I F I G
Research
Report

Institut für Informatik

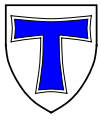
Fast One-Way Cellular Automata

Andreas Klein Martin Kutrib

IFIG Research Report 0106
September 2001

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

Justus-Liebig-



Universität
Giessen

IFIG Research Report
IFIG Research Report 0106, September 2001

Fast One-Way Cellular Automata

Andreas Klein[1]

Institut für Mathematik, Universität Kassel
Heinrich Plett Straße 40, D-34132 Kassel, Germany

Martin Kutrib[2]

Institute für Informatik, Universität Giessen
Arndtstr. 2, D-35392 Giessen, Germany

Abstract. Space-bounded one-way cellular language acceptors (OCA) are investigated. The only inclusion known to be strict in their time hierarchy from real-time to exponential-time is between real-time and linear-time! We show the surprising result that there exists an infinite hierarchy of properly included OCA-language families in that range. A generalization of a method of Terrier is shown which provides a tool for proving that languages are not acceptable by OCAs with small time bounds. The hierarchies are established by such a language and a translation result. In addition, a notion of constructibility for CAs is introduced, along with some of its properties. We prove several closure properties of the families in the hierarchy.

CR Subject Classification (1998): F.1, F.4.3, B.6.1, E.4

[1]E-mail: klein@mathematik.uni-kassel.de
[2]E-mail: kutrib@informatik.uni-giessen.de

1 Introduction

Linear arrays of interacting finite automata are models for massively parallel language acceptors. Their advantages are simplicity and uniformity. It has turned out that a large array of not very powerful processing elements operating in parallel can be programmed to be very powerful.

One type of system is of particular interest: the cellular automata whose homogeneously interconnected deterministic finite automata (the cells) work synchronously at discrete time steps obeying one common transition function. Here we are interested in a very simple type of cellular automata. The arrays are real-space bounded, i.e., the number of cells is bounded by the number of input symbols, and each cell is connected to its immediate neighbor to the right only. Due to the resulting information flow from right to left such devices are called one-way cellular automata (OCA). If the cells are connected to their both immediate neighbors the information flow becomes two-way and the device is a (two-way) cellular automaton (CA).

Although parallel language recognition by (O)CAs has been studied for more than a quarter of a century some important questions are still open. In particular, only little is known about proper inclusions in the time hierarchy. Most of the early languages known not to be real-time but linear-time OCA-languages are due to the fact that every unary real-time OCA-language is regular [3]. In [10] and [9] a method has been shown that allows proofs of non-acceptance for non-unary languages in real-time OCAs. Utilizing these ideas the non-closure of real-time OCA-languages under concatenation could be shown.

Since for separating the complexity classes in question there are no other general algebraic methods available, specific languages as potential candidates are of particular interest. In [4] several positive results have been presented. Surprisingly, so far there was only one inclusion in the time hierarchy from real-time to exponential-time known to be strict. It is the inclusion between real-time and linear-time languages. In [1] the existence of a non-real-time OCA-language which is acceptable in $(n+\log(n))$-time has been proved yielding a lower upper bound for the strict inclusion. Another valuable tool for exploring the OCA time hierarchy is the possible linear speed-up [6] from $n + r(n)$ to $n + \varepsilon \cdot r(n)$ for $\varepsilon > 0$.

One contribution of the present paper is to show that there exists an infinite time hierarchy of properly included language families. These families are located in the range between real-time and linear-time. The surprising result covers the lower part of the time hierarchy in detail. Another contribution is the investigation of several closure properties of the language families in the hierarchy.

The paper is organized as follows: In Section 2 we define the basic notions and the model in question. Since for almost all infinite hierarchies in complexity theory the constructibility of the bounding functions is indispensable, in Section 3 we present a new notion of constructibility in OCAs and prove that it covers a wider range of functions than the usual approach. Section 4 is devoted

to a generalization of the method in [9] to time complexities beyond real-time. This key tool is utilized to obtain a certain language not acceptable with a given time bound. In Section 5 the corresponding proper inclusion is extended to an infinite time hierarchy by translation arguments. Finally, in Section 6 some closure properties are investigated.

2 Basic notions

We denote the positive integers $\{1, 2, ...\}$ by $\mathbb{N}$ and the set $\mathbb{N} \cup \{0\}$ by $\mathbb{N}_0$. The empty word is denoted by λ and the reversal of a word w by w^R. For the length of w we write $|w|$. We use $\subseteq$ for inclusions and $\subset$ if the inclusion is strict. For a function $f : \mathbb{N}_0 \to \mathbb{N}$ we denote its i-fold composition by $f^{[i]}$, $i \in \mathbb{N}$. If f is increasing then its inverse is defined according to

$$f^{-1}(n) = \min\{m \in \mathbb{N} \mid f(m) \geq n\}.$$

As usual, we define the set of functions that grow strictly less than f by $o(f) = \{g : \mathbb{N}_0 \to \mathbb{N} \mid \lim_{n\to\infty} \frac{g(n)}{f(n)} = 0\}$. In terms of orders of magnitude f is an upper bound of the set $O(f) = \{g : \mathbb{N}_0 \to \mathbb{N} \mid \exists\, n_0, c \in \mathbb{N} : \forall\, n \geq n_0 : g(n) \leq c \cdot f(n)\}$. Conversely, f is a lower bound of the set $\Omega(f) = \{g : \mathbb{N}_0 \to \mathbb{N} \mid f \in O(g)\}$.

A one-way resp. two-way cellular automaton is a linear array of identical deterministic finite state machines, sometimes called cells, which are connected to their nearest neighbor to the right resp. to their both nearest neighbors. The array is bounded by cells in a distinguished so-called boundary state. For convenience we identify the cells by positive integers. The state transition depends on the current state of each cell and the current state(s) of its neighbor(s). The transition function is applied to all cells synchronously at discrete time steps. Formally:

Definition 1 *A* one-way cellular automaton (OCA) *is a system* $\langle S, \delta, \#, A, F\rangle$ *where*

1. S *is the finite, nonempty set of* cell states,
2. $\# \notin S$ *is the* boundary state,
3. $A \subseteq S$ *is the nonempty set of* input symbols,
4. $F \subseteq S$ *is the set of* accepting (or final) states, *and*
5. $\delta : (S \cup \{\#\})^2 \to S$ *is the* local transition function.

Figure 1: A one-way cellular automaton.

If the flow of information is extended to two-way the resulting device is a *(two-way) cellular automaton* (CA) and the local transition function maps from $(S \cup \{\#\})^3$ to S.

A *configuration* of a cellular automaton at some time $t \geq 0$ is a description of its global state, which is actually a mapping $c_t : \{1, \ldots, n\} \to S$ for $n \in \mathbb{N}$.

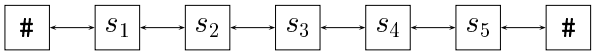


Figure 2: A (two-way) cellular automaton.

