I F I G
Research
Report

Institut für Informatik

Self-Verifying Cellular Automata

Martin Kutrib Thomas Worsch

IFIG Research Report 1803
April 2018

Institut für Informatik
JLU Gießen
Arndtstraße 2
35392 Giessen, Germany
Tel: +49-641-99-32141
Fax: +49-641-99-32149
mail@informatik.uni-giessen.de
www.informatik.uni-giessen.de

IFIG Research Report
IFIG Research Report 1803, April 2018

Self-Verifying Cellular Automata

Martin Kutrib[1]

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

and

Thomas Worsch[2]

Karlsruhe Institute of Technology

Abstract. We study the computational capacity of self-verifying cellular automata with an emphasis on one-way information flow (SVOCA). A self-verifying device is a nondeterministic device whose nondeterminism is symmetric in the following sense. Each computation path can give one of the answers *yes*, *no*, or *do not know*. For every input word, at least one computation path must give either the answer *yes* or *no*, and the answers given must not be contradictory. We show that realtime SVOCA are strictly more powerful than realtime deterministic one-way cellular automata, since they can accept non-semilinear unary languages. It turns out that SVOCA can strongly be sped-up from lineartime to realtime. They are even capable to simulate any lineartime computation of deterministic two-way cellular automata. Closure properties and decidability problems are considered as well.

Categories and Subject Descriptors according to ACM Computing Classification System:
F.1.1 [**Computation by Abstract Devices**]: Models of Computation – *Cellular Automata*;
F.1.2 [**Computation by Abstract Devices**]: Models of Computation – *Parallelism and concurrency*;
F.1.3 [**Computation by Abstract Devices**]: Models of Computation – *Complexity Measures and Classes – Complexity hierarchies*;
F.4.3 [**Mathematical Logic and Formal Languages**]: Formal Languages – *Classes Defined by Grammars or Automata*;
F.4.3 [**Mathematical Logic and Formal Languages**]: Formal Languages – *Decision problems*

MSC Classification: 68Q45 Formal languages and automata

Additional Key Words and Phrases: One-Way Cellular Automata, Self-Verification, Computational Capacity, Closure Properties, Decidability Problems;

[1] Email: kutrib@informatik.uni-giessen.de
[2] Email: worsch@kit.edu

1 Introduction

One of the central questions in complexity and language theory asks for the power of nondeterminism in bounded-resource computations. Traditionally, nondeterministic devices have been viewed as having as many nondeterministic guesses as time steps. The studies of this concept of unlimited nondeterminism led, for example, to the famous open LBA-problem or the unsolved question whether or not P equals NP. In order to gain further understanding of the nature of nondeterminism, in [9, 19] it has been viewed as an additional limited resource at the disposal of time or space bounded computations.

Here we study the computational capacity of self-verifying cellular automata (SVCA). A self-verifying device is a nondeterministic device whose nondeterminism is symmetric in the following sense. Each computation path can give one of the answers *yes*, *no*, or *do not know*. For every input word, at least one computation path must give either the answer *yes* or *no*, and the answers given must not be contradictory. So, if a computation path gives the answer *yes* or *no*, in both cases the answer is definitely correct. This justifies the notion *self-verifying* and is in contrast to general nondeterministic computations, where an answer that is not *yes* does not allow to conclude whether or not the input belongs to the language.

Self-verifying finite automata have been introduced and studied in [6, 11, 12] mainly in connection with randomized Las Vegas computations. Descriptional complexity issues for self-verifying finite automata have been studied in [17]. In particular the conversion of self-verifying finite automata to deterministic finite automata from a state complexity point of view is solved there. The main results are matching upper and lower bounds growing like $3^{n/3}$ for the costs, in terms of the number of states, of such simulations.

Another question that motivates the concept of self-verification is as follows. Given a language such that also its complement belongs to the same family, the description of which of both is more economic [15]? For example, in [16] it is shown that a nondeterministic finite automaton can require 2^n states to accept the complement of a language accepted by an n-state nondeterministic finite automaton. So, a representation of the complement by the n-state automaton together with a bit that says that actually the complement of the language accepted is meant is much more economic.

Recently, the computational and descriptional complexity of self-verifying pushdown automata has been studied in [8].

The paper is organized as follows. In Section 2 we present the basic notation and the definition of self-verifying (one-way) cellular automata as well as an introductory example. In Section 3 a strong speed-up result is shown that allows to speed-up any lineartime computation of an SVOCA to realtime. Section 4 is devoted to explore the computational capacity of realtime SVOCA. It turns out that they are even capable to simulate any lineartime computation of a two-way cellular automaton. Moreover, the closure properties of the family of languages accepted by realtime SVOCA are studied. It is shown that the family is closed under the set-theoretic operations, reversal, concatenation, and inverse homomorphisms. Finally, decidability problems are considered. In particular, the property of being self-verifying turns out to be non-semidecidable.

2 Preliminaries

We denote the positive integers $\{1, 2, \dots\}$ by $\mathbb{N}$, the set $\mathbb{N} \cup \{0\}$ by $\mathbb{N}_0$, and the *powerset* of a set S by 2^S. We write $|S|$ for the *cardinality* of S. Let Σ denote a finite set of letters. Then we write Σ^* for the *set of all finite words* (strings) consisting of letters from Σ. The *empty word* is denoted by λ, and we set $\Sigma^+ = \Sigma^* \setminus \{\lambda\}$. For the *reversal of a word* w we write w^R and for its

length we write $|w|$. A subset of Σ^+ is called a *language* over Σ. Note that the devices we will consider cannot accept the empty word. So, in order to avoid technical overloading in writing, two languages L and L' are considered to be equal, if they differ at most by the empty word, that is, if $L \setminus \{\lambda\} = L' \setminus \{\lambda\}$. Set *inclusion* is denoted by $\subseteq$ and *strict set inclusion* by $\subset$.

A two-way cellular automaton is a linear array of identical finite automata, called cells. Except for the outermost cells, each cell is connected to its both nearest neighbors. For our convenience we identify the cells by positive integers. The state transition depends on the current state of a cell itself and the current state of its neighbors, where the outermost cells receive a boundary symbol on their free input lines. The state changes take place simultaneously at discrete time steps.

Here we first define *nondeterministic* cellular automata whose nondeterminism is restricted to the first state transition. All further transitions are deterministic [2, 21]. Although this is a very restricted case, for easier writing we call such devices nondeterministic.

A *nondeterministic two-way cellular automaton* (NCA) is a system $M = \langle S, \Sigma, F, \#, \delta_{nd}, \delta_d \rangle$, where

1. S is the finite, nonempty set of *cell states*,
2. $\Sigma \subseteq S$ is the nonempty set of *input symbols*,
3. $F \subseteq S$ is the set of *accepting states*,
4. $\# \notin S$ is the *boundary symbol*,
5. $\delta_{nd} \colon (S \cup \{\#\}) \times S \times (S \cup \{\#\}) \to (2^S \setminus \emptyset)$ is the *nondeterministic local transition function* applied in the first state transition,
6. $\delta_d \colon (S \cup \{\#\}) \times S \times (S \cup \{\#\}) \to S$ is the *deterministic local transition function* applied in all further state transitions.

If the flow of information is restricted to one-way, the resulting device is a one-way cellular automaton (NOCA). In such devices the next state of each cell depends on the state of the cell itself and the state of its immediate neighbor to the right. So the domain of the transition functions is $S \times (S \cup \{\#\})$.

A *configuration* c_t of M at time $t \geq 0$ is a description of its global state, which is formally a mapping $c_t : \{1, 2, \ldots, n\} \to S$, for $n \geq 1$. The configuration at time 0, the so-called *initial configuration*, is defined by the given input $w = a_1 a_2 \cdots a_n \in \Sigma^+$. We set $c_0(i) = a_i$, for $1 \leq i \leq n$. Configurations may be represented as words over the set of cell states in their natural ordering. For example, the initial configuration of an NOCA for w is represented by $a_1 a_2 \cdots a_n$. Successor configurations are computed according to the global transition function Δ mapping each configuration to a set of successor configurations.

