Saint Petersburg State University

Olga Martynova

Diploma thesis

Bounds on the size of graph-walking automata

Educational program
Bachelor of Mathematics

Course code: 01.03.01 Mathematics
Code EP: CB.5000.2017

Supervisor:

Professor
Faculty of Mahematics and Computer Science
Saint Petersburg State University
Doctor of Philosophy
Alexander Okhotin

Reviewer:

Associate Professor
Department of Mathematics and Statistics
Masaryk University, Brno, Czech Republic
Doctor of Philosophy
Michal Kunc

Saint Petersburg
2021

Contents

1 Introduction

A graph-walking automaton moves over a labelled graph using a finite set of states and leaving no marks on the graph. This is a model of a robot finding its way in a maze. There is a classical result by Budach [3] that for every automaton there is a graph in which it cannot visit all nodes, see a modern proof by Fraigniaud et al. [6]. This work has influenced the current research on algorithms for graph traversal using various small-memory models. For example, Disser et al. [5] in their famous paper proved that if a graph-walking automaton is additionally equipped with $O(\log\log n)$ memory and $O(\log\log n)$ pebbles, then it can traverse every graph with n nodes, and this amount of resources is optimal.

Graph-walking automata (GWA) naturally generalize such important models as tree-walking automata (TWA) and two-way finite automata (2DFA). Two-way finite automata are a standard model in automata theory, and the complexity of their determinization remains a major open problem, notable for its connection to the L vs. NL problem in the complexity theory [9]. Tree-walking automata (TWA) have received particular attention in the last two decades, with important results on their expressive power established by Bojańczyk and Colcombet [1, 2].

It is known that GWA can be transformed to several important subclasses: to automata that halt on every input graph; to automata that return to the initial node in order to accept; to reversible automata. Such transformations have earlier been established for various automaton models, using a general method discovered by Sipser [19], who constructed a halting 2DFA that traverses the tree of computations of a given 2DFA leading to an accepting configuration, in search of an initial configuration. Later, Kondacs and Watrous [10] ensured the reversibility and optimized this construction for the number of states, motivated by the study of quantum automata. Sipser's idea has been adapted to proving that reversible space equals deterministic space [14], to making tree-walking automata halt [18], to complementing 2DFA [7], to making multi-head automata reversible [17], etc. Each transformation leads to a certain blow-up in the number of states, usually between linear and quadratic. No lower bounds on the transformation to halting have been established yet. For the transformation to reversible, a lower bound exists for the case of 2DFA [11], but it is quite far from the known upper bound.

For the general case of GWA, constructions of halting, returning and reversible automata were given by Kunc and Okhotin [12], who showed that an n-state GWA operating on graphs with k edge endpoint labels can be transformed to a returning GWA with $3nk$ states and to a reversible GWA with $6nk + 1$ states, which is always halting. Applied to special cases of GWA, such as TWA or multi-head automata, these generic constructions produce fewer states than the earlier specialized constructions.

The main result of this thesis are lower bounds on the complexity of these transformations. To begin with, the constructions by Kunc and Okhotin [12] are revisited in Section 3, and it turns out that the elements they are built of can be recombined more efficiently, resulting in improved upper bounds based on the existing methods. This way, the transformation to a returning GWA is improved to use $2nk + n$ states, the transformation to halting can use $2nk + 1$ states, and constructing a reversible GWA (which is both returning and halting) requires at most $4nk + 1$ states. With these improvements, each of these constructions is turn out to be asymptotically optimal.

The lower bounds are proved according to the following plan. For each n and k, one should construct an n-state automaton operating on graphs with k direction labels, so that any returning, halting or reversible automaton recognizing the same language would require many states. The n-state automaton follows a particular path in an input graph in search for a special node. The node is always on that path, so that the automaton naturally encounters it if it exists. On the other hand, the graph is constructed, so that getting back is more challenging.

The graph is made of elements called *diodes*, which are easy to traverse in one direction and hard to traverse backwards. Diodes are defined in Section 4, where it is shown that a GWA needs to employ extra states to traverse a diode backwards.

The graph used in all lower bound arguments, constructed in Section 5, has a main path made of diodes leading to a special node, which makes returning more complicated, so that a returning automaton needs at least $2(n-1)(k-3)$ states. A variant of this graph containing a cycle made of diodes, presented in Section 6, poses a challenge to a halting automaton, which needs at least $2(n-1)(k-3)$ states. Section 7 combines the two arguments to establish a lower bound of $4(n-1)(k-3)$ on the number of states of an automaton that is returning and halting at the same time. This bound is adapted to reversible automata in Section 8: at least $4(n-1)(k-3)-1$ states are required.

Overall, each transformation requires ca. $C \cdot nk$ states in the worst case, for a constant C. Each transformation has its own constant C, and these constants are determined precisely.

The second part of this thesis is about another type of transformations on GWA: operations of sets of graphs. For each graph operation, the first question is whether the family of sets of graphs recognized by graph-walking automata is closed under that operation, and if it is, then the second question is, how many states are needed for that? Not many operations on graphs are known. In this thesis, node-replacement homomorphisms, which replace nodes with subgraphs, are studied, as well as inverse homomorphisms. In the case of strings, a homomorphism is defined by the identity $h(uv) = h(u)h(v)$, and the class of regular languages is closed under all homomorphisms, as well as under their inverses, defined by $h^{-1}(L) = \{ w \mid h(w) \in L \}$. For the 2DFA model, the complexity of inverse homomorphisms is known: as shown by Jirásková and Okhotin [8], it is exactly $2n$ in the worst case, where n is the number of states in the original automaton. However, this proof is based on the transformations between one-way and two-way finite automata, which is a property unique for the string case. The state complexity of homomorphisms for 2DFA is known to lie between exponential and double exponential [8]. For tree-walking and graph-walking automata, no such questions were investigated before, and they are addressed in this thesis.

The closure of graph-walking automata under every inverse homomorphism is easy: in Section 9 it is shown that, for an n-state GWA, there is a GWA with $nk+1$ states for its inverse homomorphic image, where k is the number of labels of edge end-points. If the label of the initial node is unique, then nk states are enough. This transformation is proved to be optimal by establishing a lower bound of nk states. The proof of the lower bound makes use of a graph, constructed in Section 5, that is easy to pass in one direction and hard to pass in reverse.

The other result, presented in Section 10, is about inverse homomorphisms. It is claimed that the family of tree languages recognized by tree-walking automata is not closed under injective homomorpisms, thus settling this question for graph-walking automata as well. The result is proved by first establishing a characterization of regular tree languages by a combination of an injective homomorphism and an inverse homomorphism. This characterization generalizes a known result by Latteux and Leguy [15], see also an earlier result by Čulík et al. [4]. In light of this characterization, a closure under injective homomorphisms would imply that every regular tree language is recognized by a tree-walking automaton, which would contradict the famous result by Bojańczyk and Colcombet [2].

The results of Sections 3–8 were presented at STACS 2021 conference [16]. The results of Sections 9–10 have been submitted for publication.

2 Graph-walking automata and their subclasses

The definition of graph-walking automata (GWA) is an intuitive extension of two-way finite automata (2DFA) and tree-walking automata (TWA). However, formalizing it requires extensive notation. First, there is a notion of a *signature*, which is a generalization of an alphabet to the case of graphs.

Definition 1 (Kunc and Okhotin [12]). *A signature S consists of*

- A finite set D of directions, that is, labels attached to edge end-points;
- A bijection $-\colon D \to D$ providing an opposite direction, with $-(-d) = d$ for all $d \in D$;
- A finite set Σ of node labels;
- A non-empty subset $\Sigma_0 \subseteq \Sigma$ of possible labels of the initial node;
- A set of directions $D_a \subseteq D$ for every label $a \in \Sigma$. Every node labelled with a must be of degree $|D_a|$, with the incident edges corresponding to the elements of D_a.

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

Definition 2. *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;
- $+\colon V \times D \to V$ is a partial function, such that if $v + d$ is defined, then $(v + d) + (-d)$ is defined and equals v;
- a total mapping $\lambda\colon V \to \Sigma$, such that $v+d$ is defined if and only if $d \in D_{\lambda(v)}$, and $\lambda(v) \in \Sigma_0$ if and only if $v = v_0$.

In this thesis, all graphs are finite and connected.

Once graphs are formally defined, a graph-walking automaton is defined similarly to a 2DFA.

Definition 3. *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 a and q where δ is defined.

A computation of a GWA on a graph $(V, v_0, +, \lambda)$ is a uniquely defined sequence of configurations (q, v), with $q \in Q$ and $v \in V$. It begins with (q_0, v_0) and proceeds from (q, v) to $(q', v+d)$, where $\delta(q, \lambda(v)) = (q', d)$. The automaton accepts by reaching (q, v) with $(q, \lambda(v)) \in F$.

On each input graph, a GWA can accept, reject or loop. There is a natural subclass of GWA that never loop.

The language of GWA A is a set of graphs, which A accepts. It is denoted by $L(A)$.

Definition 4. A graph-walking automaton is said to be halting, if its computation on every input graph is finite.

Another property is getting back to the initial node before acceptance: if a GWA is regarded as a robot, it returns to its hangar, and for a generic model of computation, this property means cleaning up the memory.

Definition 5. A graph-walking automaton $A = (Q, q_0, F, \delta)$ over a signature $S = (D, -, \Sigma, \Sigma_0, (D_a)_{a \in \Sigma})$ is called returning, if $F \subseteq Q \times \Sigma_0$, which means that it can accept only in the initial node.

A returning automaton is free to reject in any node, and it may also loop, that is, it need not be halting.

The next, more sophisticated property is *reversibility*, meaning that, for every configuration, the configuration at the previous step can be uniquely reconstructed. This property is essential in quantum computing, whereas irreversibility in classical computers causes energy dissipation, which is known as *Landauer's principle* [13].

The definition of reversibility begins with the property that every state is reachable from only one direction.

Definition 6. A graph-walking automaton $A = (Q, q_0, F, \delta)$ over a signature $S = (D, -, \Sigma, \Sigma_0, (D_a)_{a \in \Sigma})$ is called direction-determinate, if there is a function $d \colon Q \to D$, such that, for all $p \in Q$ and $a \in \Sigma$, if $\delta(p, a)$ is defined, then $\delta(p, a) = (q, d(q))$ for some $q \in Q$.

Definition 7. A graph-walking automaton $A = (Q, q_0, F, \delta)$ over a signature $S = (D, -, \Sigma, \Sigma_0, (D_a)_{a \in \Sigma})$ is called reversible, if

- A is direction-determinate;
- for all $a \in \Sigma$ and $q \in Q$, there is at most one state p, such that $\delta(p, a) = (q, d(q))$; in other words, knowing a state and a previous label, one can determine the previous state;
- The automaton is returning, and for each $a_0 \in \Sigma_0$ there exists at most one such state q, that $(q, a_0) \in F$.

In theory, a reversible automaton may loop, but only through the initial configuration. In this case, it can be made halting by introducing an extra initial state.

Every GWA can be transformed to each of the above subclasses [12]. In the next section, the known transformations will be explained and slightly improved.

3 Upper bounds revisited

Before establishing the lower bounds on all transformations, the existing constructions of Kunc and Okhotin [12] will be somewhat improved by using fewer states. This is achieved by recombining the elements of the original construction, and, with these improvements, the constructions shall be proved asymptotically optimal.

The basic element for constructing reversible automata is a lemma by Kunc and Okhotin [12] (Lemma 1 below), which implements Sipser's idea of backtracking the tree of computations coming to an accepting configuration in the general case of graph-walking automata. The lemma assumes that the automaton is already direction-determinate, and it makes a further technical assumption that whenever the automaton has any behaviour defined on a pair (q, a)—transition or acceptance—the label a must support incoming transitions in the direction $d(q)$.

Definition 8 (Kunc and Okhotin [12, Defn. 6])**.** A direction-determinate automaton $A = (Q, q_0, \delta, F)$ is *without inaccessible transitions*, if, for all pairs $(q, a) \in (Q \times \Sigma)$ with $(q, a) \notin \{q_0\} \times \Sigma_0$, if $\delta_a(q)$ is defined or $(q, a) \in F$, then $-d(q) \in D_a$.

If there are any such transitions, they can be removed without affecting any computations.

Lemma 1 (Kunc and Okhotin [12, Lemma 6]). *For every direction-determinate automaton $\mathcal{A} = (Q, q_0, \delta, F)$ without inaccessible transitions, there exists a reversible automaton $\mathcal{B} = (\overrightarrow{Q} \cup [Q], \delta', F')$ without an initial state, and with states $\overrightarrow{Q} = \{\, \overrightarrow{q} \mid q \in Q \,\}$ and $[Q] = \{\, [q] \mid q \in Q \,\}$, which simulates $\mathcal{A}$ as follows. Each state $q \in Q$ has two corresponding states in $\mathcal{B}$: a* forward *state $\overrightarrow{q}$ accessible in the same direction $d'(\overrightarrow{q}) = d(q)$, and a* backward *state $[q]$ accessible in the opposite direction $d'([q]) = -d(q)$. The acceptance conditions of $\mathcal{B}$ are $F' = \big\{ ([\delta_{a_0}(q_0)], a_0) \bigm| a_0 \in \Sigma_0,\ \delta_{a_0}(q_0) \text{ is defined} \big\}$. For every finite graph $(V, v_0, \lambda, +)$, its node $\widehat{v} \in V$ and a state $\widehat{q} \in Q$ of the original automaton, for which $(\widehat{q}, \lambda(\widehat{v})) \in F$ and $-d(\widehat{q}) \in D_{\lambda(\widehat{v})}$, the computation of $\mathcal{B}$ beginning in the configuration $([\widehat{q}], \widehat{v} - d(\widehat{q}))$ has one of the following two outcomes.*

- *If $\mathcal{A}$ accepts this graph in the configuration $(\widehat{q}, \widehat{v})$, and if $(\widehat{q}, \widehat{v}) \neq (q_0, v_0)$, then $\mathcal{B}$ accepts in the configuration $([\delta_{\lambda(v_0)}(q_0)], v_0)$.*
- *Otherwise, $\mathcal{B}$ rejects in $(\overrightarrow{\widehat{q}}, \widehat{v})$.*

In particular, A in configuration (q, v) is simulated by B forward in the configuration $(\overrightarrow{q}, v)$ and backward in the configuration $([q], v - d(q))$. Note that the latter configuration is shifted by one node along the computation.

Note that the computation of A starting from the initial configuration can reach at most one accepting configuration (q, v), whereas for any other accepting configuration (q, v), the automaton B will not find the initial configuration and will reject as stated in the lemma.

To transform a given n-state GWA $\widehat{A}$ over a signature with k directions to a returning automaton, Kunc and Okhotin [12] first transform it to a direction-determinate automaton A with nk states; let B be the $2nk$-state automaton obtained from A by Lemma 1. Then they construct an automaton that first operates as A, and then, after reaching an accepting configuration, works as B to return to the initial node. This results in a returning direction-determinate automaton with $3nk$ states.

If the goal is just to return, and remembering the direction is not necessary, then $2nk + n$ states are actually enough.

Theorem 1. *For every n-state GWA over a signature with k directions, there exists a returning automaton with $2nk + n$ states recognizing the same set of graphs.*

Indeed, the original automaton $\widehat{A}$ can be first simulated as it is, and once it reaches an accepting configuration, one can use the same automaton B as in the original construction to return to the initial node. There is a small complication in the transition from $\widehat{A}$ to B, because in the accepting configuration, the direction last used is unknown. This is handled by cycling through all possible previous configurations of A at this last step, and executing B from each of them. If the direction is guessed correctly, then B finds the initial configuration and accepts. Otherwise, if the direction is wrongly chosen, B returns back, and then, instead of rejecting, it is executed again starting from the next direction. One of these directions leads it back to the initial node.

Proof of Theorem 1. Let $\widehat{A} = (Q, q_0, \delta, F)$ be the given automaton. Let A be a direction-determinate automaton with the set of states $Q \times D$ that recognizes the same set of graphs [12, Lemma 1]. Assume that all inaccessible transitions are removed from A.

Let B be the automaton obtained from A by Lemma 1: this is a reversible automaton without initial state, and it uses the set of states $\overrightarrow{Q} \cup [Q]$, where $\overrightarrow{Q} = \{\, \overrightarrow{(q, d)} \mid q \in Q, d \in D \,\}$ and $[Q] = \{\, [(q, d)] \mid q \in Q, d \in D \,\}$. Its transition function is denoted by δ'. There are $2nk$ states in B.

Assume any linear order on D.

