Arbeitsgruppe Informatik
Universität Giessen
Arndtstr. 2, D-35392 Giessen, Germany

On time computability of functions in one-way cellular automata

Thomas Buchholz   Martin Kutrib

Report 9502   September 1995

On time computability of functions in one-way cellular automata

Thomas Buchholz and Martin Kutrib

AG Informatik, University of Giessen
Arndtstr. 2, D-35392 Giessen, Germany

{buchholz,kutrib}@informatik.uni-giessen.de

September 1995

Abstract

The capability of one-way (space-bounded) cellular automata (OCA) to time-compute functions is investigated. That means given an constant input of length n a distinguished cell has to enter a distinguished state exactly after $f(n)$ time steps. The family of such functions ($\mathscr{C}$(OCA)) is characterized in terms of formal language recognition. Several functions are proved to be time-computable and properties of $\mathscr{C}$(OCA) are given. The time-computation at some points is concerned with the concept of signals and their realization which is quite formally defined for the first time.

1 Introduction

The computational complexity theory can be said to begin in the early sixties. Since these days there is a particular interest in Turing machine complexity hierarchies. To obtain infinite dense hierarchies of complexity classes defined by bounding some resources by a function f, one has to show that only a slight increase in the growth rate of f yields to a new complexity class. Most of the corresponding proofs require "well-behaved" complexity functions. The notion "well-behaved" is usually concretized in terms of the constructibility of functions with respect to the device investigated.

If we investigate the constructibility of functions in parallel polyautomata we are strongly concerned with the concept of signals. In a lot of works specific pulses (or signals) are an useful tool to construct algorithms, for example prime number generators [5] or a solution to the famous firing squad synchronization problem [1, 15]. A general investigation of signals of its own in polyautomata was started by Mazoyer and Terrier [12, 10]. They considered signals and the constructibility of functions in two-way unbounded cellular automata (cellular spaces).

Since signals can encode and propagate information through the automaton their realizability can show us the computation power and the limitations of the model. Moreover, we can regard signals as a higher programming concept which allows modularization techniques at algorithm design.

In the field of polyautomata theory a problem arises with the end of a computation. From the usual definition it follows that the machines will never halt. A common way of defining final configurations is to define a predicate these configurations have to fulfill. In case of language recognition this predicate mostly requires a border cell to be in a designated final state. But what afterwards a final configuration is reached? One can additionally require that final configurations are stable in some sense, i.e. a final configuration leads always to a final configuration. If this property is not required and a certain complexity class is under consideration, i.e. the resulting configuration is taken by the outside world at a time step $f(n)$, then the ability of recognizing time step $f(n)$ is an ability of the outside world and not necessarily of the model. But both cases coincide if the time step $f(n)$ is recognizable by the automaton itself.

Let us consider an example: An one-way real-time cellular automata language can be recognized by an one-way cellular automaton in $n-1$ time steps if we do not require stable final configurations [3]. Otherwise we need n time steps since the time step $n-1$ cannot be identified by any one-way cellular automaton.

The paper is organized as follows:
In section 2 we define the basic notions as well as give a quite formal definition of the concept of signals and its realization by one-way cellular automata. Section 3 is devoted to some examples of OCA-time-computable functions which will be used in later sections and shed some first light on the range of $\mathscr{C}$(OCA). In section 4 the family $\mathscr{C}$(OCA) is fully characterized in terms of a sub-family of the real-time OCA languages, whereas in section 5 the range of $\mathscr{C}$(OCA) is characterized by some of its properties. Especially, closure properties are considered in section 6.

2 OCAs and time computability

We denote the integers by $\mathbb{Z}$, the positive natural numbers $\{1, 2, \ldots\}$ by $\mathbb{N}$, the set $\mathbb{N} \cup \{0\}$ by $\mathbb{N}_0$.

A cellular automaton is a linear array of deterministic finite automata, sometimes called cells, each of them is connected to its nearest neighbor(s). For our convenience we identify the cells by natural numbers. The state transition depends on the actual state of each cell and the actual state(s) of its neighbor(s). The transition function is applied to all cells synchronously at discrete time steps. More formally:

Definition 1 A *cellular automaton* (CA) is a system $(S, \sigma, \#)$, where

a) S is the finite, nonempty set of *states*,
b) $\# \in S$ is the *boundary state*,
c) $\sigma : S^3 \to S$ is the *local transition function* satisfying $\forall\, s_1, s_2, s_3 \in S$:

$$\sigma(s_1, s_2, s_3) = \# \iff s_2 = \#$$

The local transition function induces a length preserving mapping $\mathcal{T} : S^+ \longrightarrow S^+$ according to the following:

Let $n \in \mathbb{N}$ be an arbitrary natural number and $s_1, \ldots, s_n \in S$

$$\mathcal{T}(s_1 \cdots s_n) := \begin{cases} \sigma(\#, s_1, \#) & \text{if } n = 1 \\ \sigma(\#, s_1, s_2)\sigma(s_1, s_2, s_3) \cdots \sigma(s_{n-1}, s_n, \#) & \text{otherwise} \end{cases}$$

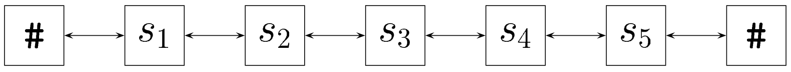

Figure 1: A cellular automaton.

