ i = 1,2,3.$$

$$D_{\{(u,i),\,(v,j)\}} = \{-(u,i,v), -(v,j,u)\},\ \text{for all } u,v \in V,\ (u,v) \in E,\ i,j \in \{1,2,3\},\ i \neq j$$

It remains to prove that the signature $S_G$ is as desired, that is, there is a graph over $S_G$ if and only if there is a correct 3-colouring of $G$.

First of all, if a coloring $c\colon V \to \{1,2,3\}$ exists, then a graph $G_c$ over $S_G$ representing this colouring is constructed with the set of nodes $V \cup E$, where each node $v \in V$ has label $(v, c(v))$, each node $(u,v) \in E$ has label $\{(u,c(u)), (v,c(v))\}$. For every edge $(u,v) \in E$ in the graph $G$, the graph $G_c$ has edges from $u$ to $(u,v)$ and from $(u,v)$ to $v$, with the appropriate directions, as illustrated in Figure 4.3.

Conversely, let $\widehat{G}$ be any graph over the signature $S_G$. It is claimed that in this case there exists a correct 3-colouring of $G$, and moreover, $\widehat{G} = G_c$ for some correct 3-colouring $c$ of $G$.

First, it is proved that for each node $v \in V$ of the graph $G$, there is exactly one node in $\widehat{G}$ with a label of the form $(v,i)$, for some $i$. Consider the shortest simple path from $v_0$ to $v$ in $G$ (it exists because $G$ is connected); the proof is by induction on the length of this path. The base case is a path of length 0: here the node corresponding to $v_0$ exists because $\widehat{G}$ must have an initial node, and it is unique because the initial node is unique. For the induction step, let $u$ be the next to the last node on the path, with $(u,v) \in E$. By the induction hypothesis, in $\widehat{G}$, there is a unique node of the form $(u,i)$, for some $i$. This node emits a unique edge in the direction $+(u,i,v)$, which must lead to a node labelled with $\{(u,i), (v,j)\}$, for some $j$, which in turn emits a unique edge in the direction $-(v,j,u)$ that ends in a node labelled with $(v,j)$—so this node exists. If there were another node in $\widehat{G}$ labelled with $(v,k)$, for any $k$, then, by the same reasoning, it would be connected to some node labelled with $(u,\ell)$ through some intermediate node; this node must be the same as the above node labelled with $(u,i)$, because such a node is unique. However, there is a unique path simulating the edge $(u,v)$, hence this node labelled with $(v,k)$ must coincide with the above node labelled with $(v,j)$.

Therefore, $\widehat{G}$ has the set of nodes $V \cup E$, which replicates the structure of $G$, with every edge split by an intermediate node. Then, it must be $G_c$ for some colouring $c$. This colouring is correct, because each intermediate node checks that the colours at both ends of the corresponding edge are distinct. Then, correct colourings of the graph $G$ correspond to graphs over $S_G$.

Note that the intermediate nodes that split the edges of $G$ are necessary, because node labels cannot accumulate information on the colours of all the neighbours of a node, as this would require an exponential number of node labels. □

The non-emptiness problem for star automata is NP-complete as well. Its membership in NP was established above, and its NP-hardness follows from the NP-hardness of non-emptiness of signatures.

**Theorem 20.** *The problem of checking whether a given star automaton accepts at least one graph is NP-hard.*

*Proof.* Non-emptiness for signatures was proved in Theorem 19 to be NP-hard. Now the NP-hardness of the non-emptiness problem for star automata is proved by reducing the non-emptiness problem for signatures to it, as follows.

Let $S$ be a given signature. Consider the automaton $A_*$ over it, that has one state, and, for each node label, has a star with this state at the centre and with this state at all rays. This star automaton accepts all graphs, so its non-emptiness is equivalent to non-emptiness of the signature $S$. And this automaton $A_*$ has size polynomial in the size of $S$. □

Now it is time to prove the NEXP-completeness of the non-emptiness problem for graph-walking automata. It was proved in Corollary 4, that this problem is in NEXP. For NEXP-hardness it will be proved that a signature and a graph-walking automaton can define a set of graphs containing a square grid of size exponential in the number of states of the automaton and in the size of the signature. And then a nondeterministic Turing machine working in exponential time will be simulated on such grids.

