Information and Computation 275 (2020) 104631

Contents lists available at ScienceDirect

Information and Computation

www.elsevier.com/locate/yinco

Reversibility of computations in graph-walking automata ☆

Michal Kunc [a], Alexander Okhotin [b,*]

[a] Department of Mathematics and Statistics, Masaryk University, Kotlářská 2, Brno 611 37, Czech Republic
[b] Department of Mathematics and Computer Science, St. Petersburg State University, 14th Line V.O., 29, Saint Petersburg 199178, Russia

ARTICLE INFO

Article history:
Received 27 March 2019
Received in revised form 21 August 2020
Accepted 30 September 2020
Available online 9 October 2020

Keywords:
Graph-walking automata
Tree-walking automata
Finite automata
Reversible computation
Halting

ABSTRACT

Graph-walking automata (GWA) are finite-state devices that traverse graphs given as an input by following their edges; they have been studied both as a theoretical notion and as a model of pathfinding in robotics. If a graph is regarded as the set of memory configurations of a certain abstract machine, then various families of devices can be described as GWA: such are two-way finite automata, their multi-head and multi-tape variants, tree-walking automata and their extension with pebbles, picture-walking automata, space-bounded Turing machines, etc. This paper defines a transformation of an arbitrary deterministic GWA to a reversible GWA. This is done with a linear blow-up in the number of states, where the constant factor depends on the degree of the graphs being traversed. The construction directly applies to all basic models representable as GWA, and, in particular, subsumes numerous existing results for making individual models halt on every input.

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 [4], as irreversible computations incur energy dissipation [35]. It is known from Lecerf [37] and Bennett [3] 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 [5], Crescenzi and Papadimitriou [13], Li et al. [38], Lange et al. [36] and Buhrman et al. [11]. Reversibility in cellular automata also has a long history of research, presented in surveys by Toffoli and Margolus [51] and by Kari [29].

High-level programming languages for writing reversible programs are being designed and studied. The first such language, *Janus*, was designed by Lutz and Derby in 1982—see an unpublished note by Lutz [40]—and later studied in detail by Yokoyama et al. [52]. By now, there is a substantial body of literature on reversible programming.

In the domain of finite automata, the reversible subclass of one-way deterministic finite automata (1DFA) defines a proper subfamily of the regular languages, which was studied, in particular, by Pin [45], by Héam [22] and by Lombardy [39]. On the other hand, every regular language is recognized by a reversible two-way finite automaton (2DFA): as shown by Kondacs and Watrous [31], every n-state 1DFA can be simulated by a $2n$-state reversible 2DFA.

☆ This work was presented at the MFCS 2013 conference held in Klosterneuburg, Austria on 26–30 August 2013, and its extended abstract appeared in the conference proceedings: K. Chatterjee, J. Sgall (Eds.), *Mathematical Foundations of Computer Science*, LNCS 8087, pp. 595–606.

* Corresponding author.
E-mail addresses: kunc@math.muni.cz (M. Kunc), alexander.okhotin@spbu.ru (A. Okhotin).

https://doi.org/10.1016/j.ic.2020.104631
0890-5401/

One of the most evident consequences of reversibility is that a reversible device halts on every input, as long as the set of available memory configurations on that input is finite. 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 [24]. 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 [48], and his approach essentially means constructing a reversible machine, although reversibility was not considered as such. In particular, Sipser [48] sketched a transformation of an n-state 2DFA to an $O(n^2)$-state halting 2DFA, which *remembers two states* of the original 2DFA, and which is actually reversible. Sipser [48] also mentioned the possibility of an improved transformation, in which the simulating halting 2DFA—actually reversible as well—*remembers a state and a symbol of the input alphabet*, and thus has $O(n)$ states, where the multiplicative factor depends upon the size of the alphabet. The fact that Sipser's idea produces reversible automata was noticed by Lange et al. [36] and used to establish the equivalence of deterministic space $O(s(\ell))$ to reversible space $O(s(\ell))$. Next, Kondacs and Watrous [31] in turn used the construction of Lange et al. [36] as a model for a new simulation of 1DFA by reversible 2DFA: their unfamiliarity with Sipser's [48] paper led them to a more optimal construction that uses only $2n$ states, that is, *remembers one state and one bit*. A similar construction for making a 2DFA halt on every input was later invented by Geffert et al. [19], who have amalgamated an independently rediscovered method of Kondacs and Watrous [31] with a pre-processing step. For tree-walking automata (TWA), a variant of Sipser's [48] construction was used by Muscholl et al. [44] 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 2DFA 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 kind 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 considers what is apparently the most general definition covering all such computational models: the notion of *graph-walking automata* (GWA). In this setting, the space of memory configurations is regarded as an input graph, where each node is a memory configuration, labelled by the data observed by the machine in this position, whereas edges are labelled with elementary operations on the memory. Then, a graph-walking automaton traverses a given input graph using a finite-state control and a transition function with a finite domain, which instructs the automaton to move along a certain edge, based on the current state and the label of the currently observed node. 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 is determined by the input data via a function mapping an input object to a graph. For graph-walking automata, this means that *input graphs must be finite* (even though the definition of automata in itself allows the graphs to be infinite). Though, in general, reversible computation is possible in devices with unbounded memory, the methods investigated in this paper rely on this limitation.
3. The machine can test whether the current memory configuration is the initial configuration. For a graph-walking automaton, this means that the initial node, where the computation begins, has a distinguished label.

Another motivation for graph-walking automata comes from the area of robotics, where, under different names, they have been used as a mathematical model for navigation in a discrete environment. The question of whether there exists a finite automaton that can traverse every undirected graph by following its edges was reportedly first proposed by Michael Rabin in his 1967 public lecture [17]. The conjecture was settled by Budach [10], who proved that for every graph-walking automaton there is a planar graph that it cannot fully traverse; a simple proof of this result was later presented by Fraigniaud et al. [17]. Knowing that finite memory is not sufficient, it is natural to ask how much memory suffices to traverse a graph; in particular, Fraigniaud et al. [17] report on the early work on the subject and establish some new lower bounds. Designing memory-efficient graph algorithms is an active research area, with some results on the space complexity of searching recently obtained by Elmasry et al. [14].

The paper begins with a definition of graph-walking automata designed according to the above principles 1–3. The definition is given in Section 2, where it is illustrated by presenting 2DFA and TWA as simple special cases of GWA. Further machine models, including multi-head 2DFA, 2DFA with pebbles and space-bounded Turing machines, are represented as GWA in the subsequent Section 3.

The definition of reversibility in GWA, developed in the next Section 4, is based upon several conditions. First, consider a subclass of GWA, 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 GWA are *returning automata*, which may accept only in the initial memory configuration, or, in other words, must clear the memory before accepting. 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 directly *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 main results of this paper are established in the next Sections 6–7: these are transformations from automata of the general form to returning automata, and from returning automata to reversible automata. Both transformations rely on the same effective construction, presented in Section 6, which generalizes the method of Kondacs and Watrous [31]; the origins of the latter can be traced to the general idea due to Sipser [48]. Both transformations of automata based on this single construction are defined and proved correct in Section 7; notably, they incur only a linear blow-up in the number of states, where the constant depends only on the number of directions, but not on the number of node labels.

In the subsequent Section 8, these general results are applied to each of the concrete models represented as GWA 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 [31], and also provides an improved construction and a rigorous argument for the result by Geffert et al. [19] on making a 2DFA halt on every input. In the case of a one-symbol 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 a halting automaton due to Muscholl et al. [44], which produced a TWA with $O(n^2)$ states.
- 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 improves the result by Morita [42], who constructed a reversible k-head 2DFA with $O(n)$ states, where the constant factor depends both on the size of the alphabet and on k.
- A deterministic Turing machine operating in marked space $s(\ell)$, as defined by Ranjan, Chang and Hartmanis [47, Sect. 5], can be transformed to a reversible Turing machine that uses the same marked space $s(\ell)$. This directly implies one of the versions of the result of Lange et al. [36] on the equivalence of deterministic space to reversible space, this time with no space overhead.

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

Finally, Section 9 establishes several further properties of computations of reversible automata produced by the constructions described in this paper, which could be useful in the future studies of this notion.

2. Graph-walking automata

Automata studied in this paper walk over undirected graphs, finite or infinite, 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 *unary minus* operation $(-\colon D \to D)$ that defines the *opposite direction* to each direction; it must satisfy $-(-d)=d$ 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.

Besides the direction labels on the edges' ends, the nodes are labelled with symbols from another set Σ. An automaton begins its computation on a graph in a unique node distinguished by a special *initial label* from a set of initial labels Σ_0. At every step, the automaton uses the label of the currently observed node, along with the current state, to decide in which direction to go. For each label $a \in \Sigma$, there is an associated set of available directions $D_a \subseteq D$, and, in every graph, each node labelled with a shall have exactly these directions. All these sets of symbols used to construct graphs form a *signature*, over which graphs are defined.

Definition 1. A *signature* is a quadruple $\mathcal{S} = (\Sigma, \Sigma_0, D, \langle D_a \rangle_{a \in \Sigma})$ that consists of

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

The notion of a signature generalizes the notion of an alphabet of a formal language. Given a signature, one can define graphs over that signature, which in turn generalize strings over an alphabet.

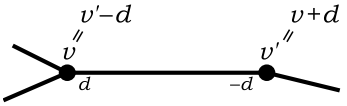

Fig. 1. An edge in a graph, with its ends marked by the directions d and $-d$.

Definition 2. A *graph* over a signature $\mathcal{S} = (\Sigma, \Sigma_0, D, \langle D_a \rangle_{a \in \Sigma})$ is a quadruple $(V, v_0, \lambda, +)$, where

- V is a set of *nodes*, not necessarily finite;
- $v_0 \in V$ is the *initial node*;
- a total mapping $\lambda \colon V \to \Sigma$ is such a labelling of nodes, that a node $v \in V$ is labelled with a symbol from Σ_0 if and only if $v = v_0$.
- a total mapping $+ \colon \{ (v, d) \mid v \in V,\ d \in D_{\lambda(v)} \} \to V$ defines, for a node $v \in V$, its neighbour in every direction d applicable in v, denoted by $v + d$. The applicable directions are determined by the label of v, that is, $v + d$ is defined for all d in $D_{\lambda(v)}$. This mapping defines the edges of the graph.
  Since every edge is traversable in both directions, the "plus" mapping must satisfy the following condition of invertibility by opposite directions: for every node $v \in V$ and direction $d \in D_{\lambda(v)}$, the opposite direction $-d$ belongs to $D_{\lambda(v+d)}$, and the following equality holds: $(v + d) + (-d) = v$. 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, $v - d$ shall be used to denote $v + (-d)$.

The representation of an edge by the mapping "+" is illustrated in Fig. 1, where $d \in D_{\lambda(v)}$, $-d \in D_{\lambda(v')}$, $v + d = v'$ and $v' + (-d) = v' - d = v$.

It is possible to define such a signature, that no graphs over that signature exist. For instance, if all initial labels have direction d, but no label has direction $-d$, then there is no way to connect the initial node to any nodes.

Definition 3. A *deterministic graph-walking automaton* over a signature $\mathcal{S} = (\Sigma, \Sigma_0, D, \langle D_a \rangle_{a \in \Sigma})$ is a quadruple $\mathcal{A} = (Q, q_0, \delta, F)$, in which

- Q is a finite set of *states*,
- $q_0 \in Q$ is the *initial state*,
- $F \subseteq Q \times \Sigma$ is a set of *acceptance conditions*, 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 the graph 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 uniquely defined (finite or infinite) sequence $(p_0, u_0), (p_1, u_1), \ldots$ of configurations, which satisfies the following conditions:

- $(p_0, u_0) = (q, v)$;
- if, for an element (p_i, u_i), the transition $\delta(p_i, \lambda(u_i))$ is defined, then the next element is $(p_{i+1}, u_i + d)$, where $\delta(p_i, \lambda(u_i)) = (p_{i+1}, d)$;
- if, for (p_i, u_i), the transition $\delta(p_i, \lambda(u_i))$ is not defined, then this is the last element of the sequence.

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 configuration (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) is accepting.

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 $\mathcal{S} = (\Sigma, \Sigma_0, D, \langle D_a \rangle_{a \in \Sigma})$, with directions $D = \{+1, -1\}$, labels $\Sigma = \Gamma \cup \{\vdash, \dashv\}$, initial labels $\Sigma_0 = \{\vdash\}$, with only one direction available at each end-marker ($D_\vdash = \{+1\}$, $D_\dashv = \{-1\}$) and with both directions ($D_a = \{+1, -1\}$) for each tape symbol $a \in \Gamma$.

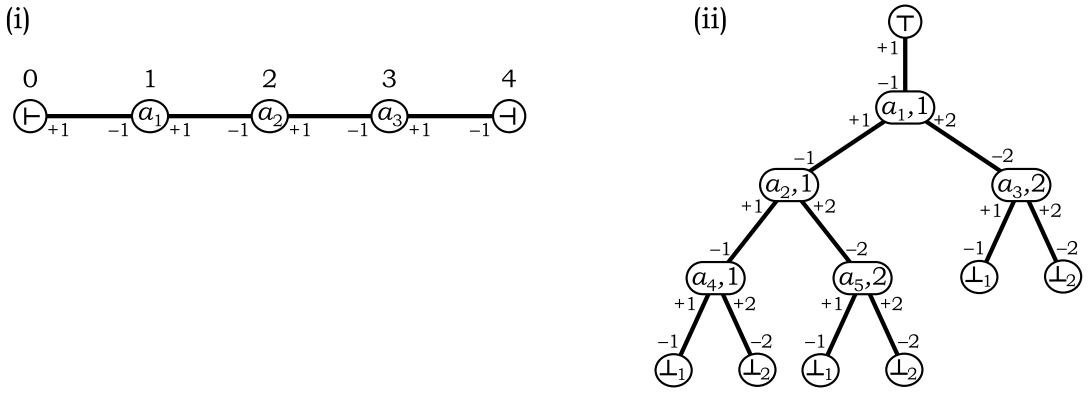

Fig. 2. Representing (i) input strings of 2DFA and (ii) input trees of TWA as graphs.

All finite connected graphs over this signature are path graphs, containing one instance of each end-marker and an arbitrary number of symbols from Γ in between. For an input string $w = a_1 \dots a_\ell$, with $\ell \geqslant 0$, the graph has the set of nodes $V = \{0, 1, \dots, \ell, \ell+1\}$ 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: $\lambda(0) = \vdash$, $\lambda(\ell+1) = \dashv$ and $\lambda(i) = a_i$ for all $i \in \{1, \dots, \ell\}$.

A sample graph over this signature is presented in Fig. 2(i).

Some authors allow stationary moves in 2DFA (and there is a folklore construction for eliminating such moves in a given 2DFA without increasing the number of states). This variant of the model is represented as a GWA as follows. In the signature, the set of directions becomes $D = \{-1, 0, +1\}$, with $-(-1) = +1$ and $-0 = 0$, and the direction 0 is made available on every label. In the graph, each node, as in Fig. 2(i), is further fitted with a loop by the direction 0.

Next, consider *tree-walking automata*, defined by Aho and Ullman [1, Sect. VI] as a model for traversing parse trees in formal grammars, and later studied by Bojańczyk and Colcombet [7,8]. 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 successors of the current node, or up to its predecessor. Furthermore, in any node except the root, the automaton is invested with the knowledge of whether this node is the first successor or the second successor [7]; without this knowledge, the expressive power of a tree-walking automaton would be severely limited [27]. The traversal of trees by these automata can be described using directions of the form "go down to the i-th successor" and the opposite "go up from the i-th successor to its predecessor".

In the notation of graph-walking automata, the knowledge of the index 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 a full set of downward directions. Furthermore, by analogy with the end-markers in 2DFA, 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.

Example 2. In order to represent a tree-walking automaton on k-ary trees over the alphabet Γ as a graph-walking automaton, let the set of directions be $D = \{+1, +2, \dots, +k, -1, -2, \dots, -k\}$, where each positive direction $+i$ points to the i-th successor and each negative direction $-i$ points from the i-th successor to its predecessor; accordingly, $-(+i) = -i$. The set of node labels is $\Sigma = \{\top, \bot_1, \dots, \bot_k\} \cup (\Gamma \times \{1, \dots, k\})$, where the top marker ($\top$), with $D_\top = \{+1\}$, is the label of the root node v_0 (that is, $\Sigma_0 = \{\top\}$), whereas each i-th bottom marker ($\bot_i$), with $D_{\bot_i} = \{-i\}$, is a label for leaves. Elements of the set $\Gamma \times \{1, \dots, k\}$ are used as labels of the internal nodes of the tree, where a label (a, i), with $a \in \Gamma$ and $i \in \{1, \dots, k\}$, means that the actual label of the node is a, and this node is the i-th successor of its predecessor; accordingly, the corresponding set of directions is $D_{(a,i)} = \{-i, +1, \dots, +k\}$. An example of a tree with $k = 2$ is given in Fig. 2(ii).

