Reversibility of Computations in Graph-Walking Automata*

Michal Kunc[1] and Alexander Okhotin[2]

[1] Department of Mathematics and Statistics,
Masaryk University, Brno, Czech Republic
kunc@math.muni.cz
[2] Department of Mathematics and Statistics, University of Turku, Finland
alexander.okhotin@utu.fi

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

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, as irreversible computations incur energy dissipation [18]. It is known from Lecerf [20] 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 [4], Crescenzi and Papadimitriou [10], Lange et al. [19] and Buhrman et al. [8]. A line of research on reversibility in high-level programming languages was initiated by Abramsky [1]. Reversibility in cellular automata also has a long history of research, presented in surveys by Toffoli and Margolus [26] and by Kari [15]. In the domain of finite automata, the reversible subclass of one-way deterministic finite automata (1DFAs) defines a proper subfamily of regular languages [23]. On the other hand, every regular language is accepted by a reversible two-way finite automaton (2DFA): as shown by Kondacs and Watrous [16], every n-state 1DFA can be simulated by a $2n$-state reversible 2DFA.

* Supported by ESF project CZ.1.07/2.3.00/20.0051 "Algebraic Methods in Quantum Logic", and by Academy of Finland project 257857.

K. Chatterjee and J. Sgall (Eds.): MFCS 2013, LNCS 8087, pp. 595–606, 2013.

One of the most evident consequences of reversibility is that a reversible automaton halts on every input (provided that the state-space is bounded). 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 [14]. 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 [24], and his approach essentially means constructing a reversible machine, though reversibility was not considered as such. In particular, Sipser [24] sketched a transformation of an n-state 2DFA to an $O(n^2)$-state halting 2DFA (which is actually reversible), and also mentioned the possibility of an improved transformation that yields $O(n)$ states, where the multiplicative factor depends upon the size of the alphabet. The fact that Sipser's idea produces reversible automata was noticed and used by Lange et al. [19] to establish the equivalence of deterministic space $s(n)$ to reversible space $s(n)$. Next, Kondacs and Watrous [16] distilled the construction of Lange et al. [19] into the mathematical essence of constructing reversible 2DFAs. A similar construction for making a 2DFA halt on any input was later devised by Geffert et al. [13], who have amalgamated an independently discovered method of Kondacs and Watrous [16] with a pre-processing step. For tree-walking automata (TWA), a variant of Sipser's [24] construction was used by Muscholl et al. [22] to transform an n-state automaton to an $O(n^2)$-state halting automaton.

The above results apply to various models that recognize input structures by traversing them: such are the 2DFAs that walk over input strings, and the TWAs 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 machine of such a type is defined by a transition table, which instructs it to apply a memory operation and to change its internal state, depending on the current state and the currently observed data stored in the memory.

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

1. Every elementary operation on the memory has an opposite elementary operation that undoes its effect. For instance, in a 2DFA, the operation of moving

the head to the left can be undone by moving the head to the right. In terms of graphs, this means that *input graphs are undirected*, and each edge has its end-points labelled by two opposite direction symbols, representing traversal of this edge in both directions.
2. The space of memory configurations on each given input object is finite, and it functionally depends on the input data. For graph-walking automata, this means that *input graphs are finite*. Though, in general, reversible computation is possible in devices with unbounded memory, the methods investigated in this paper depend upon this restriction.
3. The automaton can test whether the current memory configuration is the initial configuration. In a graph-walking automaton, this means that the initial node, where the computation begins, has a distinguished label.

Besides the aforementioned 2DFAs, TWAs and space-bounded Turing machines, graph-walking automata cover such generalizations as multi-head automata, automata with pebbles, etc.

The goal of this paper is to deal with the reversibility of computations on the general level, as represented by the model of graph-walking automata. The main results of this paper are transformations from automata of the general form to the *returning automata*, which may accept only in the initial node, and from returning to *reversible automata*. Both transformations rely on the same effective construction, which generalizes the method of Kondacs and Watrous [16], while the origins of the latter can be traced to the general idea due to Sipser [24]. The constructions involve only a linear blow-up in the number of states. Both results apply to every concrete model of computation representable as GWAs.

Investigating further properties of graph-walking automata is proposed as a worthy subject for future research. Models of this kind date back to *automata in labyrinths*, introduced by Shannon and later studied by numerous authors as a model of graph exploration by an agent following the edges of an undirected graph. This line of research has evolved into a thriving field of algorithms for searching and automatic mapping of graphs, which is surveyed in the recent paper by Fraigniaud et al. [12]. Other important models defining families of graphs are graph-rewriting systems and monadic second-order logic on graphs researched by Courcelle [9], and graph tilings studied by Thomas [25].

