

## Cellular Devices and Unary Languages

**Andreas Klein**

*Fachbereich für Mathematik und Informatik*  
*Universität Kassel*  
*Heinrich Plett Straße 40, D-34132 Kassel, Germany*  
*klein@mathematik.uni-kassel.de*

**Martin Kutrib**

*Institut für Informatik*  
*Universität Giessen*  
*Arndtstr. 2, D-35392 Giessen, Germany*  
*kutrib@informatik.uni-giessen.de*

---

**Abstract.** Devices of interconnected parallel acting sequential automata are investigated from a language theoretic point of view. Starting with the well-known result that each unary language accepted by a deterministic one-way cellular automaton (OCA) in real time has to be a regular language, we will answer the three natural questions ‘How much time do we have to provide?’ ‘How much power do we have to plug in the single cells (i.e., how complex has a single cell to be)?’ and ‘How can we modify the mode of operation (i.e., how much nondeterminism do we have to add)?’ in order to accept non-regular unary languages.

We show the surprising result that for classes of generalized interacting automata parallelism does not yield to more computational capacity than obtained by a single sequential cell. Moreover, it is proved that there exists a unary complexity class in between the real-time and linear-time OCA languages, and that there is a gap between the unary real-time OCA languages and that class.

Regarding nondeterminism as limited resource it is shown that a slight increase of the degree of nondeterminism as well as adding two-way communication reduces the time complexity from linear time to real time. Furthermore, by adding a wee bit nondeterminism an infinite hierarchy of unary language families dependent on the degree of nondeterminism is derived.

**Keywords:** Cellular automata; Unary formal languages; Limited nondeterminism; Generalized sequential machines; Parallel computing

## 1. Introduction

Devices of interconnected parallel acting automata have extensively been investigated from a language theoretic point of view. The specification of such a system includes the type and specification of the single automata, the interconnection scheme (which sometimes implies a dimension to the system), a local and/or global transition function, and the input and output modes. One-dimensional devices with nearest neighbor connections whose cells are deterministic finite automata are commonly called cellular automata (CA) resp. iterative arrays (IA) in case of parallel resp. sequential input mode. One-way information flow is indicated by the notion OCA resp. OIA. The family of languages accepted by, e.g., CA in real time (linear time) is denoted by  $\mathcal{L}_{rt}(CA)$  ( $\mathcal{L}_{lt}(CA)$ ) and the corresponding subfamily of unary languages by  $\mathcal{L}_{rt}^u(CA)$  ( $\mathcal{L}_{lt}^u(CA)$ ). There are several important open problems concerning the relations between the various families. Unary languages play an important role in investigations in that field. Though any language can be unarily encoded there are differences in time and space complexity. Several works on sequential automata and complexity theory deal with unary languages (e.g., [1, 2, 8, 11, 16]). Under unary restriction several essentially different families are identical (e.g., the regular and the context-free languages, or  $\mathcal{L}_{rt}(IA) \subset \mathcal{L}_{rt}(CA)$  [9, 25] but  $\mathcal{L}_{rt}^u(IA) = \mathcal{L}_{rt}^u(CA)$  [23], or  $\mathcal{L}(DFA) \subset \mathcal{L}_{rt}(OCA)$  [20] but  $\mathcal{L}^u(DFA) = \mathcal{L}_{rt}^u(OCA)$  [24], where  $\mathcal{L}(DFA)$  denotes the languages accepted by deterministic finite automata, the regular languages), from which follows that the capabilities of some devices become noticeable not for unary languages. There are incomparable families that become comparable (e.g.,  $\mathcal{L}_{rt}(OCA)$  is not comparable to  $\mathcal{L}_{rt}(IA)$  [10] but  $\mathcal{L}_{rt}^u(OCA) \subset \mathcal{L}_{rt}^u(IA)$  [7]), which means that the restriction affects the families differently. Moreover, for unary families some general open properties are known (e.g.,  $\mathcal{L}_{rt}^u(CA)$  is closed under concatenation [17]) and some others are still open (e.g., whether or not  $\mathcal{L}_{rt}^u(CA) = \mathcal{L}_{lt}^u(CA)$ ). From these examples one can obtain that there is no general rule for what happens if we change from arbitrary to unary languages.

Unary languages serve in many proofs as (counter-)examples. E.g., in the past the only languages known not to belong to  $\mathcal{L}_{rt}(OCA)$  have been the non-regular unary ones. If one intends to accept non-regular unary languages by parallel one-way devices at least three natural questions arise: How much time do we have to provide for OCAs? How much power do we have to plug in the single cells (i.e., how complex has a single cell to be)? and How can we modify the mode of operation (i.e., how much nondeterminism do we have to add)?

The paper is organized as follows: In section 2 we define the basic notions and the model in question. Thereby, the parallel model is derived from the definition of general sequential machines that are actually the single cells.

In order to answer the second question in Section 3 we generalize the result  $\mathcal{L}^u(DFA) = \mathcal{L}_{rt}^u(OCA)$  to one-way cellular automata whose cells are much more complex, and draw a borderline to cells that give one-way cellular automata the power to accept non-regular unary languages. A consequence is that for these devices parallelism does not yield to more computational capacity than obtained by a single sequential cell.

Section 4 is devoted to the first question. We show that there exists a complexity class between the real-time and linear-time OCA languages:  $\mathcal{L}_{rt}(OCA) \subset \mathcal{L}_{rt+\log}(OCA) \subseteq \mathcal{L}_{(1+\epsilon).rt}(OCA)$ . Moreover, there is a gap between  $\mathcal{L}_{rt}^u(OCA)$  and  $\mathcal{L}_{rt+\log}^u(OCA)$ . Thus (in terms of unary languages) it follows that at least a logarithmic number of time steps have to be added in order to increase the computational capacity of real-time one-way cellular automata.

Regarding nondeterminism as limited resource it is shown in Section 4 that a slight increase of the

degree of nondeterminism as well as adding two-way communication reduces the time complexity from linear time to real time. Furthermore, by adding a wee bit nondeterminism an infinite hierarchy of unary language families dependent on the degree of nondeterminism is derived, from which an answer to the third question follows.

## 2. Models and Definitions

We denote the non-negative integers by  $\mathbb{N}$ , and the positive integers  $\{1, 2, \dots\}$  by  $\mathbb{N}_+$ . The empty word is denoted by  $\lambda$ , and the reversal of a word  $w$  by  $w^R$ . Similarly, the notions  $L^R$  and  $\mathcal{L}^R$  are used for languages and families of languages. For the length of  $w$  we write  $|w|$ . We use  $\subseteq$  for inclusions and  $\subset$  for strict inclusions. If  $(x_1, \dots, x_d)$  is a  $d$ -tuple,  $\pi_i(x_1, \dots, x_d) = x_i$  is the projection to the  $i$ th component.

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, i.e.,  $L \setminus \{\lambda\} = L' \setminus \{\lambda\}$ . If  $\mathcal{L}$  is a family of languages, we denote the subfamily of unary languages by  $\mathcal{L}^u$ , i.e., the languages over a singleton.

For a function  $f$  we denote its  $i$ -fold composition by  $f^{[i]}$ ,  $i \in \mathbb{N}_+$ . As usual, we define the set of functions that grow strictly less than  $f$  by

$$o(f) = \{g : \mathbb{N} \rightarrow \mathbb{N}_+ \mid \lim_{n \rightarrow \infty} \frac{g(n)}{f(n)} = 0\}.$$

Conversely, the set of all functions  $g$  such that  $f$  grows strictly less than  $g$  is

$$\omega(f) = \{g : \mathbb{N} \rightarrow \mathbb{N}_+ \mid f \in o(g)\}.$$

### 2.1. Sequential Machines

A sequential machine is basically a memory augmented nondeterministic finite automaton. Its state transition depends on the current state, on the current input symbol, and on the current memory content. The memory access is restricted in such a way that the finite control obtains information only on a finite part of the memory content, although the memory itself may have infinite capacity. On the other hand, the memory content may be updated whereby a priori no memory access restriction is made.

In order to define sequential machines formally, we introduce an infinite set  $V = \{m_1, m_2, \dots\}$  of symbols from which a finite number of memory symbols have to be chosen, i.e., symbols from which the memory content is built. Certainly, this is not a real restriction. However, later it is convenient, since we are interested in special subfamilies of sequential machines where the memory update and memory access is restricted.

**Definition 2.1.** A *nondeterministic sequential machine* is a system  $\langle S, M, D, s_0, A, F \rangle$ , where

1.  $S$  is the finite, nonempty set of *states*,
2.  $M \subset V$  is the finite set of *memory symbols*,
3.  $s_0 \in S$  is the *initial state*,
4.  $A$  is the finite, nonempty set of *input symbols*,

5.  $F \subseteq S$  is the set of *accepting states*, and
6.  $D$  is the finite, nonempty set of triples  $(\delta, \mu, \alpha)$  of computable functions, where
  - (a)  $\delta : S \times (M \cup \{\lambda\}) \times A \rightarrow S$  is the *state transition function*,
  - (b)  $\mu : S \times M^* \times A \rightarrow M^*$  is the *memory update function*, and
  - (c)  $\alpha : M^* \rightarrow (M \cup \{\lambda\})$  is the *memory access function*.

Observe, that for  $M = \emptyset$  the memory access function and the memory update functions are uniquely determined.

A sequential machine is *deterministic*, if  $|D| = 1$ .

The computations of a sequential machine is now formalized in terms of configurations and a global transition function. A configuration of a sequential machine is a description of its global state which is actually an element of  $S \times M^* \times A^*$ . During its course of computation a sequential machine nondeterministically steps through a sequence of configurations. For a given input  $w$  the initial configuration is set to  $(s_0, \lambda, w)$ , while successor configurations are chosen according to the *global transition function*  $\Delta$  as follows: Let  $c = (s, u, w)$  and  $c' = (s', u', w')$  be two configurations such that  $w = ax$ , for some  $a \in A$  and  $x \in A^*$ . Then

$$c' \in \Delta(c) \iff \exists (\delta, \mu, \alpha) \in D : s' = \delta(s, \alpha(u), a) \wedge u' = \mu(s, u, a) \wedge w' = x.$$

Thus, the global transition function is induced by the set of triples  $D$ . The  $i$ -fold composition of  $\Delta$  is defined as follows:

$$\Delta^{[0]}(c) = \{c\}, \quad \Delta^{[i+1]}(c) = \bigcup_{c' \in \Delta^{[i]}(c)} \Delta(c'),$$

where  $0 \leq i < |w|$ .

**Definition 2.2.** Let  $\mathcal{M}$  be a sequential machine. Then

$$L(\mathcal{M}) = \{w \in A^* \mid \exists c \in \Delta^{|w|}(s_0, \lambda, w) : \pi_1(c) \in F\}$$

is the language *accepted* by  $\mathcal{M}$ .

Since, in particular, the memory update functions are not subject to any restriction except for being computable, the family of languages accepted by nondeterministic and even deterministic sequential machines is the well-known family of recursive languages. However, sequential machines are intended to serve as a general framework to consider various (more familiar) sequential models at the same time. To this end, we define subfamilies of sequential machines (which are actually single cells of parallel devices) by introducing predicates that restrict the memory functions. Predicate  $P$  restricts the memory update function  $\mu$ , and predicate  $Q$  restricts the memory access function  $\alpha$ .

**Definition 2.3.** Let  $P$  and  $Q$  be two decidable predicates that relate words over  $V$ , respectively, where  $P(u, u)$  for all  $u \in V^*$ . The family of all sequential machines  $\langle S, M, D, s_0, A, F \rangle$  which satisfy

$$P(u, \mu(s, u, a)) \wedge Q(u, \alpha(u)), \text{ for all } (\delta, \mu, \alpha) \in D, s \in S, u \in M^*, a \in A,$$

is denoted by  $SEQ(P, Q)$ .