One can similarly define *unranked tree automata*, operating on trees in which every node may have an unbounded finite number of successors. Such an automaton has directions to go down to the first successor and to go down to the last successor, along with the opposite directions for going up from the first successor and going up from the last successor; furthermore, the 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

Definitions of 2DFA and TWA are explicitly stated in terms of walking over graphs, with certain assumptions on the form of the graphs being traversed. Several other commonly studied abstract automata are not defined in terms of graph walking, but can naturally be represented as such.

Consider any computational device recognizing input objects of any kind, which has a fixed number of internal states and employs auxiliary memory for data such as the positions of reading heads and the contents of any additional data

structures. Assume that for each fixed input, the total space 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, along 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.

For instance, in this terminology, *the space of memory configurations of a 2DFA* consists of all positions of the head on the tape, and there are two elementary operations on the memory, namely, incrementing and decrementing the position of the head, using the directions $+1 \in D$ and $-1 \in D$. The label in each position contains the symbol observed by the head.

By the same principle, one can represent many other models of computation as graph-walking automata. As the first model, consider 2DFA equipped with multiple reading heads, which can independently move over the same input tape: the *multi-head automata*, introduced by Rabin and Scott [46]. The research on such automata is described in a survey by Holzer et al. [23]. Their reversibility was recently investigated by Morita [41,42] and by Axelsen [2].

Example 3. A k-head two-way deterministic finite automaton with a tape alphabet Γ is described by a graph-walking automaton as follows. Each 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 $(d_1, \ldots, d_k)$, with $d_i \in \{-1, 0, +1\}$, indicates that each i-th head is to be moved in the direction d_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 available 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 prevent any heads from moving beyond either end-marker.

The automaton operates on graphs of the following form. For each input string $a_1 \ldots a_\ell \in \Gamma^*$, let $a_0 = \vdash$ and $a_{\ell+1} = \dashv$ for uniformity. Then the set of nodes of the graph is a discrete k-dimensional cube $V = \{0, 1, \ldots, \ell, \ell+1\}^k$, with each node $(i_1, \ldots, i_k) \in V$ labelled with the k-tuple of symbols currently observed by the heads: $(a_{i_1}, \ldots, a_{i_k}) \in (\Sigma \cup \{\vdash, \dashv\})^k$. The initial node is $v_0 = (0, \ldots, 0)$, labelled with a k-tuple of left end-markers, $(\vdash, \ldots, \vdash)$. The transition from a node $(i_1, \ldots, i_k) \in V$ in a direction $(d_1, \ldots, d_k) \in D$ leads to their vector sum.

$$(i_1, \ldots, i_k) + (d_1, \ldots, d_k) = (i_1 + d_1, \ldots, i_k + d_k)$$

An example of such a graph is given in Fig. 3(left).

The set of graphs representing configurations of k-head 2DFA, as described in Example 3, is not the whole set of connected graphs over the given signature. For instance, the graph given in Fig. 3(right) no longer corresponds to the space of configurations of any k-head 2DFA on any input, even though all labels and connections conform to the signature. The way the graph is drawn, one can picture a broken hardware for a 2-head 2DFA, where the second head cannot visit the symbol a and only switches between the end-markers, and the movement of the first head is restricted while the second head is at the right position. Regardless of what the program (transition function) is, this 2-head 2DFA would not function correctly. However, the above definition of the GWA representation of 2-head 2DFA simply does not apply to any graphs of an incorrect form. What is important, is that *on the subset of graphs of the intended form*, as described in Example 3, the GWA correctly represents the behaviour of a k-head 2DFA.

Several other models of computation can be described by GWA in a similar way, with valid representations of these models implemented on a subset of well-formed graphs. Consider *multi-tape finite automata*, introduced in the famous paper by Rabin and Scott [46] and studied, in particular, by Harju and Karhumäki [21].

Example 4. A k-tape two-way deterministic finite automaton has k-tuples of positions of heads as memory configurations. Let $\Gamma_1, \ldots, \Gamma_k$ be the tape alphabets. The representation of these automata as GWA 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 $a_{i,\ell_i+1} = \dashv$, for each i. 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 [6], are 2DFA equipped with a fixed number of pebbles, which may be dispensed at or collected from the currently visited square; it is known that 2DFA with a single pebble recognize only regular languages [6], whereas two pebbles allow some non-regular languages to be represented.

Example 5. A 2DFA with k pebbles has memory configurations comprised of the position of the head and the positions of all k pebbles. Let Γ be the tape alphabet. The graph-walking representation uses the set of directions $D = \{-1, +1\} \cup$

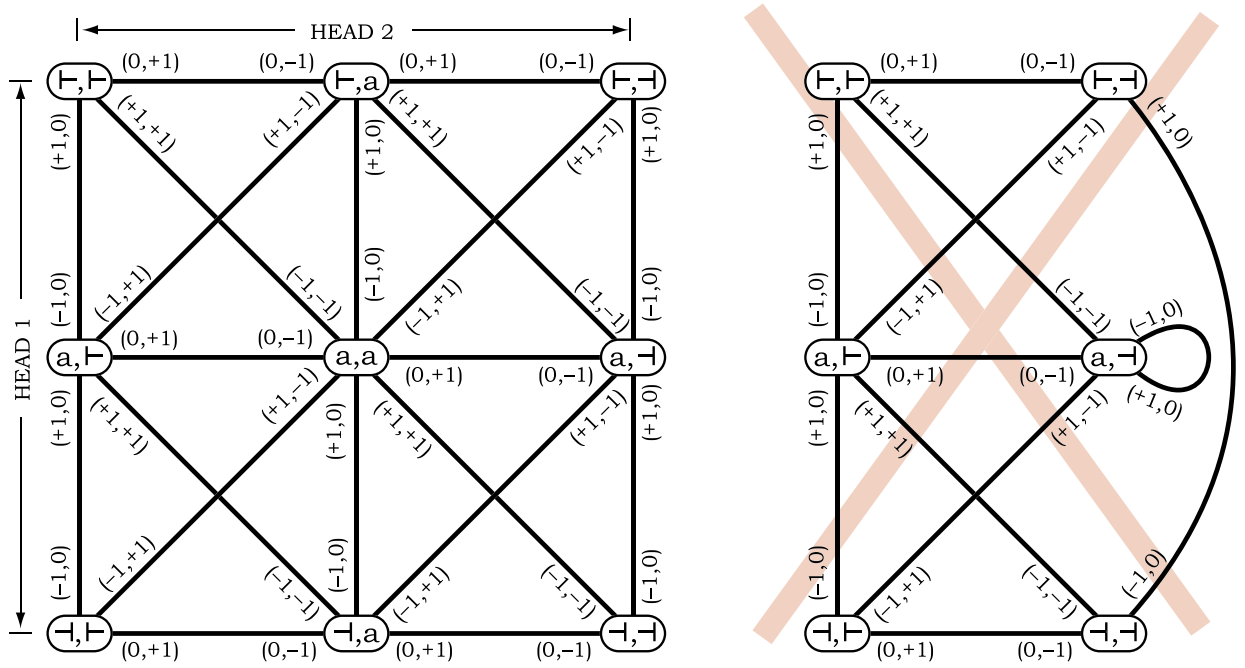

Fig. 3. (left) The space of head positions of a 2-head 2DFA on the input $w = a$, as a graph processed by a GWA; (right) A graph over the same signature that does not correspond to head positions of any 2-head 2DFA on any input.

$\{\downarrow 1, \uparrow 1, \ldots, \downarrow k, \uparrow k\}$, 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 $\Sigma = (\Gamma \cup \{\vdash, \dashv\}) \times \{\text{here}, \text{elsewhere}, \text{nowhere}\}^k$ 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 $p_t = \text{nowhere}$, and all $\uparrow t$ with $p_t = \text{here}$. The initial label represents the head at the left end-marker, with all pebbles available to be placed: $\Sigma_0 = \{(\vdash, \text{nowhere}, \ldots, \text{nowhere})\}$.

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 reports on the location of the t-th pebble.

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

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

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

The restriction that, at every step, the automaton *either* does something with a pebble *or* moves the head, is essential for these operations to have inverse operations. Indeed, for a combined operation of handling a pebble and then moving the head, the inverse operation would have to move the head and then handle a pebble at the neighbouring square; however, this cannot be done without knowing, which pebbles are placed at that square.

A restricted case of pebble 2DFA was studied by Globerman and Harel [20]: those are *2DFA with k nested pebbles*, which, at every moment, have pebbles 1, ..., i placed, and pebbles $i + 1$, ..., k available to be placed. Then, the automaton may either lift the i-th pebble, or place the $(i + 1)$-th pebble, but cannot do any other operations on pebbles. In this variant, the set of node labels is restricted to the union of $(\Gamma \cup \{\vdash, \dashv\}) \times \{\text{here}, \text{elsewhere}\}^i \times \{\text{nowhere}\}^{k-i}$, for all i, and the graphs, as described in Example 5, are restricted to the subgraph of nodes with such labels.

Many other variants of pebble automata exist, and can also be represented as graph-walking automata. For instance, pebble automata can be extended to *tree-walking automata with pebbles*, first considered by Engelfriet and Hoogeboom [15] and subsequently studied by Bojańczyk et al. [9] and by Muscholl et al. [44]. 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 GWA 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 [47, Sect. 5], rather than the more usual unmarked space with the condition of space constructibility. Besides this previously studied condition, there is one more assumption to impose on Turing machines: they shall be *initial-aware*, meaning that whenever the machine is in its initial memory configuration, that is, with empty work tape and with all heads parked at the left end-markers on their tapes, it *can detect that fact*, and use it in its transitions. This extension may sound odd, but it is necessary to conform to the condition that *the initial node must be specially labelled*, included in the basic definitions.

In general, in a graph representing a Turing machine's space of memory configurations, different nodes correspond to different work tape contents and different head positions. A node's label gives out only a pair of symbols scanned by the two heads. In a non-initial node, a GWA sees a label of the form (a, s), where a and s are the currently observed symbols on the input tape and on the work tape, respectively. The initial node of a GWA corresponds to the Turing machine's initial configuration, with an empty work tape, and with both machine's heads scanning a left end-marker; this configuration has a special initial label ι. Whenever the graph-walking automaton sees this label, it knows that the heads are parked at the left end-markers and the work tape is empty. On the other hand, if both heads see left end-markers and the work tape is *not empty*, then the node has a regular non-initial label $(\vdash, \vdash)$. This is how a GWA simulating a Turing machine knows whether the work tape is empty when both Turing machine's heads are at the leftmost position.

Example 6. Consider two-tape initial-aware Turing machines using an alphabet Γ on the read-only input tape, and 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 each node containing the position of two heads and the work tape contents. These graphs are defined with the following set of directions.

$$D = \{(-1,0), (+1,0), (0,-1), (0,+1)\} \cup \{\, cc' \mid c, c' \in \Omega,\ c \neq c' \,\}$$

Every direction (i, j), with $i, j \in \{-1, 0, +1\}$, means moving the head on the input tape by i squares, and, simultaneously, the head of the work tape by j squares; a direction cc' means replacing the currently observed symbol c on the work tape with the symbol c'. Like in Example 5, the head motion and the rewriting of symbols are separated, and 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 observed symbols on the input tape and on the work tape, that is, $\Sigma = (\Gamma \cup \{\vdash, \dashv\}) \times (\Omega \cup \{\vdash, \dashv\}) \cup \Sigma_0$, with one special label for the initial node, that is, $\Sigma_0 = \{\iota\}$. For each node label $(a, c) \in \Sigma$, the set of available 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 the initial label ι, the set of directions is $D_\iota = D_{(\vdash,\vdash)}$.

When a Turing machine operates on an input string $w = a_1 \dots a_\ell \in \Gamma^*$, its second tape initially contains the string $\vdash \sqcup^{s(\ell)} \dashv$, where $\sqcup \in \Omega$ is a special blank symbol. Memory configurations are triples containing the positions of both heads as the first two components, and the entire contents of the work tape as the third component. In the GWA representation, the space of memory configurations is accordingly represented by a graph with the following set of nodes.

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

The initial node is $v_0 = (0, 0, \sqcup^{s(\ell)})$, with $\lambda(v_0) = \iota$. Assuming $a_0 = c_0 = \vdash$ and $a_{\ell+1} = c_{s(\ell)+1} = \dashv$, the label of each non-initial node is defined as $\lambda\big((i, j, c_1 \dots c_{s(\ell)})\big) = (a_i, c_j)$. Notably, the input symbol a_i is hard-coded in the label, whereas the symbol c_j on the work tape is a part of the memory configuration in the node, which is shown to the automaton through the label. The labels $(\vdash, \vdash)$ and ι are distinct: the Turing machine's ability to distinguish between these two cases is its initial-awareness.

The transition from a node $(i, j, c_1 \dots c_{s(\ell)})$ in a direction $(-1, 0)$, $(+1, 0)$, $(0, -1)$ or $(0, +1)$ leads to the node with the head position correspondingly adjusted, and with the same tape contents.

$$(i, j, c_1 \dots c_{s(\ell)}) + (d, e) = (i + d, j + e, c_1 \dots c_{s(\ell)}), \qquad \text{for } (d, e) \in \{(-1,0), (+1,0), (0,-1), (0,+1)\}$$

The transition in a direction cc' changes the current symbol on the work tape.

$$(i, j, c_1 \dots c_{j-1} c c_{j+1} \dots c_{s(\ell)}) + cc' = (i, j, c_1 \dots c_{j-1} c' c_{j+1} \dots c_{s(\ell)}), \qquad \text{for } c, c' \in \Omega$$

This is a standard Turing machine in all respects, except that it has the additional ability to test whether the work tape is empty, whenever both heads are parked on left end-markers, implemented by having different labels ι and $(\vdash, \vdash)$.

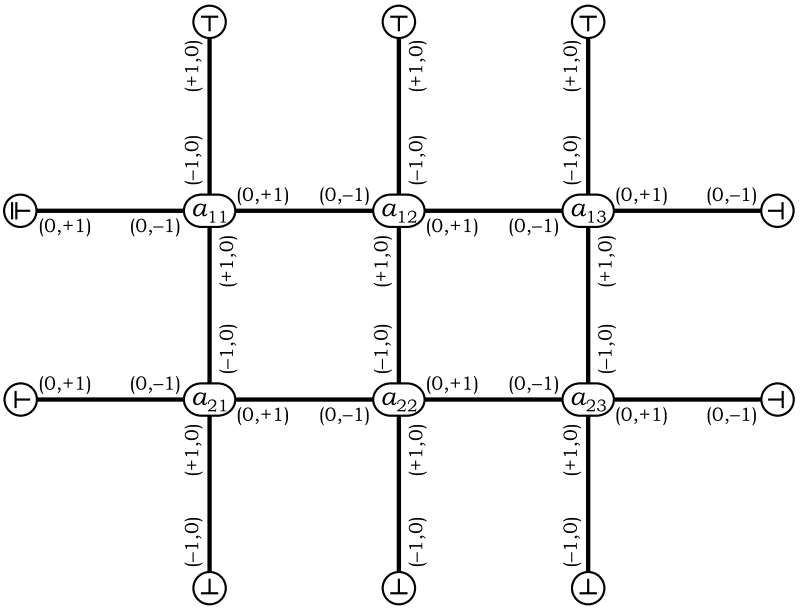

Fig. 4. The space of head positions of a 4DFA on a 2×3 input image, as a graph processed by a GWA.

Thus, an initial-aware Turing machine is a variant of a Turing machine, which has been precisely represented as a GWA on a restricted class of graphs. At the same time, it can be converted to an equivalent standard Turing machine, while preserving reversibility of all computations. This will be done in detail in Section 8, when applying the general result to particular models.

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 [6], and are described in the surveys by Inoue and Takanami [25] and by Kari and Salo [30]. Like 2DFA and TWA, these 4DFA are directly defined in terms of walking over graphs; however, the definition includes the requirement that input graphs are of the special grid form (rather than applies to all graphs over the signature).

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

Let $\Sigma = \Gamma \cup \{\top, \dashv, \bot, \vdash, \Vdash\}$ and $\Sigma_0 = \{\Vdash\}$, 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 $i \in \{1,\ldots,m\}$ and $j \in \{1,\ldots,n\}$, 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 Fig. 4.