2 Graph-Walking Automata

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

Definition 1. *A* signature $\mathcal{S}$ *consists of*

- *a finite set of directions* D*;*
- *a bijective operation* $-\colon D \to D$*, satisfying* $-(-d) = d$ *for all* $d \in D$*;*
- *a finite set* Σ *of possible labels of nodes of the graph;*
- *a non-empty subset* $\Sigma_0 \subseteq \Sigma$ *of labels allowed in the initial node;*
- *a set* $D_a \subseteq D$ *of directions for every* $a \in \Sigma$*.*

In each graph over $\mathcal{S}$, every node labelled with $a \in \Sigma$ must be of degree $|D_a|$, with the incident edges corresponding to the elements of D_a.

Definition 2. *A* graph *over the signature* $\mathcal{S}$ *is a quadruple* $(V, v_0, +, \lambda)$*, where*

- V *is a finite set of* nodes*;*
- $v_0 \in V$ *is the* initial node*;*
- $+\colon V \times D \to V$ *is a partial mapping, satisfying the following condition of invertibility by opposite directions: for every* $v \in V$ *and* $d \in D$*, if* $v + d$ *is defined, then* $(v + d) + (-d)$ *is defined too and* $(v + d) + (-d) = v$*. In the following,* $v - d$ *denotes* $v + (-d)$*;*
- *the total mapping* $\lambda\colon V \to \Sigma$ *is a labelling of nodes, such that for all* $v \in V$*,*
  (i) $d \in D_{\lambda(v)}$ *if and only if* $v + d$ *is defined,*
  (ii) $\lambda(v) \in \Sigma_0$ *if and only if* $v = v_0$*.*

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

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

Given a graph $(V, v_0, +, \lambda)$, 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.

The two most well-known special cases of GWAs are the 2DFAs, which walk over path graphs, and TWAs operating on trees.

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

All 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 $w = a_1 \dots a_n$, with $n \geqslant 0$, the corresponding graph has the set of nodes $V = \{0, 1, \dots, n, n+1\}$ representing positions on the tape, with $v_0 = 0$ and with $v + d$ defined as the sum of integers. These nodes are labelled as follows: $\lambda(0) = \vdash$, $\lambda(n+1) = \dashv$ and $\lambda(i) = a_i$ for all $i \in \{1, \dots, n\}$.

Consider *tree-walking automata*, defined by Aho and Ullman [2, Sect. VI] and later studied by Bojańczyk and Colcombet [6,7]. Given an input binary tree, a tree-walking automaton moves over it, scanning one node at a time. At each step of its computation, it may either go down to any of the sons of the current node or up to its father. Furthermore, in any node except the root, the automaton is invested with the knowledge of whether this node is the first son or the second son [6]. Traversal of trees by these automata can be described using directions of the form "go down to the i-th son" and the opposite "go up from the i-th son to its father".

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

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

In general, consider any computational device recognizing input objects of any kind, which has a fixed number of internal states and employs auxiliary memory holding such data as the positions of reading heads and the contents of any additional data structures. Assume that for each fixed input, the total space of possible memory configurations of the device and the structure of admissible transitions between these configurations are known in advance. The set of memory configurations with the structure of transitions forms a *graph of memory configurations*, which can be presented in the notation assumed in this paper by taking *elementary operations on the memory* as 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.

As an example of such a representation, consider 2DFAs equipped with multiple reading heads, which can independently move over the same input tape: *the multi-head automata*.

Example 3. A k-head 2DFA with a tape alphabet Γ is described by a graph-walking automaton as follows. Its memory configuration contains the positions of all k heads on the tape. The set of directions is $D = \{-1, 0, +1\}^k \setminus \{0\}^k$, where a direction $(s_1, \ldots, s_k)$ with $s_i \in \{-1, 0, +1\}$ indicates that each i-th head is to be moved in the direction s_i. Each label in $\Sigma = \big(\Gamma \cup \{\vdash, \dashv\}\big)^k$ contains all the data observed by the automaton in a given memory configuration: this is a k-tuple of symbols scanned by all heads. There is a unique initial label corresponding to all heads parked at the left-end marker, that is, $\Sigma_0 = \{(\vdash, \ldots, \vdash)\}$. For each node label $(s_1, \ldots, s_k) \in \Sigma$, the set of directions $D_{(s_1,\ldots,s_k)}$ contains all k-tuples $(d_1, \ldots, d_k) \in \{-1, 0, +1\}^k$, where $d_i \neq -1$ if $s_i = \vdash$ and $d_i \neq +1$ if $s_i = \dashv$; the latter conditions disallow moving any heads beyond either end-marker.