Observe, that the first condition is a very natural one. Sequential machines in  $SEQ(P, Q)$  need not change their memory content. By supplying suitable predicates we may obtain well-known subfamilies of sequential machines.

**Example 2.1.** Let  $P(u, v) \iff u = v$  and  $Q(u, v) \iff v = \lambda$ , then  $NFA = SEQ(P, Q)$  is the family of nondeterministic finite automata. If the sequential machines are required to be deterministic, we obtain the family of deterministic finite automata DFA by the same predicates.

**Example 2.2.** Let  $P(u, v) \iff (u = v) \vee (u = \lambda) \vee (u = ay \wedge v = xy)$ , where  $a \in V$  and  $x, y \in V^*$ , and let  $Q(u, v) \iff (u = v = \lambda) \vee (u = ay \wedge v = a)$ , where  $a \in V$  and  $y \in V^*$ . Then  $NPDA = SEQ(P, Q)$  is the family of nondeterministic pushdown automata. If the sequential machines are required to be deterministic, we obtain the family of deterministic pushdown automata without  $\lambda$ -moves (deterministic real-time pushdown automata) by the same predicates.

In the sequel  $SEQ$  always denotes a subfamily of sequential machines which is induced by some two predicates  $P$  and  $Q$ . The family of all languages which are accepted by some  $SEQ$  (i.e., by some sequential machine in  $SEQ$ ) is denoted by  $\mathcal{L}(SEQ)$ . Observe, that the sequential machines – following the concept of abstract families of automata [12] – have been designed to meet the requirements a model has to fulfill in order to construct an interconnected and interacting array of such machines easily. Therefore, we did not introduce the capability to perform transitions without consuming input symbols. For the same reason it is adequate to allow real-time computations only (i.e., the length of the input determines the number of transitions), although the model can easily be extended to operate beyond real time.

## 2.2. Cellular Machines

A cellular machine is an infinite linear array of copies of a sequential machine, sometimes called cells, where each of them is connected to its both nearest neighbors. For convenience we identify the cells by integers. The state transition as well as the memory update for each cell depends on its current state and memory content, and the current states of its neighbors. More precisely, at discrete time steps *one* local transition function is nondeterministically chosen and applied to all cells synchronously. More formally:

**Definition 2.4.** A *cellular machine* is a system  $\langle S, M, D, \#, A, F \rangle$ , where

1. there exists  $s_0 \in S$  such that  $\langle S, M, D, s_0, S \times S, F \rangle$  is a sequential machine,
2.  $\# \in S$  is the *boundary state* satisfying

$$\forall s \in S, \forall m \in M \cup \{\lambda\}, \forall p \in S \times S : \quad s = \# \iff \delta(s, m, p) = \#$$

and

$$\forall s \in S, \forall u \in M^*, \forall p \in S \times S : \quad s = \# \implies \mu(s, u, p) = u,$$

3.  $A \subseteq S$  is the finite, nonempty set of *input symbols*.

A configuration of a cellular machine at some time  $t \geq 0$  is a description of its global state, which is actually a mapping  $c_t : [0, \dots, n + 1] \rightarrow S \times M^*$ , for  $n \in \mathbb{N}_+$ , giving the current states and

memory contents of the cells. The initial configuration  $c_{w,0}$  at time 0 is defined by the input word  $w = a_1 \cdots a_n \in A^+$ :

$$c_{0,w}(i) = (a_i, \lambda), \text{ for } i \in \{1, \dots, n\}, \quad \text{and} \quad c_{0,w}(i) = (\#, \lambda), \text{ for } i \in \{0, n+1\}.$$

Similarly, successor configurations are chosen according to the *global transition function*  $\Delta$ : Let  $c$  and  $c'$  be two configurations with  $c(i) = (s_i, u_i)$  and  $c'(i) = (s'_i, u'_i)$ , then

$$c' \in \Delta(c) \iff \exists (\delta, \mu, \alpha) \in D : s'_i = \delta(s_i, \alpha(u_i), (\pi_1(c(i-1)), \pi_1(c(i+1)))) \wedge \\ u'_i = \mu(s_i, u_i, (\pi_1(c(i-1)), \pi_1(c(i+1))))),$$

for all  $i \in \{1, \dots, n\}$ .

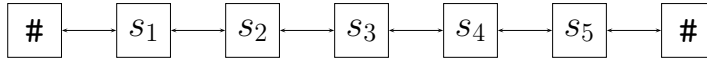


Figure 1. A two-way cellular machine. Memory contents are omitted.

If the flow of information is restricted to one-way (leftwards), the resulting device is a *one-way cellular machine*. I.e., the behavior of each cell is independent of its left neighbor.

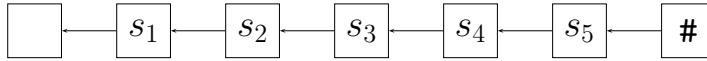


Figure 2. A one-way cellular machine. Memory contents are omitted.

An input is accepted by a cellular machine, if at some time during its course of computation the leftmost cell enters an accepting state.

**Definition 2.5.** Let  $\mathcal{M} = \langle S, M, D, \#, A, F \rangle$  be a cellular machine.

1. An input  $w \in A^+$  is *accepted* by  $\mathcal{M}$ , if there exists a time step  $i \in \mathbb{N}_+$  such that  $\pi_1(c_i(1)) \in F$  for at least one configuration  $c_i \in \Delta^{[i]}(c_{0,w})$ .
2.  $L(\mathcal{M}) = \{w \mid w \text{ is accepted by } \mathcal{M}\}$  is the *language accepted by*  $\mathcal{M}$ .
3. Let  $t : \mathbb{N}_+ \rightarrow \mathbb{N}_+$ ,  $t(n) \geq n$ , 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*  $t$ .

Let *SEQ* be a subfamily of the sequential machines, then *CSEQ* (*OCSEQ*) denotes the induced family of two-way (one-way) cellular machines. The family of all languages which are accepted by a *CSEQ* with time complexity  $t$  is denoted by  $\mathcal{L}_t(\text{CSEQ})$ . If  $t$  is the *identity function*  $t(n) = n$ , acceptance is said to be in *real time* and we write  $\mathcal{L}_{rt}(\text{CSEQ})$ . The linear-time languages  $\mathcal{L}_{lt}(\text{CSEQ})$  are defined by

$$\mathcal{L}_{lt}(\text{CSEQ}) = \bigcup_{k \in \mathbb{N}_+} \mathcal{L}_{k \cdot n}(\text{CSEQ}).$$

Whenever deterministic finite automata are used to build a cellular machine, we write CA resp. OCA for *C DFA* resp. *O C DFA* and call the machines *cellular automata* resp. *one-way cellular automata*. For simplicity, in connection with CA resp. OCA we often omit the memory augmentation such that they are considered to be merely systems  $\langle S, \delta, \#, A, F \rangle$ , where  $\delta$  maps from  $S \times S \times S$  resp. from  $S \times S$  to  $S$ .

### 3. Cells Beyond Finite Automata

It is known that  $\mathcal{L}(\text{DFA}) \subset \mathcal{L}_{rt}(\text{OCA})$  [20] but  $\mathcal{L}^u(\text{DFA}) = \mathcal{L}_{rt}^u(\text{OCA})$  [24]. I.e., although the cells of an OCA are copies of just one finite automaton, their interaction yields capabilities exceeding those of a single finite automaton. On the other hand, in case of unary languages the power gained in the parallelism collapses. A similar result is known for pushdown cellular automata. Translated to our notion,  $\mathcal{L}(\text{DPDA}) \subset \mathcal{L}_{rt}(\text{OCDPDA})$  and  $\mathcal{L}^u(\text{DPDA}) \subset \mathcal{L}_{rt}^u(\text{OCDPDA})$  has been shown [22].

The goal of this section is to show that it is an intrinsic property of real-time one-way cellular machines that their computational power becomes effective for non-unary languages only. First, we prove the inclusion  $\mathcal{L}(\text{SEQ}) \subseteq \mathcal{L}_{rt}(\text{OCSEQ})$ . In general, the inclusion is not a proper one since, as mentioned above, the family of languages accepted by general nondeterministic sequential machines is the well-known family of recursive languages and, due to the time bound, any language accepted by a real-time cellular machine has to be recursive, too.

**Theorem 3.1.** For any subfamily  $\text{SEQ}$  of sequential machines it holds

$$\mathcal{L}(\text{SEQ}) \subseteq \mathcal{L}_{rt}(\text{OCSEQ}).$$

**Proof:**

Let  $P$  and  $Q$  be two predicates such that  $\text{SEQ} = \text{SEQ}(P, Q)$ . Further, let  $\mathcal{M} = \langle S, M, D, s_0, A, F \rangle$  be some  $\text{SEQ}$ , where  $D = \{(\delta_i, \mu_i, \alpha_i) \mid 1 \leq i \leq k\}$ , for some  $k \in \mathbb{N}_+$ . The construction of an equivalent real-time  $\text{OCSEQ}$   $\mathcal{M}' = \langle S', M', D', \#, A', F' \rangle$  follows a simple idea:

The input word is shifted to the left through the cells symbol by symbol. Each cell fetching such a symbol simulates a corresponding transition of  $\mathcal{M}$ . A cell stops the simulation when it fetches information from the rightmost cell which can identify itself by the initial state  $\#$  of its right neighbor. So, the leftmost cell simulates  $\mathcal{M}$  on the entire input and, thus,  $\mathcal{M}'$  is a real-time acceptor for  $L(\mathcal{M})$ .

However,  $\text{SEQ}$  is some subfamily of sequential machines for which almost nothing is known. Therefore, it might be possible that the sequential machine we have to plug into the cells of  $\mathcal{M}'$  in order to achieve the described behavior does not belong to  $\text{SEQ}$ . Fortunately, the condition (on predicate  $P$ ) that a cell need not change its memory content overcomes this problem.

Formally,  $\mathcal{M}'$  is constructed as follows. Let  $\#$  be a new symbol not belonging to  $A$  nor to  $S \times A$ .

$$S' = S \times (A \cup \{\#\}) \cup A \cup \{\#\}, \quad M' = M, \quad A' = A, \quad F' = \{(s, \#) \mid s \in F\}$$

$$D' = \{(\delta'_i, \mu'_i, \alpha'_i) \mid 1 \leq i \leq k\}, \text{ where for all } i \in \{1, \dots, k\} :$$

$$\alpha'_i = \alpha_i$$

$$\forall s \in A, \forall r \in A \cup \{\#\}, \forall m \in M \cup \{\lambda\} :$$

$$\begin{aligned} \delta'_i(\#, m, r) &= \# \\ \delta'_i(s, m, r) &= (\delta_i(s_0, m, s), r) \end{aligned}$$

$$\forall s_1 \in S, \forall s_2 \in A, \forall (r_1, r_2) \in S \times (A \cup \{\#\}), \forall m \in M \cup \{\lambda\} :$$

$$\begin{aligned} \delta'_i((s_1, \#), m, (r_1, r_2)) &= (s_1, \#) \\ \delta'_i((s_1, s_2), m, (r_1, r_2)) &= (\delta_i(s_1, m, s_2), r_2) \end{aligned}$$

$\forall s \in A, \forall r \in A \cup \{\#\}, \forall u \in M^*$  :

$$\begin{aligned}\mu'_i(\#, u, r) &= u \\ \mu'_i(s, u, r) &= \mu_i(s_0, u, s)\end{aligned}$$

$\forall s_1 \in S, \forall s_2 \in A, \forall (r_1, r_2) \in S \times (A \cup \{\#\}), \forall u \in M^*$  :

$$\begin{aligned}\mu'_i((s_1, \#), u, (r_1, r_2)) &= u \\ \mu'_i((s_1, s_2), u, (r_1, r_2)) &= \mu_i(s_1, u, s_2)\end{aligned}$$

The function  $\mu'$  meets the predicate  $P$  since  $P(u, u)$  and  $\mu$  meets  $P$ .  $\square$

In general, the converse of the theorem is not true. For example, it has been mentioned above that the proper inclusion  $\mathcal{L}(\text{DFA}) \subset \mathcal{L}_{rt}(\text{OCA})$  is known. But nevertheless, for unary languages the converse can be shown.