Let c_t, $t \geq 0$, be a configuration with $n \geq 1$. Then the set of its possible successor configurations c_{t+1} is defined as follows:

$$c_{t+1} \in \Delta(c_t) \iff \begin{cases} c_{t+1}(1) \in \sigma(\#, c_t(1), c_t(2)) \\ c_{t+1}(i) \in \sigma(c_t(i-1), c_t(i), c_t(i+1)), \text{ for } i \in \{2, \ldots, n-1\} \\ c_{t+1}(n) \in \sigma(c_t(n-1), c_t(n), \#) \end{cases}$$

for CA, and

$$c_{t+1} \in \Delta(c_t) \iff \begin{cases} c_{t+1}(i) \in \sigma(c_t(i), c_t(i+1)), \text{ for } i \in \{1, \ldots, n-1\} \\ c_{t+1}(n) \in \sigma(c_t(n), \#) \end{cases}$$

for OCA, where $\sigma = \delta_{nd}$ if $t = 0$, and $\sigma = \delta_d$ if $t \geq 1$. Thus, Δ is induced by δ_{nd} and δ_d.

An NCA (NOCA) is *deterministic* if $\delta_{nd}(s_1, s_2, s_3)$ ($\delta_{nd}(s_1, s_2)$) is a singleton for all states $s_1, s_2, s_3 \in S \cup \{\#\}$. Deterministic cellular automata are denoted by CA and OCA.

Fig. 1. A one-way cellular automaton.

An input w is *accepted* by a cellular automaton if at some time step during the course of a computation the leftmost cell enters an accepting state, that is, the leftmost symbol of some reachable configuration is an accepting state. The *language accepted by* M is denoted by $L(M) = \{ w \in \Sigma^+ \mid w \text{ is accepted by } M \}$. Let $t : \mathbb{N} \to \mathbb{N}$ be a mapping. If all $w \in L(M)$ are accepted with at most $t(|w|)$ time steps, then M is said to be of time complexity t (cf. [20] for a discussion of this general treatment of time complexity functions). Since in general NOCA do not halt, this implies also that all $w \notin L(M)$ are not accepted, that is, rejected at time $t(|w|)$.

If $t(n) = n$ acceptance is said to be in *realtime*. If $t(n)$ is equal to $k \cdot n$ for an arbitrary rational number $k \geq 1$, then acceptance is in *lineartime*.

In general, a family of all languages that are accepted by a device X with time complexity t is denoted by $\mathscr{L}_t(X)$. The index is omitted for arbitrary time. Actually, arbitrary time is exponential time due to the space bound. We write $\mathscr{L}_{rt}(X)$ for real time and $\mathscr{L}_{lt}(X)$ for linear time.

Now we turn to *self-verifying* (one-way) cellular automata (denoted SVCA and SVOCA). These are nondeterministic CA insofar as during the first step cells may choose between several new states. But the definition of acceptance is different from nondeterministic CA.

There are now three disjoint sets of states representing answers *yes*, *no*, and *do not know*. Moreover, for every input word, at least one computation path must give either the answer *yes* or *no*, and the answers given must not be contradictory, that is, for no input there are two computation paths giving the answers *yes* and *no* respectively.

In order to implement the three possible answers the state set is partitioned into three disjoint subsets $S = F_+ \dot\cup F_- \dot\cup F_0$, where F_+ is the set of accepting states, F_- is the set of rejecting states, and $F_0 = S \setminus (F_+ \cup F_-)$ is referred to as the set of neutral states. We specify F_+ and F_- in place of the set F in the definition of an SVCA and SVOCA. So, let $M = \langle S, \Sigma, F_+, F_-, \#, \delta_{nd}, \delta_d \rangle$ be an SVOCA and, for each input word $w \in \Sigma^+$, the set S_w of states reachable by the leftmost cell be defined as $S_w = \{ s \in S \mid s \in \big(\Delta^{[t]}(w\#)\big)\,(1) \text{ for some } t \geq 0 \}$, where $\Delta^{[t]}$ denotes the t-fold composition of Δ, that is, the set of configurations reachable in t time steps. Formally, for the "self-verifying property" it is required that for each $w \in \Sigma^+$, $S_w \cap F_+$ is empty if and only if $S_w \cap F_-$ is nonempty.

If all $w \in L(M)$ are accepted and all $w \notin L(M)$ are rejected with at most $t(|w|)$ time steps, then the self-verifying cellular automaton M is said to be of time complexity t.

In order to illustrate the definitions we continue with an example.

Example 1. The non-semilinear unary language $\{\, a^{2^n} \mid n \geq 0 \,\}$ is accepted by the SVOCA $M = \langle \{a, -, 1, X, \sim, <_1, <_2, \ominus, \otimes, \oplus, 0\}, \{a\}, F_+, F_-, \#, \delta_{nd}, \delta_d \rangle$ in realtime, where $F_+ = \{\oplus\}$, $F_- = \{\ominus, \otimes\}$, and the transition functions δ_{nd} and δ_d are defined as follows.

$$
\begin{array}{lll}
(1) & \delta_{nd}(a, a) & = \{1, -\} \\
(2) & \delta_{nd}(a, \#) & = \{\oplus\} \\
(3) & \delta_d(1, -) & = X \\
(4) & \delta_d(-, -) & = - \\
(5) & \delta_d(-, 1) & = <_1 \\
(6) & \delta_d(-, <_1) & = - \\
(7) & \delta_d(-, <_2) & = <_1 \\
(8) & \delta_d(X, -) & = X \\
(9) & \delta_d(X, <_1) & = X \\
(10) & \delta_d(<_1, X) & = <_2 \\
(11) & \delta_d(<_1, \sim) & = <_2
\end{array}
\qquad
\begin{array}{lll}
(12) & \delta_d(<_2, X) & = \sim \\
(13) & \delta_d(<_2, \sim) & = \sim \\
(14) & \delta_d(\sim, \sim) & = \sim \\
(15) & \delta_d(1, \oplus) & = \oplus \\
(16) & \delta_d(X, \otimes) & = \oplus \\
(17) & \delta_d(<_1, \oplus) & = \otimes \\
(18) & \delta_d(<_1, \ominus) & = \otimes \\
(19) & \delta_d(<_2, \ominus) & = \ominus \\
(20) & \delta_d(\sim, \oplus) & = \ominus \\
(21) & \delta_d(\sim, \ominus) & = \ominus
\end{array}
$$

In addition to these transitions, δ_d maps any state from $\{\ominus, \otimes, \oplus, 0\}$ to itself, regardless of its neighbor. Furthermore, all still undefined transitions map to the state 0.

The idea of the construction is as follows (see Figure 2 for an example). Assume that the cells are numbered from 1 to n from right to left. In the first step, each cell guesses whether its position is 2^i, for some $i \geq 1$ (Transition 1). Accordingly they enter state 1 or $-$. The rightmost cell can identify itself and always enters state $\oplus$ (Transition 2). Next, each cell in state 1 sends a signal with speed 1/2 to the left. The signal is realized by states $<_1$ and $<_2$ (Transitions 5–7 and 10–13). Moreover, cells in state 1 change to state X (Transition 3) and each cell passed through by such a signal changes to state $\sim$ (Transitions 12–14).

In addition to these signals, initially a signal s is sent by the rightmost cell to the left with speed 1. This signal is realized by the states $\{\ominus, \otimes, \oplus\}$ and possibly by state 0 if an initial guess is wrong. The states $\{\ominus, \otimes, \oplus\}$ represent accepting and rejecting decisions of the cells. Once such state is entered it is never left again. Therefore the decisions are not contradictory. Now the idea is that the initial guess is verified if and only if signal s meets a 1/2-speed signal in a cell that initially guessed to be at some position 2^i and, thus, is now in state X (Transitions16–21).

In order to evidence the correctness of the construction, let us first assume the initial guesses are correct. Then cells 1 and 2 behave as required by Transitions 2 and 15. Now let some cell 2^i enter the accepting state $\oplus$ at time 2^i (which is true for cells 1 and 2). Then the 1/2-speed signal sent by that cell has reached cell $2^i + 2^{i-1}$. This implies that the fast and slow signal will meet in cell 2^{i+1}, as required. Altogether, for the case of initially correct guesses, the decisions are never contradictory, the decisions are correct, and the guesses are verified to be true.

For the cases where one of the initial guesses is wrong, the neutral state 0 is used. Whenever a slow and the fast signal do not meet in a cell being in state X, state 0 is entered. Moreover, it is entered whenever two neighboring cells are in state 1. In particular, since the state 0 is never left, the fast signal checks the correctness of the initial guesses from right to left. It is stopped by any cell in the neutral state 0. This means that, again, no contradictory decisions are made and, in particular, no decision is made by the leftmost cell in case of wrong guesses.

