Michal Kunc | Alexander Okhotin

Making graph-walking automata reversible

Turku Centre *for* Computer Science

TUCS Technical Report
No 1042, September 2012

Making graph-walking automata reversible

Michal Kunc
Department of Mathematics, Masaryk University,
Brno, Czech Republic.
E-mail: kunc@math.muni.cz
Alexander Okhotin
Department of Mathematics and Statistics, University of Turku, *and*
Turku Centre for Computer Science
Turku FI–20014, Finland. E-mail: alexander.okhotin@utu.fi

TUCS Technical Report
No 1042, September 2012

Abstract

The paper proposes a general notation for deterministic automata traversing finite undirected structures: the graph-walking automata. This abstract notion covers such models as two-way finite automata, including their multi-tape and multi-head variants, tree-walking automata and their extension with pebbles, picture-walking automata, space-bounded Turing machines, etc. It is then demonstrated that every graph-walking automaton can be transformed to an equivalent reversible graph-walking automaton, so that every step of its computation is logically reversible. This is done with a linear blow-up in the number of states, where the linear factor depends on the degree of graphs being traversed. The construction directly applies to all basic models covered by this abstract notion.

Keywords: Graph-walking automata, tree-walking automata, finite automata, reversible computation, halting.

TUCS Laboratory
FUNDIM, Fundamentals of Computing and Discrete Mathematics

1 Introduction

Logical reversibility of computations is an important property of computational devices in general, which can be regarded as a stronger form of determinism. Informally, a machine is reversible, if, given its configuration, one can always uniquely determine its configuration at the previous step. This property is particularly relevant to the physics of computation [5], as irreversible computations incur energy dissipation [32]. It is known from Lecerf [34] and Bennett [4] that every Turing machine can be simulated by a reversible Turing machine. Later, the time and space cost of reversibility was analyzed in the works of Bennett [6], Crescenzi and Papadimitriou [14], Lange *et al.* [33] and Buhrman *et al.* [12]. A line of research on reversibility in high-level programming languages was initiated by Abramsky [1]. Reversibility in cellular automata also has a long history of research, presented in surveys by Toffoli and Margolus [45] and by Kari [26]. In the domain of finite automata, the reversible subclass of one-way deterministic finite automata (1DFAs) defines a proper subfamily of regular languages [39], which was studied, in particular, by Héam [20] and by Lombardy [35]. On the other hand, every regular language is accepted by a reversible two-way finite automaton (2DFA): as shown by Kondacs and Watrous [28], every n-state 1DFA can be simulated by a $2n$-state reversible 2DFA.

One of the most evident consequences of reversibility is that a reversible automaton halts on every input. The property of halting on all inputs has received attention on its own. For time-bounded and space-bounded Turing machines, halting can be ensured by explicitly counting the number of steps, as done by Hopcroft and Ullman [22]. A different method for transforming a space-bounded Turing machine to an equivalent halting machine operating within the same space bounds was proposed by Sipser [42], and his approach essentially means constructing a reversible machine, though reversibility was not considered as such. In particular, Sipser [42] sketched a transformation of an n-state 2DFA to an $O(n^2)$-state halting 2DFA (which is actually reversible), and also mentioned the possibility of an improved transformation that yields $O(n)$ states, where the multiplicative factor depends upon the size of the alphabet. The fact that Sipser's idea produces reversible automata was noticed and used by Lange *et al.* [33] to establish the equivalence of deterministic space $s(n)$ to reversible space $s(n)$. Next, Kondacs and Watrous [28] distilled the construction of Lange *et al.* [33] into the mathematical essence of constructing reversible 2DFAs. A similar construction for making a 2DFA halt on any input was later devised by Geffert et al. [17], who have amalgamated an independently discovered method of Kondacs and Watrous [28] with a pre-processing step. For tree-walking automata (TWA), a variant of Sipser's [42] construction was used by Muscholl et al. [37] to transform an n-state automaton to an $O(n^2)$-state halting automaton.

The above results apply to various models that recognize input structures

by traversing them: such are the 2DFAs that walk over input strings, and the TWA walking over input trees. More generally, these results apply to such models as deterministic space-bounded Turing machines, which have extra memory at their disposal, but the amount of memory is bounded by a function of the size of the input. What do these models have in common? They are equipped with a fixed finite-state control, as well as with a finite space of memory configurations determined by the input data, and with a fixed finite set of operations on this memory. A particular machine of such a type is defined by a transition table, which instructs it to apply a memory operation and to change its internal state, depending on the current state and the currently observed data stored in the memory.

This paper proposes a general notation for such computational models: the *graph-walking automata* (GWA). In this setting, the space of memory configurations is regarded as an input graph, where each vertex is a memory configuration, labelled by the data observed by the machine in this position, and the operations on the memory become labels of the edges. Then a graph-walking automaton traverses a given input graph using a finite-state control, and a transition function with a finite domain, that is, at each step there are only finitely many possibilities that the automaton can observe. The definitions assume the following conditions on the original models, which accordingly translate to graph-walking automata; these assumptions are necessary to transform deterministic machines to reversible ones:

1. Every elementary operation on the memory has an opposite elementary operation that undoes its effect. For instance, in a 2DFA, the operation of moving the head to the left can be undone by moving the head to the right. In terms of graphs, this means that *input graphs are undirected*, and each edge has its end-points labelled by two opposite direction symbols, representing traversal of this edge in both directions.

2. The space of memory configurations on each given input object is finite, and it functionally depends on the input data. For graph-walking automata, this means that *input graphs are finite*. Though, in general, reversible computation is possible in devices with unbounded memory, the methods investigated in this paper depend upon this restriction.

3. The automaton can test whether the current memory configuration is the initial configuration. In a graph-walking automaton, this means that the initial node, where the computation begins, has a distinguished label.

A definition of graph-walking automata designed according to the above principles is given in Section 2, where it is illustrated by presenting 2DFAs and TWAs as simple special cases of GWAs. Further machine models, including multi-head 2DFAs, 2DFAs with pebbles and space-bounded Turing machines, are represented as GWAs in the subsequent Section 3.

The definition of reversibility in GWAs, developed in the next Section 4, is based upon several conditions. First, consider a subclass of GWAs, where each state is accessible from a unique direction, or, in other words, the last operation on the memory is remembered in the internal state; in this paper, such automata are called *direction-determinate*. Another subclass of GWAs are *returning automata*, which may accept only in the initial memory configuration. Then, a *reversible automaton* is defined as a direction-determinate returning GWA with an injective transition function on each label, and with at most one accepting state for every initial label.

The most distinctive property of reversible automata is that they can be straightforwardly *reversed*: that is, given a reversible GWA, one can construct a GWA with at most one extra state, which carries out the same computations in the reverse direction, and accepts the same set of graphs. This construction, which mostly amounts to replacing the transition function by each label with its inverse, is presented in Section 5.

The next Section 6 establishes the main results of this paper: these are transformations from automata of the general form to returning automata, and from returning to reversible automata. Both transformations rely on the same effective construction, which generalizes the method of Kondacs and Watrous [28], while the origins of the latter can be traced to the general idea due to Sipser [42]. The constructions involve only a linear blow-up in the number of states. In the subsequent Section 8, these results are applied to each of the concrete models represented as GWAs in this paper, leading, in particular, to the following results:

- An n-state 2DFA can be transformed to a reversible 2DFA with $4n+3$ states. This generalizes the construction of Kondacs and Watrous [28], and also provides an improved construction and a rigorous argument for the result by Geffert et al. [17] on making a 2DFA halt on every input. In the case of a one-letter alphabet, an n-state 2DFA is shown to have an equivalent reversible 2DFA with $2n + 5$ states.

- An n-state TWA over k-ary trees can be transformed to a $(4kn+2k+1)$-state reversible TWA. This improves the transformation to halting due to Muscholl et al. [37].

- An n-state k-head 2DFA can be transformed to a reversible k-head 2DFA with $2(3^k - 1)n + 2k + 1$ states. This establishes the result sought in the recent papers by Morita [36] and Axelsen [3].

- A deterministic Turing machine operating in marked space $s(n)$ (as defined by Ranjan, Chang and Hartmanis [41, Sect. 5]) can be transformed to a reversible Turing machine that uses the same marked space $s(n)$. This directly implies one of the versions of the result of Lange *et al.* [33] on the equivalence of deterministic space to reversible space.

Figure 1: An edge in a graph, with its ends marked by the directions d and $-d$.

A few extra results of this kind are included in the paper, and it is easy to infer more, for any automaton models representable as GWA.

Investigating further properties of graph-walking automata is proposed as a worthy subject for future research. Models of this kind date back to *automata in labyrinths*, introduced by Shannon and later studied by Budach [11]: these automata walk over directed graphs and may leave marks in nodes. There was a related work by Panaite and Pelc [38] on automatic mapping of a graph by an agent following its edges. Other important models defining families of graphs are graph-rewriting systems and monadic second-order logic on graphs researched by Courcelle [13], and graph tilings studied by Thomas [44].

2 Graph-walking automata

The automata studied in this paper walk over finite undirected graphs, in which every edge can be traversed in both directions. The directions are identified by labels attached to both ends of an edge. These labels belong to a finite set of *directions* D, with a bijective operation $-\colon D \to D$, representing *opposite directions* and satisfying $-(-d) = d$ for all $d \in D$.

If a graph models the memory, the directions represent elementary operations on this memory, and the existence of opposite directions means that every elementary operation on the memory can be reversed by applying its opposite.

A *signature* $\mathcal{S}$ consists of

- a direction set D, with a bijective operator $-\colon D \to D$, as described above,
- a finite set Σ of possible labels of nodes of the graph,
- a non-empty subset $\Sigma_0 \subseteq \Sigma$ of labels allowed in the initial node; all other nodes have to be labelled with elements of $\Sigma \setminus \Sigma_0$,
- a set $D_a \subseteq D$ of directions for every $a \in \Sigma$, so that every node labelled with a must be of degree $|D_a|$, with the incident edges corresponding to the elements of D_a.

A *graph* over the signature $\mathcal{S}$ 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 mapping, which must satisfy the following condition of invertibility by opposite directions: for every $v \in V$ and $d \in D$, if $v + d$ is defined, then $(v + d) + (-d)$ is defined too and $(v + d) + (-d) = v$. In other words, for every $d \in D$, the partial mapping $v \mapsto v + (-d)$ is the inverse of the partial mapping $v \mapsto v + d$. In the following, $v - d$ will be used to denote $v + (-d)$;
- the total mapping $\lambda \colon V \to \Sigma$ is a labelling of nodes, such that for every $v \in V$ the following two conditions are satisfied:
  1. $d \in D_{\lambda(v)}$ if and only if $v + d$ is defined,
  2. $\lambda(v) \in \Sigma_0$ if and only if $v = v_0$.

The representation of an edge by the mapping $+$ is illustrated in Figure 1.

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

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

Given a graph $(V, v_0, +, \lambda)$ over the signature $\mathcal{S}$, the automaton begins its computation in the state q_0, observing the node v_0. At each step of the computation, with the automaton in a state $q \in Q$ observing a node v, the automaton looks up the transition table δ for q and the label of v. If $\delta(q, \lambda(v))$ is defined as (q', d), the automaton enters the state q' and moves to the node $v + d$. If $\delta(q, \lambda(v))$ is undefined, then the automaton accepts if $(q, \lambda(v)) \in F$ and rejects otherwise.

Formally, for a graph $(V, v_0, +, \lambda)$ and an automaton $\mathcal{A} = (Q, q_0, \delta, F)$, every pair (q, v), with $q \in Q$ a state and $v \in V$ a node, is called a *configuration* of $\mathcal{A}$ over this graph. The *computation* of $\mathcal{A}$ on this graph beginning in the configuration (q, v) is the unique (finite or infinite) sequence $(p_0, u_0), (p_1, u_1), \ldots$ of configurations, which satisfies the following conditions:

- $(p_0, u_0) = (q, v)$.
- For every element (p_i, u_i) of the sequence, $\delta(p_i, \lambda(u_i))$ is undefined if and only if it is the last element of the sequence.

- If (p_i, u_i) is not the last element of the sequence, then $\delta(p_i, \lambda(u_i)) = (p_{i+1}, d)$, where $u_{i+1} = u_i + d$.

If the computation is infinite, then, due to the finiteness of the graph, it goes into a loop, which is repeated forever. If the computation is finite, then it is either accepting (if the last (p_i, u_i) satisfies $(p_i, \lambda(u_i)) \in F$) or rejecting by undefined transition.

A graph $(V, v_0, +, \lambda)$ is *accepted* by $\mathcal{A}$, if the computation of $\mathcal{A}$ on $(V, v_0, +, \lambda)$ starting in the initial configuration (q_0, v_0) terminates in a configuration (q, v) such that $(q, \lambda(v)) \in F$.

The two most well-known special cases of graph-walking automata are the *two-way finite automata*, which walk over path graphs, and *tree-walking* automata operating on trees.

Example 1. *A two-way deterministic finite automaton (2DFA) operating on a tape delimited by a left-end marker $\vdash$ and a right-end marker $\dashv$, with the tape alphabet Γ, is a graph-walking automaton operating on graphs over the signature $\mathcal{S}$ with $D = \{+1, -1\}$, $\Sigma = \Gamma \cup \{\vdash, \dashv\}$, $\Sigma_0 = \{\vdash\}$, $D_\vdash = \{+1\}$, $D_\dashv = \{-1\}$ and $D_a = \{+1, -1\}$ for all $a \in \Gamma$. A sample graph over this signature is demonstrated in Figure 2(i).*

All connected graphs over this signature are path graphs, containing one instance of each marker and an arbitrary number of symbols from Γ in between. For an input string $w = a_1 \dots a_n$, with $n \geqslant 0$, the graph has the set of nodes $V = \{0, 1, \dots, n, n+1\}$ representing positions on the tape, with $v_0 = 0$ and with $v + d$ defined as the sum of integers. These nodes are labelled as follows: $\lambda(0) = \vdash$, $\lambda(n+1) = \dashv$ and $\lambda(i) = a_i$ for all $i \in \{1, \dots, n\}$.

Consider *tree-walking automata*, defined by Aho and Ullman [2, Sect. VI] as a formal model for traversing context-free parse trees, and later studied by Bojańczyk and Colcombet [8, 9]. Given an input binary tree, a tree-walking automaton moves over it, scanning one node at a time. At each step of its computation, it may either go down to any of the sons of the current node or up to its father. Furthermore, in any node except the root, the automaton is invested with the knowledge of whether this node is the first son or the second son [8]; without this knowledge, the expressive power of a tree-walking automaton would be severely limited [24]. The traversal of trees by these automata can be described using directions of the form "go down to the i-th son" and the opposite "go up from the i-th son to its father".

In the notation of graph-walking automata, the knowledge of the number of the current node among its siblings is contained in its label: for each label a, the set of valid directions D_a contains exactly one upward direction and the full set of downward directions. Furthermore, by analogy with the end-markers in 2DFAs, the input trees of tree-walking automata shall have end-markers attached to the root and to all leaves; in both cases, these markers allow a better readable definition[1].

[1]Note that both for 2DFA and TWA it would be possible not to use any end-markers,

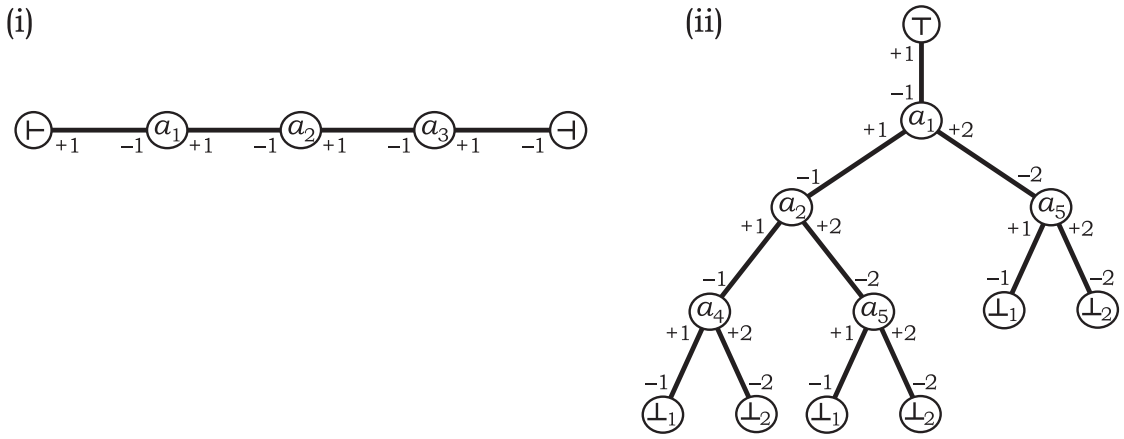

Figure 2: Representing (i) input strings of 2DFAs and (ii) input trees of TWAs as graphs.