**Theorem 21.** *The problem of whether there is at least one graph accepted by a given graph-walking automaton is NEXP-hard.*

*Proof.* Fix some NEXP-complete problem and some nondeterministic Turing machine $M$ that solves this problem in exponential time. It can be assumed that the Turing machine is one-tape with the tape infinite to the right, and that the machine never moves to the left from the first position of the tape, in which an input string begins. The number of states, the number of transitions in the transition function, the sizes of the input aphabet and of the work alphabet are constant, as the Turing machine $M$ is fixed.

The problem whether a given string $w$ over the input alphabet is accepted by the Turing machine $M$ is NEXP-complete. So to prove the theorem it is enough to reduce in polynomial time this problem about $M$ to the non-emptiness problem for a graph-walking automaton. That is, such a deterministic polynomial-time algorithm is needed, that for a given string $w$ it constructs a signature $S_w$ and a graph-walking automaton $A_w$ so that a graph accepted by $A_w$ will exist if and only if there exists an accepting computation of the machine $M$ on the string $w$.

Let $f\colon \mathbb{N} \to \mathbb{N}$ be a polynomial-time computable function that, for each length $\ell$ of an input string, gives a number $f(\ell)$, bounded by a polynomial in $\ell$, such that $f(\ell) \geqslant \max\{\ell, 2\}$, and that the Turing machine $M$ halts on every string of length at most $\ell$ in not more than $2^{f(\ell)} - 1$ steps. Then, each computation of $M$ on each string of length at most $\ell$ can be written on a grid of length $2^{f(\ell)} \times 2^{f(\ell)}$.

The signature $S_w = (D, -, \Sigma, \Sigma_0, (D_a)_{a\in\Sigma})$ depends only on the length of $w$ and is constructed as follows.

Let $n = f(|w|)$, so that each computation of $M$ on a string $w$ can be written on a grid of size $2^n \times 2^n$; the number $n = f(|w|)$ can be computed in polynomial time and is polynomial in the length of $w$.

The signature $S_w$ is composed of three parts: $D = D_1 \cup D_2 \cup D_3$, $\Sigma = \Sigma_1 \cup \Sigma_2 \cup \Sigma_3$, all sets here are disjoint. And for each node label $a \in \Sigma_i$, it should hold that $D_a \subseteq D_i$, for $i \in \{1, 2, 3\}$. In every graph over $S_w$ all nodes are divided into three sets: $V = V_1 \cup V_2 \cup V_3$, where $V_i$ consists of the nodes with labels in $\Sigma_i$, for $i = 1, 2, 3$. There are two special pairs of opposite directions: $+d \in D_1$ and $-d \in D_2$, and $+d' \in D_2$ and $-d' \in D_3$. For every other direction, the opposite direction lies in the same set. Thus, nodes in $V_1$ and nodes in $V_2$ can be connected only by $(+d, -d)$-edges; similarly, nodes in $V_2$ and nodes in $V_3$ can be connected only by $(+d', -d')$-edges. A node from $V_1$ and a node from $V_3$ cannot be connected with an edge.

The idea is that nodes with labels in $\Sigma_2$ form a grid on which the Turing machine working on $w$ will be simulated. Each node label from $\Sigma_2$ will have both directions $-d$ and $+d'$. Labels from $\Sigma_1$ will allow the nodes in $V_1$ to form only a full binary tree of height $2n$ that emits exactly $2^{2n}$ edges in the direction $+d$ from its leaves, thus ensuring that in every graph the number of nodes in $V_2$ is exactly $2^{2n}$. Labels from $\Sigma_3$ will be used to attach a chain of length $2n$ to every node with label in $\Sigma_2$, with the chain consisting

of zeros and ones. The automaton $A_w$ will check that nodes in $V_2$ form a $2^n \times 2^n$ grid, and that chains attached to these nodes correctly encode the row number and the column number in the grid for each node. Next, the automaton $A_w$ will check that some accepting computation of the Turing machine $M$ on the string $w$ is encoded on the grid. Figure 4.4 shows a graph over some signature $S_w$ with $n = 2$, that defines a correct grid on nodes with labels in $\Sigma_2$.

