Theoretical Computer Science 411 (2010) 812–822

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Real-time reversible iterative arrays

Martin Kutrib*, Andreas Malcher

Institut für Informatik, Universität Giessen, Arndtstr. 2, 35392 Giessen, Germany

ARTICLE INFO

Keywords:
Iterative arrays
Real-time reversibility
Language recognition
Decidability
Closure properties

ABSTRACT

Iterative arrays are one-dimensional arrays of interconnected interacting finite automata. The cell at the origin is equipped with a one-way read-only input tape. We investigate iterative arrays as acceptors for formal languages. In particular, we consider real-time devices which are reversible on the core of computation, i.e., from initial configuration to the configuration given by the time complexity. This property is called real-time reversibility. It is shown that real-time reversible iterative arrays can simulate restricted variants of stacks and queues. It turns out that real-time reversible iterative arrays are strictly weaker than real-time reversible cellular automata. On the other hand, a non-semilinear language is accepted. We show that real-time reversibility itself is not even semidecidable, which extends the undecidability for cellular automata and contrasts with the general case, where reversibility is decidable for one-dimensional devices. Moreover, we prove the non-semidecidability of several other properties. Several closure properties are also derived.

1. Introduction

Reversibility in the context of computing devices means that deterministic computations are also backward deterministic. Roughly speaking, in a reversible device no information is lost and every configuration occurring in any computation has at most one predecessor. Many different formal models have been studied in connection with reversibility. For example, reversible Turing machines have been introduced in [3], where it is shown that any irreversible Turing machine can be simulated by a reversible one. With respect to the number of tapes and tape symbols the result is significantly improved in [21]. On the opposite end of the automata hierarchy, reversibility in very simple devices, namely deterministic finite automata, has been studied in [2] and [26].

Here we study linear arrays of identical copies of deterministic finite automata. The single nodes, except the node at the origin, are homogeneously connected to both immediate neighbors. Moreover, they work synchronously at discrete time steps. The distinguished cell at the origin, the communication cell, is equipped with a one-way read-only input tape. Such devices are commonly called *iterative arrays* (IA). In connection with formal language recognition, IAs have been introduced in [7]. In [9] a real-time acceptor for prime numbers has been constructed. A characterization of IAs of various types in terms of restricted Turing machines and several results, in particular speed-up theorems, are given in [11]. Some recent results concern infinite hierarchies beyond linear time [13] and between real time and linear time [6], hierarchies depending on the amount of nondeterminism [5], and descriptional complexity issues [19]. Closely related to iterative arrays are *cellular automata* (CA). The main difference is that cellular automata receive their input in parallel. That is, in our setting the input is fed to the cells 0 to $n-1$ in terms of states during a pre-initial step. There is no extra input tape. It is well known that conventional *real-time* cellular automata are strictly more powerful than *real-time* iterative arrays [27]. Our particular interest lies in reversible iterative arrays as acceptors for formal languages. An early result on general reversible CAs is the possibility of making any CA, possibly irreversible, reversible by increasing the dimension. In detail, in [28] it is shown that

* Corresponding author.
E-mail addresses: kutrib@informatik.uni-giessen.de (M. Kutrib), malcher@informatik.uni-giessen.de (A. Malcher).

0304-3975/$ – see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2009.10.017

any k-dimensional CA can be embedded into a $(k+1)$-dimensional reversible CA. Again, this result has been significantly improved by showing how to make irreversible one-dimensional CAs reversible without increasing the dimension [24]. A solution is presented which preserves the neighborhood but increases the time ($O(n^2)$ time for input length n). Furthermore, it is known that even reversible one-dimensional one-way CAs are computationally universal [20,22]. Once a reversible computing device is under consideration, the natural question arises of whether reversibility is decidable. For example, reversibility of a given deterministic finite automaton or of a given regular language is decidable [26]. For cellular automata, injectivity of the global transition function is equivalent to the reversibility of the automaton. It is shown in [1] that global reversibility is decidable for one-dimensional CAs, whereas the problem is undecidable for higher dimensions [14]. Additional information on some aspects of CAs may be found in [15]. All of these results concern cellular automata with unbounded configurations. Moreover, in order to obtain a reversible device the neighborhood as well as the time complexity may be increased. In [8] it is shown that the neighborhood of a reverse CA is at most $n-1$ when the given reversible CA has n states. Additionally, this upper bound is shown to be tight. In connection with pattern recognition, reversible two-dimensional partitioned cellular automata have been investigated in [23,25].

Here we consider iterative arrays that are reversible on the core of computation, i.e., from the initial configuration to the configuration given by the time complexity. Our main interest is in fast computations, i.e., real-time computations. Consequently, we call such devices real-time reversible. Recently, cellular automata have been investigated as regards this aspect [17]. Here we continue this work. In particular, we want to know whether for a given real-time IA there exists a reverse real-time IA with the same neighborhood. At first glance, such a setting should simplify matters. But quite the contrary, we prove that real-time reversibility is not even semidecidable, which extends the undecidability for cellular automata and contrasts with the general case, where reversibility is decidable for one-dimensional devices. Moreover, in Section 5 we prove the non-semidecidability of several other properties. Several closure properties of the language families in question are derived in Section 4, whereas Section 3 is devoted to the simulation of restricted variants of stacks and queues by real-time reversible iterative arrays. It turns out that real-time reversible iterative arrays are strictly weaker than real-time reversible cellular automata. The particularities in connection with reversibility are identified by an example which deals with a non-semilinear language.

2. Real-time reversible iterative arrays

We denote the set of non-negative integers by $\mathbb{N}$. The empty word is denoted by λ, and the reversal of a word w by w^R. For the length of w we write $|w|$. We use $\subseteq$ for inclusions and $\subset$ for strict inclusions. An iterative array is a semi-infinite array of deterministic finite automata, sometimes called cells. Except for the leftmost automaton each one is connected to both nearest neighbors. For convenience we identify the cells by their coordinates, i.e., by non-negative integers. The distinguished leftmost cell at the origin is connected to its right neighbor and, additionally, equipped with a one-way read-only input tape. At the outset of a computation the input is written with an infinite number of end-of-input symbols to the right on the input tape, and all cells are in the so-called quiescent state. The finite automata work synchronously at discrete time steps. The state transition of all cells but the communication cell depends on the current state of the cell itself and the current states of its neighbors. The state transition of the communication cell additionally depends on the input symbol to be read next. The head of the one-way input tape is moved at any step to the right. With an eye towards language recognition the machines have no extra output tape but the states are partitioned into accepting and rejecting states.

Definition 1. An *iterative array* (IA) is a system $\langle S, A, F, s_0, \triangleleft, \delta, \delta_0\rangle$ where S is the finite, nonempty set of *cell states*, A is the finite, nonempty set of *input symbols*, $F \subseteq S$ is the set of *accepting states*, $s_0 \in S$ is the *quiescent state*, $\triangleleft \notin A$ is the *end-of-input symbol*, $\delta_0 : (A \cup \{\triangleleft\}) \times S^2 \to S$ is the *local transition function for the communication cell*, and $\delta : S^3 \to S$ is the *local transition function for non-communication cells* satisfying $\delta(s_0, s_0, s_0) = s_0$.