So, this realtime one-way cellular automaton accepts language $\{\, a^{2^n} \mid n \geq 0 \,\}$ and it is self verifying. ■

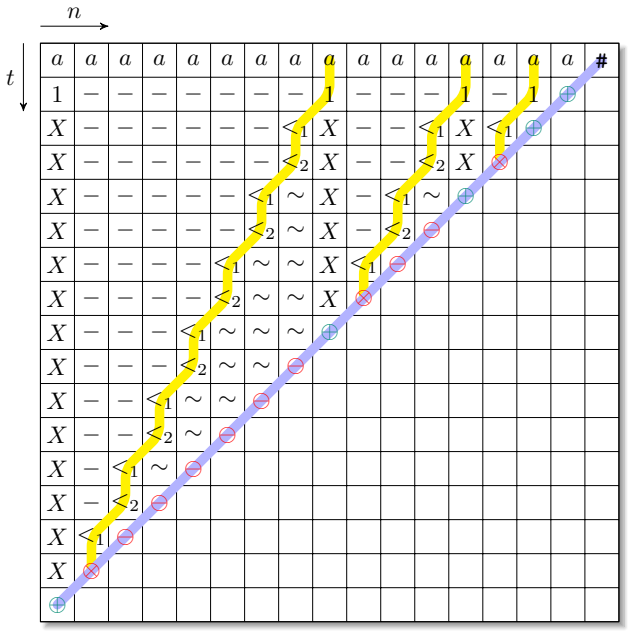

Fig. 2. Example computation of a realtime self-verifying one-way cellular automaton accepting the language $\{a^{2^n} \mid n \geq 0\}$. The slow signals moving with speed $1/2$ are depicted in yellow, the fast signal is depicted in lightblue.

3 Structural Properties and Speed-Up

First we give evidence that self-verifying (one-way) cellular automata are in fact a generalization of deterministic (one-way) cellular automata. However, to this end, it is reasonable to consider only such time complexities t that allow the leftmost cell to recognize the time step $t(n)$. Such functions are said to be *time computable*. For example, the identity $t(n) = n$ is a time-computable time complexity for (O)CA. A signal which is initially emitted by the rightmost cell and moves with maximal speed, arrives at the leftmost cell exactly at time step n. By slowing down the signal to speed $\frac{x}{y}$ (that is, the signal moves x cells to the left and then stays in a cell for $y - x$ time steps), it is seen that the time complexities $\lfloor \frac{y}{x} \cdot n \rfloor$, for any positive integers $x < y$, are also time computable. Other examples are exponential time complexities $t(n) = k^n$, for any integer $k \geq 2$. Further details on time-computable functions can be found in [3].

Lemma 2. *Any (one-way) deterministic cellular automaton with a time-computable time complexity t can effectively be converted into an equivalent (one-way) self-verifying cellular automaton with the same time complexity t.*

Proof. Since the time complexity t is computable, we may safely assume that its computation is performed on an extra track in parallel to the actual computation of a given deterministic cellular automaton M. Let S be the state set and F be the set of accepting states of M. An equivalent self-verifying cellular automaton M' basically simulates M. However, its state set is $S \cup S_+ \cup S_-$, where S_+ and S_- are disjoint copies of S. The simulation is extended such that the leftmost cell enters the corresponding state from S_+ instead of entering an accepting state. Subsequent steps are simulated just by staying in states from S_+. If the leftmost cell is still in a state from S when it recognizes the time step $t(n)$ from the time computation of t, then it enters the corresponding state from S_- in its next step and remains in states from S_-. Now it is

sufficient to define S to be the set of neutral states, S_+ to be the set of accepting, and S_- to be the set of rejecting states of M'. Since in one-way cellular automata the leftmost cell does not know that it is the leftmost one, the aforementioned behavior simply applies to all cells. In this way, clearly, M' is a self-verifying cellular automaton that accepts $L(M)$ with time complexity t. Moreover, M' is one-way if M is. □

It is known that several types of cellular automata can be sped-up by a constant amount of time as long as the remaining time complexity does not fall below realtime. A proof in terms of trellis automata can be found in [4]. In [14, 13] the speed-up results are shown for deterministic and nondeterministic cellular and iterative automata. The proofs are based on sequential machine characterizations of the parallel devices. In particular, deterministic CA and OCA can be sped-up from $(n+t(n))$-time to $(n+\frac{t(n)}{k})$-time [1, 14, 13]. Thus, lineartime is close to realtime. The question whether every lineartime CA can be sped-up to realtime is an open problem. The problem is solved for OCA. The realtime OCA languages are a proper subfamily of the lineartime OCA languages [4, 26].

Next we are going to show a stronger result for SVOCA from which follows that realtime is as powerful as lineartime. In order to prove the theorem we apply the *packing-and-checking* technique introduced in [2]. The basic principle is to guess the input in a packed form on the left of the array. The verification of the guess can be done by a deterministic OCA in realtime as shown by the next two lemmata.

Let Σ be an arbitrary alphabet that does neither contain the blank symbol $\sqcup$ nor the border symbol #. Following the idea, each cell of the OCA has k registers for the packed part of the input and one *input register* for its original input. The next two mappings extract the packed and the original input from a cell.

$$\begin{aligned} h_{k,1} \colon (\Sigma \cup \{\sqcup\})^k \times \Sigma &\to (\Sigma \cup \{\sqcup\})^k \\ (x_1, x_2, \ldots, x_{k+1}) &\mapsto x_1 x_2 \cdots x_k \end{aligned}$$

and

$$\begin{aligned} h_{k,2} \colon (\Sigma \cup \{\sqcup\})^k \times \Sigma &\to \Sigma \\ (x_1, x_2, \ldots, x_{k+1}) &\mapsto x_{k+1} \end{aligned}$$

The following lemma is used to verify whether (after a guess) the concatenation of the first k registers of all cells yields a word beginning with n symbols from Σ followed by $(k-1)\cdot n$ blank symbols $\sqcup$, that is, whether the packed input has the correct length and is contained in the leftmost $\lceil \frac{n}{k} \rceil$ cells.

Lemma 3. *Let $k > 1$. Then*

$$\begin{aligned} L_{k,1} = \{\, x_1 x_2 \cdots x_n \mid x_1, x_2, \ldots, x_n \in (\Sigma \cup \{\sqcup\})^k \times \Sigma \text{ and} \\ h_{k,1}(x_1) h_{k,1}(x_2) \cdots h_{k,1}(x_n) \in \Sigma^n \sqcup^{(k-1)\cdot n} \,\} \end{aligned}$$

is a realtime OCA language.

Proof. A corresponding OCA has to perform two checking tasks (see Figure 3). The first is to verify that $h_{k,1}(x_1)h_{k,1}(x_2)\cdots h_{k,1}(x_n)$ is of the form $\Sigma^*\sqcup^*$. To this end, the cell which contains the last symbol of the packed input generates a signal $\bullet$ in the corresponding register. The signal passes through the registers and cells in descending order and must not meet a symbol $\sqcup$.

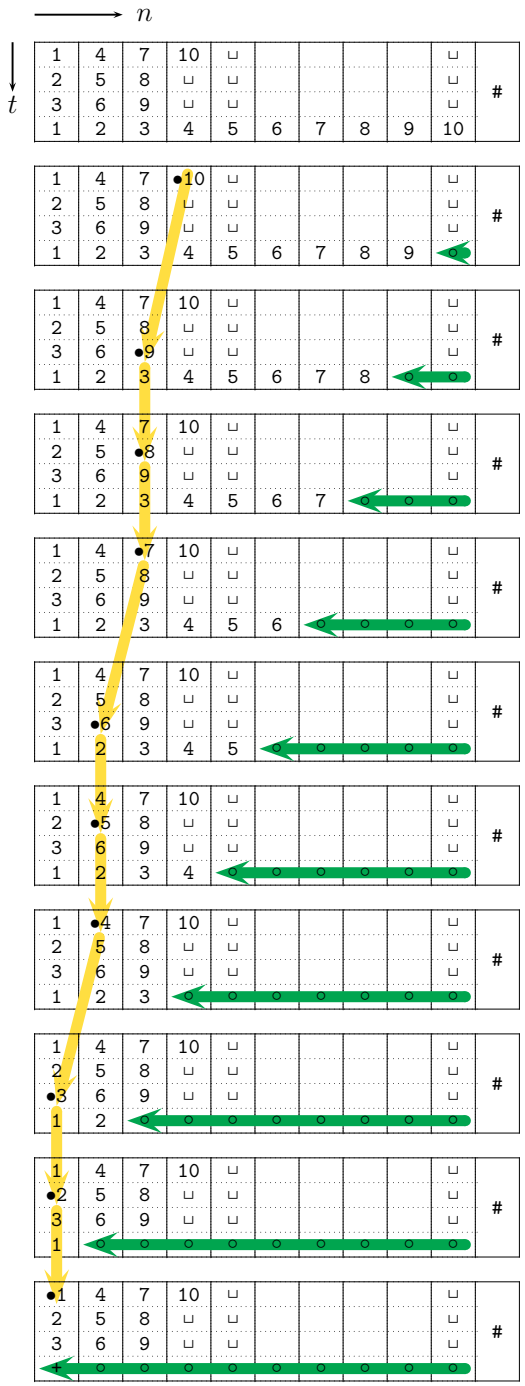

Fig. 3. Example for the proof of Lemma 3 ($k = 3$).