The configuration at time 0 is defined by the initial sequence of states. For a given input $w = a_1 \cdots a_n \in A^+$ we set $c_{0,w}(i) = a_i$ for $1 \leq i \leq n$. During a computation the (O)CA steps through a sequence of configurations whereby successor configurations are computed according to the global transition function Δ:

Let c_t for $t \geq 0$ be a configuration, then its successor configuration is as follows:

$$\begin{aligned} c_{t+1} = \Delta(c_t) \iff & \\ & c_{t+1}(1) = \delta\big(\#, c_t(1), c_t(2)\big) \\ & c_{t+1}(i) = \delta\big(c_t(i-1), c_t(i), c_t(i+1)\big), i \in \{2, \ldots, n-1\} \\ & c_{t+1}(n) = \delta\big(c_t(n-1), c_t(n), \#\big) \end{aligned}$$

for CAs and correspondingly for OCAs. Thus, Δ is induced by δ.

If the state set is a Cartesian product of some smaller sets $S = S_0 \times S_1 \times \cdots \times S_r$, we will use the notion *register* for the single parts of a state.

An input w is accepted by an (O)CA if at some time i during its course of computation the leftmost cell enters an accepting state.

Definition 2 *Let $\mathcal{M} = \langle S, \delta, \#, A, F\rangle$ be an* (O)CA.

1. *An input $w \in A^+$ is* accepted *by $\mathcal{M}$ iff there exists a time step $i \in \mathbb{N}$ such that $c_i(1) \in F$ holds for the configuration $c_i = \Delta^{[i]}(c_{0,w})$.*
2. *$L(\mathcal{M}) = \{w \in A^+ \mid w$ is accepted by $\mathcal{M}\}$ is the* language accepted by $\mathcal{M}$.
3. *Let $t : \mathbb{N} \to \mathbb{N}$, $t(n) \geq n$, be a mapping. If all $w \in L(\mathcal{M})$ can be accepted with at most $t(|w|)$ time steps, then L is said to be of time complexity t.*

The family of all languages that are acceptable by some OCA (CA) with time complexity t is denoted by $\mathscr{L}_t(\text{OCA})$ ($\mathscr{L}_t(\text{CA})$). If t equals the identity function $id(n) = n$, acceptance is said to be in *real-time*, and if t is equal to $k \cdot id$ for an arbitrary rational number $k \geq 1$, then acceptance is carried out in *linear-time*. Correspondingly, we write $\mathscr{L}_{rt}((\text{O})\text{CA})$ and $\mathscr{L}_{lt}((\text{O})\text{CA})$.

In the following we are going to prove our main result:

Theorem 3 *Let $r_1, r_2 : \mathbb{N} \to \mathbb{N}$ be two increasing functions.*
If $r_2 \log(r_2) \in o(r_1)$ and r_1^{-1} is constructible, then

$$\mathscr{L}_{n+r_2(n)}(\text{OCA}) \subset \mathscr{L}_{n+r_1(n)}(\text{OCA})$$

Example 4 Let $0 \leq p < q \leq 1$ be two rational numbers. Clearly, $n^p \cdot \log(n^p)$ is of order $o(n^q)$. In the next section the constructibility of the inverse of n^q will be established. Thus, an application of Theorem 3 yields the strict inclusion

$$\mathscr{L}_{n+n^p}(\text{OCA}) \subset \mathscr{L}_{n+n^q}(\text{OCA})$$

Example 5 Let $i < j$ be two positive integers, then $\log^{[j]} \cdot \log^{[j+1]}$ is of order $o(\log^{[i]})$. Again, in the next section the constructibility of the inverse of $\log^{[i]}$ will be established. Thus, an application of Theorem 3 yields the strict inclusion

$$\mathscr{L}_{n+\log^{[j]}}(\text{OCA}) \subset \mathscr{L}_{n+\log^{[i]}}(\text{OCA})$$

3 Constructible Functions

For the proof of Theorem 3 it will be necessary to control the lengths of words with respect to some internal substructures. The following notion of constructibility expresses the idea that the length of a word relative to the length of a subword should be computable.

Definition 6 *A function $f : \mathbb{N} \to \mathbb{N}$ is* constructible *if there exists an λ-free homomorphism h and a language $L \in \mathscr{L}_{rt}(\text{OCA})$ such that*

$$h(L) = \{a^{f(n)-n}b^n \mid n \in \mathbb{N}\}$$

Since constructible functions describe the length of the whole word dependent on the length of a subword it is obvious that each constructible function must be greater than or equal to the identity. At a first glance this notion of constructibility might look somehow unusual or restrictive. But λ-free homomorphisms are very powerful such that the family of (in this sense) constructible functions is very rich, and is, in fact, a generalization of the usual notion. The remainder of this section is devoted to clarify the presented notion and its power.

The next lemma states that we can restrict our considerations to length preserving homomorphisms. The advantage is that for length preserving homomorphisms each word in L is known to be of length $f(m)$ for some $m \in \mathbb{N}$.

Lemma 7 *Let $f : \mathbb{N} \to \mathbb{N}$ be a constructible function. Then there exists a length preserving λ-free homomorphism h and a language $L \in \mathscr{L}_{rt}(\text{OCA})$ such that*

$$h(L) = \{a^{f(n)-n}b^n \mid n \in \mathbb{N}\}$$

Proof. Since f is constructible there exists an λ-free homomorphism h' and a language $L' \in \mathscr{L}_{rt}(\text{OCA})$ such that $h'(L') = \{a^{f(n)-n}b^n \mid n \in \mathbb{N}\}$. In order to prove the assertion it suffices to construct a language $L \in \mathscr{L}_{rt}(\text{OCA})$ and a length-preserving λ-free homomorphism h with $h(L) = h'(L')$. Let L' be defined over an alphabet $A = \{a_1, \ldots, a_m\}$ and h' according to

$$h'(a_1) = b_{1,1} \cdots b_{1,n_1}, \quad h'(a_2) = b_{2,1} \cdots b_{2,n_2}, \quad \ldots \quad , h'(a_m) = b_{m,1} \cdots b_{m,n_m}$$

where the symbols $b_{i,j}$ are not necessarily different. We introduce an alphabet $B = \{\bar{b}_{1,1}, \ldots, \bar{b}_{1,n_1}, \ldots, \bar{b}_{m,n_m}\}$ of different symbols and define the length preserving λ-free homomorphism h by

$$h(\bar{b}_{i,j}) = b_{i,j}$$

for $1 \leq i \leq m$ and $1 \leq j \leq n_i$.

In order to construct L we define a homomorphism $\hat{h}$ by

$$\hat{h}(\bar{b}_{i,1}) = a_i \quad \text{and} \quad \hat{h}(\bar{b}_{i,j}) = \lambda$$

for $1 \leq i \leq m$ and $2 \leq j \leq n_i$, and set

$$L = \hat{h}^{-1}(L') \cap \{\bar{b}_{i,1} \cdots \bar{b}_{i,n_i} \mid 1 \leq i \leq m\}^*$$

By construction $h'(L') = h(L)$ follows. Since $\mathscr{L}_{rt}(\text{OCA})$ is closed under inverse homomorphisms and intersection with regular sets L belongs to $\mathscr{L}_{rt}(\text{OCA})$.