Let $\mathcal{M}$ be an IA. A configuration of $\mathcal{M}$ at some time $t \geq 0$ is a description of its global state which is a triple (w, p, c_t), where $w \in A^*$ is the input sequence, $p \geq 0$ is the current head position, and $c_t : \mathbb{N} \to S$ is a mapping that maps the single cells to their current states. The configuration $(w, 0, c_0)$ at time 0 is defined by the input word w and the mapping $c_0(i) = s_0$, $i \geq 0$ (Fig. 1), while subsequent configurations are chosen according to the global transition function Δ: Let (w, p, c_t), $t \geq 0$, be a configuration. Then its successor configuration $(w, p+1, c_{t+1})$ is as follows:

$$(w, p+1, c_{t+1}) = \Delta\big((w, p, c_t)\big) \Leftrightarrow \begin{cases} c_{t+1}(i) = \delta\big(c_t(i-1), c_t(i), c_t(i+1)\big), & i \geq 1, \\ c_{t+1}(0) = \delta_0\big(a_t, c_t(0), c_t(1)\big) \end{cases}$$

where $w = a_0 a_1 \cdots a_{n-1}$, and $a_t = \triangleleft$ if $t \geq n$. Thus, the global transition function Δ is induced by δ and δ_0.

A word is accepted by an IA if at some time during its course of computation the communication cell becomes accepting. Let $\mathcal{M} = \langle S, A, F, s_0, \triangleleft, \delta, \delta_0\rangle$ be an IA. A word $w \in A^*$ is *accepted* by $\mathcal{M}$ if there exists a time step $i \geq 1$ such that $c_i(0) \in F$. $L(\mathcal{M}) = \{w \in A^* \mid w \text{ is accepted by } \mathcal{M}\}$ is the *language accepted* by $\mathcal{M}$. Let $t : \mathbb{N} \to \mathbb{N}$, $t(n) \geq n+1$, be a mapping. If all $w \in L(\mathcal{M})$ are accepted with at most $t(|w|)$ time steps, then L is said to be of *time complexity* t.

Now we turn to iterative arrays that are reversible on the core of computation, i.e., from the initial configuration to the configuration given by the time complexity—consequently, we call them t-time reversible if the time complexity t is obeyed. Reversibility is meant with respect to the possibility of stepping the computation back and forth. Due to the domain S^3 and the range S, obviously, the local transition function cannot be injective in general. But for reverse computation steps we may

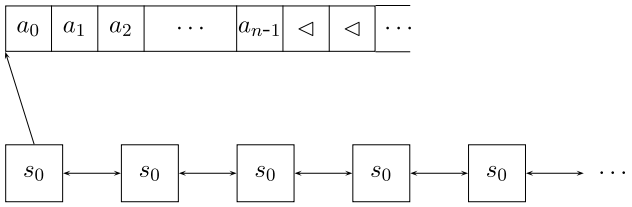

Fig. 1. Initial configuration of an iterative array.

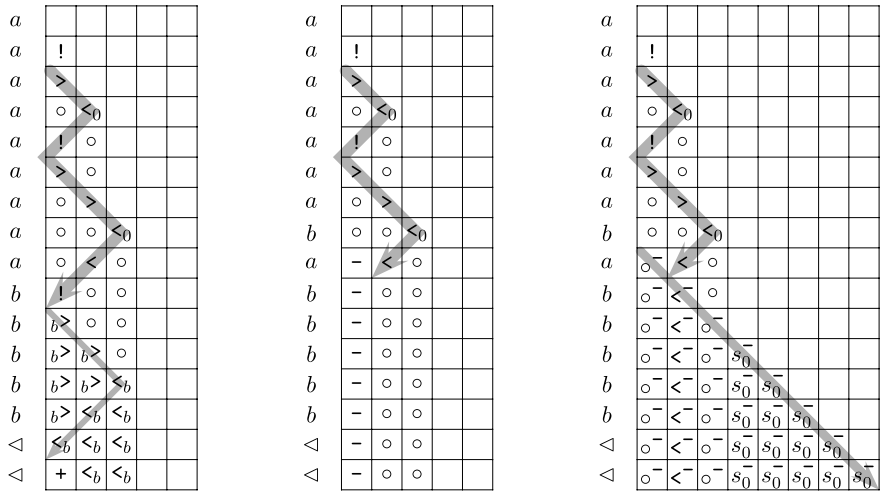

Fig. 2. Real-time IA accepting $\{a^{n^2}b^{2n-1} \mid n \geq 1\}$ (left), not being reversible (middle), rejecting reversibly (right). Cells in the quiescent state are left blank.

utilize the information which is available for the cells, that is, the states of their neighbors. Moreover, for reverse computation steps the head of the input tape is always moved to the *left*. Therefore, the communication cell rereads the input symbol which would have been read in a preceding forward computation step. So, for t-time reversible iterative arrays there must exist reverse local transition functions.

For some mapping $t : \mathbb{N} \to \mathbb{N}$ let $\mathcal{M} = \langle S, A, F, s_0, \lhd, \delta, \delta_0 \rangle$ be a t-time iterative array. Then $\mathcal{M}$ is defined to be *t-reversible* (REV-IA) if there exist reverse local transition functions δ_R and δ_0^R such that $\Delta_R(\Delta((w, i, c_i))) = (w, i, c_i)$, for all configurations (w, i, c_i) of $\mathcal{M}$, $0 \leq i \leq t(|w|) - 1$. The global transition functions Δ and Δ_R are induced by δ, δ_0 and δ_R, δ_0^R, respectively. For distinctness, we denote $\langle S, A, F, s_0, \lhd, \delta_R, \delta_0^R \rangle$ by $\mathcal{M}_R$. The family of all languages that are accepted by some REV-IA with time complexity t is denoted by $\mathscr{L}_t$(REV-IA). If t equals the function $n + 1$, acceptance is said to be in *real time*, and we write $\mathscr{L}_{rt}$(REV-IA).

In order to introduce some of the particularities in connection with reversible language recognition we continue with an example. The goal is to define a real-time REV-IA that accepts the non-context-free language $\{a^{n^2}b^{2n-1} \mid n \geq 1\}$, which is not even semilinear. We start with a conventional iterative array. Basically, the idea is to recognize time steps which are square numbers. To this end, assume that k cells of the array are marked at time k^2. Then a signal can be emitted by the communication cell. The signal moves through the marked area, extends it by one cell, and moves back again. So, the signal arrives at the communication cell at time $(k + 1)^2$. Finally, the number of bs is checked by sending another signal through the marked area and back. Fig. 2 (left) shows an accepting computation. But the transition functions have to be extended in order to reject words not belonging to the language. To this end, we consider possible errors and observe that all errors can be detected by the communication cell. We identify the following errors: (1) the first input symbol is a b, (2) an a follows b, (3) the number of as is not a square number, (4) the number of bs is insufficient, or (5) there are too many bs. Accordingly, we provide rules to cope with these situations. An example of a rejecting computation is given in Fig. 2 (middle). Moreover, in our current construction the whole computation may get frozen before time step $n + 1$, for inputs not belonging to the language. Clearly, this implies non-reversibility. One reason is that for conventional computations we do not care about rejecting computations, except for keeping them rejecting. Nor do we care about the part of the computation that cannot influence the overall result, that is, the computation of cell $i \geq 1$ after time step $n + 1 - i$, i.e., the area below the diagonal starting from the lower left corner of the space-time diagram. For reversible computations we do have to care about rejecting computations as well as for computations in the area mentioned. The idea of our construction is to send a signal from left to right which freezes the computation, whereby each cell passed through has to remember its current state. Clearly, this idea does not work in general. Sometimes much more complicated computations are necessary in order to obtain reversible computations. The complete transition functions of a REV-IA accepting $\{a^{n^2}b^{2n-1} \mid n \geq 1\}$ are presented in Fig. 3.