Otherwise an error signal is generated that prohibits the leftmost cell to accept. Additionally, error signals are generated if a register of a cell contains the blank symbol followed by a nonblank symbol.

The second task is to verify that the length of the packed input meets the length of the array. In order to implement this task, a left moving signal $\circ$ is initially emitted in the input

register of the rightmost cell. The lengths are identical if and only if the $\circ$ signal arrives at the left border exactly when the signal $\bullet$ arrives in the first register of the leftmost cell. □

To verify the guess it remains to be shown that the packed input is identical to the original input on the $(k+1)$-th track.

Lemma 4. *Let $k > 1$. Then*

$$L_{k,2} = \{\, x_1x_2\cdots x_n \mid x_1, x_2, \ldots, x_n \in (\Sigma \cup \{\sqcup\})^k \times \Sigma \text{ and } h_{k,1}(x_1)h_{k,1}(x_2)\cdots h_{k,1}(x_n) = h_{k,2}(x_1)h_{k,2}(x_2)\cdots h_{k,2}(x_n)\sqcup^{(k-1)\cdot n} \,\}$$

is a realtime OCA language.

Proof. Since the family $\mathscr{L}_{rt}(\text{OCA})$ is closed under intersection [18] we may assume that the input belongs to the language $L_{k,1}$ of Lemma 3.

The $\lceil \frac{n}{k} \rceil$ cells containing the packed input are able to identify themselves by the contents of their first k registers. Let every cell have another k registers which work like a first-in-first-out (FIFO) queue of fixed length k (see Figure 4). Whenever a symbol is stored in the input register, it is always fed to the queue (from the bottom in Figure 4). Whenever (because of the fixed length) a symbol has to be removed from the queue at the other end, it will be stored in the input register of the left neighboring cell. That way the short FIFO queues are "glued together".

In particular, each cell that has guessed a compressed part of the input initializes its FIFO queue in the first step by storing the current input symbol into its queue.

The rightmost $n - \lceil \frac{n}{k} \rceil$ cells shift the content on the track of input registers successively to the left. Thus, they are implementing the input stream to the long FIFO queue. At the end of the input stream – marked by a $\circ$ – each of the leftmost $\lceil \frac{n}{k} \rceil$ cells compares its FIFO contents to its packed input. If the comparisons of all cells are successful the input is accepted by a signal + in the input registers. Since the + is generated one time step after the arrival of the end-of-input-stream marker $\circ$, the OCA works in $n+1$ time, but can be sped-up by one time step to realtime. □

Next we turn to prove the strong speed-up result for SVOCA:

Theorem 5. *Let $k \geq 1$. Then $\mathscr{L}_{k\cdot t}(SVOCA) = \mathscr{L}_t(SVOCA)$, for all time complexities $t\colon \mathbb{N} \to \mathbb{N}$, $t(n) \geq n$.*

Proof. The inclusion $\mathscr{L}_t(\text{SVOCA}) \subseteq \mathscr{L}_{k\cdot t}(\text{SVOCA})$ follows immediately from the definition. So, let L be a language belonging to $\mathscr{L}_{k\cdot t}(\text{SVOCA})$ and let M be an SVOCA that accepts L with time complexity $k\cdot t$. We construct an SVOCA M' that simulates M in time $t(n)$. The underlying technique is packing-and-checking.

The idea is as follows: On an input of length n, each cell i with $1 \leq i \leq \lceil \frac{n}{k} \rceil$ guesses the initial states of the cells $k(i-1)+1, k(i-1)+2, \ldots, ki$ in its first k registers and remembers its original input in its $(k+1)$-th register. Based on this compressed representation, in each time step, M' can simulate k steps of M, which yields the required speed-up.

In parallel, M' has to check whether the guesses were correct, which can be done by the simulation of the acceptor of Lemma 4. For the verification n time steps are needed. So, M' accepts $L(M)$ with time complexity $t(n)$. □

Although the simulation on the compressed representation may be faster than realtime a speed-up below realtime is, of course, not possible due to the time needed for packing-and-checking.

Corollary 6. *The families $\mathscr{L}_{rt}(SVOCA)$ and $\mathscr{L}_{lt}(SVOCA)$ coincide.*

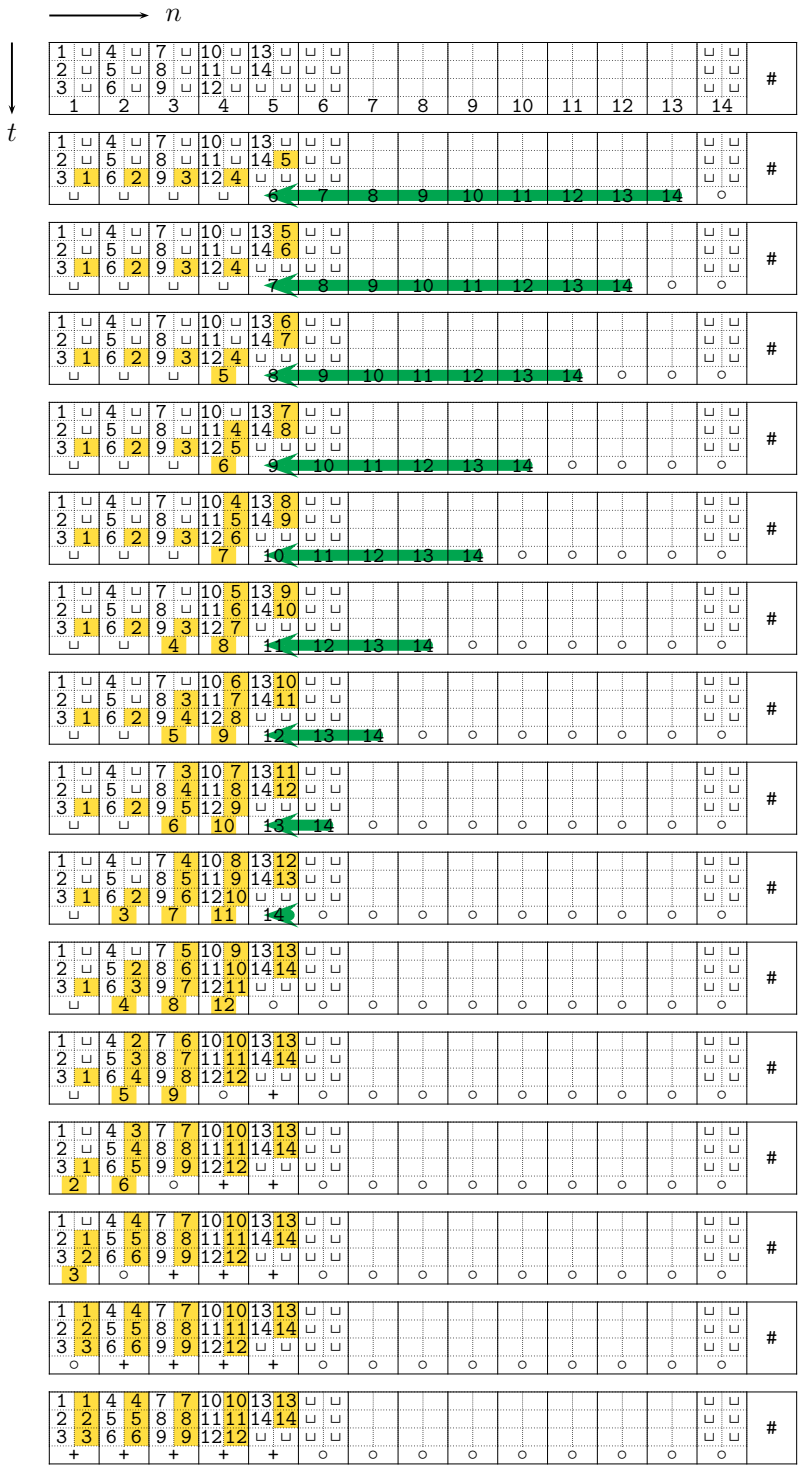

Fig. 4. Example for the proof of Lemma 4 ($k = 3$).