Example 2. *A tree-walking automaton on k-ary trees uses the set of directions $D = \{+1, +2, \ldots, +k, -1, -2, \ldots, -k\}$, with $-(+i) = -i$, where positive directions go to children and negative ones to fathers. Trees are graphs labelled with symbols in $\Sigma = \{\top, \bot_1, \ldots, \bot_k\} \cup \Gamma$, where the top marker $\top$ with $D_\top = \{+1\}$ is the label of the root v_0 (and accordingly, $\Sigma_0 = \{\top\}$), while each i-th bottom marker $\bot_i$ with $D_{\bot_i} = \{-i\}$ is a label for leaves. Elements of the set Γ are used to label internal nodes of the tree, so that for each $a \in \Gamma$ there exists $i \in \{1, \ldots, k\}$ with $D_a = \{-i, +1, \ldots, +k\}$, which means that every node labelled by a is the ith child of its father. An example of a tree with $k = 2$ is given in Figure 2(ii).*

One can similarly define the *unranked tree automata*, in which every internal node may have any finite number of successors. Such an automaton has directions to go down to the first son, and to go down to the last son, along with the opposite directions for going up from the first son and going up from the last son; furthermore, an unranked tree automaton has directions to go left to the previous sibling and to go right to the next sibling.

3 Common models of computation as graph-walking automata

Consider any computational device recognizing input objects of any kind, which has a fixed number of internal states and employs auxiliary memory holding such data as the positions of reading heads and the contents of any additional data structures. Assume that for each fixed input, the total space

and instead have something like $\Sigma = \{\vdash, \text{middle}, \dashv\} \times \Gamma$ in the case of 2DFAs and $\Sigma = \{\top, \text{middle}, \bot\} \times \{1, \ldots, k\} \times \Gamma$ for TWAs.

of possible memory configurations of the device and the structure of admissible transitions between these configurations are known in advance. The set of memory configurations with the structure of transitions forms a *graph of memory configurations*, which can be presented in the notation assumed in this paper by taking the set of *elementary operations on the memory* as the set of directions. The label attached to the currently observed node represents the information on the memory configuration available to the original device, such as the contents of cells observed by heads; along with its internal state, this is all the data it can use to determine its next move. Thus the device is represented as a graph-walking automaton.

To begin with the simplest model, consider a 2DFA. Its space of memory configurations includes all positions of the head over the tape, and these positions are connected into a path graph, as in Figure 2(i). Elementary operations on the memory include incrementing and decrementing the position of the head by $+1 \in D$ and $-1 \in D$, respectively, and the label in each position contains the symbol observed by the head.

By the same principle, one can describe many other models of computation by graph-walking automata. As the first model, consider 2DFAs equipped with multiple reading heads, which can independently move over the same input tape: *the multi-head automata*, introduced by Rabin and Scott [40]. The research on such automata is described in a survey by Holzer et al. [21]. Their reversibility was recently investigated by Morita [36] and by Axelsen [3].

Example 3. *A k-head two-way deterministic finite automaton with a tape alphabet Γ is described by a graph-walking automaton as follows. Its memory configuration contains the positions of all k heads on the tape. The set of directions is $D = \{-1, 0, +1\}^k \setminus \{0\}^k$, where a direction $(s_1, \ldots, s_k)$ with $s_i \in \{-1, 0, +1\}$ indicates that each i-th head is to be moved in the direction s_i. Each label in $\Sigma = \big(\Gamma \cup \{\vdash, \dashv\}\big)^k$ contains all the data observed by the automaton in a given memory configuration: this is a k-tuple of symbols scanned by all heads. There is a unique initial label corresponding to all heads parked at the left-end marker, that is, $\Sigma_0 = \{(\vdash, \ldots, \vdash)\}$. For each node label $(s_1, \ldots, s_k) \in \Sigma$, the set of directions $D_{(s_1,\ldots,s_k)}$ contains all k-tuples $(d_1, \ldots, d_k) \in \{-1, 0, +1\}^k$, where $d_i \neq -1$ if $s_i = \vdash$ and $d_i \neq +1$ if $s_i = \dashv$; the latter conditions disallow moving any heads beyond either end-marker.*

The automaton operates on graphs of the following form. For each input string $a_1 \ldots a_n \in \Gamma^*$, let $a_0 = \vdash$ and $a_{n+1} = \dashv$ for uniformity. Then the set of nodes of the graph is a discrete k-dimensional cube $V = \{0, 1, \ldots, n, n+1\}^k$, with each node $(i_1, \ldots, i_k) \in V$ labelled with $(a_{i_1}, \ldots, a_{i_k}) \in \Sigma$. The initial node is $v_0 = (0, \ldots, 0)$, labelled with $(\vdash, \ldots, \vdash)$. An example of such a graph is given in Figure 3.

The set of graphs representing computations of k-head 2DFAs, as described in Example 3, is not the whole set of connected graphs over the given

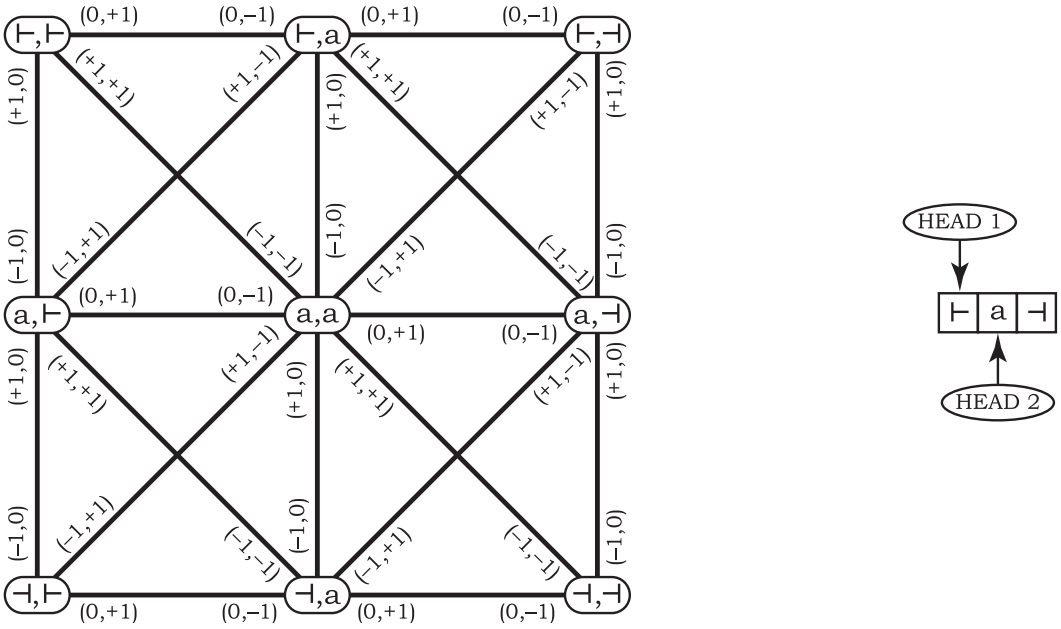

Figure 3: Representing the space of head positions of a 2-head 2DFA on the input $w = a$ as a graph.

signature. For instance, one can reconnect the edges in the graph in Figure 3 to obtain another graph over this signature, which no longer corresponds to the space of configurations of k-head 2DFAs on any input. However, on the subset of graphs of the intended form, a GWA defined in Example 3 correctly represents the behaviour of a k-head 2DFA.

Several other models of computation can be described by GWAs in a similar way. Consider *multi-tape finite automata*, introduced in the famous paper by Rabin and Scott [40] and studied, in particular, by Harju and Karhumäki [19].

Example 4. *A k-tape two-way deterministic finite automaton with tape alphabets $\Gamma_1, \ldots, \Gamma_k$ uses directions $D = \{-1, 0, +1\}^k \setminus \{0\}^k$ and labels $\Sigma = \big(\Gamma_1 \cup \{\vdash, \dashv\}\big) \times \ldots \times \big(\Gamma_k \cup \{\vdash, \dashv\}\big)$. The set of directions $D_{(s_1,\ldots,s_k)}$ available in a node labelled by $(s_1, \ldots, s_k) \in \Sigma$ is defined as in Example 3.*

Given a k-tuple of input strings $(w_1, \ldots, w_k) \in \Gamma_1^* \times \ldots \times \Gamma_k^*$ with $w_i = a_{i,1} \ldots a_{i,\ell_i}$, let $a_{i,0} = \vdash$ and $a_{i,\ell_i+1} = \dashv$; then the automaton operates on the graph with the set of nodes $V = \{0, 1, \ldots, \ell_1, \ell_1{+}1\} \times \ldots \times \{0, 1, \ldots, \ell_k, \ell_k{+}1\}$, where each node $(i_1, \ldots, i_k) \in V$ is labelled with $(a_{1,i_1}, \ldots, a_{k,i_k}) \in \Sigma$. The initial node $v_0 = (0, \ldots, 0)$ has the label $\lambda(v_0) = (\vdash, \ldots, \vdash)$.

Two-way finite automata with pebbles, introduced by Blum and Hewitt [7] and further studied by Globerman and Harel [18], are 2DFAs equipped with a fixed number of pebbles, which may be dispensed at or collected from the currently visited square; it is known that 2DFAs with a single pebble

recognize only regular languages, while two pebbles allow representing some non-regular languages.

Example 5. *A 2DFA with k pebbles with a tape alphabet Γ uses the set of directions $D = \{-1, +1\} \cup \{\downarrow 1, \uparrow 1, \ldots, \downarrow k, \uparrow k\}$, where -1 and $+1$ refer to moving the head, $\downarrow t$ means putting the t-th pebble at the current position, and $\uparrow t$ means picking up the t-th pebble from the current square. Accordingly, the opposite directions are defined as $-(+1) = -1$ and $-(\downarrow t) = \uparrow t$. Every label from the set $\Sigma = (\Gamma \cup \{\vdash, \dashv\}) \times \{here, elsewhere, nowhere\}^k$ determines the symbol at the current square and the status of each pebble, whether it is placed at the current position, is placed elsewhere, or is available to be placed. For each node label $(s, p_1, \ldots, p_k) \in \Sigma$, the set of directions $D_{(s,p_1,\ldots,p_k)}$ contains -1 (unless $s = \vdash$), $+1$ (unless $s = \dashv$), all $\downarrow t$ with $p_t = nowhere$, and all $\uparrow t$ with $p_t = here$.*

For an input string $a_1 \ldots a_\ell \in \Gamma^*$, let $P = \{0, 1, \ldots, \ell, \ell + 1\}$ be the set of positions in the input. As before, let $a_0 = \vdash$ and $a_{\ell+1} = \dashv$. The automaton operates on a graph with the set of nodes $V = P \times (P \cup \{\text{␣}\})^k$, where the first component defines the position of the head, and the remaining k components determine the position of each pebble, with the space (␣) indicating that a pebble has not been placed. Each node $(i, j_1, \ldots, j_k) \in V$ is labelled with $(a_i, p_1, \ldots, p_k) \in \Sigma$, where

$$p_t = \begin{cases} nowhere, & \text{if } j_t = \text{␣}, \\ here, & \text{if } j_t = i, \\ elsewhere, & \text{otherwise.} \end{cases}$$

The initial node is $(0, \text{␣}, \ldots, \text{␣})$. The sum $v + d$ is defined as follows:

$$\begin{aligned} (i, j_1, \ldots, j_k) + d &= (i + d, j_1, \ldots, j_k), \qquad \text{for } d \in \{-1, +1\}, \\ (i, j_1, \ldots, j_{t-1}, \text{␣}, j_{t+1}, \ldots, j_k) + \downarrow t &= (i, j_1, \ldots, j_{t-1}, i, j_{t+1}, \ldots, j_k), \\ (i, j_1, \ldots, j_{t-1}, i, j_{t+1}, \ldots, j_k) + \uparrow t &= (i, j_1, \ldots, j_{t-1}, \text{␣}, j_{t+1}, \ldots, j_k). \end{aligned}$$

The model defined in Example 5 can be extended to *tree-walking automata with pebbles*, first considered by Engelfriet and Hoogeboom [15] and subsequently studied by Bojańczyk et al. [10] and by Muscholl et al. [37]. This model can be restricted to such variants as tree-walking automata with nested pebbles [15, 16]. All these models can be further extended to have multiple reading heads, etc., and each case can be described by an appropriate kind of GWAs operating over graphs that encode the space of memory configurations of the desired automata.

The next model to be defined as graph-walking automata are the space-bounded Turing machines; to be precise, Turing machines operating in *marked space*, as in the paper by Ranjan, Chang and Hartmanis [41, Sect. 5], rather than the more usual unmarked space with the condition of space constructibility.

Example 6. *Consider two-tape Turing machines with an alphabet Γ on the read-only input tape, and with an alphabet Ω on the work tape, where the number of squares on the work tape is defined by a function $s\colon \mathbb{N} \to \mathbb{N}$ of the length of the input string. The set of memory configurations of such machines is represented by graphs with the nodes corresponding to different positions of heads and different contents of the work tape. These graphs are defined with the set of directions $D = \{(-1,0),(+1,0),(0,-1),(0,+1)\} \cup \{\, cc' \mid c,c' \in \Omega,\ c \neq c' \,\}$, where directions (i,j) with $i,j \in \{-1,0,+1\}$ refer to moving the heads on the input tape and on the work tape, respectively, while a direction cc' means replacing the currently read symbol c on the work tape with the symbol c'. The opposite directions are defined as $-(+1,0) = (-1,0)$, $-(0,+1) = (0,-1)$ and $-(cc') = c'c$. The label of each node contains the currently scanned symbols on the input tape and on the work tape, that is, $\Sigma = (\Gamma \cup \{\vdash,\dashv\}) \times (\Omega \cup \{\vdash,\dashv\})$. For each node label $(a,c) \in \Sigma$, the set of directions $D_{(a,c)}$ contains $(-1,0)$ (unless $a = \vdash$), $(+1,0)$ (unless $a = \dashv$), $(0,-1)$ (unless $c = \vdash$), $(0,+1)$ (unless $c = \dashv$), and all cc' (for every $c' \in \Omega \setminus \{c\}$, unless $c \in \{\vdash,\dashv\}$).*

For an input string $a_1 \dots a_n \in \Gamma^*$, the second tape initially contains the string $\vdash \llcorner^{s(n)} \dashv$, where $\llcorner \in \Omega$ is a special blank symbol. The space of memory configurations is represented by a graph with the set of nodes

$$V = \{0,\dots,n+1\} \times \{0,\dots,s(n)+1\} \times \Gamma^{s(n)},$$

with the positions of the heads as the first two components, and with the contents of the work tape in the third component. The initial node is $v_0 = (0,0,\vdash\llcorner^{s(n)}\dashv)$. Assuming $a_0 = c_0 = \vdash$ and $a_{n+1} = c_{s(n)+1} = \dashv$, the label of each node is defined as $\lambda(i,j,c_1 \dots c_{s(n)}) = (a_i, c_j)$.

Consider one more variant of finite automata, which operates on two-dimensional pictures. Such machines—the so-called *four-way finite automata*, 4DFA—were defined by Blum and Hewitt [7] and are described in a recent survey by Kari and Salo [27].