**Lemma 3.1.** For any subfamily  $SEQ$  of sequential machines it holds

$$\mathcal{L}_{rt}^u(\text{OCSEQ}) \subseteq \mathcal{L}^u(\text{SEQ}).$$

**Proof:**

Let  $L \in \mathcal{L}_{rt}^u(\text{OCSEQ})$  be a language defined over the alphabet  $A = \{a\}$ , and  $\mathcal{M} = \langle S, M, D, \#, A, F \rangle$  be a real-time  $\mathcal{L}_{rt}^u(\text{OCSEQ})$  accepting  $L$ , where  $D = \{(\delta_i, \mu_i, \alpha_i) \mid 1 \leq i \leq k\}$ , for some  $k \in \mathbb{N}_+$ . Without loss of generality we may assume that once a cell of  $\mathcal{M}$  has entered an accepting state it remains in an accepting state.

On input  $w = a^n$  we make two observations (cf. Figure 3): At every time step one of the local transition functions is applied to all cells. Therefore, two cells having the same state and memory content behave equally if they fetch the same state information from their corresponding right neighbors. Therefore,  $c_1(1) = \dots = c_1(n-1)$ , since  $c_{o,w}(1) = \dots = c_{0,w}(n) = (a, \lambda)$ . Similarly, we have  $c_t(1) = \dots = c_t(n-t)$ , for all time steps  $t$  with  $0 \leq t \leq n-1$ .

The behavior of the leftmost cell of  $\mathcal{M}'$  is not affected by the states of a cell  $i$ ,  $1 \leq i \leq n$ , at time  $t$  where  $t > n-i+1$ . Furthermore, a cell is not able to access the memory of its neighbor directly. Hence, together with the first observation it suffices to know the state of the cells  $i$  at time  $n-i+1$  in order to simulate the state and memory transitions of the leftmost cell up to time  $n$ .

A corresponding  $SEQ$   $\mathcal{M}' = \langle S', M', D', s'_0, F', A' \rangle$  accepting  $L$  uses two registers. In the first one it simulates the behavior of the cells  $1, \dots, i$  at time  $n-i$ , and in the second one the behavior of cell  $i$  at time  $n-i+1$ , for  $1 \leq i \leq n$ . Now the construction of  $\mathcal{M}'$  is straightforward:

$$S' = S \times S, \quad M' = M, \quad A' = \{a\}, \quad F' = \{(s, r) \mid r \in F\}, \quad s_0 = (a, \#)$$

$$D' = \{(\delta'_i, \mu'_i, \alpha'_i) \mid 1 \leq i \leq k\}, \text{ where for all } i \in \{1, \dots, k\}: \quad \alpha'_i = \alpha_i$$

$$\forall (s, r) \in S', \forall m \in M \cup \{\lambda\}: \quad \delta'_i((s, r), m, a) = (\delta_i(s, m, s), \delta_i(s, m, r))$$

$\forall (s, r) \in S', \forall u \in M^*$  :  $\mu'_i((s, r), u, a) = \mu_i(s, u, s)$  The function  $\mu'$  meets the predicate  $P$  since  $\mu$  meets  $P$ .  $\square$

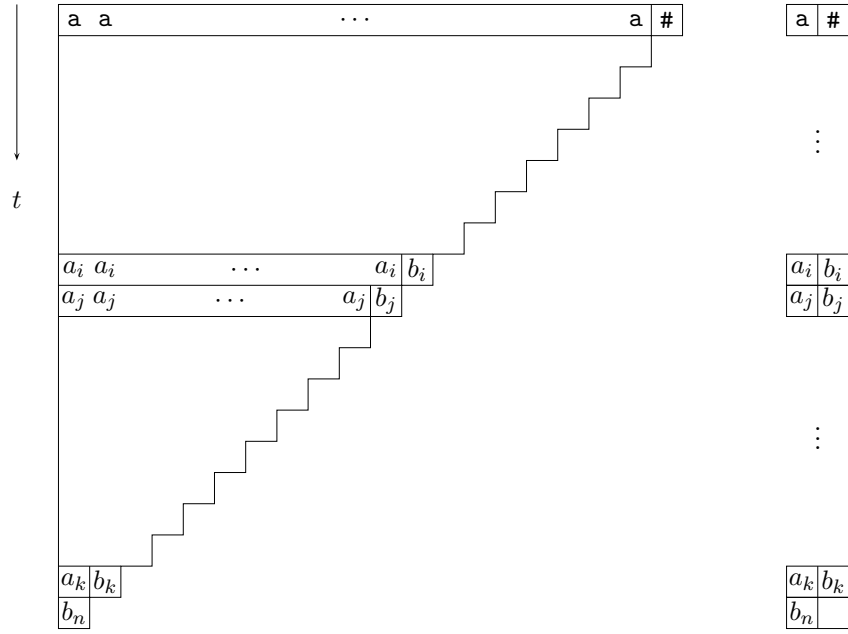


Figure 3. Example to the proof of Lemma 3.1. Memory contents are omitted.

The previous lemma together with Theorem 3.1 yields the main result in this section:

**Theorem 3.2.** For any subfamily *SEQ* of sequential machines it holds

$$\mathcal{L}_{rt}^u(\text{OCSEQ}) = \mathcal{L}^u(\text{SEQ}).$$

In [21] so-called *nondeterministic time-varying cellular automata* (which actually are CNFA) have been investigated and related to classical *nondeterministic cellular automata* (NCA). The difference is that at every time step all cells of a CNFA apply the same nondeterministically chosen local transition function, whereas the cells of a classical nondeterministic cellular automaton may nondeterministically choose a local transition function individually. It has been left open whether or not the inclusion between the real-time one-way families  $\mathcal{L}_{rt}(\text{OCNFA})$  and  $\mathcal{L}_{rt}(\text{NOCA})$  is a proper one. Now we can answer it in the affirmative.

**Corollary 3.1.**  $\mathcal{L}_{rt}(\text{OCNFA}) \subset \mathcal{L}_{rt}(\text{NOCA})$

**Proof:**

By Theorem 3.2 we obtain  $\mathcal{L}_{rt}^u(\text{OCNFA}) = \mathcal{L}^u(\text{NFA})$  which, in turn, is the family of regular unary languages. On the other hand, in [3]  $\mathcal{L}_{lt}(\text{CA}) \subseteq \mathcal{L}_{rt}(\text{NOCA})$  has been shown. Since  $\mathcal{L}_{lt}(\text{CA})$  contains non-regular unary languages [7], the assertion follows.  $\square$

Intrinsically, the first question ‘How much power do we have to plug in the single cells’ in order to accept non-regular unary languages has been answered. We need sequential machines that accept non-regular unary languages. For example, a *deterministic one-way stack automaton* (DOSA) which is a

deterministic pushdown automaton with an additional feature (cf. Figure 4). In addition to push and pop at the top of the stack a stack head can enter the stack in read-only mode and move up and down the stack without rewriting any symbol. So, intuitively, one-way stack automata are only slightly more complex than pushdown automata. But on the other hand, their parallel variant OCDOSA draws a borderline since it is able to accept non-regular unary languages in real time.

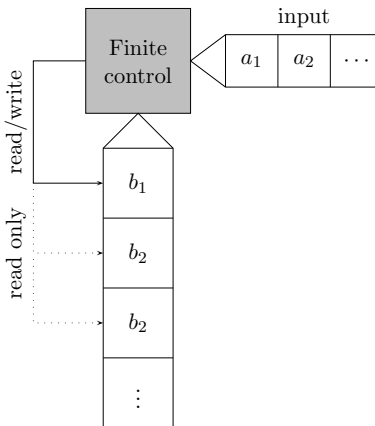


Figure 4. A one-way stack automaton.

Now we define stack automata in terms of sequential machines satisfying predicates  $P$  and  $Q$ . The transition function of a stack automaton depends on the currently read stack symbol. In addition, it depends on whether the stack head is at the top of the stack or not. For example, in the former case a write operation may be performed whereas this is impossible in the latter case. In order to distinguish between both cases, we use two memory symbols for each stack symbol. Moreover, we have to keep track of the position of the simulated stack head. To this end, the symbol  $m_1$  is used to indicate the position (for convenience we identify it by  $\bullet$  in the sequel). Now for each  $i \geq 2$  the memory symbols  $m_i$  and  $m_{i+1}$  represent the same stack symbol. The difference is that an even index indicates that the stack head is at the top of the stack in read/write mode, and an odd index indicates that the stack head is inside the stack in read only mode. Thus, let  $V_0 = \{m_2, m_4, m_6, \dots\}$ , and let  $a, m_i \in V_0$  and  $x, y \in V_0^*$ .

Predicate  $Q(u, v)$  restricts the memory access. If the current memory content is empty, i.e.,  $u = \lambda$  or  $u = \bullet$ , then the access yields to  $v = \lambda$ , i.e.,  $\alpha(u) = \lambda$ . If the current memory content is not empty and the head is at the top of the stack, i.e.,  $u = \bullet m_i y$ , then  $\alpha(u) = m_i$ . Since  $m_i \in V_0$ , the even index indicates the fact that the head is at the top of the stack. If otherwise the head is inside the stack, i.e.,  $u = x a \bullet m_i y$ , then  $\alpha(u) = m_{i+1}$ , where  $m_i$  and  $m_{i+1}$  represent the same stack symbol, but the odd index  $i + 1$  indicates that the head is inside the stack. Predicate  $Q$  reads

$$Q(u, v) \iff (u \in \{\lambda, \bullet\} \wedge v = \lambda) \vee (u = \bullet m_i y \wedge v = m_i) \vee (u = x a \bullet m_i y \wedge v = m_{i+1}).$$

Predicate  $P(u, v)$  restricts the memory update. A stack automaton is allowed to change nothing, i.e.,  $u = v$ . If the stack is empty, i.e.,  $u = \lambda$  or  $u = \bullet$ , some word can be pushed, whereby the head stays at the top of the stack, i.e.,  $v = \bullet y$ . If the head is at the top of a non-empty stack, i.e.,  $u = \bullet a y$ , the topmost symbol can be replaced by some word, i.e.,  $v = \bullet x y$ . If it is not at the bottom of the stack, i.e.,  $u = x \bullet a y$ , then it can move one position down, i.e.,  $v = x a \bullet y$ . Moreover, whenever the head is inside

the stack, i.e.,  $u = xa\bullet y$ , it can move one position up, i.e.,  $v = x\bullet ay$ . Predicate  $P$  reads

$$P(u, v) \iff (u = v) \vee (u = \lambda \wedge v = \bullet y) \vee (u = \bullet \wedge v = \bullet y) \vee \\ (u = \bullet ay \wedge v = \bullet xy) \vee (u = xa\bullet y \wedge v = x\bullet ay) \vee (u = x\bullet ay \wedge v = xa\bullet y).$$

**Lemma 3.2.**  $\mathcal{L}_{rt}^u(\text{OCA}) \subset \mathcal{L}_{rt}^u(\text{OCDOA})$

**Proof:**

It is known that  $\mathcal{L}^u(\text{DOA})$  contains non-regular languages [13, 14] but  $\mathcal{L}_{rt}^u(\text{OCA})$  does not.  $\square$

## 4. One-Way Arrays Beyond Real Time

In particular, from the previous section we derive that the computational capacity of finite automata is as powerful as the computational capacity of much more complex real-time one-way cellular automata as far as unary languages are considered. Now we turn to the proof that the situation does not change, even if we provide more time of sublogarithmic order.