In order to conclude the construction we present the reverse local transition functions δ_0^R and δ_R in Fig. 4.

δ_0	δ	δ_0	δ
$a\ s_0\ s_0 \to\ !$	$>\ s_0\ s_0 \to <_0$	$b\ s_0\ s_0 \to s_0^-$	$>^-\ s_0\ s_0 \to s_0^-$
$a\ !\ z \to\ >$	$>\ \circ\ y \to\ >$	$a\ {}_b\!>\ z \to {}_b\!>^-$	$>^-\ \circ\ y \to \circ^-$
$a\ >\ z \to\ \circ$	$x\ \circ\ <_0 \to\ <$	$a\ <_b\ z \to <_b^-$	$x^-\ \circ\ <_0 \to \circ^-$
$a\ \circ\ <\ \to\ !$	$x\ \circ\ <\ \to\ <$	$b\ \circ\ z \to \circ^-$	$x^-\ \circ\ <\ \to \circ^-$
$a\ \circ\ <_0 \to\ !$	$x\ >\ z \to\ \circ$	$b\ >\ z \to\ >^-$	$x^-\ >\ z \to\ >^-$
$b\ !\ s_0 \to <_b$	$x\ <_0\ s_0 \to\ \circ$	$b\ <_b\ z \to <_b^-$	$x^-\ <_0\ s_0 \to <_0^-$
$b\ !\ \circ \to\ {}_b\!>$	$x\ <\ z \to\ \circ$	$\lhd\ s_0\ z \to\ s_0^-$	$x^-\ <\ z \to\ <^-$
$b\ {}_b\!>\ <_b \to <_b$	${}_b\!>\ \circ\ \circ \to\ {}_b\!>$	$\lhd\ !\ z \to\ !^-$	${}_b\!>^-\ \circ\ \circ \to \circ^-$
$\lhd\ <_b\ z \to\ +$	${}_b\!>\ \circ\ s_0 \to <_b$	$\lhd\ >\ z \to\ >^-$	${}_b\!>^-\ \circ\ s_0 \to \circ^-$
	$x\ {}_b\!>\ <_b \to <_b$	$\lhd\ \circ\ z \to\ \circ^-$	$x^-\ {}_b\!>\ <_b \to {}_b\!>^-$
		$\lhd\ {}_b\!>\ z \to {}_b\!>^-$	$x^-\ s_0\ s_0 \to s_0^-$

Fig. 3. For convenience, $\delta(p, q, r) = s$ is written as $pqr \to s$, and the same holds for δ_0. By x, z we denote arbitrary states. The two blocks of transition rules at the left are for accepting computations. The third block provides rules for detecting that the input is of wrong format. The rules of the fourth block are for the freezing error signal. An example for a reversible rejecting computation is given in Fig. 2 (right).

δ_0^R	δ_R	δ_0^R	δ_R
$a\ >\ z \to\ !$	$x\ >\ z \to\ \circ$	$j\ y^-\ z \to y$	$x\ y^-\ z \to y$
$a\ \circ\ <_0 \to\ >$	$x\ \circ\ >\ \to\ >$		
$a\ \circ\ >\ \to\ >$	$x\ \circ\ <_0 \to\ >$		
$i\ !\ s_0 \to s_0$	$x\ <\ z \to\ \circ$		
$i\ !\ \circ \to\ \circ$	$\circ\ <_0\ s_0 \to s_0$		
$b\ {}_b\!>\ \circ\ \to\ !$	$<\ \circ\ s_0 \to <_0$		
$\lhd\ <_b\ s_0 \to\ !$	$<\ \circ\ \circ\ \to\ <$		
$\lhd\ <_b\ <_b \to {}_b\!>$	$!\ \circ\ s_0 \to <_0$		
$\lhd\ +\ z \to <_b$	$!\ \circ\ \circ\ \to\ <$		
	$x\ {}_b\!>\ \circ\ \to\ \circ$		
	${}_b\!>\ <_b\ s_0 \to\ \circ$		
	${}_b\!>\ <_b\ <_b \to {}_b\!>$		

Fig. 4. By x, z we denote arbitrary states *without* superscript $^-$. The rules presented are for any $i \in \{a, b\}$ and $j \in \{a, b, \lhd\}$.

3. Reversible simulation of data structures

We next want to explore the computational capacity of real-time REV-IAs. To this end, we first consider the data structures *stack* and *queue*, and show that REV-IAs can simulate special variants thereof. We start with the stack. In detail, we consider real-time deterministic pushdown automata accepting linear context-free languages. Moreover, the stack behavior is restricted in such a way that in every step exactly one symbol is pushed onto or popped from the stack. For convenience, we denote the family of languages accepted by such automata by DLR.

Theorem 2. *Every language from* DLR *belongs to the family* $\mathscr{L}_{rt}$(REV-IA).

Proof. The principal idea is to simulate a stack by using the three-register technique described in [4]. The content of the stack is stored in the first two registers and the third register is used as a buffer. Due to the fact that the given pushdown automaton accepts a linear language, we know that there is at most one change between increasing and decreasing the stack. Thus, the stack behavior can be described as a sequence of push operations followed by a sequence of pop operations in which exactly one stack symbol is pushed or popped, respectively. An example of a computation is shown in Fig. 5. Cells which represent an increasing stack or a decreasing stack are marked with the symbol ↑ or ↓, respectively. When the stack changes from increasing to decreasing, a signal → is sent to the right. With an eye towards reversibility, the communication cell stores a popped symbol on an additional track, and this information is shifted to the right. Thus, the history of the stack content is stored in the cells and can be reconstructed. Finally, the communication cell also simulates the state transition of the pushdown automaton, and stores the states on an additional track which is also shifted to the right.

We now have to show that a computation as described above is reversible. Obviously, shifting to the right can be made reversible by shifting to the left. The first phase of the computation (increasing stack height) is reversible, since in one step exactly one symbol has to be shifted backwards through the three registers in all cells. The second phase of the computation (decreasing stack height) consists of shifting the first register of each cell to the left. Thus, we obtain reversibility by shifting

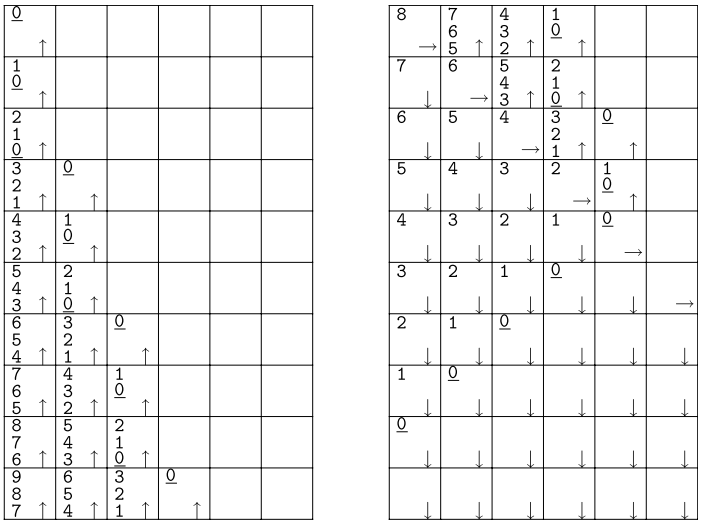

Fig. 5. Pushing (left) and popping (right) of ten pushdown symbols in real time. The left half of each cell contains the three registers for simulating the stack. The first two registers of the right half are used to store the current state of the communication cell and the last popped stack symbol, respectively. The last register indicates whether the stack is increasing (↑), decreasing (↓), or a switch takes place (→). The first entry of the stack is marked by underlining.