Example 7. *A 4DFA has $D = \{(-1,0),(+1,0),(0,-1),(0,+1)\}$. It operates on rectangular $m \times n$ arrays with $a_{i,j} \in \Gamma$ for $i,j \in \{1,\dots,m\} \times \{1,\dots,n\}$. Such an array is bordered by special symbols of four different types, representing four directions ($\top$, $\dashv$, $\bot$, $\vdash$). The topmost left special symbol has a special label $\Vdash$.*

Let $\Sigma = \Gamma \cup \{\top,\dashv,\bot,\vdash,\Vdash\}$, let $D_a = D$ for all $a \in \Gamma$ and let $D_\top = \{(+1,0)\}$, $D_\dashv = \{(0,-1)\}$, $D_\bot = \{(-1,0)\}$, $D_\vdash = D_\Vdash = \{(0,+1)\}$. For each input picture, consider the graph with $V = \{\, (i,j) \mid 1 \leqslant i \leqslant m,\ 1 \leqslant j \leqslant n \,\} \cup \{\, (i,j) \mid 1 \leqslant i \leqslant m,\ j \in \{0,n+1\} \,\} \cup \{\, (i,j) \mid i \in \{0,m+1\}, 1 \leqslant j \leqslant n \,\}$, where $v_0 = (1,0)$. The sum $v + d$ is defined as the sum of vectors. For all $i \in \{1,\dots,m\}$ and $j \in \{1,\dots,n\}$, let $\lambda((i,j)) = a_{i,j}$, $\lambda((0,j)) = \top$, $\lambda((i,n+1)) = \dashv$, $\lambda((m+1,j)) = \bot$, $\lambda((1,0)) = \Vdash$ and $\lambda((i,0)) = \vdash$ for $i \neq 1$. Such a graph for $m = 2$ and $n = 3$ is illustrated in Figure 4.

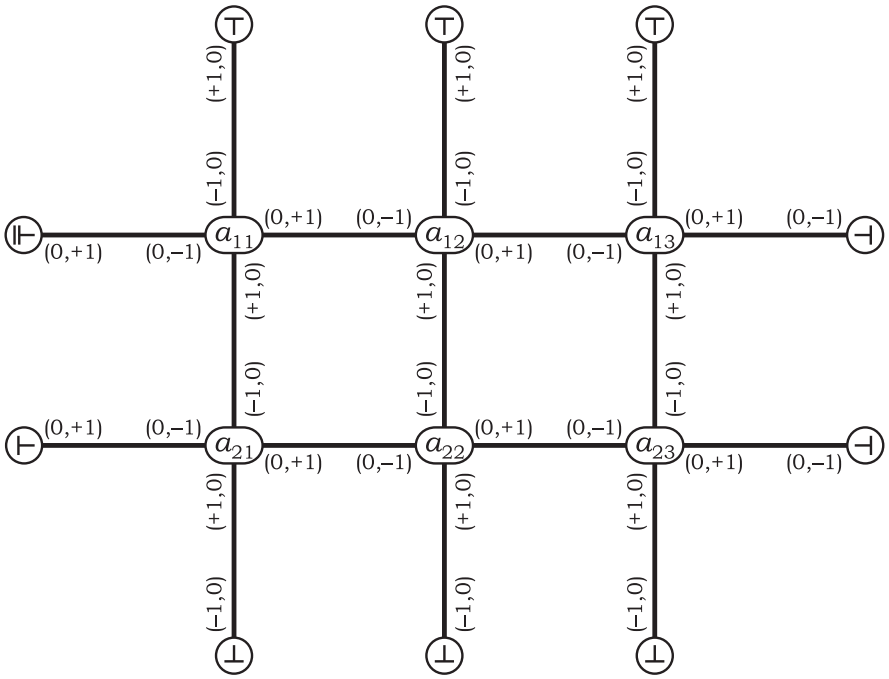

Figure 4: A 4DFA as a graph-walking automaton.

Because the setting of graph-walking automata on undirected graphs, as they are defined in this paper, was designed with reversibility in mind, many models cannot be described within this setting. Consider any models, which cannot immediately return to the previous configuration after any operation. This includes any models, where the previous configuration may never be revisited, such as one-way finite automata [39] or pushdown automata [31]. Models that may possibly revisit earlier configurations, but cannot do that at will by a single instruction, cause the same problem: for instance, such are the *sweeping two-way finite automata* introduced by Sipser [43] and the *rotating automata* of Kapoutsis et al. [25], where the head may revisit the previous configuration only after traversing the whole input. The same can be said about the *restarting automata* of Jančar et al. [23], in which the head may jump to the beginning of the input by a single instruction, thus resetting the memory configuration. No such models can be expressed by GWA on undirected graphs, because an operation on the memory without an opposite operation means that some edges cannot be traversed in both directions.

Another type of models not considered in this paper are any devices with unbounded memory, where, given an input, one does not know the space of memory configurations in advance. Such are the Turing machines of the general form, and they may be regarded as GWA operating on infinite graphs. The results of this paper essentially rely on the finiteness of input graphs.

4 Reversibility and related notions

This section develops the definition of logical reversibility for graph-walking automata. This definition is comprised of several conditions, and the first condition is that each state may be accessed from a unique direction.

Definition 1. *A graph-walking automaton is called* direction-determinate*, if every state is reachable from a unique direction, that is, there exists a partial function $d\colon Q \to D$, such that $\delta(q, a) = (q', d')$ implies $d' = d(q')$.*

As the direction is always known, the notation for the transition function can be simplified as follows: for each $a \in \Sigma$, let $\delta_a\colon Q \to Q$ be a partial function defined by $\delta_a(p) = q$ if $\delta(p, a) = (q, d(q))$.

According to this definition, $d(q)$ may be undefined only if q is not reachable by any transitions, which may be useful only if $q = q_0$. Note that the requirement $\delta(q, a) \in Q \times D_a$ on the transition function in the definition of automata translates to this notation as $d(\delta_a(p)) \in D_a$, whenever $\delta_a(p)$ is defined.

Any GWA can be made direction-determinate by storing the last used direction in its state.

Lemma 1. *For every graph-walking automaton with the set of states Q, there exists a direction-determinate automaton with the set of states $Q \times D$, which recognizes the same set of graphs.*

Another subclass of automata requires returning to the initial node after acceptance.

Definition 2. *A graph-walking automaton is called* returning*, if it has $F \subseteq Q \times \Sigma_0$, that is, if it accepts only at the initial node.*

A returning graph-walking automaton is still allowed to reject in any way, whether by an undefined transition at any node, or by looping.

For each computational model represented by a GWA in Sections 2–3, returning after acceptance is straightforward: for instance, a 2DFA moves its head to the left, a 2DFA with pebbles picks up all its pebbles, a space-bounded Turing machine erases its work tape, etc. However, for graphs of the general form, finding a way back to the initial node from the place where the acceptance decision was reached is not a trivial task. This paper defines a transformation to a returning automaton, which finds the initial node by backtracking the accepting computation.

Theorem 1. *For every direction-determinate graph-walking automaton with n states, there exists a direction-determinate returning graph-walking automaton with $3n$ states recognizing the same set of graphs.*

Another important kind of graph-walking automata are those that halt on any input.

Definition 3. *A graph-walking automaton is* halting*, if it never goes into a loop, that is, if its computation on every input graph is either accepting or rejecting by an undefined transition.*

For every direction-determinate graph-walking automaton, consider the inverses of transition functions by all labels, $\delta_a^{-1}\colon Q \to 2^Q$ for $a \in \Sigma$, defined by $\delta_a^{-1}(q) = \{\, p \mid \delta_a(p) = q \,\}$. Given a configuration of a direction-determinate automaton, one can always determine the direction, from which the automaton came to the current node v at the previous step; and if the inverse function $\delta^{-1}_{\lambda(v)}$ is furthermore injective, then the state at the previous step is also known, and hence the configuration at the previous step is uniquely determined. This leads to the following definition of automata, whose computations can be uniquely reconstructed from their final configurations:

Definition 4. *A direction-determinate graph-walking automaton is called* reversible*, if*

i. every partial transformation δ_a is injective, that is, $|\delta_a^{-1}(q)| \leqslant 1$ for all $a \in \Sigma$ and $q \in Q$, and

ii. the automaton is returning, and for each $a_0 \in \Sigma_0$, there exists at most one state q such that $(q, a_0) \in F$ (this state is denoted by $q^{a_0}_{\mathrm{acc}}$). These states may be equal for different a_0.

The second condition ensures that if an input graph $(V, v_0, +, \lambda)$ is accepted, then it is accepted in the configuration $(q_{\mathrm{acc}}^{\lambda(a_0)}, v_0)$. Therefore, this assumed accepting computation can be traced back, beginning from its final configuration, until either the initial configuration (q_0, v_0) is reached (which means that the automaton accepts), or a configuration without predecessors is encountered (then the automaton does not accept this graph). This reverse computation can be carried out by another reversible GWA, which will be constructed in the next Section 5.

Lemma 2. *On each finite input graph $(V, v_0, +, \lambda)$, a reversible graph-walking automaton beginning in an arbitrary configuration $(\widehat{q}, \widehat{v})$ either halts after finitely many steps, or returns to the configuration $(\widehat{q}, \widehat{v})$ and loops indefinitely.*

Proof. Suppose, for the sake of contradiction, that the computation of a reversible graph-walking automaton beginning in $(\widehat{q}, \widehat{v})$ goes into an infinite loop, and this loop does not pass through the configuration $(\widehat{q}, \widehat{v})$. Let $(\widetilde{q}, \widetilde{v})$ be the first repeated configuration. Since it is not equal to the configuration $(\widehat{q}, \widehat{v})$, it had predecessors both the first and the second time it was visited. Let (q, v) and (q', v') be these two predecessor configurations;

they must be distinct, because otherwise this would be a repeated configuration encountered earlier than $(\widetilde{q}, \widetilde{v})$. Because the automaton is direction-determinate, $\widetilde{v} = v + d(\widetilde{q}) = v' + d(\widetilde{q})$, and hence $v = v' = \widetilde{v} - d(\widetilde{q})$. Then $\delta_{\lambda(v)}(q) = \delta_{\lambda(v)}(q') = \widetilde{q}$, while $q \neq q'$, which contradicts the reversibility of the automaton. □

The second case in Lemma 2 allows a reversible automaton to be non-halting, if its initial configuration can be re-entered. This possibility is ruled out by the following extra condition on the automaton.

Definition 5. *A direction-determinate automaton $\mathcal{A} = (Q, q_0, \delta, F)$ is said to* have separated initial state*, if*

- *$\delta_a(q_0)$ is undefined for all $a \in \Sigma \setminus \Sigma_0$,*
- *the initial state of $\mathcal{A}$ is not re-enterable, that is, $\delta_a^{-1}(q_0) = \varnothing$ for every $a \in \Sigma$ (in particular, $d(q_0)$ is undefined).*

Lemma 3. *A reversible graph-walking automaton with a separated initial state, starting in the initial configuration, halts on every input graph.*

Lemma 3 immediately follows from Lemma 2, as Definition 5 rules out one of its cases.

The above definition of reversible automata has one more imperfection: while such automata may accept only in a single designated configuration, there are no limitations on where they may reject. Thus, backtracking a rejecting computation is not possible, because it is not known where it ends. The below strengthened definition additionally requires rejection to take place in a unique configuration, analogous to the accepting configuration.

Definition 6. *A* strongly reversible *automaton is a reversible automaton $\mathcal{A} = (Q, q_0, \delta, F)$ with separated initial state, which additionally satisfies the following conditions:*

iii. for every non-initial label $a \in \Sigma \setminus \Sigma_0$, the partial function δ_a is a bijection from $\{\, p \in Q \mid -d(p) \in D_a \,\}$ to $\{\, q \in Q \mid d(q) \in D_a \,\}$,

iv. for each initial label $a_0 \in \Sigma_0$, there is at most one designated rejecting state $q_{\mathrm{rej}}^{a_0} \in Q$ (these states may be equal for different a_0), for which neither $\delta_{a_0}(q_{\mathrm{rej}}^{a_0})$ is defined, nor $(q_{\mathrm{rej}}^{a_0}, a_0)$ is in F,

v. for all $a_0 \in \Sigma_0$ and for all states $q \in Q \setminus \{q_{\mathrm{acc}}^{a_0},\ q_{\mathrm{rej}}^{a_0}\}$, $\delta_{a_0}(q)$ is defined if and only if $-d(q) \in D_{a_0}$ or $q = q_0$.

The requirement on the range of δ_a, with $a \in \Sigma \setminus \Sigma_0$, in condition (iii) means that if a-labelled nodes have a direction $d \in D_a$ for reaching a state $q \in Q$, then there is a state $p \in Q$, in which this direction d can be used to

get to q. The requirement on the domain of δ_a means that this function is defined precisely for those states p, which can be possibly entered in a-labelled nodes, that is, for such states p, that the direction for entering p leads to these nodes. This in particular implies that whenever a computation of a strongly reversible automaton enters any configuration (p, v) with $\lambda(v) \notin \Sigma_0$ (that is, $v \neq v_0$), the next step of the computation is defined and the automaton cannot halt in this configuration. Similarly, condition (v) ensures that the computation cannot halt in the initial node (labelled with a_0), unless it reaches either the corresponding accepting state $q_{\mathrm{acc}}^{a_0}$ or the corresponding rejecting state $q_{\mathrm{rej}}^{a_0}$. Since Lemma 3 guarantees that the computation of a strongly reversible automaton beginning in the initial configuration always halts, one can conclude that it always halts with its head scanning the initial node, and either in the accepting state or in the rejecting state.

Lemma 4. *For every finite input graph $(V, v_0, +, \lambda)$, a strongly reversible graph-walking automaton, starting in the initial configuration, either accepts in the configuration $(q_{\mathrm{acc}}^{\lambda(v_0)}, v_0)$ or rejects in the configuration $(q_{\mathrm{rej}}^{\lambda(v_0)}, v_0)$.*

The transformation of a deterministic automaton to a reversible one developed in this paper ensures this strongest form of reversibility.

Theorem 2. *For every direction-determinate returning graph-walking automaton with n states, there exists a strongly reversible graph-walking automaton with $2n + 1$ states recognizing the same set of graphs.*

Theorems 1–2, along with Lemma 1, together imply the following transformation:

Corollary 1. *For every graph-walking automaton with n states and d directions, there exists a strongly reversible graph-walking automaton with $6dn+1$ states recognizing the same set of graphs.*

A given n-state GWA is first transformed to a direction-determinate GWA with dn states according to Lemma 1. The latter GWA has an equivalent returning direction-determinate GWA with $3dn$ states by Theorem 1. Finally, applying Theorem 2 to this automaton yields a strongly reversible GWA with $6dn + 1$ states.

Corollary 2. *For every signature, the class of sets of graphs over that signature recognized by graph-walking automata is effectively closed under all Boolean operations.*

More precisely, for a signature with d directions, the complement of an n-state GWA can be represented using $6dn+1$ states, by making the automaton strongly reversible and then inverting the acceptance decisions. The union of an m-state and an n-state GWAs is representable using $6dm + n + 1$ states, by making the first automaton strongly reversible, accepting if it accepts,

and switching to the second automaton if the first automaton rejects. The intersection of such automata can be represented using $3dm + n$ states, by making the first automaton direction-determinate and returning according to Lemma 1 and Theorem 1; if this automaton accepts, one can proceed with executing the second automaton.

5 Reversing a reversible automaton

The overall idea of reversibility is that a computation can be reconstructed from its final configuration, by retracting the steps of the automaton. This section shows how to construct a GWA $\mathcal{A}^{\mathrm{r}}$ that simulates a given reversible GWA $\mathcal{A}$ backwards.

Given an input graph, the automaton $\mathcal{A}^{\mathrm{r}}$ shall follow the unique computation of $\mathcal{A}$ leading to its accepting configuration, reconstructing its steps in the reverse order. If this computation leads $\mathcal{A}^{\mathrm{r}}$ to the initial configuration of $\mathcal{A}$, then $\mathcal{A}^{\mathrm{r}}$ may accept. Otherwise, if $\mathcal{A}$ rejects this graph, then the automaton $\mathcal{A}^{\mathrm{r}}$ will eventually find a configuration of $\mathcal{A}$ without predecessors, where it can reject.

The details of the construction are much simplified by assuming the following property.

Definition 7. *A direction-determinate automaton is said to be* without inaccessible transitions*, if whenever a state $q \in Q$ has a transition defined on a symbol $a \in \Sigma$, or is accepting there, this state must be reachable from the appropriate direction, unless the pair (q, a) corresponds to an initial configuration. In other words, for all $(q, a) \in (Q \times \Sigma) \setminus (\{q_0\} \times \Sigma_0)$, if $\delta_a(q)$ is defined or $(q, a) \in F$, then $-d(q) \in D_a$.*

If a given automaton does not satisfy this condition, then its inaccessible transitions can be undefined without affecting the set of graphs accepted by this automaton.