□

Given an increasing constructible function $f : \mathbb{N} \to \mathbb{N}$ and a language $L_a \subseteq A^+$ acceptable by some OCA with time complexity $n + r(n)$, where $r : \mathbb{N} \to \mathbb{N}$, we now define a language that plays an important role in the sequel. Let the language $L_f \subseteq B^+$ be a witness for the constructibility of f, i.e., $L_f \in \mathscr{L}_{rt}(\text{OCA})$ and $h(L_f) = \{a^{f(n)-n}b^n \mid n \in \mathbb{N}\}$ for a length preserving λ-free homomorphism h. The language

$$L_1(L_a, L_f) \subseteq ((A \cup \{\sqcup\}) \times B)^+$$

is constructed as follows:

1. The second component of each word w in $L_1(L_a, L_f)$ is a word of L_f that implies that w is of length $f(m)$ for some $m \in \mathbb{N}$.
2. The first component of w contains exactly $f(m) - m$ blank symbols and m non-blank symbols.
3. The non-blank symbols in the first component of w form a word in L_a.

The following proposition is used in later sections. Besides, it is an example that demonstrates how to use constructible functions. In Lemma 16 we will prove that the shown bound for the time complexity of L_1 is minimal.

Proposition 8 *The language $L_1(L_a, L_f)$ is acceptable by some* OCA *with time complexity $n + r(f^{-1}(n))$.*

Proof. We construct an OCA $\mathcal{A}$ with three registers that accepts L_1 obeying the time complexity $n + r(f^{-1}(n))$.

In its first register $\mathcal{A}$ verifies that the second component of each word in L_1 is a word of L_f. By definition of L_f this can be done in real-time.

In its second register $\mathcal{A}$ checks that the first component of L_1 contains exactly $f(m) - m$ blank symbols. Because it can be verified that the second component of L_1 belongs to L_f, we know that the first $f(m) - m$ symbols of the second component are mapped to a's and the last m symbols of the second component are mapped to b's. The task is to check that the number of a's in the second component is equal to the number of blank symbols in the first component. Therefore, $\mathcal{A}$ shifts the blank symbols from right to left. Each symbol a in the second component consumes one blank symbol. A signal that goes from the right to the left with full speed can check that no blank symbol has reached the

leftmost cell and that each letter a has consumed one blank symbol, i.e., that the number of a's is equal to the number of blank symbols. The test can be done in real-time.

In order to verify that the non-blank symbols in the first component form a word of L_a the automaton $\mathcal{A}$ simulates the OCA that accepts L_a. But for every blank symbol $\mathcal{A}$ needs one time step in addition as illustrated below. Therefore, $\mathcal{A}$ needs $m + r(m) + (f(m) - m)$ steps for the simulation ($m + r(m)$ time steps for the simulation itself and $f(m) - m$ time steps delaying time). Substituting $m = f^{-1}(n)$ completes the proof. □

The basic idea for accepting words with OCAs that have blank cells is as follows (cf. Figure 3). Initially, the blank cells are marked as transportation cells. At every time step they simply store the state of their right neighbors in some register. All non-blank cells can be blocked or not. A cell gets blocked if its neighbor is blocked or if its neighbor transports a blocked or blank symbol. A blocked cell gets released if its neighbor is released or transports a released symbol. Blocked cells keep their states and released cells work as usual.

It is easily seen that the total delay of a computation equals the number of blank cells.

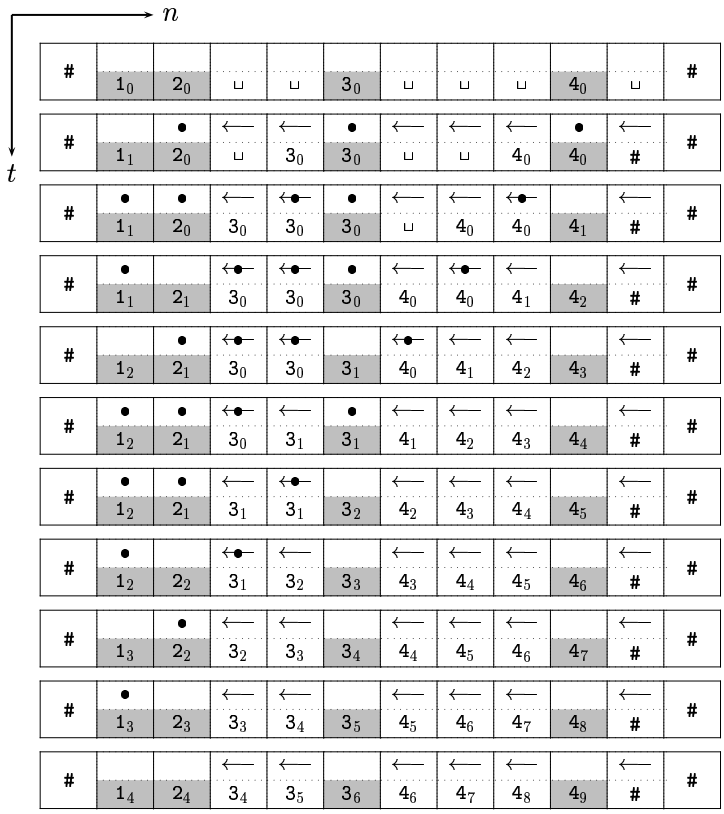


Figure 3: Transportation cells are marked by ←. The bullet indicates a blocked cell.

Now we prove that the family of constructible functions is very rich. In particular, all Fischer-constructible functions are constructible in the sense of Definition 6. A function f is said to be *Fischer-constructible* if there exists an unbounded two-way CA such that the initially leftmost cell enters a final state at time $i \in \mathbb{N}$ if and only if $i = f(m)$ for some $m \in \mathbb{N}$. Moreover, the CA starts with a configuration in which all cells except the leftmost one are quiescent. Thus, the Fischer-constructibility is an important notion that meets the intuition of constructible functions. For a detailed study of these functions see [7] where also the name has been introduced according to the author of [5].

For example, n^k for $k \in \mathbb{N}$, 2^n, $n!$, and p_n, where p_n is the nth prime number, are Fischer-constructible. Moreover, the class is closed under several operations.

Lemma 9 *If a function $f : \mathbb{N} \to \mathbb{N}$ is Fischer-constructible, then it is constructible in the sense of Definition 6.*

Proof. Let f be a Fischer-constructible function. In a first step we prove that $\{b^n a^{f(n)-n} \mid n \in \mathbb{N}\}$ is a real-time CA-language.

The leftmost cell of the CA starts the construction of f, i.e., it distinguishes the time steps $f(1), f(2), \ldots$ The rightmost cell send a signal to the left. The CA accepts a word of the form b^*a^* if and only if this signal reaches the leftmost cell at time step $f(n)$ for some $n \in \mathbb{N}$ and the number of b's is equal to n.

Now a language $L \subseteq (\{a, b\} \times \{a, b, \sqcup\}^2)^*$ is constructed as follows:

1. The first component of each word $w \in L$ belongs to $\{a^{f(n)-n}b^n \mid n \in \mathbb{N}\}$.
2. The second component is the first component compressed by a factor 2 followed by (pairs of) blank symbols.

L is a real-time OCA-language. A corresponding OCA can easily verify that the first component is of the form a^*b^* and that the second component contains the compressed first component. In addition, it simulates the real-time CA for $\{b^n a^{f(n)-n} \mid n \in \mathbb{N}\}$ on the left part of its second track. This is possible since $\mathscr{L}_{rt}(\mathrm{CA})^R = \mathscr{L}_{2\cdot n}(\mathrm{OCA})$. Due to the compression the OCA works in real-time.