A new automaton C is constructed with the set of states $Q \cup \overrightarrow{Q} \cup [Q]$ containing $n + 2nk$ states, and with a transition function δ'', to be defined below. If the initial configuration of C is an accepting configuration of $\widehat{A}$, then C immediately accepts as well. Otherwise, C begins by simulating $\widehat{A}$ in the states Q.

$$\delta''(q, a) = \delta(q, a), \qquad \text{for } q \in Q \text{ and } a \in \Sigma, \text{ if } \delta(q, a) \text{ is defined}$$

If the simulated $\widehat{A}$ reaches an accepting configuration $(\widetilde{q}, \widehat{v})$, this means that A, operating on the same graph, would reach an accepting configuration of the form $((\widetilde{q}, \widehat{d}), \widehat{v})$, for some direction $\widehat{d} \in -D_a$. However, C does not know this direction $\widehat{d}$, so it tries moving in all directions in D_a, beginning with the least one.

$$\delta''(\widetilde{q}, a) = ([(\widetilde{q}, -\min D_a)], \min D_a) \quad \text{for all } (\widetilde{q}, a) \in F \setminus (\{q_0\} \times \Sigma_0)$$

Let $d = -\min D_a$. In the notation of Lemma 1, $\widehat{q} = (\widetilde{q}, \widehat{d})$ and $d(\widehat{q}) = \widehat{d}$ and $-\widehat{d} \in D_a$. According to the lemma, if the direction was correctly guessed as $d = \widehat{d}$, then B, having started in the configuration $([\widehat{q}], \widehat{v} - d)$, accepts at the initial node. The automaton C simulates B to do the same.

$$\delta''([(q, d)], a) = \delta'([(q, d)], a), \qquad \text{for all } a \in \Sigma \text{ and } [(q, d)] \in [Q]$$
$$\delta''(\overrightarrow{(q, d)}, a) = \delta'(\overrightarrow{(q, d)}, a), \qquad \text{for all } a \in \Sigma \text{ and } \overrightarrow{(q, d)} \in \overrightarrow{Q}$$

If the direction was wrongly guessed as $d \neq \widehat{d}$, then $(\widetilde{q}, d)$ is still a valid state of A, so, by the lemma, B returns to the configuration $(\overrightarrow{(\widetilde{q}, d)}, \widehat{v})$, in which it would reject. The automaton C, instead of rejecting, tries the next available direction from D_a.

$$\delta''(\overrightarrow{(\widetilde{q}, d)}, a) = ([(\widetilde{q}, d')], -d') \qquad \text{for all } (\widetilde{q}, a) \in F \setminus (\{q_0\} \times \Sigma_0),\ d' = -next_{-d}(D_a)$$

Here $next_{-d}(D_a)$ denotes the least element of D_a greater than $-d$. The above transition is defined, assuming that $-d$ is not the greatest element of D_a. Since one of the directions in D_a is the true $-\widehat{d}$, it is eventually found, and the case of all directions failing need not be considered.

The acceptance conditions of C are the same as in B, and so C is returning. As argued above, C recognizes the same set of graphs as $\widehat{A}$. □

Kunc and Okhotin [12] did not consider halting automata separately. Instead, they first transform an n-state GWA to a $3nk$-state returning direction-determinate automaton, then use Lemma 1 to obtain a $6nk$-state reversible automaton, and add an extra initial state to start it. The resulting $(6nk + 1)$-state automaton is always halting.

If only the halting property is needed, then the number of states can be reduced.

Theorem 2. *For every n-state direction-determinate automaton, there exists a $(2n + 1)$-state halting and direction-determinate automaton that recognizes the same set of graphs.*

First, an n-state automaton $\widehat{A}$ is transformed to a direction-determinate nk-state automaton A, and Lemma 1 is used to construct a $2nk$-state automaton B. Then, the automaton B is *reversed* by the method of Kunc and Okhotin [12], resulting in an automaton B^R with $2nk + 1$ states that carries out the computation of B backwards. The automaton B^R is a halting automaton that recognizes the same set of graphs as $\widehat{A}$: it starts in the initial configuration, and if B accepts from an accepting configuration of A, then B^R finds this configuration and accepts; otherwise, B^R halts and rejects.

Proof of Theorem 2. Let A be the original automaton, let Q be its set of states. Assume that all inaccessible transitions have already been removed from A. Lemma 1 is used to construct a $2n$-state reversible automaton B without an initial state. The latter automaton B is then subjected to another transformation: by the method of Kunc and Okhotin [12], B is transformed to a *reversed* automaton B^R, which carries out the computation of B backwards, starting from the accepting configuration of B, and using a state $[\boldsymbol{q}]$ to simulate B in a state $\boldsymbol{q}$; in the simulation, the configurations are shifted by one node relative to B, so that B in $(\boldsymbol{q}, v)$ corresponds to B^R in $(\boldsymbol{q}, v - d(\boldsymbol{q}))$. Since B has no initial configuration defined, the acceptance conditions of B^R are not defined either, and shall be supplemented in the following. The automaton B^R has $2n+1$ states.

By Lemma 1, the set of states of B is $Q' = [Q] \cup \overrightarrow{Q}$. Then, B^R has the set of states $Q'' = [[Q]] \cup [\overrightarrow{Q}] \cup \{q_0''\}$, where $[[Q]] = \{\, [[q]] \mid q \in Q \,\}$ and $[\overrightarrow{Q}] = \{\, [\overrightarrow{q}] \mid q \in Q \,\}$, where q_0'' is the new initial state. The directions of states in B are $d'([q]) = -d(q)$ and $d'(\overrightarrow{q}) = d(q)$, and B^R uses the opposite directions: $d''([[q]]) = d(q)$ and $d''([\overrightarrow{q}]) = -d(q)$, for all $q \in Q$.

Let δ be the transition function of A, let the transition function of B be δ', and let B^R use δ''.

The transitions of B^R are defined by joining two constructions by Kunc and Okhotin [12], and are listed below. Transitions in the initial state q_0'' correspond to accepting conditions of B.

$$\delta''_{a_0}(q_0'') = \begin{cases} [[\delta_{a_0}(q_0)]], & \text{if } \delta_{a_0}(q_0) \text{ is defined in } A \\ \text{undefined}, & \text{otherwise} \end{cases}$$

The rest of the transitions carry out the computation backwards; each of them is uniquely defined, because B is reversible.

$$\delta''_a([\boldsymbol{q}]) = \begin{cases} [\boldsymbol{p}], & \text{if } \delta'_a(\boldsymbol{p}) = \boldsymbol{q} \\ \text{undefined}, & \text{otherwise} \end{cases}$$

The acceptance conditions of B^R are defined, so that it accepts a graph if so does A.

$$F'' = \{\, ([[q]], a) \bigm| (q, a) \in F \,\} \cup \{\, (q_0'', a_0) \bigm| (q_0, a_0) \in F \,\}$$

Overall, after being shifted by one node *twice*, the state $[[q]]$ corresponds to the state q of A, and B^R in a configuration $([[q]], v)$ corresponds to A in (q, v).

It is claimed that the automaton B^R is a halting automaton that recognizes the same set of graphs as A.

Its transitions are reversible by construction, and therefore it can loop only through the initial configuration. But since its initial state is not reenterable, it cannot loop at all, and is therefore halting.

To see that B^R recognizes the same graphs as A, first assume that A accepts some graph in a configuration $(\widehat{q}, \widehat{v})$. Then, B backtracks from $([\widehat{q}], \widehat{v} - d(\widehat{q}))$ to the configuration $([\delta_{a_0}(q_0)], v_0)$. The automaton B^R in turn moves from the configuration (q_0'', v_0) to $([[\widehat{q}]], \widehat{v})$ and accepts there.

Conversely, if B^R accepts a graph, then it does so in a configuration $([[\widehat{q}]], \widehat{v})$. The corresponding computation of B proceeds from $([\widehat{q}], \widehat{v} - d(\widehat{q}))$ to $([\delta_{a_0}(q_0)], v_0)$, and then, by Lemma 1, A accepts this graph in the configuration $(\widehat{q}, \widehat{v})$. $\square$

The construction of a reversible automaton with $6nk+1$ states can be improved to $4nk+1$ by merging the automata B and B^R. The new automaton first works as B^R to find the accepting configuration of A. If it finds it, then it continues as B to return to the initial node. In addition, this automaton halts on every input.

Theorem 3. *For every n-state direction-determinate automaton there exists a $(4n+1)$-state reversible and halting automaton recognizing the same set of graphs.*

Proof. Let A be a given direction-determinate automaton with n states. Assume that it is without *inaccessible transitions.*

First, as in the proof of Theorem 2, let B be the reversible automaton without an initial state constructed by Lemma 1, and then let B^R be another automaton with reversible transitions that simulates B backwards.

The goal is to combine B^R with B to obtain a new reversible and halting automaton C. The automaton C first operates as B^R, which is also known to be halting. Then, if A rejects or loops, then B^R halts and rejects, and C rejects accordingly. If A accepts immediately in its initial configuration, then so do B^R and C.

Assume that A accepts in a non-initial configuration $(\widehat{q}, \widehat{v})$. Then, B^R arrives at a configuration $([[\widehat{q}]], \widehat{v})$ and accepts. The automaton C needs to return to the initial node, so it does not accept immediately, as does B^R, and instead enters the configuration $([\widehat{q}], \widehat{v} - d(\widehat{q}))$ and continues as B. By definition, B, having started from this configuration, reaches the configuration $([\delta_{a_0}(q_0)], v_0)$ and accepts there; C does the same.

The set of states of C is the union of the sets of states of B and of B^R. There are $4n+1$ states in total, and they are enterable in the same directions as in B and in B^R. The initial state of C is q_0'' from B^R. Its transitions are all the transitions of B and B^R, as well as the following transitions that transfer control from B^R to B. These transitions are defined for every accepting pair $([[\widehat{q}]], a)$ of B^R, unless it is initial.

$$\delta_a'''([[\widehat{q}]]) = [\widehat{q}] \qquad\qquad ((\widehat{q}, a) \in F \setminus \{q_0\} \times \Sigma_0)$$

The acceptance conditions of C are the same as in B.

By the construction, the automaton C accepts the same set of graphs as A. It has $4n+1$ states. It should be proved that it is reversible.

For every initial label, C has at most one acceptance condition involving this label, since this is the case for B. The automaton C is returning. It remains to prove that, for every label a and for every state $\boldsymbol{q}$ of C, there is at most one state $\boldsymbol{p}$ with $\delta_a'''(\boldsymbol{p}) = \boldsymbol{q}$.

The automata B and B^R had this property by construction. The only new transitions are transitions of the form $\delta_a'''([[\widehat{q}]]) = [\widehat{q}]$, for non-initial acceptance conditions $(\widehat{q}, a)$ of the automaton A. It is enough to show that B does not have any other way of reaching a state $[\widehat{q}]$ from the label a.

Some details of the construction of B need to be examined [12, Lemma 6]. There are two kinds of states in B: $[p]$ and $\overrightarrow{p}$. By the construction of B, the state $[\widehat{q}]$ can potentially be reached by the following two transitions.

First, this could be a transition of the form $\delta_a'([p]) = [\widehat{q}]$ [12, Eq. (1)]. However, by the definition of B, this transition is defined only if A has a transition $\delta_a(\widehat{q}) = p$. This is impossible, since $(\widehat{q}, a)$ is an accepting pair for A.

The other possible transition is $\delta_a'(\overrightarrow{p}) = [\widehat{q}]$ [12, Eq. (3)]. For this transition to be defined, A must have a transition $\delta_a(\widehat{q}) = \delta_a(p)$, and $\delta_a(p)$ must accordingly be defined. Again, this cannot be the case, because the pair $(\widehat{q}, a)$ is accepting in A.

The definition of B includes one further type of transitions leading to a state $[\widehat{q}]$ [12, Eq. (5)]. Those transitions are actually never used and can be omitted; they were defined in the original paper for the sole reason of making the transition function bijective.

This confirms that the transfer of control from B^R to B is done reversibly, and so C is reversible. $\square$

With the upper bounds improved and some new upper bounds established, it is time to prove asymptotically matching lower bounds.

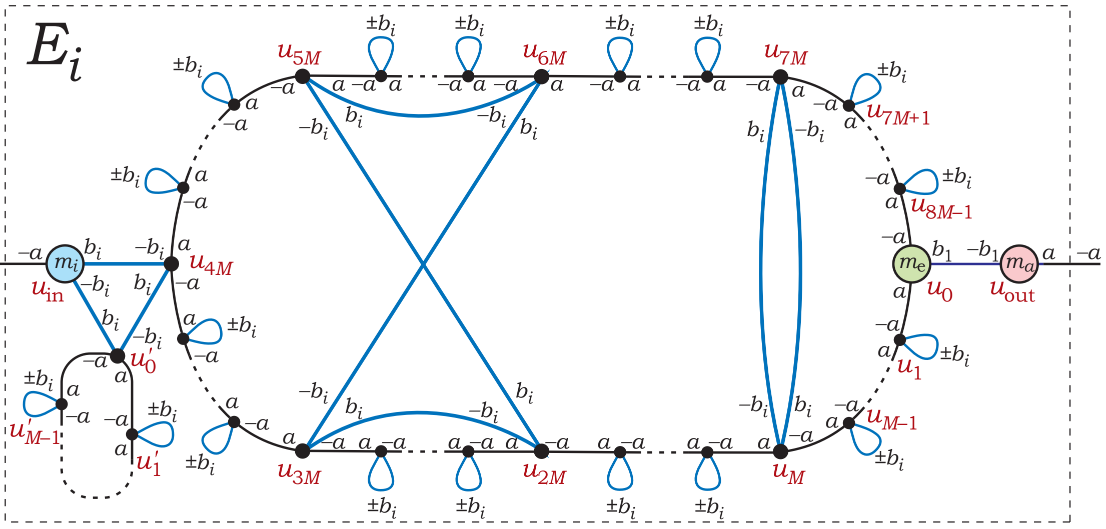

Figure 1: Element E_i. Filled circles are nodes labelled with m, each with $r-1$ loops in directions $\pm b_s$, with $s \neq i$.

4 Construction of a "diode"

Lower bounds on the size of GWA obtained in this thesis rely on quite involved constructions of graphs that are easy to traverse from the initial node to the moment of acceptance, whereas traversing the same path backwards is hard. An essential element of this construction is a subgraph called a *diode*; graphs in the lower bound proofs are made of such elements.

A diode is designed to replace an $(a, -a)$-edge. An automaton can traverse it in the direction a without changing its state. However, traversing it in the direction $-a$ requires at least $2(|D|-3)$ states, where D is the set of directions in the diode's signature.

If an automaton never moves in the direction $-a$, then it can be transformed to an automaton with the same number of states, operating on graphs in which every $(a, -a)$-edge is replaced with a diode.

Lower bound proofs for automata with n states over a signature with k directions use a diode designed for these particular values of n and k. This diode is denoted by $\Delta_{n,k}$.

For $n \geqslant 2$ and $k \geqslant 4$, let $M = (4nk)!$, and let $r = \lfloor \frac{k-2}{2} \rfloor$. A diode $\Delta_{n,k}$ is defined over a signature S_k that does not depend on n.

Definition 9. A signature $S_k = (D, -, \Sigma, \Sigma_0, (D_a)_{a \in \Sigma})$ consists of:

- the set of directions $D = \{a, -a\} \cup \{b_1, b_{-1}, \dots, b_r, b_{-r}\}$;
- opposite directions $-(a) = (-a)$, $-b_i = b_{-i}$, for $1 \leqslant i \leqslant r$;
- the set of node labels $\Sigma = \{m_1, \dots, m_r\} \cup \{m_{-1}, \dots, m_{-r}\} \cup \{m, m_e, m_a\}$, with no initial labels defined ($\Sigma_0 = \varnothing$) since the diode is inserted into graphs;
- sets of directions allowed at labels: $D_m = D$, $D_{m_i} = D_{m_{-i}} = \{-a, b_i, -b_i\}$, for $i = 1, \dots, r$, $D_{m_e} = \{b_1, a, -a\}$, $D_{m_a} = \{-b_1, a\}$.

A diode is comprised of $2r$ elements E_i, E_{-i}, for $i \in \{1, \dots, r\}$. Each element E_i and E_{-i} is a graph over the signature S_k, with two external edges, one with label a, the other with $-a$. By these edges, the elements are connected in a chain.