Let $\mathcal{A} = (Q, q_0, \delta, F)$ be a reversible automaton without inaccessible transitions. Define a new direction-determinate automaton $\mathcal{A}^{\mathrm{r}} = (Q', q_0', \delta', F')$ with separated initial state, using the set of states

$$Q' = [Q] \cup \{q_0'\},$$

where $[Q] = \{\, [q] \mid q \in Q \,\}$ is a copy of Q and q_0' is a new initial state. These states are accessed from the directions $d'([q]) = -d(q)$ for all $q \in Q$. Whenever the new automaton is in a state $[q]$, with the head observing a node v, this corresponds to the original automaton's being in the state q, observing the neighbouring node $v + d(q)$. Note that by looking at v, the automaton $\mathcal{A}^{\mathrm{r}}$ knows the label of the node from which $\mathcal{A}$ came to $v + d(q)$, and according to this label it decides from which state $\mathcal{A}$ came to the configuration $(q, v+d(q))$. Thus the reverse computation carried out by $\mathcal{A}^{\mathrm{r}}$ follows the state and the

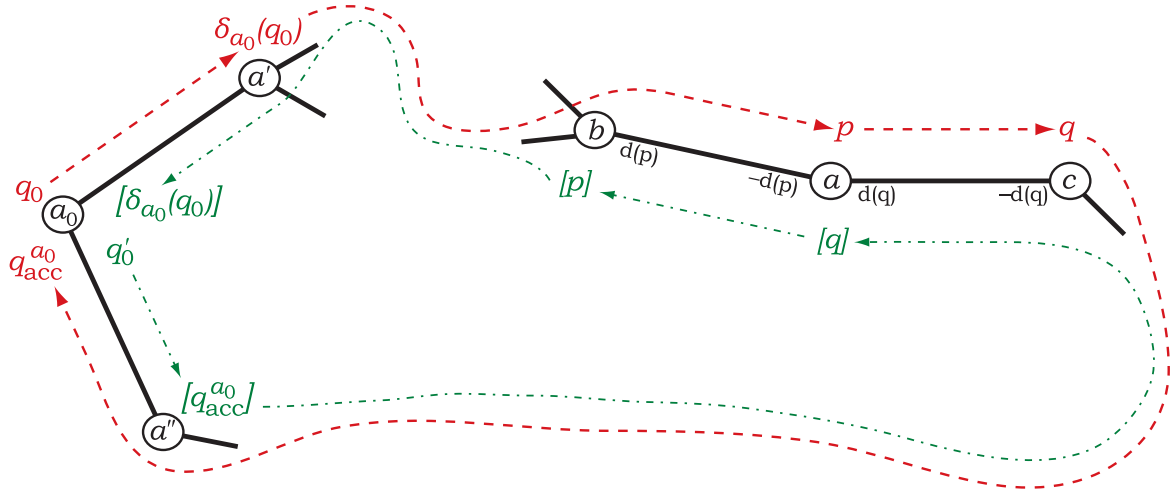

Figure 5: Simulating a computation of a reversible GWA $\mathcal{A}$ (from q_0 to $q_{\mathrm{acc}}^{a_0}$) by a reverse computation of the GWA $\mathcal{A}^{\mathrm{r}}$ (from q_0' to $[\delta_{a_0}(q_0)]$).

position of the head of a forward computation of $\mathcal{A}$, but the state and the position are always out of synchronization by one step.

The computation of the reversed automaton begins by the following transitions from q_0', for every initial label $a_0 \in \Sigma_0$:

$$\delta'_{a_0}(q_0') = \begin{cases} [q_{\mathrm{acc}}^{a_0}], & \text{if } q_{\mathrm{acc}}^{a_0} \text{ exists and is different from } q_0, \\ \text{undefined}, & \text{otherwise.} \end{cases}$$

After this initial transition, the computation proceeds by executing transitions of $\mathcal{A}$ in reverse. This is achieved by defining the transition from a state $[q]$, with $q \in Q$, in a node labelled with $a \in \Sigma$, as follows:

$$\delta'_a([q]) = \begin{cases} [p], & \text{if } \delta_a^{-1}(q) = \{p\} \text{ and } (p, a) \notin \{q_0\} \times \Sigma_0, \\ \text{undefined}, & \text{otherwise.} \end{cases}$$

Note that $\delta_a^{-1}(q) = \{p\}$ is equivalent to $\delta_a(p) = q$, because the automaton $\mathcal{A}$ is reversible. There are two cases of undefined transitions. First, if $\delta_a^{-1}(q)$ is not a singleton, then $\delta_a^{-1}(q) = \varnothing$, that is, the current configuration of $\mathcal{A}$ has no predecessors; then the automaton $\mathcal{A}^{\mathrm{r}}$ rejects. The second case is $\delta_a^{-1}(q) = \{q_0\}$ and $a = a_0 \in \Sigma_0$: if $\mathcal{A}^{\mathrm{r}}$ encounters such a configuration, this means that the initial configuration of $\mathcal{A}$ has been reached, and hence the graph is to be accepted. Accordingly, the set of accepting configurations of $\mathcal{A}^{\mathrm{r}}$ is

$$F' = \left\{ ([\delta_{a_0}(q_0)], a_0) \;\middle|\; a_0 \in \Sigma_0,\ \delta_{a_0}(q_0) \text{ is defined} \right\} \cup \{ (q_0', a_0) \mid (q_0, a_0) \in F \}.$$

In the notation of Definition 4(ii), for every initial label $a_0 \in \Sigma_0$, $q'^{a_0}_{\mathrm{acc}} = [\delta_{a_0}(q_0)]$, if $\delta_{a_0}(q_0)$ is defined, and $q'^{a_0}_{\mathrm{acc}} = q_0'$, if $q_{\mathrm{acc}}^{a_0} = q_0$.

Lemma 5. *For every reversible automaton $\mathcal{A}$ without inaccessible transitions, the above construction correctly defines a reversible automaton $\mathcal{A}^{\mathrm{r}}$ without inaccessible transitions and with separated initial state, which recognizes the same set of graphs as $\mathcal{A}$.*

Proof. First, it will be verified that the construction correctly defines a direction-determinate automaton, that is, if a transition from a node labelled by a is defined, then the necessary direction to the target state is in D_a. First consider transitions from the new initial state, which take place only in the initial node. Assume that $\delta'_{a_0}(q'_0)$ is defined; then it is equal to $[q^{a_0}_{\mathrm{acc}}]$, where $q^{a_0}_{\mathrm{acc}}$ is different from q_0. Since $\mathcal{A}$ is without inaccessible transitions, this accepting state should be enterable in the initial node, and thus the set D_{a_0} contains the direction $-d(q^{a_0}_{\mathrm{acc}}) = d'([q^{a_0}_{\mathrm{acc}}])$, as required. It remains to consider transitions from the states of the form $[q]$, with $q \in Q$. If $\delta'_a([q])$ is defined, then it equals $[p]$ for some $p \in Q$ satisfying $\delta_a(p) = q$ and $(p, a) \notin \{q_0\} \times \Sigma_0$. Since $\mathcal{A}$ is without inaccessible transitions, the state p must be reachable in a-labelled nodes, and so D_a contains the direction $-d(p)$. As this direction $-d(p)$ equals $d'([p])$, there exists a direction from nodes labelled by a, which can be used by the reversed automaton $\mathcal{A}^{\mathrm{r}}$ to go to the state $[p]$.

By definition, the resulting automaton is returning and has only one accepting state for each $a_0 \in \Sigma_0$. It remains to show that each δ'_a is injective. If $\delta'_{a_0}(q'_0)$ were equal to $\delta'_{a_0}([q])$, for some $q \in Q$, then their common value must be $[q^{a_0}_{\mathrm{acc}}]$, and hence $\delta_{a_0}(q^{a_0}_{\mathrm{acc}}) = q$, which is impossible, as the transition function is not defined on accepting configurations. If $\delta'_a([q]) = \delta'_a([s])$, for some $q, s \in Q$, then their common value is $[p]$, where $p \in Q$ satisfies both $\delta_a(p) = q$ and $\delta_a(p) = s$, which implies $q = s$. This confirms that $\mathcal{A}^{\mathrm{r}}$ is reversible.

In order to verify that $\mathcal{A}^{\mathrm{r}}$ is without inaccessible transitions, assume first that $\delta'_a([q])$ is defined. Then it is equal to $[p]$, where the state $p \in Q$ satisfies $q = \delta_a(p)$. This implies that the direction $d(q)$ belongs to D_a, and since $d(q) = -d'([q])$, it means that $-d'([q]) \in D_a$, as required. Secondly, consider any acceptance condition of $\mathcal{A}^{\mathrm{r}}$ outside the initial state, which is of the form $([\delta_{a_0}(q_0)], a_0) \in F'$. Because $\delta_{a_0}(q_0)$ is defined, the corresponding direction $d(\delta_{a_0}(q_0))$ must belong to D_{a_0}, which is again equivalent to the required condition $-d'([\delta_{a_0}(q_0)]) \in D_{a_0}$.

The automaton $\mathcal{A}^{\mathrm{r}}$ has separated initial state by definition.

To see that the reversed automaton recognizes the same graphs, consider the computations of $\mathcal{A}$ and $\mathcal{A}^{\mathrm{r}}$ on a graph $(V, v_0, +, \lambda)$. First note that $\mathcal{A}^{\mathrm{r}}$ accepts immediately in the initial configuration if and only if $\mathcal{A}$ does. Now let $(p_0, u_0), (p_1, u_1), \ldots, (p_\ell, u_\ell)$ be a non-trivial accepting computation of $\mathcal{A}$. It will be verified that $(q'_0, u_\ell), ([p_\ell], u_{\ell-1}), \ldots, ([p_1], u_0)$ is an accepting computation of $\mathcal{A}^{\mathrm{r}}$. The configuration (q'_0, u_ℓ) is the initial configuration, because $u_\ell = v_0$ as the automaton $\mathcal{A}$ is returning. Since $p_\ell = q^{\lambda(u_\ell)}_{\mathrm{acc}}$ and $p_\ell \neq$

q_0 ($p_\ell = q_0$ would mean that already (p_0, u_0) is an accepting configuration), the first transition from (q'_0, u_ℓ) is $\delta'_{\lambda(u_\ell)}(q'_0) = [p_\ell]$, and the automaton goes from u_ℓ to $u_\ell + d'([p_\ell]) = u_\ell - d(p_\ell) = u_{\ell-1}$. For all $i \in \{1, \ldots, \ell - 1\}$, the computation of $\mathcal{A}$ satisfies $\delta_{\lambda(u_i)}(p_i) = p_{i+1}$ and the pair $(p_i, \lambda(u_i))$ does not belong to $\{q_0\} \times \Sigma_0$. Therefore, $\delta'_{\lambda(u_i)}([p_{i+1}]) = [p_i]$ according to the definition of $\delta'_{\lambda(u_i)}$, which shows that the above computation of $\mathcal{A}^{\mathrm{r}}$ is correct on states. Concerning directions, $\mathcal{A}^{\mathrm{r}}$ goes from each configuration $([p_{i+1}], u_i)$, with $i \in \{1, \ldots, \ell - 1\}$, to the node $u_i + d'([p_i]) = u_i - d(p_i) = u_{i-1}$. Finally, the configuration $([p_1], u_0)$ is accepting, since $u_0 = v_0$ and $p_1 = \delta_{\lambda(v_0)}(q_0)$.

Conversely, let $(p'_0, u_0), (p'_1, u_1), \ldots, (p'_\ell, u_\ell)$ be a non-trivial accepting computation of $\mathcal{A}^{\mathrm{r}}$. Then for each $i \in \{1, \ldots, \ell\}$, $p'_i = \delta'_{\lambda(u_{i-1})}(p'_{i-1})$, which means that $p'_i = [p_i]$ for some $p_i \in Q$. Consider the sequence $(q_0, v_0), (p_\ell, u_{\ell-1}), \ldots, (p_1, u_0)$ of configurations of $\mathcal{A}$. Because (p'_ℓ, u_ℓ) is an accepting configuration of $\mathcal{A}^{\mathrm{r}}$, $u_\ell = v_0$ and the corresponding state of $\mathcal{A}$ satisfies $p_\ell = \delta_{\lambda(v_0)}(q_0)$. Additionally, $u_{\ell-1} = u_\ell - d'(p'_\ell) = v_0 + d(p_\ell)$. This verifies that the second configuration in this sequence is obtained from the initial configuration by one step of $\mathcal{A}$. The definition of transitions of $\mathcal{A}^{\mathrm{r}}$ directly gives $\delta_{\lambda(u_i)}(p_{i+1}) = p_i$ for $i \in \{1, \ldots, \ell - 1\}$, while the corresponding motion over the nodes is $u_i + d(p_i) = u_i - d'(p'_i) = u_{i-1}$. Finally, the equality $p'_1 = \delta'_{\lambda(v_0)}(q'_0)$ gives $p_1 = q_{\mathrm{acc}}^{\lambda(v_0)}$, which implies that $\mathcal{A}$ accepts in the configuration (p_1, u_0). This shows that the given sequence is an accepting computation of $\mathcal{A}$. □

If the reversible automaton $\mathcal{A}$ additionally has separated initial state, then the definition of transitions $\delta'_a([q])$ of $\mathcal{A}^{\mathrm{r}}$ can be equivalently reformulated as follows:

$$\delta'_a([q]) = \begin{cases} [p], & \text{if } \delta_a^{-1}(q) = \{p\}, \text{ with } p \neq q_0, \\ \text{undefined}, & \text{otherwise (that is, if } \delta_a^{-1}(q) = \varnothing \text{ or } \delta_a^{-1}(q) = \{q_0\}), \end{cases}$$

for $q \in Q$ and $a \in \Sigma$. This in particular implies that for such an automaton $\mathcal{A}$, no transition of $\mathcal{A}^{\mathrm{r}}$ leads to the state $[q_0]$. Therefore, the automaton $\mathcal{A}^{\mathrm{R}}$ obtained from $\mathcal{A}^{\mathrm{r}}$ by removing $[q_0]$ is still equivalent to $\mathcal{A}$.

Lemma 6. *For every reversible automaton $\mathcal{A}$ without inaccessible transitions and with separated initial state, the automaton $(\mathcal{A}^{\mathrm{R}})^{\mathrm{R}}$ is isomorphic to $\mathcal{A}$.*

Proof. Assuming $\mathcal{A} = (Q, q_0, \delta, F)$, the set of states of $(\mathcal{A}^{\mathrm{R}})^{\mathrm{R}}$ is $\{\, [[q]] \mid q \in Q,\ q \neq q_0 \,\} \cup \{q''_0\}$. The isomorphism maps q_0 to q''_0 and each $q \in Q \setminus \{q_0\}$ to $[[q]]$.

First, it will be verified that accepting states in $\mathcal{A}$ and $(\mathcal{A}^{\mathrm{R}})^{\mathrm{R}}$ exist for the same initial labels $a_0 \in \Sigma_0$, and that every accepting state $q_{\mathrm{acc}}^{a_0}$ of $\mathcal{A}$ is mapped to the corresponding accepting state $q''^{a_0}_{\mathrm{acc}}$ of $(\mathcal{A}^{\mathrm{R}})^{\mathrm{R}}$. The accepting state $q''^{a_0}_{\mathrm{acc}}$ is defined as the initial state q''_0 if and only if $q'^{a_0}_{\mathrm{acc}}$ is equal to the initial state q'_0 of $\mathcal{A}^{\mathrm{R}}$, which is equivalent to $q_{\mathrm{acc}}^{a_0} = q_0$. The other accepting

states of $(\mathcal{A}^{\mathrm{R}})^{\mathrm{R}}$ are ${q''}^{a_0}_{\mathrm{acc}} = [\delta'_{a_0}(q'_0)]$, for $a_0 \in \Sigma_0$, such that $\delta'_{a_0}(q'_0)$ is defined. However, $\delta'_{a_0}(q'_0)$ is defined if and only if $q^{a_0}_{\mathrm{acc}} \neq q_0$ exists, and it is equal to $[q^{a_0}_{\mathrm{acc}}]$. This shows that ${q''}^{a_0}_{\mathrm{acc}}$ exists and is different from q''_0 precisely when $q^{a_0}_{\mathrm{acc}}$ exists and is different from q_0, and in this case ${q''}^{a_0}_{\mathrm{acc}} = [[q^{a_0}_{\mathrm{acc}}]]$.