the first register of each cell to the right. The signal → is sent to the right and forces each cell to switch from increasing to decreasing. To achieve reversibility here, we send the signal to the left and observe that it meets the first entry of the stack (which is marked suitably) in the rightmost cell, which carries stack symbols in its first registers. In the next time step, cells with increasing stack height can be reconstructed. Finally, due to the history of states and stack contents, the communication cell can always compute its predecessor state. □

Now we can utilize the simulation in order to derive particular languages belonging to the family $\mathscr{L}_{rt}$(REV-IA).

Corollary 3. *Every regular language belongs to the family* $\mathscr{L}_{rt}$(REV-IA).

Example 4. The languages $\{a^n b^n \mid n \geq 1\}$ and $\{wccw^R \mid w \in \{a, b\}^+\}$ are in DLR and thus belong to $\mathscr{L}_{rt}$(REV-IA).

Remark 5. The construction of Theorem 2 can be generalized to simulate real-time deterministic pushdown automata accepting metalinear languages with the above-described stack behavior. Moreover, with similar techniques it is possible to construct a real-time REV-IA which accepts the non-metalinear language $\{a^n b^n \mid n \geq 1\}^+$. An example of a non-semilinear language has been given in the previous section.

Like for the restricted stack behavior, we can show that real-time REV-IAs can simulate queues under certain conditions.

Lemma 6. *Let $\mathcal{Q}$ be an empty queue which is filled by a number of* `in` *operations, and then emptied by a sequence of* `out` *operations. Moreover, in every time step exactly one* `in` *or* `out` *operation is performed. Then $\mathcal{Q}$ can be simulated by a real-time* REV-IA.

Proof. The idea is again to simulate a queue by using the three-register technique. The content of the queue is stored in the first two registers from left to right. The third register is used as a buffer which transports information to the end of the queue, i.e., the first cell with an empty first or second register. Due to the assumption we know that the queue behavior can be described as a sequence of `in` operations followed by a sequence of `out` operations. An example of a computation is shown in Fig. 6. Cells which represent an increasing queue or a decreasing queue are marked with the symbol ↑ and ↓, respectively. The switch from increasing to decreasing is indicated by a signal →, which is sent to the right. Again, in order to preserve the history of the queue contents, the communication cell stores a removed symbol on an additional track, and this information is shifted to the right.

It can be shown like in the proof of Theorem 2 that the above-described computation is reversible. The first phase and second phase of the computation which simulate an increasing and decreasing queue, respectively, are easily observed to be reversible. Due to the history of queue contents, the communication cell can always compute its predecessor state. Finally, the signal → is again reversible. When it meets the last entry of the queue (identified by the border symbol #), cells with increasing queue are reconstructed step by step. □

Example 7. Consider the language $L = \{wcwc \mid w \in \{a, b\}^+\}$. First, the input prefix wc is inserted into a queue. Then, the content of the queue is removed step by step, whereby it is matched against the remaining input wc. Due to Lemma 6, language L belongs to the family $\mathscr{L}_{rt}$(REV-IA).

Remark 8. Again, the construction of Lemma 6 can be generalized in order to simulate iterated queues. Therefore, with similar techniques a real-time REV-IA can be constructed which accepts the language $\{wcwc \mid w \in \{a, b\}^+\}^+$.

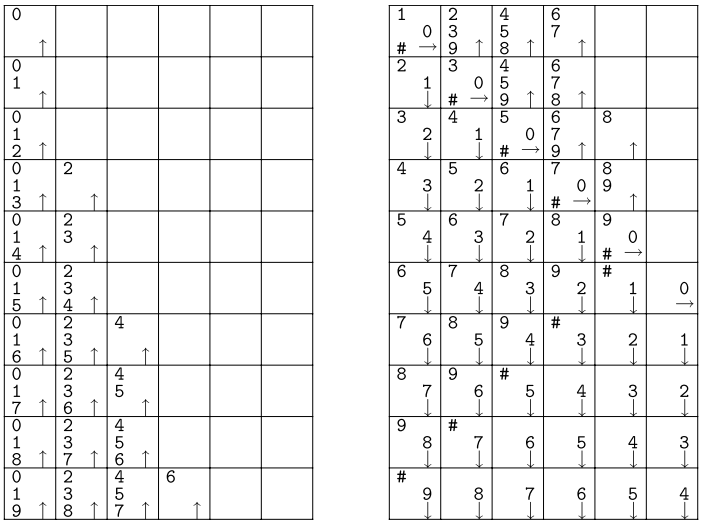

Fig. 6. Insertion (left) and deletion (right) of ten queue symbols in real time. The left half of each cell contains the three registers for simulating the queue. The second register of the right half is used to store the last removed symbol. The last register indicates whether the queue is increasing (↑), decreasing (↓), or a switch takes place (→). The last inserted symbol is followed by a special border symbol #.

Once the stack and queue simulation principle is known, the previous examples are straightforward. But nevertheless, it seems that the simulations mark a sharp boundary between what we can do and cannot do. The restrictions on the data structures seem to be very natural, since in [16] it has been shown that there is a deterministic, linear context-free language not accepted by any conventional real-time iterative array.

Related to iterative arrays are cellular automata. Basically, the difference is that cellular automata receive their input in parallel. That is, in our setting the input is fed to the cells 0 to $n-1$ in terms of states during a pre-initial step. There is no extra input tape. It is well known, and one of the fundamental results, that conventional real-time cellular automata are strictly more powerful than real-time iterative arrays [27]. In [17] reversible language recognition by cellular automata is investigated. The next theorem establishes a reversible relationship equal to the relationship in the conventional case.

Theorem 9. *The family* $\mathscr{L}_{rt}$(REV-IA) *is properly included in the family of languages accepted by real-time reversible cellular automata.*

Proof. We sketch the construction of a real-time reversible cellular automaton $\mathcal{M}$ which accepts the deterministic, linear context-free language $L_d = \{\$x_k\$\cdots\$x_1\#y_1\$\cdots\$y_k\$ \mid 1 \le k, x_i^R = y_i z_i, x_i, y_i, z_i \in \{a,b\}^*, 1 \le i \le k\}$ that is not accepted by any conventional real-time iterative array [16].

Automaton $\mathcal{M}$ uses four tracks (cf. Fig. 7). On the fourth track, the unchanged input is kept. On the second track, basically, the input is shifted to the left. The symbols shifted out of the leftmost cell are stored on the first track which, in turn, is successively shifted to the right. So, no input symbol of the second track gets lost.