The form of an element E_i is illustrated in Figure 1. Its main part is a cycle of length $8M$ in directions $a, -a$; these are nodes $u_0, \dots, u_{8M-1}$, where the arithmetic in the node numbers is

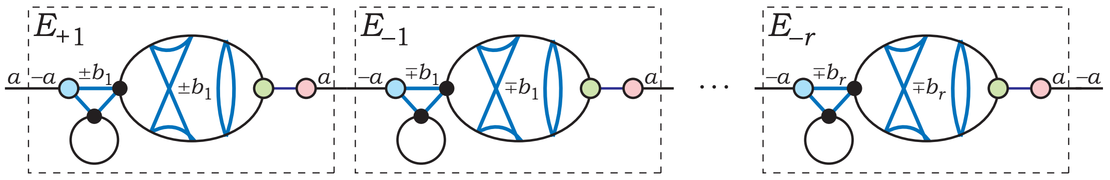

Figure 2: Diode $\Delta_{n,k}$: a chain of elements E_1, E_{-1}, E_2, E_{-2}, …, E_r, E_{-r}.

modulo $8M$, e.g., $u_{-1} = u_{8M-1}$. The node numbers are incremented in direction a. Besides the main cycle, there are two extra nodes: the entry point u_{in} and the exit u_{out}, as well as a small circle of length M in directions $a, -a$ with the nodes $u'_0, \ldots, u'_{M-1}$. All nodes are labelled with m, except three: u_{in} with label m_i matching the index of the element, u_{out} with label m_a, and u_0 has label m_{e}.

An element E_i has specially defined edges in directions b_i and $-b_i$. Each node u_j with $j \not\equiv 0 \pmod M$ has a $(b_i, -b_i)$-loop. The nodes u_j with $j \in \{M, 2M, 3M, 5M, 6M, 7M\}$ are interconnected with edges, as shown in Figure 1; these edges serve as traps for an automaton traversing the element backwards. The node u_{4M} has a different kind of trap in the form of a cycle u'_0, …, u'_{M-1}. For all $s \neq i$, each node labelled with m has a $(b_s, -b_s)$-loop.

The element E_{-i} is the same as E_i, with the directions b_i and $-b_i$ swapped.

Definition 10. A diode element E_i, with $i \in \{\pm 1, \ldots, \pm r\}$, is formally defined as follows.

- The set of nodes is $V = \{u_0, \ldots, u_{8M-1}\} \cup \{u'_0, \ldots, u'_{M-1}\} \cup \{u_{in}, u_{out}\}$.

  Node numbers are defined modulo $8M$. For instance, u_{-1} is a valid notation for u_{8M-1}.

- No initial node is defined, since a diode is a subgraph substituted into graphs, and not a valid graph on its own.

- The nodes have the following labels.

$$\begin{aligned} \lambda(u_{in}) &= m_i \\ \lambda(u_{out}) &= m_a \\ \lambda(u_0) &= m_{\mathrm{e}} \end{aligned}$$

  The rest of the nodes have label m.

- The edges are defined as follows.

$$\begin{aligned}
u_{in} + (-a) &= \text{outside (point of entrance by } a) \\
u_{in} + b_i &= u_{4M} \\
u_{4M} + b_i &= u'_0 \\
u'_0 + b_i &= u_{in} \\
u'_j + a &= u'_{j+1} \\
u'_j - a &= u'_{j-1} \\
u'_j \pm b_s &= u'_j, && \text{where } s \in \{1, \dots, r\}, (u'_j, b_s) \neq (u'_0, \pm b_i) \\
u_{out} + a &= \text{outside (point of exit by } a) \\
u_{out} - b_1 &= u_0 \\
u_0 + b_1 &= u_{out} \\
u_j + a &= u_{j+1} \\
u_j - a &= u_{j-1} \\
u_j \pm b_s &= u_j, && \text{where } s \in \{1, \dots, r\} \setminus \{i, -i\}, j \neq 0 \\
u_j \pm b_i &= u_j, && \text{where } j \notin \{0, M, 2M, 3M, 4M, 5M, 6M, 7M\} \\
u_M \pm b_i &= u_{-M} \\
u_{2M} + b_i &= u_{3M} \\
u_{3M} + b_i &= u_{-2M} \\
u_{-2M} + b_i &= u_{-3M} \\
u_{-3M} + b_i &= u_{2M}
\end{aligned}$$

The diode $\Delta_{n,k}$ is a chain of such elements.

Definition 11. The diode $\Delta_{n,k}$, defined over a signature S_k, is a chain of elements E_i, joined sequentially as shown in Figure 2. The order of elements is: $E_1, E_{-1}, E_2, E_{-2}, \dots, E_r, E_{-r}$. For every element in the chain, there is an a-edge from its node u_{out} to the node u_{in} in the next element. The entry point to the diode by a is by the $-a$-edge of the first element E_1, and the exit point is the a-edge of the last element E_{-r}.

Each element can be traversed from the entrance to the exit without changing the state: at first, the automaton sees the label m_i, and accordingly moves in the direction b_i; then, on labels m, it proceeds in the direction a until it reaches u_0, labelled with m_e. Then the automaton leaves the element by following directions b_1 and a.

The diode is hard to traverse backwards, because the node u_{4M} is not specifically labelled, and in order to locate it, the automaton needs to move in directions $\pm b_i$ from many nodes, and is accordingly prone to falling into traps.

The diode is used as a subgraph connecting two nodes of a graph as if an $(a, -a)$-edge.

Definition 12. For a graph G over some signature $\widetilde{S}$, let G' be a graph obtained by replacing every $(a, -a)$-edge in G with the diode $\Delta_{n,k}$. Denote this graph operation by $h_{n,k} \colon G \mapsto G'$.

The following lemma states that if an automaton never traverses an $(a, -a)$-edge backwards, then its computations can be replicated on graphs with these edges substituted by diodes, with no extra states needed.

Lemma 2. *Let $\widetilde{S}$ be any signature containing directions a, $-a$, which has no node labels from the signature S_k. Let $A = (Q, q_0, F, \delta)$ be a GWA over the signature $\widetilde{S}$, which never moves in the direction $-a$.*

Then, there exists a GWA $A' = (Q, q_0, F, \delta')$ over a joint signature $\widetilde{S} \cup S_k$, so that A accepts a graph G if and only if A' accepts the graph $G' = h_{n,k}(G)$.

Proof. The automaton A' uses the same states as A, but the transition function is augmented with transitions by labels from the diode's signature. Since none of the labels exists in the signature $\widetilde{S}$, the new transitions would not contradict the existing ones.

For each state $q \in Q$, the following transitions are added to δ'.

$$\begin{aligned}
\delta'(q, m_i) &= (q, b_i), && \text{for } i \in \{\pm 1, \ldots, \pm r\} \\
\delta'(q, m) &= (q, a) \\
\delta'(q, m_{\mathrm{e}}) &= (q, b_1) \\
\delta'(q, m_a) &= (q, a)
\end{aligned}$$

Let G be any graph over the signature $\widetilde{S}$, and let $G' = h_{n,k}(G)$ be the graph constructed by replacing every $(a, -a)$-edge in G with a diode. Let V be the set of nodes of G. Then the set of nodes of G', denoted by V', contains V as a subset, and has extra nodes used by the substituted diodes.

Consider the computation of A on the graph G. It corresponds to the computation of A' on G' as follows. Let $t_1, t_2, \ldots$ be the sequence of moments in the computation of A' on G', at which the automaton visits any nodes from V. It is claimed that each i-th step of the computation of A corresponds to the step t_i of the computation of A', as follows.

Claim 1. *The automaton A, at the i-th step of its computation on G, is in a node v in a state q in and only if A', at the step t_i on G' is in the same node v of G', in the same state q.*

The claim is proved by induction on i. The base case is step $i = 0$, when both A and A' are in their initial configurations. Each configuration is of the form (q_0, v_0).

Induction step. Assume that both A and A' visit the same node v in the same state q at the moments i and t_i, respectively. The claim is that next, at the moments $i + 1$ and t_{i+1}, both automata are again in the same node and in the same state. Let f be the label of v. There are the following cases.

- If this is an accepting pair $(q, f) \in F$, then, since $F = F'$, both automata accept.
- If the transition $\delta(q, f)$ is undefined, then, since $v \in V$, its label is not from the diode's signature, and A' has no new transitions at the label $\lambda(v)$. Then, the transition $\delta'(q, f)$ is undefined too, and both automata reject their graphs.
- If the next transition is not in the directions $\pm a$, then $\delta(q, f) = \delta'(q, f) = (p, d)$, and both automata proceed to the next configuration $(p, v + d)$, where $v + d \in V$.
- Let the next transition be in the direction a, so that $\delta(q, f) = (p, a)$. At the next step, the automaton A comes to the configuration (p, u), where u denotes the node $v + a$ in the graph G.

  For A', the moment t_{i+1} is the next visit to any of the nodes from V after t_i. At the next step after t_i, the automaton A' enters the diode connecting the nodes v and u in the graph G', and changes its state to p, because $\delta'(q, f) = \delta(q, f) = (p, a)$. The transitions at the labels from the diode's signature are defined so that the automaton A' traverses the entire diode in the state p, and leaves the diode for the node u. This u is the next node from V visited by A' after t_i, and A' visits it in the state p, the same as the state of A.
- The transition $\delta(q, f)$ cannot be in the direction $-a$, because, by the assumption, A never moves in this direction.

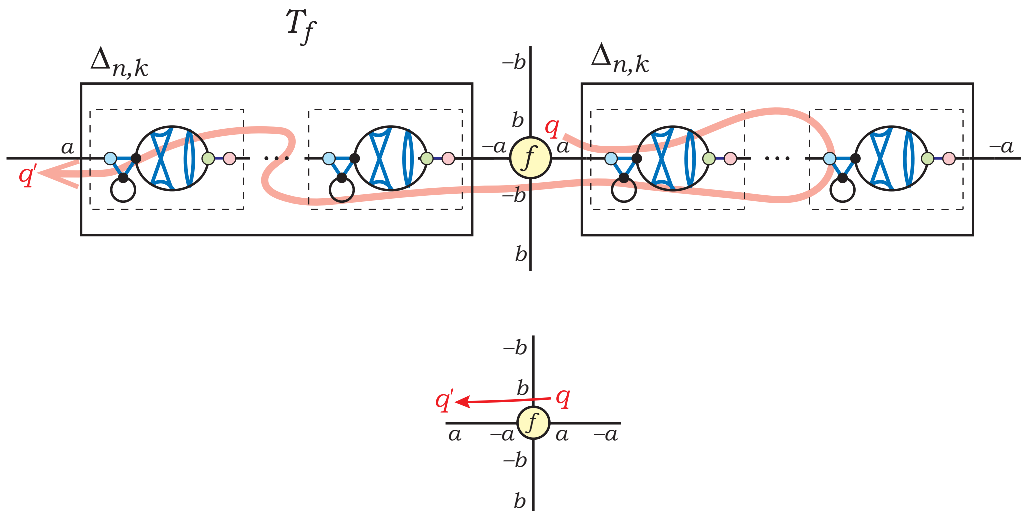

Figure 3: The graph T_f in the proof of Lemma 3.

Thus, the correspondence between i and t_i has been established, and then the automaton A accepts the graph G if and only if A' accepts G'. □

Lemma 2 shows that, under some conditions, a substitution of diodes can be implemented on GWA without increasing the number of states. The next lemma presents an *inverse substitution of diodes*: the set of pre-images under $h_{n,k}$ of graphs accepted by a GWA can be recognized by another GWA with the same number of states.

Lemma 3. *Let $k \geqslant 4$ and $n \geqslant 2$, denote $h(G) = h_{n,k}(G)$ for brevity. Let $\widetilde{S}$ be a signature containing the directions $a, -a$ and no node labels from the diode's signature S_k. Let $B = (Q, q_0, H, \sigma)$ be a GWA over the signature $\widetilde{S} \cup S_k$. Then there exists an automaton $C = (Q, q_0, F, \delta)$ over the signature $\widetilde{S}$, with the following properties.*

- *For every graph G over $\widetilde{S}$, the automaton C accepts G if and only if B accepts $h(G)$.*
- *If C can enter a state q by a transition in the direction $-a$, then B can enter the state q after traversing the diode backwards.*
- *if C can enter a state q by a transition in the direction a, then B can enter the state q after traversing the diode forward.*
- *If B is returning, then so is C.*
- *If B is halting, then C is halting as well.*

The automaton C is constructed by simulating B on small graphs, and using the outcomes of these computations to define the transition function and the set of acceptance conditions of C. Note that the signatures $\widetilde{S}$ and S_k may contain any further common directions besides $a, -a$: this does not cause any problems with the proof, because the node labels are disjoint, and thus B always knows whether it is inside or outside a diode.

Proof of Lemma 3. The automaton $C = (Q, q_0, F, \delta)$ is constructed by simulating $B = (Q, q_0, H, \sigma)$ on small graphs and using the outcomes of these computations to define the transition function δ and the set of acceptance conditions F.

For every state $q \in Q$ and label f from $\widetilde{S}$, the transition $\delta(q, f)$ is defined by simulating the computation of B on a graph denoted by T_f and constructed as follows, see Figure 3. Let v be a node with label f, with attached edges in all directions in D_f, leading outside of T_f. Edges in directions a and $-a$ are replaced with diodes. The automaton B begins its computation at the node v of T_f in the state q. If it eventually leaves T_f by one of the edges leading outside, in some direction d and in a state q', then the transition is defined as $\delta(q, f) = (q', d)$. If the automaton B accepts without leaving T_f, then the pair (q, f) is accepting in C. If B loops or encounters an undefined transition, then the transition $\delta(q, f)$ is undefined.

Claim 2. *For every graph G over the signature $\widetilde{S}$, the automaton C accepts G if and only if B accepts $h(G)$. Furthermore, if B is returning, then C is returning, and if B is halting, then C is halting.*

Proof. Let V be the set of nodes of G. Then, $h(G)$ has the set of nodes $V \cup V_\Delta$, where V_Δ is the set of all internal nodes in all diodes in $h(G)$.

Let $R_C = \{ (q_i, v_i) \mid i = 0, \ldots, N_C \}$ be the computation of C on G, and let $R_B = \{ (p_j, u_j) \mid j = 0, \ldots, N_B \}$ be the computation of B on $h(G)$. The length of either computation, N_C or N_B, can be infinite.

Each step of C either repeats the corresponding step of B, or contracts several moves of B on T_f into a single transition. Let $m \colon \{0, \ldots, N_C\} \to \{0, \ldots, N_B\}$ be the function that maps the number i of a configuration in the computation R_C to the number $j = m(i)$, so that $(q_i, v_i) = (p_j, u_j)$ and $m(i+1) > m(i)$. Note that configurations of two automata may be equal, because B and C share the same set of states, and every node of G is a node of $h(G)$.

The function m is constructed inductively on i.

- Basis: $i = 0$. The configuration (q_0, v_0) is initial in both computations, and $m(0) = 0$.
- Induction step. Let $(q_i, v_i) = (p_{m(i)}, u_{m(i)})$, and assume that it is not the last configuration in R_C. The goal is to find a number $m(i+1)$, such that $m(i) < m(i+1) \leqslant N_B$ and $(q_{i+1}, v_{i+1}) = (p_{m(i+1)}, u_{m(i+1)})$.

  Let f be the label of v_i. Then, T_f is a subgraph of $h(G)$ centered around v_i. Since $\delta(q_i, v_i) = (q_{i+1}, d_{i+1})$, where d_{i+1} is the direction from v_i to v_{i+1} in the graph G, the automaton B, having arrived to the node v_i of $h(G)$ in the state q_i, eventually leaves T_f in the direction d_{i+1} in the state q_{i+1}. The direction d_{i+1} from T_f leads to the node v_{i+1}. Thus, some time after the configuration (q_i, v_i), the automaton B reaches the configuration (q_{i+1}, v_{i+1}). Let $m(i+1)$ be this moment in the computation R_B.

Now it will be shown that if C accepts G, then B accepts $h(G)$, and if C does not accept G, then B does not accept $h(G)$.

Let C accept the graph G. Consider the last configuration (q, v) in the computation R_C. It is accepting for C. The automaton B reaches the same configuration in its computation on $h(G)$. Let f be the label of v. Consider the subgraph T_f of $h(G)$ around the node v. Since (q, v) is an accepting configuration of C, the automaton B must accept $h(G)$ starting from the configuration (q, v), without leaving T_f. So, B accepts $h(G)$. Note that if B is returning, then it can accept only in the initial node, that is, $f = a_0$, and then C can accept only in the initial node as well.

Assume that C does not accept G. If C loops, then $N_C = \infty$, and in this case, since m is an injection, then $N_B = \infty$, and B loops as well. This, in particular, implies that if B is halting, then C is halting too.

The other case is when C reaches a configuration (q, v), the label of v is f, and $\delta(q, f)$ is undefined, while (q, f) is not an accepting pair. Then the automaton B, having started on T_f at the node v in the state q, reaches an undefined transition without leaving T_f, or loops inside T_f. In both cases B does not accept $h(G)$. This confirms that if C does not accept G, then B does not accept $h(G)$. □

Claim 3. *If C can come to a state q by a transition in the direction $-a$, then B can come to q after traversing a diode backwards.*

Proof. Let this transition in C be $\delta(p, f) = (q, -a)$. By the construction of C, this means that if B runs on T_f beginning in v in the state p, then it leaves T_f in the direction $-a$ in the state q. Then, this transition is made after traversing the diode backwards. □

Analoguosly, if C come to a state q by a transition in the direction a, then B can come to q after traversing a diode forward. □

The next lemma formally establishes that it is hard to traverse a diode backwards.