Because the setting of graph-walking automata on undirected graphs, as they are defined in this paper, was designed with reversibility in mind, some models cannot be described within this setting. In the first place, this includes any models that cannot immediately return to the previous memory configuration after applying some operation. These and other cases shall be explained later in the Conclusion.

4. Reversibility and related notions

This section develops the definition of logical reversibility for graph-walking automata. This definition is comprised of several conditions, the first of which is that every state must be accessible from a single direction. In general, if a GWA observes some node v in a state q, and there are multiple directions in v, then it is not known, from which direction it came to this configuration, as shown in Fig. 5(left). In a *direction-determinate* GWA, there is a unique direction $d(q)$ associated with each state q, and every transition leading to q must follow that direction. Thus, when a GWA is in a state q in some node, it is known from which direction it came, but not necessarily from which state; this is illustrated in Fig. 5(right).

Definition 4. A graph-walking automaton $\mathcal{A} = (Q, q_0, \delta, F)$ is called *direction-determinate*, if every state is reachable from a unique direction, that is, there exists such a partial function $d\colon Q \to D$, that $\delta(q,a) = (q', d')$ implies $d' = d(q')$.

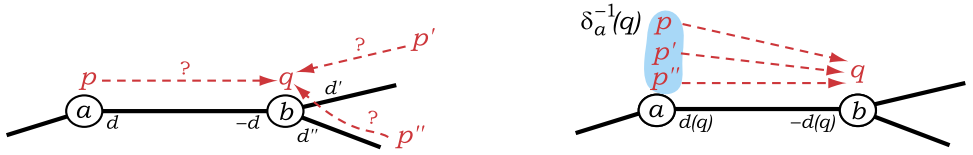

Fig. 5. (left) A GWA of a general form could have arrived to a node in a state q from any direction. (right) In a direction-determinate GWA, the direction is determined by q.

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 $q = q_0$.

Any GWA can be made direction-determinate by storing the direction it used at the last step 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.*

Proof. The new automaton has the set of states $Q' = Q \times D$, where a pair $(q, d) \in Q'$ indicates the automaton's being in the state q, having just arrived to the current node in the direction d. The initial state is $q'_0 = (q_0, d)$, where a direction $d \in D$ is chosen arbitrarily. The transition function simulates the transition of the original automaton and remembers the direction it has taken.

$$\delta'((q, d), a) = \begin{cases} ((q', d'), d'), & \text{if } \delta(q, a) = (q', d'), \\ \text{undefined}, & \text{if } \delta(q, a) \text{ undefined}. \end{cases}$$

A state $(q, d) \in Q'$ is accepting at a-labelled nodes, if q is accepting at a in the original automaton.

$$F' = \left\{ ((q, d), a) \;\middle|\; (q, a) \in F,\ d \in D \right\}$$

The constructed automaton accepts the same graphs, because it only *remembers* the direction it takes, but *never uses* this extra information. Hence, it behaves exactly as the original automaton. □

In direction-determinate automata, the requirement $\delta(q, a) \in Q \times D_a$ on the transition function in the definition of automata can be reformulated as $d(\delta_a(p)) \in D_a$, whenever $\delta_a(p)$ is defined. For that reason, a state q can be potentially reached in a node only if the node's label allows incoming transitions in the direction $d(q)$, and also, exceptionally, in an initial configuration; this is formalized in the following definition.

Definition 5. For a direction-determinate GWA $\mathcal{A} = (Q, q_0, \delta, F)$ and for a graph $(V, v_0, \lambda, +)$, a configuration $(q, v) \in Q \times V$ is called *admissible*, if $-d(q) \in D_{\lambda(v)}$, or if it is the initial configuration, $(q, v) = (q_0, v_0)$.

Any transitions or acceptance conditions invoked in inadmissible configurations are called *inaccessible transitions*, and it would be convenient to eliminate them. Since the admissibility of a configuration (q, v) depends only on the state q and on the label $\lambda(v)$, inaccessible transitions can be ruled out in the definition as follows.

Definition 6. A direction-determinate automaton $\mathcal{A} = (Q, q_0, \delta, F)$ 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, a-labelled nodes 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 $-d(q) \in D_a$.

For an automaton $\mathcal{A} = (Q, q_0, \delta, F)$ without inaccessible transitions and for an arbitrary graph $(V, v_0, \lambda, +)$, if a configuration (q, v) of $\mathcal{A}$ on this graph is not admissible, then no computation can continue from this configuration, neither by transition, nor by accepting in (q, v). In other words, all actual computations of an automaton without inaccessible transitions proceed only through admissible configurations.

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

Another subclass of automata requires returning to the initial node in order to accept.

Definition 7. A graph-walking automaton $\mathcal{A} = (Q, q_0, \delta, F)$ is called *returning*, if it has $F \subseteq Q \times \Sigma_0$, that is, if it may accept only at the initial node.

When a returning graph-walking automaton does not accept a graph, it may reject it by an undefined transition at any node, or, alternatively, it is free to loop. Thus, it is still allowed to reach a configuration, from which it cannot find the way back to the initial node of the graph; this would only mean that it has no possibility of accepting that graph.

For any standard model of computation represented as a graph-walking automaton, returning means *erasing the work memory after acceptance*, and one would expect this to be straightforward: naturally, the structure of memory configurations is unlikely to be a sophisticated maze. For instance, for each model represented by a GWA in Sections 2–3, returning after acceptance is easy: a 2DFA parks its head at the left end-marker, a 2DFA with pebbles picks up all its pebbles and also moves the head to the left, a space-bounded Turing machine erases its work tape, etc. However, for graph-walking automata operating on 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 shall establish a general transformation to a returning automaton, which finds the initial node by backtracking the accepting computation.

Theorem 1 *(proved in Section 7). 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 finite graphs.*

To be precise, the constructed automaton is always returning, in the sense that it may accept only in the initial node. For any finite graph, the constructed automaton accepts it if and only if so does the original automaton. However, the behaviour of the constructed automaton on infinite graphs is unspecified.

The proof of Theorem 1 is based on several constructions explained in the following, and it will be completed in Section 7.

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 $a \in \Sigma$, which map a state to the set of its pre-images: $\delta_a^{-1}(q) = \{\, p \mid \delta_a(p) = q \,\}$. This notation was illustrated in the earlier Fig. 5(right). Given a configuration (q, v) of a direction-determinate automaton, one can always determine the direction d, in which the automaton came to the current node v at the previous step ($d = d(q)$). If the transition function $\delta_{\lambda(v-d)}$ is furthermore *injective*, then the state at the previous step is also known ($p = \delta^{-1}_{\lambda(v-d)}(q)$), and hence the configuration at the previous step is uniquely determined—namely, it is $(p, v-d)$. This leads to the following definition of automata, whose computations can be uniquely reconstructed from their final configurations.

Definition 8. A direction-determinate graph-walking automaton $\mathcal{A} = (Q, q_0, \delta, F)$ without inaccessible transitions 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 initial label $a_0 \in \Sigma_0$, there exists at most one such state q, that $(q, a_0) \in F$, which is denoted by $q_{\mathrm{acc}}^{a_0}$. These states need not be distinct for different labels a_0.

The second condition ensures that if an input graph $(V, v_0, \lambda, +)$ is accepted, then it is accepted in the configuration $(q_{\mathrm{acc}}^{\lambda(v_0)}, v_0)$ that is known beforehand. 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.

On an infinite graph, a reversible automaton need not halt: it can just follow an infinite path in the graph, and do so perfectly reversibly. This paper concentrates on reversible automata operating on finite graphs. When the graph is finite, the natural expectation is that a reversible automaton halts; however, besides that expected outcome, the computation may also form a closed cycle.

Lemma 2. *On each finite input graph* $(V, v_0, \lambda, +)$*, a reversible graph-walking automaton, having started 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, each time it enters $(\widetilde{q}, \widetilde{v})$, it moves in the same direction $d(\widetilde{q})$. Thus, $\widetilde{v} = v + d(\widetilde{q}) = v' + d(\widetilde{q})$, and hence v and v' are the same node, $\widetilde{v} - d(\widetilde{q})$. Then, $\delta_{\lambda(v)}(q) = \delta_{\lambda(v)}(q') = \widetilde{q}$, whereas $q \neq q'$, which contradicts the automaton's reversibility. □

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

Definition 9. A reversible automaton $\mathcal{A} = (Q, q_0, \delta, F)$ is said to *have separated initial state*, if

III. the initial state of $\mathcal{A}$ is not re-enterable, that is, $\delta_a^{-1}(q_0) = \varnothing$ for every $a \in \Sigma$; furthermore, $d(q_0)$ is undefined, because there is no longer any need for it to be defined.

In particular, this definition implies that for every non-initial label $a \in \Sigma \setminus \Sigma_0$, the transition $\delta_a(q_0)$ is undefined (indeed, if it were defined, then this transition would have to be accessible, and so $-d(q_0)$ would have to be defined).

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

Lemma 3 immediately follows from Lemma 2 applied to the initial configuration (q_0, v_0), as Definition 9 rules out the undesired case in Lemma 2.

Besides the non-halting issue, the above definition of reversible automata has another imperfection: whereas such automata may accept a given graph 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 following strengthened definition additionally requires rejection to take place in a unique configuration, analogous to the accepting configuration. Furthermore, transition functions on non-initial labels are now required to be bijective.

Definition 10. A *strongly reversible* graph-walking automaton is a reversible automaton $\mathcal{A} = (Q, q_0, \delta, F)$ with separated initial state, which additionally satisfies the following conditions.

IV. For every non-initial label $a \in \Sigma \setminus \Sigma_0$, the partial function δ_a is a *bijection* from the set $\{ p \in Q \mid -d(p) \in D_a \}$ of states that can be possibly entered in a-labelled nodes, to the set $\{ q \in Q \mid d(q) \in D_a \}$ of states reachable in the directions available at a.

V. For every initial label $a_0 \in \Sigma_0$, there is at most one so-called rejecting state $q_{\mathrm{rej}}^{a_0} \in Q$, with neither $\delta_{a_0}(q_{\mathrm{rej}}^{a_0})$ defined, nor $(q_{\mathrm{rej}}^{a_0}, a_0)$ in F. Then, for each state $p \in Q$, any behaviour is defined on a_0-labelled nodes (that is, transition, acceptance in $q_{acc}^{a_0}$, or rejection in $q_{rej}^{a_0}$) if and only if either the direction for entering p is available at a_0-labelled nodes, or p is the initial state.

To be precise, for each $p \in Q$ and $a_0 \in \Sigma_0$, the following two statements are equivalent.

$$\delta_{a_0}(p) \text{ is defined, or } p = q_{\mathrm{acc}}^{a_0}, \text{ or } p = q_{\mathrm{rej}}^{a_0} \qquad \textit{if and only if} \qquad -d(p) \in D_{a_0} \text{ or } p = q_0.$$

Condition (IV) states that whenever there is a potential way to arrive at a label a in the state p—that is, the corresponding direction is available in a-labelled nodes—then the computation can continue; and vice versa: if the computation can continue from the state p in a, then there is a way of arriving there. Furthermore, the condition asserts that there is a one-to-one correspondence between possible arrivals and possible departures, so that the node can be reversibly traversed in both directions, without ever getting stuck, and without any computations beginning nowhere. This applies to all nodes except the initial node.

Condition (V) explains the operation of the automaton on initial labels. In reversible automata, the need to begin and to end somewhere is a certain annoyance that disrupts the otherwise perfect flow of computation, as it is described in condition (IV). Notably, at an initial label, there is *one* beginning of computations (the initial configuration), and there can be *two* possible endings for computations (acceptance and rejection). This means that, at an initial label, there is one less outgoing transition than there are incoming transitions, which rules out having a bijective transition function. Therefore, at an initial label, there is only the injectivity (condition (I)), and a separately formulated rule, that if a configuration is entered, then there is something for the automaton to do (condition (V)).

Note that conditions (IV) and (V) imply the condition of having no inaccessible transitions, which is therefore no longer used for strongly reversible automata.

The structure of all possible computations in strongly reversible graph-walking automata is described in the next lemma, which, in particular, explains how having one initial configuration and two final configurations affects this structure.

Lemma 4. *For every finite input graph $(V, v_0, \lambda, +)$, a strongly reversible graph-walking automaton, having started 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 *(proved in Section 7). 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 finite graphs.*

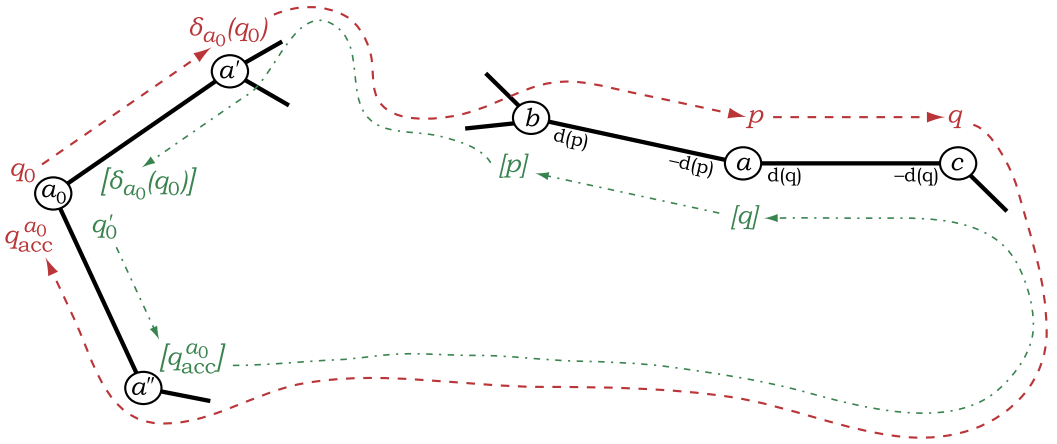

Fig. 6. Simulating a computation of a reversible GWA $\mathcal{A}$ (from q_0 to $q_{\mathrm{acc}}^{a_0}$) by a reverse computation of the GWA $\mathcal{A}^{\mathrm{r}}$ (from q_0' to $[\delta_{a_0}(q_0)]$).

Like in the case of Theorem 1, this construction always produces a strongly reversible automaton, and for any finite graph, this automaton accepts it if and only if it was accepted by the original automaton. However, the behaviour of the constructed automaton on infinite graphs is unspecified.

The proof of Theorem 2 proceeds in the following sections, and shall be finished in Section 7.

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 finite graphs.*

Proof. 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. □

5. Reversing a reversible automaton

The overall idea of reversibility is that a computation can be reconstructed from its final configuration by retracting every step of the automaton. In this section, it is shown 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, from the accepting configuration backwards. If this computation leads $\mathcal{A}^{\mathrm{r}}$ to the initial configuration of $\mathcal{A}$, then $\mathcal{A}^{\mathrm{r}}$ will 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.

Let $\mathcal{A} = (Q, q_0, \delta, F)$ be a reversible automaton. The new direction-determinate automaton with separated initial state, $\mathcal{A}^{\mathrm{r}} = (Q', q_0', \delta', F')$, simulates each state $q \in Q$ in the corresponding *backward state*, denoted by $[q]$, and also has a new initial state q_0'.

$$Q' = [Q] \cup \{q_0'\}, \qquad \text{where } [Q] = \{ [q] \mid q \in Q \}$$

Each state $[q]$ is accessed from the direction opposite to the one from which q is accessed in $\mathcal{A}$, that is, $d'([q]) = -d(q)$ for all $q \in Q$. When the new automaton is in a state $[q]$, with the head at a node v, this corresponds to the original automaton's being in the state q at *the neighbouring node* $v + d(q)$. Then, 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 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, illustrated in Fig. 6, begins by one of the following transitions from q_0', defined for every initial label $a_0 \in \Sigma_0$.

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

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.

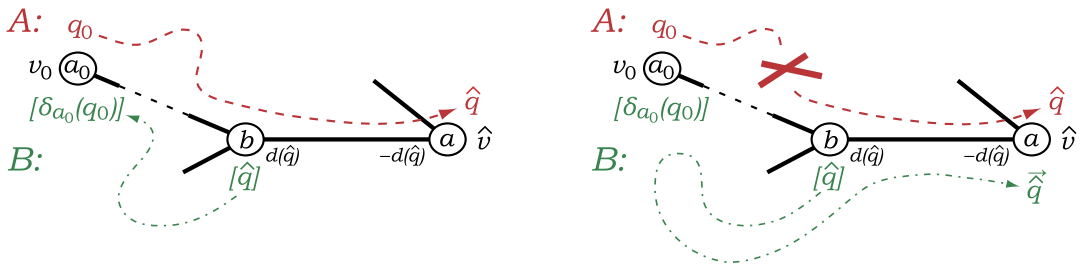

Fig. 7. Two cases in Lemma 6: (left) acceptance; (right) rejection.