The only initial node label in the signature $S_w$ is $a_0 \in \Sigma_1$. The first part $\Sigma_1$ and $D_1$ should be defined so that the nodes with labels in $\Sigma_1$ can form only one graph: a full binary tree of height $2n$ with $2^{2n}$ leaves. The set of node labels is $\Sigma_1 = \{a_0, a_1, b_1, a_2, b_2, \ldots, a_{2n}, b_{2n}\}$, and the set of directions is $D_1 = \{\pm\ell_1, \pm r_1, \pm\ell_2, \pm r_2, \ldots, \pm\ell_{2n}, \pm r_{2n}\} \cup \{+d\}$. Here the label $a_0$ is initial, it is used for the root of a tree (level 0), the labels $a_i$ and $b_i$ are used for left and right children of the $i$-th level. The node label $a_0$ has the set of directions $D_{a_0} = \{+\ell_1, +r_1\}$, that is, the root has two edges to the two nodes of level 1. Labels $a_i$ and $b_i$, for $i \in \{1, \ldots, 2n-1\}$, have the sets of directions $D_{a_i} = \{-\ell_i, +\ell_{i+1}, +r_{i+1}\}$ and $D_{b_i} = \{-r_i, +\ell_{i+1}, +r_{i+1}\}$. So the $i$-th level generates twice as many nodes on level $i+1$. The node labels of the last level $2n$ (for the leaves of the tree) have sets of directions $D_{a_{2n}} = \{-\ell_{2n}, +d\}$ and $D_{b_{2n}} = \{-r_{2n}, +d\}$, that is, each leaf emits one edge in the direction $+d$, which is used for connection with nodes in $V_2$.

Thus, in every graph over the signature $S$ the initial node is labelled with $a_0 \in \Sigma_1$ and all nodes in $V_1$ form a full binary tree with $2^{2n}$ leaves and each leaf emits an edge in the direction $+d$.

The part $\Sigma_3$, $D_3$ is constructed to allow only chains of nodes of length $2n$ with one direction $-d'$ in each chain, with zeros and ones in nodes. This part of the signature is defined by $\Sigma_3 = \{0_1, \ldots, 0_{2n}\} \cup \{1_1, \ldots, 1_{2n}\}$, $D_3 = \{\pm d_1, \ldots, \pm d_{2n-1}\} \cup \{-d'\}$. And $D_{0_1} = D_{1_1} = \{-d', +d_1\}$; $D_{0_i} = D_{1_i} = \{-d_{i-1}, +d_i\}$, for $i \in \{1, \ldots, 2n-1\}$; and $D_{0_{2n}} = D_{1_{2n}} = \{-d_{2n-1}\}$.

Then, each node in $V_2$ has a chain attached to it in the direction $+d'$. Every such chain consists of nodes with labels in $\Sigma_3$, has length $2n$ and encodes a number from 0 to $2^{2n} - 1$ in a sequence of zeros and ones in nodes. Let some node $v$ in a graph have a label in $\Sigma_2$. Then, the *coordinates* of $v$ are the pair of numbers $(i_v, j_v)$, for $i_v, j_v \in \{0, \ldots, 2^n - 1\}$, where the number $i_v$ is defined by the first $n$ bits in the chain of nodes in $V_3$ attached to $v$, and the number $j_v$ is defined by the second $n$ bits. The number $i_v$ is meant to be the number of the row in the grid where $v$ is located, and $j_v$ is meant to be the number of the column. Note that the coordinates of the node $v$ are by definition just a pair of numbers, encoded in a chain, even if these numbers do not correspond to the actual position of the node $v$ in a grid.

Now to the main part of the signature: $\Sigma_2$ and $D_2$. There are 6 directions in $D_2$: two of them, $-d$ and $+d'$, are used for connection with $V_1$ and $V_3$, and 4 directions are used for a grid: $\pm 1$ are horizontal ($+1$ is right, $-1$ is left), and $\pm 2$ are vertical ($+2$ is up, $-2$ is down), so $D_2 = \{\pm 1, \pm 2\} \cup \{-d, +d'\}$. The set of node labels is $\Sigma_2 = \mathrm{Pos} \times \mathrm{Alph} \times \mathrm{Head}$, that is, each node label in $\Sigma_2$ is of the form $(pos, alph, head)$. The first component $pos$ gives the type of a position of a node in a grid: in one of 4 corners, on the side or in the