Lemma 4. *Let $A = (Q, q_0, F, \delta)$ be a GWA over a signature that includes diode's signature S_k, with $|Q| \leqslant 4nk$. Assume that A, after traversing the diode $\Delta_{n,k}$ backwards, can leave the diode in any of h distinct states. Then A has at least $2h(k-3)$ distinct states.*

While moving through an element E_i backwards, the automaton sees labels m most of the time, and soon begins repeating some sequence of states periodically. Without loss of generality, assume that this periodic sequence contains more transitions in the direction a than in $-a$. Then the automaton reaches the node u_M, and at this point it may teleport between u_M and u_{-M} several times. Let $w \in \{b_i, -b_i\}^*$ be the sequence of these teleportation moves, and let x be the corresponding sequence of states. Depending on the sequence w, the automaton may eventually exit the cycle to the node u_{in}, or fall into one of the traps and get back to u_0. It is proved that for the automaton to reach u_{in}, the string w must be non-empty and of even length; furthermore, if $|w| = 2$, then $w = (-b_i)b_i$.

Now consider the h backward traversals of the diode ending in some states $p_1, \ldots, p_h$. When the traversal ending in p_j proceeds through the element E_i, the strings $w_{i,j} \in \{b_i, -b_i\}^*$ and $x_{i,j}$ are defined as above. Then, as the last step of the argument, it is proved that whenever $|w_{i,j}| = 2$, the states in $x_{i,j}$ cannot occur in any other string $x_{i',j'}$. For $w_{i,j}$ of length 4 or more, the states in $x_{i,j}$ can repeat in other strings $x_{i',j'}$, but only once. It follows that there are at least $2h(k-3)$ distinct states in these strings.

Proof of Lemma 4. Assume that A moves through the element E_i from the node u_{out} to the node u_{in}. In this computation, there is the moment when A visits u_0 for the last time before leaving the element E_i. Let t_0 denote this moment, as the number of a computation step. At this moment, the automaton makes a transition that puts it either on the segment from u_0 to u_M, or on the segment from u_0 to u_{-M}. Since, by assumption, the automaton shall never return to u_0 and shall eventually reach u_{in}, it must traverse this segment and reach the corresponding node $u_{\pm M}$. Let t_1 be the moment of the first visit to any of the nodes $u_{\pm M}$ after t_0.

Since the moment t_0 and until the subsequent first visit to u_{in}, the automaton shall move over nodes labelled with m. Therefore, already after $|Q|$ steps or earlier—and much before the moment t_1, since $M \gg |Q|$—the automaton loops, that is, begins repeating some sequence of states $q_1, \ldots, q_p$, and some sequence of directions $d_1, \ldots, d_p$, where p is the minimal period, and the enumeration of states in the period is chosen so that at the moment t_1 the automaton is in the state q_1.

In the sequence $d_1, \ldots, d_p$, one of the directions $a, -a$ must occur more often than the other, since otherwise the automaton never reaches the node $u_{\pm M}$. Assume that the direction a is the one that occurs more often (the case of $-a$ is symmetric), and the automaton accordingly moves from u_0 to u_M. On the way, the automaton actually moves only in directions $\pm a$, while all its transitions in directions $\pm b_j$ follow the loops and do not move the automaton. However, as the automaton moves through the node u_M, the transitions in directions $\pm b_i$ are executed, and the automaton, without noticing that, teleports to the node with an opposite number. While following the sequence of directions $d_1, \ldots, d_p$, the automaton may move away from the nodes $u_{\pm M}$ by several edges in directions a and $-a$, then get back, get teleported to the other part of the graph in the directions $\pm b_i$, then again move away, etc. But since a occurs in the periodic part more often than $-a$, eventually the automaton passes through the pair of nodes $u_{\pm M}$ and moves on.

Let $w \in \{b_i, -b_i\}^*$ be the string of directions, in which the automaton, having arrived to the node u_M in the state q_1, teleports between the nodes u_M and u_{-M}, until the general flow of its motion in the direction a moves it away from this pair of nodes. Let x be the corresponding string of states, in which the automaton makes the transitions in the string w.

If the automaton teleports between the nodes u_M and u_{-M} an even number of times, that is, if $|w|$ is even, then it proceeds further to the segment from u_M to u_{2M}. If the length of w is odd, then the automaton is teleported to the node u_{-M} and sets foot on the path in the direction of u_0: and then, moving in the general direction a, it reaches the node u_0, which contradicts the assumption. Thus, it has been proved that the number of teleportations is even.

Claim 4. *The length of w is even, and the automaton, having arrived to the node u_M in the state q_1, continues in the direction of the node u_{2M}.*

Let s be the difference between the number of occurrences of a and $-a$ in the sequence $d_1, \ldots, d_p$. Since $s \leqslant p \leqslant |Q|$, the number M is divisible by s, and thus, from the moment t_1 of the first visit to u_M and until the moment of the first visit to u_{2M}, the automaton makes a whole number of periods $\frac{M}{s}$, and accordingly comes to u_{2M} in the state q_1.

If the string w is empty, then the automaton moves directly through u_{2M}, without teleporting anywhere in the directions $\pm b_i$, and then passes by u_{iM}, for $i = 3, 4, 5, 6, 7$, in the same way, getting to each of these nodes in the state q_1. Eventually, contrary to the assumption, it comes to the node u_0. Therefore, this case is impossible.

Claim 5. *The string w is non-empty.*

Thus, having reached the node u_{2M}, the automaton, following directions from a non-empty string w of even length, teleports several times between the nodes u_{2M}, u_{3M}, u_{5M} and u_{6M}. In the following, it will be proved that if w is of length 2, then it is uniquely defined.

Claim 6. *If $|w| = 2$, then $w = (-b_i)b_i$.*

Indeed, if $w = b_i b_i$ or $w = (-b_i)(-b_i)$, then the automaton, while passing through the node u_{2M}, teleports to the node u_{6M}, and then, gradually moving in the general direction a, reaches the node u_0. This is a contradiction.

If $w = b_i(-b_i)$, then the automaton passes through the nodes u_{iM} for $i = 3, 4, 5, 6, 7$, first getting into each of these nodes in the state q_1, and finally comes to the node u_0. It should be mentioned that, while passing through the node u_{4M}, the automaton teleports to the node u'_0, but since its vicinity in the directions $\pm a$ is indistinguishable from long segments of the large cycle, the automaton does not see any difference and eventually teleports back to u_{4M}, without ever paying a visit in the direction b_i and without seeing the node u_{in}.

Therefore, there is only one possibility left, that $w = (-b_i)b_i$, or otherwise w must be of length at least 4.

In the case when $-a$ occurs in the period more often than a, Claims 4–6 on the properties of the string w hold as stated and can be proved analogously.

By the assumption, there are h backward traversals of the diode ending in h distinct states: p_1, $\ldots$, p_h. Each of these traversals includes passing through each element E_i, for $i \in \{\pm 1, \ldots, \pm r\}$. For the computation that eventually leaves the diode in the state p_j, one can define the string $w_{i,j} \in \{b_i, -b_i\}^*$ of directions in which the automaton "teleports", as w was defined above. Let $x_{i,j}$ be the corresponding string of states. All properties of these strings hold as stated.

It will be proved that in the sequences of states $x_{i,j}$, for $i = \pm 1, \ldots, \pm r$ and $j = 1, \ldots, h$, there are in total at least $4rh$ distinct states, which will prove the lemma.

First, consider that in the states from $x_{i,j}$, the automaton moves in the directions listed in $w_{i,j}$, while $w_{i,j}$ contains only directions $b_i, -b_i$. Then, as long as $i_1 \neq i_2$ and $i_1 \neq -i_2$, the sets of states used in $x_{i_1,j}$ and $x_{i_2,j}$ are disjoint. Then, it is sufficient to prove that, for each i, the strings $x_{i,j}, x_{-i,j}$, for all $j = 1, \ldots, h$, together contain at least $4h$ distinct states.

It is claimed that, for each $i \in \{\pm 1, \ldots, \pm r\}$, all states in the strings $x_{i,1}, \ldots, x_{i,h}$ are pairwise distinct. Indeed, suppose that some state q occurs twice in these strings. Within a single string $x_{i,j}$, all states are known to be distinct. Then, there exist different j and j', such that q occurs both in $x_{i,j}$ and in $x_{i,j'}$. By definition, if the state q is in the string $x_{i,j}$, this means that the computation through E_i passes through one of the nodes u_M and u_{-M} in the state q. This computation does not return to u_0 anymore, and eventually leads the automaton out of the diode in the state p_j; and if the automaton is put in the same state q into the other of the two nodes u_M and u_{-M}, then it returns to the node u_0 of the element E_i. Since q also occurs in the string $x_{i,j'}$, then, by the same reasoning, the computation starting in one of the nodes u_M and u_{-M} proceeds out of the diode in the state $p_{j'}$. This is a contradiction, which establishes the following claim.

Claim 7. *All states in the strings $x_{i,1}, \ldots, x_{i,h}$ are distinct, and all states in the strings $x_{-i,1}, \ldots, x_{-i,h}$ are distinct as well.*

Note that the strings $x_{i,j}$ and $w_{i,j}$ can be uniquely reconstructed from each state q from $x_{i,j}$. Indeed, a state q occurs in the sequence of states repeated periodically on the labels m. Then the automaton A can be put in the state q into one of the nodes u_M, u_{-M}—the one from which it will not return to the node u_0 of E_i. Then, the cycle can be unrolled forward and backward, and the periodic part of the computation is thus reconstructed. And then, the strings of states and directions $x_{i,j}$ and $w_{i,j}$ are extracted from this periodic part.

The strings $w_{i,j}$ can be of two kinds: either $w_{i,j} = (-b_i)b_i$, or $|w_{i,j}| \geqslant 4$. If a string $w_{i,j}$ is of length 2, then it cannot coincide with any of the strings $w_{-i,j'}$, because $w_{i,j} = (-b_i)b_i$, whereas $w_{-i,j'}$ is either equal to $b_i(-b_i)$, or is of length at least 4. Since the strings x and w are reconstructed from a single state, if a string $x_{i,j}$ is of length 2, then its states do not occur in any other strings $x_{i',j'}$.

Now it can be proved, for each i, that there are at least $4h$ distinct states in the strings $x_{i,j}$, $x_{-i,j}$, for $j = 1, \ldots, h$. All states in the strings $x_{i,1}, \ldots, x_{i,h}$ are pairwise distinct; so are the states in the strings $x_{-i,1}, \ldots, x_{-i,h}$. Therefore, every state can occur at most twice: once in one of the strings $x_{i,1}, \ldots, x_{i,h}$, and the other time in one of the strings $x_{-i,1}, \ldots, x_{-i,h}$. Let c be the number of strings of length 2 among the $2h$ strings $x_{\pm i,j}$, with $j = 1, \ldots, h$. Then, there are $2h - c$ strings of length at least 4. All states occurring in the strings of length 2 are unique; states in the rest of the strings can coincide, but only in pairs. Overall, there are no fewer than $2c + \frac{4(2h-c)}{2} = 4h$ distinct states, as desired.

□

5 Lower bound on the size of returning automata

By the construction of Kunc and Okhotin [12], as improved in Section 3, an n-state GWA over a signature with k directions can be transformed to a returning GWA with $2nk + n$ states. A closely matching lower bound will now be proved by constructing an automaton with n states over a signature with k directions, such that every returning automaton that recognizes the same set of graphs must have at least $2(n-1)(k-3)$ states.

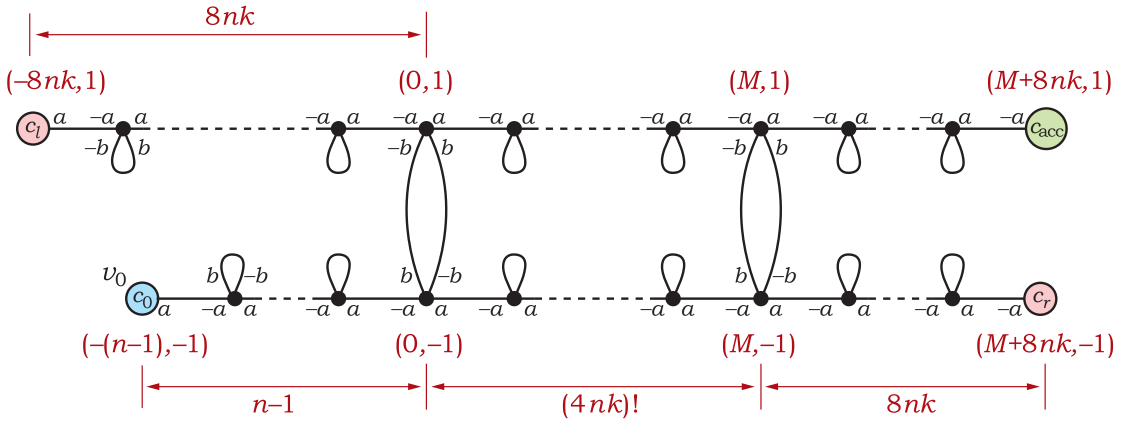

Figure 4: The graph $G^{accept}_{n,k}$.

The first step is a construction of a simple automaton over a signature $\widetilde{S}$ with four directions $a, -a, b, -b$ and two graphs over this signature, so that the automaton accepts one of them and rejects the other. The automaton will have n states, it will never move in the direction $-a$, and every returning automaton recognizing the same set of graphs can enter $n-1$ distinct states after transitions in the direction $-a$. Then, Lemma 4 shall assert that every returning automaton recognizing the same graphs with diodes substituted must have the claimed number of states.

Definition 13. The signature $\widetilde{S} = (D, -, \Sigma, \Sigma_0, (D_a)_{a\in\Sigma})$ uses the set of directions $D = \{a, -a, b, -b\}$, with $-(a) = (-a)$, $-(b) = (-b)$. The set of node labels is $\Sigma = \{c_0, c, c_l, c_r, c_{acc}\}$, with initial labels $\Sigma_0 = \{c_0\}$. The allowed directions are $D_c = D$, $D_{c_0} = D_{c_l} = \{a\}$, and $D_{c_r} = D_{c_{acc}} = \{-a\}$.

For $n \geqslant 2$ and $k \geqslant 4$, let $M = (4nk)!$ be as in the definition of the diode $\Delta_{n,k}$. Let $G^{accept}_{n,k}$ and $G^{reject}_{n,k}$ be two graphs over the signature $\widetilde{S}$, defined as follows. The graph $G^{accept}_{n,k}$ is illustrated in Figure 4; the other graph $G^{reject}_{n,k}$ is almost identical, but the node that determines acceptance is differently labelled.

Both graphs consist of two horizontal chains of nodes, connected by bridges at two places. Nodes are pairs (x, y), where $y \in \{-1, 1\}$ is the number of the chain, and x is the horizontal coordinate, with $-(n-1) \leqslant x \leqslant M+8nk$ for the lower chain ($y = -1$) and $-8nk \leqslant x \leqslant M+8nk$ for the upper chain ($y = 1$).

All nodes except the ends of chains have labels c. The node $(-(n-1), -1)$ is the initial node, with label c_0. The other left end $(-8nk, 1)$ is labelled with c_l. The node $(M + 8nk, -1)$ has label c_r. The node $(M + 8nk, 1)$ is labelled with c_{acc} in $G^{accept}_{n,k}$ and with c_r in $G^{reject}_{n,k}$; this is the only difference between the two graphs.

The horizontal chains are formed of $(a, -a)$-edges, with a incrementing x and $-a$ decrementing it. Edges with labels $(b, -b)$ are loops at all nodes except for $(0, 1)$, $(0, -1)$, $(M, 1)$ and $(M, -1)$. The latter four nodes form two pairs connected with bridges in directions $(b, -b)$.

Definition 14. The graphs $G^{accept}_{n,k}$ and $G^{reject}_{n,k}$ over the signature $\widetilde{S}$ are defined as follows.

- The set of nodes is $V = \{\, (x, -1) \mid x \in \{-(n-1), \ldots, M + 8nk\} \,\} \cup \{\, (x, 1) \mid x \in \{-8nk, \ldots, M + 8nk\} \,\}$.
- The initial node is $v_0 = (-(n-1), -1)$.