**Theorem 4.1.** Let  $f : \mathbb{N}_+ \rightarrow \mathbb{N}_+$ ,  $f \in o(\log)$  be a mapping, then

$$\mathcal{L}_{rt+f}^u(\text{OCA}) = \mathcal{L}^u(\text{DFA}).$$

**Proof:**

Let  $L \subseteq \{a\}^+$  be a unary language accepted by some  $(rt + f)$ -time OCA  $\mathcal{M} = \langle S, \delta, \#, A, F \rangle$ . Without loss of generality we may assume that  $L$  is infinite.

For each  $a^n \in L$  we consider the smallest time  $q_n$  such that  $a^n$  is accepted in  $n + q_n$  time steps, i.e.,  $(\Delta^{[n+q_n]}(w_{0,a^n}))(1) \in F$ . We are going to prove that the set  $R = \{q_n \mid a^n \in L\}$  is bounded by some constant  $k$ . Since  $\mathcal{L}_{rt+k}(\text{OCA}) = \mathcal{L}_{rt}(\text{OCA})$  [18] and all unary languages in  $\mathcal{L}_{rt}(\text{OCA})$  are regular [24], in this case  $L$  must be regular, too.

Assume contrarily, the set  $R$  is unbounded, and let  $b = (|S| + 1)^3$ . Since  $f \in o(\log)$ , there exists  $n_0 \in \mathbb{N}_+$  such that  $f(n) < \lfloor \log_b(n) \rfloor$  and  $1 \leq \lfloor \log_b(n) \rfloor$ , for all  $n \geq n_0$ . Moreover, there are infinitely many words  $a^n \in L$ ,  $n \geq n_0$ , such that  $q_n > q_m$ , for all  $a^m \in L$  with  $n > m$ , since  $R$  is unbounded (cf. Figure 5).

Now assume for the rest of the proof that the cells are numbered 1 to  $n$  from right to left. For input  $a^n$  we denote the sequences  $c_{i-1}(i)c_i(i)c_{i+1}(i) \cdots c_{i+\lfloor \log_{|S|^2}(n) \rfloor - 1}(i)$  by  $e_i$ , for  $1 \leq i \leq n$ . Sequence  $e_i$  represents the evolution of the  $i$ th cell (from right) from time  $i - 1$  to time  $i + \lfloor \log_{|S|^2}(n) \rfloor - 1$ . Obviously, all these sequences have the same length  $\lfloor \log_{|S|^2}(n) \rfloor + 1$ . Therefore, there are at most

$$\begin{aligned} |S|^{\lfloor \log_{|S|^2}(n) \rfloor + 1} &= |S| \cdot |S|^{\lfloor \log_{|S|^2}(n) \rfloor} \\ &\leq |S| \cdot \lfloor |S|^{\log_{|S|^2}(n)} \rfloor \\ &= |S| \cdot \lfloor |S|^{\frac{1}{2} \log_{|S|}(n)} \rfloor \\ &= |S| \cdot \lfloor (|S|^{\log_{|S|}(n)})^{\frac{1}{2}} \rfloor \\ &= |S| \cdot \lfloor n^{\frac{1}{2}} \rfloor \\ &< \lfloor n^{\frac{1}{2}} \rfloor \cdot \lfloor n^{\frac{1}{2}} \rfloor \\ &\leq n \end{aligned}$$

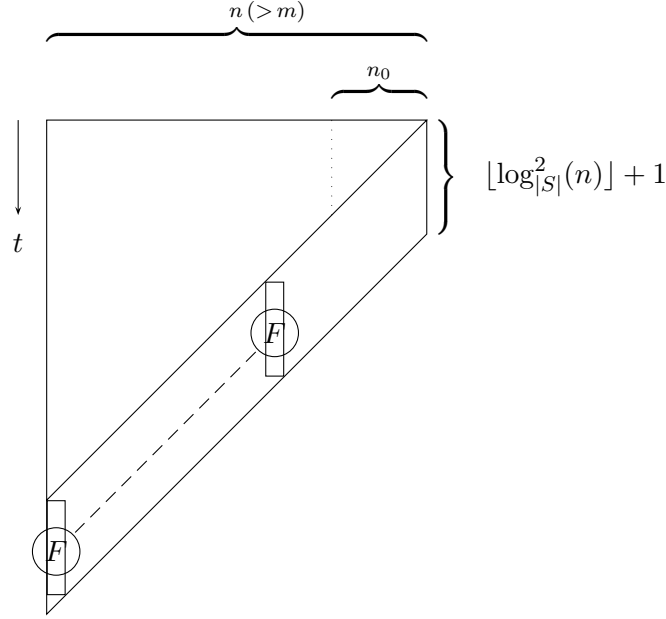


Figure 5. Example to the proof of Theorem 4.1.

different sequences, since we know  $1 \leq \lfloor \log_{(|S|+1)^3}(n) \rfloor$ , and therefore  $|S|^3 < n$ , and thus  $|S| < \lfloor n^{\frac{1}{2}} \rfloor$  since  $|S| \geq 2$ .

Now the pigeonhole principle ensures that at least two of the sequences  $e_1, \dots, e_n$  are equal, say  $u_i$  and  $u_j$  with  $1 \leq i < j \leq n$ . Due to the deterministic behavior and the same initial state for all cells we know that  $e_{i+1}$  is uniquely determined by  $e_i$ . Therefore, the equality  $e_i = e_j$  implies the equality  $e_{n+i-j} = e_n$ .

Since  $a^n$  is accepted in  $n + f(n)$  time steps, and  $f(n) < \lfloor \log_{(|S|+1)^3}(n) \rfloor \leq \lfloor \log_{|S|^2}(n) \rfloor + 1$ , at least one symbol of  $e_n = e_{n+i-j}$  belongs to  $F$ . Due to the one-way information flow the evolution of a cell is independent of the behavior of the cells on its left. So we can conclude that  $a^{n+i-j} \in L$  and, moreover,  $q_n = q_{n+i-j}$  which contradicts our choice of  $n$ .  $\square$

The theorem reveals a gap in between real-time and real-time plus logarithmic time for unary OCA languages. The following lemma shows that this gap is in some sense maximal, since if we allow logarithmically more time steps, the computational capacity strictly increases, which answers our first question. This result is independent of the base of the logarithm since we can speed-up the computation time beyond real time by any constant factor.

**Lemma 4.1.** The non-regular language  $\{a^{2^n} \mid n \in \mathbb{N}_+\}$  belongs to  $\mathcal{L}_{rt+\log}^u(\text{OCA})$ .

**Proof:**

The following OCA  $\mathcal{M} = \langle S, \delta, \#, \{a\}, F \rangle$  accepts  $L = \{a^{2^n} \mid n \in \mathbb{N}_+\}$  with time complexity  $rt + \log$ :

$$S = \{a, e, 1, +, 0, \bar{0}, 1^+, \#\}, \quad A = \{a\}, \quad F = \{+\}$$



Apart from the self-explanatory states, state  $\bar{0}$  represents digit zero with carry over flag, and state  $1^+$  represents digit one with the additional information that all former digits which have passed through the cell have been ones. A cell changes to the accepting state iff the entire counter which passes through consists of ones only. In this case, the counter represents a number  $2^k - 1$ , for some  $k \in \mathbb{N}_+$ .

Altogether, the counter reaches the leftmost cell at time step  $n$ . After that it takes another  $\log(n)$  time steps in order to let the counter pass through. So, the OCA obeys the time complexity  $rt + \log$ .  $\square$

**Corollary 4.1.** Let  $f : \mathbb{N}_+ \rightarrow \mathbb{N}_+$ ,  $f \in o(\log)$  be a mapping, then

$$\mathcal{L}^u(\text{DFA}) = \mathcal{L}_{rt}^u(\text{OCA}) = \mathcal{L}_{rt+f}^u(\text{OCA}) \subset \mathcal{L}_{rt+\log}^u(\text{OCA}) \subseteq \mathcal{L}_{lt}^u(\text{OCA}).$$

## 5. A Wee Bit Nondeterminism

We turn to our last question ‘How much nondeterminism do we have to add?’ in order to accept non-regular unary languages. To this end, we consider cellular automata with limited nondeterminism. In general, there are two natural limitations. One can limit the nondeterminism in time, that is the number of time steps at which nondeterministic transitions are allowed, or one can limit the nondeterminism in space, that is the number of cells which are allowed to perform nondeterministic transitions. In [3] cellular automata have been investigated where all cells may perform one nondeterministic transition. It turned out that the corresponding devices are surprisingly powerful. Here we are going to limit the nondeterminism more strictly in the following way. The set of local transition functions is divided into two parts  $D = \{\delta_d\} \cup D_{nd}$ , where  $\delta_d$  is a distinguished so-called deterministic local transition function and  $D_{nd}$  is a possibly empty set of so-called nondeterministic local transition functions. During the first time step a nondeterministically chosen transition function from  $D_{nd}$  is applied to at most  $k$  nondeterministically chosen cells, where  $k \in \mathbb{N}$  is a constant. The other cells change states according to  $\delta_d$ . The cells performing a local transition function from  $D_{nd}$  are called nondeterministic and the others deterministic with respect to the current computation. After the first time step  $\delta_d$  is applied to all cells. We denote such models by  $k\text{C } 1\text{G-CA}$  ( $k$  cells one guess CA). The aim of the next two subsections is to investigate  $k\text{C } 1\text{G-CA}$  and their one-way variants the  $k\text{C } 1\text{G-OCA}$  with respect to unary languages. Observe that in case of  $k = 0$  these models are deterministic CA resp. OCA. It will turn out that a speed-up from linear time to real time is possible at the cost of one additional nondeterministic cell. Similarly, one can simplify the information flow from two-way to one-way, if one nondeterministic cell is added. In general, we obtain an infinite hierarchy dependent on the number of nondeterministic cells.

### 5.1. Time versus Nondeterminism

The following theorem states that it is possible to trade time for nondeterminism.

**Theorem 5.1.** Let  $t : \mathbb{N}_+ \rightarrow \mathbb{N}_+$ ,  $t(n) \geq n$ , be a mapping and  $k \in \mathbb{N}_+$  and  $k_1 \in \mathbb{N}$  be constants, then

$$\mathcal{L}_{k \cdot t}^u(k_1\text{C } 1\text{G-OCA}) \subseteq \mathcal{L}_t^u((k_1 + 1)\text{C } 1\text{G-OCA}).$$

**Proof:**

Let  $\mathcal{M}$  be a  $k_1\text{C } 1\text{G-OCA}$  accepting a unary language  $L \subseteq \{a\}^+$  with time complexity  $k \cdot t$ . In order to achieve the desired speed-up an equivalent  $(k_1 + 1)\text{C } 1\text{G-OCA}$  acceptor  $\mathcal{M}'$  has to simulate  $\mathcal{M}$  exactly  $k$

times faster. To this end, in principle, each cell of  $\mathcal{M}'$  is designed to simulate  $k$  consecutive cells of  $\mathcal{M}$ . This implies that  $\mathcal{M}'$  simulates  $k$  time steps of  $\mathcal{M}$  at every time step.