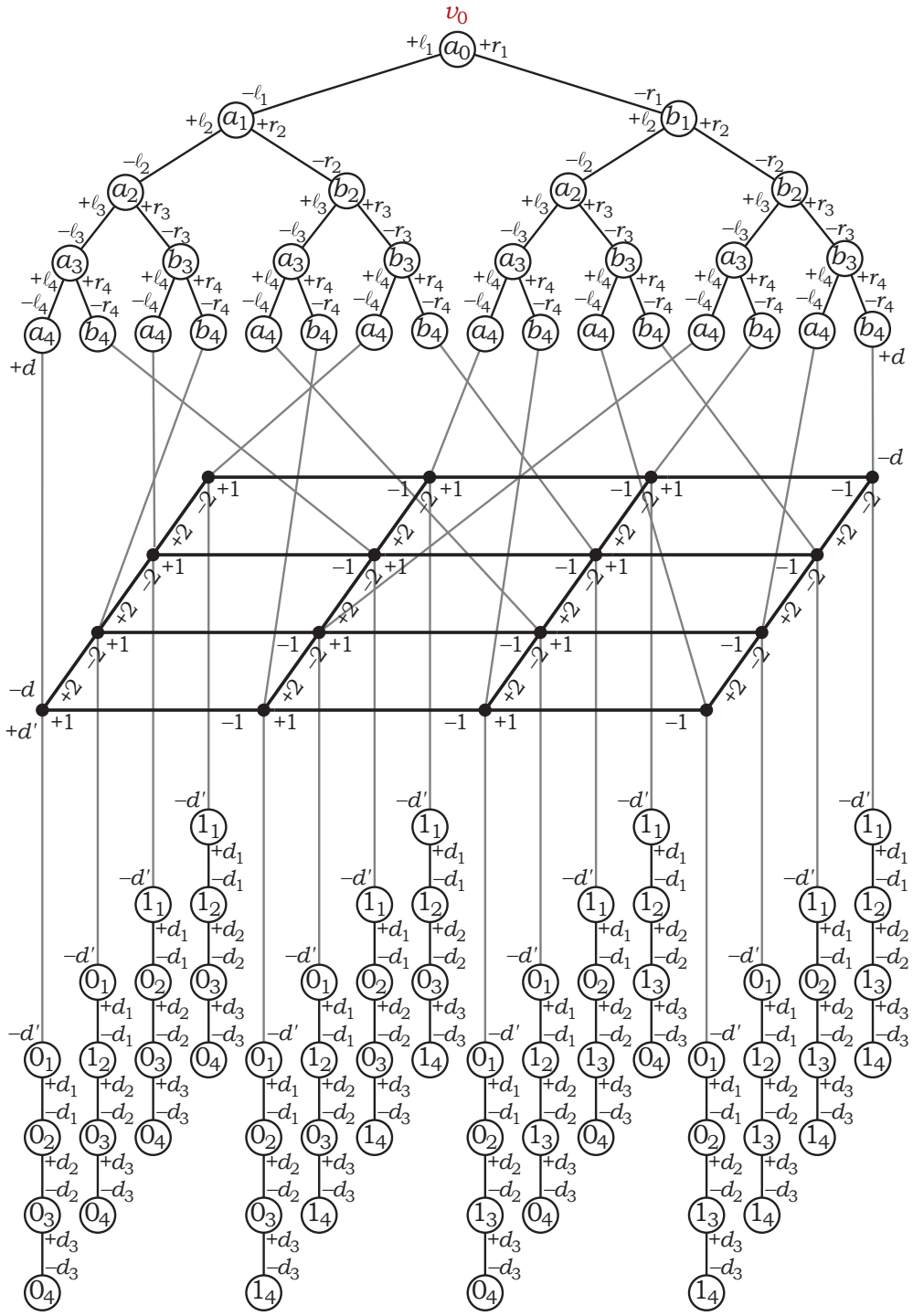


Figure 4.4: A graph that defines a correct $2^n \times 2^n$ grid, for $n = 2$. The graph has three levels: the tree on the nodes in $V_1$ at the top, the grid on the nodes in $V_2$ in the middle, and chains on nodes in $V_3$ at the bottom, which encode the coordinates of nodes in the grid (the upper two bits encode the row number, and the lower two bits encode the column number).

centre. So there are 9 variants of the first component of a node label:

$$\begin{aligned}\mathrm{Pos} = \{&\mathrm{LU}, \mathrm{CU}, \mathrm{RU},\\ &\mathrm{LC}, \mathrm{CC}, \mathrm{RC},\\ &\mathrm{LD}, \mathrm{CD}, \mathrm{RD}\},\end{aligned}$$

where the first letter of *pos* gives the type of horizontal position ($L$, $C$ or $R$), and the second letter gives the type of vertical position ($D$, $C$ or $U$). The set of directions $D_a$ for each node label $a \in \Sigma_2$ depends only on the component *pos* of the label $a$: the directions $-d$ and $+d'$ are always in $D_a$; the direction $+1$ is in $D_a$ if the node is not at the right border of a grid, that is, if $pos \notin \{RU, RC, RD\}$; the direction $-1$ is in $D_a$ if and only if $pos \notin \{LU, LC, LD\}$; similarly $+2 \in D_a$ if and only if $pos \notin \{LU, CU, RU\}$; and $-2 \in D_a$ if and only if $pos \notin \{LD, CD, RD\}$.