- The labels of the nodes are as follows.

$$\begin{aligned}
\lambda((x,y)) &= c, && \text{except for } (x,y) \in \{(-(n-1),-1),\\
& && (M+8nk,-1),(-8nk,1),(M+8nk,1)\}.\\
\lambda((-(n-1),-1)) &= c_0\\
\lambda((M+8nk,-1)) &= c_r\\
\lambda((-8nk,1)) &= c_l\\
\lambda((M+8nk,1)) &= \begin{cases} c_{acc}, & \text{for } G^{accept}_{n,k}\\ c_r, & \text{for } G^{reject}_{n,k}\end{cases}
\end{aligned}$$

- The following edges are defined.

$$\begin{aligned}
(x,y)+a &= (x+1,y), && \text{if } (x,y) \neq (M+8nk,-1), (x,y) \neq (M+8nk,1)\\
(x,y)+(-a) &= (x-1,y), && \text{if } (x,y) \neq (-(n-1),-1), (x,y) \neq (-8nk,1)\\
(x,y)+b &= (x,y), && \text{if } (x,y) \notin \{(M+8nk,\pm1),(-8nk,1),(-(n-1),-1)\},\\
& && x \neq 0, x \neq M\\
(x,y)+(-b) &= (x,y), && \text{if } (x,y) \notin \{(M+8nk,\pm1),(-8nk,1),(-(n-1),-1)\},\\
& && x \neq 0, x \neq M\\
(0,y)+b &= (0,-y)\\
(0,y)+(-b) &= (0,-y)\\
(M,y)+b &= (M,-y)\\
(M,y)+(-b) &= (M,-y)
\end{aligned}$$

An n-state automaton A, that accepts the graph $G^{accept}_{n,k}$, does not accept any graphs without labels c_{acc} and never moves in the direction $-a$, is defined as follows. In the beginning, it moves in the direction a in the same state q_0, then makes $n-2$ further steps in the direction a, incrementing the number of state. Next, it crosses the bridge in the direction b and enters the last, n-th state, in which it moves in the direction a until it sees the label c_{acc}.

Definition 15. The graph-walking automaton $A = (Q, q_0, \delta, F)$ over the signature $\widetilde{S}$ consists of:

- the set of states $Q = \{q_0, \ldots, q_{n-1}\}$;
- the initial state q_0;
- the transition function δ, defined by

$$\begin{aligned}
\delta(q_0, c_0) &= (q_0, a)\\
\delta(q_i, c) &= (q_{i+1}, a), && \text{if } i \in \{0, \ldots, n-3\}\\
\delta(q_{n-2}, c) &= (q_{n-1}, b)\\
\delta(q_{n-1}, c) &= (q_{n-1}, a)
\end{aligned}$$

- the set of acceptance conditions $F = \{(q_{n-1}, c_{acc})\}$.

Lemma 5. *Every returning automaton A' that accepts the same set of graphs as A, and has at most $4nk$ states, can enter at least $n-1$ distinct states after transitions in the direction $-a$.*

The proof of Lemma 5 is inferred from the following lemma.

Lemma 6. *Let $n \geqslant 2$ and $k \geqslant 4$. Let an automaton with at most $4nk$ states operate on a graph $G^{accept}_{n,k}$. Assume that it begins its computation at one of the ends of the upper chain, $(-8nk, 1)$ or $(M + 8nk, 1)$, and assume that it arrives at one of the ends of the lower chain, $(-(n-1), -1) = v_0$ or $(M+8nk, -1)$, without visiting either end of the upper chain on the way. Then it must be v_0 that the automaton arrives to, and in the minimal sequence of states which it ultimately repeats periodically, it makes at least $n - 1$ moves in the direction a and at least $n - 1$ moves in the direction $-a$.*

Proof. All nodes visited by the automaton during this computation are labelled with c, and early in the computation it begins repeating the same sequence of states periodically. Let p be the minimal length of this period, with $p \leqslant 4nk$, and let $d_1 \ldots d_p$ be the corresponding sequence of directions.

The proof is slightly different for computations beginning at the node $(-8nk, 1)$ and at the node $(M + 8nk, 1)$.

Case 1: the automaton begins at the node $(M + 8nk, 1)$. Since the automaton eventually arrives at a coordinate $x \leqslant M$, the sequence $d_1 \ldots d_p$ has more occurences of $-a$ than of a. Hence, it is sufficient to show that the number of occurences of a is at least $n - 1$. The proof is by contradiction. Suppose that there are at most $n - 2$ occurrences of a. Let s be the difference between the number of occurrences of $-a$ and a. It is claimed that the automaton visits the node $(-8nk, 1)$, which would contradict the assumption that it never returns to either end of the upper chain.

Moving periodically, the automaton moves in directions a and $-a$, shifting by s edges to the left at each period, while also applying transitions in directions b and $-b$, which at first follows the loops. When the automaton reaches the point $x = M$ along the x axis, the same transitions may change its y-coordinate. As the automaton continues shifting horizontally, it will eventually reach $(M - (n - 1), 1)$ in the upper chain or $(M - (n - 1), -1)$ in the lower chain. It is claimed that, by this moment, it will have moved far enough from $x = M$, so that it would not return to that position on its way from right to left, and continue moving along the current chain. Indeed, since, by the assumption, the sequence $d_1 \ldots d_p$ has at most $n-2$ moves in the direction a, there is no way the automaton could get back.

Assume that the automaton remains on the upper chain, that is, passes through the node $(M - (n-1), 1)$, and does not visit $(M - (n-1), -1)$. Since the number $M = (4nk)!$ is divisible by s, the automaton reaches the coordinate $x = 0$ in the same state in which it has earlier arrived to the coordinate $x = M$. This means that it repeats the same transitions as before and again stays on the upper chain, that is, comes to the node $(-(n - 1), 1)$, without paying a visit to the node $(-(n - 1), -1) = v_0$. Therefore, it continues its periodic motion until it comes to the node $(-8nk, 1)$. A contradiction has been obtained.

The other possibility is that the automaton passes through the node $(M - (n - 1), -1)$ without visiting the node $(M - (n - 1), 1)$, and thus moves to the lower chain. The automaton arrives to the coordinate $x = 0$ in the same state as it arrived to $x = M$. It repeats the same transitions and again moves to the other chain, this time back to the upper chain. Then, as in the previous case, it arrives to the node $(-(n - 1), 1)$ without visiting $(-(n - 1), -1) = v_0$.

Case 2: the automaton begins at the node $(-8nk, 1)$. The automaton soon begins periodically repeating a sequence of states of length p, with $p \leqslant 4nk$, moving in the corresponding sequence of directions $d_1 \ldots d_p$. This time, moves in the direction a are more frequent than moves in the direction $-a$. Let s be their difference. This time it is enough to prove that moves in the direction $-a$ occur in the sequence at least $n - 1$ times.

The proof is by contradiction. Suppose that the sequence has fewer than $n-1$ occurrences of $-a$. Then, gradually shifting from left to right the automaton cannot reach the node $(-(n-1),-1)=v_0$.

While passing through the node $(0,1)$, the automaton may continue on the upper chain or move to the lower chain. In both cases it arrives to a node with coordinate $x=M$ in the same state in which it came to the coordinate $x=0$. If it moved to the lower chain at the first time, it returns to the upper chain at the second time; and if it stayed on the upper chain at the first time, it stays on it at the second time. Thus, the automaton continues on the upper chain and arrives to the node $(M+8nk,1)$. This contradicts the assumption that the automaton never returns to any end of the upper chain. □

Proof of Lemma 5. Consider the computation of A' on the graph $G^{accept}_{n,k}$. The automaton A' must visit the node $(M+8nk,1)$, since its label is the only difference between the graphs $G^{accept}_{n,k}$ and $G^{reject}_{n,k}$. Denote by t_0 the moment of the last visit to any of the nodes $(M+8nk,1)$ and $(-8nk,1)$. At some point after t_0, the automaton starts behaving periodically, and, by Lemma 6, in the minimal repeated sequence of states, the automaton makes at least $n-1$ moves in the direction $-a$. Since all states in this minimal period are pairwise distinct, the automaton A' enters at least $n-1$ distinct states after transitions in the direction $-a$. □

It remains to combine this lemma with the properties of the diode to obtain the desired theorem.

Theorem 4. *For every $k \geqslant 4$, there exists a signature with k directions, such that, for every $n \geqslant 2$, there is an n-state graph-walking automaton, such that every returning automaton recognizing the same set of graphs must have at least $2(n-1)(k-3)$ states.*

Proof. The proof uses the automaton A defined above. By Lemma 2, the n-state automaton A over the signature $\widetilde{S}$, is transformed to n-state automaton A' over the signature $\widetilde{S} \cup S_k$. The directions $\pm a$ are the same for $\widetilde{S}$ and S_k, and $\pm b$ in $\widetilde{S}$ are merged with $\pm b_1$ in S_k, so there are k directions in total.

For every graph G, the automaton A' accepts a graph $h_{n,k}(G)$ with $(a,-a)$-edges replaced by diodes, if and only if A accepts G. The automaton A' is the desired example: it is claimed that every returning automaton B recognizing the same set of graphs as A' has at least $2(n-1)(k-3)$ states.

Let B be any returning automaton with at most $4nk$ states recognizing these graphs. By Lemma 3, there is an automaton C over the signature $\widetilde{S}$ and with the same number of states, which accepts a graph G if and only if B accepts $h(G)$. This is equivalent to A accepting G, and so C and A accept the same set of graphs. Since B is returning, by Lemma 3, C is returning too. Then, Lemma 5 asserts that the automaton C may enter $n-1$ distinct states after moving in the direction $-a$.

Then, according to Lemma 3, the automaton B enters at least $n-1$ distinct states after traversing the diode backwards. Therefore, by Lemma 4, this automaton should have at least $2(k-3)(n-1)$ states. □

6 Lower bound on the size of halting automata

Every n-state GWA with k directions can be transformed to a halting GWA with $2nk+1$ states, as shown in Section 3. In this section, the following lower bound for this construction is established.

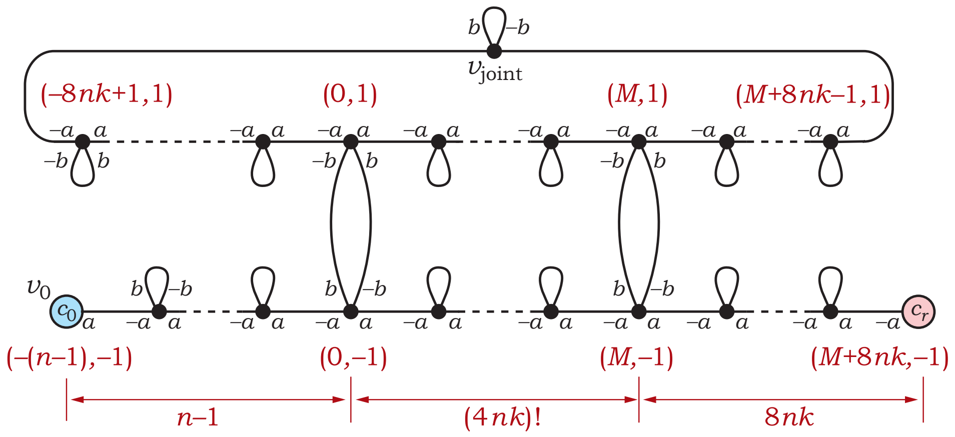

Figure 5: The graph G in the proof of Lemma 7.

Theorem 5. *For every $k \geqslant 4$, there is a signature with k directions, such that for every $n \geqslant 2$ there is an n-state GWA, such that every halting automaton accepting the same set of graphs has at least $2(n-1)(k-3)$ states.*

The argument shares some ideas with the earlier proof for the case of returning automata: the signature $\widetilde{S}$, the graphs $G^{accept}_{n,k}$ and $G^{reject}_{n,k}$, and the automaton A are the same as constructed in Section 5. The proof of Theorem 5 uses the following lemma, stated similarly to Lemma 5 for returning automata.

Lemma 7. *Every halting automaton A', accepting the same set of graphs as A and using at most $4nk$ states, must enter at least $n-1$ distinct states after transitions in the direction $-a$.*

Proof. Consider the computation of A' on the graph G, defined by modifying $G^{reject}_{n,k}$ as follows: the nodes $(M+8nk,1)$ and $(-8nk,1)$ are merged into a single node v_{joint}, with label c, and with a loop by $\pm b$.

The automaton A' must visit the node $v_{joint} = (M+8nk,1) = (-8nk,1)$, because this node is the only difference between G and $G^{accept}_{n,k}$. By the time the automaton reaches this node, it already executes a periodic sequence of states and directions. Since A' should halt at some time after visiting v_{joint}, it needs to stop its periodic behaviour, which requires visiting any label other than c. Hence, the automaton should reach one of the ends of the lower chain.

The argument in the proof of Lemma 6 is applicable to the segment of the computation from the last visit to v_{joint} until arriving to one the ends of the lower chain. Then, the periodically repeated sequence of states on this segment should contain at least $n-1$ occurrences of states reached after a transition in the direction $-a$. □

Proof of Theorem 5. The proof is analogous to the proof of Theorem 4. Lemma 7 is used instead of Lemma 5; and the application of Lemma 3 now uses the preservation of the halting property. □

7 Lower bound on the size of returning and halting automata

An n-state GWA over a signature with k directions can be transformed to an automaton that *both* halts on every input *and* accepts only in the initial node: a reversible automaton with $4nk+1$ states, described in Section 3, will do.

This section establishes a close lower bound on this transformation. The witness n-state automaton is the same as in Sections 5–6, for which Theorem 4 asserts that a returning automaton needs at least $2(n-1)(k-3)$ states, whereas Theorem 5 proves that a halting automaton needs at least $2(n-1)(k-3)$ states. The goal is to prove that these two sets of states must be disjoint, leading to the following lower bound.

Theorem 6. *For every $k \geqslant 4$, there exists a signature with k directions, such that for every $n \geqslant 2$, there is an n-state graph-walking automaton, such that every returning and halting automaton recognizing the same set of graphs must have at least $4(n-1)(k-3)$ states.*

As before, the automaton is obtained from A by Lemma 2. For the argument to proceed, the following property needs to be established.

Lemma 8 (cf. Lemma 5)**.** *Every returning and halting automaton that recognizes the same set of graphs as A, and has at most $4nk$ states, enters at least $2(n-1)$ distinct states after transitions in the direction $-a$.*

Consider any such returning and halting automaton. Since it is returning, as shown in Lemma 5, on the graph $G_{n,k}^{accept}$, the automaton uses a periodic sequence of states to return from $(M+8nk,1)$ to v_0. Since it is at the same time halting, Lemma 7 asserts that on the graph G it uses another periodic sequence of states to escape the cycle after visiting v_{joint}. Each of these two sequences makes transitions in the direction $-a$ in at least $n-1$ distinct states. It remains to prove that these sequences are disjoint.

Suppose the sequences have a common element, then they coincide up to a cyclic shift. Then it is possible to modify G so that the computation coming to v_{joint} later continued as the computation on $G_{n,k}^{accept}$, and led to acceptance.

Proof of Lemma 8. Let A' be any such returning and halting automaton. Since it is returning, on the graph $G_{n,k}^{accept}$, it must come to the node $(M+8nk,1)$ in order to see the label c_{acc}, and then find its way back to v_0. Consider the case when the automaton's last visit to any of the ends of the upper chain is to $(M+8nk,1)$ (the other case is proved similarly).

On the way from $(M+8nk,1)$ to v_0, the automaton sees only labels c. At some point not far from $(M+8nk,1)$, it starts behaving periodically, repeating a certain sequence of states $q_1,\ldots,q_p$, with a minimal period $p \leqslant 4nk$, and moving in a sequence of directions $d_1,\ldots,d_p$. The sequence $d_1,\ldots,d_p$ should move the automaton to the left, so it contains more occurrences of $-a$ than of a. Let $v_{\text{departure}}$ be the node at which the periodic behaviour starts, visited in the state q_1. Lemma 6 asserts that the direction $-a$ occurs in the sequence $d_1,\ldots,d_p$ at least $n-1$ times.

Since A' is halting, the arguments in Lemma 7 also apply. The lemma used a graph G with a node v_{joint} that merges $(M+8nk,1)$ and $(-8nk,1)$. The automaton A' must visit v_{joint} in order to tell G from $G_{n,k}^{accept}$. Because the node v_{joint} is far from any nodes labelled not with c, when the automaton first comes to v_{joint}, it already behaves periodically, with some minimal period t, with $t \leqslant 4nk$. Let $q'_1,\ldots,q'_t$ and $d'_1,\ldots,d'_t$ be the sequences of states and directions which it repeats periodically. Since this periodic behaviour must eventually stop, this sequence must lead A' from v_{joint} to one of the ends of the lower chain. Then, by Lemma 6, there are at least $n-1$ occurrences of $-a$ in the sequence $d'_1,\ldots,d'_t$.