The directions for entering states in $(\mathcal{A}^{\mathrm{R}})^{\mathrm{R}}$ correspond to those in $\mathcal{A}$, since $d''([[q]]) = -d'([q]) = d(q)$ for all $q \in Q \setminus \{q_0\}$. Transitions $\delta''_a(q''_0)$ from the initial state of $(\mathcal{A}^{\mathrm{R}})^{\mathrm{R}}$ are defined only for $a \in \Sigma_0$, which is the same as for $\delta_a(q_0)$ in $\mathcal{A}$. For $a_0 \in \Sigma_0$, $\delta''_{a_0}(q''_0) = [{q'}^{a_0}_{\mathrm{acc}}]$, if the accepting state ${q'}^{a_0}_{\mathrm{acc}}$ exists and is different from q'_0. This is the case if and only if $\delta_{a_0}(q_0)$ is defined, and in this case ${q'}^{a_0}_{\mathrm{acc}} = [\delta_{a_0}(q_0)]$, which proves that transitions from the initial state in $(\mathcal{A}^{\mathrm{R}})^{\mathrm{R}}$ agree with those in $\mathcal{A}$. Finally, for $q \in Q \setminus \{q_0\}$, according to the above reformulation of the definition of transitions, $\delta''_a([[q]])$ is defined if and only if ${\delta'_a}^{-1}([q])$ contains a state of $\mathcal{A}^{\mathrm{R}}$ different from q'_0, that is, there exists $p \in Q \setminus \{q_0\}$ such that $[q] = \delta'_a([p])$. Additionally, if it is the case, then $\delta''_a([[q]]) = [[p]]$. Since $q \neq q_0$, the condition $[q] = \delta'_a([p])$ is in turn equivalent to $p = \delta_a(q)$. As $\mathcal{A}$ has separated initial state, the requirement that $p \neq q_0$ also follows from $p = \delta_a(q)$. Therefore, $\delta''_a([[q]])$ is defined if and only if $\delta_a(q)$ is defined, and it is equal to $[[\delta_a(q)]]$, as required. □

6 The reversibility construction

This section presents the fundamental construction behind all results of this paper: the reversible simulation of an arbitrary deterministic graph-walking automaton. Quite expectedly, this simulation will then be used to obtain Theorem 2 on transforming a returning automaton to an equivalent reversible automaton. Not so expectedly, the same simulation is behind Theorem 1 on transforming an automaton to an equivalent returning automaton.

The construction to be presented is a generalization of Lemma 5 on simulating a given reversible automaton backwards, by backtracking its potential accepting computation. The generalized construction applies to any direction-determinate automaton, which need not be reversible. Because of the irreversibility, the original automaton may have multiple computations arriving to the accepting configuration, of which at most one could begin in the initial configuration. The task of the constructed automaton is to traverse the tree of all computations leading to the accepting configuration (or, more generally, to any fixed configuration), searching for the initial configuration. The simulation alternates between following the transitions of the original automaton as they are, and following them in the backward direction, as in the above Lemma 5.

Lemma 7. *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, where $\overrightarrow{Q} = \{\, \overrightarrow{q} \mid q \in Q \,\}$ and $[Q] = \{\, [q] \mid q \in Q \,\}$ are disjoint copies of Q, with the corresponding di-*

rections $d'(\overrightarrow{q}) = d(q)$ *and* $d'([q]) = -d(q)$*, and with accepting states* $F' = \left\{ ([\delta_{a_0}(q_0)], a_0) \;\middle|\; a_0 \in \Sigma_0,\ \delta_{a_0}(q_0) \text{ is defined} \right\}$*, which has the following property: For every 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* $([\widehat{q}], \widehat{v} - d(\widehat{q}))$*,*

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

Furthermore,

i. For every $q \in Q$ *and* $a \in \Sigma$*, the transition* $\delta'_a([q])$ *is undefined if and only if* $d(q) \notin D_a$ *or* $([q], a) \in F'$ *(where the latter means that* $a \in \Sigma_0$ *and* $q = \delta_a(q_0)$*).*

ii. For every $p \in Q$ *and* $a \in \Sigma$*, the transition* $\delta'_a(\overrightarrow{p})$ *is undefined if and only if* $-d(p) \notin D_a$ *or* $(p, a) \in F$*.*

iii. For every $r \in Q$ *and* $a \in \Sigma \setminus \Sigma_0$*,* ${\delta'_a}^{-1}(\overrightarrow{r}) = \varnothing$ *if and only if* $d(r) \notin D_a$*.*

iv. For every $r \in Q$ *and* $a \in \Sigma \setminus \Sigma_0$*,* ${\delta'_a}^{-1}([r]) = \varnothing$ *if and only if* $-d(r) \notin D_a$ *or* $(r, a) \in F$*. Additionally, for* $a_0 \in \Sigma_0$*, if* $(r, a_0) \in F$*, then* ${\delta'_{a_0}}^{-1}([r]) = \varnothing$*.*

Proof. Assume any linear ordering on Q, under which q_0 is the least element of Q. Let $\min S$ and $\max S$ denote the least and the greatest element of a nonempty set $S \subseteq Q$ with respect to this ordering. Let $\mathrm{next}_S(q)$ with $q \in S \subseteq Q$ denote the least element of S strictly greater than q, provided that it exists.

The new automaton searches through the tree of computations leading to the configuration $(\widehat{q}, \widehat{v})$, looking for the initial configuration. This involves both backward simulation of the original automaton, when exploring each branch of this tree, as well as forward simulation, which is used when the backward search results in a configuration unreachable by the original automaton, and hence the search should turn to the next branch. For that purpose, the new automaton has the set of states $\overrightarrow{Q} \cup [Q]$, with each state from Q represented by two states, $\overrightarrow{q} \in \overrightarrow{Q}$ and $[q] \in [Q]$. In the states of the form $[q]$, the automaton simulates the computation backwards and has $d'([q]) = -d(q)$, while states $\overrightarrow{q}$ with $d'(\overrightarrow{q}) = d(q)$ are used for forward simulation. Whenever the new automaton reaches a state $[q]$ in a node v, this means that the computation of the original automaton, beginning in the state q with the head in the neighbouring node $v + d(q)$, eventually leads to the configuration $(\widehat{q}, \widehat{v})$. In this way, exactly as in Lemma 5, the backward computation traces the state and the position of the head in a forward computation, but the state and the position are always out of synchronization by

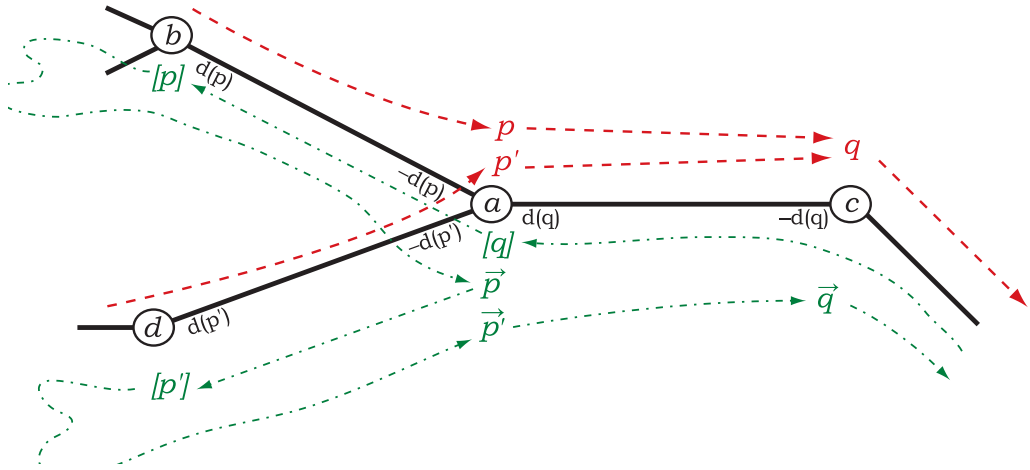

Figure 6: A reversible GWA $\mathcal{B}$ searching the tree of computations of a GWA $\mathcal{A}$: handling irreversibility $\delta_a(p) = \delta_a(p') = q$ with $\delta_a^{-1}(q) = \{p, p'\}$ and $p < p'$.

one step. When the automaton switches to forward simulation, and reaches a state $\overrightarrow{q}$, its head position is synchronized with the state, and this represents the original automaton's being in the state q, observing the same node.

Consider the behaviour of the new automaton in a state $[q]$, while observing a node labelled by $a \in \Sigma$. It is convenient to begin with the acceptance conditions. If $a \in \Sigma_0$ and $q = \delta_a(q_0)$, then the new automaton $\mathcal{B}$ has found the initial configuration of $\mathcal{A}$, and it may accept; accordingly, the pair $([q], a)$ belongs to F'. Otherwise, if the state q is reachable from some other states by the symbol a in the original automaton, then the constructed automaton continues with the backward simulation by choosing the least of those predecessor states, $p = \min \delta_a^{-1}(q)$, and moving to the state $[p]$ in the direction $-d(p)$, as illustrated in Figure 6. This is done by the transition

$$\delta'_a([q]) = [\min \delta_a^{-1}(q)], \qquad \text{if } \delta_a^{-1}(q) \neq \varnothing \text{ and } ([q], a) \notin F'. \tag{1}$$

This is a transition in the direction $d'([\min \delta_a^{-1}(q)])$, and it remains to argue that nodes with the label a allow moving in this direction. Since $\delta_a(p)$ is defined and the situation $(p, a) \in \{q_0\} \times \Sigma_0$ is ruled out by the condition $([q], a) \notin F'$, the assumption that the automaton $\mathcal{A}$ is without inaccessible transitions implies that the direction $-d(p) = d'([p])$ belongs to D_a, as required by the definition of graph-walking automata.

The other principal case in the backward simulation is when a branch of the tree of computations is traced back to a configuration without predecessors. Therefore, these computations cannot take place when starting from the initial configuration of $\mathcal{A}$, and the constructed automaton switches to forward simulation, until it finds the next suitable branch of the tree. This

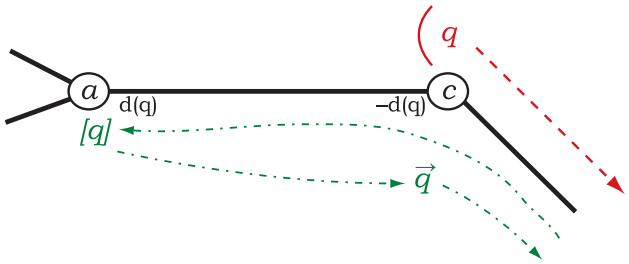

Figure 7: A reversible GWA $\mathcal{B}$ searching the tree of computations of an irreversible GWA $\mathcal{A}$: handling an unreachable configuration.

is done by the following transition:

$$\delta'_a([q]) = \overrightarrow{q}, \qquad \text{if } \delta_a^{-1}(q) = \varnothing \text{ and } d(q) \in D_a, \tag{2}$$

which is compatible with the direction of the state $\overrightarrow{q}$, as $d'(\overrightarrow{q}) = d(q) \in D_a$. This case is illustrated in Figure 7.

The transitions on $[q]$ with $d(q) \notin D_a$ are undefined, because such states cannot be reached from the proper direction in the backward simulation. Actually, a configuration $([q], v)$, with $\lambda(v) = a$, makes no sense, since it should represent the original automaton's being in the state q in the node $v + d(q)$, whereas the latter node does not exist.

Transitions from forward states $\overrightarrow{p} \in \overrightarrow{Q}$ generally simulate the original automaton $\mathcal{A}$, while looking for converging computations, ready to backtrack the next available branch in the tree of computations. Consider the case when multiple branches of this tree converge at the present point, that is, $\delta_a(p) = q$ and there exist other states $p' \neq p$ with $\delta_a(p') = q$. Assume that the current branch is not the last of these branches, so that there is a state $p' > p$ with $\delta_a(p') = q$. Let p' be the least of such states, denoted by $\mathrm{next}_{\delta_a^{-1}(q)}(p)$. Then, as illustrated in Figure 6, the simulation switches to backtracking the next branch, proceeding to the state $[p']$ in the appropriate direction. Accordingly, define the following transition:

$$\delta'_a(\overrightarrow{p}) = [p'], \quad \text{if } \delta_a(p) \text{ is defined, } p' = \mathrm{next}_{\delta_a^{-1}(\delta_a(p))}(p) \text{ and } -d(p) \in D_a. \tag{3}$$

The direction required for this transition always exists in D_a, because the automaton $\mathcal{A}$ has the transition $\delta_a(p')$ defined, and $p' \neq q_0$, as p' is not the least element of Q. Accordingly, the state p' must be accessible by transitions in the direction $d(p')$, which necessitates $d'([p']) = -d(p') \in D_a$.

If there was no branching at the present point, or if there was a branching, but the current branch is already the last of them (this is a single case), then the forward simulation continues. This is what happens in the state $\overrightarrow{p'}$ over

the symbol a in Figure 6. In general, such transitions are defined as follows:

$$\delta'_a(\overrightarrow{p}) = \overrightarrow{\delta_a(p)}, \quad \text{if } \delta_a(p) \text{ is defined, } p = \max \delta_a^{-1}(\delta_a(p)) \text{ and } -d(p) \in D_a, \tag{4}$$

where the corresponding direction $d'(\overrightarrow{\delta_a(p)})$ exists in D_a, because it is the same as $d(\delta_a(p))$.

It remains to define the transitions of $\mathcal{B}$ in a state $\overrightarrow{p}$ in the case of undefined $\delta_a(p)$:

$$\delta'_a(\overrightarrow{p}) = [p], \qquad \text{if } \delta_a(p) \text{ is undefined, } -d(p) \in D_a, (p,a) \notin F, \tag{5}$$

where the assumption $-d(p) \in D_a$ guarantees that the required direction $d'([p]) = -d(p)$ exists in D_a. These transitions shall never be used, but they are necessary to comply with the definition of strong reversibility.

If the constructed automaton ever reaches any configuration $(\overrightarrow{p}, v)$ corresponding to an accepting configuration (p, v) of $\mathcal{A}$, then it rejects, that is, for every $(p, a) \in F$, $\delta'_a(\overrightarrow{p})$ is not defined. Note that these are the only undefined transitions in the automaton, besides the inaccessible transitions $\delta'_a(\overrightarrow{q})$ with $-d(q) \notin D_a$ and $\delta'_a([q])$ with $d(q) \notin D_a$. Altogether, the above construction correctly defines a direction-determinate automaton $\mathcal{B}$, which satisfies claims (i) and (ii) of the lemma.

Fix an input graph $(V, v_0, +, \lambda)$. For every configuration (p, v), let $\pi(p, v)$ denote the (uniquely determined) computation of $\mathcal{A}$ from (p, v). The above ordering of states induces the following strict partial ordering on computations of $\mathcal{A}$, which accept in the configuration $(\widehat{q}, \widehat{v})$: a computation $\pi(p_\ell, u_\ell) = (p_\ell, u_\ell) \dots (p_1, u_1)(\widehat{q}, \widehat{v})$ is less than $\pi(p'_{\ell'}, u'_{\ell'}) = (p'_{\ell'}, u'_{\ell'}) \dots (p'_1, u'_1)(\widehat{q}, \widehat{v})$ if there exists such a number $m \leqslant \ell, \ell'$ that $(p_i, u_i) = (p'_i, u'_i)$ for all $i \in \{1, \dots, m-1\}$ and $p_m < p'_m$.

$$\begin{array}{cccccccc}
(p_\ell, u_\ell) & & \dots & (p_m, u_m) & (p_{m-1}, u_{m-1}) & \dots & (p_1, u_1) & (\widehat{q}, \widehat{v}) \\
 & & & \wedge & \| & \dots & \| & \\
 & (p'_{\ell'}, u'_{\ell'}) & \dots & (p'_m, u'_m) & (p'_{m-1}, u'_{m-1}) & \dots & (p'_1, u'_1) & (\widehat{q}, \widehat{v})
\end{array}$$

Because the automaton $\mathcal{A}$ is direction-determinate, every configuration (p_i, u_i) in a computation uniquely determines the previous node $u_{i-1} = u_i - d(p_i)$, and thus a computation is determined by its sequence of states and its last node. Therefore, two computations ending with the same configuration are incomparable if and only if one of them is a suffix of the other.