The recognition is controlled on the second and third tracks. To this end, the third track is initially filled with • symbols. Every cell passed through by the # symbol possibly needs to be compared with a mate. This is indicated by deleting the •. Now, cells with empty third register wait for the next symbol which is shifted to the left on the second track. If this is a matching mate, that is, it is the same symbol as stored in the fourth register, then the symbol is removed from the second register and stored in the third register. If the incoming symbol is a \$, the comparison of the current subword is completed, a symbol + is stored in the third register, and the \$ continues to move to the left until it reaches a cell with a matching \$. Finally, at the beginning of the shifting a permanent symbol ✓ is generated in the rightmost cell which propagates to the right as long as all comparisons are successful. Otherwise, a symbol − is propagated instead, which indicates an error. So, the input is accepted if and only if the checkmark reaches the leftmost cell. Taking a closer look at the construction immediately shows that $\mathcal{M}$ is real-time reversible. □

4. Closure properties

The technique for sending a signal that freezes the computation in order to maintain reversibility in certain situations yields the closure of the family in question under Boolean operations. A family of languages is said to be *effectively* closed under some operation if the result of the operation can be constructed from the given language(s).

Lemma 10. *The family* $\mathscr{L}_{rt}$(REV-IA) *is effectively closed under the Boolean operations complementation, union, and intersection.*

Proof. A real-time REV-IA accepts an input if and only if the communication cell becomes accepting at any time during the computation. Once this happens, a freezing signal can be sent. So, the communication cell remembers forever that it has accepted. Next, we provide a copy of the non-accepting states, and modify the transition function δ_0 such that it changes to the corresponding new state if and only if the end-of-input symbol appears and the computation is not accepting.

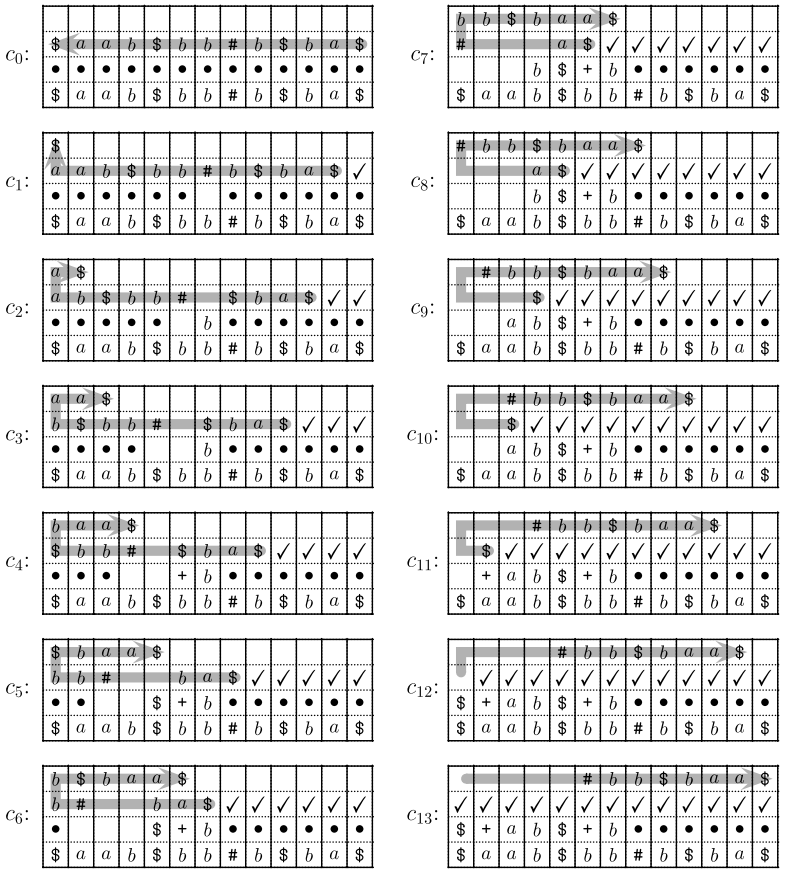

Fig. 7. Example computation of a real-time reversible cellular automaton accepting $\{\$x_k\$\cdots\$x_1\#y_1\$\cdots\$y_k\$ \mid 1 \le k, x_i^R = y_i z_i, x_i, y_i, z_i \in \{a,b\}^*, 1 \le i \le k\}$.

The real-time REV-IA is still reversible. Moreover, it accepts if and only if the state of the communication cell is accepting at time $n+1$, and it rejects if and only if the state of the communication cell is a copied one at time $n+1$. Simply defining the copied states to be accepting states shows the effective closure under complementation.

The closure under union and intersection follows by the well-known two-track technique. When two reversible computations are performed separately on different tracks, clearly, the whole computation is reversible, too. The interpretation of the states of the communication cell on both tracks at time $n+1$ yields the closures. □

Next, we want to show closure under inverse homomorphism. We start with some preliminaries.

Let A and B be two alphabets. The shuffle of two words $x \in A^*$ and $y \in B^*$ is $x \sqcup\!\sqcup y = \{x_1 y_1 \dots x_k y_k \mid x = x_1 \dots x_k, y = y_1 \dots y_k, 1 \le i \le k, k \ge 1\}$. The shuffle of two languages $L \subseteq A^*$ and $L' \subseteq B^*$ is defined as $L \sqcup\!\sqcup L' = \{x \sqcup\!\sqcup y \mid x \in L \text{ and } y \in L'\}$.

Lemma 11. *Let A and B be two disjoint alphabets. If $L \subseteq A^*$ is accepted by a real-time* REV-IA*, then $L \sqcup\!\sqcup B^*$ is accepted by a real-time* REV-IA *as well.*

Proof. Let L be accepted by a real-time REV-IA $\mathcal{M}$. The main idea for the construction of a real-time REV-IA $\mathcal{M}'$ for $L \sqcup\!\sqcup B^*$ is to simulate $\mathcal{M}$ and to ignore input symbols from B. To this end, $\mathcal{M}'$ uses two tracks. Whenever a symbol from B is read the computation has to be frozen. This behavior is realized in the communication cell by marking the current state of $\mathcal{M}$ in its first track suitably, e.g., by putting a bar on it. To keep the computation synchronized all remaining cells must also freeze their computations for one time step. This is realized by sending the freezing signal with maximum speed to the right which forces each cell to change to a barred state. Whenever some cell simulates a computational step of $\mathcal{M}$, its predecessor state is stored in its second component. This gives every cell enough information to restart the simulation whenever some input symbol from A is processed. An example computation is shown in Fig. 8. It can be observed that $\mathcal{M}'$ accepts $L \sqcup\!\sqcup B^*$. Furthermore, since $\mathcal{M}$ is real-time reversible and the freezing signals can be implemented reversibly, we may conclude that $\mathcal{M}'$ is real-time reversible. □

Lemma 12. *The family $\mathscr{L}_{rt}$*(REV-IA) *is closed under inverse homomorphism.*

Proof. Let $\mathcal{M}$ be a real-time REV-IA accepting a language $L \subseteq A^*$, $h : A \to B^*$ be a homomorphism and let $m = \max\{|h(a)| \mid a \in A\}$ be the maximum length of images of h. We now construct an IA which is allowed to make λ-transitions, i.e., it is allowed to make a computational step without reading an input symbol. An IA $\mathcal{M}'$ works as follows on input $a \in A$.

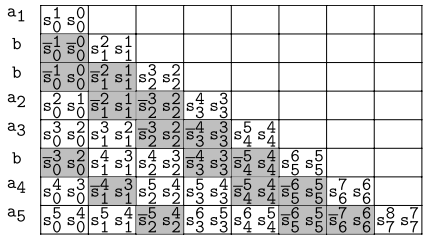

Fig. 8. Reversible implementation of Lemma 11. Frozen cells are marked by barred states and a shaded background.