Together with the λ-free homomorphism that maps a word to its first component this proves that f is constructible in the sense of Definition 6. □

Theorem 10 *The class of constructible functions is closed under addition, multiplication and composition.*

Proof. Let f_1 and f_2 be two constructible functions. Further let $L_1, L_2 \in \mathscr{L}_{rt}(\mathrm{OCA})$ and h_1, h_2 be two length preserving homomorphisms with $h_i(L_i) = \{a^{f_i(n)-n}b^n \mid n \in \mathbb{N}\}$, $1 \leq i \leq 2$. Without loss of generality we assume that L_1 and L_2 are defined over disjoint alphabets.

In order to prove that $f_1(n) + f_2(n)$ is constructible a real-time OCA-language is constructed by the following description of an acceptor. It performs the following tasks in parallel:

1. It checks that the input is of the form w_1w_2 with $w_i \in L_i$, $1 \leq i \leq 2$. This is possible because L_1 and L_2 are defined over disjoint alphabets.

2. It computes $h_1(w_1) = a^{f_1(n)-n}b^n$ and $h_2(w_2) = a^{f_2(m)-m}b^m$ and verifies $n = m$. A OCA can compare two numbers because $\{a^n b^n \mid n \in \mathbb{N}\} \in \mathscr{L}_{rt}(\text{OCA})$.

Together with the homomorphism that maps w_1 to $a^{f_1(n)}$ and w_2 to $a^{f_2(n)-n}b^n$ this proves that $f_1(n)+f_2(n)$ is constructible. (The homomorphism utilizes the fact that L_1 and L_2 are defined over disjoint alphabets.)

In order to prove that $f_1(n)f_2(n)$ is constructible, again, a real-time OCA-language is described by an accepting OCA. It works on an input tape with two registers and verifies that:

1. The first track has the form $a_1 \sqcup^n a_2 \sqcup^n \cdots a_m \sqcup^n$, where $\sqcup$ is a special symbol not belonging to the alphabets of L_1 and L_2. (Once again the OCA has to compare numbers.)
2. $a_1 a_2 \cdots a_m$ is a word in L_1. (The OCA can ignore the blank characters.)
3. The second track has the form $\sqcup \cdots \sqcup w_2$ with $w_2 \in L_2$.
4. The non-blank part of the second track has length $n+1$, i.e., the first component of the non-black part is $a_m \sqcup^n$.
5. $n_1 = n_2$ after computing $h_1(a_1 a_2 \cdots a_m) = a^{f_1(n_1)-n_1}b^{n_1}$ and $h_2(w_2) = a^{f_2(n_2)-n_2}b^{n_2}$.

Together with the homomorphism that applies h_2 to the second track and maps all blank symbols on the second track to a this proves that $f_1(n)f_2(n)$ is constructible.

In order to prove that $f_1(f_2(n))$ is constructible the accepting real-time OCA verifies that:

1. The first component of its input is a word $w_1 \in L_1$.
2. The second component has the form $\sqcup \cdots \sqcup w_2$ with $w_2 \in L_2$.
3. $|w_2| = n$ after computing $h_1(w_1) = a^{f_1(n)-n}b^n$.

By the same homomorphism as in the previous part we see that $f_1(f_2(n))$ is constructible. □

4 Equivalence Classes

To prove lower bounds for the time complexity we generalize a lemma shown in [9] which gives a necessary condition for a language to be real-time acceptable by a OCA. At first we need the following definition:

Definition 11 *Let L be a language and X and Y be two sets of words. Two words w and w' are equivalent with respect to L, X and Y (in short (L, X, Y)-equivalent) if and only if*

$$xwy \in L \iff xw'y \in L$$

for all $x \in X$ and $y \in Y$.

Let $L_d \subset \{0,1,(,),|\}^+$ be a language whose words are of the form

$$x\,(\,x_1\,|\,y_1\,)\cdots(\,x_n\,|\,y_n\,)\,y$$

where $x, x_i, y, y_i \in \{0,1\}^*$ for $1 \leq i \leq n$, and $(\,x\,|\,y\,) = (\,x_i\,|\,y_i\,)$ for at least one $i \in \{1,\ldots,n\}$.

The language L_d can be thought of as a dictionary. The task for the OCA is to check whether the pair $(\,x\,|\,y\,)$ appears in the dictionary or not.

Proposition 12 *Let $X = Y = \{0,1\}^*$. Two words $w = (\,x_1\,|\,y_1\,)\cdots(\,x_n\,|\,y_n\,)$ and $w' = (\,x'_1\,|\,y'_1\,)\cdots(\,x'_m\,|\,y'_m\,)$ are equivalent with respect to L_d, X and Y if and only if $\{(\,x_1\,|\,y_1\,),\ldots,(\,x_n\,|\,y_n\,)\} = \{(\,x'_1\,|\,y'_1\,),\ldots,(\,x'_m\,|\,y'_m\,)\}$.*

Proof. First assume that the two sets are equal. Let $x \in X$ and $y \in Y$, then $xwy \in L_d$ implies $(\,x\,|\,y\,) = (\,x_i\,|\,y_i\,)$ for some i. Since the two sets are equal we have $(\,x\,|\,y\,) = (\,x'_j\,|\,y'_j\,)$ for some j. Therefore, $xwy \in L_d$ implies $xw'y \in L_d$ and vice versa, i.e., w and w' are (L_d, X, Y)-equivalent.

Now assume the two sets are not equal. Without loss of generality we can assume that there exist $x \in X$ and $y \in Y$ with $(\,x\,|\,y\,) = (\,x_i\,|\,y_i\,)$ for some i, but $(\,x\,|\,y\,) \neq (\,x'_j\,|\,y'_j\,)$ for all $j = 1,\ldots,m$. Then $xwy \in L_d$ but $xw'y \notin L_d$ and, thus, w and w' are not (L_d, X, Y)-equivalent. □

Now we are prepared to formulate the lemma we are going to use in order to prove lower bounds for the time complexities.

Lemma 13 *Let $r : \mathbb{N} \to \mathbb{N}$ be an increasing function, $L \in \mathscr{L}_{n+r(n)}(\mathrm{OCA})$ and $X = \{a_1,\ldots,a_p\}^{m_1}$ and $Y = \{b_1,\ldots,b_q\}^{m_2}$ be two sets of words for positive integers p, q, m_1 and m_2. There exists an integer $s \in \mathbb{N}$ such that the number N of (L, X, Y)-equivalence classes of the words at most of length $n - m_1 - m_2$ is bounded by*

$$N \leq s^{|X|}s^{(m_2+r(n))|Y|}$$

Proof. Let $\bar{s}$ be the minimal number of states needed by a OCA to accept L in $n + r(n)$ time steps. Let $\mathcal{A}$ be such a OCA with state set S.

We consider the computation of $\mathcal{A}$ on the word xwy for some $x \in X$ and $y \in Y$. After $|w|$ time steps the interesting part of the configuration of $\mathcal{A}$ can be described by $f_w(x)f'_w(y)$ where (cf. Figure 4)

1. $f_w(x) \in S^*$ and $f'_w(y) \in S^*$.
2. $|f_w(x)| = |x|$ and $|f'_w(y)| = y + r(n)$. During the remaining $|xwy| + r(|xwy|) - |w| \leq |x| + |y| + r(n)$ time steps the result of the computation of $\mathcal{A}$ depends only on the states of the $|x| + |y| + r(n)$ leftmost cells.
3. $f'_w(y)$ depends only on w and y since no information can move from left to right.
4. $f_w(x)$ depends only on w and x since during $|w|$ time steps only the leftmost $|x| + |w|$ cells can influence the states of the leftmost $|x|$ cells.