4 Self-Verifying One-Way Cellular Automata

4.1 Computational Capacity

At first we show that the computing power of realtime SVOCA is, in fact, strictly stronger than that of realtime OCA.

Theorem 7. *The family $\mathscr{L}_{rt}(OCA)$ is properly included in $\mathscr{L}_{rt}(SVOCA)$.*

Proof. The inclusion between the families follows from Lemma 2. Example 1 shows that there is a unary non-semilinear language accepted by some realtime SVOCA. On the other hand, the language does not belong to $\mathscr{L}_{rt}$(OCA) since it is not a regular language and in [24] it has been shown that realtime OCA cannot accept non-regular unary languages. Thus, the inclusion is proper. □

The inclusion of the previous result can be pushed higher in the hierarchy of language families. However, the strictness of the inclusion gets lost. The question of the strictness is strongly related to the famous open problem whether or not the realtime CA languages are a proper subfamily of the CA languages.

Theorem 8. *The family $\mathscr{L}_{lt}(CA)$ is included in $\mathscr{L}_{rt}(SVOCA)$.*

Proof. Let $L \in \mathscr{L}_{lt}$(CA). Since the family $\mathscr{L}_{lt}$(CA) is closed under reversal [25], there exists a lineartime CA that accepts L^R. This CA, in turn, can be sped-up by a multiplicative and additive constant [13]. Hence there is a CA $M = \langle S, \Sigma, F, \#, \delta \rangle$ that accepts L^R with time complexity $2n-1$.

Now, in a first step a deterministic OCA $M' = \langle S', \Sigma', F, \#, \delta' \rangle$ is constructed such that M' accepts the language $\{ \sqcup^{|w|} w^R \mid w \in L(M) \}$ with time complexity $2n-2$, where we tacitly assume $\sqcup \notin S$ and $n > 1$:

$$S' = (S \cup \{\sqcup\}) \cup (S \cup \{\sqcup\})^2, \qquad A' = A \cup \{\sqcup\},$$

$\forall s_1, s_2 \in S \cup \{\sqcup\}$:

$$\begin{aligned} \delta'(s_1, \#) &= (s_1, \sqcup), \\ \delta'(s_1, s_2) &= (s_1, s_2), \end{aligned}$$

$\forall (s_1, s_2), (s_2, s_3) \in (S \cup \{\sqcup\})^2$:

$$\delta'((s_1, s_2), (s_2, s_3)) = \begin{cases} \delta(s_3, s_2, s_1) & \text{if } (s_1 \neq \sqcup \wedge s_2 \neq \sqcup \wedge s_3 \neq \sqcup) \\ \delta(\#, s_2, s_1) & \text{if } (s_1 \neq \sqcup \wedge s_2 \neq \sqcup \wedge s_3 = \sqcup) \\ \delta(s_3, s_2, \#) & \text{if } (s_1 = \sqcup \wedge s_2 \neq \sqcup \wedge s_3 \neq \sqcup) \\ \sqcup & \text{otherwise} \end{cases}.$$

The basic idea is that during an intermediate step the cells of M' are collecting the information needed to simulate one step of the CA (see Figure 5). Due to the one-way information flow a cell i thereby can collect information from the cells $i+1$ and $i+2$ and, thus, simulate one step of the CA cell $i+1$. Therefore, the relevant part of the configuration shifts in space to the left.

Since $\mathscr{L}_{rt}$(OCA) is closed under reversal, Lemma 4 applies to $L^R_{k,2}$ as well, and $L^R_{k,2}$ is a realtime OCA language.

The cells of an SVOCA M'' that accepts the language $\{ w^R \mid w \in L(M) \}$ are constructed such that they can store two input symbols. Under input w^R the SVOCA M'' guesses in its first step the configuration $\sqcup^{|w|} w^R$ whereby two adjacent symbols are stored in one cell, respectively. The verification of the guess corresponds to the acceptance of the language $L^R_{k,2}$, for $k = 2$.

In parallel to the verification M'' simulates the OCA M' with double speed on the compressed input. Therefore, M'' has time complexity $1 + \frac{2n-2}{2} = n$. Since

$$L(M'') = \{ w^R \mid w \in L(M) \} = L^R(M) = (L^R)^R = L$$

the theorem follows.

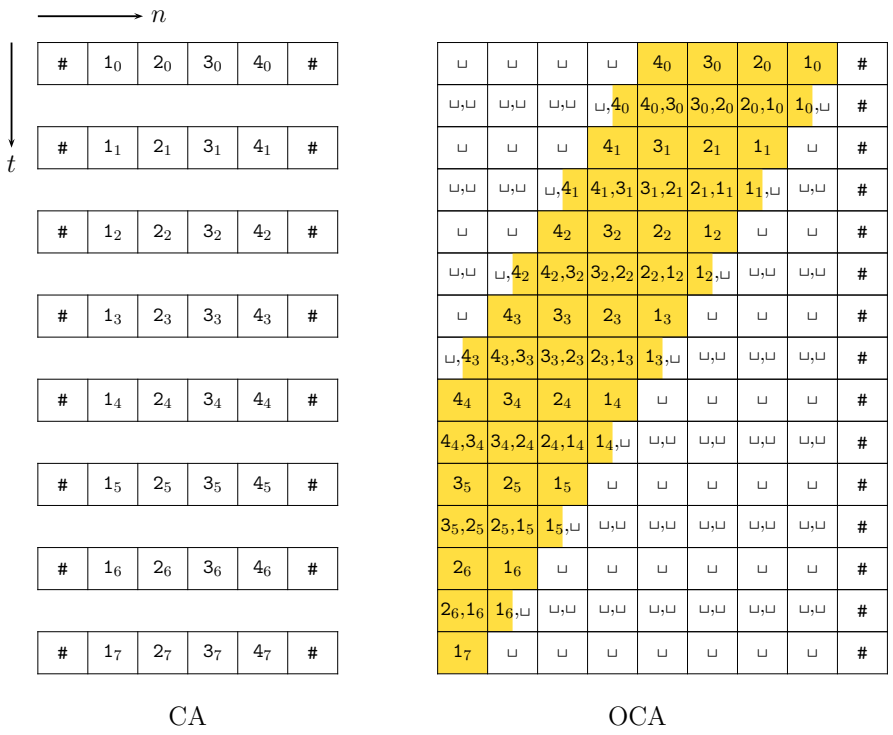

Fig. 5. Example for the proof of Theorem 8.

In order to make M'' self-verifying it enters accepting states if the guesses are correct and the simulation ends accepting, and enters rejecting states when the guesses are correct and the simulation does *not* end in an accepting state. All other states, in particular the states entered in case of wrong guesses, are neutral. □

4.2 Closure Properties

This section is devoted to the closure properties of the family of realtime SVOCA languages. They are summarized in Table 1. We start with the Boolean operations.

Proposition 9. *The family of languages accepted by realtime SVOCA is closed under complementation, union, and intersection.*

Proof. In order to construct an SVOCA that accepts the complement of the language accepted by a given SVOCA, it is sufficient to interchange the accepting and rejecting states while the neutral states remain as they are.

For the union and intersection, it is sufficient to simulate two given realtime SVOCA in parallel on different tracks, that is, to use the same two channel technique of [7] and [25]. Then it remains to adapt the sets of accepting, rejecting and neutral states accordingly. Let F_+, F_-, F_0 and F'_+, F'_-, F'_0 be the partitions of the states sets of the given realtime SVOCA. For the union, the set of accepting states is defined to be $\{\,(s,s') \mid s \in F_+ \text{ or } s' \in F'_+\,\}$, the set of rejecting states is $F_- \times F'_-$, and all remaining states are neutral.

Similarly, for the intersection, we define the set of accepting states as $F_+ \times F'_+$, the set of rejecting states as $\{\,(s,s') \mid s \in F_- \text{ or } s' \in F'_-\,\}$, and all remaining states are neutral. □

It is known that $\mathscr{L}_{rt}(\text{OCA})$ is closed under reversal [4], which is a long-standing open problem for $\mathscr{L}_{rt}(\text{CA})$.