If $h(a) \neq \lambda$, $\mathcal{M}'$ simulates $\mathcal{M}$ on input $h(a)$. This can be realized with $|h(a)| - 1$ additional λ-transitions after reading a. Then, the communication cell of $\mathcal{M}'$ sends in $m - |h(a)|$ subsequent time steps $m - |h(a)|$ freezing signals to the right according to the construction of Lemma 11. If $h(a) = \lambda$, then the communication cell of $\mathcal{M}'$ sends in m time steps m freezing signals to the right. The accepting states of $\mathcal{M}'$ are those of $\mathcal{M}$. It can be observed that $L(\mathcal{M}') = h^{-1}(L)$ and that $\mathcal{M}'$ is reversible, since $\mathcal{M}$ is real-time reversible and the construction of Lemma 11 preserves reversibility. Furthermore, $\mathcal{M}'$ makes exactly $m - 1$ λ-transitions after reading an input symbol. By grouping m cells into one cell, we obtain an equivalent real-time IA which is in addition real-time reversible, since grouping does not affect the reversibility of the IA. □

Lemma 13. *The family $\mathscr{L}_{rt}$(REV-IA) is closed under marked concatenation and right concatenation with regular languages.*

Proof. To show closure under marked concatenation, consider two languages $L_1, L_2 \in \mathscr{L}_{rt}$(REV-IA) and some separating symbol &. Let L_1 and L_2 be accepted by real-time REV-IAs $\mathcal{M}_1$ and $\mathcal{M}_2$. In order to construct a real-time REV-IA $\mathcal{M}$ accepting $L_1\&L_2$, we consider $\mathcal{M}$ to have two tracks, and start by simulating $\mathcal{M}_1$ on the first track. When reading the separating symbol, some signal is sent to the right with maximum speed which freezes the computation on the first track. In the next time step, the simulation of $\mathcal{M}_2$ is started on the second track. Since $\mathcal{M}_1$ and $\mathcal{M}_2$ are real-time reversible and the freezing signal can be implemented reversibly, we obtain $L_1\&L_2 \in \mathscr{L}_{rt}$(REV-IA).

Basically, the classical construction for accepting the right concatenation of some language L with a regular language R is as follows. All cells simulate the acceptor for L. In addition, the communication cell has a second register which is used to simulate several instances of a deterministic finite automaton A accepting R. Whenever the acceptor for L is in an accepting state, a new instance of A is started. Clearly, by the powerset construction all possible instances can be simulated deterministically in parallel.

In order to simulate this behavior reversibly, all cells use two registers. At every time step, the communication cell sends current states of A as a signal to the right. Obviously, this behavior is reversible and allows one to restore all information in the communication cell. Thus, the family $\mathscr{L}_{rt}$(REV-IA) is closed under right concatenation with regular languages. □

We derive further closure properties using the above-mentioned language $L_d = \{\$x_k\$ \cdots \$x_1\#y_1\$ \cdots \$y_k\$ \mid 1 \leq k, x_i^R = y_iz_i, x_i, y_i, z_i \in \{a, b\}^*, 1 \leq i \leq k\}$ which is known to be not acceptable by any conventional real-time iterative array (cf. [16]).

Lemma 14. *The family $\mathscr{L}_{rt}$(REV-IA) is not closed under reversal, left concatenation with regular languages, and λ-free homomorphism.*

Proof. Since L_d does not belong to $\mathscr{L}_{rt}$(IA), it does not belong to the family $\mathscr{L}_{rt}$(REV-IA), either. On the other hand, the reversal L_d^R belongs to DLR. By Theorem 2 we obtain $L_d^R \in \mathscr{L}_{rt}$(REV-IA) and, thus, $\mathscr{L}_{rt}$(REV-IA) is not closed under reversal.

In contrast to the assertion assume that $\mathscr{L}_{rt}$(REV-IA) is closed under left concatenation with regular languages. The language $L_1 = \{\$x(\$\{a, b\}^+)^k\#\$^k y\$ \mid 1 \leq k, x^R = yz, x, y, z \in \{a, b\}^*\}$ clearly belongs to DLR and, thus, to $\mathscr{L}_{rt}$(REV-IA). Language $L_2 = (\$\{a, b\}^+)^*$ is regular and, therefore, accepted by some real-time REV-IA, too. Due to the assumption $L_3 = L_2L_1\$^*$ belongs also to $\mathscr{L}_{rt}$(REV-IA). Next, language $L_4 = \{(\$\{a, b\}^+)^k\#(\{a, b\}^*\$)^k \mid 1 \leq k\}$ belongs to DLR and therefore it is accepted by some real-time REV-IA. Finally, we obtain $L_5 = L_3 \cap L_4 \in \mathscr{L}_{rt}$(REV-IA) from the closure under intersection. But language $L_5 \subset L$ contains the words used to show $L_d \notin \mathscr{L}_{rt}$(IA). We conclude that $L_5 \notin \mathscr{L}_{rt}$(REV-IA), a contradiction.

Next, by way of contradiction we assume that $\mathscr{L}_{rt}$(REV-IA) is closed under λ-free homomorphism. Consider the regular language $L_2' = (\$\{a, b\}^+)^*\$\{a, b\}^*$. Let & be a new alphabet symbol. By Lemma 13 we obtain that $L_3' = L_2'\&L_1\$^*$ belongs to $\mathscr{L}_{rt}$(REV-IA). Consider two λ-free homomorphisms h_a and h_b mapping from $\{a, b, \$, \#, \&\}$ to $\{a, b, \$, \#\}$. We define $h_a(\&) = a$, $h_b(\&) = b$, and, for $u \in \{a, b, \$, \#\}$, $h_a(u) = h_b(u) = u$. Since $\mathscr{L}_{rt}$(REV-IA) is closed under union by Lemma 10 and closed under λ-free homomorphisms by assumption, we obtain $L_1\$^* \cup h_a(L_3') \cup h_b(L_3') \in \mathscr{L}_{rt}$(REV-IA). Since $L_1\$^* \cup h_a(L_3') \cup h_b(L_3') = L_3$, we obtain $L_3 \in \mathscr{L}_{rt}$(REV-IA) which leads to a contradiction analogous to that of the proof of non-closure under left concatenation with regular sets. □

5. Decidability questions

Now we turn to exploring undecidable properties for real-time REV-IAs. To this end, we consider *valid computations of Turing machines* [10]. Roughly speaking, these are histories of accepting Turing machine computations. It suffices to consider deterministic Turing machines with a single tape and a single read–write head. Without loss of generality and for technical reasons, one can assume that any accepting computation has at least three and, in general, an odd number of steps. Therefore,

it is represented by an even number of configurations. Moreover, it is assumed that the Turing machine cannot print blanks, and that a configuration is halting if and only if it is accepting. The language accepted by some machine $\mathcal{M}$ is denoted by $L(\mathcal{M})$.