The components *alph* and *head* of node labels in $\Sigma_2$ will be used for simulating configurations of the Turing machine $M$ on rows of a grid. Let $\Gamma$ be the work alphabet of $M$, it contains the input alphabet, the new blank symbol and maybe some other symbols; let $Q$ be a finite set of states of the Turing machine. Then, $\mathrm{Alph} = \Gamma$, that is, the component *alph* gives one of the symbols in the work alphabet of $M$, and $\mathrm{Head} = Q \cup \{0\}$, where $0 \notin Q$, that is, the component *head* gives either a state of the Turing machine $M$ if the head is simulated at the current position, or $head = 0$ if there is no head in this position.

This signature $S_w$ is constructed in time linear in $n$.

Now a graph-walking automaton $A_w$ over the signature $S_w$ should be constructed, so that it accepts only graphs, in which nodes in $V_2$ form a correct grid, and the components *alph* and *head* of the labels in these nodes encode a correct accepting computation of the Turing machine $M$ on the string $w$. The work of the automaton $A_w$ on a graph is divided into two phases: checking the grid and checking the encoding of the Turing machine's computation on that grid.

In the first phase the automaton does not distinguish the components *alph* and *head* in labels in $\Sigma_2$, its actions on a node labelled with $(pos, alph, head) \in \Sigma_2$ depend only on the component *pos*.

The goal of the first phase is to check that nodes with labels in $\Sigma_2$ form a $2^n \times 2^n$ grid on directions $\pm 1$ and $\pm 2$, and that the coordinates $(i_v, j_v)$ of each node $v$ in $V_2$ are numbers of its row and its column in a grid. For convenience, the automaton also checks that the leftmost path in the tree on nodes in $V_1$ leads to a node in $V_2$ with coordinates $(0, 0)$. If all these conditions hold for a graph, then this graph is said to define a correct grid.

The automaton checks whether a graph defines a correct grid as follows.

1. At the beginning, the automaton checks that the leftmost path in the tree on nodes with labels in $\Sigma_1$ leads to a node with a label in $\Sigma_2$ that has coordinates $(0, 0)$. The automaton starts at the initial node at the root of the tree, then it moves to the left child until it comes to a node with label in $\Sigma_2$. Then it checks that all nodes in the attached chain contain zeros. This can be done with a constant number of states.

2. The automaton checks, for each node $v$ with label in $\Sigma_2$, that the component *pos* of the label agrees with coordinates $(i_v, j_v)$ given in the chain of nodes from $V_3$ attached

to the node $v$. For that, it should be checked that $pos = XY$, where

$$X = \begin{cases} L & \text{if } j_v = 0 \\ C & \text{if } 0 < j_v < 2^n - 1 \\ R & \text{if } j_v = 2^n - 1 \end{cases} \qquad Y = \begin{cases} D & \text{if } i_v = 0 \\ C & \text{if } 0 < i_v < 2^n - 1 \\ U & \text{if } i_v = 2^n - 1 \end{cases}$$

.

When the automaton visits some node $v \in V_2$, it can check this condition for the node $v$ using a constant number of states and return to the node. Indeed, it needs just to check several conditions of the form that all bits of the first or the second $n$ bits of a chain are all zeros or are all ones.

To do such a check for each node in $V_2$, the automaton needs to visit somehow all nodes in $V_2$. This can be done by traversing the tree on nodes in $V_1$. This tree can be traversed with a constant number of states. The leaves in this tree correspond to nodes in $V_2$, each leaf is connected by a $(+d, -d)$ edge to some node in $V_2$, and each node in $V_2$ is connected to some leaf. Thus, the automaton checks for each leaf in a tree that its neighbour in $V_2$ has the component *pos* agree with the coordinates. This can be done using a constant number of states.

3. Then the automaton checks that directions $\pm 1, \pm 2$ in the grid lead to correct nodes. That is, for each node $v$ in $V_2$ with coordinates $(i_v, j_v)$, the following conditions must hold. If an edge in the direction $-1$ exists ($j_v > 0$), then it should lead to a node with coordinates $(i_v, j_v - 1)$. If an edge by $+1$ exists ($j_v < 2^n - 1$), then it should lead to a node with coordinates $(i_v, j_v + 1)$. Similarly, the direction $+2$ must increase the coordinate $i_v$, and the direction $-2$ must decrease it.