It remains to prove that none of the states $q_1,\ldots q_p$ may coincide with any of the states $q'_1,\ldots,q'_t$. Suppose the contrary, that the periodic sequences are not disjoint. Then both cycles consist of the same states, cyclically shifted, and $p=t$. The goal is to construct a graph without any labels c_{acc}, which, however, would be accepted by A', leading to a contradiction.

For each $x \in \{0,\ldots,4nk\}$, consider a graph G_x, which is obtained from the graph G by prolonging the upper cycle with x extra nodes, all labelled with c. Then the graph G from Lemma 7 is the graph G_0.

The automaton is known to accept the graph $G_{n,k}^{accept}$ starting from the node $v_{departure}$ in the state q_1, without visiting either end of the upper chain. In each graph G_x, let $v_{departure}$ be the node located at the same distance from the nearest bridge to the lower chain. If, on the graph G_x, the automaton can be "lured" into this node in the state q_1, then this graph will be accepted.

Since the sequences of directions $d_1, \ldots, d_p$ and $d'_1, \ldots, d'_t$ coincide up to cyclic shift, the direction $-a$ occurs more often than a in $d'_1, \ldots, d'_t$. Then, when the automaton working on G visits the node v_{joint} for the first time, it is moving in the direction $-a$, and accordingly visits the node $v_{departure}$ after visiting v_{joint}.

Then it is possible to choose the length x of a sequence of edges inserted after v_{joint}, so that the automaton comes to the node $v_{departure}$ in the state q_1. Then the automaton A' accepts the graph G_x, and this is impossible.

A contradiction has thus been obtained, and hence the sequences of states $q_1, \ldots, q_p$ and $q'_1, \ldots, q'_t$ are disjoint. Together, they contain at least $2(n-1)$ distinct states that the automaton enters after transitions in the direction $-a$. □

The proof of the theorem is inferred from Lemmata 2, 3, 4 and 8, as in the earlier arguments.

8 Lower bound on the size of reversible automata

For the transformation of a GWA with n states and k directions to a reversible automaton, $4nk + 1$ states are sufficient. A close lower bound shall now be established.

Theorem 7. *For every $k \geqslant 4$, there exists a signature with k directions, such that for every $n \geqslant 2$, there is an n-state GWA, such that every reversible GWA recognizing the same set of graphs has at least $4(n-1)(k-3) - 1$ states.*

Proof. By Theorem 6, there is such an n-state automaton A' that every returning and halting automaton recognizing the same set of graphs has at least $4(n-1)(k-3)$ states. Suppose that there is a reversible automaton with fewer than $4(n-1)(k-3) - 1$ states that accepts the same graphs as A'. Let m be the number of states in it. Then, by the construction of reversing a reversible automaton given by Kunc and Okhotin [12], there is a returning and halting automaton with $m + 1$ states, that is, with fewer than $4(n-1)(k-3)$ states. This contradicts Theorem 6. □

9 Inverse homomorphisms: upper and lower bounds

An operation on graphs $h_{n,k}$ which replaces every $(a, -a)$-edge in the graph with diode $\Delta_{n,k}$ was defined in Section 4: this is an edge-replacement homomorphism.

Operations investigated in this section are *node-replacement homomorphisms* (or just *homomorphisms*) which replace every node with a subgraph depending on the label of this node.

Definition 16 (Graph homomorphism)**.** Let S and $\widehat{S}$ be two signatures, with the set of directions of S contained in the set of directions of $\widehat{S}$. A mapping $h \colon L(S) \to L(\widehat{S})$ is a (node-replacement) homomorphism, if, for every graph G over S, the graph $h(G)$ is constructed out of G as follows. For every node label a in S, there is a connected subgraph $h(a)$ over the signature $\widehat{S}$, which has an edge leading outside for every direction in D_a; these edges are called *external.* Then, $h(G)$ is obtained out of G by replacing every node v with a subgraph $h(v) = h(a)$, where a is the label of v, so that the edges that come out of v in G become the external edges of this copy of $h(a)$.

The subgraph $h(a)$ must contain at least one node. It contains an initial node if and only if the label a is initial.

Given a graph-walking automaton A and a homomorphism h, the inverse homomorphic image $h^{-1}(L(A))$ can be recognized by another automaton that, on a graph G, simulates the operation of A on the image $h(G)$. A construction of such an automaton is presented in the following theorem.

Theorem 8. *Let S be a signature with $k \geqslant 1$ directions, and let $\widehat{S}$ be a signature containing all directions from S. Let $h\colon L(S) \to L(\widehat{S})$ be a graph homomorphism between these signatures. Let A be a graph-walking automaton with n states that operates on graphs over $\widehat{S}$. Then there exists a graph-walking automaton B with $nk + 1$ states, operating on graphs over S, which accepts a graph G if and only if A accepts its image $h(G)$. If S contains a unique initial label, then it is sufficient to use nk states.*

In order to carry out the simulation of A on $h(G)$ while working on G, it is sufficient for B to remember the current state of A and the direction in which A has entered the image in $h(G)$ of the current node of B.

Proof. Let the first signature be $S = (D, -, \Sigma, \Sigma_0, (D_a)_{a\in\Sigma})$. Let $A = (Q, q_0, F, \delta)$. The new automaton is defined as $B = (P, p_0, E, \sigma)$.

When B operates on a graph G, it simulates the computation of A on $h(G)$. The set of states of B is $P = (Q \times D) \cup \{p_0\}$, where p_0 is a non-reenterable initial state; if there is only one initial label in S, then the state p_0 is omitted. All other states in B are of the form (q, d), where q is a state of A, and d is a direction in G. When B is at a node v in a state (q, d), it simulates A having entered the subgraph $h(v)$ from the direction d in the state q.

The transition function σ and the set of accepting states E of B are defined by simulating A on subgraphs. For a state of the form (q, d), and for every label $a \in \Sigma$, with $-d \in D_a$, the goal is to decide whether $((q, d), a)$ is an accepting pair, and if not, then what is the transition $\sigma((q, d), a)$. To this end, the automaton A is executed on the subgraph $h(a)$, entering this subgraph in the direction d in the state q. If A accepts without leaving $h(a)$, then the pair $((q, d), a)$ is defined as accepting in B. Otherwise, if A rejects or loops inside $h(a)$, then $\sigma((q, d), a)$ is left undefined. If A leaves $h(a)$ by an external edge in the direction d' in a state q', then B has a transition $\sigma((q, d), a) = ((q', d'), d')$.

If S has a unique initial label, $\Sigma_0 = \{a_0\}$, then the automaton A always starts in the subgraph $h(a_0)$, and its initial state can be defined by the same method as above, by considering the computation of A on this subgraph starting in the initial state at the initial node. If A accepts, rejects or loops without leaving the subgraph $h(a_0)$, then it is sufficient to have B with a single state, in which it gives an immediate answer. If A leaves the subgraph in the direction d, changing from q to a state q', then the state $(q, -d)$ can be taken as the initial state of B; then B starts simulating the computation of A from this point.

If there are multiple initial labels in Σ_0, then the automaton B uses a separate initial state p_0. The transitions in p_0 and its accepting status are defined only on initial labels, as follows. Let $a_0 \in \Sigma_0$ be an initial label, and consider the computation of A on the subgraph $h(a_0)$, starting at the initial node therein, in the initial state. If A accepts inside $h(a_0)$, then (p_0, a_0) is an accepting pair. Otherwise, if A rejects or loops without leaving $h(a_0)$, then $\sigma(p_0, a_0)$ is not defined. If A leaves $h(a_0)$ in the direction d' in the state q', then the transition is $\sigma(p_0, a_0) = ((q', d'), d')$.

The automaton B has nk or $nk + 1$ states, and it operates over S. The following correctness claim for this construction can be proved by induction on the number of steps made by B on G.

Claim 8. *Assume that the automaton B, after $t \geqslant 1$ steps of its computation on G, is in a state (q', d') at a node v. Then, in the computation of A on $h(G)$ there is a moment $\widehat{t} \geqslant t$, at which A enters the subgraph $h(v)$ in the direction d' in the state q' (the only exception is the initial state of B in the case p_0 is not used).*

It follows that the automaton B thus defined indeed accepts a graph G if and only if A accepts $h(G)$. □

It turns out that this expected construction is actually optimal, as long as the initial label is unique: the matching lower bound of nk states is proved below.

Theorem 9. *For every $k \geqslant 9$, there is a signature S with k directions and a homomorphism $h \colon L(S) \to L(S)$, such that for every $n \geqslant 4$, there exists an n-state automaton A over the signature S, such that every automaton B, which accepts G if and only if A accepts $h(G)$, has at least nk states.*

Proving lower bounds on the size of graph-walking automata is generally not easy. Informally, it has to be proved that the automaton must remember a lot; however, in theory, it can always return to the initial node and recover all the information is has forgotten. In order to eliminate this possibility, the initial node shall be placed in a special subgraph H_{start}, from which the automaton can easily get out, but if it ever needs to reenter this subgraph, finding the initial node would require too many states. This subgraph is constructed in the following lemma; besides H_{start}, there is another subgraph $H_{\text{dead end}}$, which is identical to H_{start} except for not having an initial label; then, it would be hard for an automaton to distinguish between these two subgraphs from the outside, and it would not identify the one in which it has started.

Lemma 9. *For every $k \geqslant 4$ there is a signature S_{start} with k directions, with two pairs of opposite directions a, $-a$ and b, $-b$, such that for every $n \geqslant 2$ there are two graphs H_{start} and $H_{\text{dead end}}$ over this signature, with the following properties.*

I. *The subgraph H_{start} contains an initial node, whereas $H_{\text{dead end}}$ does not; both have one external edge in the direction a.*

II. *There is an n-state automaton, which begins its computation on H_{start} in the initial node, and leaves this subgraph by the external edge.*

III. *Every automaton with fewer than $2(k-3)(n-1)$ states, having entered H_{start} and $H_{\text{dead end}}$ by the external edge in the same state, either leaves both graphs in the same state, or accepts both, or rejects both, or loops on both.*

The proof reuses a graph constructed in Section 5. Originally, it was used to show that there is an n-state graph-walking automaton, such that every automaton that accepts the same graphs and returns to the initial node after acceptance must have at least $2(k-3)(n-1)$ states. A summary of the proof is included for completeness, as well as adapted to match the statement of the lemma.

Summary of a proof. The graph is constructed in two stages. First, there is a graph G presented in Figure 6, with two long chains of nodes in the direction $\pm a$ connected by two bridges in the direction $\pm b$, which are locally indistinguishable from loops by $\pm b$ at other nodes. In order to get from the initial node v_0 to the node v_{exit}, an n-state automaton counts up to $n-1$ to locate the left bridge, then crosses the bridge and continues moving to the right. The journey back from v_{exit} to v_0 requires moving in the direction $-a$ in at least $n-1$ distinct states by Lemma 5.

In order to get a factor of $2(k-3)$, another construction is used on top of this. Every $(a, -a)$-edge in the horizontal chains is replaced with a certain subgraph called a *diode*, with $9(4nk)!+2$ nodes. This subgraph is easy to traverse in the direction a: an automaton can traverse it in a single state, guided by labels inside the diode, so that the graph G in Figure 6, with diodes substituted, can be traversed from v_0 to v_{exit} using n states. However, as its name implies, the diode is hard to traverse backwards: for every state, in which the automaton finishes the

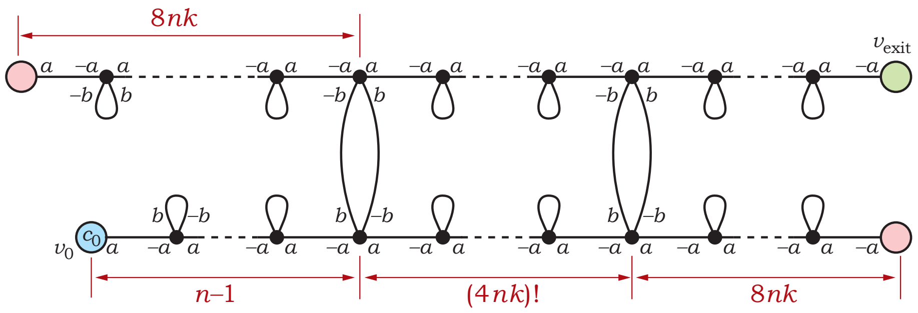

Figure 6: The graph G.

traversal in the direction $-a$, it must contain $2(k-3)-1$ extra states by lemma 4. Combined with the fact that there need to be at least $n-1$ states after moving by $-a$ for the automaton to get from v_{exit} to v_0, this shows that $2(k-3)(n-1)$ states are necessary to get from v_{exit} to v_0 after the substitution of diodes.

Let G_{diodes} be the graph in Figure 6, with diodes substituted. It is defined over a signature with k directions, and among them the directions $\pm a$ and $\pm b$. This signature is taken as S_{start} in Lemma 9.

The graph H_{start} is defined by removing the node v_{exit} from G_{diodes}, and the edge it was connected by becomes an external edge in the direction a. The other graph $H_{\text{dead end}}$ is obtained by relabelling the initial node v_0, so that it is no longer initial. An n-state automaton that gets out of H_{start} has been described above.

Every automaton that enters H_{start} or $H_{\text{dead end}}$ from the outside needs at least $2(k-3)(n-1)$ states to get to v_0, because returning from v_{exit} to v_0 on G_{diodes} requires this many states. Then, an automaton with fewer states never reaches v_0, and thus never encounters any difference between these subgraphs. Thus, it carries out the same computation on both subgraphs H_{start} and $H_{\text{dead end}}$, with the same result. □

Now, using the subgraphs H_{start} and $H_{\text{dead end}}$ as building blocks, the next goal is to construct a subgraph which encodes a number from 0 to $n-1$, so that this number is easy to calculate along with getting out of this subgraph for the first time, but if it is ever forgotten, then it cannot be recovered without using too many states. For each number $i \in \{0, \dots, n-1\}$ and for each direction $d \in D$, this is a graph $F_{i,d}$ that contains the initial label and encodes the number i, and a graph F_d with no initial label that encodes no number at all.

Lemma 10. *For every $k \geqslant 4$ there is a signature S_F obtained from S_{start} by adding several new node labels, such that, for every $n \geqslant 2$ there are subgraphs $F_{i,d}$ and F_d, for all $i \in \{0, \dots, n-1\}$ and $d \in D$, with the following properties.*

I. Each subgraph $F_{i,d}$ and F_d has one external edge in the direction d. Subgraphs of the form $F_{i,d}$ have an initial node, and subgraphs F_d do not have one.

II. There is an automaton with states $\{q_0, \dots, q_{n-1}\}$, which, having started on every subgraph $F_{i,d}$ in the initial node, eventually gets out in the state q_i.

III. Every automaton with fewer than $2(k-3)(n-1)$ states, having entered $F_{i,d}$ and F_d with the same d by the external edge in the same state, either leaves both subgraphs in the same state, or accepts both, or rejects both, or loops on both.

Each subgraph $F_{i,d}$ is a chain of n nodes, with the subgraph H_{start} attached at the i-th position, and with $n-1$ copies of $H_{\text{dead end}}$ attached at the remaining positions, as illustrated in Figure 7. The automaton in Part II gets out of H_{start} and then moves along the chain to the left,

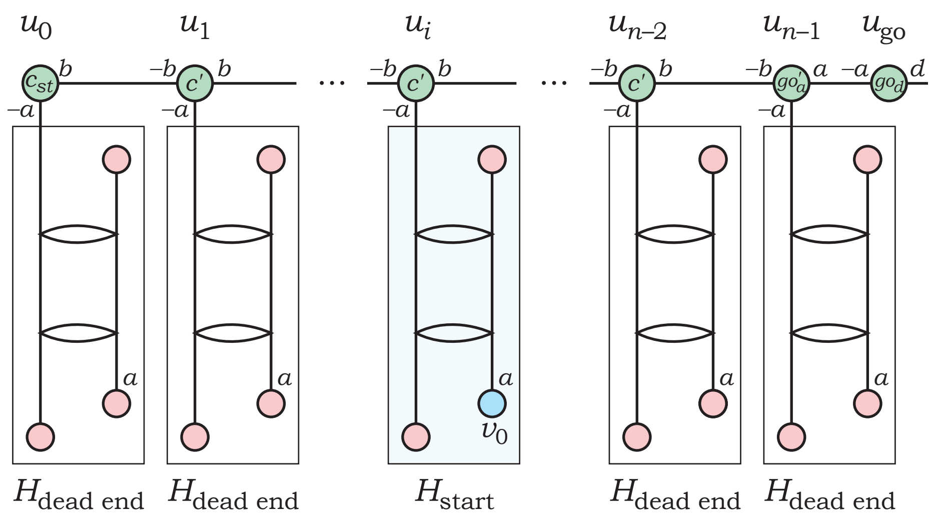

Figure 7: The subgraph $F_{i,d}$, with $d \neq -a$; for $d = -a$ the subgraph has u_{n-1} labelled with go'_b, and a $(b, -b)$-edge to u_{go}.

counting the number of steps, so that it gets out of the final node u_{go} in the state q_i. The proof of Part III relies on Lemma 9 (part III): if an automaton enters $F_{i,d}$ and F_d from the outside, it ends up walking over the chain and every time it enters any of the attached subgraphs H_{start} and $H_{\text{dead end}}$, it cannot distinguish between them and continues in the same way on all $F_{i,d}$ and F_d.