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

Note that $\delta_a(p) = q$ is equivalent to $\delta_a^{-1}(q) = \{p\}$, because the automaton $\mathcal{A}$ is reversible. There are two cases of undefined transitions. First, if there is no such state $p \in Q$ that $\delta_a(p) = q$, then the current configuration of $\mathcal{A}$ has no predecessors. This means that the unique computation of $\mathcal{A}$ leading to the accepting configuration begins at this point, and is accordingly disjoint with the computation that begins in the initial configuration. Therefore, $\mathcal{A}$ does not accept this graph, and $\mathcal{A}^{\mathrm{r}}$ rejects it as well. 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 accepted. Another kind of accepting configuration corresponds to the case of $\mathcal{A}$ accepting immediately, with $(q_0, a_0) \in F$; in this case, $\mathcal{A}^{\mathrm{r}}$ does the same in the state q'_0. These two cases form the set of accepting conditions of $\mathcal{A}^{\mathrm{r}}$.

$$F' = \left\{ \left([\delta_{a_0}(q_0)], a_0\right) \,\middle|\, a_0 \in \Sigma_0,\ \delta_{a_0}(q_0) \text{ is defined} \right\} \cup \{ (q'_0, a_0) \mid a_0 \in \Sigma_0,\ (q_0, a_0) \in F \}$$

In the notation of Definition 8(II), for every initial label $a_0 \in \Sigma_0$, the accepting state is $q'^{a_0}_{\mathrm{acc}} = [\delta_{a_0}(q_0)]$ if $\delta_{a_0}(q_0)$ is defined, and $q'^{a_0}_{\mathrm{acc}} = q'_0$ if $q^{a_0}_{\mathrm{acc}} = q_0$.

Lemma 5 *(proved in Section 9). For every reversible automaton $\mathcal{A}$, the above construction correctly defines a reversible automaton $\mathcal{A}^{\mathrm{r}}$ with separated initial state, which recognizes the same set of graphs as $\mathcal{A}$.*

As the proof of this lemma is not used for establishing the main results of this paper, it is deferred to the last section.

6. Construction of reversible automata

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 a direction-determinate 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 an 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 an accepting configuration, in search 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 6. *For every direction-determinate automaton $\mathcal{A} = (Q, q_0, \delta, F)$ without inaccessible transitions, there exists a reversible automaton $\mathcal{B} = (\overrightarrow{Q} \cup [Q], \delta', F')$ without an initial state, and with states $\overrightarrow{Q} = \{ \overrightarrow{q} \mid q \in Q \}$ and $[Q] = \{ [q] \mid q \in Q \}$, which simulates $\mathcal{A}$ as follows. Each state $q \in Q$ has two corresponding states in $\mathcal{B}$: a* forward state $\overrightarrow{q}$ *accessible in the same direction $d'(\overrightarrow{q}) = d(q)$, and a* backward state $[q]$ *accessible in the opposite direction $d'([q]) = -d(q)$. The acceptance conditions of $\mathcal{B}$ are $F' = \left\{ \left([\delta_{a_0}(q_0)], a_0\right) \,\middle|\, a_0 \in \Sigma_0,\ \delta_{a_0}(q_0) \text{ is defined} \right\}$. For every finite graph $(V, v_0, \lambda, +)$, its node $\widehat{v} \in V$ and a state $\widehat{q} \in Q$ of the original automaton, for which $(\widehat{q}, \lambda(\widehat{v})) \in F$ and $-d(\widehat{q}) \in D_{\lambda(\widehat{v})}$, the computation of $\mathcal{B}$ beginning in the configuration $([\widehat{q}], \widehat{v} - d(\widehat{q}))$ has one of the following two outcomes, illustrated in Fig. 7.*

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

Furthermore, the undefined values of the transition function of $\mathcal{B}$ are characterized as follows.

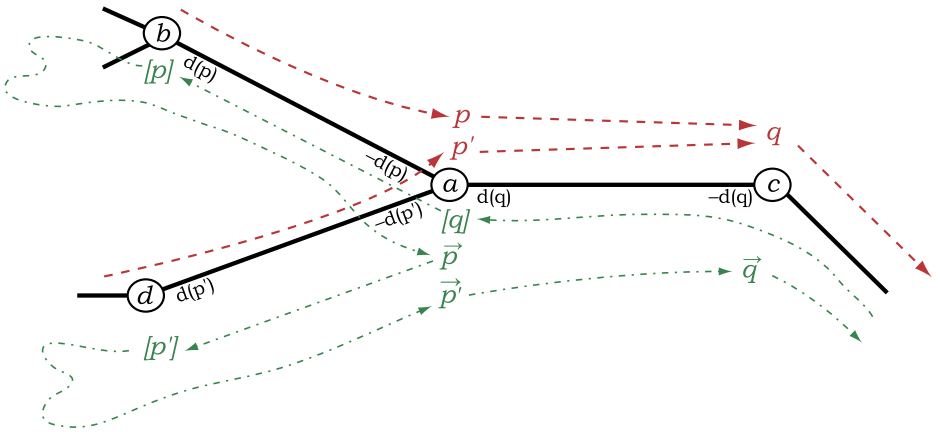

Fig. 8. A reversible GWA $\mathcal{B}$ searching the tree of computations of a GWA $\mathcal{A}$: handling irreversibility $\delta_a(p) = \delta_a(p') = q$, with $\delta_a^{-1}(q) = \{p, p'\}$ and $p < p'$.

i. *Transition* $\delta'_a([q])$ *in a backward state* $[q] \in [Q]$ *at any label* $a \in \Sigma$ *is undefined if and only if either* q *is not enterable from* a*-labelled nodes* ($d(q) \notin D_a$)*, or* $a \in \Sigma_0$ *and* $q = \delta_a(q_0)$*. In the latter case,* $\mathcal{B}$ *accepts by* $([q], a) \in F'$*.*

ii. *Transition* $\delta'_a(\overrightarrow{p})$ *in a forward state* $\overrightarrow{p} \in \overrightarrow{Q}$ *at any label* $a \in \Sigma$ *is undefined if and only if this state is not enterable in* a*-labelled nodes* ($-d(p) \notin D_a$) *or* $\mathcal{A}$ *accepts in the state* p ($(p, a) \in F$)*.*

iii. *On a non-initial label* $a \in \Sigma \setminus \Sigma_0$*, there is no transition to a forward state* $\overrightarrow{r} \in \overrightarrow{Q}$ *(that is,* $(\delta'_a)^{-1}(\overrightarrow{r}) = \varnothing$*) if and only if* $d(r) \notin D_a$*.*

iv. *On a non-initial label* $a \in \Sigma \setminus \Sigma_0$*, there is no transition to a backward state* $[r] \in [Q]$ *(that is,* $(\delta'_a)^{-1}([r]) = \varnothing$*) if and only if* $-d(r) \notin D_a$ *or* $(r, a) \in F$*.*

For an initial label $a_0 \in \Sigma_0$*, if* $(r, a_0) \in F$*, then there is no transition to* $[r]$*.*

For understanding the main idea of this lemma, assertions i–iv can be safely ignored; they shall be used later to construct strongly reversible automata using this lemma. Assertion i says that if there is a direction available in a-labelled nodes, which can be used to get to these nodes in state $[q]$, then the new automaton in such configurations either accepts or has a transition defined. This corresponds to the requirement on domains in Definition 10(IV) for backward-moving states. Analogously, assertion ii of the lemma corresponds to the requirement on domains in Definition 10(IV) for forward-moving states. The only exception is when an a-labelled node is reached in a state $\overrightarrow{p}$ such that (p, a) is an acceptance condition of the original automaton; then $\mathcal{B}$ rejects, and these are the only rejecting configurations of $\mathcal{B}$. Assertions iii and iv mean that if there is a direction for reaching the state $\overrightarrow{r}$ ($[r]$, respectively) from a-labelled nodes, then there exists a state from which one can use this direction to get to $\overrightarrow{r}$ ($[r]$, respectively). This corresponds to the requirement on ranges in Definition 10(IV). The only exceptions are configurations of $\mathcal{B}$ with states $[r]$, which correspond to accepting configurations of $\mathcal{A}$; these are not reached by any transitions, because they are the configurations where the backward search begins.

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, a *forward state* $\overrightarrow{q} \in \overrightarrow{Q}$ and a *backward state* $[q] \in [Q]$. In the states of the form $[q]$, the automaton simulates the computation backwards and has $d'([q]) = -d(q)$, whereas states $\overrightarrow{q}$, with $d'(\overrightarrow{q}) = d(q)$, are used for forward simulation. Whenever the new automaton reaches a backward state $[q]$ in a node v, this means that the computation of the original automaton, beginning in the state q with the head at 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 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 backward state $[q]$, while observing a node labelled with $a \in \Sigma$. 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'.

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

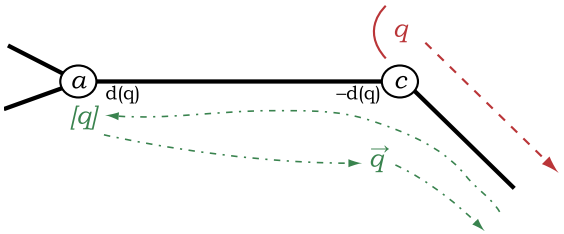

Fig. 9. A reversible GWA $\mathcal{B}$ searching the tree of computations of an irreversible GWA $\mathcal{A}$: handling an unreachable configuration.

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 by moving to the state $[p]$ in the direction $-d(p)$, as illustrated in Fig. 8. This is done by the following 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 $d'([p])$, and it remains to argue that this direction is available in nodes labelled with a. Since $\delta_a(p)$ is defined, and the possibility $(p, a) \in \{q_0\} \times \Sigma_0$ is ruled out by the assumption that $([q], a) \notin F'$, the condition 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 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. The following transition implements this.

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

This transition is compatible with the direction of the state $\overrightarrow{q}$, as $d'(\overrightarrow{q}) = d(q) \in D_a$. This case is illustrated in Fig. 9.

The transitions in a state $[q]$ 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 $\text{next}(\delta_a^{-1}(q), p)$. Then, as illustrated in Fig. 8, the simulation switches to backtracking the next branch, proceeding to the state $[p']$ in the appropriate direction. Accordingly, the following transition is defined.

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

The direction needed 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 in turn requires that $d'([p']) = -d(p') \in D_a$.

It is also worth mention that since $\delta_a(p)$ is defined and $\mathcal{A}$ has no inaccessible transitions by assumption, this implies that $-d(p)$ is in D_a, *or* $p = q_0$ and $a \in \Sigma_0$. The definition of the above transition (3) explicitly mentions that $-d(p) \in D_a$ only for the case when the transition $\delta_a(p)$ is defined in the initial configuration.

The next case is when there was no branching at the present point, or if branching took place, 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 Fig. 8. In general, such transitions are defined as follows.

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

Here 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 $\delta_a(p)$. These transitions shall never be used, but they are necessary to comply with the definition of strong reversibility.

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

This time, the assumption $-d(p) \in D_a$ guarantees that the required direction $d'([p]) = -d(p)$ exists in D_a.

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 the computations of $\mathcal{A}$ that 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 $p_m < p'_m$.

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

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, having 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, $\pi(q_0, v_0) = (q_0, v_0) \dots (p', v) \cdot \pi(q, v + d(q))$ 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 in 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, having started in $(q, v + d(q))$, 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, having started in (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 $\mathcal{A}$, having started 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 if $\pi(q_0, v_0)$ 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 $\pi(q_0, v_0) = (q_0, v_0) \dots (p', v) \cdot \pi(q, v + d(q))$ 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 at 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 $|\delta'^{-1}_a(\overrightarrow{r})| \leqslant 1$.

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 $|\delta'^{-1}_a([r])| \leqslant 1$.

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 $\overrightarrow{r}$ 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, $\delta'^{-1}_a([r]) \neq \varnothing$. 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, whereas 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 for $\mathcal{B}$ to 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 $([\delta_{\lambda(v_0)}(q_0)], v_0)$, $(\widehat{q}, \widehat{v}) \neq (q_0, v_0)$, whereas $\mathcal{A}$ 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. □

The above proof essentially uses the finiteness of the graph, because it refers to Lemma 2 in order to prove that the constructed automaton halts. In fact, this automaton need not halt on an infinite graph, because it may indefinitely backtrack an infinite computation beginning nowhere. This limitation of Lemma 6 is shared by all the theorems that rely upon it. These theorems are proved in the next section.

7. Proofs of the theorems

With Lemma 6 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 6. If $\mathcal{A}$ rejects or loops, the constructed automaton shall reject or loop in the same way, never reaching the backtracking stage.

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 finite graphs.*

Proof. 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 6, construct a reversible automaton $\mathcal{B} = (\overrightarrow{Q} \cup [Q], \delta', F')$ without an initial state that simulates the computation of $\mathcal{A}$ backwards. 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 in $\mathcal{B}$, is defined as follows.

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

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

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 $\mathcal{B}$ in the states from $\overrightarrow{Q} \cup [Q]$. The switch is made by a transition of the following 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 $\mathcal{C}$ proceeds from the configuration $(\widehat{q}, \widehat{v})$ to the configuration $([\widehat{q}], \widehat{v} - d(\widehat{q}))$, and from this moment on it operates as $\mathcal{B}$. Then, according to Lemma 6, the automaton $\mathcal{B}$ accepts in the configuration $([\delta_{\lambda(v_0)}(q_0)], v_0)$, 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 the acceptance conditions of $\mathcal{C}$ to simulate $\mathcal{B}$. The transition function δ'' is defined as δ' for all states from $\overrightarrow{Q} \cup [Q]$.

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

The constructed automaton $\mathcal{C}$ is set to accept whenever the simulated $\mathcal{B}$ accepts. It is also set to accept in the special case of $\mathcal{A}$ accepting in its initial configuration.

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

The second theorem, which asserts that a returning direction-determinate GWA $\mathcal{A}$ can be simulated by a strongly reversible GWA, is proved as follows. The core of the construction is a reversible automaton $\mathcal{B}$ constructed for $\mathcal{A}$ as in Lemma 6. 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 verifies that the resulting automaton recognizes the same language as $\mathcal{A}$.

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 finite graphs.*

Proof. 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 6. Assume any linear ordering on Q. The strongly reversible automaton will successively consider all possible accepting configurations of $\mathcal{A}$, according to the chosen ordering of states, and for each of them it will search the tree of computations leading to this configuration. Since $\mathcal{A}$ is returning, acceptance can only happen in the initial node. Furthermore, the search can only be started from the states in which $\mathcal{A}$ can potentially arrive at the initial node, in the sense that there is an appropriate direction for doing that; thus, the automaton has to consider, for every initial label $a_0 \in \Sigma_0$, the set $F_{a_0} = \{ p \in Q \mid (p, a_0) \in F,\ -d(p) \in D_{a_0} \}$. The assumption that $\mathcal{A}$ is without inaccessible transitions implies that the only state which can be possibly excluded by the requirement $-d(p) \in D_{a_0}$ is the initial state q_0.

A new direction-determinate graph-walking automaton $\mathcal{C} = (\{q'_0\} \cup \overrightarrow{Q} \cup [Q], q'_0, \delta'', F'')$ is defined by modifying $\mathcal{B}$ as follows. In all states from $\overrightarrow{Q} \cup [Q]$, the modified automaton has the same transitions as $\mathcal{B}$. The automaton $\mathcal{C}$ starts in a new separated initial state q'_0, in which it begins backtracking all possible accepting computations of $\mathcal{A}$, one by one.

The computation of $\mathcal{C}$ is illustrated in Fig. 10, where $F_{a_0} = \{p, p'\}$, with $p < p'$, and the actual computation of $\mathcal{A}$ ends with acceptance in p'. Not knowing that in advance, $\mathcal{C}$ begins with invoking $\mathcal{B}$ to backtrack $\mathcal{A}$'s computation ending in p, beginning in the state $[p]$ in the node $v_0 - d(p)$. This backtracking fails, and $\mathcal{B}$ eventually returns to the initial node in the state $\overrightarrow{p}$, which indicates the failure. Then $\mathcal{C}$ chooses the next accepting state p' and initiates another backtracking. In the example in Fig. 10, $\mathcal{B}$ successfully finds the initial configuration by returning to the initial node in the state $[r]$, with $r = \delta_{a_0}(q_0)$, in which it accepts as per Lemma 6.