If $f_w(x) = f_{w'}(x)$ and $f'_w(y) = f'_{w'}(y)$ for all $x \in X$ and $y \in Y$, then w and w' are equivalent with respect to L, X and Y. Thus, if w and w' are not equivalent, then $f_w \neq f_{w'}$ or $f'_w \neq f'_{w'}$.

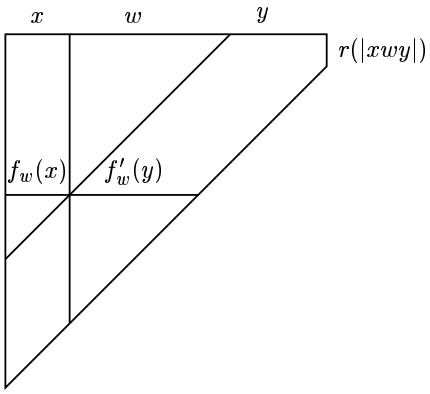


Figure 4: OCA computation in the proof of Lemma 13.

Now we count the number of functions f_w and f'_w. Since f'_w maps Y into the set $S^{m_2+r(n)}$ which contains $\bar{s}^{m_2+r(n)}$ elements, the number of different functions f'_w is bounded by $(\bar{s}^{m_2+r(n)})^{|Y|}$.

We can utilize the nature of OCAs to give a more precise upper bound for the number of different functions f_w that map X into the set S^{m_1}. If a word $x = x_{m_1} \cdots x_1$ is mapped to $s_{m_1} \cdots s_1$ then due to the one-way information flow s_i depends on $x_i \cdots x_1$ only. Thus, for s_i there are at most $\bar{s}^{p^i}$ different (sub-)functions. It follows that the number of different functions f_w is bounded by

$$\prod_{i=1}^{m_1} \bar{s}^{p^i} < \bar{s}^{p^{m_1+1}}$$

For $s = \bar{s}^p$ we obtain $\bar{s}^{p^{m_1+1}} = s^{p^{m_1}} = s^{|X|}$.

Since $s \geq \bar{s}$ and each upper bound on the number of pairs (f_w, f'_w) is also an upper bound on the number of (L, X, Y)-equivalence classes the lemma follows. □

In the previous lemma the sets of words X and Y are in some sense complete with respect to the underlying alphabets. In the general case where X and Y are arbitrary sets of words we obtain a slightly weaker bound:

Lemma 14 *Let $r : \mathbb{N} \to \mathbb{N}$ be an increasing function, $L \in \mathscr{L}_{n+r(n)}$(OCA) and X and Y be two sets of words. Let s be the minimal number of states needed by an* OCA *to accept L in $n + r(n)$ time steps.*

If all words in X are of length m_1 and all words in Y are of length m_2, then the number N of (L, X, Y)-equivalence classes of the words at most of length $n - m_1 - m_2$ is bounded by

$$N \leq s^{m_1|X|} s^{(m_2+r(n))|Y|}$$

For the special case $L \in \mathscr{L}_{rt}$(OCA) the lemma has been shown in [9].

Now we apply the lemma to the language L_d.

Proposition 15 *Let $r : \mathbb{N} \to \mathbb{N}$ be an increasing function. If $r(n)\log(r(n)) \in o(n)$, then L_d is not acceptable by any* OCA *with time complexity $n + r(n)$ but L_d belongs to $\mathscr{L}_{lt}$*(OCA).

Proof. For fixed $m_1 \in \mathbb{N}$ and $m_2 \in \mathbb{N}$ we investigate all words of the form $(\,x_1 \,|\, y_1\,) \cdots (\,x_k \,|\, y_k\,)$ with $x_i \in \{0,1\}^{m_1}$ and $y_i \in \{0,1\}^{m_2}$ for all $i \in \{1,\ldots,k\}$ and $(\,x_i \,|\, y_i\,) \neq (\,x_j \,|\, y_j\,)$ for $i \neq j$. We call this words of type (m_1, m_2).

Since there are at most $2^{m_1+m_2}$ different pairs the length of words of type (m_1, m_2) is at most $2^{m_1+m_2} \cdot (m_1 + m_2 + 3)$.

As it has been shown in Proposition 12 two words are equivalent iff the sets of subwords are equal. Thus, there are $2^{2^{m_1+m_2}}$ words of type (m_1, m_2) which belong to different equivalence classes with respect to L_d, $X = \{0,1\}^{m_1}$ and $Y = \{0,1\}^{m_2}$. (For each subset of $X \times Y$ there exists one equivalence class.)

Assume L_d belongs to $\mathscr{L}_{n+r(n)}$(OCA), then an accepting OCA must be able to distinguish all these equivalence classes. By Lemma 13 there must exist a number s such that

$$2^{2^{m_1+m_2}} \leq s^{2^{m_1}} s^{(m_2+r(n))2^{m_2}}$$

for $n = 2^{m_1+m_2} \cdot (m_1 + m_2 + 3) + m_1 + m_2$.

In order to obtain a contradiction we proceed as follows:

Approximating the order of n we obtain

$$n \in O(2^{m_1+m_2} \cdot (m_1 + m_2))$$

Since $r(n)\log(r(n)) \in o(n)$ it holds

$$r(2^{m_1+m_2}(m_1 + m_2))\log(r(2^{m_1+m_2}(m_1 + m_2))) \in o(2^{m_1+m_2}(m_1 + m_2))$$

Observe, $2^{m_1+m_2}\log(2^{m_1+m_2}) = 2^{m_1+m_2}(m_1 + m_2)$. Therefore, $r(2^{m_1+m_2}(m_1 + m_2))$ and, hence, $r(n)$ must be of order $o(2^{m_1+m_2})$. It follows:

$$\forall\, \varepsilon' \,\exists\, M \,\forall\, n > M :\ r(n) < \varepsilon' 2^{m_1+m_2}$$

In particular

$$\forall\, m_2 \,\forall\, \varepsilon' \,\exists\, M \,\forall\, m_1 > M :\ r(n) < \varepsilon' 2^{m_1+m_2}$$

Choosing $\varepsilon' = \frac{\varepsilon}{2^{m_2}}$ yields

$$\forall\, m_2 \,\forall\, \varepsilon \,\exists\, M \,\forall\, m_1 > M :\ r(n) < \varepsilon 2^{m_1}$$

On the other hand,

$$\forall\, m_2 \,\forall\, \varepsilon \,\exists\, M \,\forall\, m_1 > M :\ m_2 < \varepsilon 2^{m_1}$$

Together we obtain $\forall\, m_2 \,\forall\, \varepsilon \,\exists\, M \,\forall\, m_1 > M$:

$$s^{2^{m_1}} s^{(m_2+r(n))2^{m_2}} < s^{2^{m_1}} s^{(m_2+\varepsilon 2^{m_1})2^{m_2}} < s^{2^{m_1}} s^{2\varepsilon 2^{m_1} 2^{m_2}}$$

$$= s^{2^{m_1}} s^{2\varepsilon 2^{m_1+m_2}} = s^{2^{m_1+m_2}\cdot(2^{-m_2}+2\varepsilon)} = 2^{2^{m_1+m_2}\cdot\log(s)\cdot(2^{-m_2}+2\varepsilon)}$$

If we choose m_2 such that $2^{-m_2} < \frac{1}{2\log(s)}$ and ε such that $2\varepsilon < \frac{1}{2\log(s)}$, then there exists M such that for all $m_1 > M$:

$$s^{2^{m_1}} s^{(m_2+r(n))2^{m_2}} < 2^{2^{m_1+m_2}\cdot\log(s)\cdot(2^{-m_2}+2\varepsilon)} < 2^{2^{m_1+m_2}}$$

This is a contradiction, thus, L is not acceptable by an OCA in $(n+r(n))$-time.