Proof. The new signature S_F has the following new non-initial node labels: $\{c_{st}, c', go'_a, go'_b\} \cup \{go_d \mid d \in D\}$. The labels have the following sets of directions: $D_{c_{st}} = \{-a, b\}$, $D_{c'} = \{-a, -b, b\}$, $D_{go'_a} = \{-a, -b, a\}$, $D_{go'_b} = \{-a, -b, b\}$, $D_{go_d} = \{-a, d\}$ with $d \neq -a$, and $D_{go_{-a}} = \{-b, -a\}$.

For $n \geqslant 2$, the subgraphs $F_{i,d}$ and F_d are constructed as follows, using the subgraphs H_{start} and $H_{\text{dead end}}$ given in Lemma 9.

The subgraph $F_{i,d}$, illustrated in Figure 7, is a chain of nodes $u_0, \ldots, u_{n-1}, u_{go}$; the first $n-1$ nodes are linked with $(b, -b)$-edges. The node u_{n-1} is linked to u_{go} by an edge $(a, -a)$ if $d \neq -a$, and by an edge $(b, -b)$ for $d = -a$. The label of u_0 is c_{st}, the nodes $u_1, \ldots, u_{n-2}$ all have label c', and u_{n-1} is labelled with go'_a, if $d \neq -a$, or with go'_b, if $d = -a$. The node u_{go} has label go_d, and has an external edge in the direction d.

Each node $u_0, \ldots, u_{n-1}$ has a subgraph H_{start} or $H_{\text{dead end}}$ attached in the direction $-a$. This is H_{start} for u_i, and $H_{\text{dead end}}$ for the rest of these nodes.

The subgraph F_d is the same as $F_{i,d}$, except for having $H_{\text{dead end}}$ attached to all nodes $u_0, \ldots, u_{n-1}$.

It is left to prove that the subgraphs $F_{i,d}$ and F_d thus constructed satisfy the conditions in the lemma.

Part II of this lemma asserts that there is an n-state automaton that gets out of $F_{i,d}$ in the state q_i, for all i and d. Having started in the initial node inside a subgraph H_{start}, the automaton operates as the n-state automaton given in Lemma 9(part II) which leaves H_{start} in some state q. Denote this state by q_{n-1}, and let $\{q_0, \ldots, q_{n-2}\}$ be the remaining states (it does not matter which of these states is initial). Then the automaton follows the chain of nodes to the right, decrementing the number of the state at each node labelled with c_{st} or c'. At the nodes labelled with go'_a, go'_b or go_d, the automaton continues to the right without changing its state. Thus, for each subgraph $F_{i,d}$, the automaton gets out in the state q_i, as desired.

Turning to the proof of Part III, consider an automaton with fewer than $2(k-3)(n-1)$

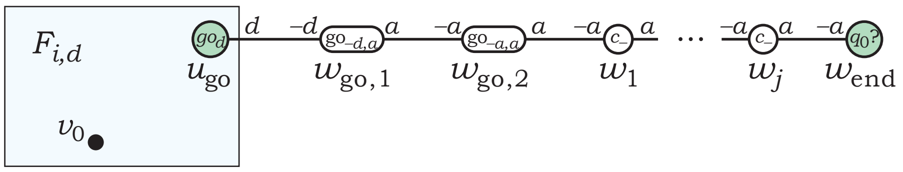

Figure 8: The graph $G_{i,j,d}$, with $d \neq -a$; for $d = -a$ the graph has $w_{go,1}$ labelled with $go_{a,b}$ and $w_{go,2}$ labelled with $go_{-b,a}$, linked with a $(b, -b)$-edge.

states and let $d \in D$ be any direction. The subgraphs $F_{i,d}$ for various i, as well as the subgraph F_d, differ only in the placement of the subgraph H_{start} among the subgraphs $H_{\text{dead end}}$, or in its absense. On each of the subgraphs $F_{i,d}$ or F_d, the automaton first moves over the chain of nodes $u_0, \ldots, u_{n-1}, u_{go}$, which is the same in all subgraphs. Whenever, at some node u_j, it enters the j-th attached subgraph, whether it is H_{start} or $H_{\text{dead end}}$, according to Lemma 9, it is not able to distinguish between them, and the computation has the same outcome: it either emerges out of each of the attached subgraphs in the same state, or accepts on either of them, etc. If the computation continues, it continues from the same state and the same node in all $F_{i,d}$ and F_d, and thus the computations on all these subgraphs proceed in the same way and share the same outcome. □

Proof of Theorem 9. The signature S is S_F from Lemma 10, with a few extra node labels. Let the directions be cyclically ordered, so that $next(d)$ is the next direction after d and $prev(d)$ is the previous direction. The order is chosen so that, for each direction d, its opposite direction is neither $next(d)$ nor $next(next(d))$.

The new node labels, all non-initial, are: $\{\, go_{-d,a} \mid d \in D \setminus \{-a\}\,\} \cup \{go_{a,b}, c_-, q_0?\} \cup \{\, d? \mid d \in D\,\} \cup \{\, acc_d, rej_d \mid d \in D\,\}$. These labels have the following sets of directions: $D_{go_{d_1,d_2}} = \{d_1, d_2\}$; $D_{c_-} = \{-a, a\}$; $D_{q_0?} = \{-a\}$; $D_{d?} = D$ for all $d \in D$; $D_{acc_d} = D_{rej_d} = \{-d, -next(d), next(next(d))\}$ for all $d \in D$, where the directions $-d, -next(d), next(next(d))$ are pairwise distinct by assumption.

The homomorphism h affects only new labels of the form $d?$, with $d \in D$, whereas the rest of the labels are unaffected. Each label $d?$, for $d \in D$, has $D_{d?} = D$, and is replaced with a circular subgraph $h(d?)$ as illustrated in Figure 9. Its nodes are v_e, for all $e \in D$. The node v_d has label acc_d, and every node v_e, with $e \neq d$, is labelled with rej_e. Each node v_e, with $e \in D$, is connected to the next node $v_{next(e)}$ by an edge in the direction $next(next(e))$; also it has an external edge in the direction $-e$. Overall, the subgraph $h(d?)$ has an external edge in each direction, as it should have, since $D_{d?} = D$. When the automaton enters this subgraph in the image, it knows the direction it came from, whereas in the original graph, it has to remember this direction in its state.

The graph $G_{i,j,d}$ is defined by taking $F_{i,d}$ from Lemma 10 and attaching to it a chain of $j+3$ nodes, as shown in Figure 8. The new nodes are denoted by $w_{go,1}, w_{go,2}, w_1, \ldots, w_j, w_{end}$, where the external edge of $F_{i,d}$ is linked to $w_{go,1}$ in the direction d. If $d \neq -a$, then the nodes $w_{go,1}$ and $w_{go,2}$ have labels $go_{-d,a}$ and $go_{-a,a}$, and are connected with an $(a, -a)$-edge; and if $d = -a$, then the labels are $go_{a,b}$ and $go_{-b,a}$, and the edge is $(b, -b)$. The nodes $w_1, \ldots, w_j$ are labelled with c_-, the label of w_{end} is $q_0?$, and all of them are connected with $(a, -a)$-edges.

The graph $G_{i,d,d'}$ is presented in Figure 9 for the case $d = d'$. It has a subgraph $F_{i,d}$ with the initial node, and $k-1$ subgraphs F_e, with $e \in D \setminus \{d\}$. The external edges of these k subgraphs are all linked to a new node v labelled with $d'?$.

Claim 9. *There exists an n-state automaton A, which accepts $h(G_{i,j,d})$ if and only if $i = j$, and which accepts $h(G_{i,d,d'})$ if and only if $d = d'$.*

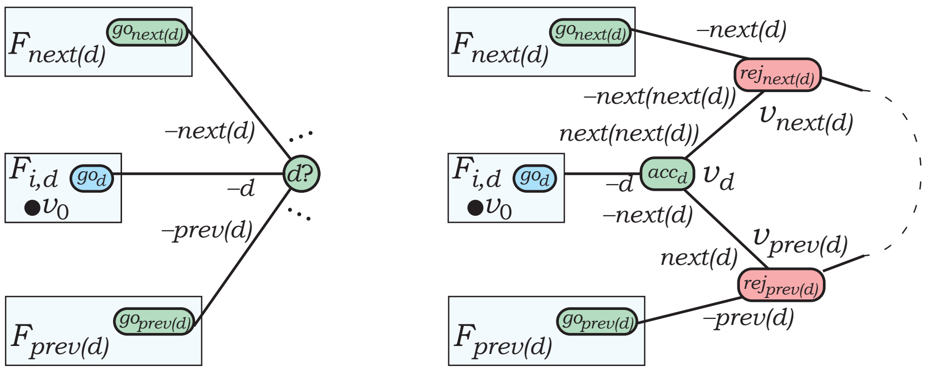

Figure 9: The graph $G_{i,d,d}$ and its image $h(G_{i,d,d})$.

Proof. The automaton is based on the one defined in Lemma 10 (part II). It works over the signature S_F and has n states $\{q_0, \ldots, q_{n-1}\}$. Having started on a graph $F_{i,d}$, it eventually gets out in the state q_i. It remains to define the right transitions by the new labels in the signature S. At each label go_{d_1,d_2}, the automaton moves in the direction d_2 in the same state. At a label c_- the automaton decrements the number of its current state and moves in the direction a. If it ever comes to a label c_- in the state q_0, it rejects. At the label $q_0?$, the automaton accepts if its current state is q_0, and rejects in all other states. Turning to the labels introduced by the homomorphism, for all $d \in D$, the automaton immediately accepts at acc_d and rejects at rej_d, regardless of its current state.

To see that the automaton A operates as claimed, first consider its computation on the graph $h(G_{i,j,d}) = G_{i,j,d}$. It starts at the initial node in $F_{i,d}$, then leaves $F_{i,d}$ in the state q_i, passes through the nodes $w_{go,1}$ and $w_{go,2}$ without changing its state, and then decrements the number of the state at the nodes $w_1, \ldots, w_j$. If $i = j$, then the automaton A makes j decrementations, and arrives to the node with the label $q_0?$ in the state q_0, and accordingly accepts. If $i > j$, then it comes to $q_0?$ in the state $q_{i-j} \neq q_0$ and rejects. If $i < j$, then A enters the state q_0 at one of the labels c_-, and rejects there. Thus, A works correctly on graphs of the form $h(G_{i,j,d})$.

In the graph $h(G_{i,d,d'})$, the homomorphism has replaced the node v from $G_{i,d,d'}$ with a ring of nodes with labels $acc_{d'}$ and rej_e, with $e \neq d'$. The automaton A starts in the subgraph $F_{i,d}$ and leaves it in the direction d, thus entering the ring at the node v_d. Then, if $d = d'$, it sees the label $acc_{d'}$ and accepts, and otherwise it sees the label rej_d and rejects. The automaton does not move along the circle. ☐

Claim 10. *Let an automaton B accept a graph G if and only if A accepts $h(G)$. Then B has at least nk states.*

Proof. The proof is by contradiction. Suppose that B has fewer than nk states. Since $nk \leqslant 2 \cdot \frac{2}{3}k \cdot \frac{3}{4}n \leqslant 2(k-3)(n-1)$, Lemma 10 (part III) applies, and the automaton B cannot distinguish between the subgraphs $F_{i,d}$ and F_d if it enters them from the outside.

On the graph $G_{i,j,d}$, the automaton must check that i is equal to j, where the latter is the number of labels c_- after the exit from $F_{i,d}$. In order to check this, B must exit this subgraph. Denote by $q_{i,d}$ the state, in which the automaton B leaves the subgraph $F_{i,d}$ for the first time. There are nk such states $\{\, q_{i,d} \mid i = 0, \ldots, n-1; d \in D \,\}$, and since B has fewer states, some of these states must coincide. Let $q_{i,d} = q_{j,d'}$, where $d \neq d'$ or $i \neq j$. There are two cases to consider.

- Case 1: $d \neq d'$. The automaton B must accept $G_{i,d,d}$ and reject $G_{j,d',d}$. On either graph, it first arrives to the corresponding node v in the same state $q_{i,d} = q_{j,d'}$, without remembering the last direction taken. Then, in order to tell these graphs apart, the automaton must carry out some further checks. However, every time B leaves the node v in any direction $e \in D$, it enters a subgraph, which is either the same in $G_{i,d,d}$ and $G_{j,d',d}$ (if $e \neq d, d'$), or it is a subgraph that is different in the two graphs, but, according to Lemma 10 (part III), no automaton of this size can distinguish between these subgraphs. Therefore, B either accepts both graphs, or rejects both graphs, or loops on both, which is a contradiction.

- Case 2: $d = d'$ and $i \neq j$. In this case, consider the computations of B on the graphs $G_{i,j,d}$ and $G_{j,j,d}$: the former must be rejected, the latter accepted. However, by the assumption, the automaton leaves $F_{i,d}$ and $F_{j,d}$ in the same state $q_{i,d} = q_{j,d}$. From this point on, the states of B in the two computations are the same while it walks outside of $F_{i,d}$ and $F_{j,d}$, and each time it reenters these subgraphs, by Lemma 10 (part III), it either accepts both, or rejects both, or loops on both, or leaves both in the same state. Thus, the whole computations have the same outcome, which is a contradiction.

$\square$

The contradiction obtained shows that B has at least nk states. $\square$

10 A characterization of regular tree languages

The next question investigated in this thesis is whether the family of graph languages recognized by graph-walking automata is closed under homomorphisms. In this section, non-closure is established already for tree-walking automata and for injective homomorphisms.

The proof is based on a seemingly unrelated result. Consider the following known representation of regular string languages.

Theorem A (Latteux and Leguy [15]). For every regular language $L \subseteq \Sigma^*$ there exist alphabets Ω and Γ, a special symbol $\#$, and homomorhisms $f\colon \Omega^* \to \#^*$, $g\colon \Omega^* \to \Gamma^*$ and $h\colon \Sigma^* \to \Gamma^*$, such that $L = h^{-1}(g(f^{-1}(\#)))$.

A similar representation shall now be established for regular tree languages, that is, those recognized by deterministic bottom-up tree automata.

For uniformity of notation, tree and tree-walking automata shall be represented in the notation of graph-walking automata, as in Section 2, which is somewhat different from the notation used in the tree automata literature. This is only notation, and the trees and the automata are mathematically the same.

Definition 17. A signature $S = (D, -, \Sigma, \Sigma_0, (D_a)_{a\in\Sigma})$ is a *tree signature*, if it is of the following form. The set of directions is $D = \{+1, -1, \ldots, +k, -k\}$, for some $k \geqslant 1$, where directions $+i$ and $-i$ are opposite to each other. For every label $a \in \Sigma$, the number of its children is denoted by $\operatorname{rank} a$, with $0 \leqslant \operatorname{rank} a \leqslant k$. Every initial label $a_0 \in \Sigma_0$ has directions $D_{a_0} = \{+1, \ldots, +\operatorname{rank} a_0\}$. Every non-initial label $a \in \Sigma \setminus \Sigma_0$ has the set of directions $D_a = \{-d, +1, \ldots, +\operatorname{rank} a\}$, for some $d \in \{1, \ldots, k\}$.

A tree is a connected graph over a tree signature.

This definition implements the classical notion of a tree as follows. The initial node is the root of a tree. In a node v with label a, the directions $\{+1, \ldots, +\operatorname{rank} a\}$ lead to its children. The child in the direction $+i$ accordingly has direction $-i$ to its parent. This direction to the parent is absent in the root node. Labels a with $\operatorname{rank} a = 0$ are used in the leaves.

Definition 18. A (deterministic bottom-up) tree automaton over a tree signature $S = (D, -, \Sigma, \Sigma_0, (D_a)_{a\in\Sigma})$ is a triple $A = (Q, q_{acc}, (\delta_a)_{a\in\Sigma})$, where

- Q is a finite set of states;
- $q_{acc} \in Q$ is the accepting state, effective in the root node;
- $\delta_a \colon Q^{\operatorname{rank} a} \to Q$, for each $a \in \Sigma$, is a function computed at the label a. If $\operatorname{rank} a = 0$, then δ_a is a constant that sets the state in a leaf.