Proposition 10. *The family of languages accepted by realtime SVOCA is closed under reversal.*

Proof. Let Σ be an arbitrary alphabet. In [7] it is shown that the language

$$L_R = \{\, w \in \Sigma^+ \mid w = w^R \,\}$$

belongs to the family $\mathscr{L}_{rt}(\text{OCA})$.

Let M be an SVOCA that accepts a language $L \subseteq \Sigma^*$ in realtime. An SVOCA M' that accepts L^R in $n+1$ time steps works as follows. On input $w = x_1x_2\cdots x_n$ every cell $1 \leq i \leq n$ of M' guesses the symbol x_{n-i+1} and stores it in an additional register. If the guesses are correct then M' has the symbols $x_nx_{n-1}\cdots x_2x_1 = w^R$ on its additional track. Furthermore, the cell in the center of the array is nondeterministically marked (if n is even the two cells in the center). Altogether, after the first time step M' performs three tasks in parallel.

Task 1 is to simulate M on w^R because $w \in L^R$ if and only if $w^R \in L = L(M)$.

The second task is to verify that the cell(s) in the center is (are) marked. It is realized by a signal which moves with speed $\frac{1}{2}$ from the marked cell(s) to the left. Additionally, a left moving signal is initially emitted in the rightmost cell. The marking is correct if and only if both signal meet in the leftmost cell.

The last task is to check that the guessed word w^R was correct. Since the input as well as its (guessed) reversal are stored on different tracks and the center is marked, M' can simulate two realtime OCA for the language L_R where the input is the left half of one track and the right half of the other track, respectively.

Finally, M' accepts if the guesses and the marking are correct, and the simulation of the first task is accepting. Automaton M' rejects if the guesses and the marking are correct, and the simulation is rejecting. In all other cases, M' remains in neutral states. We conclude that M' is an SVOCA that can be sped-up to realtime by Theorem 5. □

Proposition 11. *The family of languages accepted by realtime SVOCA is closed under concatenation.*

Proof. Let $L_1, L_2 \in \mathscr{L}_{rt}(\text{SVOCA})$. If the empty word belongs to L_1 then language L_2 belongs to the concatenation and vice versa. Since the family of languages accepted by realtime SVOCA is closed under union by Proposition 9, it remains to consider languages $L_1, L_2 \in \mathscr{L}_{rt}(\text{SVOCA})$ that do not contain the empty word. Let M_1, M_2 be acceptors for L_1 and L_2. As an intermediate step, we construct a self-verifying cellular automaton M with two-way information flow, that is, each cell is connected to its both nearest neighbors and the leftmost cell receives a boundary symbol on its free input line.

Since the family $\mathscr{L}_{rt}(\text{SVOCA})$ is closed under reversal, there is a realtime SVOCA M_1^R that accepts the reversal L_1^R of L_1. Now M has two tracks with identical inputs. On one track it simulates M_2, whereby each cell that enters an accepting or rejecting state is marked accordingly. On the second track, M simulates M_1^R from left to right. That is, the simulation is such that each cell receives the state from its left neighbor. So, the information flow is from left to right. Again, each cell that enters an accepting or rejecting state is marked accordingly.

Let the input be $x_1x_2\cdots x_n$. If a cell at position i is marked accepting by the simulation of M_2, the word $x_ix_{i+1}\cdots x_n$ belongs to the language L_2. If a cell at position i is marked accepting by the simulation of M_1^R, the word $x_ix_{i-1}\cdots x_1$ belongs to the language L_1^R and, thus,

$x_1x_2\cdots x_i$ belongs to the language L_1. So, the input $x_1x_2\cdots x_n$ belongs to the concatenation L_1L_2 if and only if M_1^R may mark a cell at position i and M_2 a cell at position $i+1$ accepting, for $1 \leq i < n$.

In order to check this condition, M uses a signal that is emitted from the rightmost cell when the simulation of M_1^R reaches that cell at time step n. The signal moves to the left and informs the leftmost cell at time step $2n$.

When the signal arrives, the leftmost cell enters an accepting state if and only if the signal has found two adjacent cells marked accepting. So, M accepts any input from L_1L_2 and only inputs from the concatenation L_1L_2. If the signal found neither two adjacent cells marked accepting, nor two adjacent cells that are marked accepting and unmarked, nor two adjacent cells unmarked the leftmost cell enters a rejecting state. In this case, no matter between which two adjacent symbols one assumes the cut between first and second factor, M has explicitly rejected at least one of them. Clearly, in this case the input cannot belong to the concatenation. On the other hand, if some input does not belong to the concatenation, then there is always a computation of M that results in such a marking. So, M rejects any input that does not belong to L_1L_2 and only inputs that do not belong to L_1L_2. In any other case, the leftmost cell remains in a neutral state.

So far, we have constructed a two-way self-verifying cellular automaton with time complexity $2n$. The proof of Theorem 8 can almost literally be used to show that also a lineartime two-way self-verifying cellular automaton can be simulated by a realtime SVOCA. □

Next, we turn to the operations homomorphism and inverse homomorphism.

Proposition 12. *The family of languages accepted by realtime SVOCA is not closed under homomorphisms.*

Proof. It is well known that every recursively enumerable language is the homomorphic image of the intersection of two context-free languages [10]. Moreover, every context-free language is the homomorphic image of the intersection of a regular language and a Dyck language [5].

The Dyck languages as well as the regular languages are realtime OCA languages [7] and therefore realtime SVOCA languages. Additionally, the family of realtime SVOCA languages is closed under intersection by Proposition 9. So, if the family $\mathscr{L}_{rt}(\text{SVOCA})$ would be closed under homomorphisms, it would contain every recursively enumerable language. Due to the time bound to realtime this is a contradiction. □

Proposition 13. *The family of languages accepted by realtime SVOCA is closed under inverse homomorphisms.*

Proof. Let M be a realtime SVOCA that accepts a language $L \subseteq \Sigma^*$, and $h\colon \Gamma^* \to \Sigma^*$ be a homomorphism. The maximal length of an image of some letter from Γ is $m = \max\{\, |x| \mid x \in \Gamma \,\}$.

The basic idea for a realtime SVOCA M' with input alphabet Γ that accepts $h^{-1}(L)$ is as follows. In its first transition, each cell of M' applies the homomorphism h to its input symbol. So, each cell of M' has to store up to m states of M. In the subsequent computation M' simulates M step by step.

There are two technical problems with this approach. One is caused by the fact that the homomorphism may be erasing symbols, leading to non-adjacent cells of M' having to simulate adjacent cells of M. The other problem is that M' (like M) is only allowed to make nondeterministic choices during the very first step. We first describe a solution taking care of the first but ignoring the second problem; the latter will be fixed afterwards.