More precisely, on input  $a^n$  automaton  $\mathcal{M}'$  works as follows. Since  $L$  is a unary language, each cell ‘knows’ the initial states of the cells it has to simulate, i.e., state  $a$ . The additional nondeterministic power of  $\mathcal{M}'$  is used to guess and to mark cell  $i = \lceil n/k \rceil$ . Additionally, the marked cell guesses  $r = n \bmod k$ . If  $r = 0$  it simulates  $k$ , otherwise  $r$  cells of  $\mathcal{M}$ , whereas the cells to its left simulate  $k$  cells, respectively.

In order to verify the guesses, initially, a signal is generated in the rightmost cell. The signal moves with maximal speed to the left. It can verify that exactly one cell has been marked. In particular, it prohibits each cell it passes through to switch into an accepting state unless it has already passed through exactly one marked cell. Moreover, initially another signal is generated in the marked cell  $i$ . This signal is delayed for  $r$  time steps. Subsequently, it moves to the left one cell per  $k$  time steps. Now, the leftmost cell may switch to an accepting state only if both signals arrive at the same time, that is, if  $k(i-1) + r = n$ . So, if the leftmost cell accepts,  $\mathcal{M}'$  has simulated exactly  $n$  cells of  $\mathcal{M}$ .

Finally, it is straightforward to use the first signal to verify that  $\mathcal{M}'$  has simulated at most  $k$  nondeterministic cells of  $\mathcal{M}$ .  $\square$

**Corollary 5.1.** Let  $k \in \mathbb{N}$  be a constant, then

$$\mathcal{L}_{lt}^u(k\text{C 1G-OCA}) \subseteq \mathcal{L}_{rt}^u((k+1)\text{C 1G-OCA}).$$

Theorem 5.1 can directly be transferred to two-way information flow.

**Theorem 5.2.** Let  $t : \mathbb{N}_+ \rightarrow \mathbb{N}_+$ ,  $t(n) \geq n$ , be a mapping and  $k \in \mathbb{N}_+$  and  $k_1 \in \mathbb{N}$  be constants, then

$$\mathcal{L}_{k,t}^u(k_1\text{C 1G-CA}) \subseteq \mathcal{L}_t^u((k_1+1)\text{C 1G-CA}).$$

Here we traded speed for nondeterminism. The converse, the reduction of the number of nondeterministic cells at the cost of linear slow-down does not follow for structural reasons from the properties of the models. The next result shows the converse for real time and linear time. It improves Corollary 5.1: not only inclusion but equality holds. So, in this situation we have a trade-off between time and nondeterminism.

**Theorem 5.3.** Let  $k \in \mathbb{N}$  be a constant, then

$$\mathcal{L}_{lt}^u(k\text{C 1G-OCA}) = \mathcal{L}_{rt}^u((k+1)\text{C 1G-OCA}).$$

**Proof:**

It suffices to prove  $\mathcal{L}_{rt}^u((k+1)\text{C 1G-OCA}) \subseteq \mathcal{L}_{lt}^u(k\text{C 1G-OCA})$ . Roughly, the outline of the proof is as follows. If one of the nondeterministic cells is located at the rightmost position, it can be made deterministic as usual. In this case all nondeterministic choices are simulated in parallel on different tracks. Therefore, we shift the computation to the right until the rightmost nondeterministic cell is at the rightmost position of the automaton. In order to be able to shift computations, we make them independent of the right border. That is, the rightmost cell is modified such that it behaves as if there were infinitely many cells to the right of it and, moreover, the automaton is modified that it recognizes any accepted prefix, i.e., it recognizes any time step  $t$  such that  $a^t$  belongs to the accepted language. After the shift, we have to ensure that only the input word is accepted. We have to prevent from accepting the input, if

only a prefix belongs to the language. To this end, the computation is delayed from real time to linear time such that specific signals can be set up which realize this verification. According to the outline, we transform a  $(k + 1)$ C 1G-OCA real-time acceptor  $\mathcal{M}$  accepting a unary language  $L \subseteq \{a\}$  in five steps into a desired linear-time acceptor with the reduced amount of nondeterminism.

**Step 1** At first,  $\mathcal{M}$  is modified such that the leftmost cell recognizes at time  $t$ , if the already processed prefix of length  $t$  of the input  $a^n$  belongs to  $L$ . More precisely, if the current distribution and action of the nondeterministic cells within the leftmost  $t$  cells would lead to acceptance of  $a^t$ .

To this end, each cell of the resulting  $(k + 1)$ C 1G-OCA  $\mathcal{M}_1$  consists of two registers. The first register is used to simulate an unmodified transition of  $\mathcal{M}$ . In addition, during the first time step each cell computes its state under the assumption that its right neighbor is in the boundary state, and stores the result in its second register. Subsequently, the content of the second register is updated by applying the deterministic local transition function (of  $\mathcal{M}$ ) to the state stored in its first register and to the content of the second register of its neighbor. So, the second register of cell  $i$ ,  $1 \leq i \leq n$ , stores at time  $t$ ,  $1 \leq t \leq n - i + 1$ , the state cell  $i$  of  $\mathcal{M}$  would be in, if the input were  $a^{i+t-1}$ . In particular, cell 1 stores at time  $1 \leq t \leq n$  the state cell 1 of  $\mathcal{M}$  would be in, if the input were  $a^t$ . We conclude that cell 1 of  $\mathcal{M}_1$  recognizes at time  $t$ , whether or not  $a^t$  belongs to the languages  $L$ .

**Step 2** Next we have to overcome the concrete number of cells in between the rightmost nondeterministic cell and the right border. That is, we want cell 1 to recognize at time  $t$ , whether or not  $a^t$  belongs to the language  $L$ , also for  $t \geq n$ . To this end,  $\mathcal{M}_1$  is modified such that its rightmost cell behaves as if there were infinitely many cells to the right of it. Concerning the distribution of the nondeterministic cells in the simulation, clearly, we can only cover cases where the  $k + 1$  nondeterministic cells are located somewhere in the first  $n$  cells.

For the modification, the rightmost cell of the newly obtained  $(k + 1)$ C 1G-OCA  $\mathcal{M}_2$  assumes that there are another infinitely many deterministic cells to its right, which are initialized by the input symbol  $a$ . Clearly, at every time step all these cells are in the same state as the rightmost cell of  $\mathcal{M}_2$ . So, the desired state transition can easily be realized.

By the modification, it follows that the leftmost cell of  $\mathcal{M}_2$  (which is no longer real-time bounded) can recognize at time  $t \geq 1$  whether or not there is an accepting computation of  $\mathcal{M}$  on input  $a^t$ , such that the  $k + 1$  nondeterministic cells are located somewhere in the first  $n$  cells.

**Step 3** Now we are prepared to consider the right-shift of the computation in order to reduce the number of nondeterministic cells by one. We denote the  $k$ C 1G-OCA which will be constructed from  $\mathcal{M}_2$  by  $\mathcal{M}_3$  (cf. Figure 7 (a) and (b)). Let cell  $i$  of  $\mathcal{M}_2$  be the  $(k + 1)$ st, i.e., the rightmost, nondeterministic cell, and let  $d = n - i$ . By the modifications which led to  $\mathcal{M}_2$  the computation is independent of the right border cell. Therefore, we simply can assume without further modifications that the computation has been shifted  $d$  cells to the right. That is, since the leftmost cell of  $\mathcal{M}_2$  recognizes at time  $t \geq 1$  whether or not there is an accepting computation of  $\mathcal{M}$  on input  $a^t$ , such that the  $k + 1$  nondeterministic cells are located somewhere in the first  $n$  cells, now cell  $d + 1$  of  $\mathcal{M}_3$  recognizes at time  $t \geq 1$  whether or not there is an accepting computation of  $\mathcal{M}$  on input  $a^t$ , such that the  $k + 1$  nondeterministic cells are located somewhere in the first  $n$  cells. Moreover, after the shift the  $(k + 1)$ st nondeterministic cell is located at the rightmost position of the automaton. It is made deterministic by the usual construction and now deterministically simulates all nondeterministic choices in parallel on different tracks. To this end, all cells have an appropriate number of registers. Therefore,  $\mathcal{M}_3$  uses only  $k$  nondeterministic cells.

**Step 4** Cell  $d + 1$  of  $\mathcal{M}_3$  recognizes any accepted prefix, i.e., it recognizes any time step  $t \geq 1$  such that  $a^t$  belongs to the accepted language. Next we have to ensure that only the input word  $a^n$  is

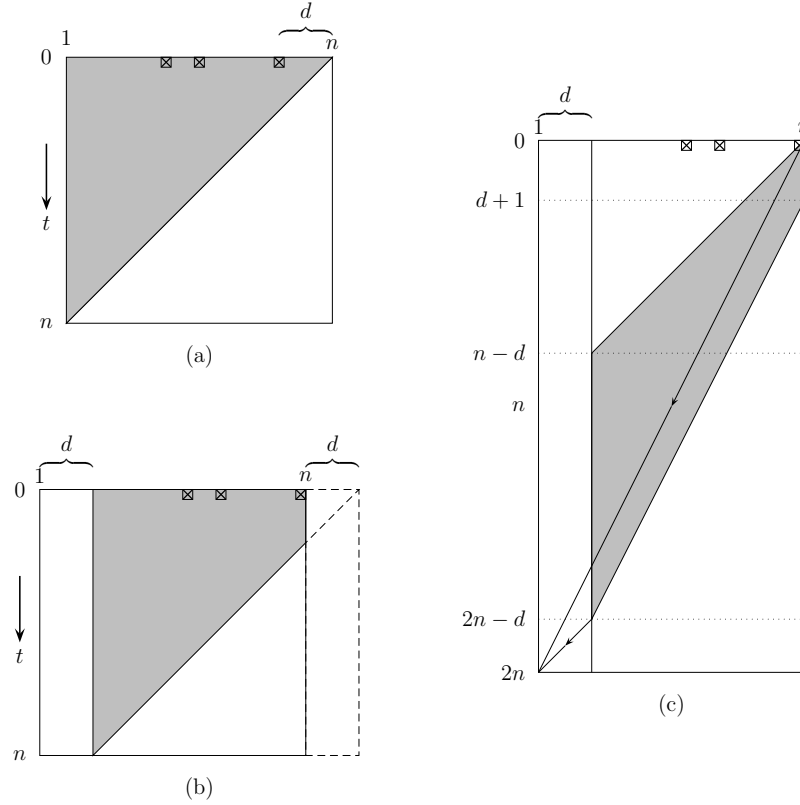


Figure 7. Overview of the construction in the proof of Theorem 5.3. (a) Real-time  $(k+1)C$  1G-OCA, (b) right-shift of the computation in step 3, (c) delayed computation in step 5. Nondeterministic cells are marked by  $\boxtimes$ .

accepted. Moreover, it has to be accepted at time  $n$  by the leftmost cell. In order to achieve the desired behavior, the next step is to delay the computation of  $\mathcal{M}_3$  by one time step for each cell from right to left. I.e., a cell  $i$  of the resulting  $kC$  1G-OCA  $\mathcal{M}_4$  simulates at time  $t + (n - i + 1)$  the state of cell  $i$  of  $\mathcal{M}_3$  at time  $t$ .

For the construction, the cells of  $\mathcal{M}_4$  have two registers (cf. Figure 8). During the first time step each cell simulates a transition of  $\mathcal{M}_3$  and stores the resulting state in its first register. Subsequently, the second register is used to store the previously simulated state. If a cell is allowed to proceed with the simulation only if the second register of its neighbor is not empty, then the required delay can be achieved.