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

The graphs representing memory configurations of k-head 2DFAs, as described in Example 3, are not all connected graphs over the given signature. If edges are connected differently than in a grid of the form given above, the resulting graph no longer corresponds to the space of configurations of a k-head 2DFA on any input. However, on the subset of graphs of the intended form, a GWA defined in Example 3 correctly represents the behaviour of a k-head 2DFA.

Several other models of computation can be described by GWAs in a similar way. Consider *two-way finite automata with pebbles*, introduced by Blum and Hewitt [5]: these are 2DFAs equipped with a fixed number of pebbles, which may be dispensed at or collected from the currently visited cell. When such automata are represented as GWAs, the currently visited node of a graph represents the positions of the head and pebbles, while the label encodes the symbol observed by the head, together with the information on which pebbles are currently placed, and which of them are placed at the observed cell. This model can be extended to *tree-walking automata with pebbles*, first considered by Engelfriet and Hoogeboom [11] and subsequently studied by Muscholl et al. [22]. All these models can be further extended to have multiple reading heads, to work over multidimensional arrays (such as the 4DFAs of Blum and Hewitt [5]), etc., and each case can be described by an appropriate kind of GWAs operating over graphs that encode the space of memory configurations of the desired automata.

Typical models that cannot be described as automata walking on undirected graphs are those, which cannot immediately return to the previous configuration after any operation. Such are the 1DFAs [23] or pushdown automata [17].

3 Reversibility and Related Notions

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

Definition 4. *A graph-walking automaton is called* direction-determinate*, if every state is reachable from a unique direction, that is, there exists a partial function* $d\colon Q \to D$*, such that* $\delta(q,a) = (q',d')$ *implies* $d' = d(q')$*. As the direction is always known, the notation for the transition function can be simplified as follows: for each* $a \in \Sigma$*, let* $\delta_a\colon Q \to Q$ *be a partial function defined by* $\delta_a(p) = q$ *if* $\delta(p,a) = (q, d(q))$*.*

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

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

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

Definition 5. *A graph-walking automaton is called* returning*, if it has* $F \subseteq Q \times \Sigma_0$*, that is, if it accepts only at the initial node.*

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

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

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

Definition 6. *A direction-determinate graph-walking automaton is* reversible*, if*

i. every partial function δ_a *is injective, that is,* $|\delta_a^{-1}(q)| \leqslant 1$ *for all* $a \in \Sigma$ *and* $q \in Q$*, and*
ii. the automaton is returning, and for each $a_0 \in \Sigma_0$*, there exists at most one state* q*, such that* $(q, a_0) \in F$ *(this state is denoted by* $q_{\mathrm{acc}}^{a_0}$*).*

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

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

The second case in Lemma 2 allows a reversible automaton to be non-halting, if its initial configuration can be re-entered. This possibility may be ruled out by disallowing any transitions leading to the initial state. Another imperfection of reversible automata is that while they may accept only in a single designated configuration, there are no limitations on where they may reject. Thus, backtracking a rejecting computation is not possible, because it is not known where it ends. The below strengthened definition additionally requires rejection to take place in a unique configuration, analogous to the accepting configuration.

Definition 7. *A* strongly reversible *automaton is a reversible automaton $\mathcal{A} = (Q, q_0, \delta, F)$ with non-reenterable initial state, which additionally satisfies the following conditions:*

iii. for every non-initial label $a \in \Sigma \setminus \Sigma_0$, the partial function δ_a is a bijection from $\{\, p \in Q \mid -d(p) \in D_a \,\}$ to $\{\, q \in Q \mid d(q) \in D_a \,\}$,
iv. for each initial label $a_0 \in \Sigma_0$, there is at most one designated rejecting state $q_{\mathrm{rej}}^{a_0} \in Q$, for which neither $\delta_{a_0}(q_{\mathrm{rej}}^{a_0})$ is defined, nor $(q_{\mathrm{rej}}^{a_0}, a_0)$ is in F,
v. for all $a_0 \in \Sigma_0$ and for all states $q \in Q \setminus \{q_{\mathrm{acc}}^{a_0},\ q_{\mathrm{rej}}^{a_0}\}$, $\delta_{a_0}(q)$ is defined if and only if $-d(q) \in D_{a_0}$ or $q = q_0$.