The correctness statement of the construction reads as follows:

Claim 1. *If the computation of the new automaton $\mathcal{B}$ beginning in $([\widehat{q}], \widehat{v} - d(\widehat{q}))$ reaches a configuration $([q], v)$ in zero or more steps, then the computation of $\mathcal{A}$ beginning in $(q, v + d(q))$ accepts in the configuration $(\widehat{q}, \widehat{v})$. If additionally $(\widehat{q}, \widehat{v}) \neq (q_0, v_0)$ and the computation of $\mathcal{A}$ beginning in (q_0, v_0) accepts*

in $(\widehat{q},\widehat{v})$ *too, then the computation* $\pi(q_0,v_0)$ *is neither less than* $\pi(q,v+d(q))$*, nor a suffix of* $\pi(q,v+d(q))$*.*

If the computation of $\mathcal{B}$ *beginning in* $([\widehat{q}],\widehat{v}-d(\widehat{q}))$ *reaches a configuration* $(\overrightarrow{p},v)$*, then the computation of* $\mathcal{A}$ *beginning in* (p,v) *accepts in* $(\widehat{q},\widehat{v})$*. If additionally* $(\widehat{q},\widehat{v})\neq(q_0,v_0)$ *and the computation of* $\mathcal{A}$ *beginning in* (q_0,v_0) *accepts in* $(\widehat{q},\widehat{v})$ *too, then* $\pi(q_0,v_0)$ *is greater than* $\pi(p,v)$*.*

The claim is proved by induction on the length of the computation of the new automaton.

Basis. After zero steps of the new automaton, its current configuration $([q],v)$ has $q=\widehat{q}$ and $v=\widehat{v}-d(\widehat{q})$, which means that $(q,v+d(q))=(\widehat{q},\widehat{v})$, which is an accepting configuration of $\mathcal{A}$. If $(\widehat{q},\widehat{v})\neq(q_0,v_0)$, then this zero-step computation $\pi(q,v+d(q))=(\widehat{q},\widehat{v})$ is incomparable with every non-trivial computation accepting in the configuration $(\widehat{q},\widehat{v})$.

Induction step I. Assume that the constructed automaton $\mathcal{B}$ reaches a configuration $([q],v)$, for which the statement of the lemma holds true, that is, the original automaton started in $(q,v+d(q))$ accepts in $(\widehat{q},\widehat{v})$ and $\pi(q_0,v_0)$ is neither less than, nor a suffix of $\pi(q,v+d(q))$, all provided that $(\widehat{q},\widehat{v})\neq(q_0,v_0)$ and the computation of the original automaton beginning in (q_0,v_0) accepts in $(\widehat{q},\widehat{v})$.

Assuming that the constructed automaton does not accept in $([q],v)$, consider the next step of its computation. First assume that $\delta^{-1}_{\lambda(v)}(q)\neq\varnothing$ and let $p=\min\delta^{-1}_{\lambda(v)}(q)$. Then the new automaton makes a transition (1) from $([q],v)$ to $([p],v-d(p))$, while the original automaton goes from $\big(p,(v-d(p))+d(p)\big)=(p,v)$ to $(q,v+d(q))$, from whence it accepts in $(\widehat{q},\widehat{v})$ by the induction assumption. Now let $(\widehat{q},\widehat{v})\neq(q_0,v_0)$. Suppose, for the sake of contradiction, that the computation of the original automaton beginning in (q_0,v_0) accepts in $(\widehat{q},\widehat{v})$ and $\pi(q_0,v_0)<\pi(p,v)$. Since $\pi(p,v)=(p,v)\cdot\pi(q,v+d(q))$ and $\pi(q_0,v_0)$ is not less than $\pi(q,v+d(q))$ by the assumption, it follows that $\pi(q_0,v_0)$ and $\pi(p,v)$ must be different in the first step of $\pi(p,v)$, that is, $\pi(q_0,v_0)=(q_0,v_0)\ldots(p',v)\cdot\pi(q,v+d(q))$ and $p'<p$. Then $\delta_{\lambda(v)}(p')=q$ and so $p'\in\delta^{-1}_{\lambda(v)}(q)$, which contradicts the assumption that p is the least element of $\delta^{-1}_{\lambda(v)}(q)$. Finally, it has to be verified that $\pi(q_0,v_0)$ is not a suffix of $\pi(p,v)$. Since it is not a suffix of $\pi(q,v+d(q))$ by the induction hypothesis, this could only happen if $\pi(q_0,v_0)=\pi(p,v)$. However, this would imply that $v=v_0$ and $q=\delta_{\lambda(v)}(p)=\delta_{\lambda(v_0)}(q_0)$, and therefore $([q],\lambda(v))$ would belong to F', which would contradict the assumption that $\mathcal{B}$ does not accept in $([q],v)$.

The other possible next step of the new automaton from the configuration $([q],v)$ occurs for $\delta^{-1}_{\lambda(v)}(q)=\varnothing$: then it proceeds to the configuration $(\overrightarrow{q},v+d(q))$, and it has to be established that the original automaton, starting from $(q,v+d(q))$, accepts in $(\widehat{q},\widehat{v})$. That is directly given by the induction hypothesis for $([q],v)$, which furthermore states that if the original automaton, starting from (q_0,v_0), accepts in $(\widehat{q},\widehat{v})\neq(q_0,v_0)$, then $\pi(q_0,v_0)$ cannot be less than $\pi(q,v+d(q))$ and cannot be a suffix of $\pi(q,v+d(q))$.

To see that $\pi(q_0, v_0)$ is greater than $\pi(q, v + d(q))$, it remains to argue that $\pi(q, v + d(q))$ is not a proper suffix of $\pi(q_0, v_0)$. Indeed, if it were a proper suffix, then there would have been a previous configuration (p, v), from which the original automaton would go to $(q, v + d(q))$; but this would imply that $p \in \delta^{-1}_{\lambda(v)}(q)$ and thus contradict the assumption.

Induction step II. Let the claim hold for a configuration $(\overrightarrow{p}, v)$ of the automaton $\mathcal{B}$, that is, the computation $\pi(p, v)$ of $\mathcal{A}$ is assumed to accept in $(\widehat{q}, \widehat{v})$, and in case the automaton $\mathcal{A}$, beginning in (q_0, v_0), accepts in $(\widehat{q}, \widehat{v}) \neq (q_0, v_0)$ too, the computation $\pi(q_0, v_0)$ is greater than $\pi(p, v)$. The statement of the claim is to be established for the configuration of the automaton $\mathcal{B}$ at the next step after $(\overrightarrow{p}, v)$. Since the computation $\pi(p, v)$ is accepting, there are two possibilities: either it consists of a unique accepting configuration, or begins with a transition to $\big(\delta_{\lambda(v)}(p), v + d(\delta_{\lambda(v)}(p))\big)$.

In the former case, the pair $(p, \lambda(v))$ belongs to F, which implies that $\delta'_{\lambda(v)}(\overrightarrow{p})$ is not defined. Therefore, the configuration $(\overrightarrow{p}, v)$ of $\mathcal{B}$ is rejecting, and it is not followed by any other configuration, so there is nothing to prove.

Assume that the computation $\pi(p, v)$ of $\mathcal{A}$ is non-trivial, that is, it begins with a transition. Denote the next state $\delta_{\lambda(v)}(p)$ by q. If p is **the greatest** state in $\delta^{-1}_{\lambda(v)}(q)$, then the constructed automaton follows the original automaton forward, going from $(\overrightarrow{p}, v)$ to $\big(\overrightarrow{q}, v + d(q)\big)$. The computation $\pi(q, v + d(q))$ of the original automaton accepts in $(\widehat{q}, \widehat{v})$, since it is a suffix of the computation $\pi(p, v)$. If additionally the original automaton accepts in $(\widehat{q}, \widehat{v}) \neq (q_0, v_0)$ when beginning in (q_0, v_0), and $\pi(q_0, v_0)$ were less than $\pi(q, v + d(q))$, then $\pi(q_0, v_0)$ would also be less than the longer computation $\pi(p, v)$, contradicting the induction hypothesis. For the same reason, the computation $\pi(q_0, v_0)$ cannot be a suffix of $\pi(q, v + d(q))$. Suppose $\pi(q, v + d(q))$ is a proper suffix of $\pi(q_0, v_0)$. Then, since $\pi(p, v)$ is not a suffix of $\pi(q_0, v_0)$ by the induction hypothesis, the computations $\pi(p, v) = (p, v) \cdot \pi(q, v + d(q))$ and $\pi(q_0, v_0)$ differ on the first step of $\pi(p, v)$, and $\pi(q_0, v_0) = (q_0, v_0) \ldots (p', v) \cdot \pi(q, v + d(q))$ for some state $p' \neq p$ with $\delta_{\lambda(v)}(p') = q$. Then $p' < p$, because p is known to be the greatest of such states, and therefore $\pi(q_0, v_0) < \pi(p, v)$, a contradiction.

Otherwise, if **there is a greater** state $p' > p$ with $\delta_{\lambda(v)}(p') = q$, the automaton $\mathcal{B}$ proceeds from $(\overrightarrow{p}, v)$ to $([p'], v - d(p'))$, where p' is the least of such states, that is, $p' = \mathrm{next}_{\delta^{-1}_{\lambda(v)}(q)}(p)$. Then the original automaton in the configuration $(p', v - d(p') + d(p')) = (p', v)$ will apply the transition $\delta(p', \lambda(v)) = (q, d(q))$ and arrive to the configuration $(q, v + d(q))$; this is also the next configuration after (p, v), and hence the original automaton accepts from it in $(\widehat{q}, \widehat{v})$. Now assume that the original automaton, beginning in (q_0, v_0), accepts in $(\widehat{q}, \widehat{v}) \neq (q_0, v_0)$. The computation $\pi(q_0, v_0)$ cannot be a proper suffix of $\pi(p', v)$, because $\pi(p', v)$ differs from $\pi(p, v)$ only in the first configuration. The computations $\pi(q_0, v_0)$ and $\pi(p', v)$ also cannot be equal, since that would imply $p < p' = q_0$, which is impossible, because q_0 is the

least element of Q. Finally, suppose that the computation $\pi(q_0, v_0)$ is less than $\pi(p', v) = (p', v) \cdot \pi(q, v + d(q))$. Since $\pi(q_0, v_0)$ is greater than $\pi(p, v)$ by the assumption, it is not less than $\pi(q, v + d(q))$, and hence the computation $\pi(q_0, v_0)$ must differ from $\pi(p', v)$ in the first configuration of $\pi(p', v)$, that is, $\pi(q_0, v_0)$ has a suffix $(p'', v) \cdot \pi(q, v + d(q))$ with $\delta_{\lambda(v)}(p'') = q$ and $p'' < p'$. On the other hand, $p'' > p$, because otherwise $\pi(q_0, v_0)$ would not be greater than $\pi(p, v)$. Altogether, $p < p'' < p'$ and $\delta_{\lambda(v)}(p) = \delta_{\lambda(v)}(p'') = \delta_{\lambda(v)}(p') = q$, which contradicts the choice of p' as the *least* element of $\delta^{-1}_{\lambda(v)}(q)$ greater than p. This completes the proof of Claim 1.

Claim 2. *The constructed automaton $\mathcal{B}$ is reversible.*

To see that for every label $a \in \Sigma$, no state of $\mathcal{B}$ can be reached from two different states by a transition reading a, first consider any state $\overrightarrow{r} \in \overrightarrow{Q}$. This state can be reached by δ'_a only using transitions of one of the forms (2) and (4). There can be only one transition of the form (2) (it goes from the state $[r]$), and it can occur only if $\delta_a^{-1}(r) = \varnothing$. On the other hand, transitions of the form (4) occur only if $\delta_a^{-1}(r) \neq \varnothing$, and $\overrightarrow{r}$ can be reached by such a transition from exactly one state, namely $\overrightarrow{p}$ with $p = \max \delta_a^{-1}(r)$. This shows that $|{\delta'_a}^{-1}(\overrightarrow{r})| \leqslant 1$.

Every state $[r] \in [Q]$ can be possibly reached by transitions of three different types: (1), (3) and (5). Transitions of the first type occur only if r is the least element of $\delta_a^{-1}(\delta_a(r))$, and in this case, $[r]$ can only be reached from the state $[q]$ with $q = \delta_a(r)$. Transitions of the second type (3) can only occur if $\delta_a(r)$ is defined, but r is not the least element of $\delta_a^{-1}(\delta_a(r))$; then there is also at most one state from which $[r]$ can be reached, namely $\overrightarrow{p}$, where p is the predecessor of r in $\delta_a^{-1}(\delta_a(r))$. Finally, there can be only one possible transition of type (5), reaching $[r]$ from the state $\overrightarrow{r}$, and it occurs only if $\delta_a(r)$ is not defined. Altogether, this proves that $|{\delta'_a}^{-1}([r])| \leqslant 1$.

Turning to the second condition in the definition of reversibility, by the definition of F', the automaton $\mathcal{B}$ is returning and has at most one accepting state for each $a_0 \in \Sigma_0$, which concludes the proof of Claim 2.

Claim 3. *The automaton $\mathcal{B}$ satisfies assertions* (iii) *and* (iv) *of the lemma.*

This claim is established by an analysis similar to the proof of Claim 2. Let $a \in \Sigma \setminus \Sigma_0$ and consider any state $\overrightarrow{r} \in \overrightarrow{Q}$. If $\delta_a^{-1}(r) = \varnothing$, then $\overrightarrow{r}$ can be reached by δ'_a only using the transition (2), which is not defined if and only if $d(r) \notin D_a$. If $\delta_a^{-1}(r) \neq \varnothing$, then this state can only be reached by a transition (4) from a state $\overrightarrow{p}$ with $\delta_a(p) = r$. Such a transition (4) is always defined, because the last condition $-d(p) \in D_a$ is true due to the assumptions that $\mathcal{A}$ is without inaccessible transitions and $a \notin \Sigma_0$. Since $d(r) \in D_a$ always holds in the case $\delta_a^{-1}(r) \neq \varnothing$, assertion (iii) is verified also in this case.

In order to verify the first part of assertion (iv), let $a \in \Sigma \setminus \Sigma_0$ and $[r] \in [Q]$. If $\delta_a(r)$ is defined and r is the least element of $\delta_a^{-1}(\delta_a(r))$, then $[r]$

can be reached from the state $[q]$ with $q = \delta_a(r)$ by a transition (1), that is, ${\delta'_a}^{-1}([r]) \neq \varnothing$. At the same time, $-d(r) \in D_a$ (since there are no inaccessible transitions in $\mathcal{A}$) and $(r, a) \notin F$ (because the transition $\delta_a(r)$ is defined). If $\delta_a(r)$ is defined, but r is not the least element of $\delta_a^{-1}(\delta_a(r))$, then $[r]$ can be reached by a transition (3) from the state $\overrightarrow{p}$, where p is the predecessor of r in $\delta_a^{-1}(\delta_a(r))$, and the assertion is verified in the same way as in the previous case. Finally, if $\delta_a(r)$ is not defined, then the only possibility for reaching $[r]$ is by a transition (5) from the state $\overrightarrow{r}$, and this transition does not exist if and only if $-d(r) \notin D_a$ or $(r, a) \in F$, as required. Altogether, it was verified that $\mathcal{B}$ satisfies the first part of assertion (iv). In order to verify the second part of this assertion, note that if $a_0 \in \Sigma_0$ and $(r, a_0) \in F$, then $\delta_{a_0}(r)$ is undefined, and therefore only transitions (5) could be used to reach $[r]$. However, no such transition exists due to the assumption that $(r, a_0) \in F$. This concludes the proof of Claim 3.

It remains to verify the main claim of the lemma. Because the automaton $\mathcal{B}$ is reversible, by Lemma 2, either the computation of $\mathcal{B}$ beginning in the configuration $([\widehat{q}], \widehat{v} - d(\widehat{q}))$ is finite, or it eventually returns back to $([\widehat{q}], \widehat{v} - d(\widehat{q}))$.