**Step 5** Finally,  $\mathcal{M}_4$  is modified in the following way resulting in  $\mathcal{M}_5$  (cf. Figure 7 (c)). During the first time step the rightmost cell sends a signal  $s$  with speed  $1/2$  to the left. Furthermore, when some cell recognizes that an input prefix belongs to the accepted language, it sends a signal  $u$  with maximal speed to the left. If the signal  $s$  meets some signal  $u$  in some cell, the cell enters an accepting state. The leftmost cell is reached by  $s$  at time  $2n$ . Hence,  $\mathcal{M}_5$  is a linear-time  $kC$  1G-OCA acceptor.

Let  $a^n \in L$ . Then there exists some cell  $i$  of  $\mathcal{M}_4$  that recognizes this fact at time  $n + (n - i + 1)$ . Its signal  $u$  arrives at the leftmost cell at time  $n + (n - i + 1) + i - 1 = 2n$ . Thus,  $a^n \in L(\mathcal{M}_5)$ .

Conversely, if  $a^n \in L(\mathcal{M}_5)$ , then the leftmost cell of  $L(\mathcal{M}_5)$  has been reached by a signal  $u$  which

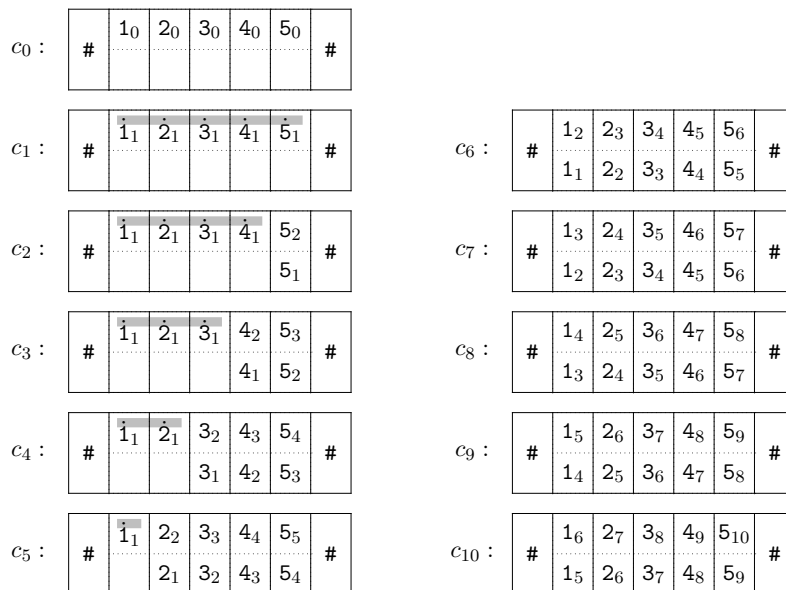


Figure 8. Delayed computation in step 4 of Theorem 5.3.

was generated in some cell  $i$  at time  $2n - i + 1 = n + (n - i + 1)$ . This is true also for cell  $i$  of  $L(\mathcal{M}_4)$  and, thus,  $a^n \in L$ , i.e.,  $L = L(\mathcal{M}_5)$ .  $\square$

## 5.2. Information Flow versus Nondeterminism

In the sequel the relation between one-way  $kC$  1G-OCA and two-way  $kC$  1G-CA is investigated with respect to the amount of nondeterminism. The following result is already known for the deterministic case  $k = 0$ . In [29]  $\mathcal{L}_{rt}(\text{CA}) \subseteq \mathcal{L}_{2-id}^R(\text{OCA})$  has been shown, and in [7] the converse was proved. By speed-up theorems for cellular automata [19] one obtains the corresponding result for linear time. Clearly, for unary languages it holds  $\mathcal{L}_{rt}^u(\text{CA}) = \mathcal{L}_{lt}^u(\text{OCA})$ .

**Theorem 5.4.** Let  $k \in \mathbb{N}$  be a constant, then

$$\mathcal{L}_{lt}^u(kC \text{ 1G-OCA}) = \mathcal{L}_{rt}^u(kC \text{ 1G-CA}).$$

### Proof:

The proof is analogous to the previously cited deterministic case. However, the adaption could require a cell of a  $kC$  1G-OCA to perform a nondeterministic transition during the second time step. Therefore, observe that a local transition function can be represented as a finite array of  $S^2 \times S$  states. So, a cell may guess such an array (and hence a local transition function) during the first time step. Subsequently, it is applied to the cell in the second time step.  $\square$

It is possible to speed-up linear time to real time at the cost of one additional nondeterministic cell. The next corollary says that it is possible to simplify two-way to one-way information flow for real-time computations at the same cost. The converse shows that we can save one nondeterministic cell when we provide two-way instead one-way information flow.

**Corollary 5.2.** Let  $k \in \mathbb{N}$  be a constant, then

$$\mathcal{L}_{rt}^u(kC\ 1G\text{-CA}) = \mathcal{L}_{rt}^u((k+1)C\ 1G\text{-OCA}).$$

**Proof:**

The assertion follows by Theorem 5.3 and Theorem 5.4, since  $\mathcal{L}_{rt}^u(kC\ 1G\text{-CA}) = \mathcal{L}_{lt}^u(kC\ 1G\text{-OCA})$  and  $\mathcal{L}_{lt}^u(kC\ 1G\text{-OCA}) = \mathcal{L}_{rt}^u((k+1)C\ 1G\text{-CA})$ .  $\square$

From Corollary 5.2 the inclusion  $\mathcal{L}_{lt}^u(kC\ 1G\text{-CA}) \supseteq \mathcal{L}_{rt}^u((k+1)C\ 1G\text{-OCA})$  and the inclusion  $\mathcal{L}_{rt}^u(kC\ 1G\text{-CA}) \subseteq \mathcal{L}_{lt}^u((k+1)C\ 1G\text{-OCA})$  follow. So, we obtain the relations depicted in Figure 9.

$$\begin{array}{ccc}
 \vdots & & \vdots \\
 \cup & & \cup \\
 \mathcal{L}_{lt}^u(2C\ 1G\text{-CA}) & & \mathcal{L}_{rt}^u(3C\ 1G\text{-OCA}) \\
 \cup & & \parallel \\
 \mathcal{L}_{rt}^u(2C\ 1G\text{-CA}) & = & \mathcal{L}_{lt}^u(2C\ 1G\text{-OCA}) \\
 \cup & & \cup \\
 \mathcal{L}_{lt}^u(1C\ 1G\text{-CA}) & & \mathcal{L}_{rt}^u(2C\ 1G\text{-OCA}) \\
 \cup & & \parallel \\
 \mathcal{L}_{rt}^u(1C\ 1G\text{-CA}) & = & \mathcal{L}_{lt}^u(1C\ 1G\text{-OCA}) \\
 \cup & & \cup \\
 \mathcal{L}_{lt}^u(\text{CA}) & & \mathcal{L}_{rt}^u(1C\ 1G\text{-OCA}) \\
 \cup & & \parallel \\
 \mathcal{L}_{rt}^u(\text{CA}) & = & \mathcal{L}_{lt}^u(\text{OCA}) \\
 & & \cup \\
 & & \mathcal{L}_{rt}^u(\text{OCA}) \\
 & & \parallel \\
 & & \mathcal{L}^u(\text{DFA})
 \end{array}$$

Figure 9. A hierarchy of unary families.

### 5.3. Superconstant Nondeterminism for Arbitrary Alphabets

Up to now we have investigated  $kC\ 1G\text{-CA}$ , which are devices with a constant amount of nondeterminism. A natural generalization is to consider devices with superconstant nondeterminism. As mentioned before, in [3] cellular automata have been investigated where all cells may perform one nondeterministic transition. In our terms these are  $nC\ 1G\text{-CA}$ , where  $n$  denotes the length of the cellular automaton. So, the number of nondeterministic cells is limited by a function  $f : \mathbb{N}_+ \rightarrow \mathbb{N}_+$ ,  $f(n) \leq n$ , which depends on the number of cells. Exactly this is our point of view in the present subsection. We write  $fC\ 1G\text{-CA}$  and, obviously, obtain devices in between  $kC\ 1G\text{-CA}$ ,  $k \in \mathbb{N}_+$ , and  $nC\ 1G\text{-CA}$ .

We are going to compare the real-time  $fC$  1G-CA and real-time  $fC$  1G-OCA with other variants of cellular automata with respect to their capacities to accept languages over arbitrary alphabets. To this end, a notion of computability is useful that is closely related to the notion of time-constructible functions in deterministic cellular automata [4, 5, 6].

**Definition 5.1.** A mapping  $f : \mathbb{N}_+ \rightarrow \mathbb{N}_+$  is defined to be *OCA-time-computable*, iff there is an OCA  $\mathcal{M} = \langle S, \delta, \#, \{a\}, F \rangle$  with global transition function  $\Delta$  such that  $f(n)$ , for all  $n \in \mathbb{N}_+$ , is the smallest positive integer  $t$  for which

$$\pi_1(\Delta^{[t]}(c_{0,a^n})) \in F.$$

The following strong relation between OCA-time-computability and the notion of time-constructible functions in so-called deterministic cellular spaces has been shown in [4].

Let  $f^{-1}(m) = \max(\{n \in \mathbb{N}_+ \mid f(n) \leq m\} \cup \{1\})$  be the inverse of a strictly increasing mapping  $f : \mathbb{N}_+ \rightarrow \mathbb{N}_+$ . Then  $f$  is time-constructible if and only if  $f^{-1} + 2n$  is OCA-time-computable. The family of time-constructible functions is very rich. For example, it contains  $n^k$ , for  $k \in \mathbb{N}_+$ ,  $2^n$ ,  $n!$ , and  $p_n$ , where  $p_n$  is the  $n$ th prime number. Moreover, the family is closed under several operations. So, the condition of the next theorem is a weak one. In the sequel sometimes we use functions  $f$  whose range includes real numbers in the context of integers. For simplicity, we often write  $f$  instead of  $\lfloor f \rfloor$ .

**Theorem 5.5.** Let  $A$  be an alphabet and  $f : \mathbb{N}_+ \rightarrow \mathbb{N}_+$  be an increasing mapping such that  $f + 2n$  is OCA-time-computable, and define  $f' : \mathbb{N}_+ \rightarrow \mathbb{N}$  by  $f'(n) = \max\{m \in \mathbb{N} \mid f(m) \cdot m \leq n\}$ . Then

$$L = \{w^{f(|w|)} \mid w \in A^+\} \in \mathcal{L}_{rt}(\tilde{f}C \text{ 1G-OCA})$$

for  $\tilde{f} : \mathbb{N}_+ \rightarrow \mathbb{N}_+$ , where  $\tilde{f}(n) = f(f'(n))$  if  $f'(n) > 0$ , and  $\tilde{f}(n) = 1$  otherwise.

**Proof:**

For some arbitrary  $w \in A^+$  with  $|w| = m$  we have  $w^{f(|w|)} \in L$  and  $|w^{f(|w|)}| = m \cdot f(m)$ . This implies  $\tilde{f}(m \cdot f(m)) = f(f'(m \cdot f(m))) = f(\max\{m' \in \mathbb{N} \mid f(m') \cdot m' \leq f(m) \cdot m\}) = f(m) = f(|w|)$ .

Basically, the idea of a real-time  $\tilde{f}C$  1G-OCA accepting  $L$  is, that all cells which represent the last symbol of one of the subwords  $w$  are nondeterministically marked during the first time step. Since an input of length  $n$  consists of  $f(|w|) = \tilde{f}(n)$  copies of  $w$ , the number of allowed nondeterministic cells suffices. It remains to be shown how the guesses can be verified, i.e., how to verify that at most  $\tilde{f}(n)$  appear, and how to verify that all input fragments between nondeterministic cells are identical.