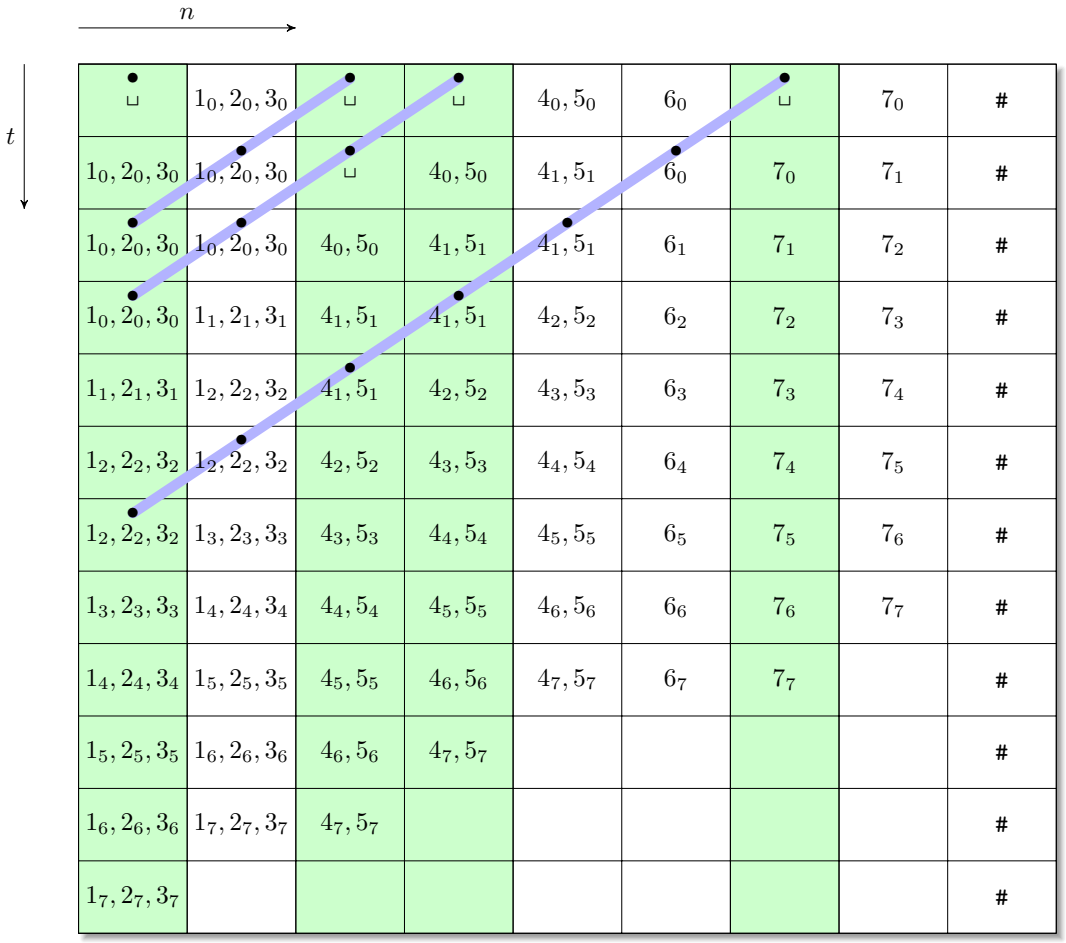

Fig. 6. Example for the proof of Proposition 13. The (blue) diagonal lines indicate the delay signals. The (green) shaded cells are initially empty and shift the neighboring states only. The guessed transitions are not depicted.

Each cell of M' is able to store the states of up to m cells of M. In its first transition, each cell of M' applies the homomorphism h to its input symbol. A cell of M' stores a $\sqcup$ if $h(x) = \lambda$ or $y_1, y_2, \ldots, y_k$ if $h(x) = y_1 y_2 \cdots y_k$. In addition, each cell memorizes whether it stores a $\sqcup$ now or not. In the subsequent computation $\sqcup$-cells only take over the state of its right neighbor (without changing it). A cell storing $y_1, y_2, \ldots, y_k$, for $1 \leq k \leq m$, simulates the state transitions for all k states at once if possible. However, a cell with blank right neighbor cannot do that. In order to cope with this problem, initially all $\sqcup$-cells send a delay signal $\bullet$ to the left with speed 1. Whenever such a signal arrives in a cell, the cell does not change its state. It continues to simulate the behavior of M if its neighbor does not carry a delay signal any more (see Figure 6).

In this way, all but the very first step of M can be simulated. In its first step, the cells of M may perform guesses dependent on their states and the states of their neighbors. But a cell with a neighboring $\sqcup$-cell cannot simulate the guess directly. So, the first step of M' is organized such that each cell (in addition to applying the homomorphism) guesses the transition that the corresponding cell of M would apply during its first transition. The simulation continues with the state resulting from the guessed transition. These guesses can be verified immediately when a cell receives the first non-blank state from its neighbor. In this way, all steps of a realtime computation of M are simulated unless all input symbols are erased by the homomorphism. In

this case, a state corresponding to the boundary symbol is sent from the rightmost cell through the entire array. If this signal arrives at some $\sqcup$-cell, the cell accepts if the empty word belongs to L, otherwise it rejects. In order to make M' self-verifying it is sufficient to recall that M is self-verifying, to define a state consisting of several states of M to be accepting, rejecting, or neutral dependent on the leftmost state, to define the state $\sqcup$ neutral, and to send a neutral signal to the left whenever a cell detects that its initial guessed transition of M is wrong. Together, the leftmost cell of M' accepts (rejects) if and only if the leftmost cell of M accepts (rejects).

Finally, M' runs at most twice as long as M since there are no more than n delay signals. So, M' runs in lineartime and can be sped-up to realtime again. □

The closure properties of $\mathscr{L}_{rt}$(SVOCA) with respect to iteration (Kleene star) and non-erasing homomorphisms are open problems. They are settled for nondeterministic devices since, basically, for iteration it is sufficient to guess the the positions in the array at which words are concatenated, and for non-erasing homomorphism it is sufficient to guess the pre-image of the input. However, self-verifying devices have to reject explicitly if the input does not belong to the language. Intuitively, this means that they have to 'know' that all possible guesses either do not lead to accepting computations or are 'wrong.'

Family	—	$\cup$	$\cap$	R	$\cdot$	$*$	h_λ	h	h^{-1}
$\mathscr{L}_{rt}$(SVOCA)	✓	✓	✓	✓	✓	?	?	✗	✓
$\mathscr{L}_{rt}$(OCA)	✓	✓	✓	✓	✗	✗	✗	✗	✓

Table 1. Closure properties of the language family $\mathscr{L}_{rt}$(SVOCA) in comparison with the family $\mathscr{L}_{rt}$(OCA), where h_λ denotes λ-free homomorphisms.

4.3 Decidability Questions

Now we turn to decidability questions. Clearly, the membership problem is decidable for realtime SVOCA languages since the family is effectively included in the deterministic context-sensitive languages. However, even realtime OCA can accept the so-called valid computations of Turing machines. Roughly speaking, these are languages of encodings of accepting Turing machine computations (see [22, 23] for the details or [20] for a survey and discussion). This means that many of the not even semi-decidable problems for Turing machines can be reduced to realtime OCA. The following theorem is from [20].

Theorem 14. *For any language family that effectively contains $\mathscr{L}_{rt}(OCA)$ the problems emptiness, universality, finiteness, infiniteness, context-freeness, and regularity are not semidecidable.*

So, we have the following consequences.

Corollary 15. *The problems emptiness, universality, finiteness, infiniteness, inclusion, equivalence, regularity, and context-freeness are not semidecidable for realtime SVOCA.*

Finally, we turn to the problem to decide whether a given realtime nondeterministic one-way cellular automaton is self-verifying or not.

Theorem 16. *Given a realtime deterministic one-way cellular automaton M, it is not semidecidable whether or not M is an SVOCA.*

Proof. Let M be a realtime OCA and F its set of accepting states. From M an equivalent realtime SVOCA M' is constructed according to Lemma 2. Next, M' is modified in such a way that first a new input symbol (and neutral state) $\boxminus$ and new states $\ominus$ and $\oplus$ are added. State $\ominus$ is a rejecting state and $\oplus$ is accepting. The transition functions are modified such that a cell in state $\boxminus$ in the first step nondeterministically can either change to $\ominus$ and remain in that state forever or to stay in $\boxminus$ unless its right neighbor is in an accepting state. In the latter case, the cell changes from state $\boxminus$ to $\oplus$ and stays in that state from then on.

We claim that M' is self-verifying if and only if $L(M')$ is empty. If $L(M')$ is empty, none of its cells will ever enter an accepting state. So, any cell that is initially in state $\ominus$ remains in $\ominus$ and, thus, will not give a contradictory answer. On the other hand, if there is some word w in $L(M')$, then on input $\boxminus w$ there obviously is a rejecting computation, but an accepting one as well. After at most $|w|$ time steps the second cell enters an accepting state. In the case that the leftmost cell still is in state $\boxminus$, one time step later it will change to the accepting state $\oplus$. Therefore, in this case, M' is not self-verifying.

Finally, assume that it is semidecidable whether a realtime OCA is self-verifying. This implies that we can semidecide its emptiness which is a contradiction to Corollary 15. □

Interestingly, by Lemma 2 any (one-way) deterministic cellular automaton with a time-computable time complexity can effectively be made self-verifying. On the other hand, it is non-semidecidable whether it already *is* self-verifying. The non-semidecidability generalizes immediately to nondeterministic cellular automata. However, Lemma 2 does not since an input may induce accepting as well as non-accepting computations. By the construction of Lemma 2 the latter would become rejecting. In fact, it is an open problem whether the family of realtime one-way nondeterministic cellular automata is closed under complementation or not.