In general, $\mathcal{C}$ begins by invoking $\mathcal{B}$ to backtrack the computations of $\mathcal{A}$ leading to the *least* among the states in F_{a_0}, where a_0 is the label of the initial node.

$$\delta''_{a_0}(q'_0) = [\min F_{a_0}], \qquad \text{for every } a_0 \in \Sigma_0 \text{ with } (q_0, a_0) \notin F \text{ and } F_{a_0} \neq \varnothing$$

Let $p = \min F_{a_0}$. From the state $[p]$, the transitions defined for $\mathcal{B}$ carry out the backtracking.

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

If an initial configuration of $\mathcal{A}$ is found, the new automaton accepts, as defined in $\mathcal{B}$. Otherwise, $\mathcal{B}$ eventually returns to the initial node in the state $\overrightarrow{p}$. If p is not the greatest state in F_{a_0}, then $\mathcal{C}$ chooses the next accepting state p' and initiates another backtracking.

$$\delta''_{a_0}(\overrightarrow{p}) = [\text{next}(F_{a_0}, p)], \qquad \text{for every } a_0 \in \Sigma_0 \text{ and } p \in F_{a_0} \setminus \{\max F_{a_0}\}$$

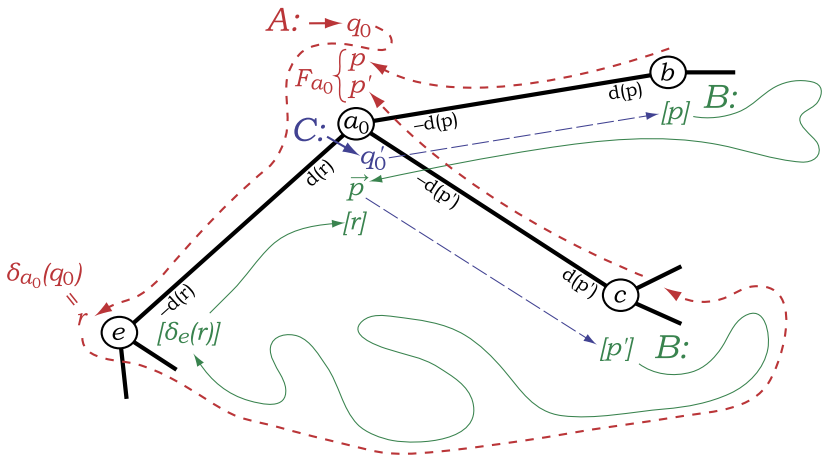

Fig. 10. A strongly reversible GWA $\mathcal{C}$ searching through the computations of a GWA $\mathcal{A}$ leading to acceptance: here, $F_{a_0} = \{p, p'\}$, with $p < p'$, and $r = \delta_{a_0}(q_0)$.

If backtracking fails for each of the accepting states of $\mathcal{A}$, then $\mathcal{B}$ eventually returns to the initial node in the state $\overrightarrow{\max F_{a_0}}$, which is the rejecting state of $\mathcal{C}$ on a_0. In the special case of $F_{a_0} = \varnothing$, no computation of $\mathcal{A}$ on a graph with the initial node labelled with a_0 can ever return to the initial node in an accepting state, and consequently, such a graph can only be accepted if $(q_0, a_0) \in F$. Accordingly, rejecting states of $\mathcal{C}$ are defined for every $a_0 \in \Sigma_0$ as follows.

$$q_{\mathrm{rej}}^{a_0} = \begin{cases} \overrightarrow{\max F_{a_0}}, & \text{if } F_{a_0} \neq \varnothing, \\ q'_0, & \text{if } F_{a_0} = \varnothing \text{ and } (q_0, a_0) \notin F, \\ \text{undefined}, & \text{otherwise.} \end{cases}$$

The acceptance is normally in the state $[\delta_{a_0}(q_0)]$, as defined for $\mathcal{B}$ in Lemma 6. There is again a special case of $(q_0, a_0) \in F$, meaning that $\mathcal{A}$ accepts the graph immediately; in this case, $\mathcal{C}$ does the same by a special acceptance condition.

$$F'' = \underbrace{\left\{ \left([\delta_{a_0}(q_0)], a_0\right) \,\middle|\, a_0 \in \Sigma_0,\ \delta_{a_0}(q_0) \text{ is defined} \right\}}_{F'} \cup \left\{ (q'_0, a_0) \,\middle|\, a_0 \in \Sigma_0,\ (q_0, a_0) \in F \right\}$$

This completes the definition of $\mathcal{C}$.

The first goal is to verify that $\mathcal{C}$ is correctly defined. For the direction-determinance, consider that the appropriate directions for each of the transitions defined above 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 6.

Turning to the reversibility, 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 6(iv). The automaton $\mathcal{C}$ is returning by definition, and for each $a_0 \in \Sigma_0$ there exists at most one state $s \in \overrightarrow{Q} \cup [Q] \cup \{q'_0\}$ with $(s, a_0) \in F'$, 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 6, and $\delta'_a(\overrightarrow{p})$ is defined if and only if $-d'(\overrightarrow{p}) \in D_a$, by claim (ii) of Lemma 6 (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 6 imply that the range of δ'_a satisfies this condition as well.

It remains to verify the last condition V in the definition of strong reversibility. In one direction, it has to be proved that, if there is any behaviour defined in a certain non-initial state at an initial label $a_0 \in \Sigma_0$, then there is a direction for accessing this state at a_0-labelled nodes. For a state $[q] \in [Q]$, if $\delta''_{a_0}([q])$ is defined, then $\delta'_{a_0}([q])$ is defined as well, which implies $-d'([q]) \in D_{a_0}$, due to Lemma 6(i). For a state $\overrightarrow{p} \in \overrightarrow{Q}$, if $\delta'_{a_0}(\overrightarrow{p})$ was defined already in $\mathcal{B}$, then the required property $-d'(\overrightarrow{p}) \in D_{a_0}$ is given in Lemma 6(ii). If $\delta''_{a_0}(\overrightarrow{p})$ is defined in $\mathcal{C}$ in addition to the rules of $\mathcal{B}$, then p belongs to F_{a_0}, which implies that $-d'(\overrightarrow{p}) \in D_{a_0}$ holds as well. Every accepting state other than q'_0 is of the form $[\delta_{a_0}(q_0)]$, and this state exists only if the direction $d(\delta_{a_0}(q_0))$, which is equal to $-d'([\delta_{a_0}(q_0)])$, belongs to D_{a_0}. Finally, every rejecting state other than q'_0 is of the form $\overrightarrow{\max F_{a_0}}$, with $-d'(\overrightarrow{\max F_{a_0}}) = -d(\max F_{a_0}) \in D_{a_0}$ by the definition of the set F_{a_0}.

The converse implication in condition V in the definition of strong reversibility is that at every initial label $a_0 \in \Sigma_0$, there is some behaviour defined in every state that is accessible in any direction available at a_0-labelled nodes; and some behaviour is defined at an initial state as well. In order to prove that, assume first that $-d'([q]) \in D_{a_0}$, that is $d(q) \in D_{a_0}$. Under this assumption, by Lemma 6(i), either $\delta''_{a_0}([q])$ is defined, or $([q], a_0) \in F' \subseteq F''$, which shows that $[q] = q_{\mathrm{acc}}^{a_0}$. The

second case to consider is that of a forward state $\overrightarrow{p}$ with $-d'(\overrightarrow{p}) \in D_{a_0}$, that is $-d(p) \in D_{a_0}$. In this case, Lemma 6(ii) gives that either $\delta''_{a_0}(\overrightarrow{p})$ is defined, or (p, a_0) is in F. However, the latter condition implies that p belongs to F_{a_0}, and consequently either $\delta''_{a_0}(\overrightarrow{p})$ is defined as $[\mathrm{next}(F_{a_0}, p)]$ (if $p \neq \max F_{a_0}$) or $\overrightarrow{p} = q^{a_0}_{\mathrm{rej}}$ (if $p = \max F_{a_0}$). Finally, in order to deal with the initial state q'_0, note that if $\delta''_{a_0}(q'_0)$ is not defined, then either $(q_0, a_0) \in F$ or $F_{a_0} = \varnothing$. In the former case, the initial state q'_0 was defined to be the accepting state $q^{a_0}_{\mathrm{acc}}$, and if $(q_0, a_0) \notin F$, but $F_{a_0} = \varnothing$, then it was defined to be the rejecting state $q^{a_0}_{\mathrm{rej}}$. This completes the proof that $\mathcal{C}$ is strongly reversible.

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 $q \in F_{\lambda(v_0)}$ and $q \neq q_0$. 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 6, 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

In Sections 2–3, several models of computation have been represented in terms of graph-walking automata. As graph-walking automata, they are subject to all the results of Sections 4–6 on transforming GWA to several special forms, such as direction-determinate, returning and reversible. The goal of this section is to see how those general transformations of GWA apply to these particular models, and what kind of transformations of the latter models can be inferred from the general results.

8.1. Two-way finite automata

The first to consider is the case of 2DFA. This is a simple case, because connected finite graphs over the signature given in Definition 1 are in one-to-one correspondence with strings, and a GWA over this signature is nothing but a 2DFA written in a different notation. Therefore, general results on making a GWA direction-determinate and on making a GWA reversible apply to 2DFA directly, whereas for making a 2DFA returning, instead of a costly general transformation given in Theorem 1, one can simply move the head to the left before acceptance. These three stages of transformation produce 2DFA of the following size.

Proposition 1. *Every 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.*

Proof. Indeed, for 2DFA, the set of directions $D = \{-1, +1\}$ is a two-element set, and hence the transformation to direction-determinate in Lemma 1 doubles the number of states.

In order to make a $2n$-state direction-determinate 2DFA returning, it is sufficient to add one extra state, q_{park}, with $d(q_{\mathrm{park}}) = -1$, in which the automaton will park the head at the left end-marker after it decides to accept. This is done by the transitions $\delta_a(q_{\mathrm{park}}) = q_{\mathrm{park}}$, defined for every input symbol $a \in \Gamma$, and by a new acceptance condition $(q_{\mathrm{park}}, \vdash) \in F$. Then, every time the original automaton accepts not at the left end-marker, that is, in a state $q \in Q$ at a label $a \in \Gamma \cup \{\dashv\}$, this is replaced with a transition $\delta_a(q) = q_{\mathrm{park}}$.

Applying Theorem 2 to the resulting automaton gives the promised strongly reversible 2DFA with $4n+3$ states. □

In the case of 2DFA, Theorem 2 is essentially a generalization of the construction by Kondacs and Watrous [31] from 1DFA to direction-determinate 2DFA. The transformation of an n-state 2DFA to a 2DFA with $4n + O(1)$ states that halts on every input, presented by Geffert et al. [19], 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 what was actually done in the construction by Geffert et al. [19]. Already in the simplest case of 2DFA, the only known way to ensure that an automaton halts on every input is to make it reversible.

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

Proposition 2. *An n-state 2DFA over a one-symbol 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-symbol alphabet to an equivalent $(n+1)$-state sweeping 2DFA, established by the authors [32,

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.

8.2. Tree-walking automata

Similarly to the case of 2DFA, for tree-walking automata (TWA), the set of connected finite graphs over the signature in Example 2 is also in one-to-one correspondence with trees. Then, a GWA over this signature is a TWA written in a different notation, and the general transformations for GWA are directly applicable. In order to make a TWA return to the initial node, instead of using Theorem 1, it is sufficient to move the head up to the root of the tree. Altogether, the transformation produces TWA with the following number of states.

Proposition 3. *Every 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.*

Proof. The transformation to direction-determinate multiplies the number of states by the number of directions, which is $|D| = 2k$.

Parking the head after acceptance generally requires only one extra state, in which the automaton will go up and eventually park the head at the root node. However, in order to keep the resulting automaton direction-determinate, one has to use k different states for that purpose, so that the automaton always remembers from which of the k successors it has just arrived to the current node. Thus, the construction uses the states $q^1_{\text{park}}, \ldots, q^k_{\text{park}}$, with $d(q^i_{\text{park}}) = -i$ for each i, and when the automaton is at a j-th successor of some node, it goes up using a transition $\delta_{(a,j)}(q^i_{\text{park}}) = q^j_{\text{park}}$.

Reversibility is ensured by Theorem 2, which produces $2(2kn+k)+1$ states, as stated. □

Earlier, Muscholl et al. [44] proved that an n-state TWA can be transformed to a halting TWA with $O(n^2)$ states, by adapting Sipser's [48] first method that involves remembering two states. The above result based on GWA reversibility improves over that transformation.

8.3. Multi-head and multi-tape automata

Multi-head automata, represented as GWA as in Example 3, are different from 2DFA and TWA in one respect: the graph of their valid memory configurations cannot be just any connected graph over the given signature, but must be constructed in a specific way for each input string, so as to implement the operation of the multi-head automaton. What would this GWA do on graphs of any other form over that signature, does not matter at all. Applying the general constructions on GWA to the GWA in Example 3 shall yield a GWA that operates on graphs of the valid form as a valid multi-head automaton. Thus, a transformation of multi-head 2DFA can be inferred from the general transformation of GWA.

Although Theorem 1 could be applied in order to ensure the returning property, again, the plan is to use an efficient method for parking the heads of multi-head automata. However, this method is not designed to work on graphs of an unintended form, and the transformation does not guarantee that their accepting status would be preserved. However, that does not actually matter, because only the operation on well-formed graphs is of interest.

By these observations, the following transformations of k-head 2DFA can be achieved using their GWA representation.

Proposition 4. *Every 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.*

Proof. Since the set of directions $D = \{-1,0,+1\}^k \setminus \{0\}^k$ has cardinality $|D| = 3^k - 1$, the transformation to direction-determinate in Lemma 1 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 at the left end-marker one by one, using k states $q_{\text{park }1}, \ldots, q_{\text{park }k}$, with $d(q_{\text{park }i}) = (0,\ldots,0,-1,0,\ldots,0)$, where -1 is the i-th component. The heads are parked using the new transition rules $\delta_{(a_1,\ldots,a_k)}(q_{\text{park }i}) = q_{\text{park }j}$, defined for each head number $i \in \{1,\ldots,k\}$, where j the least head number with $a_j \neq \vdash$. The new acceptance conditions $(q_{\text{park }i}, (\vdash,\ldots,\vdash))$, with $i \in \{1,\ldots,k\}$, are set to accept as soon as the initial node is reached. Then, whenever the original automaton has an acceptance condition $(q,(a_1,\ldots,a_k))$ in a non-initial node, the new automaton instead has a new transition rule $\delta_{(a_1,\ldots,a_k)}(q) = q_{\text{park }j}$, where j is the least integer with $a_j \neq \vdash$. The resulting GWA correctly returns to the initial node on any well-formed graph—and this is all that is needed to ensure the desired transformation of k-head automata.

Applying Theorem 2 to the resulting automaton with $(3^k-1)n+k$ states yields the desired strongly reversible automaton. □

Note that Proposition 4 is a generalization of Proposition 1, and setting $k=1$ in Proposition 4 produces exactly Proposition 1.

Reversibility of multi-head automata was first studied by Morita [41], who defined reversible automata as those satisfying the first condition of Definition 8 in this paper, but not its second condition (the one on acceptance only in the initial node and in a unique state). Morita [41] presented the construction for reversing a reversible automaton, and then showed that an n-state k-head 2DFA can be transformed to a reversible k-head 2DFA with $O(n)$ states, where the constant factor depends both on k and on the alphabet [42]; this was an implementation of the second form of Sipser's method [48], where a state and a symbol are remembered. The general results of this paper imply a transformation with the constant factor independent of the alphabet.

Multi-tape automata are another generalization of 2DFA, and their reversibility is handled identically to Proposition 4.

Proposition 5. *An n-state k-tape 2DFA can be transformed to equivalent direction-determinate, returning and direction-determinate, and strongly reversible k-tape 2DFA with the same number of states as in Proposition 4.*

8.4. Pebble automata

The GWA representation of a 2DFA with k pebbles was defined in Example 5. Like in the case of multi-head and multi-tape automata, this representation uses graphs of a specific form that implements the definition of pebble automata. The below transformation of these automata to reversible follows the usual scheme, with general GWA-based transformations to direction-determinate and to strongly reversible, and with a specially optimized method for returning the automaton to its initial configuration.