The requirement on the range of δ_a, with $a \notin \Sigma_0$, in condition (iii) means that if a-labelled nodes have a direction $d \in D_a$ for reaching a state $q \in Q$, then there is a state $p \in Q$, in which this direction d can be used to get to q. The requirement on the domain of δ_a means that this function is defined precisely for those states p, which can be possibly entered in a-labelled nodes, that is, for such states p, that the direction for entering p leads to these nodes. This in particular implies that whenever a computation of a strongly reversible automaton enters a configuration (p, v) with $\lambda(v) \notin \Sigma_0$ (that is, $v \neq v_0$), the next step of the computation is defined and the automaton cannot halt in this configuration. Similarly, condition (v) ensures that the computation cannot halt in the initial node, unless it reaches either the corresponding accepting state $q_{\mathrm{acc}}^{a_0}$ or the corresponding rejecting state $q_{\mathrm{rej}}^{a_0}$. Because the initial state of a strongly reversible automaton is not re-enterable, Lemma 2 guarantees that its computation beginning in the initial configuration always halts, with its head scanning the initial node, and either in the accepting state or in the rejecting state.

Lemma 3. *For every finite input graph $(V, v_0, +, \lambda)$, a strongly reversible graph-walking automaton, starting in the initial configuration, either accepts in the configuration $(q_{\mathrm{acc}}^{\lambda(v_0)}, v_0)$ or rejects in the configuration $(q_{\mathrm{rej}}^{\lambda(v_0)}, v_0)$.*

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

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

Theorems 1–2 and 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 automaton with $6dn + 1$ states recognizing the same set of graphs.*

4 Reversible Simulation of Irreversible Automata

The fundamental construction behind all results of this paper is the following reversible simulation of an arbitrary deterministic graph-walking automaton.

Lemma 4. *For every direction-determinate automaton $\mathcal{A} = (Q, q_0, \delta, F)$ there exists a reversible automaton $\mathcal{B} = (\overrightarrow{Q} \cup [Q], \delta', F')$ without an initial state, where $\overrightarrow{Q} = \{\, \overrightarrow{q} \mid q \in Q \,\}$ and $[Q] = \{\, [q] \mid q \in Q \,\}$ are disjoint copies of Q, with the corresponding directions $d'(\overrightarrow{q}) = d(q)$ and $d'([q]) = -d(q)$, and with acceptance conditions $F' = \left\{\, ([\delta_{a_0}(q_0)], a_0) \;\middle|\; a_0 \in \Sigma_0,\ \delta_{a_0}(q_0) \text{ is defined} \,\right\}$, which has the following property: For every graph $(V, v_0, +, \lambda)$, its node $\widehat{v} \in V$ and a state $\widehat{q} \in Q$ of the original automaton, for which $(\widehat{q}, \lambda(\widehat{v})) \in F$ and $-d(\widehat{q}) \in D_{\lambda(\widehat{v})}$, the computation of $\mathcal{B}$ beginning in the configuration $([\widehat{q}], \widehat{v} - d(\widehat{q}))$,*

- *accepts in the configuration $([\delta_{\lambda(v_0)}(q_0)], v_0)$, if $(\widehat{q}, \widehat{v}) \neq (q_0, v_0)$ and $\mathcal{A}$ accepts this graph in the configuration $(\widehat{q}, \widehat{v})$, as shown in Figure 1 (case 1).*
- *rejects in $(\overrightarrow{\widehat{q}}, \widehat{v})$, otherwise (see Figure 1, case 2).*

Proof (the overall idea). As per Sipser's [24] general approach, the automaton $\mathcal{B}$ searches through the tree of the computations of $\mathcal{A}$ leading to the configuration $(\widehat{q}, \widehat{v})$, until it finds the initial configuration of $\mathcal{A}$ or until it verifies that the initial configuration is not in the tree. While searching, it remembers *a single state* of $\mathcal{A}$, as well as *one bit* of information indicating the current direction of search: a state $[q] \in [Q]$ means tracing the computation in reverse, while in a state $\overrightarrow{q} \in \overrightarrow{Q}$ the computation of $\mathcal{A}$ is simulated forward[1].