   When the automaton visits some node $v$ in $V_2$, it can check these conditions using $O(n)$ states and return to the node $v$. Indeed, to check the equality of two vectors of length $n$ contaning 0s and 1s, the automaton can compare them bit by bit remembering only the position of the current bit in a vector and the value of this bit. To check that the number encoded in the first vector is greater by 1 than the number encoded in the second vector, the automaton can check that the binary representations of the vectors are of the form $x10^i$ and $x01^i$, with $x \in \{0,1\}^*$ and $i \geqslant 0$, and this can be checked bit by bit.

   To make these checks for all nodes in $V_2$ the automaton traverses the tree on the nodes in $V_1$ as at the previous step.

4. If the automaton did not reject at the previous steps, then it returns to the node with coordinates $(0, 0)$ and starts the second phase.

If the automaton rejects at the first phase, then the graph does not define a correct grid. It is claimed that the checks the automaton makes are sufficient, that is, that if the automaton starts the second phase, then the graph defines a correct grid. Let the automaton start the second phase on some graph $G$.

First, it is shown that all nodes with labels in $\Sigma_2$ have distinct coordinates and that every pair of coordinates $(i, j)$, for $i, j \in \{0, \ldots, 2^n - 1\}$, occurs somewhere.

The node with coordinates $(0, 0)$ exists because such a node is on the leftmost path. For each node with some coordinates $(i, j)$, the automaton has checked that its neighbours

in directions $\pm 1, \pm 2$ exist and have coordinates $(i, j+1)$, $(i, j-1)$, $(i+1, j)$, $(i-1, j)$, as long as these coordinates are between 0 and $2^n - 1$. Then, for all $i, j \in \{0, \ldots, 2^n - 1\}$, there is a node in $V_2$ with coordinates $(i, j)$. As the tree on nodes in $V_1$ is defined uniquely, $|V_2| = 2^{2n}$ in every graph. So the node with each pair of coordinates is unique.

Note that the automaton has no way to distinguish a node from its copy locally, so it is important that counting arguments give uniqueness to each pair of coordinates.

Then, as a node with each pair of coordinates exists and is unique, and coordinates increase or decrease along the directions in the grid, the graph defines a correct grid.

The states and transitions used by the automaton in the first phase can be written down in time quadratic in $n$, as both the number of states and the number of node labels in the signature are linear in $n$.

In the second phase, the automaton checks that some accepting computation of the Turing machine $M$ on the string $w$ is encoded in the grid.

The automaton should check that the initial row encodes the initial configuration of the Turing machine $M$ on the string $w$, that the next row encodes one of possible next configurations, and so on, up to an accepting configuration. Rows after the accepting configuration are allowed to contain anything.

How are configurations encoded in rows? The Turing machine $M$ works in exponential time, and the number $n$ was chosen so that every computation on $w$ contains at most $2^n - 1$ steps, and that $|w| \leqslant n$. Thus, the head of the Turing machine never visits positions beyond $2^n - 1$ on the tape, and during the computation the symbols at these positions are blank symbols. So the tape contents in a configuration can be thought of as a string of length $2^n$. This string is encoded in the nodes of a row in the components *alph* of node labels, one symbol of the string per node. The position of the head is encoded by having the component *head* non-zero only in one node; in this node, the component *head* encodes a state of the Turing machine.

The automaton works in the second phase as follows.

1. The automaton $A_w$ starts the second phase on a graph at the node with coordinates $(0, 0)$, and the graph is known to define a correct grid of size $2^n \times 2^n$.

2. First, the automaton checks the encoding of the initial configuration. It goes through the first $|w|$ nodes in the first row remembering in a state the number of moves $j$ it made, and for each node it checks that the component *alph* of the node label is the $j$-th symbol of $w$. Then it continues moving to the right using one state for that, and checking that the components *alph* in all other nodes in the first row contain blank symbols. While moving from $(0, 0)$ to $(0, 2^n - 1)$ the automaton additionally checks that in the node $(0, 0)$ the component *head* contains one of the initial states of the Turing machine, and that in all other nodes of the first row the component *head* of the label is 0.

3. For each row $i \in \{0, \ldots, 2^n - 1\}$, starting from the row $i = 0$, the automaton makes the following two actions.