Given a tree T over a signature S, a tree automaton A computes the state in each node, bottom-up. The state in each leaf v labelled with a is set to be the constant $\delta_a()$. Once a node v labelled with a has the states in all its children computed as $q_1, \ldots, q_{\operatorname{rank} a}$, the state in the node v is computed as $\delta_a(q_1, \ldots, q_{\operatorname{rank} a})$. This continues until the value in the root is computed. If it is q_{acc}, then the tree is accepted, and otherwise it is rejected. The tree language recognized by A is the set of all trees over S that A accepts. A tree language is called regular if it is recognized by some tree automaton.

The generalization of Theorem A to the case of trees actually uses only two homomorphisms, not three. The inverse homomorphism f^{-1} in Theorem A is used to generate the set of all strings with a marked first symbol out of a single symbol. Trees cannot be generated this way. The characterization given below starts from the set of all trees over a certain signature, in which the root is already marked by definition; this achieves the same effect as $f^{-1}\big(\{\#\}\big)$ in Theorem A. The remaining two homomorphisms do basically the same as in the original result, only generalized to trees.

Theorem 10. *Let L be a regular tree language over some tree signature S_{reg}. Then there exist tree signatures S_{comp} and S_{mid}, and injective homomorphisms $g \colon L(S_{comp}) \to L(S_{mid})$ and $h \colon L(S_{reg}) \to L(S_{mid})$, such that $L = h^{-1}(g(L(S_{comp})))$.*

Proof. The signature S_{mid} extends S_{reg} with a few new non-initial node labels; the set of directions is preserved. The new labels are k labels for internal nodes, $e_1, \ldots, e_k$, with $\operatorname{rank} e_i = k$ and $D_{e_i} = \{-i, +1, \ldots, +k\}$, and k more labels for leaves, $end_1, \ldots, end_k$, with $\operatorname{rank} end_i = 0$ and $D_{end_i} = \{-i\}$. These labels are used to construct a *fishbone subgraph*: a fishbone subgraph of length ℓ in the direction i is a chain of ℓ internal nodes, all labelled with e_i, which begins and ends with external edges in the directions $-i$ and $+i$; all directions except $\pm i$ lead to leaves labelled with end_j.

An injective homomorphism $h \colon L(S_{reg}) \to L(S_{mid})$ is defined to effectively replace each $(+i, -i)$-edge with a fishbone subgraph of length n in the direction i, as illustrated in Figure 10, without affecting the original nodes and their labels. Formally, h replaces each non-initial node labelled with $a \in \Sigma \setminus \Sigma_0$ as follows. Let $D_a = \{-d, +1, \ldots, +\operatorname{rank} a\}$ be its set of directions. Then, $h(a)$ is the following subgraph: it consists of a node with the same label a, a fishbone subgraph of length n attached in the direction $-d$, and $\operatorname{rank} a$ external edges in the directions $+1, \ldots, +\operatorname{rank} a$. The initial node is mapped to itself.

The main idea of the construction is to take a tree accepted by A and annotate node labels with the states in the accepting computation of A on this tree. Another homomorphism g maps such annotated trees to trees over the signature S_{mid}, with fishbones therein. Annotated trees that correctly encode a valid computation are mapped to trees with all fishbones of length exactly n; then, h^{-1} decodes the original tree out of this encoding. On the other hand, any mistakes in the annotation are mapped by g to a tree with some fishbones of length other than n, and these trees have no pre-images under h.

Trees with annotated computations are defined over the signature S_{comp}. This signature uses the same set of directions as in S_{reg}. For every non-initial label $a \in \Sigma \setminus \Sigma_0$ in S_{reg}, the signature S_{comp} has $|Q|^{\operatorname{rank} a}$ different labels corresponding to all possible vectors of states in its children. Thus, for every $\boldsymbol{q} = (q_1, \ldots, q_{\operatorname{rank} a}) \in Q^{\operatorname{rank} a}$, there is a non-initial label $(a, \boldsymbol{q})$ in S_{comp}, with

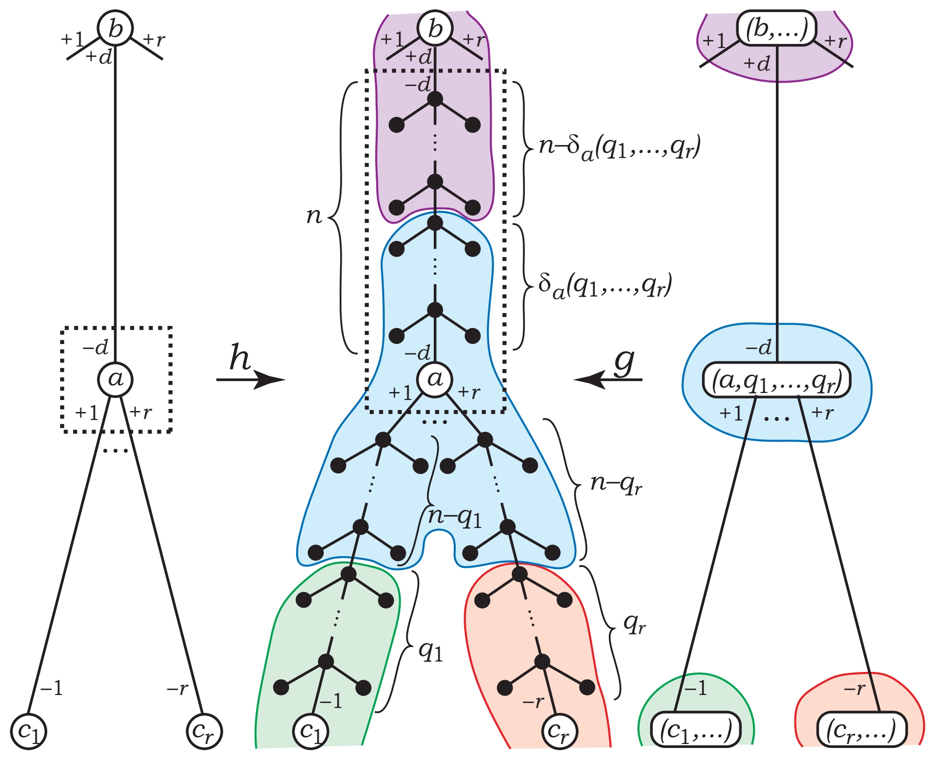

Figure 10: Homomorphisms h and g mapping the original tree T (left) and the corresponding valid annotated tree T_{comp} (right) to the same tree with fishbones.

$\operatorname{rank}(a, \boldsymbol{q}) = \operatorname{rank} a$ and $D_{(a,\boldsymbol{q})} = D_a$. For every initial label $a_0 \in \Sigma_0$ in S_{reg}, the signature S_{comp} contains only those initial labels $(a_0, \boldsymbol{q})$, for which the vector $\boldsymbol{q} \in Q^{\operatorname{rank} a_0}$ of states in the children leads to acceptance, that is, $\delta_{a_0}(\boldsymbol{q}) = q_{acc}$. The rank and the set of directions are also inherited: $\operatorname{rank}(a_0, \boldsymbol{q}) = \operatorname{rank} a_0$ and $D_{(a_0,\boldsymbol{q})} = D_{a_0}$. There is at least one initial label in S_{comp}, because $L \neq \varnothing$. If $\operatorname{rank} a = 0$, then the set $Q^{\operatorname{rank} a}$ contains a unique vector $\boldsymbol{q}$ of length 0. Such a label has only one copy $(a, \boldsymbol{q})$ in the signature S_{comp}, or none at all, if $a = a_0 \in \Sigma_0$ and $\delta_a(\boldsymbol{q}) \neq q_{acc}$.

For every tree T over S_{reg} that is accepted by A, the accepting computation of A on T is represented by a tree T_{comp} over the signature S_{comp}, in which every label is annotated with the vector of states in the children of this node. Annotated trees that do not encode a valid computation have a mismatch in at least one node v, that is, the state in some i-th component of the vector in the label does not match the state computed in the i-th child. It remains to separate valid annotated trees from invalid ones.

The homomorphism $g\colon L(S_{comp}) \to L(S_{mid})$ is formally defined as follows. Let $(a, \boldsymbol{q})$ be a non-root label with $\boldsymbol{q} = (q_1, \ldots, q_{\operatorname{rank} a}) \in Q^{\operatorname{rank} a}$ and $D_a = \{-d, +1, \ldots, +\operatorname{rank} a\}$. Then, g maps $(a, \boldsymbol{q})$ to a subgraph $g((a, \boldsymbol{q}))$, which is comprised of a central node v_{center} labelled with a, with fishbone subgraphs attached in all directions. The direction $-d$ is attached to the bottom of a fishbone graph in the direction d of length $\delta_a(\boldsymbol{q})$. The subgraph attached in each direction $+i$ is a fishbone of length $n - q_i$ in the direction i. The external edges of the subgraph $g((a, \boldsymbol{q}))$ come out of these fishbones. If $n - q_i = 0$, then the fishbone of length 0 is an external edge in the direction $+i$. The image of a root label $(a_0, \boldsymbol{q})$ under g is defined in the same way, except for not having a direction $-d$ and the corresponding fishbone.

Images of trees under the homomorhism g are of the following form.

Claim 11. *Let $\widetilde{T}$ be an annotated tree over the signature S_{comp}, with the nodes $v^1, \ldots, v^m$ labelled with $(a^1, \boldsymbol{q^1}), \ldots, (a^m, \boldsymbol{q^m})$. Then the tree $g(\widetilde{T})$ is obtained from $\widetilde{T}$ as follows: every* *label $(a^t, \boldsymbol{q^t})$ is replaced with a^t, and every edge $(+i, -i)$ linking a parent v^s to a child v^t in $\widetilde{T}$ is replaced with a fishbone of length $n - q_i^s + \delta_{a^t}(\boldsymbol{q^t})$ in the direction i.*

The image of all valid annotated trees under g is exactly $h(L)$.

Claim 12. *Let T be a tree accepted by A, and let T_{comp} be an annotated tree that encodes the computation of A on T. Then, the homomorphism g maps T_{comp} to $h(T)$.*

Indeed, if an annotated tree represents a valid computation, then, in Claim 11, $q_i^s = \delta_{a^t}(\boldsymbol{q^t})$ holds for every pair of a parent v_s and its i-th child v_t, and thus all fishbones are of length n, as in $h(T)$. For the same reason, g maps invalid annotated trees to trees without pre-images under h. Therefore, $h^{-1}(g(L(S_{comp}))) = L$.

The homomorphism h is injective, because it does not affect the node labels and only attaches fixed subgraphs to them. On the other hand, g erases the second components of labels, and its injectivity requires an argument.

Claim 13. *The homomorphism g is injective.*

Proof. Let T and T' be trees over S_{comp} that are mapped to the same tree $g(T) = g(T')$. It is claimed that $T = T'$. By Claim 11, both trees T and T' have the same set of nodes and the same edges between these nodes, as well as the same first components of their labels.

It remains to show that the second components of labels at the corresponding nodes of T and T' also coincide. This is proved by induction, from leaves up to the root. For a leaf, the second component is an empty vector in both trees. For every internal node v^s in these trees, let $(a^s, \boldsymbol{q^s})$ be its label in T and let $(a^s, \boldsymbol{r^s})$ be its label in T'. Consider its i-th child v^t; by the induction hypothesis it has the same label $(a^t, \boldsymbol{q^t})$ in both trees. Claim 11 asserts that the fishbone between v_s and v_t in $g(T)$ is of length $n - q_i^s + \delta_{a^t}(\boldsymbol{q^t})$, and the length of the fishbone between v_s and v_t in $g(T')$ is $n - r_i^s + \delta_{a^t}(\boldsymbol{q^t})$. Since this is actually the same fishbone, this implies that $q_i^s = r_i^s$, and the labels of v_s in both trees are equal. This completes the induction step and proves that $T = T'$. □

Thus, the homomorphisms h and g are as desired. □

Theorem 11. *The class of tree languages recognized by tree-walking automata is not closed under injective homomorphisms.*

Proof. Suppose it is closed. It is claimed that then every regular tree language is recognized by a tree-walking automaton. Let L be a regular tree language over some tree signature S_{reg}. Then, by Theorem 10, there exist tree signatures S_{comp} and S_{mid}, and injective homomorphisms $g\colon L(S_{comp}) \to L(S_{mid})$ and $h\colon L(S_{reg}) \to L(S_{mid})$, such that $L = h^{-1}(g(L(S_{comp})))$. The language $L(S_{comp})$ is trivially recognized by a tree-walking automaton that accepts every tree right away. Then, by the assumption on the closure under g, the language $g(L(S_{comp}))$ is recognized by another tree-walking automaton. By Theorem 8, its inverse homomorphic image L is recognized by a tree-walking automaton as well. This contradicts the result by Bojańczyk and Colcombet [2] on the existence of regular tree languages not recognized by any tree-walking automata. □

11 Conclusion

The new bounds on the complexity of transforming graph-walking automata to automata with returning, halting and reversibility properties are fairly tight. However, for their important special cases, such as two-way finite automata (2DFA) and tree-walking automata (TWA), the gaps between lower bounds and upper bounds are still substantial.

For an n-state 2DFA, the upper bound for making it halting is $4n + \text{const}$ states [7]. No lower bound is known, and any lower bound would be interesting to obtain. A 2DFA can be made reversible using $4n + 3$ states [12], with a lower bound of $2n - 2$ states [11]; it would be interesting to improve these bounds.

The same question applies to tree-walking automata: they can be made halting [18], and, for k-ary trees, it is sufficient to use $4kn + 2k + 1$ states to obtain a reversible automaton [12]. No lower bounds are known, and this subject is suggested for further research.

The lower bound on the complexity of inverse homomorphisms is obtained using graphs with cycles. So it does not apply to the important case of tree-walking automata. On the other hand, in the even more restricted case of two-way finite automata, the state complexity of inverse homomorphisms is known to be $2n$ [8], which is in line of the kn bound in this thesis, as 2DFA have $k = 2$. It would be interesting to fill in the missing case of TWA.

References

[1] M. Bojańczyk, T. Colcombet, “Tree-walking automata cannot be determinized”, *Theoretical Computer Science*, 350:2–3 (2006), 164–173.

[2] M. Bojańczyk, T. Colcombet, “Tree-walking automata do not recognize all regular languages”, *SIAM Journal on Computing*, 38:2 (2008), 658–701.

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

[4] K. Čulík II, F. E. Fich, A. Salomaa, “A homomorphic characterization of regular languages”, *Discrete Applied Mathematics*, 4:2 (1982), 149–152.

[5] Y. Disser, J. Hackfeld, M. Klimm, “Tight bounds for undirected graph exploration with pebbles and multiple agents”, *Journal of the ACM*, 66:6 (2019), 40:1-40:41.

[6] 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.

[7] V. Geffert, C. Mereghetti, G. Pighizzini, “Complementing two-way finite automata”, *Information and Computation*, 205:8 (2007), 1173–1187.

[8] G. Jirásková, A. Okhotin, “On the state complexity of operations on two-way finite automata”, *Information and Computation*, 253:1 (2017), 36–63.

[9] C. A. Kapoutsis, “Two-way automata versus logarithmic space”, *Theory of Computing Systems*, 55:2 (2014), 421–447.

[10] A. Kondacs, J. Watrous, “On the power of quantum finite state automata”, *38th Annual Symposium on Foundations of Computer Science* (FOCS 1997, Miami Beach, Florida, USA, 19–22 October 1997), IEEE, 66–75.

[11] M. Kunc, A. Okhotin, “Reversible two-way finite automata over a unary alphabet”, TUCS Technical Report 1024, Turku Centre for Computer Science, December 2011.

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

[13] R. Landauer, “Irreversibility and heat generation in the computing process”, *IBM Journal of Research and Development*, 5:3 (1961), 183–191.

[14] K.-J. Lange, P. McKenzie, A. Tapp, “Reversible space equals deterministic space”, *Journal of Computer and System Sciences*, 60:2 (2000), 354–367.

[15] M. Latteux, J. Leguy, “On the composition of morphisms and inverse morphisms”, *ICALP 1983*, 420–432.

[16] 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.

[17] K. Morita, “A deterministic two-way multi-head finite automaton can be converted into a reversible one with the same number of heads”, *Reversible Computation* (RC 2012, Copenhagen, Denmark, 2–3 July 2012), LNCS 7581, 29–43.

[18] A. Muscholl, M. Samuelides, L. Segoufin, “Complementing deterministic tree-walking automata”, *Information Processing Letters*, 99:1 (2006), 33-39.

[19] M. Sipser, “Halting space-bounded computations”, *Theoretical Computer Science*, 10:3 (1980), 335–338.