Let S be the state set of some Turing machine $\mathcal{M}$, where s_0 is the initial state, $T \cap S = \emptyset$ is the tape alphabet containing the blank symbol, $A \subset T$ is the input alphabet, and $F \subseteq S$ is the set of accepting states. Then a configuration of $\mathcal{M}$ can be written as a word of the form T^*ST^* such that $t_1 \cdots t_i s t_{i+1} \cdots t_n$ is used to express that $\mathcal{M}$ is in state s, scanning tape symbol t_{i+1}, and t_1 to t_n is the support of the tape inscription. The set of valid computations VALC($\mathcal{M}$) is now defined to be the set of words of the form $w_1\#\#\#\#w_2\#\#\#\# \cdots \#\#\#\#w_{2m}\#\#\#\#$, where $m \geq 2$, $\# \notin T \cup S$, $w_i \in T^*ST^*$ are configurations of $\mathcal{M}$, w_1 is an initial configuration of the form s_0A^*, w_{2m} is an accepting configuration of the form T^*FT^*, and w_{i+1} is the successor configuration of w_i, with $0 \leq i \leq 2m-1$. The set of *invalid computations* INVALC($\mathcal{M}$) is the complement of VALC($\mathcal{M}$) with respect to the alphabet $\{\#\} \cup T \cup S$. The following lemma is the key tool for proving undecidability properties for real-time REV-IAs.

Lemma 15. *Let $\mathcal{M}$ be a Turing machine. Then the set* VALC($\mathcal{M}$) *can be represented as the intersection of two languages from* $\mathscr{L}_{rt}$(REV-IA).

Proof. Let L_3 be the language $\{y\#\#\#\#z\#\#\#\# \mid z \text{ is successor of } y\}$. Then VALC($\mathcal{M}$) is equal to the intersection $L_1 \cap L_2$, where $L_1 = L_3^+$ and $L_2 = s_0A^*\#\#\#\#L_3^*T^*FT^*\#\#\#\#$. We first describe how L_3 can be accepted by a real-time REV-IA. The principal idea is to read y, to compute its successor configuration y', and to store y' in a queue $\mathcal{Q}$. Then y' is matched against the input z. If $y' = z$, then the input is accepted, and otherwise it is rejected. To compute y' from y, we consider the four possible steps of $\mathcal{M}$: (1) ZqX is replaced by pZY if $\mathcal{M}$ writes Y and moves the head to the left, (2) $Zq\#$ is replaced by $pZY\#$ if $\mathcal{M}$ writes Y, extending the support of the configuration at the right, and moves the head to the left, (3) qX is replaced by Yp if $\mathcal{M}$ writes Y and moves the head to the right, (4) $q\#$ is replaced by $Yp\#$ if $\mathcal{M}$ writes Y, extending the support of the configuration at the left, and moves the head to the right. Thus, a string of length at most 3 is replaced by some string of length at most 4. Since $\mathcal{M}$ is deterministic, we know which of the above four cases applies. We add to the communication cell two buffers (`buffer1`, `buffer2`) of length 3 and 4, respectively. Now, the input is read and the first three input symbols are written into `buffer1`. Any next input symbol is written into the third place of `buffer1` and the contents of the third and second place are shifted to the second and first place, respectively. The content of the first place is inserted into the queue. If `buffer1` contains some triple to which δ can be applied, we write the result of the replacement into `buffer2`. While reading the next input symbols, the first place of `buffer2` is inserted into the queue, all other places are shifted to the left, and the last read input symbol is written into the rightmost place of `buffer2`. It should be remarked that the handling of `buffer2` is depending on which case has to be simulated. If the fourth symbol # is read, we start to empty the queue and match the input with the queue. We observe that after filling `buffer1` within the first three time steps, there is exactly one `in` or `out` operation in $\mathcal{Q}$. Due to Lemma 6, we know that $\mathcal{Q}$ can be implemented reversibly. Finally, since the management of the buffers and the checking of the correct format take place in the communication cell only, the computation can be made reversible by storing a "protocol" of their states on an additional track similar to the construction in Theorem 2. Thus, we obtain that $L_3 \in \mathscr{L}_{rt}$(REV-IA). By a simple extension of the queue simulation we obtain $L_1 = L_3^+ \in \mathscr{L}_{rt}$(REV-IA).

To accept L_2, we consider the above-constructed IA and check in the communication cell whether the input starts with a string of the form $s_0A^*\#\#\#\#$. If so, the simulation of the queue is started. Otherwise, the input is rejected. To check that the input has a suffix of the form $T^*FT^*\#\#\#\#$, we implement a deterministic finite automaton $\mathcal{A}$ in the communication cell, which starts after every substring #### to check whether the next input is of the form $T^*FT^*\#\#\#\#$. If the end-of-input symbol is read and $\mathcal{A}$ is in an accepting state, then the input is accepted, and otherwise it is rejected. □

From the closure of $\mathscr{L}_{rt}$(REV-IA) under intersection we obtain the following corollary.

Corollary 16. *Let $\mathcal{M}$ be a Turing machine. Then the set* VALC($\mathcal{M}$) *belongs to the family* $\mathscr{L}_{rt}$(REV-IA).

Now we are prepared to gather undecidability results. In fact, we obtain not even semidecidability.

Theorem 17. *Emptiness, finiteness, infiniteness, universality, inclusion, equivalence, regularity, and context-freedom are not semidecidable for real-time* REV-IA*s.*

Proof. Let $\mathcal{M}$ be a Turing machine. By simple pumping arguments it can be shown that VALC($\mathcal{M}$) is context-free if and only if $\mathcal{M}$ accepts a finite set. The finiteness problem of Turing machines is known to be not semidecidable. If, e.g., regularity were semidecidable for real-time REV-IAs, then we could semidecide whether a real-time REV-IA accepting VALC($\mathcal{M}$) accepts a regular language. Thus, we could semidecide the finiteness of Turing machines which is a contradiction. Similarly, the problems of emptiness, finiteness, inclusion, equivalence, and context-freedom can be proven to be not semidecidable for real-time REV-IAs. If VALC($\mathcal{M}$) is infinite, then $\mathcal{M}$ accepts an infinite set. Thus, infiniteness is also not semidecidable. Since $\mathscr{L}_{rt}$(REV-IA) is closed under complementation, universality is not semidecidable as well. □

It is shown in [18] that there cannot exist pumping lemmas or minimization algorithms for cellular automata. The proofs rely on the fact that infiniteness and emptiness are not semidecidable for cellular automata. Thus, we immediately obtain the following two statements.

Corollary 18. *The family* $\mathscr{L}_{rt}$(REV-IA) *and each language family containing* $\mathscr{L}_{rt}$(REV-IA) *do not possess a pumping lemma.*

Corollary 19. *For real-time* REV-IA*s there is no minimization algorithm converting an arbitrary real-time* REV-IA *to an equivalent real-time* REV-IA *which has a minimal number of states.*

Finally, we prove that real-time reversibility itself is not semidecidable.

Theorem 20. *Let $\mathcal{M}$ be a real-time* IA*. It is not semidecidable whether or not $\mathcal{M}$ is real-time reversible.*

Proof. Let $\mathcal{M}'$ be a real-time REV-IA. We consider a real-time IA $\mathcal{M}''$ accepting the language

$$\{w\#\#va^{4(|w|+2)} \mid w \in L(\mathcal{M}'),\ v = \lambda \text{ if } |w| \text{ is even, and } v = \#a^4 \text{ if } |w| \text{ is odd}\},$$

where a and # are new alphabet symbols. We show that $\mathcal{M}''$ is reversible if and only if $L(\mathcal{M}')$ is empty. Since emptiness is not semidecidable for real-time reversible IAs, we obtain that reversibility is not semidecidable for real-time IAs.