References

1. Bucher, W., Čulik II, K.: On real time and linear time cellular automata. RAIRO Inform. Théor. 18, 307–325 (1984)
2. Buchholz, Th., Klein, A., Kutrib, M.: On interacting automata with limited nondeterminism. Fund. Inform. 52, 15–38 (2002)
3. Buchholz, Th., Kutrib, M.: On time computability of functions in one-way cellular automata. Acta Inform. 35, 329–352 (1998)
4. Choffrut, C., Čulik II, K.: On real-time cellular automata and trellis automata. Acta Inform. 21, 393–407 (1984)
5. Chomsky, N.: Context-free grammars and pushdown storage. Tech. Rep. QPR 65, Massachusetts Institute of Technology (1962)
6. Duris, P., Hromkovic, J., Rolim, J.D.P., Schnitger, G.: Las vegas versus determinism for one-way communication complexity, finite automata, and polynomial-time computations. In: Reischuk, R., Morvan, M. (eds.) Theoretical Aspects of Computer Science (STACS 1997). LNCS, vol. 1200, pp. 117–128. Springer (1997)
7. Dyer, C.R.: One-way bounded cellular automata. Inform. Control 44, 261–281 (1980)
8. Fernau, H., Kutrib, M., Wendlandt, M.: Self-verifying pushdown automata. In: Freund, R., Mráz, F., Průša, D. (eds.) Non-Classical Models of Automata and Applications (NCMA 2017). books@ocg.at, vol. 329, pp. 103–117. Austrian Computer Society, Vienna (2017)
9. Fischer, P.C., Kintala, C.M.R.: Real-time computations with restricted nondeterminism. Math. Systems Theory 12, 219–231 (1979)
10. Ginsburg, S., Greibach, S.A., Harrison, M.A.: One-way stack automata. J. ACM 14, 389–418 (1967)
11. Hromkovic, J., Schnitger, G.: On the power of las vegas for one-way communication complexity, obdds, and finite automata. Inform. Comput. 169, 284–296 (2001)
12. Hromkovic, J., Schnitger, G.: Nondeterministic communication with a limited number of advice bits. SIAM J. Comput. 33, 43–68 (2003)
13. Ibarra, O.H., Kim, S.M., Moran, S.: Sequential machine characterizations of trellis and cellular automata and applications. SIAM J. Comput. 14, 426–447 (1985)

14. Ibarra, O.H., Palis, M.A.: Some results concerning linear iterative (systolic) arrays. J. Parallel Distributed Comput. 2, 182–218 (1985)
15. Ilie, L., Păun, G., Rozenberg, G., Salomaa, A.: On strongly context-free languages. Discrete Appl. Math. 103, 158–165 (2000)
16. Jirásková, G.: State complexity of some operations on binary regular languages. Theoret. Comput. Sci. 330(2), 287–298 (2005)
17. Jirásková, G., Pighizzini, G.: Optimal simulation of self-verifying automata by deterministic automata. Inform. Comput. 209, 528–535 (2011)
18. Kasami, T., Fuji, M.: Some results on capabilities of one-dimensional iterative logical networks. Electronics and Communications in Japan 51-C, 167–176 (1968)
19. Kintala, C.M.R.: Computations with a Restricted Number of Nondeterministic Steps. Ph.D. thesis, Pennsylvania State University (1977)
20. Kutrib, M.: Cellular automata and language theory. In: Meyers, R. (ed.) Encyclopedia of Complexity and System Science, pp. 800–823. Springer (2009)
21. Kutrib, M.: Non-deterministic cellular automata and languages. Int. J. General Systems 41, 555–568 (2012)
22. Malcher, A.: Descriptional complexity of cellular automata and decidability questions. J. Autom. Lang. Comb. 7, 549–560 (2002)
23. Malcher, A.: On the descriptional complexity of iterative arrays. IEICE Trans. Inf. Syst. E87-D(3), 721–725 (2004)
24. Seidel, S.R.: Language recognition and the synchronization of cellular automata. Tech. Rep. 79-02, Department of Computer Science, University of Iowa, Iowa City (1979)
25. Smith III, A.R.: Real-time language recognition by one-dimensional cellular automata. J. Comput. System Sci. 6, 233–253 (1972)
26. Umeo, H., Morita, K., Sugata, K.: Deterministic one-way simulation of two-way real-time cellular automata and its related problems. Inform. Process. Lett. 14, 158–161 (1982)

Institut für Informatik

Justus-Liebig-Universität Giessen
Arndtstr. 2, 35392 Giessen, Germany

Recent Reports

(Further reports are available at `www.informatik.uni-giessen.de`.)

S. Beier, M. Holzer, *Properties of Right One-Way Jumping Finite Automata*, Report 1802, March 2018.

B. Truthe, *Hierarchy of Subregular Language Families*, Report 1801, February 2018.

M. Holzer, M. Hospodár, *On the Magic Number Problem of the Cut Operation*, Report 1703, October 2017.

M. Holzer, S. Jakobi, *A Note on the Computational Complexity of Some Problems for Self-Verifying Finite Automata*, Report 1702, April 2017.

S. Beier, M. Holzer, M. Kutrib, *On the Descriptional Complexity of Operations on Semilinear Sets*, Report 1701, April 2017.

M. Holzer, S. Jakobi, M. Wendlandt, *On the Computational Complexity of Partial Word Automata Problems*, Report 1404, May 2014.

H. Gruber, M. Holzer, *Regular Expressions From Deterministic Finite Automata, Revisited*, Report 1403, May 2014.

M. Kutrib, A. Malcher, M. Wendlandt, *Deterministic Set Automata*, Report 1402, April 2014.

M. Holzer, S. Jakobi, *Minimal and Hyper-Minimal Biautomata*, Report 1401, March 2014.

J. Kari, M. Kutrib, A. Malcher (Eds.), *19th International Workshop on Cellular Automata and Discrete Complex Systems AUTOMATA 2013 Exploratory Papers*, Report 1302, September 2013.

M. Holzer, S. Jakobi, *Minimization, Characterizations, and Nondeterminism for Biautomata*, Report 1301, April 2013.

A. Malcher, K. Meckel, C. Mereghetti, B. Palano, *Descriptional Complexity of Pushdown Store Languages*, Report 1203, May 2012.

M. Holzer, S. Jakobi, *On the Complexity of Rolling Block and Alice Mazes*, Report 1202, March 2012.

M. Holzer, S. Jakobi, *Grid Graphs with Diagonal Edges and the Complexity of Xmas Mazes*, Report 1201, January 2012.

H. Gruber, S. Gulan, *Simplifying Regular Expressions: A Quantitative Perspective*, Report 0904, August 2009.

M. Kutrib, A. Malcher, *Cellular Automata with Sparse Communication*, Report 0903, May 2009.

M. Holzer, A. Maletti, *An $n \log n$ Algorithm for Hyper-Minimizing States in a (Minimized) Deterministic Automaton*, Report 0902, April 2009.

H. Gruber, M. Holzer, *Tight Bounds on the Descriptional Complexity of Regular Expressions*, Report 0901, February 2009.

M. Holzer, M. Kutrib, and A. Malcher (Eds.), *18. Theorietag Automaten und Formale Sprachen*, Report 0801, September 2008.

M. Holzer, M. Kutrib, *Flip-Pushdown Automata: Nondeterminism is Better than Determinism*, Report 0301, February 2003

M. Holzer, M. Kutrib, *Flip-Pushdown Automata: $k + 1$ Pushdown Reversals are Better Than k*, Report 0206, November 2002

M. Holzer, M. Kutrib, *Nondeterministic Descriptional Complexity of Regular Languages*, Report 0205, September 2002

H. Bordihn, M. Holzer, M. Kutrib, *Economy of Description for Basic Constructions on Rational Transductions*, Report 0204, July 2002

M. Kutrib, J.-T. Löwe, *String Transformation for n-dimensional Image Compression*, Report 0203, May 2002

A. Klein, M. Kutrib, *Grammars with Scattered Nonterminals*, Report 0202, February 2002

A. Klein, M. Kutrib, *Self-Assembling Finite Automata*, Report 0201, January 2002

M. Holzer, M. Kutrib, *Unary Language Operations and its Nondeterministic State Complexity*, Report 0107, November 2001

A. Klein, M. Kutrib, *Fast One-Way Cellular Automata*, Report 0106, September 2001

M. Holzer, M. Kutrib, *Improving Raster Image Run-Length Encoding Using Data Order*, Report 0105, July 2001

M. Kutrib, *Refining Nondeterminism Below Linear-Time*, Report 0104, June 2001