To see that L is acceptable in linear-time, we construct an appropriate OCA. Starting with an input word of the form $x\,(\,x_1\,|\,y_1\,)\cdots(\,x_m\,|\,y_m\,)\,y$ the OCA shifts the subword y with full speed to the left. During the first n time steps the OCA marks all pairs $(\,x_i\,|\,y_i\,)$ with $y_i = y$. Each marked pair starts moving to the left with half speed. Each time a pair $(\,x_i\,|\,y\,)$ reaches the left hand side the OCA checks whether $x_i = x$. The pairs of the form $(\,x_i\,|\,y\,)$ reach the leftmost cell sequentially because y moves with full speed but the pairs of form $(\,x_i\,|\,y\,)$ with half speed only. This guarantees that the OCA has sufficient time to check whether $x = x_i$. Figure 5 illustrates the computation. The basic task for the OCA is to check whether $y = y_i$. This is equivalent to the acceptance of the real-time OCA-language $\{w \bullet w \mid w \in \{0,1\}^+\}$. □

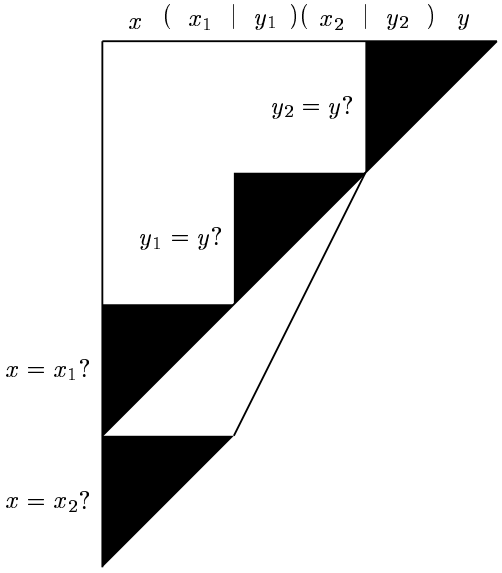


Figure 5: Linear-time acceptance of L_d. The black triangles mark the areas where a check of the form $y_i = y$ takes place. It is easy to see that the black triangels are disjoint, i.e., the checks can be done one after the other with a finite number of states.

5 Time Hierarchies

The section is devoted to the proof of the result stated in Theorem 3. The next step towards the proof is a translation lemma which allows to extend a single proper inclusion to a time hierarchy.

Lemma 16 *Let $t_1, t_2 : \mathbb{N} \to \mathbb{N}$ be two functions and L_a be a $(n + t_1(n))$-time* OCA*-language that is not acceptable by any* OCA *within $n + o(t_2(n))$ time since its equivalence classes are not bounded according to Lemma 13. Further let $f : \mathbb{N} \to \mathbb{N}$ be a constructible function and $r_1, r_2 : \mathbb{N} \to \mathbb{N}$ be two functions such that $r_1(f(n)) \in \Omega(t_1(n))$ and $r_2(f(n)) \in o(t_2(n))$. Then*

$$\mathscr{L}_{n+r_2(n)}(\text{OCA}) \subset \mathscr{L}_{n+r_1(n)}(\text{OCA})$$

Proof. For $f(n) \in O(n)$ we have $r_2(O(n)) \in o(t_2(n))$ what implies $r_2(n) \in o(t_2(n))$ and, thus, $L_a \notin \mathscr{L}_{n+r_2(n)}(\text{OCA})$. Conversely, $r_1(O(n)) \in \Omega(t_1(n))$ and, therefore, $r_1(n) \in \Omega(t_1(n))$. It follows $L_a \in \mathscr{L}_{n+r_1(n)}(\text{OCA})$ and, hence, the assertion.

In order to prove the lemma for $n \in o(f(n))$ let L_f be a language that proves the constructibility of f in Lemma 7. At first we show that we can always find such an L_f whose words are of the form $a^n w b^n$ with $|w| = f(n) - 2n$.

By $w/3$ we denote the word w compressed by the factor 3, i.e., one symbol of $w/3$ is interpreted as three symbols of w.

Now define $\bar{L}_f$ such that $a^{f(n)-n-\frac{1}{3}f(n)}(w/3)b^n \in \bar{L}_f$ iff $w \in L_f$. Clearly, the words of $\bar{L}_f$ are of the desired form since $n \in o(f(n))$. Moreover, there exists a trivial λ-free, length preserving homomorphism that maps $\bar{L}_f$ to $\{a^{f(n)-n}b^n \mid n \in \mathbb{N}\}$. Also, $\bar{L}_f$ belongs to $\mathscr{L}_{rt}(\text{OCA})$ since an OCA can verify in real-time that an input w

1. belongs to L_f,
2. the length of the word $a^{f(n)-n-\frac{1}{3}f(n)}(w/3)b^n \in \bar{L}_f$ is equal to the length of w, and
3. that n is equal to the number of b's in $h(w)$ where h denotes the homomorphism that maps L_f to $\{a^{f(n)-n}b^n \mid n \in \mathbb{N}\}$.

Thus, from now on we may assume w.l.o.g. that the words of L_f are of the form $a^n w b^n$ with $|w| = f(n) - 2n$. From Proposition 8 follows that the language $L_1 = L_1(L_a, L_f)$ belongs to $\mathscr{L}_{n+t_1(f^{-1}(n))}(\text{OCA})$. By the assumption on $r_1(f(n))$ we obtain $r_1(n) = r_1(f(f^{-1}(n))) \in \Omega(t_1(f^{-1}(n)))$ and, therefore, $L_1 \in \mathscr{L}_{n+r_1(n)}(\text{OCA})$.

It remains to show that $L_1 \notin \mathscr{L}_{n+r_2(n)}(\text{OCA})$.

Since $r_2(f(n)) \in o(t_2(n))$ and L_a is not acceptable within $n + o(t_2(n))$ time by any OCA, the language L_a is not acceptable within $n + r_2(f(n))$ time by any OCA, either. Due to the assumption, by Lemma 13 for every $s \in \mathbb{N}$ there must exist sets X and Y and an $n \in \mathbb{N}$ such that all words in X are of length m_1, all words in Y are of length m_2, and the number of (L_a, X, Y)-equivalence classes of the words at most of length $n - m_1 - m_2$ is not bounded by $s^{|X|}s^{(m_2+r_2(f(n)))|Y|}$.