First assume that the configuration $([\widehat{q}], \widehat{v} - d(\widehat{q}))$ is re-entered. Then the node visited in the previous configuration must be $\widehat{v}$, and the automaton has to use one of the rules (1), (3), or (5) in the last step. However, the first two rules require that $\delta_{\lambda(\widehat{v})}(\widehat{q})$ is defined, while the last rule requires that $(\widehat{q}, \lambda(\widehat{v})) \notin F$. Thus in all three cases a contradiction with the assumption $(\widehat{q}, \lambda(\widehat{v})) \in F$ is obtained, which shows that the computation of $\mathcal{B}$ is finite.

Consider the case where the computation ends immediately in the configuration $([\widehat{q}], \widehat{v} - d(\widehat{q}))$. Because $-d(\widehat{q}) \in D_{\lambda(\widehat{v})}$ by assumption, the opposite direction $d(\widehat{q})$ belongs to $D_{\lambda(\widehat{v} - d(\widehat{q}))}$. Therefore, by the earlier proved assertion (i) of this lemma, the only reason why $\mathcal{B}$ could have undefined transition from the configuration $([\widehat{q}], \widehat{v} - d(\widehat{q}))$ can be that $\lambda(\widehat{v} - d(\widehat{q})) \in \Sigma_0$ and $\widehat{q} = \delta_{\lambda(\widehat{v} - d(\widehat{q}))}(q_0)$. This implies that $\widehat{v} - d(\widehat{q}) = v_0$, and consequently the configuration $([\widehat{q}], \widehat{v} - d(\widehat{q}))$ of $\mathcal{B}$ is accepting, while the original automaton goes from the initial configuration directly to $(\widehat{q}, \widehat{v})$.

If the computation of $\mathcal{B}$ ends after at least one step in a configuration $([q], v)$, for some $q \in Q$ and $v \in V$, then $d(q) \in D_{\lambda(v)}$, because this configuration was entered using the direction $d'([q]) = -d(q)$. Since $\delta'_{\lambda(v)}([q])$ is not defined, by assertion (i), it must be the case that $\lambda(v) \in \Sigma_0$ and $q = \delta_{\lambda(v)}(q_0)$, and so $\mathcal{B}$ accepts in $([q], v) = ([\delta_{\lambda(v_0)}(q_0)], v_0)$. At the same time, the original automaton goes from the initial configuration to $\big(\delta_{\lambda(v_0)}(q_0), v_0 + d(\delta_{\lambda(v_0)}(q_0))\big)$ and by Claim 1 it continues to $(\widehat{q}, \widehat{v})$, where it accepts.

Finally, if the computation of $\mathcal{B}$ ends in a configuration $(\overrightarrow{p}, v)$, for some $p \in Q$ and $v \in V$, then $-d(p) \in D_{\lambda(v)}$, because the direction $d'(\overrightarrow{p}) = d(p)$ was used at the last step. Therefore, by assertion (ii), the only reason why the computation ends in $(\overrightarrow{p}, v)$ could be that $(p, \lambda(v)) \in F$. Because Claim 1

states that the computation of $\mathcal{A}$ beginning in (p, v) accepts in $(\widehat{q}, \widehat{v})$, this computation must consist of a single configuration $(p, v) = (\widehat{q}, \widehat{v})$, which shows that the configuration in which $\mathcal{B}$ rejects is $(\overrightarrow{\widehat{q}}, \widehat{v})$. For the sake of contradiction, suppose that $\mathcal{A}$ accepts the graph in $(\widehat{q}, \widehat{v})$, and $(\widehat{q}, \widehat{v}) \neq (q_0, v_0)$. Then, by Claim 1, this accepting computation would be greater than the computation of $\mathcal{A}$ beginning in (p, v). However, this is impossible, since the computation beginning in (p, v) consists of a single configuration $(\widehat{q}, \widehat{v})$, which is the last configuration of the accepting computation beginning in (q_0, v_0), and, by definition, a computation cannot be greater than its suffix. Therefore, in this case, either $(\widehat{q}, \widehat{v}) = (q_0, v_0)$ or the computation of $\mathcal{A}$ beginning in (q_0, v_0) does not accept in $(\widehat{q}, \widehat{v})$.

It has been verified that in each case one of the following two situations arises: (i) the computation of $\mathcal{B}$ accepts in the configuration $([\delta_{\lambda(v_0)}(q_0)], v_0)$, $(\widehat{q}, \widehat{v}) \neq (q_0, v_0)$, while $\mathcal{A}$ accepts in the configuration $(\widehat{q}, \widehat{v})$; (ii) the computation of $\mathcal{B}$ rejects in $(\overrightarrow{\widehat{q}}, \widehat{v})$, and either $(\widehat{q}, \widehat{v}) = (q_0, v_0)$ or $\mathcal{A}$ does not accept in $(\widehat{q}, \widehat{v})$. This concludes the proof of the lemma. □

7 Proofs of the theorems

With Lemma 7 established, proofs of Theorems 1 and 2 shall now be obtained by building upon the construction presented in the lemma.

In the first theorem, an arbitrary direction-determinate GWA $\mathcal{A}$ is transformed to a returning direction-determinate GWA, which operates as follows: first it simulates $\mathcal{A}$ until it accepts, and then backtracks the accepting computation of $\mathcal{A}$ to its initial configuration, using the reversible automaton constructed from $\mathcal{A}$ according to Lemma 7. If $\mathcal{A}$ rejects or loops, the constructed automaton will reject or loop in the same way, as it will never reach the backtracking stage.

Proof of Theorem 1. Consider any direction-determinate automaton $\mathcal{A} = (Q, q_0, \delta, F)$ and, without loss of generality, assume that this automaton is without inaccessible transitions. Using Lemma 7, construct a reversible automaton $\mathcal{B} = (\overrightarrow{Q} \cup [Q], \delta', F')$ without an initial state. Define a new direction-determinate returning automaton $\mathcal{C} = (Q \cup \overrightarrow{Q} \cup [Q], q_0, \delta'', F'')$, where the directions of the states are the same as in $\mathcal{A}$ and $\mathcal{B}$.

The new automaton $\mathcal{C}$ begins its computation by simulating the original automaton $\mathcal{A}$ using the states from Q; for that purpose, its transition function δ'' is defined as δ for states from Q:

$$\delta''_a(q) = \delta_a(q), \qquad \text{if } \delta_a(q) \text{ is defined.}$$

By this definition, if $\mathcal{A}$ loops, then $\mathcal{C}$ loops as well. If $\mathcal{A}$ rejects, then so does $\mathcal{C}$. But if the simulated automaton $\mathcal{A}$ is about to accept in a configuration

$(\widehat{q},\widehat{v})$, then $\mathcal{C}$ instead switches to simulating the reversible automaton $\mathcal{B}$ using the states from $\overrightarrow{Q} \cup [Q]$. The switch is done by a transition of the form

$$\delta''_a(\widehat{q}) = [\widehat{q}], \qquad \text{for all } (\widehat{q},a) \in F \setminus (\{q_0\} \times \Sigma_0).$$

By this transition, the automaton $\mathcal{C}$ switches from the configuration $(\widehat{q},\widehat{v})$ to the configuration $([\widehat{q}],\widehat{v}-d(\widehat{q}))$, from whence it operates as $\mathcal{B}$. Then, according to Lemma 7, the automaton $\mathcal{B}$ accepts in the configuration $([\delta_a(q_0)],v_0)$, as it is known that the original automaton $\mathcal{A}$ accepts this graph in $(\widehat{q},\widehat{v})$. It remains to set the transitions and acceptance conditions of $\mathcal{C}$ to simulate $\mathcal{B}$. The transition function δ'' is defined as δ' for states from $\overrightarrow{Q} \cup [Q]$:

$$\begin{aligned} \delta''_a([q]) &= \delta'([q]), && \text{for } [q] \in [Q], \\ \delta''_a(\overrightarrow{q}) &= \delta'(\overrightarrow{q}), && \text{for } \overrightarrow{q} \in \overrightarrow{Q}. \end{aligned}$$

The constructed automaton $\mathcal{C}$ is set to accept whenever the simulated $\mathcal{B}$ accepts, as well as in the special case of $\mathcal{A}$ accepting in its initial configuration:

$$F'' = F' \cup \{\, (q_0,a_0) \mid a_0 \in \Sigma_0,\ (q_0,a_0) \in F \,\}.$$

□

The second theorem, which asserts that a returning direction-determinate GWA can be simulated by a strongly reversible GWA, is proved as follows. Naturally, a given automaton $\mathcal{A}$ is simulated by a reversible automaton $\mathcal{B}$ of Lemma 7. However, $\mathcal{B}$ is defined without an initial state, and its behaviour in the beginning of the computation has to be defined. If $\mathcal{A}$ has a unique accepting state, then it can only accept in a single configuration, and $\mathcal{B}$ can directly proceed to backtrack the tree of computations leading to this configuration. In the case of $\mathcal{A}$ with multiple accepting states, the automaton $\mathcal{B}$ has multiple trees of accepting computations to backtrack, and one should modify $\mathcal{B}$, so that it considers these trees one by one. The below proof fills out all details of this construction, necessary to satisfy all conditions in the definition of strongly reversible automata, and proves that the resulting automaton recognizes the same language as $\mathcal{A}$.

Proof of Theorem 2. Let $\mathcal{A} = (Q, q_0, \delta, F)$ be a direction-determinate returning graph-walking automaton. Without loss of generality, it can be assumed that this automaton has no inaccessible transitions. Let $\mathcal{B} = (\overrightarrow{Q} \cup [Q], \delta', F')$ be the reversible automaton without the initial state constructed from $\mathcal{A}$ according to Lemma 7. Assume any linear ordering on Q. For every $a_0 \in \Sigma_0$, denote by F_{a_0} the set $\{\, p \in Q \mid (p,a_0) \in F,\ -d(p) \in D_{a_0} \,\}$ of all states of $\mathcal{A}$, which are accepting on initial nodes labelled by a_0, excluding the case of $p = q_0$ with $-d(p) \notin D_{a_0}$. Define a new direction-determinate graph-walking automaton $\mathcal{C}$ by modifying $\mathcal{B}$ as follows:

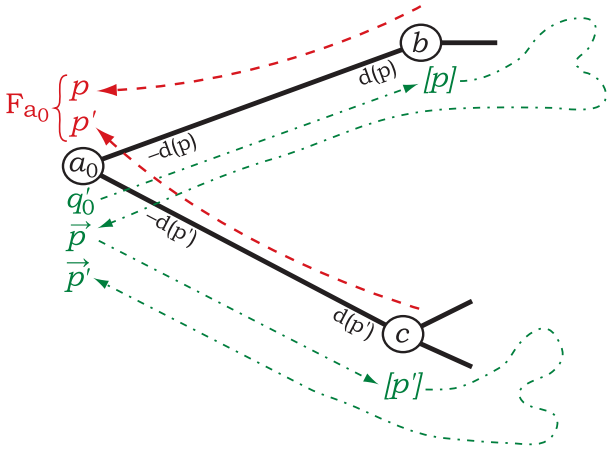

Figure 8: A strongly reversible GWA $\mathcal{C}$ searching through the accepting computations of a GWA $\mathcal{A}$.

- add a new separated initial state q_0',
- for every $a_0 \in \Sigma_0$ such that $(q_0, a_0) \in F$, add a new acceptance condition (q_0', a_0) to F',
- for every $a_0 \in \Sigma_0$ such that $(q_0, a_0) \notin F$ and $F_{a_0} \neq \varnothing$, add a new transition $\delta'_{a_0}(q_0') = [\min F_{a_0}]$,
- for every $a_0 \in \Sigma_0$ and every $p \in F_{a_0}$, which is not greatest in F_{a_0}, add a new transition $\delta'_{a_0}(\overrightarrow{p}) = [\mathrm{next}_{F_{a_0}}(p)]$.

The appropriate directions for each of these rules belong to D_{a_0} by the definition of F_{a_0}, and none of these transitions were previously defined in $\mathcal{B}$ according to claim (ii) of Lemma 7. Figure 8 illustrates how the constructed automaton $\mathcal{C}$ considers the trees of all computations of $\mathcal{A}$ leading to its accepting configurations, one by one.

All mappings δ_a, for $a \in \Sigma \setminus \Sigma_0$, in the automaton $\mathcal{C}$ are the same as in $\mathcal{B}$, and mappings δ_{a_0}, for $a \in \Sigma_0$, remain injective, since no states $[r]$, with $r \in F_{a_0}$, belonged to the range of δ_{a_0} in $\mathcal{B}$ by the second assertion of Lemma 7(iv). The automaton $\mathcal{C}$ is trivially returning, and for each $a_0 \in \Sigma_0$ there exists at most one state $s \in \overrightarrow{Q} \cup [Q] \cup \{q_0'\}$ such that $(s, a_0) \in F'$, because such a state existed in $\mathcal{B}$ only if $\delta_{a_0}(q_0)$ was defined. This verifies that $\mathcal{C}$ is reversible.

In order to verify that $\mathcal{C}$ is strongly reversible, let first $a \in \Sigma \setminus \Sigma_0$. Then $\delta'_a([q])$ is defined if and only if $-d'([q]) \in D_a$, by claim (i) of Lemma 7, and $\delta'_a(\overrightarrow{p})$ is defined if and only if $-d'(\overrightarrow{p}) \in D_a$, by claim (ii) of Lemma 7 (taking into account that $\mathcal{A}$ is returning). This shows that the domain of δ'_a is as required by the first condition of the definition of strong reversibility. In the same way, claims (iii) and (iv) of Lemma 7 imply that the range of δ'_a satisfies this condition as well.

For $a_0 \in \Sigma_0$, the rejecting state $q^{a_0}_{\mathrm{rej}}$ of $\mathcal{C}$ will be defined if and only if $F_{a_0} \neq \varnothing$ or $(q_0, a_0) \notin F$. If $F_{a_0} \neq \varnothing$, then define $q^{a_0}_{\mathrm{rej}}$ as $\overrightarrow{\max F_{a_0}}$. If $F_{a_0} = \varnothing$ and $(q_0, a_0) \notin F$, then define $q^{a_0}_{\mathrm{rej}}$ as q'_0. It remains to verify the last condition in the definition of strong reversibility. For $q \in Q$ such that $[q] \neq q^{a_0}_{\mathrm{acc}}$, $\delta'_{a_0}([q])$ is defined if and only if $-d([q]) \in D_{a_0}$, due to Lemma 7(i). For $p \in Q$ such that $\overrightarrow{p} \neq q^{a_0}_{\mathrm{rej}}$, according to Lemma 7(ii), $\delta'_{a_0}(\overrightarrow{p})$ is defined in $\mathcal{B}$ if and only if $-d(\overrightarrow{p}) \in D_{a_0}$ and $(p, a_0) \notin F$, and it is additionally defined in $\mathcal{C}$ if and only if $p \in F_{a_0}$ and $p \neq \max F_{a_0}$. Altogether, $\delta'_{a_0}(\overrightarrow{p})$ is defined in $\mathcal{C}$ if and only if $-d(\overrightarrow{p}) \in D_{a_0}$ and $p \neq \max F_{a_0}$, where the latter condition can be removed, since it follows from the assumption $\overrightarrow{p} \neq q^{a_0}_{\mathrm{rej}}$. Finally, it has to be proved that $\delta'_{a_0}(q'_0)$ is always defined, provided that $q'_0 \notin \{q^{a_0}_{\mathrm{acc}}, q^{a_0}_{\mathrm{rej}}\}$. According to the definition of $\mathcal{C}$, $\delta'_{a_0}(q'_0)$ is not defined only if $(q_0, a_0) \in F$ or $F_{a_0} = \varnothing$. However, the former condition implies that $q'_0 = q^{a_0}_{\mathrm{acc}}$, and if it is not true, that is, if $(q_0, a_0) \notin F$, then $F_{a_0} = \varnothing$ implies $q'_0 = q^{a_0}_{\mathrm{rej}}$. Thus both conditions contradict the assumption on q'_0.