Proposition 6. *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.*

Proof. For k-pebble 2DFA, 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 uses $k+2$ extra states, because in order to get back to the initial node, one must sweep through the entire tape to pick up all the pebbles, and remember the last direction at each step. □

For an n-state 1-pebble 2DFA, this gives a strongly reversible 1-pebble automaton with $8n+7$ states. This improves over an earlier result by Geffert and Ištoňová [18, Cor. 3.5], who proved that an n-state 1-pebble 2DFA can be transformed to a 1-pebble automaton with $60n$ states recognizing the complement of the original language.

8.5. Space-bounded Turing machines

In Example 6, it was shown how to encode a two-tape initial-aware Turing machine operating in marked space $s(\ell)$ as a graph-walking automaton. The encoding relied on a set of graphs representing the machine's valid configurations and operations that lead from one configuration to another. The reversibility of this model is ensured by the same method as above, using general methods for direction-determinacy and strong reversibility, as well as a Turing-machine-specific method for restoring the initial configuration.

Proposition 7. *An n-state initial-aware Turing machine operating in marked space $s(\ell)$ 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 initial-aware TM operating in the same space using the same alphabet, to an $((m^2-m+4)n+m+2)$-state returning direction-determinate initial-aware TM of the same kind, and to a $(2(m^2-m+4)n+2m+5)$-state strongly reversible initial-aware TM that again works in the same marked space $s(n)$ using the same m-symbol work alphabet.*

Proof. In initial-aware Turing machines, there are $m(m-1)$ directions for rewriting symbols and 4 directions for moving the heads, to the total of $|D| = m^2 - m + 4$.

Once a direction-determinate automaton with $|D| \cdot n$ states is obtained, it can be transformed into a returning automaton, while preserving direction-determinacy, as follows. The returning automaton shall use $1+1+(m-1)+1$ extra states with the following meaning. First, the automaton moves the head on the work tape all the way to the right in the state q_{r}, with $d(q_{\mathrm{r}}) = (0, +1)$, using the following transitions.

$$\delta_{(a,c)}(q_{\mathrm{r}}) = q_{\mathrm{r}} \qquad (a \in \Gamma \cup \{\vdash, \dashv\},\ c \in \Omega)$$

Next, it scans the work tape from right to left, rewriting each non-blank symbol with a blank: it moves in the state q_ℓ, with $d(q_\ell) = (0, -1)$, and rewrites each symbol $c \in \Omega \setminus \{\textvisiblespace\}$ in the corresponding state q_c, with $d(q_c) = c\textvisiblespace$.

$$\delta_{(a,\dashv)}(q_{\mathrm{r}}) = q_\ell \qquad (a \in \Gamma \cup \{\vdash, \dashv\})$$
$$\delta_{(a,\textvisiblespace)}(q_\ell) = q_\ell \qquad (a \in \Gamma \cup \{\vdash, \dashv\})$$
$$\delta_{(a,c)}(q_\ell) = q_c \qquad (a \in \Gamma \cup \{\vdash, \dashv\},\ c \in \Omega \setminus \{\textvisiblespace\})$$
$$\delta_{(a,\textvisiblespace)}(q_c) = q_\ell \qquad (a \in \Gamma \cup \{\vdash, \dashv\},\ c \in \Omega \setminus \{\textvisiblespace\})$$

Finally, the automaton moves its head on the input tape to the left in the state q_{i}, with $d(q_{\mathrm{i}}) = (-1, 0)$.

$$\delta_{(a,\vdash)}(q_\ell) = q_{\mathrm{i}} \qquad (a \in \Gamma \cup \{\dashv\})$$
$$\delta_{(a,\vdash)}(q_{\mathrm{i}}) = q_{\mathrm{i}} \qquad (a \in \Gamma \cup \{\dashv\})$$

The acceptance conditions of the returning automaton are (q_{i}, ι) and (q_ℓ, ι). Finally, every acceptance condition $(q, (a, c))$ is replaced with the transition $\delta_{(a,c)}(q) = q_{\mathrm{r}}$ if $c \neq \dashv$, and with the transition $\delta_{(a,c)}(q) = q_\ell$ if $c = \dashv$.

Using Theorem 2, the resulting initial-aware Turing machine can be turned into a strongly reversible initial-aware Turing machine with $2(m^2 - m + 4)n + 2m + 5$ states. □

For initial-aware Turing machines, as introduced in Section 3 and used in Proposition 7, the reversibility construction has now been established. However, this might not be an entirely convincing representation for reversibility in standard Turing machines. In order to justify the initial-aware model, with its special ability to test the emptiness of the work tape, it has to be shown that initial-aware Turing machines can be transformed to standard Turing machines *while maintaining the reversibility property*. Once done, this will confirm the relevance of Proposition 7 to the *reversible space* in the complexity theory.

For that reason, a separate notion of reversibility has to be defined for standard Turing machines. Such definitions are known from the literature, and this paper assumes the following variant of those definitions.

Definition 11 *(cf. Bennett [3], Lange et al. [36], etc.).* Consider a standard two-tape Turing machine operating in marked space $s(\ell)$, with an input alphabet Γ, a work alphabet Ω, a set of states Q, an initial state $q_0 \in Q$, and a transition function $\Delta\colon Q \times \Gamma \times \Omega \to (Q \times I) \cup \{\text{Accept}\}$, where the set I contains instructions $(i, j) \in \{(-1, 0), (+1, 0), (0, -1), (0, +1)\}$ for moving the heads, as well as instructions $c' \in \Omega$ for rewriting the current work tape symbol with c'.

Such a machine is called reversible if, for every $p, \tilde{p}, q \in Q$, $a, \tilde{a} \in \Gamma$, $c, \tilde{c}, c' \in \Omega$ and $g, \tilde{g} \in I$, it satisfies the following three conditions.

i. If $\Delta(p, a, c) = (q, g)$ and $\Delta(\tilde{p}, \tilde{a}, \tilde{c}) = (q, \tilde{g})$, then $g = \tilde{g}$.
ii. If $\Delta(p, a, c) = \Delta(\tilde{p}, \tilde{a}, \tilde{c}) = (q, c')$, then $c = \tilde{c}$.
iii. If $\Delta(p, a, c) = \Delta(\tilde{p}, a, c)$, then $p = \tilde{p}$.

The first two conditions together mean exactly direction-determinacy in terms of GWA. The third condition is the same as the injectivity condition in the definition of reversibility of GWA (Definition 8(I)).

In general, this definition means that, in any configuration, the current state determines the previous tape contents (for a rewrite instruction), and the previous head position (for a move instruction), while the previous state is determined by the tape symbols observed in the previous configuration. The precise definition assumed by Bennett [3] and by Lange et al. [36] ensures that the previous configuration is uniquely determined by the current state and the currently observed tape symbols. In either case, these conditions are sufficient to guarantee that on any input, every configuration of the Turing machine has at most one predecessor—and this is reversibility in the broad sense. Furthermore, under either of these conditions, the predecessor configuration can be determined by some local computations.

Proposition 8. *For every n-state reversible initial-aware Turing machine operating in marked space $s(\ell)$, there exists a $3n$-state reversible standard Turing machine using marked space $s(\ell)$, that recognizes the same language.*

Proof. Since initial-aware Turing machines are faithfully represented as graph-walking automata in Example 6, this proof uses the GWA notation for them. The only essential ability of initial-aware machines, as compared to standard Turing machines, is the possibility of having different transition rules for the configurations with both heads scanning the left end-marker, depending on whether the work tape is currently empty or not. Simulating such rules in a standard Turing machine is an exercise, the question is to do this reversibly.

Let $\mathcal{A} = (Q, q_0, \delta, F)$ be an initial-aware machine in the GWA notation. A standard Turing machine $\mathcal{M}$ is constructed with a triplicated set of states, so that each state $q \in Q$ has a *normal version* q, a *right-bound version* $\overrightarrow{q}$ and a *left-bound version* $\overleftarrow{q}$. Most of the time, $\mathcal{M}$ is in one of the normal states, $q \in Q$, and operates exactly as $\mathcal{A}$.

$$\Delta(q,a,c) = (r,(i,j)) \qquad \big(\delta(q,(a,c)) = (r,(i,j)),\ (a,c) \neq (\vdash,\vdash)\big)$$
$$\Delta(q,a,c) = (r,c') \qquad \big(\delta(q,(a,c)) = (r,cc'),\ (a,c) \neq (\vdash,\vdash)\big)$$

Every time when $\mathcal{M}$ has both heads at the left end-markers, in order to decide which transition of $\mathcal{A}$ to apply, it has to test whether its work tape is empty. If $q \in Q$ is its current state, then it switches to the right-bound state $\overrightarrow{q}$ and sallies forth across the work tape in search of any non-empty cells.

$$\Delta(q,\vdash,\vdash) = (\overrightarrow{q},(0,+1))$$
$$\Delta(\overrightarrow{q},\vdash,\textvisiblespace) = (\overrightarrow{q},(0,+1))$$

If a non-empty cell is found in the middle of the tape, or if no such cell exists and the right end-marker ($\dashv$) is reached, at this point, the automaton *knows which transition it should have applied* before it began testing the work tape for emptiness. That transition of $\mathcal{A}$ is either of the form $\delta_{(\vdash,\vdash)}(q) = r$ or of the form $\delta_{\iota}(q) = r$, for some state $r \in Q$; furthermore, it could be that, instead of the transition, $\mathcal{A}$ accepts in this configuration. In the latter case, $\mathcal{M}$ accepts at this point.

$$\Delta(\overrightarrow{q^{\iota}_{\mathrm{acc}}},\vdash,\dashv) = \text{Accept}$$

Otherwise, if there is a transition to r, then $\mathcal{M}$ switches to the left-bound version of this state, $\overleftarrow{r}$.

$$\Delta(\overrightarrow{q},\vdash,c) = (\overleftarrow{\delta_{(\vdash,\vdash)}(q)},(0,-1)) \qquad (\text{if } \delta_{(\vdash,\vdash)}(q) \text{ is defined},\ c \in \Omega \setminus \{\textvisiblespace\})$$
$$\Delta(\overrightarrow{q},\vdash,\dashv) = (\overleftarrow{\delta_{\iota}(q)},(0,-1)) \qquad (\text{if } \delta_{\iota}(q) \text{ is defined})$$

In this state, the head on the work tape is pulled back to the left end-marker ($\vdash$).

$$\Delta(\overleftarrow{r},\vdash,\textvisiblespace) = (\overleftarrow{r},(0,-1))$$

Finally, upon reaching the left end-marker, $\mathcal{M}$ finishes the implementation of $\mathcal{A}$'s transition by moving the heads as defined in the direction $d(r)$, and entering the state r.

$$\Delta(\overleftarrow{r},\vdash,\vdash) = (r,d(r))$$

It remains to show that the constructed standard Turing machine $\mathcal{M}$ is reversible, in the sense of Definition 11, and this is to be inferred from the reversibility of the original initial-aware Turing machine $\mathcal{A}$ (Definition 8). The first two conditions of Definition 11 are satisfied by the normal states in $\mathcal{M}$, because $\mathcal{A}$ is direction-determinate, and, for each state $q \in Q$ in $\mathcal{A}$, its direction $d(q)$ is reused in all transitions for the same state in $\mathcal{M}$. Right-bound states $\overrightarrow{q}$ and left-bound states $\overleftarrow{q}$ are always reached by moving the work tape head to the right and to the left, respectively, and thus the first two conditions of Definition 11 hold for these states as well.

The last condition is verified by showing that if $\Delta(r,a,c) = (s,g)$, then r is uniquely determined by s, a and c (whereas g is uniquely determined by s, by the first condition). If the state s belongs to Q and $(a,c) \neq (\vdash,\vdash)$, then this follows from the reversibility of $\mathcal{A}$. If s belongs to Q and $(a,c) = (\vdash,\vdash)$, then $r = \overleftarrow{s}$: this is the case when the automaton has just returned from its sally in the state $\overleftarrow{s}$, and now switches to the normal state s.

In the case $s = \overrightarrow{q}$, if $(a,c) = (\vdash,\vdash)$, then $r = q$: this is the beginning of a sally. Otherwise, $(a,c) = (\vdash,\textvisiblespace)$, and then the automaton is in the middle of a sally, with $r = s$.

Finally, in the case $s = \overleftarrow{q}$, there are three possibilities. If $(a,c) = (\vdash,c)$, with $c \in \Omega \setminus \{\textvisiblespace\}$, then $r = \overrightarrow{p}$, where p is the unique state in $\delta^{-1}_{(\vdash,\vdash)}(q)$. If $(a,c) = (\vdash,\dashv)$, then $r = \overrightarrow{p}$, where p is the only state in $\delta_{\iota}^{-1}(q)$. Otherwise, $(a,c) = (\vdash,\textvisiblespace)$, and then $r = s$. □

Corollary 2. *For every n-state (standard) Turing machine operating in marked space $s(\ell)$ using an m-symbol work alphabet, there exists a reversible Turing machine with $6(m^2-m+4)n+6m+15$ states operating in the same marked space $s(\ell)$ and using the same work alphabet, which recognizes the same language.*

Proof. A standard Turing machine can be regarded as an initial-aware machine that never uses its initial-awareness. Then, this n-state initial-aware machine is transformed, by Proposition 7, to a $(2(m^2-m+4)n+2m+5)$-state strongly reversible initial-aware machine. By Proposition 8, it is in turn transformed to a reversible standard Turing machine with thrice as many states, $6(m^2-m+4)n+6m+15$. □

This result is related to the result of Lange et al. [36] on the equality of reversible space to deterministic space. The present version of this result is notable for having no space overhead at all. Another construction leading to the same result was independently obtained by Morita [42,43].

8.6. Four-way finite automata on pictures

For 4DFA recognizing pictures, the transformation to reversible proceeds as in the previous cases, without any new ideas.

Proposition 9. *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.*

Proof. Transformation to direction-determinate multiplies the number of states by the number of directions, $|D| = 4$.

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 top 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 fact that it was not necessary to come up with any new ideas in order to make another model reversible, is a good point in favour of using the general GWA-based methods.

9. Further properties of reversible automata

The purpose of this section is to prove a few results announced earlier, which have so far remained unproved. This is Lemma 4 on the structure of computations in strongly reversible automata and Lemma 5 on reversing a reversible automaton. The analysis necessary to establish these results actually leads to a more detailed understanding of strongly reversible automata and their properties, which might be useful in future research.

The following properties of strongly reversible automata shall now be established, leading to a stronger form of Lemma 4.

First, so far, for each initial label a_0, it has been said that there is at most one accepting state and at most one rejecting state. The definition of strongly reversible automata allows both of them to be absent. But, in fact, at least one of these "terminal states" must always be present, provided that there exists at least one finite graph over the given signature with the initial label a_0.

The **second** property is that if there is only one terminal state, then the computation beginning in the initial configuration ends in that state.

The **third** property concerns the case of two terminal states. In this case, these are two ending points for computations, and a counting argument shows that there must be two starting points. The first starting point is the initial configuration; so where is the second starting point? It turns out that the second starting point is always in one of the neighbours of the initial node. This is the **pseudo-initial configuration** $(q, v_0 + d(q))$, where q is the unique enterable state that is not in the image of the transition function $\delta_{\lambda(v_0)}$. Then, there are two starting points—the initial and the pseudo-initial configurations—and two ending points—the accepting and the rejecting configurations—and they are connected by two computation paths. However, it is not known in advance, which path leads where.

These three properties are established in the following stronger form of Lemma 4.

Lemma 7. *For every strongly reversible graph-walking automaton $\mathcal{A} = (Q, q_0, \delta, F)$ and for every finite input graph $(V, v_0, \lambda, +)$, at least one of the states $q_{\mathrm{acc}}^{\lambda(v_0)}$ and $q_{\mathrm{rej}}^{\lambda(v_0)}$ is defined. Moreover, each of the configurations $(q_{\mathrm{acc}}^{\lambda(v_0)}, v_0)$ and $(q_{\mathrm{rej}}^{\lambda(v_0)}, v_0)$ (whichever of them exist) is the last configuration in a computation of $\mathcal{A}$. One of those computations begins in the initial configuration (q_0, v_0), and the other one (if it exists) begins in the* pseudo-initial *configuration $(q, v_0 + d(q))$, where q is the unique state with $d(q) \in D_{\lambda(v_0)}$ and $\delta^{-1}_{\lambda(v_0)}(q) = \varnothing$.*

All other computations of $\mathcal{A}$ on admissible configurations are cycles, not reachable from the outside.

Proof. The proof is by analyzing the *computation graph* of $\mathcal{A}$ on an input graph $(V, v_0, \lambda, +)$.