Define

$$X' = \{(x_1, a) \cdots (x_{m_1}, a) \mid x = x_1 \ldots x_{m_1} \in X\}$$

and

$$Y' = \{(y_1, b) \cdots (y_{m_2}, b) \mid y = y_1 \ldots y_{m_2} \in Y\}$$

and for every word $v = v_1 \cdots v_{n-m_1-m_2}$ a word v' by

$$v' = (v_1, w_1) \cdots (v_{n-m_1-m_2}, w_{n-m_1-m_2})(\sqcup, w_{n-m_1-m_2+1}) \cdots (\sqcup, w_{f(n)-m_1-m_2})$$

where $a^{m_1} w_1 \cdots w_{f(n)-m_1-m_2} b^{m_2}$ is a word of L_f. (Remember that each word in L_f starts with n symbols a and ends with n symbols b and $m_1 + m_2 \leq n$.)

For $x \in X$ and $y \in Y$ let x' and y' denote the corresponding words in X' and Y'.

By construction $xvy \in L_a$ iff $x'v'y' \in L_1$. (The word $x'v'y'$ belongs to L_1 if the second component of $x'v'y'$ is a word in L_f, which is always true, and the first component of $x'v'y'$ is a word in L_a concatenated with some blank symbols, i.e., $xvy \in L_a$.) Thus, the (L_a, X, Y)-equivalence classes have corresponding (L_1, X', Y')-equivalence classes and the number of (L_1, X', Y')-equivalence classes under the words whose length is at most $f(n) - m_1 - m_2$ is not bounded by $s^{|X|} s^{(m_2 + r_2(f(n)))|Y|}$.

Applying Lemma 13 with L_1, X', Y' and $f(n)$ in place of L_a, X, Y and n yields that $L_1 \notin \mathscr{L}_{n+r_2(n)}(\mathrm{OCA})$. This completes the proof. □

Finally the main Theorem 3 is just a combination of the preceding lemmas:

Theorem 3. Let $r_1, r_2 : \mathbb{N} \to \mathbb{N}$ be two increasing functions. If $r_2 \log(r_2) \in o(r_1)$ and r_1^{-1} is constructible, then

$$\mathscr{L}_{n+r_2(n)}(\mathrm{OCA}) \subset \mathscr{L}_{n+r_1(n)}(\mathrm{OCA})$$

Proof. Proposition 15 shows that the previously defined language L_d is acceptable in linear-time but is not acceptable in $n + r(n)$ time if $r(n) \log(r(n)) \in o(n)$. Now set $t_1(n) = n$ and t_2 such that $t_2(n) \log(t_2(n)) = n$. Inserting yields $r(n) \log(r(n)) \in o(t_2(n) \log(t_2(n)))$.

We conclude $r(n) \in o(t_2(n))$ and, thus, L_d is not acceptable in $n + o(t_2(n))$ time. In order to apply Lemma 16 we consider the constructible function $f = r_1^{-1}$.

Clearly, $r_1(f(n)) = n \in \Omega(n) = \Omega(t_1(n))$.

Since $r_2(n) \log(r_2(n)) \in o(r_1(n))$ we have

$$r_2(f(n)) \log(r_2(f(n))) \in o(r_1(f(n))) = o(n) = o(t_2(n) \log(t_2(n)))$$

We conclude $r_2(f(n)) \in o(t_2(n))$.

Now all conditions of Lemma 16 are satisfied and an application proves the assertion $\mathscr{L}_{n+r_2(n)}(\mathrm{OCA}) \subset \mathscr{L}_{n+r_1(n)}(\mathrm{OCA})$. □

6 Closure Properties

In the following we are exploring some of the closure properties of the OCA-language families in the range between real-time and linear-time.

Our first results in this section deal with Boolean operations. Since the OCAs are space-bounded deterministic devices the positive properties are natural.

Lemma 17 *Let $r : \mathbb{N} \to \mathbb{N}$ be a function, then $\mathscr{L}_{n+r(n)}(\mathrm{OCA})$ is closed under union and intersection.*

Proof. Using the same two chanel technique of [4] and [8] the assertion can easily be seen. Each cell consists of two registers in which acceptors for both languages are simulated in parallel. □

The closure under complement is expected for deterministic devices. But here an input is rejected by not entering an accepting state. So in order to accept the complement of a language the OCA has to calculate the latest time step at which the input would have been accepted. With other words, it has to calculate the time step $n + r(n)$. The corresponding functions are called *computable* [2].

Lemma 18 *Let $r : \mathbb{N} \to \mathbb{N}$ be a computable function, then $\mathscr{L}_{n+r(n)}(\mathrm{OCA})$ is closed under complement.*

Proof. Let $\mathcal{A}$ be some $(n + r(n))$-time OCA. A OCA $\mathcal{A}'$ that accepts the complement of $L(\mathcal{A})$ works as follows. On its first track it computes the function $n + r(n)$. On its second track $\mathcal{A}'$ simulates $\mathcal{A}$ for at most $n + r(n)$ time steps. If during the simulation $\mathcal{A}$ would become accepting, then the leftmost cell of $\mathcal{A}'$ changes to a distinguished state from which no accepting state is reachable. So $\mathcal{A}'$ rejects the input.

If, on the other hand, $\mathcal{A}$ would not become accepting until time step $n + r(n)$, then $\mathcal{A}'$ simply accepts its input. □

Now we turn to more language specific closure properties. The families are closed under marked concatenation but are not closed under concatenation.

Lemma 19 *Let $r : \mathbb{N} \to \mathbb{N}$ be an increasing function, then $\mathscr{L}_{n+r(n)}(\mathrm{OCA})$ is closed under marked concatenation and marked iteration.*

Proof. Let $\mathcal{A}$ and $\mathcal{A}'$ be two $(n + r(n))$-time OCAs. A OCA $\mathcal{A}''$ that accepts the marked concatenation $L(\mathcal{A})L(\mathcal{A}')$ with time complexity $n + r(n)$ has two tracks. Initially all inner cells start to simulate $\mathcal{A}$ and $\mathcal{A}'$ in parallel. The rightmost non-border cell identifies itself and starts to simulate $\mathcal{A}'$ only. This behavior proceeds to the left such that the cells between the right border and the mark are successively identified.

Analogously, the cell whose right neighbor contains the mark can identify itself. It starts to simulate $\mathcal{A}$ only and signals this behavior to the left.

Let $n = n_1 + 1 + n_2$ be the length of the input where the first n_1 symbols form a word from $L(\mathcal{A})$ and the last n_2 symbols form a word from $L(\mathcal{A}')$. In between the words there is exactly one marking symbol. Then the simulation of $\mathcal{A}$ takes $n_1 + r(n_1)$ time steps and the simulation of $\mathcal{A}'$ takes $n_2 + r(n_2)$ time steps. Since the latter result has to move into the leftmost cell the time complexity of $\mathcal{A}''$ is

$$\max\{n_1 + r(n_1), n_2 + r(n_2) + n_1\} + 1$$

In both cases $\mathcal{A}''$ obeys the time complexity $n + r(n)$.