   First, the automaton checks whether the current configuration is accepting. It finds the node in which the head is encoded, and if $(\mathit{head}, \mathit{alph})$ is an accepting pair of $M$, then the automaton immediately accepts.

   If the configuration encoded in the $i$-th row is not accepting, then the automaton checks that the next row encodes one of the possible next configurations. This check

can be done using a constant number of states as follows. In the neighbourhood of the head in the $i$-th row, the automaton checks that a transition is correctly made; elsewhere, the automaton checks that the tape symbols are unchanged, and no extra heads appear. Once the check is complete, the automaton moves to the next row.

Working as described above, the automaton accepts a graph in the second phase if and only if one of the accepting computations of $M$ on $w$ is encoded on the grid, and otherwise it rejects. The automaton $A_w$ can be constructed in time polynomial in $n$, and the NEXP-complete problem of whether the Turing machine $M$ accepts a given string $w$ or not is reduced to the problem of whether the graph-walking automaton $A_w$ over $S_w$ accepts at least one graph. Thus, non-emptiness for graph-walking automata is NEXP-hard. □

# Chapter 5

# Conclusion

In Chapter 2 the maximum length of the shortest accepted string for direction-determinate 2DFA has been determined precisely, whereas for 2DFA of the general form, a lower bound of the order $2^n$ has been established. The known upper bound on this length is of the order $4^n$. It should be noted that the computed values reported in Section 2.4 exceed the theoretical lower bound $\frac{3}{4} \cdot 2^n - 1$ proved in Section 2.3, and are much less than the known upper bound $\binom{2n}{n+1} - 1$. Thus, the bounds for 2DFA of the general form are still in need of improvement.

Another parameter to be refined is the size of the alphabet of the construction. Both constructions in Chapter 2 use an alphabet of exponential size. For a fixed alphabet, the maximum known length of shortest strings is $\Omega(1.275^n)$ [17]. Would it be possible to encode the construction in this thesis over a fixed alphabet to surpass this bound? What is the exact maximum length of shortest strings accepted by $n$-state 2DFAs over an $m$-symbol alphabet?

The maximum size of smallest accepted trees for tree automata was determined precisely in Chapter 3. But there is a gap between a lower bound $2^{\Omega(r \cdot 1.618^n)}$ and an upper bound $2^{O(rn \cdot 3.572^n)}$ on the maximum number of nodes in the smallest accepted trees for tree-walking automata with $n$ states, that work on trees with degree of a node at most $r$. What is an actual constant $c$ in a bound like $2^{\Theta(r \cdot c^n)}$ for tree-walking automata?

In Chapter 4, it has been shown that emptiness problems for star automata and for graph-walking automata are decidable. And the computational complexity classes for these problems were determined: the non-emptiness problem for star automata is NP-complete, whereas the non-emptiness problem for graph-walking automata is NEXP-complete. Table 5.1 compares these new results about automata on graphs with the previous results for similar automata on strings and on trees.

| | strings | trees | graphs |
|---|---|---|---|
| walking | (2DFA)<br>PSPACE-complete [16] | (DTWA)<br>EXP-complete [3] | (DGWA)<br>NEXP-complete (Cor 4, Thm 21) |
| tilings by edges/stars | (NFA)<br>NL-complete [14] | (tree automata)<br>P-complete [27] | (star automata)<br>NP-complete (Cor 2, Thm 20) |

Table 5.1: Complexity of the non-emptiness problem for different families of automata.

In Chapter 4, several upper bounds on the number of nodes in minimal accepted graphs have been obtained. Upper bounds have been proved for graph-walking automata

(Corollary 5), and for star automata (Corollary 3). Lower bounds proved for tree-walking and for tree automata in Chapter 3 also apply to graphs. Perhaps it would be possible to prove some better lower bounds using more complicated graphs than trees. Also the upper bounds given in Chapter 4 could be improved.

Star automata in this thesis are a special case of elementary acceptors of Thomas [26], they are elementary acceptors without conditions on the number of occurrences of each star. Is the emptiness problem for elementary acceptors of Thomas also decidable?

# Bibliography


[1] L. Alpoge, Th. Ang, L. Schaeffer, J. Shallit, "Decidability and shortest strings in formal languages", *Descriptional Complexity of Formal Systems* (DCFS 2011, Limburg, Germany, 25–27 July 2011), LNCS 6808, 55–67.