The computation graph is a finite directed graph defined as follows. Its vertices are *admissible configurations* $(q, v) \in Q \times V$, with either $-d(q) \in D_{\lambda(v)}$ or $(q, v) = (q_0, v_0)$. The arcs of the computation graph represent all possible transitions of $\mathcal{A}$ between these configurations. If (q, v) is neither the accepting configuration $(q_{\mathrm{acc}}^{\lambda(v_0)}, v_0)$, nor the rejecting configuration $(q_{\mathrm{rej}}^{\lambda(v_0)}, v_0)$, then the conditions (IV) and (V) in Definition 10 guarantee that $\delta_{\lambda(v)}(q)$ is defined and the vertex (q, v) in the graph of computations has out-degree one, with the single arc leading to the admissible configuration $\big(\delta_{\lambda(v)}(q), v + d(\delta_{\lambda(v)}(q))\big)$. Moreover, no transitions are defined in the configurations $(q_{\mathrm{acc}}^{\lambda(v_0)}, v_0)$ and $(q_{\mathrm{rej}}^{\lambda(v_0)}, v_0)$, that is, either vertex has out-degree zero—whereas all other vertices have out-degree one.

On the other hand, no admissible configuration (q, v) has in-degree more than one, because, by the direction-determinacy, every arc leading to (q, v) must begin in a configuration of the form $(p, v - d(q))$, with $p \in \delta^{-1}_{\lambda(v-d(q))}(q)$, which is uniquely determined due to the reversibility of $\mathcal{A}$. This shows that all computations on admissible configurations are either cycles or finite paths ending in one of the configurations $(q_{\mathrm{acc}}^{\lambda(v_0)}, v_0)$ and $(q_{\mathrm{rej}}^{\lambda(v_0)}, v_0)$; in particular, there are at

most two non-cyclic paths, each of which begins in an admissible configuration without predecessors. Because the initial state is separated, the configuration (q_0, v_0) has in-degree zero, and thus one of these non-cyclic paths must begin there.

If both states $q_{\mathrm{acc}}^{\lambda(v_0)}$ and $q_{\mathrm{rej}}^{\lambda(v_0)}$ exist, then there is another non-cyclic computation, which begins in another admissible configuration (q, v) with in-degree zero—and this is the promised *pseudo-initial* configuration. Admissibility of this configuration means that $-d(q)$ belongs to $D_{\lambda(v)}$, which can be equivalently stated as $d(q) \in D_{\lambda(v-d(q))}$. It is claimed that $v - d(q)$ must be the initial node.

Assuming the contrary, that $v - d(q) \neq v_0$, the condition (IV) applies to the transition function $\delta_{\lambda(v-d(q))}$ at that node, and the state q belongs to its image. This implies that the configuration (q, v) has a predecessor, which contradicts the assumption.

This proves that $v - d(q) = v_0$, and, therefore, the second configuration without predecessors, (q, v), has to be of the form $(q, v_0 + d(q))$, where q is not in the image of $\delta_{\lambda(v_0)}$.

Finally, in order to verify that q is the unique state with such properties, consider any state $p \in Q$ satisfying $d(p) \in D_{\lambda(v_0)}$ and $\delta^{-1}_{\lambda(v_0)}(p) = \varnothing$. Then the configuration $(p, v_0 + d(p))$ is admissible and has no predecessors in the computation graph, and so it must be the first configuration of one of the two non-cyclic computations of $\mathcal{A}$, that is, $(q, v_0 + d(q))$. □

Another result so far left unproved is Lemma 5, which claims the correctness of the construction for simulating a reversible automaton backwards, as given in Section 5.

Lemma 5. *For every reversible automaton $\mathcal{A}$, the above construction correctly defines a reversible automaton $\mathcal{A}^{\mathrm{r}}$ with separated initial state, which recognizes the same set of graphs as $\mathcal{A}$.*

Proof. First, it will be verified that the construction in Section 5 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 $[q_{\mathrm{acc}}^{a_0}]$, where $q_{\mathrm{acc}}^{a_0}$ 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 $-d(q_{\mathrm{acc}}^{a_0}) = d'([q_{\mathrm{acc}}^{a_0}])$, 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 $d'([p])$, nodes labelled with a have a direction that 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 at most 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 $[q_{\mathrm{acc}}^{a_0}]$, and hence $\delta_{a_0}(q_{\mathrm{acc}}^{a_0}) = q$, 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 $\mathcal{A}^{\mathrm{r}}$ other than (q'_0, a_0), which must be of the form $([\delta_{a_0}(q_0)], a_0) \in F'$. Because $\delta_{a_0}(q_0)$ is defined, the corresponding direction $d(\delta_{a_0}(q_0))$ belongs 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 as the original automaton, consider the computations of $\mathcal{A}$ and $\mathcal{A}^{\mathrm{r}}$ on a graph $(V, v_0, \lambda, +)$ and denote $a_0 = \lambda(v_0)$. First, note that $\mathcal{A}^{\mathrm{r}}$ accepts immediately in the initial configuration if and only if $\mathcal{A}$ does. Now let $(q_0, v_0), (q_1, v_1), \ldots, (q_\ell, v_\ell)$ be a non-trivial accepting computation of $\mathcal{A}$, where $q_1, \ldots, q_\ell \in Q$ and $v_1, \ldots, v_\ell \in V$. It has to be verified that $(q'_0, v_\ell), ([q_\ell], v_{\ell-1}), \ldots, ([q_1], v_0)$ is an accepting computation of $\mathcal{A}^{\mathrm{r}}$. The assumption that the automaton $\mathcal{A}$ is returning implies that $v_\ell = v_0$, and thus the configuration (q'_0, v_ℓ) is in fact the initial configuration of $\mathcal{A}^{\mathrm{r}}$. Since $q_\ell = q_{\mathrm{acc}}^{a_0}$ and $q_\ell \neq q_0$ ($q_\ell = q_0$ would mean that $\mathcal{A}$ accepts in the initial configuration), the first transition from (q'_0, v_ℓ) is $\delta'_{a_0}(q'_0) = [q_\ell]$, and the automaton goes from the node v_ℓ to $v_\ell + d'([q_\ell]) = v_\ell - d(q_\ell) = v_{\ell-1}$. For all $i \in \{1, \ldots, \ell - 1\}$, the computation of $\mathcal{A}$ satisfies $\delta_{\lambda(v_i)}(q_i) = q_{i+1}$, and the pair $(q_i, \lambda(v_i))$ does not belong to $\{q_0\} \times \Sigma_0$ (if it did, then $\mathcal{A}$ would loop). Therefore, $\delta'_{\lambda(v_i)}([q_{i+1}]) = [q_i]$ according to the definition of $\delta'_{\lambda(v_i)}$, 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 $([q_{i+1}], v_i)$, with $i \in \{1, \ldots, \ell - 1\}$, to the node $v_i + d'([q_i]) = v_i - d(q_i) = v_{i-1}$. Finally, the configuration $([q_1], v_0)$ is accepting, since $q_1 = \delta_{a_0}(q_0)$.

Conversely, consider an arbitrary non-trivial accepting computation of $\mathcal{A}^{\mathrm{r}}$, which is of the form $(q'_0, v_0), ([q_\ell], v_{\ell-1}), \ldots, ([q_1], v_0)$, with $q_1, \ldots, q_\ell \in Q$ and $v_1, \ldots, v_{\ell-1} \in V$, because the automaton is returning and has separated initial state. It will be shown that the sequence $(q_0, v_0), (q_1, v_1), \ldots, (q_{\ell-1}, v_{\ell-1}), (q_\ell, v_0)$ is an accepting computation of $\mathcal{A}$. First, the state q_1 is $\delta_{a_0}(q_0)$, as $([q_1], v_0)$ is an accepting configuration of $\mathcal{A}^{\mathrm{r}}$. Additionally, $v_1 = v_0 - d'([q_1]) = v_0 + d(q_1)$. 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 $\delta_{\lambda(v_i)}(q_i) = q_{i+1}$ for $i \in \{1, \ldots, \ell - 1\}$, while the corresponding motion over the nodes is $v_i + d(q_{i+1}) = v_i - d'([q_{i+1}]) = v_{i+1}$ for $i \in \{1, \ldots, \ell - 2\}$, and $v_{\ell-1} + d(q_\ell) = v_{\ell-1} - d'([q_\ell]) = v_0$. Finally, the equality $[q_\ell] = \delta'_{a_0}(q'_0)$ gives $q_\ell = q_{\mathrm{acc}}^{a_0}$, which implies that $\mathcal{A}$ accepts in the configuration (q_ℓ, v_0). □

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, for all $q \in Q$ and $a \in \Sigma$.

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

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 $\mathcal{A}$. Thus, reversible automata with separated initial state can be reversed without using any extra states. Applying this construction twice always leads back to the original automaton.

Lemma 8. *For every reversible automaton $\mathcal{A}$ without inaccessible transitions and with separated initial state, the automaton $(\mathcal{A}^{\mathrm{R}})^{\mathrm{R}}$ is the same as $\mathcal{A}$, up to renaming of states.*

Proof. Let $\mathcal{A} = (Q, q_0, \delta, F)$. Then the reversed automaton $\mathcal{A}^{\mathrm{R}} = (Q', q'_0, \delta', F')$ has the set of states $Q' = \{[q] \mid q \in Q,\ q \neq q_0\} \cup \{q'_0\}$, and the doubly reversed automaton $(\mathcal{A}^{\mathrm{R}})^{\mathrm{R}} = (Q'', q''_0, \delta'', F'')$ uses the set of states $Q'' = \{[[q]] \mid q \in Q,\ q \neq q_0\} \cup \{q''_0\}$. 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 $q \in Q \setminus \{q_0\}$, whereas $d(q_0)$, $d'(q'_0)$ and $d''(q''_0)$ are all undefined. The goal is to prove that $(\mathcal{A}^{\mathrm{R}})^{\mathrm{R}}$ is identical to $\mathcal{A}$, under the following correspondence between their states: q_0 is mapped to q''_0, and each $q \in Q \setminus \{q_0\}$ to $[[q]]$.

The first claim is that, for every initial label $a_0 \in \Sigma_0$, a state is accepting at a_0 in $\mathcal{A}$ if and only if the corresponding state is accepting at a_0 in $(\mathcal{A}^{\mathrm{R}})^{\mathrm{R}}$. First, the initial state q''_0 of $(\mathcal{A}^{\mathrm{R}})^{\mathrm{R}}$ is accepting at a_0 if and only if q'_0 is accepting at a_0 in $\mathcal{A}^{\mathrm{R}}$, which is in turn equivalent to q_0 being accepting at a_0 in $\mathcal{A}$. Otherwise, the accepting state of $(\mathcal{A}^{\mathrm{R}})^{\mathrm{R}}$ at a_0 can be of the form $[\delta'_{a_0}(q'_0)]$, with $\delta'_{a_0}(q'_0)$ defined in $\mathcal{A}^{\mathrm{R}}$. However, the transition $\delta'_{a_0}(q'_0)$ is defined if and only if $\mathcal{A}$ has an accepting state $q_{\mathrm{acc}}^{a_0}$ at a_0, and in this case $\delta'_{a_0}(q'_0) = [q_{\mathrm{acc}}^{a_0}]$, whereas the accepting state of $(\mathcal{A}^{\mathrm{R}})^{\mathrm{R}}$ at a_0 is $[\delta'_{a_0}(q'_0)] = [[q_{\mathrm{acc}}^{a_0}]]$.

The second claim is that the transitions in the initial state are the same in $\mathcal{A}$ and in $(\mathcal{A}^{\mathrm{R}})^{\mathrm{R}}$. In both automata, these transitions are defined only for initial labels. For $a_0 \in \Sigma_0$, the transition in $(\mathcal{A}^{\mathrm{R}})^{\mathrm{R}}$ is $\delta''_{a_0}(q''_0) = [\bar{q}_{\mathrm{acc}}^{a_0}]$, as long as the accepting state $\bar{q}_{\mathrm{acc}}^{a_0}$ of $\mathcal{A}^{\mathrm{R}}$ at a_0 exists and is different from q'_0. This is the case if and only if $\delta_{a_0}(q_0)$ is defined, and, if so, then $\bar{q}_{\mathrm{acc}}^{a_0} = [\delta_{a_0}(q_0)]$. Overall, either $\delta''_{a_0}(q''_0) = [[\delta_{a_0}(q_0)]]$, or both states are undefined, as claimed.

Finally, for $q \in Q \setminus \{q_0\}$, it has to be proved that transitions from q in $(\mathcal{A}^{\mathrm{R}})^{\mathrm{R}}$ agree with those from $[[q]]$ in $\mathcal{A}$. The transition $\delta''_a([[q]])$ is defined if and only if there exists a state $p \in Q \setminus \{q_0\}$ with $[q] = \delta'_a([p])$. Furthermore, if this 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. □

A statement analogous to Lemma 5 can be proved also for strongly reversible automata, that is, that the computation of a strongly reversible GWA can be backtracked by another strongly reversible GWA. This requires a few changes to the construction from Section 5, as well as an extra assumption on the signature, that for each initial label $a_0 \in \Sigma_0$ there exists a finite graph with the initial node labelled with a_0. Under this assumption, by Lemma 7, each reversed computation of $\mathcal{A}$ either reaches the initial configuration, or ends up in the pseudo-initial configuration, or is a closed cycle. Thus, in order to conform to the definition of strongly reversible automata, the reversed automaton $\mathcal{A}^{\mathrm{r}}_{\mathrm{s}} = (Q', q'_0, \delta', F'_{\mathrm{s}})$ must have its behaviour defined if it backtracks a computation of $\mathcal{A}$ to $\mathcal{A}$'s pseudo-initial configuration. For every initial label $a_0 \in \Sigma_0$, this is done as in one of the following three cases.

The typical case is when there is both an accepting state $q_{\mathrm{acc}}^{a_0}$ and a rejecting state $q_{\mathrm{rej}}^{a_0}$ in $\mathcal{A}$, and $\mathcal{A}$ does not immediately reject in its initial configuration—that is, $q_0 \neq q_{\mathrm{rej}}^{a_0}$. Then, the reversed automaton $\mathcal{A}^{\mathrm{r}}_{\mathrm{s}}$ rejects if it ever backtracks to the pseudo-initial configuration of $\mathcal{A}$, which conveniently happens at the initial node. Thus, its rejecting state is defined as follows.

$$\bar{q}_{\mathrm{rej}}^{a_0} = [q], \qquad \text{if } q_0 \neq q_{\mathrm{rej}}^{a_0}, \text{ where } q \text{ is the unique state with } d(q) \in D_{a_0} \text{ and } \delta_{a_0}^{-1}(q) = \varnothing$$

The second case is when both an accepting state $q_{\mathrm{acc}}^{a_0}$ and a rejecting state $q_{\mathrm{rej}}^{a_0}$ exist, and $\mathcal{A}$ rejects in its initial configuration ($q_0 = q_{\mathrm{rej}}^{a_0}$). In this case, $\mathcal{A}^{\mathrm{r}}_{\mathrm{s}}$ rejects in its initial configuration as well.

$$\bar{q}_{\mathrm{rej}}^{a_0} = q'_0, \qquad \text{if } q_0 = q_{\mathrm{rej}}^{a_0}$$

At the same time, $\mathcal{A}$ has another computation path from its pseudo-initial configuration to its accepting configuration, and $\mathcal{A}^{\mathrm{r}}_{\mathrm{s}}$ must backtrack that path and accept upon finding the pseudo-initial configuration of $\mathcal{A}$.

$$\bar{q}_{\mathrm{acc}}^{a_0} = [q], \qquad \text{if } q_0 = q_{\mathrm{rej}}^{a_0}, \text{ where } q \text{ is the unique state with } d(q) \in D_{a_0}, \text{ and } \delta_{a_0}^{-1}(q) = \varnothing$$

If there is no accepting state in $\mathcal{A}$, then there is no accepting computation to backtrack, and the reversed automaton $\mathcal{A}^{\mathrm{r}}_{\mathrm{s}}$ rejects in its initial configuration.

$$\bar{q}^{a_0}_{\mathrm{rej}} = q'_0, \qquad \text{if } q^{a_0}_{\mathrm{acc}} \text{ does not exist}$$

The rest of the construction is the same as in Section 5, and it remains to prove that the resulting automaton satisfies all conditions in the definition of strong reversibility.