The generalization to an arbitrary number of concatenated input words and, thus, to marked iteration is straightforward. □

Lemma 20 *Let $r : \mathbb{N} \to \mathbb{N}$ be an increasing function. If $r(n)\log(r(n)) \in o(n)$, then $\mathscr{L}_{n+r(n)}(\mathrm{OCA})$ is not closed under concatenation.*

Proof. Let us consider the language L_l whose words are of the form

$$x\,(\,x_1 \mid y_1\,)\cdots(\,x_n \mid y_n\,)\,(\,x \mid$$

where $x, x_i, y_i \in \{0,1\}^*$ for $1 \leq i \leq n$. Since $\{w \bullet w \mid w \in \{0,1\}^+\}$ is a real-time OCA-language L_l is a real-time OCA-language. The same holds for the language L_r whose words are of the form

$$y\,)\,(\,x_1 \mid y_1\,)\cdots(\,x_n \mid y_n\,)\,y$$

where $y, x_i, y_i \in \{0,1\}^*$ for $1 \leq i \leq n$.

It follows L_l and L_r are $(n + r(n))$-time OCA-languages. But the concatenation L_lL_r equals the language L_d which has been proven not to belong to $\mathscr{L}_{n+r(n)}(\mathrm{OCA})$ in Propostion 15. □

Corollary 21 *Let $r : \mathbb{N} \to \mathbb{N}$ be an increasing function. If $r(n)log(r(n)) \in o(n)$, then $\mathscr{L}_{n+r(n)}(\mathrm{OCA})$ is not closed under iteration.*

Proof. A word of the form

$$x\,(\,x_1 \mid y_1\,)\cdots(\,x_n \mid y_n\,)\,y$$

where $x, x_i, y, y_i \in \{0,1\}^+$ for $1 \leq i \leq n$ belongs to L_d if and only if it belongs to $(L_l \cup L_r)^*$. Since the structure of the words in L_d is regular the structure is a real-time OCA-language. Since $\mathscr{L}_{n+r(n)}(\mathrm{OCA})$ is closed under intersection with regular sets and union the closure under iteration would imply $L_d \in \mathscr{L}_{n+r(n)}(\mathrm{OCA})$ what contradicts Proposition 15. □

Now some closure properties concerning homomorphisms are shown.

Lemma 22 *Let $r : \mathbb{N} \to \mathbb{N}$ be an increasing function. If $r(O(n)) \subseteq O(r(n))$, then $\mathscr{L}_{n+r(n)}(\mathrm{OCA})$ is closed under inverse homomorphisms.*

Proof. Let $L \in \mathscr{L}_{n+r(n)}(\mathrm{OCA})$ be a language over some alphabet A and $h : B^* \to A^*$ be a homomorphism. A $(n + r(n))$-time OCA $\mathcal{A}$ which accepts the language $h^{-1}(L)$ works as follows.

Let $k = \max\{|h(b)| \mid b \in B\}$ be the maximal length of the symbol images of h. Then $\mathcal{A}$ has k registers.

Basically, each cell of $\mathcal{A}$ maps its input symbol according to h, stores the image into its registers and simulates the acceptor for L.

Let n be the length of the input w of $\mathcal{A}$ and m be the length of $h(w)$. We are concerned with two cases.

If some input symbol is mapped to λ, the resulting empty registers of the corresponding cell cause a delay until the necessary information for the next simulation step is available (cf. proof of Proposition 8). So the total delay is bounded by the number of empty cells.

On the other hand, if some symbol is mapped to more than one symbol, the registers of the corresponding cell and the registers of its left neighbor can simulate more than one transition during one time step.

The leftmost cell of $\mathcal{A}$ has to simulate at least $m + r(m)$ transitions. Since it may happen that the rightmost non-empty cell contains only one symbol, $\mathcal{A}$ has to simulate this cell for $r(m)$ time steps, i.e., $r(m)$ transitions. After time $r(m)$ the state of this cell will not influence the overall computation result of the simulated acceptor. Since all the cells on the left simulate as many transitions per time step as possible (depending on the number of filled registers and the number of simulated steps of the right neighbor), $\mathcal{A}$ needs another n time steps to complete the simulation. Altogether the simulation takes $n + r(m)$ time steps.

If $m \leq n$ then $\mathcal{A}$ works fine and trivially obeys the time complexity $n + r(n)$.

In the second case we have $m > n$. Since m is linearly bounded by n, i.e., $m \leq k \cdot n$ for the constant k, $n + r(k \cdot n)$ is an upper bound of the time complexity. Due to the assumption there exists a constant k' such that $n + r(k \cdot n) \leq n + k' \cdot r(n)$. By the well-known result in [6] the computation can be sped-up linearly from $n + k' \cdot r(n)$ to $n + \varepsilon \cdot r(n)$ for $\varepsilon > 0$. □

Lemma 23 *Let $r : \mathbb{N} \to \mathbb{N}$ be an increasing function. If $r(n)\log(r(n)) \in o(n)$, then $\mathscr{L}_{n+r(n)}$(OCA) is not closed under λ-free homomorphisms.*

Proof. Construct a language $L \subseteq (\{\sqcup, \bullet\} \times \{0, 1, (,), |\})^*$ as follows:

1. The second component of each word in L is of the form
$$x\,(\,x_1 \,|\, y_1\,) \cdots (\,x_n \,|\, y_n\,)\,y$$
2. The first component is such that exactly one pair $(\,x_i \,|\, y_i\,)$, $1 \leq i \leq n$, is marked by $\bullet$. All the other registers contain $\sqcup$.
3. The components of the marked pair match x and y, respectively. I.e. $x = x_i$ and $y = y_i$.

Clearly, L is a real-time OCA-language. But the image of L under the λ-free homomorphism $h : (\{\sqcup, \bullet\} \times \{0, 1, (,), |\})^* \to \{0, 1, (,), |\}^*$, $h(a, b) = b$, that maps a pair to its second component is the language L_d which does not belong to $\mathscr{L}_{n+r(n)}$(OCA).

Acknowledgements

The authors are greatful to Véronique Terrier whose valuable comments improved the upper bound in Lemma 13 and thus the tightness of the hierarchy.

References

[1] Th. Buchholz, A. Klein, M. Kutrib, On tally languages and generalized interacting automata, in: G. Rozenberg, W. Thomas (Eds.), Developments in Language Theory IV. Foundations, Applications, and Perspectives, World Scientific Publishing, Singapore, 2000, pp. 316–325.

[2] Th. Buchholz, M. Kutrib, On time computability of functions in one-way cellular automata, Acta Inf. 35 (1998) 329–352.

[3] C. Choffrut, K. Čulik II, On real-time cellular automata and trellis automata, Acta Inf. 21 (1984) 393–407.

[4] C. R. Dyer, One-way bounded cellular automata, Inform. Control 44 (1980) 261–281.

[5] P. C. Fischer, Generation of primes by a one-dimensional real-time iterative array, J. Assoc. Comput. Mach. 12 (1965) 388–394.

[6] O. H. Ibarra, M. A. Palis, Some results concerning linear iterative (systolic) arrays, J. Parallel and Distributed Comput. 2 (1985) 182–218.

[7] J. Mazoyer, V. Terrier, Signals in one dimensional cellular automata, Theoret. Comput. Sci. 217 (1999) 53–80.

[8] A. R. Smith III, Real-time language recognition by one-dimensional cellular automata, J. Comput. System Sci. 6 (1972) 233–253.

[9] V. Terrier, Language not recognizable in real time by one-way cellular automata, Theoret. Comput. Sci. 156 (1-2) (1996) 281–287.

[10] V. Terrier, On real time one-way cellular array, Theoret. Comput. Sci. 141 (1995) 331–335.