[2] J.-C. Birget, "Intersection and union of regular languages and state complexity", *Information Processing Letters*, 43 (1992), 185–190.

[3] M. Bojańczyk, "Tree-walking automata", LATA 2008, LNCS 5196, 1–2. Extended version available at `https://www.mimuw.edu.pl/~bojan/upload/conflataBojanczyk08.pdf`.

[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] L. Budach, "Automata and labyrinths", *Mathematische Nachrichten*, 86:1 (1978), 195–282.

[7] D. Chistikov, W. Czerwinski, P. Hofman, M. Pilipczuk, M. Wehar, "Shortest paths in one-counter systems", *FoSSaCS 2016: Foundations of Software Science and Computation Structures*, LNCS 9634, 462–478.

[8] E. Dobronravov, N. Dobronravov, A. Okhotin, "On the length of shortest strings accepted by two-way finite automata", *Fundamenta Informaticae*, 180:4 (2021), 315–331.

[9] K. Ellul, B. Krawetz, J. Shallit, M.-w. Wang, "Regular expressions: new results and open problems", *Journal of Automata, Languages and Combinatorics*, 10:4 (2005), 407–437.

[10] P. Fraigniaud, D. Ilcinkas, G. Peer, A. Pelc, D. Peleg, "Graph exploration by a finite automaton", *Theoretical Computer Science*, 345:2–3 (2005), 331–344.

[11] I. Glaister, J. Shallit, "A lower bound technique for the size of nondeterministic finite automata", *Information Processing Letters*, 59 (1996), 75–77.

[12] V. Geffert, A. Okhotin, "One-way simulation of two-way finite automata over small alphabets", *NCMA 2013* (Umeå, Sweden, 13–14 August 2013).

[13] J. Hadamard, "Résolution d'une question relative aux déterminants", *Bulletin des Sciences Mathématiques*, 17 (1893), 240–246.

[14] N. D. Jones, "Space bounded reducibility among combinatorial problems", *Journal of Computer and System Sciences*, 11:1 (1975), 68–85.

[15] C. A. Kapoutsis, "Removing bidirectionality from nondeterministic finite automata", *Mathematical Foundations of Computer Science* (MFCS 2005, Gdansk, Poland, 29 August–2 September 2005), LNCS 3618, 544–555.

[16] D. Kozen, "Lower bounds for natural proof systems", *FOCS 1977*, 254–266.

[17] S. Krymski, A. Okhotin, "Longer shortest strings in two-way finite automata", In: G. Jirásková, G. Pighizzini (Eds.), *Descriptional Complexity of Formal Systems*, LNCS 12442, 2020, 104–116.

[18] M. Kunc, A. Okhotin, "Reversibility of computations in graph-walking automata", *Information and Computation*, 275 (2020), article 104631.

[19] O. Martynova, "Complexity of the emptiness problem for graph-walking automata and for tilings with star subgraphs" arXiv:2212.02380 [cs.FL] (2022), submitted for publication.

[20] O. Martynova, A. Okhotin, "Lower bounds for graph-walking automata", *38th Annual Symposium on Theoretical Aspects of Computer Science* (STACS 2021, Saarbrücken, Germany, 16–19 March 2021), LIPIcs 187, 52:1–52:13.

[21] O. Martynova, A. Okhotin, "State complexity of union and intersection on graph-walking automata", *Descriptional Complexity of Formal Systems 2021*, LNCS 13037, 125–136.

[22] O. Martynova, A. Okhotin, "Shortest accepted strings for two-way finite automata: approaching the $2^n$ lower bound", *Descriptional Complexity of Formal Systems* (DCFS 2023, Potsdam, Germany, 4–6 July 2023), LNCS 13918, to appear.

[23] L. Pierre, "Rational indexes of generators of the cone of context-free languages", *Theoretical Computer Science*, 95:2 (1992), 279–305.

[24] H. A. Rollik, "Automaten in planaren Graphen", *Acta Informatica*, 13:3 (1980), 287–298.

[25] E. N. Shemetova, A. Okhotin, S. V. Grigorev, "Rational index of languages with bounded dimension of parse trees", *DLT 2022*, 263–273.

[26] W. Thomas, "On logics, tilings, and automata", *Automata, Languages and Programming* (ICALP 1991, Madrid, Spain, 8–12 July 1991), LNCS 510, 441–454.

[27] M. Veanes, "On computational complexity of basic decision problems of finite tree automata", Technical Report 133, Uppsala University, Computing Science Department, 1997.