Lemma 9. *Let $\mathcal{S} = (\Sigma, \Sigma_0, D, \langle D_a \rangle_{a\in\Sigma})$ be such a signature that for every $a_0 \in \Sigma_0$ there exists a finite graph over $\mathcal{S}$ with the initial node labelled with a_0. Then, for every strongly reversible automaton $\mathcal{A}$, the above construction correctly defines a strongly reversible automaton $\mathcal{A}^{\mathrm{r}}_{\mathrm{s}}$, which recognizes the same set of graphs as $\mathcal{A}$.*

Proof. The additional accepting states in $\mathcal{A}^{\mathrm{r}}_{\mathrm{s}}$, as compared to $\mathcal{A}^{\mathrm{r}}$, can never be used to accept any graph, because they are introduced only for those initial labels, where $\mathcal{A}^{\mathrm{r}}_{\mathrm{s}}$ rejects immediately in the initial configuration. Therefore, the new automaton accepts the same graphs as $\mathcal{A}$.

Reversibility of the automaton $\mathcal{A}^{\mathrm{r}}_{\mathrm{s}}$ follows directly from the reversibility of $\mathcal{A}^{\mathrm{r}}$ (new accepting states are introduced only for those a_0, for which they were not defined in $\mathcal{A}^{\mathrm{r}}$, and they are uniquely determined according to Lemma 7).

In order to verify condition IV, note that, by definition, for every $a \in \Sigma \setminus \Sigma_0$, the partial function δ'_a is the inverse of the partial function δ_a, that is, $\delta'_a([q]) = [p]$ if and only if $\delta_a(p) = q$, for all states $p, q \in Q$. Then $[q]$ belongs to the domain of δ'_a if and only if q belongs to the image of δ_a, which is equivalent to $d(q) \in D_a$ by the strong reversibility of $\mathcal{A}$; the latter condition can be written as $-d([q]) \in D_a$, giving the required characterization of the domain of δ'_a. Similarly, $[p]$ belongs to the image of δ'_a if and only if p belongs to the domain of δ_a, and strong reversibility of $\mathcal{A}$ implies that it is equivalent to $-d(p) \in D_a$, that is, $d([p]) \in D_a$; this is the required characterization of the image of δ'_a.

For each initial label a_0, at most one rejecting state is defined, because of the uniqueness of the pseudo-initial configuration, and because the pseudo-initial configuration can only exist if $q^{a_0}_{\mathrm{acc}}$ is defined (Lemma 7).

It remains to verify the equivalence of the two conditions in condition V of the definition of strong reversibility. First it has to be verified that the behaviour on the initial state is defined, that is, $\delta'_{a_0}(q'_0)$ is defined, or $q'_0 = \bar{q}^{a_0}_{\mathrm{acc}}$, or $q'_0 = \bar{q}^{a_0}_{\mathrm{rej}}$. Since $\mathcal{A}$ is strongly reversible, one of the following three conditions holds: $\delta_{a_0}(p)$ is defined, or $p = q^{a_0}_{\mathrm{acc}}$, or $p = q^{a_0}_{\mathrm{rej}}$. If the second or third of these conditions is true, then $q'_0 = \bar{q}^{a_0}_{\mathrm{acc}}$ or $q'_0 = \bar{q}^{a_0}_{\mathrm{rej}}$, respectively. If the first condition is true, then either $\delta'_{a_0}(q'_0)$ is defined, or $q'_0 = \bar{q}^{a_0}_{\mathrm{rej}}$, depending on whether $q^{a_0}_{\mathrm{acc}}$ exists or not.

Second, it has to be proved that for every $q \in Q$, the behaviour of $\mathcal{A}^{\mathrm{r}}_{\mathrm{s}}$ on $[q]$ is defined (that is, $\delta'_{a_0}([q])$ is defined, or $[q] = q'^{a_0}_{\mathrm{acc}}$, or $[q] = q'^{a_0}_{\mathrm{rej}}$) if and only if $d(q)$ belongs to D_{a_0}. According to the definition of $\mathcal{A}^{\mathrm{r}}_{\mathrm{s}}$, the transition $\delta'_{a_0}([q])$ is defined if and only if (1) there exists $p \in Q \setminus \{q_0\}$ such that $\delta_{a_0}(p) = q$. Further, $[q]$ is defined to be accepting at a_0 if and only if either (2) $q = \delta_{a_0}(q_0)$, or (3) $q_0 = q^{a_0}_{\mathrm{rej}}$, $d(q) \in D_{a_0}$ and $\delta^{-1}_{a_0}(q) = \varnothing$. Finally, $[q]$ is defined to be rejecting at a_0 if and only if (4) $q_0 \neq q^{a_0}_{\mathrm{rej}}$, $d(q) \in D_{a_0}$ and $\delta^{-1}_{a_0}(q) = \varnothing$. The disjunction of the conditions (1) and (2) can be written as $\delta^{-1}_{a_0}(q) \neq \varnothing$, while the disjunction of the conditions (3) and (4) is the requirement that $d(q) \in D_{a_0}$ and $\delta^{-1}_{a_0}(q) = \varnothing$. As the above condition $\delta^{-1}_{a_0}(q) \neq \varnothing$ in particular implies $d(q) \in D_{a_0}$, the disjunction of all conditions (1)–(4) is exactly $d(q) \in D_{a_0}$. □

10. Conclusion

The general method of transforming deterministic models of computation to the corresponding reversible models by backtracking their computations is known from Sipser [48] and Kondacs and Watrous [31], and, in the literature, it was reimplemented for many individual models, leading to complicated constructions of varied efficiency. The main contribution of this paper is in presenting this method in the form applicable to graph-walking automata, which generalize quite a few models in automata theory. Hence, there is no longer any need to reimplement the same ideas for any further particular models, as it is now sufficient to apply the general results for GWA.

The graph-walking automata, as defined in this paper, have the property that every edge can always be traversed backwards, or, equivalently, that every operation on the memory can be undone: this may be regarded as *local reversibility*, and the result of this paper is that **every locally reversible device can be made globally reversible**.

What about the models without this property, which cannot immediately return to the previous configuration after some operation? This includes any models, where the previous configuration may never be revisited, such as one-way finite automata [45], one-way multihead automata [34] or pushdown automata [33]. Models that may possibly revisit earlier configurations, but cannot do that at will by a single instruction, are subject to the same problem: for instance, such are the *sweeping two-way finite automata* introduced by Sipser [49] and the *rotating automata* of Kapoutsis et al. [28], 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. [26], 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 directly expressed as GWA on undirected graphs, because an operation on the memory without an opposite operation means that some edges cannot be traversed in both directions. However, in some cases, such models can be represented as a GWA by *serializing* a memory reset operation into a sequence of elementary erasing operations, each of which could be undone by an opposite operation. The alternative is to extend a given locally irreversible model with the opposite to each operation: this is how Kondacs and Watrous [31] obtained a 2DFA while reversing a 1DFA.

Another kind of models not considered in this paper are any devices with unbounded memory, where, given an input, one does not know the finite 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. However, the results of this paper essentially rely on the finiteness of input graphs.

The graph-walking automaton model is interesting not only as an intermediate abstraction for making other models reversible: indeed, the classical finite automata on strings, and the relatively well-known automata on trees are naturally generalized to automata on graphs. The 2DFA, the TWA and the GWA are the families of deterministic automata walking over structures of different complexity. Another important kind of automata are the one-way nondeterministic finite automata (1NFA), which further extend to tree automata, and have a generalization to graphs studied by Thomas [50]. Yet another kind of models in automata theory are rewriting systems acting on strings, on trees and on graphs [12]. These results suggest treating graph-walking automata as one of the basic models in automata theory, of which we know sadly very little, and which deserves further investigation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The authors are grateful to the anonymous reviewers for careful reading and helpful comments, which allowed the authors to improve the presentation all over the text.

This research was supported by the research center Institute for Theoretical Computer Science (ITI), project No. P202/12/G061 of the Czech Science Foundation.

References

[1] A.V. Aho, J.D. Ullman, Translations on a context free grammar, Inf. Control 19 (5) (1971) 439–475.
[2] H.B. Axelsen, Reversible multi-head finite automata characterize reversible logarithmic space, in: Language and Automata Theory and Applications, LATA 2012, A Coruña, Spain, 5–9 March 2012, in: LNCS, vol. 7183, 2012, pp. 95–105.
[3] C.H. Bennett, Logical reversibility of computation, IBM J. Res. Dev. 17 (6) (1973) 525–532.
[4] C.H. Bennett, The thermodynamics of computation—a review, Int. J. Theor. Phys. 21 (12) (1982) 905–940.
[5] C.H. Bennett, Time/space trade-offs for reversible computation, SIAM J. Comput. 81 (1989) 766–776.
[6] M. Blum, C. Hewitt, Automata on a 2-dimensional tape, in: 8th Annual Symposium on Switching and Automata Theory, In: SWAT 1967, Austin, Texas, USA, 18–20 October 1967, 1967, pp. 155–160.
[7] M. Bojańczyk, T. Colcombet, Tree-walking automata cannot be determinized, Theor. Comput. Sci. 350 (2–3) (2006) 164–173.
[8] M. Bojańczyk, T. Colcombet, Tree-walking automata do not recognize all regular languages, SIAM J. Comput. 38 (2) (2008) 658–701.
[9] M. Bojańczyk, M. Samuelides, T. Schwentick, L. Segoufin, Expressive power of pebble automata, in: ICALP 2006, Venice, Italy, 9–16 July 2006, Vol. 1, in: LNCS, vol. 4051, 2006, pp. 157–168.
[10] L. Budach, Automata and labyrinths, Math. Nachr. 86 (1) (1978) 195–282.
[11] H. Buhrman, J. Tromp, P. Vitányi, Time and space bounds for reversible simulation, J. Phys. A, Math. Gen. 34 (35) (2001) 6821–6830.
[12] B. Courcelle, Graph rewriting: an algebraic and logic approach, in: Handbook of Theoretical Computer Science, Volume B, 1990, pp. 193–242.
[13] P. Crescenzi, C.H. Papadimitriou, Reversible simulation of space-bounded computations, Theor. Comput. Sci. 143 (1) (1995) 159–165.
[14] A. Elmasry, T. Hagerup, F. Kammer, Space-efficient basic graph algorithms, in: STACS 2015, 2015, pp. 288–301.
[15] J. Engelfriet, H.J. Hoogeboom, Tree-walking pebble automata, in: Jewels are Forever, Contributions on Theoretical Computer Science in Honor of Arto Salomaa, 1999, pp. 72–83.
[16] J. Engelfriet, H.J. Hoogeboom, Automata with nested pebbles capture first-order logic with transitive closure, Log. Methods Comput. Sci. 3 (2–3) (2007) 1–27.
[17] P. Fraigniaud, D. Ilcinkas, G. Peer, A. Pelc, D. Peleg, Graph exploration by a finite automaton, Theor. Comput. Sci. 345 (2–3) (2005) 331–344.
[18] V. Geffert, L. Ištoňová, Translation from classical two-way automata to pebble two-way automata, RAIRO Theor. Inform. Appl. 44 (4) (2010) 507–523.
[19] V. Geffert, C. Mereghetti, G. Pighizzini, Complementing two-way finite automata, Inf. Comput. 205 (8) (2007) 1173–1187.
[20] N. Globerman, D. Harel, Complexity results for two-way and multi-pebble automata and their logics, Theor. Comput. Sci. 169 (2) (1996) 161–184.
[21] T. Harju, J. Karhumäki, The equivalence problem of multitape finite automata, Theor. Comput. Sci. 78 (2) (1991) 347–355.
[22] P.-C. Héam, A lower bound for reversible automata, RAIRO Theor. Inform. Appl. 34 (5) (2000) 331–341.
[23] M. Holzer, M. Kutrib, A. Malcher, Complexity of multi-head finite automata: origins and directions, Theor. Comput. Sci. 412 (1–2) (2011) 83–96.
[24] J.E. Hopcroft, J.D. Ullman, Some results on tape bounded Turing machines, J. ACM 16 (1967) 168–177.
[25] K. Inoue, I. Takanami, A survey of two-dimensional automata theory, Inf. Sci. 55 (1–3) (1991) 99–121.
[26] P. Jančar, F. Mráz, M. Plátek, J. Vogel, Restarting automata, in: Fundamentals of Computation Theory, FCT 1995, Dresden, Germany, 22–25 August 1995, in: LNCS, vol. 965, 1995, pp. 283–292.
[27] T. Kamimura, G. Slutzki, Parallel two-way automata on directed ordered acyclic graphs, Inf. Control 49 (1) (1981) 10–51.
[28] C.A. Kapoutsis, R. Královic, T. Mömke, Size complexity of rotating and sweeping automata, J. Comput. Syst. Sci. 78 (2) (2012) 537–558.
[29] J. Kari, Reversible cellular automata, in: Developments in Language Theory, DLT 2005, Palermo, Italy, 4–8 July 2005, in: LNCS, vol. 3572, 2005, pp. 57–68.
[30] J. Kari, V. Salo, A survey on picture-walking automata, in: Algebraic Foundations in Computer Science, in: LNCS, vol. 7020, 2011, pp. 183–213.
[31] A. Kondacs, J. Watrous, On the power of quantum finite state automata, in: 38th Annual Symposium on Foundations of Computer Science, FOCS 1997, Miami Beach, Florida, USA, 19–22 October 1997, IEEE, 1997, pp. 66–75.
[32] M. Kunc, A. Okhotin, Describing periodicity in two-way deterministic finite automata using transformation semigroups, in: Developments in Language Theory, DLT 2011, Milan, Italy, 19–22 July 2011, in: LNCS, vol. 6795, 2011, pp. 324–336.
[33] M. Kutrib, A. Malcher, Reversible pushdown automata, J. Comput. Syst. Sci. 78 (6) (2012) 1814–1827.
[34] M. Kutrib, A. Malcher, One-way reversible multi-head finite automata, Theor. Comput. Sci. 682 (2017) 149–164.

[35] R. Landauer, Irreversibility and heat generation in the computing process, IBM J. Res. Dev. 5 (3) (1961) 183–191.
[36] K.-J. Lange, P. McKenzie, A. Tapp, Reversible space equals deterministic space, J. Comput. Syst. Sci. 60 (2) (2000) 354–367.
[37] Y. Lecerf, Machines de Turing réversibles, C. R. Acad. Sci. 257 (1963) 2597–2600.
[38] M. Li, J. Tromp, P. Vitányi, Reversible simulation of irreversible computation, Physica D 120 (1998) 168–176.
[39] S. Lombardy, On the construction of reversible automata for reversible languages, in: Automata, Languages and Programming, ICALP 2002, Málaga, Spain, 8–13 July 2002, in: LNCS, vol. 2380, 2002, pp. 170–182.
[40] C. Lutz, Janus: A Time-Reversible Language, a letter to R. Landauer, 1986.
[41] K. Morita, Two-way reversible multi-head finite automata, Fundam. Inform. 110 (1–4) (2011) 241–254.
[42] K. Morita, A deterministic two-way multi-head finite automaton can be converted into a reversible one with the same number of heads, in: Reversible Computation, RC 2012, Copenhagen, Denmark, 2–3 July 2012, in: LNCS, vol. 7581, 2012, pp. 29–43.
[43] K. Morita, Reversibility in space-bounded computation, Int. J. Gen. Syst. 43 (7) (2014) 697–712.
[44] A. Muscholl, M. Samuelides, L. Segoufin, Complementing deterministic tree-walking automata, Inf. Process. Lett. 99 (1) (2006) 33–39.
[45] J.-E. Pin, On the languages accepted by finite reversible automata, in: Automata, Languages and Programming, ICALP 1987, Karlsruhe, Germany, 13–17 July 1987, in: LNCS, vol. 267, 1987, pp. 237–249.
[46] M.O. Rabin, D. Scott, Finite automata and their decision problems, IBM J. Res. Dev. 3 (2) (1959) 114–125.
[47] D. Ranjan, R. Chang, J. Hartmanis, Space bounded computations: review and new separation results, Theor. Comput. Sci. 80 (2) (1991) 289–302.
[48] M. Sipser, Halting space-bounded computations, Theor. Comput. Sci. 10 (3) (1980) 335–338.
[49] M. Sipser, Lower bounds on the size of sweeping automata, J. Comput. Syst. Sci. 21 (2) (1980) 195–202.
[50] W. Thomas, On logics, tilings, and automata, in: Automata, Languages and Programming, ICALP 1991, Madrid, Spain, 8–12 July 1991, in: LNCS, vol. 510, 1991, pp. 441–454.
[51] T. Toffoli, N.H. Margolus, Invertible cellular automata: a review, Phys. D: Nonlinear Phenom. 45 (1–3) (1990) 229–253.
[52] T. Yokoyama, H.B. Axelsen, R. Glück, Principles of a reversible programming language, in: Proceedings of the 5th Conference on Computing Frontiers, Ischia, Italy, May 5–7, 2008, ACM, 2008, pp. 43–54.