In order to prove that the automaton $\mathcal{C}$ accepts the same graphs as $\mathcal{A}$, observe first that $\mathcal{C}$ accepts immediately in the initial configuration (q'_0, v_0) if and only if $\mathcal{A}$ accepts immediately in the initial configuration (q_0, v_0). Assume that they do not accept immediately. Then $\mathcal{A}$ can only accept in one of the configurations (q, v_0), with $q \in F_{\lambda(v_0)} \setminus \{q_0\}$. If $\mathcal{A}$ rejects because of $F_{\lambda(v_0)} = \varnothing$, then $\mathcal{C}$ rejects immediately in the initial configuration. Otherwise, the computation of $\mathcal{C}$ begins by moving to the configuration $([\min F_{\lambda(v_0)}], v_0 - d(\min F_{\lambda(v_0)}))$. By the main claim of Lemma 7, the computation continues by successively considering all states $p \in F_{\lambda(v_0)}$, according to the ordering of states; for each of these states it begins in the configuration $([p], v_0 - d(p))$, accepts if $p \neq q_0$ and $\mathcal{A}$ accepts in (p, v_0), and otherwise goes to $(\overrightarrow{p}, v_0)$, from where it switches to the next state from the set $F_{\lambda(v_0)}$ by continuing to the configuration $([\mathrm{next}_{F_{\lambda(v_0)}}(p)], v_0 - d(\mathrm{next}_{F_{\lambda(v_0)}}(p)))$ using the new transitions added to $\mathcal{B}$. If $\mathcal{A}$ does not accept in any of the configurations (p, v_0) with $p \neq q_0$, then $\mathcal{C}$ eventually rejects in the configuration $(\overrightarrow{\max F_{\lambda(v_0)}}, v_0)$. □

8 Application to various types of automata

The results on transforming graph-walking automata to several special forms presented in Sections 4–6 apply to GWAs with any set of directions and over any signature, so that in each case, each transformation preserves the set of accepted graphs. Therefore, these results also apply to each of the well-known models of computation presented in Sections 2–3 as graph-walking automata. The aim of this section is to revisit all those models and apply the results of Section 4 to each of them.

Proposition 1. *An arbitrary n-state 2DFA can be transformed to a $2n$-state direction-determinate 2DFA, to a $(2n+1)$-state returning direction-determinate 2DFA, and to a $(4n+3)$-state strongly reversible 2DFA.*

Indeed, for 2DFAs, the set of directions $D = \{-1, +1\}$ is a two-element set, and hence the transformation to direction-determinate duplicates the number of states. In order to make a direction-determinate 2DFA returning, it is sufficient to add one extra state, in which the automaton will move the head to the left-end marker after it decides to accept. Applying Theorem 2 to the resulting automaton gives the promised strongly reversible 2DFA with $4n + 3$ states.

In the case of 2DFAs, Theorem 2 is essentially a generalization of the construction by Kondacs and Watrous [28] from 1DFAs to direction-determinate 2DFAs. The transformation of an n-state 2DFA to a 2DFA with $4n + \text{const}$ states that halts on every input, presented by Geffert et al. [17], most likely results in the same reversible automaton as constructed in Proposition 1, but both main steps of the construction are amalgamated into one. Thus, the two-step transformation proving Proposition 1 explains the construction given by Geffert et al. [17].

In the special case of 2DFAs over a one-letter alphabet, the above construction can be implemented using fewer states, as follows.

Proposition 2. *An n-state 2DFA over a one-letter input alphabet $\Sigma = \{a\}$ can be transformed to an $(n + 1)$-state direction-determinate 2DFA, to an $(n + 2)$-state returning direction-determinate 2DFA, and to a $(2n + 5)$-state strongly reversible 2DFA.*

The improvement lies with the efficient transformation to direction-determinate, which follows from a transformation of an n-state 2DFA over a one-letter alphabet to an equivalent $(n+1)$-state sweeping 2DFA, due to the authors [29, Thm. 2]. Then, one extra state is used to return to the left-end marker, and Theorem 2 is applied as in the proof of Proposition 1.

It is possible to improve the construction in Proposition 1 to yield only $4n + 1$ states, and the construction in Proposition 2 can similarly produce $2n+3$ states [30]. This is achieved essentially by integrating the construction for parking the head within the construction of Theorem 2, specialized for two-way automata.

Turning to tree-walking automata, Muscholl et al. [37] proved that an n-state TWA can be transformed to a halting TWA with $O(n^2)$ states, using another implementation of the general method described by Sipser [42]. This result can be now improved as follows.

Proposition 3. *Any n-state TWA over k-ary trees can be transformed to a $2kn$-state direction-determinate TWA, to a $(2kn + k)$-state returning direction-determinate TWA, and to a $(4kn + 2k + 1)$-state strongly reversible TWA.*

Here the transformation to direction-determinate multiplies the number of states by $|D| = 2k$. Parking the head after acceptance generally requires only one extra state, in which the automaton will go up to the root. However,

in order to keep the resulting automaton direction-determinate, one has to use k extra states $q^1_{\text{return}}, \ldots, q^k_{\text{return}}$ with $d(q^i_{\text{return}}) = -i$. Reversibility is ensured by Theorem 2, which produces $2(2kn + k) + 1$ states, as stated.

Proposition 4. *An n-state 4DFA can be transformed to a $4n$-state direction-determinate 4DFA, to a $(4n+4)$-state returning direction-determinate 4DFA, and to a $(8n + 9)$-state strongly reversible 4DFA.*

Transformation to direction-determinate multiplies the number of states by $|D| = 4$. In order to return to the initial node after acceptance, it is sufficient to use four states: one state to move in the direction $(-1, 0)$ up to a marker $\top$, another one to make one step from that marker in the direction $(+1, 0)$, one more state to move in the direction $(0, -1)$ to the initial node, and finally, an extra state to move in the direction $(0, +1)$, in case the acceptance decision is made at a left marker $\vdash$. If the acceptance decision is made at a right marker $\dashv$, then the existing state for moving in the direction $(0, -1)$ can be used to get out of that node. It remains to apply Theorem 2 to the resulting automaton.

The next model are the multi-head automata. Their reversibility was first investigated by Morita [36], who defined reversible automata as those satisfying the first condition of Definition 4 in this paper, but not its second condition (the one on acceptance only in the initial node and in a unique state). Morita [36] presented the construction for reversing a reversible automaton, as in Lemma 5 in this paper, as well as a construction for transforming a "weak" reversible automaton (without the conditions on acceptance assumed in this paper) to a reversible automaton in the sense of this paper, which furthermore rejects only in the initial node (as in Definition 6(iv) in this paper). However, the general problem of transforming any deterministic k-head 2DFA to a reversible multi-head 2DFA recognizing the same language was left open. In a follow-up paper, Axelsen [3] has shown that the family of reversible multi-head 2DFAs is equal in power to the reversible logarithmic space, and hence, by the results of Lange et al. [33], to the deterministic logarithmic space. These results imply that reversible multi-head 2DFAs are equivalent in power to deterministic multi-head 2DFAs, though it remained unclear, whether transforming a multi-head 2DFA to an equivalent reversible machine may require using any extra heads.

This paper contributes a stronger result, that for every $k \geqslant 1$, the reversible k-head 2DFAs recognize the same languages as the deterministic k-head 2DFAs.

Proposition 5. *Any n-state k-head 2DFA can be transformed to a $(3^k-1)n$-state direction-determinate k-head 2DFA, to a $((3^k-1)n+k)$-state returning direction-determinate k-head 2DFA, and to a $(2(3^k - 1)n + 2k + 1)$-state strongly reversible k-head 2DFA.*

Since the set of directions $D = \{-1, 0, +1\}^k \setminus \{0\}^k$ has cardinality $|D| = 3^k - 1$, the transformation to direction-determinate incurs a $(3^k - 1)$-times blowup. The resulting automaton is then modified to return to the initial node after acceptance by parking its k heads to the left-end marker one by one, using k states $q_{\text{park } 1}, \ldots, q_{\text{park } k}$ with $d(q_{\text{park } i}) = (0, \ldots, 0, -1, 0, \ldots, 0)$, with -1 as the i-th component. Applying Theorem 2 to the resulting automaton with $(3^k-1)n+k$ states yields the desired strongly reversible automaton.

The case of multi-tape 2DFAs is handled identically.

Proposition 6. *An n-state k-tape 2DFA can be transformed to equivalent direction-determinate, returning and direction-determinate, and strongly reversible k-tape 2DFAs with the same number of states as in Proposition 5.*

Proposition 7. *An n-state k-pebble 2DFA can be transformed to a $(2k+2)n$-state direction-determinate k-pebble 2DFA, to a $((2k + 2)n + k + 2)$-state returning direction-determinate k-pebble 2DFA, and to a $((4k+4)n+2k+5)$-state strongly reversible k-pebble 2DFA.*

For k-pebble 2DFAs, the set of directions $D = \{-1, +1, \downarrow 1, \uparrow 1, \ldots, \downarrow k, \uparrow k\}$ has cardinality $2k + 2$, which explains the size of the direction-determinate automaton. The transformation to a returning direction-determinate automaton requires $k + 2$ extra states, because in order to return to the initial node, one must sweep through the entire tape to pick up all the pebbles.

Proposition 8. *An n-state Turing machine operated in marked space $s(n)$ using an m-symbol work alphabet (including the blank symbol, but not including the end-markers) can be transformed to an $(m^2-m+4)n$-state direction-determinate Turing machine operating in the same space using the same alphabet, to an $((m^2 - m + 4)n + m + 2)$-state returning direction-determinate Turing machine of the same kind, and to a $(2(m^2 - m + 4)n + 2m + 5)$-state strongly reversible Turing machine that again works in the same marked space $s(n)$ using the same m-symbol work alphabet.*

Here $|D| = m^2-m+4$, because there are $m(m-1)$ directions for rewriting letters and 4 directions for moving the heads. Once a direction-determinate automaton with $|D| \cdot n$ states is obtained, it can be made returning by employing $1 + (m - 1) + 1 + 1$ extra states to (a) move the head on the work tape to the right, (b) scan the work tape from right to left, rewriting each non-blank symbol with a blank, and then (c) move the head on the input tape to the left.

The list of such results can be continued further, by representing various models of computation with a bounded graph of memory configurations as graph-walking automata, and then applying the results of this paper.

References

[1] S. Abramsky, "A structural approach to reversible computation", *Theoretical Computer Science*, 347:3 (2005), 441–464.

[2] A. V. Aho, J. D. Ullman, "Translations on a context free grammar", *Information and Control*, 19:5 (1971), 439–475.

[3] H. B. Axelsen, "Reversible multi-head finite automata characterize reversible logarithmic space", *Language and Automata Theory and Applications* (LATA 2012, A Coruña, Spain, 5–9 March 2012), LNCS 7183, 95–105.

[4] C. H. Bennett, "Logical reversibility of computation", *IBM Journal of Research and Development*, 17:6 (1973), 525–532.

[5] C. H. Bennett, "The thermodynamics of computation—a review", *International Journal of Theoretical Physics*, 21:12 (1982), 905–940.

[6] C. H. Bennett, "Time/space trade-offs for reversible computation", *SIAM Journal on Computing*, 81 (1989), 766–776.

[7] M. Blum, C. Hewitt, "Automata on a 2-dimensional tape", *8th Annual Symposium on Switching and Automata Theory* (*SWAT 1967*, Austin, Texas, USA, 18–20 October 1967), 155–160.

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

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

[10] M. Bojańczyk, M. Samuelides, T. Schwentick, L. Segoufin, "Expressive power of pebble automata", (ICALP 2006, Venice, Italy, 9–16 July 2006), vol. 1, LNCS 4051, 157–168.

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

[12] H. Buhrman, J. Tromp, P. Vitányi, "Time and space bounds for reversible simulation", *Journal of Physics A: Mathematical and General*, 34:35 (2001), 6821–6830.

[13] B. Courcelle, "Graph rewriting: An algebraic and logic approach", *Handbook of Theoretical Computer Science, Volume B*, 1990, 193–242.

[14] P. Crescenzi, C. H. Papadimitriou, "Reversible simulation of space-bounded computations", *Theoretical Computer Science*, 143:1 (1995), 159–165.

[15] J. Engelfriet, H. J. Hoogeboom, "Tree-walking pebble automata", *Jewels are Forever, Contributions on Theoretical Computer Science in Honor of Arto Salomaa*, 1999, 72–83.

[16] J. Engelfriet, H. J. Hoogeboom, "Automata with nested pebbles capture first-order logic with transitive closure", *Logical Methods in Computer Science* 3:2–3 (2007), 1–27.

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

[18] N. Globerman, D. Harel, "Complexity results for two-way and multi-pebble automata and their logics", *Theoretical Computer Science*, 169:2 (1996), 161–184.

[19] T. Harju, J. Karhumäki, "The equivalence problem of multitape finite automata", *Theoretical Computer Science*, 78:2 (1991), 347–355.

[20] P.-C. Héam, "A lower bound for reversible automata" *RAIRO Informatique Théorique et Applications*, 34:5 (2000), 331–341.

[21] M. Holzer, M. Kutrib, A. Malcher, "Complexity of multi-head finite automata: Origins and directions", *Theoretical Computer Science*, 412:1–2 (2011), 83–96.

[22] J. E. Hopcroft, J. D. Ullman, "Some results on tape bounded Turing machines", *Journal of the ACM*, 16 (1967), 168–177.

[23] P. Jančar, F. Mráz, M. Plátek, J. Vogel, "Restarting automata", *Fundamentals of Computation Theory* (FCT 1995, Dresden, Germany, 22–25 August 1995), LNCS 965, 283–292.

[24] T. Kamimura, G. Slutzki, "Parallel two-way automata on directed ordered acyclic graphs", *Information and Control*, 49:1 (1981), 10–51.

[25] C. A. Kapoutsis, R. Královic, T. Mömke, "Size complexity of rotating and sweeping automata", *Journal of Computer and System Sciences*, 78:2 (2012), 537–558.

[26] J. Kari, "Reversible cellular automata", *Developments in Language Theory* (DLT 2005, Palermo, Italy, 4–8 July 2005), LNCS 3572, 57–68.

[27] J. Kari, V. Salo, "A survey on picture-walking automata", *Algebraic Foundations in Computer Science*, LNCS 7020, 2011, 183–213.

[28] 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.

[29] M. Kunc, A. Okhotin, "Describing periodicity in two-way deterministic finite automata using transformation semigroups", *Developments in Language Theory* (DLT 2011, Milan, Italy, 19–22 July 2011), LNCS 6795, 324–336.

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

[31] M. Kutrib, A. Malcher, "Reversible pushdown automata", *Journal of Computer and System Sciences*, 78:6 (2012), 1814–1827.

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

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

[34] Y. Lecerf, "Machines de Turing réversibles", *Comptes Rendus de l'Académie des Sciences*, 257 (1963), 2597–2600.

[35] S. Lombardy, "On the construction of reversible automata for reversible languages", *Automata, Languages and Programming* (ICALP 2002, Málaga, Spain, 8–13 July 2002), LNCS 2380, 170–182.

[36] K. Morita, "Two-way reversible multi-head finite automata", *Fundamenta Informaticae*, 110:1–4 (2011), 241–254.

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

[38] P. Panaite, A. Pelc, "Exploring unknown undirected graphs", *Journal of Algorithms*, 33:2 (1999), 281–295.

[39] J.-E. Pin, "On the languages accepted by finite reversible automata", *Automata, Languages and Programming* (ICALP 1987, Karlsruhe, Germany, 13–17 July 1987), LNCS 267, 237–249.

[40] M. O. Rabin, D. Scott, "Finite automata and their decision problems", *IBM Journal of Research and Development*, 3:2 (1959), 114–125.

[41] D. Ranjan, R. Chang, J. Hartmanis, "Space bounded computations: review and new separation results", *Theoretical Computer Science*, 80:2 (1991), 289–302.

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

[43] M. Sipser, "Lower bounds on the size of sweeping automata", *Journal of Computer and System Sciences*, 21:2 (1980), 195–202.

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

[45] T. Toffoli, N. H. Margolus, "Invertible cellular automata: A review", *Physica D: Nonlinear Phenomena*, 45:1–3 (1990), 229–253.

Turku
Centre *for*
Computer
Science

Lemminkäisenkatu 14 A, 20520 Turku, Finland | www.tucs.fi

University of Turku
- Department of Information Technology
- Department of Mathematical Sciences

Åbo Akademi University
- Department of Computer Science
- Institute for Advanced Management Systems Research

Turku School of Economics and Business Administration
- Institute of Information Systems Sciences

ISBN 978-952-12-2732-5
ISSN 1239-1891