We start with the second task. Since  $L' = \{wcw \mid w \in A^+, c \notin A\}$  is a real-time OCA language, we can use a slightly modified version of a corresponding acceptor. The nondeterministic cells simulate the last symbols of  $w$ , i.e., their input symbol, and, in addition, a virtual symbol  $c$ . So, each two adjacent subwords  $w$  can be checked. Moreover, due to the overlapping (for  $w_1w_2w_3$  the subwords  $w_1w_2$  and  $w_2w_3$  are both checked), it is ensured that the entire input has the correct form.

Now we turn to verify the number of nondeterministic cells. If  $f$  is a constant mapping, a left-moving signal which is initially generated at the rightmost cell can do the task. So, assume  $f \in \omega(1)$ . At first we need to stop a OCA computation at some time  $t_1$  along the diagonal  $c_{t_1}(n)c_{t_1+1}(n-1) \cdots c_{t_1+n-1}(1)$ , and to continue the computation at time  $t_2$  along the diagonal  $c_{t_2}(n) \cdots c_{t_2+n-1}(1)$  (in the space time diagram). By stopping the computation again at time  $t_2 + 1$ , exactly one more transition per cell is simulated. In order to realize such a behavior, we use another register in which any cell can be marked.

The marker of the rightmost cell participating in the task controls the process. More precisely, for some OCA with state set  $S$  and local transition function  $\delta$  a modified OCA with state set  $S'$  and local transition function  $\delta'$  is constructed as follows. Let  $\dot{S} = \{\dot{s} \mid s \in S\}$  be a dotted copy of  $S$ .

$$S' = S \cup \dot{S}$$

$$\forall s, r \in S, \dot{s}, \dot{r} \in \dot{S}:$$

$$\begin{aligned} \delta'(s, r) &= \delta(s, r) \\ \delta'(s, \dot{r}) &= \dot{p} \text{ with } p = \delta(s, r) \\ \delta'(\dot{s}, \dot{r}) &= \dot{s} \\ \delta'(\dot{s}, r) &= s \end{aligned}$$

After stopping for the first time at step  $t_1 + n - 1$ , the leftmost cell is in state  $c_{t_1+n-1}(1)$ , i.e., it has performed all transitions up to that time (cf. Figure 10).

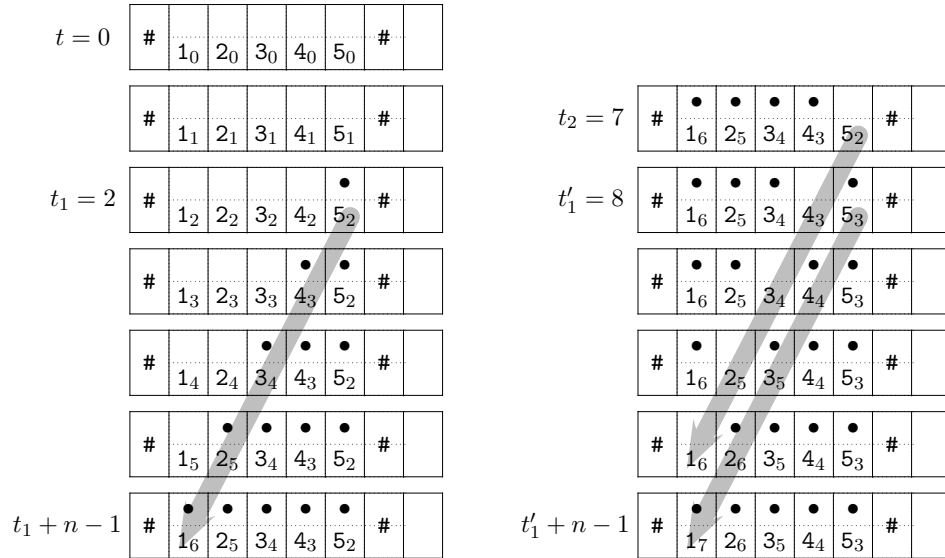


Figure 10. Stop ( $t_1 = 2$ ), continue ( $t_2 = 7$ ), stop ( $t'_1 = 8$ ) of a computation.

Now we come back to the verification. Since  $f \in \omega(1)$  for all but finitely many numbers  $n$  we have  $f(n) \geq 3$ . The real-time  $\tilde{f}C1G$ -OCA determines the leftmost nondeterministic cell at position  $|w|$  nondeterministically during its first transition. This guess is easily verified by some left-moving signal. This cell and all of its left neighbors simulate the OCA-time-computer for  $f + 2n$ , where the computation of cell  $|w|$  is controlled as follows. All nondeterministic cells send initially signals to the left. The cell at position  $|w|$  stops the computation to its left two time steps after the arrival of the first signal. Therefore, cell 1 stops at time  $2 \cdot |w| + 2$ . All subsequent signals cause cell  $|w|$  to continue the computation for one time step. There should be  $\tilde{f}(m \cdot f(m)) - 2 = f(|w|) - 2$  subsequent signals. The last one arrives from the rightmost cell. At its arrival at cell 1 step  $2 \cdot |w| + 2 + f(|w|) - 2 = 2 \cdot |w| + f(|w|)$  of

the OCA-time-computation has been performed. This is exactly the time step to be distinguished. We conclude, that all words from  $L$  are accepted.

Conversely, let  $v$  be some accepted word. It has to be of the form  $w^i$  for some  $w \in A^+$  and  $i \geq 3$  (constant mappings  $f$  have been treated separately). We derive  $i = f(|w|)$  from the time-computation. Therefore,  $|v| = |w| \cdot f(|w|)$  and  $\tilde{f}(|v|) = f(|w|)$ , which implies that there are  $\tilde{f}(|v|)$  nondeterministic cells.  $\square$

Now we are prepared for the comparison with the deterministic real-time OCA.

**Lemma 5.1.** Let  $A$  be an alphabet and  $f : \mathbb{N}_+ \rightarrow \mathbb{N}_+$ ,  $f \in \omega(1)$ , be an increasing mapping such that  $f + 2n$  is OCA-time-computable, then

$$L = \{w^{f(|w|)} \mid w \in A^+\} \notin \mathcal{L}_{rt}(\text{OCA}).$$

**Proof:**

Let  $a \in A$  be an arbitrary symbol, and assume in contrast to the assertion  $L$  belongs to the family  $\mathcal{L}_{rt}(\text{OCA})$ . Since  $\mathcal{L}_{rt}(\text{OCA})$  is closed under intersection with regular languages [20], we conclude  $L' = L \cap \{a\}^+ \in \mathcal{L}_{rt}(\text{OCA})$ . But  $L'$  is the non-regular unary language  $\{a^{n \cdot f(n)} \mid n \in \mathbb{N}_+\}$  since  $f \in \omega(1)$ . So,  $L' \notin \mathcal{L}_{rt}(\text{OCA})$ .  $\square$

**Example 5.1.** The identity mapping  $f(n) = n$  is increasing and of order  $\omega(1)$ . In [6] it has been shown that  $f + 2n$  is OCA-time-computable. Moreover,  $|w^{f(|w|)}| = |w|^2$ . By Theorem 5.5 language  $L = \{w^{|w|} \mid w \in A^+\}$  belongs to  $\mathcal{L}_{rt}(n^{\frac{1}{2}}\text{C 1G-OCA})$ . On the other hand, by Lemma 5.1 language  $L$  does not belong to  $\mathcal{L}_{rt}(\text{OCA})$  (see also [26]).

In the sequel the previous comparison is further improved. For two language families  $\mathcal{L}_a$  and  $\mathcal{L}_b$  their concatenation is defined by  $\mathcal{L}_a \cdot \mathcal{L}_b = \{L_a \cdot L_b \mid L_a \in \mathcal{L}_a \wedge L_b \in \mathcal{L}_b\}$ . It is known that real-time OCA languages are not closed under concatenation [27]. On the other hand, they are closed under left as well as under right concatenation with regular languages [24]. So, it is natural to ask whether the concatenation closure of  $\mathcal{L}_{rt}(\text{OCA})$  is characterized by  $\mathcal{L}_{rt}(\text{1C 1G-OCA})$ , i.e., by devices that can guess the position of the concatenation.

One inclusion is easily derived:

**Corollary 5.3.**  $\mathcal{L}_{rt}(\text{OCA}) \cdot \mathcal{L}_{rt}(\text{OCA}) \subseteq \mathcal{L}_{rt}(\text{1C 1G-OCA})$

**Proof:**

In order to accept the concatenation of two languages  $L_a$  and  $L_b$  from  $\mathcal{L}_{rt}(\text{OCA})$  it suffices to construct a real-time 1C 1G-OCA that guesses the position of the concatenation and simulates in both parts appropriate acceptors separately. The assertion follows since  $\mathcal{L}_{rt}(\text{OCA})$  is closed under marked concatenation [24].  $\square$

The next lemma provides us with an important witness language from  $\mathcal{L}_{rt}(\text{1C 1G-OCA})$ .

**Lemma 5.2.** Let  $A$  be an alphabet, then

$$L = \{wvw \mid w \in A^+ \wedge v \in A^*\} \in \mathcal{L}_{rt}(\text{1C 1G-OCA}).$$

**Proof:**

Let  $wvw$  be a word belonging to  $L$ . The cells of a real-time 1C 1G-OCA accepting  $L$  are designed to have three registers. The first register is used to keep the input symbol. The cell which stores the last symbol of the first subword  $w$  is nondeterministically determined during the first time step. It sends a signal to the left that identifies the leftmost  $|w|$  cells. The second register is used to shift the input one cell per time step to the left. The leftmost  $|w|$  cells delay the shifting such that symbols flow with speed  $\frac{1}{2}$ , i.e., one cell for every other time step. To this end, they use their third registers. In every time step the content of the second register is moved to the third one, and the content of the third register is taken into the second register of the left neighbor.

When a left-moving signal initially generated in the rightmost cell arrives at cell  $|w|$ , it starts to compare the original input symbol with the content of the second register, respectively. More precisely, a 1C 1G-OCA  $\mathcal{M} = \langle S, \delta, \#, A, F \rangle$  is constructed as follows (cf. Figure 11). Let  $\dot{A} = \{\dot{a} \mid a \in A\}$  be a dotted copy of  $A$ .

$$S = \{\#\} \cup A \cup ((A \cup \dot{A}) \times (A \cup \{+, -\}) \times (A \cup \{\sqcup\}))$$

$$F = \{s \in S \setminus (\{\#\} \cup A) \mid \pi_1(s) \in \dot{A} \wedge \pi_2(s) = +\}$$

$$\forall s \in S :$$

$$\delta(\#, s) = \{\#\}$$

$$\forall s, r \in A :$$

$$\delta_{nd}(s, r) = \{(\dot{s}_1, r, \sqcup)\}$$

$$\delta_d(s, r) = (s, r, \sqcup)$$

$$\delta_d(s, \#) = (s, +, \sqcup)$$

$$\forall (p_1, p_2, p_3), (q_1, q_2, q_3) \in S \setminus (\{\#\} \cup A):$$

$$\delta_d((p_1, p_2, p_3), \#) = (p_1, +, \sqcup)$$

$$\delta_d((p_1, p_2, p_3), (q_1, q_2, q_3)) = (r_1, r_2, r_3), \text{ where}$$

$$r_1 = \begin{cases} \dot{p}_1 & \text{if } p_1 \notin \dot{A} \wedge q_1 \in \dot{A} \\ p_1 & \text{otherwise} \end{cases}$$

$$r_2 = \begin{cases} q_2 & \text{if } (p_1 \notin \dot{A}) \vee (p_1 \in \dot{A} \wedge q_1 \notin \dot{A} \wedge q_2 \in A) \\ + & \text{if } p_1 \in \dot{A} \wedge p_2 \in A \wedge q_2 = + \wedge p_1 = \dot{p}_2 \\ - & \text{if } p_1 \in \dot{A} \wedge p_2 \in A \wedge q_2 = + \wedge p_1 \neq \dot{p}_2 \\ q_3 & \text{if } p_1 \in \dot{A} \wedge q_1 \in \dot{A} \wedge q_2 \in A \\ p_2 & \text{otherwise} \end{cases}$$

$$r_3 = \begin{cases} r_2 & \text{if } r_1 \in \dot{A} \wedge r_2 \in A \\ p_3 & \text{otherwise} \end{cases}$$