[1] To compare, Sipser [24], followed by Muscholl et al. [22], has the simulating automaton remember *two states* of the original automaton, leading to a quadratic size blowup, while Morita's [21] simulation remembers a state and a symbol.

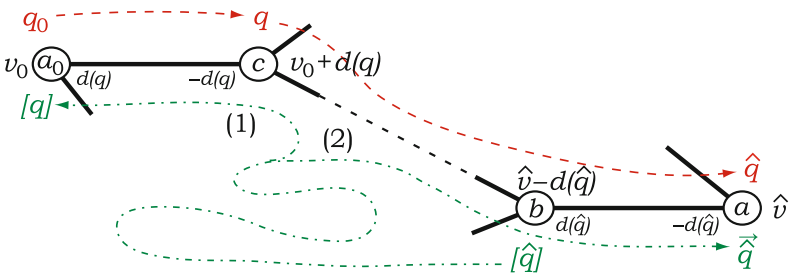

Fig. 1. Reversible GWA $\mathcal{B}$ in Lemma 4 checking whether $\mathcal{A}$ accepts in $(\widehat{q}, \widehat{v})$: (1) if so, accept in $([q], v_0)$, where $q = \delta_{a_0}(q_0)$; (2) otherwise, reject in $(\vec{\widehat{q}}, \widehat{v})$

Whenever $\mathcal{B}$ reaches a state $[q]$ in a node v, this means that the computation of $\mathcal{A}$, beginning in the state q with the head in the neighbouring node $v + d(q)$, eventually leads to the configuration $(\widehat{q}, \widehat{v})$. In this way, 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.

The proofs of both theorems follow from this lemma. In the proof of Theorem 1, 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 4. If $\mathcal{A}$ rejects or loops, the constructed automaton will reject or loop in the same way, as it will never reach the backtracking stage.

In the proof of Theorem 2, a given returning direction-determinate automaton $\mathcal{A}$ is simulated by a reversible automaton $\mathcal{B}$ of Lemma 4.

5 Application to Various Types of Automata

The aim of this section is to revisit several models of computation represented as GWAs in Section 2, and apply the results of this paper to each of them.

Proposition 1. *Each n-state 2DFA has an equivalent $(4n + 3)$-state strongly reversible 2DFA.*

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

In the case of 2DFAs, Theorem 2 is essentially a generalization of the construction by Kondacs and Watrous [16] from 1DFAs to direction-determinate 2DFAs. The transformation of an n-state 2DFA to a 2DFA with $4n + \text{const}$ states that halts on every input, presented by Geffert et al. [13], most likely results in the same reversible automaton as constructed in Proposition 1, but both main steps of the construction are amalgamated into one. Thus, the two-step transformation proving Proposition 1 explains the construction given by Geffert et al. [13].

Turning to tree-walking automata, Muscholl et al. [22] proved that an n-state TWA can be transformed to a halting TWA with $O(n^2)$ states, using another implementation of Sipser's method [24]. This can be now improved as follows.

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

Here the transformation to direction-determinate multiplies the number of states by $|D| = 2k$. Parking the head after acceptance generally requires only one extra state, in which the automaton will go up to the root. However, in order to keep the resulting automaton direction-determinate, one has to use k extra states $q^1_{\text{return}}, \ldots, q^k_{\text{return}}$ with $d(q^i_{\text{return}}) = -i$. Reversibility is ensured by Theorem 2, which produces $2(2kn + k) + 1$ states, as stated.

The next model are the multi-head automata, for which Morita [21] proved 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. The general results of this paper imply a transformation with the constant factor independent of the alphabet.

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

Since there are $3^k - 1$ directions, the transformation to direction-determinate automaton incurs a $(3^k - 1)$-times blowup. Adding k extra states to park all k heads after acceptance produces an automaton with $(3^k - 1)n + k$ states, to which Theorem 2 is applied.

In the full paper, it is similarly shown how to transform an n-state Turing machine operating in marked space $s(\ell)$, with an m-symbol work alphabet, to a $(6(m^2 - m + 4)n + 6m + 16)$-state reversible Turing machine of the same kind. There are also transformations of an n-state 4DFA to a $(8n + 9)$-state strongly reversible 4DFA, and of an n-state k-pebble 2DFA to a $((4k+4)n+2k+5)$-state strongly reversible k-pebble 2DFA. The list of such results can be continued further, by representing various models of computation with a bounded graph of memory configurations as graph-walking automata, and then applying the general theorems of this paper.