The construction of $\mathcal{M}''$ may be sketched as follows. We consider four tracks. The correct input format is checked in the communication cell. On the first track the correct number of as is verified. To this end, all input symbols up to the first a are stored in a queue which uses $n = (|w| + 2)/2$ cells. In detail, we implement four copies of an empty queue (queue1, ...,queue4) and insert all incoming symbols into queue1. When reading the first a we start to empty queue1 and insert all deleted symbols from queue1 into queue2. When queue1 is empty, we start to empty queue2 and insert the symbols into queue3. Then, queue3 is copied into queue4 and finally queue4 is emptied. When reading the end-of-input symbol we know whether the number of as is correct, and can accept or reject. Clearly, the computation on the first track is reversible.

The second track is used to store the input $w\#\#$ with $w = a_1a_2 \dots a_{|w|}$ in a stack. Observe that a_1 arrives in cell $n - 1$ at time $|w| + 2 + n$. Subsequently, the simulation of $\mathcal{M}'$ is started on the third track from right to left, i.e., cell $n - 1$ serves as the communication cell which gets its input from the stack stored on the second track. Additionally, two cells of $\mathcal{M}'$ are packed into one cell of the third track. Observe that the simulation takes time $|w| + 1$ and that we can decide after at most $2|w| + 3 + n$ time steps in cell $n - 1$ whether or not the input w is accepted in $\mathcal{M}'$. We want to achieve that cell $n - 1$ accepts or rejects after exactly $2(|w| + 2) + n$ time steps. This can be realized reversibly by using techniques similar to those in Lemma 10. Thus, we can observe that the computation on the second and third track is reversible, because a stack can be simulated reversibly and $\mathcal{M}'$ is reversible.

If w is accepted, then we send a signal with speed 1/5 to the left which causes each cell to enter some new permanent state g. Obviously, g erases any information from the cells. The communication cell changes to state g at time $5(|w| + 2)$. If the input has the correct format and the correct number of as, then we accept the input, and we reject it in all other cases. Thus, $w \in L(\mathcal{M}')$ results in an accepting, non-reversible computation of $\mathcal{M}''$.

When the first a is read, a reversible version of the Firing Squad Synchronization Problem (FSSP) according to the construction given in [12] is started on the fourth track. At time $2(|w| + 2) + n$ the cells $0, \dots, n - 1$ are synchronized and change synchronously to some states which preserve the current contents of their second and third tracks. On the fourth track the reverse FSSP is started. So, the whole computation is reversible as long as no state g occurs. Thus, $w \notin L(\mathcal{M}')$ results in a non-accepting, reversible computation of $\mathcal{M}''$. Altogether, we obtain that $\mathcal{M}''$ is reversible if and only if $L(\mathcal{M}')$ is empty. □

References

[1] S. Amoroso, Y.N. Patt, Decision procedures for surjectivity and injectivity of parallel maps for tesselation structures, J. Comput. System Sci. 6 (1972) 448–464.
[2] D. Angluin, Inference of reversible languages, J. ACM 29 (1982) 741–765.
[3] C.H. Bennet, Logical reversibility of computation, IBM J. Res. Dev. 17 (1973) 525–532.
[4] Th Buchholz, M. Kutrib, Some relations between massively parallel arrays, Parallel Comput. 23 (1997) 1643–1662.
[5] Th Buchholz, A. Klein, M. Kutrib, Iterative arrays with limited nondeterministic communication cell, in: Words, Languages and Combinatorics III, World Scientific Publishing, 2003, pp. 73–87.
[6] Th Buchholz, A. Klein, M. Kutrib, Iterative arrays with small time bounds, in: Mathematical Foundations of Computer Science, MFCS 1998, in: LNCS, vol. 1893, Springer, 2000, pp. 243–252.
[7] S.N. Cole, Real-time computation by n-dimensional iterative arrays of finite-state machines, IEEE Trans. Comput. C-18 (1969) 349–365.
[8] E. Czeizler, J. Kari, A tight linear bound on the neighborhood of inverse cellular automata, in: International Colloquium on Automata, Languages and Programming, ICALP 2005, in: LNCS, vol. 3580, Springer, 2005, pp. 410–420.
[9] P.C. Fischer, Generation of primes by a one-dimensional real-time iterative array, J. ACM 12 (1965) 388–394.
[10] J. Hartmanis, Context-free languages and Turing machine computations, Proc. Sympos. Appl. Math. 19 (1967) 42–51.
[11] O.H. Ibarra, M.A. Palis, Some results concerning linear iterative (systolic) arrays, J. Parallel Distrib. Comput. 2 (1985) 182–218.
[12] K. Imai, K. Morita, Firing squad synchronization problem in reversible cellular automata, Theoret. Comput. Sci. 165 (1996) 475–482.
[13] C. Iwamoto, T. Hatsuyama, K. Morita, K. Imai, Constructible functions in cellular automata and their applications to hierarchy results, Theoret. Comput. Sci. 270 (2002) 797–809.
[14] J. Kari, Reversibility and surjectivity problems of cellular automata, J. Comput. System Sci. 48 (1994) 149–182.
[15] J. Kari, Theory of cellular automata: A survey, Theoret. Comput. Sci. 334 (2005) 3–33.
[16] M. Kutrib, Automata arrays and context-free languages, in: Where Mathematics, Computer Science and Biology Meet, Kluwer Academic Publishers, 2001, pp. 139–148.
[17] M. Kutrib, A. Malcher, Fast reversible language recognition using cellular automata, Inform. and Comput. 206 (2008) 1142–1151.
[18] A. Malcher, Descriptional complexity of cellular automata and decidability questions, J. Autom., Lang. Comb. 7 (2002) 549–560.
[19] A. Malcher, On the descriptional complexity of iterative arrays, IEICE Trans. Inf. Syst. E87-D (2004) 721–725.
[20] K. Morita, M. Harao, Computation universality of one dimensional reversible injective cellular automata, Trans. IEICE E72 (1989) 758–762.
[21] K. Morita, A. Shirasaki, Y. Gono, A 1-tape 2-symbol reversible Turing machine, Trans. IEICE E72 (1989) 223–228.
[22] K. Morita, Computation-universality of one-dimensional one-way reversible cellular automata, Inform. Process. Lett. 42 (1992) 325–329.

[23] K. Morita, S. Ueno, Parallel generation and parsing of array languages using reversible cellular automata, Int. J. Pattern Recognit. Artif. Intell. 8 (1994) 543–561.
[24] K. Morita, Reversible simulation of one-dimensional irreversible cellular automata, Theoret. Comput. Sci. 148 (1995) 157–163.
[25] K. Morita, S. Ueno, K. Imai, Characterizing the ability of parallel array generators on reversible partitioned cellular automata, Int. J. Pattern Recognit. Artif. Intell. 13 (1999) 523–538.
[26] J.E. Pin, On reversible automata, in: Theoretical Informatics, LATIN 1992, in: LNCS, vol. 583, Springer, 1992, pp. 401–416.
[27] A.R. Smith III, Real-time language recognition by one-dimensional cellular automata, J. Comput. System Sci. 6 (1972) 233–253.
[28] T. Toffoli, Computation and construction universality of reversible cellular automata, J. Comput. System Sci. 15 (1977) 213–231.