Let  $|w| = m$ ,  $|wvw| = n$ , and denote the configurations of a computation on  $wvw$  by  $c_t$ ,  $0 \leq t \leq n$ . Due to the shifting, for  $m + 1 \leq i \leq n$ , the symbol  $c_0(i)$  appears at time  $t$ ,  $1 \leq t \leq i - m$ , in the

$c_0$ :	#	1	2	3	4	5	6	7	1	2	3	4	#	$c_6$ :	#	1	2	3	4	5	6	7	1	2	3	4	#	
$c_1$ :	#	1	2	3	4	5	6	7	1	2	3	4	#	$c_7$ :	#	1	2	3	4	5	6	7	1	2	3	4	#	
$c_2$ :	#	1	2	3	4	5	6	7	1	2	3	4	#	$c_8$ :	#	1	2	3	4	5	6	7	1	2	3	4	#	
$c_3$ :	#	1	2	3	4	5	6	7	1	2	3	4	#	$c_9$ :	#	1	2	3	4	5	6	7	1	2	3	4	#	
$c_4$ :	#	1	2	3	4	5	6	7	1	2	3	4	#	$c_{10}$ :	#	1	2	3	4	5	6	7	1	2	3	4	#	
$c_5$ :	#	1	2	3	4	5	6	7	1	2	3	4	#	$c_{11}$ :	#	1	2	3	4	5	6	7	1	2	3	4	#	

Figure 11. Example computation for Lemma 5.2 ( $w = 1234$ ,  $v = 567$ ).

second register of cell  $i - t$  :  $\pi_2(c_t(i - t)) = c_0(i)$ . Its speed is reduced at time  $i - m$ . So, we have  $\pi_2(c_t(m - j)) = c_0(i)$ , for  $t = i - m + 2j$ ,  $0 \leq j \leq m - 1$ . The comparison takes place in cell  $m - j$ ,  $0 \leq j \leq m - 1$ , when the signal arrives at cell  $m - j + 1$ , i.e., at time step  $n - m + j$ . The comparison is between the content of the first register  $c_0(m - j)$  and the content of the second register. The content of the second one is determined by the time step  $t = n - m + j = i - m + 2j$ , which implies  $i = n - j$ . Therefore,  $c_0(m - j)$  is compared with  $c_0(i) = c_0(n - j)$ . Since the comparison is done for all  $m - j$  with  $0 \leq j \leq m - 1$ , an input is accepted if and only if it is of the form  $wvw$ .  $\square$

Now we are prepared to improve the known relation between  $\mathcal{L}_{rt}(\text{OCA})$  and  $\mathcal{L}_{rt}(\text{1C 1G-OCA})$ .

**Theorem 5.6.**  $\mathcal{L}_{rt}(\text{OCA}) \cdot \mathcal{L}_{rt}(\text{OCA}) \subset \mathcal{L}_{rt}(\text{1C 1G-OCA})$

**Proof:**

Due to Corollary 5.3 it suffices to prove the properness of the inclusion. To this end, we show that the language  $L = \{wvw \mid w \in A^+ \wedge v \in A^*\}$  for  $|A| \geq 2$  is not represented by the concatenation of two real-time OCA languages. Then the assertion follows by Lemma 5.2.

In contrast to the assertion, assume  $L$  is the concatenation of  $L_a$  and  $L_b$  belonging to  $\mathcal{L}_{rt}(\text{OCA})$ . Since  $L \notin \mathcal{L}_{rt}(\text{OCA})$  [28], we conclude  $L_a \neq L$  and  $L_b \neq L$ . Furthermore,  $L_a$  as well as  $L_b$  are infinite. Otherwise, if at least one of the languages is finite, it is regular. But since  $\mathcal{L}_{rt}(\text{OCA})$  is closed under left and right concatenation with regular languages [24], this implies  $L = L_a \cdot L_b \in \mathcal{L}_{rt}(\text{OCA})$ , a contradiction.

Now let  $v = v_1 \cdots v_m \in L_a$ . For all  $w = w_1 \cdots w_n \in L_b$  with  $n \geq m$  we have  $wv \in L$ . If for such a  $w$  there is a  $1 \leq i \leq m$  such that  $w_{n-i+1} \cdots w_n = v_1 \cdots v_i$ , then  $w$  contains at least one symbol  $v_1$ . If there is such a  $w$  that does not end with some prefix of  $v$ , then due to  $v_1 \cdots v_m w_1 \cdots w_n \in L$  and  $n \geq m$  there exists a  $1 \leq i \leq \lfloor \frac{m+n}{2} \rfloor - m$  such that  $v_1 \cdots v_m w_1 \cdots w_i = w_{n-(m+i)+1} \cdots w_n$ . We conclude, that also in this case  $w$  contains at least one symbol  $v_1$ .

Let  $c \neq v_1$  be a symbol in  $A$ . For  $k \geq 2$  all words  $v_1^k$  and  $c^k$  belong to  $L$ , that is, they have to be represented by concatenations. Therefore, the languages  $L_{v_1} = (L_a \cup L_b) \cap \{v_1\}^*$  and  $L_c = (L_a \cup L_b) \cap \{c\}^*$  are infinite.

If all words in  $L_c$  which are longer than  $n$  belong to  $L_a$ , then all words from  $L_{v_1}$  which are longer than  $n$  belong to  $L_a$  also. Otherwise  $c^{k_1} \cdot v_1^{k_2}$ ,  $k_1, k_2 > n$ , would not belong to  $L$ . Let  $u$  be a word in  $L_b$  with  $|u| > n$ . Due to the assumption there are words  $c^{k_3} \in L_a$  and  $v_1^{k_4} \in L_b$  with  $k_3, k_4 \geq |u|$ . Now, if  $c^{k_3}u \in L$ , then  $v_1^{k_4}u \notin L$  and vice versa. From the contradiction we derive that there is a word  $c^{k_5} \in L_b$  with  $k_5 > n$ . This word does not contain symbol  $v_1$ . Therefore,  $vc^{k_5} \notin L$  for  $v \in L_a$  and  $c^{k_5} \in L_b$ . The contradiction concludes the proof.  $\square$

This result emphasizes the computational capacity of real-time 1C 1G-OCA, since they do not characterize the family  $\mathcal{L}_{rt}(\text{OCA}) \cdot \mathcal{L}_{rt}(\text{OCA})$  but accept a strict superfamily.

## References

- [1] Book, R. V.: Tally languages and complexity classes, *Information and Control*, **26**, 1974, 186–193.
- [2] Book, R. V.: Context-sensitive tally languages, *Bulletin of the European Association for Theoretical Computer Science*, **15**, 1981, 31–34.
- [3] Buchholz, Th., Klein, A., Kutrib, M.: On interacting automata with limited nondeterminism, *Fundamenta Informaticae*, **52**, 2002, 15–38.
- [4] Buchholz, Th., Kutrib, M.: On the power of one-way bounded cellular time computers, *Developments in Language Theory* (S. Bozapalidis, Ed.), Aristotle University of Thessaloniki, Thessaloniki, Greece, 1997, 365–375.
- [5] Buchholz, Th., Kutrib, M.: Some relations between massively parallel arrays, *Parallel Computing*, **23**, 1997, 1643–1662.
- [6] Buchholz, Th., Kutrib, M.: On time computability of functions in one-way cellular automata, *Acta Informatica*, **35**, 1998, 329–352.
- [7] Choffrut, C., Čulik II, K.: On real-time cellular automata and trellis automata, *Acta Informatica*, **21**, 1984, 393–407.
- [8] Chrobak, M.: Finite automata and unary languages, *Theoretical Computer Science*, **47**, 1986, 149–158.
- [9] Cole, S. N.: Real-time computation by n-dimensional iterative arrays of finite-state machines, *IEEE Transactions on Computers*, **C-18**, 1969, 349–365.
- [10] Dyer, C. R.: One-way bounded cellular automata, *Information and Control*, **44**, 1980, 261–281.
- [11] Geidmanis, D., Kaņeps, J., Apsītis, K., Taimiņa, D., Calude, E.: Tally languages accepted by alternating multitape finite automata, *Computing and Combinatorics* (T. Jiang, D. T. Lee, Eds.), LNCS 1276, Springer-Verlag, Berlin, 1997, 422–430.
- [12] Ginsburg, S., Greibach, S.: Abstract families of languages, *Memoirs of the American Mathematical Society*, **87**, 1969, 1–32.
- [13] Ginsburg, S., Greibach, S. A., Harrison, M. A.: One-way stack automata, *Journal of the ACM*, **14**, 1967, 389–418.
- [14] Ginsburg, S., Greibach, S. A., Harrison, M. A.: Stack automata and compiling, *Journal of the ACM*, **14**, 1967, 172–201.

- [15] Ginsburg, S., Rice, H. G.: Two families of languages related to ALGOL, *Journal of the ACM*, **9**, 1962, 350–371.
- [16] Hemachandra, L. A., Rubinfeld, R. S.: Separating complexity classes with tally oracles, *Theoretical Computer Science*, **92**, 1992, 309–318.
- [17] Ibarra, O. H., Jiang, T.: Relating the power of cellular arrays to their closure properties, *Theoretical Computer Science*, **57**, 1988, 225–238.
- [18] Ibarra, O. H., Kim, S. M., Moran, S.: Sequential machine characterizations of trellis and cellular automata and applications, *SIAM Journal on Computing*, **14**, 1985, 426–447.
- [19] Ibarra, O. H., Palis, M. A.: Some results concerning linear iterative (systolic) arrays, *Journal of Parallel and Distributed Computing*, **2**, 1985, 182–218.
- [20] Kasami, T., Fuji, M.: Some results on capabilities of one-dimensional iterative logical networks, *Electronics and Communications in Japan*, **51-C**, 1968, 167–176.
- [21] Krithivasan, K., Mahajan, M.: Nondeterministic, probabilistic and alternating computations on cellular array models, *Theoretical Computer Science*, **143**, 1995, 23–49.
- [22] Kutrib, M.: Pushdown cellular automata, *Theoretical Computer Science*, **215**, 1999, 239–261.
- [23] Mazoyer, J., Terrier, V.: Signals in one-dimensional cellular automata, *Theoretical Computer Science*, **217**, 1999, 53–80.
- [24] Seidel, S. R.: *Language Recognition and the Synchronization of Cellular Automata*, Technical Report 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, *Journal of Computer and System Sciences*, **6**, 1972, 233–253.
- [26] Terrier, V.: Language recognizable in real time by cellular automata, *Complex Systems*, **8**, 1994, 325–336.
- [27] Terrier, V.: On real time one-way cellular array, *Theoretical Computer Science*, **141**, 1995, 331–335.
- [28] Terrier, V.: Language not recognizable in real time by one-way cellular automata, *Theoretical Computer Science*, **156**, 1996, 281–287.
- [29] Umeo, H., Morita, K., Sugata, K.: Deterministic one-way simulation of two-way real-time cellular automata and its related problems, *Information Processing Letters*, **14**, 1982, 158–161.