References

1. Abramsky, S.: A structural approach to reversible computation. Theoretical Computer Science 347(3), 441–464 (2005)
2. Aho, A.V., Ullman, J.D.: Translations on a context free grammar. Information and Control 19(5), 439–475 (1971)

3. Bennett, C.H.: Logical reversibility of computation. IBM Journal of Research and Development 17(6), 525–532 (1973)
4. Bennett, C.H.: Time/space trade-offs for reversible computation. SIAM Journal on Computing 81, 766–776 (1989)
5. Blum, M., Hewitt, C.: Automata on a 2-dimensional tape. In: SWAT 1967, pp. 155–160 (1967)
6. Bojańczyk, M., Colcombet, T.: Tree-walking automata cannot be determinized. Theoretical Computer Science 350(2-3), 164–173 (2006)
7. Bojańczyk, M., Colcombet, T.: Tree-walking automata do not recognize all regular languages. SIAM Journal on Computing 38(2), 658–701 (2008)
8. Buhrman, H., Tromp, J., Vitányi, P.: Time and space bounds for reversible simulation. Journal of Physics A: Mathematical and General 34(35), 6821–6830 (2001)
9. Courcelle, B.: Graph rewriting: An algebraic and logic approach. In: Handbook of Theoretical Computer Science, vol. B, pp. 193–242 (1990)
10. Crescenzi, P., Papadimitriou, C.H.: Reversible simulation of space-bounded computations. Theoretical Computer Science 143(1), 159–165 (1995)
11. Engelfriet, J., Hoogeboom, H.J.: Tree-walking pebble automata. Jewels are Forever, Contributions on Theoretical Computer Science in Honor of Arto Salomaa, 72–83 (1999)
12. Fraigniaud, P., Ilcinkas, D., Peer, G., Pelc, A., Peleg, D.: Graph exploration by a finite automaton. Theoretical Computer Science 345(2-3), 331–344 (2005)
13. Geffert, V., Mereghetti, C., Pighizzini, G.: Complementing two-way finite automata. Information and Computation 205(8), 1173–1187 (2007)
14. Hopcroft, J.E., Ullman, J.D.: Some results on tape bounded Turing machines. Journal of the ACM 16, 168–177 (1967)
15. Kari, J.: Reversible cellular automata. In: De Felice, C., Restivo, A. (eds.) DLT 2005. LNCS, vol. 3572, pp. 57–68. Springer, Heidelberg (2005)
16. Kondacs, A., Watrous, J.: On the power of quantum finite state automata. In: FOCS 1997, pp. 66–75 (1997)
17. Kutrib, M., Malcher, A.: Reversible pushdown automata. Journal of Computer and System Sciences 78(6), 1814–1827 (2012)
18. Landauer, R.: Irreversibility and heat generation in the computing process. IBM Journal of Research and Development 5(3), 183–191 (1961)
19. Lange, K.-J., McKenzie, P., Tapp, A.: Reversible space equals deterministic space. Journal of Computer and System Sciences 60(2), 354–367 (2000)
20. Lecerf, Y.: Machines de Turing réversibles. Comptes Rendus de l'Académie des Sciences 257, 2597–2600 (1963)
21. Morita, K.: A deterministic two-way multi-head finite automaton can be converted into a reversible one with the same number of heads. In: Glück, R., Yokoyama, T. (eds.) RC 2012. LNCS, vol. 7581, pp. 29–43. Springer, Heidelberg (2013)
22. Muscholl, A., Samuelides, M., Segoufin, L.: Complementing deterministic tree-walking automata. Information Processing Letters 99(1), 33–39 (2006)
23. Pin, J.-É.: On the languages accepted by finite reversible automata. In: Ottmann, T. (ed.) ICALP 1987. LNCS, vol. 267, pp. 237–249. Springer, Heidelberg (1987)
24. Sipser, M.: Halting space-bounded computations. Theoretical Computer Science 10(3), 335–338 (1980)
25. Thomas, W.: On logics, tilings, and automata. In: Leach Albert, J., Monien, B., Rodríguez-Artalejo, M. (eds.) ICALP 1991. LNCS, vol. 510, pp. 441–454. Springer, Heidelberg (1991)
26. Toffoli, T., Margolus, N.H.: Invertible cellular automata: A review. Physica D: Nonlinear Phenomena 45(1-3), 229–253 (1990)