We often refer to configurations c_i of CAs at time steps $i \geq 0$. Let c_0 be defined by the initial sequence of states in a CA, then we define $c_{i+1} := \mathcal{T}(c_i)$ and $c_i(j) := \pi_j(c_i)$. $\pi_i(s_1 \cdots s_n) := s_i$ selects the ith component of $s_1 \cdots s_n$. 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. Accordingly we define $c_i^k(j) := \pi_k(\pi_j(c_i))$.

If the flow of information is restricted to one-way (e.g. from right to left), the resulting device is an *one-way cellular automaton* (OCA). I.e. the next state of

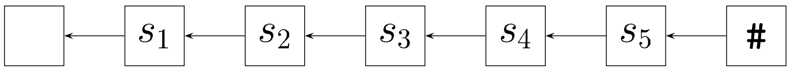

Figure 2: An one-way cellular automaton.

each cell depends on the state of the cell itself and the state of its right neighbor only.

In the rest of the paper we are strongly concerned with functions mapping the natural numbers to $\mathbb{N}$ or sometimes to $\mathbb{Z}$.

Definition 2 A function $f : \mathbb{N} \longrightarrow \mathbb{N}$ or $f : \mathbb{N} \longrightarrow \mathbb{Z}$ is said to be

a) *ultimately periodic* with *period length* $p \iff \exists\, n_0 \in \mathbb{N} : \forall\, n \geq n_0 : f(n) = f(n+p)$,
b) *periodic* with *period length* $p \iff \forall\, n \geq 1 : f(n) = f(n+p)$,
c) *increasing* $\iff \forall\, n \in \mathbb{N} : f(n) \leq f(n+1)$,
d) *strictly increasing* $\iff \forall\, n \in \mathbb{N} : f(n) < f(n+1)$,
e) *ultimately constant* $\iff \exists\, n_0, k \in \mathbb{N} : \forall\, n \geq n_0 : f(n) = k$,
f) *constant* $\iff \exists\, k \in \mathbb{N} : \forall\, n \geq 1 : f(n) = k$.

Now we are going to formalize the concept of signals. Intuitively during its existence a signal resides in exactly one cell at every time step and can propagate to it neighbors only.

Definition 3 A *signal* is a partial function $\xi : \mathbb{N}_0 \longrightarrow \mathbb{Z}$ which satisfies the properties $\xi(0) = 0$ and $\xi(i)$ undefined $\implies \forall\, j \in \mathbb{N} : \xi(i+j)$ undefined. We denote the greatest natural number on which ξ is defined by l_ξ. In case of a total function the value of l_ξ is ∞, otherwise ξ is said to be *finite.*

What we have not yet defined is the realization of signals in CAs. Due to the finiteness of the state set one single cell can only carry a finite number of signals at the same time. If the initial configuration is long enough then we can realize each signal arbitrary times as long as the instances are not intersect.

Definition 4 Let $\xi_0, \xi_1 \ldots, \xi_m, \ldots$ be a (possibly infinite) numbering of signals and let the set of its indexes be denoted by I. A CA $(S = S_0 \times S_1 \times \cdots \times S_r, \sigma, \#)$ *realizes* for all $i \in I$ the signal ξ_i in register r_i at time t_i in cell k_i if the following is true

a) $\forall\, i \in I : \forall\, m \in \mathbb{N} :$
$\big(\xi_i(m+1)$ undefined $\vee\ \xi_i(m+1) \in \{\xi_i(m) - 1, \xi_i(m), \xi_i(m) + 1\}\big)$

b) $\forall\, i,j \in I : \Big(i = j \vee r_i \neq r_j \vee$
$\forall\ \max\{t_i, t_j\} \leq t \leq \min\{t_i + l_{\xi_i}, t_j + l_{\xi_j}\} : \xi_i(t-t_i) + k_i \neq \xi_j(t-t_j) + k_j\Big)$

c) $\forall\, i \in I : \Big(1 \leq r_i \leq r \wedge \exists\, D_{r_i} \subset S_{r_i} : D_{r_i} \cap \{\bot\} = \emptyset\Big)$

d) $\forall\, 1 \leq m \leq r : \quad c_t^m(j) \in$
$$\begin{cases} \{\bot\} & \text{if } \forall\, i \in I : \Big(r_i \neq m \vee \xi_i(t-t_i) \text{ undefined } \vee \xi_i(t-t_i) + k_i \neq j\Big) \\ D_m & \text{otherwise} \end{cases}$$

Observe that we have simply to change the condition $\forall\, i \in I : \forall\, m \in \mathbb{N} : \Big(\xi_i(m+1) \text{ undefined } \vee \xi_i(m+1) \in \{\xi_i(m)-1, \xi_i(m), \xi_i(m)+1\}\Big)$ to

$$\forall\, i \in I : \forall\, m \in \mathbb{N} : \Big(\xi_i(m+1) \text{ undefined } \vee \xi_i(m+1) \in \{\xi_i(m)-1, \xi_i(m)\}\Big)$$

to obtain a definition for a signal realizing OCA.

The previous definition gives insight in whether a set of signals is realized by a specific (O)CA or not; but it gives no insight in how the realization can be done.

Mazoyer and Terrier [10] have investigated realizations of signals in unbounded two-way so-called *impulse cellular automata*. They call a signal *basic* if its sequence of elementary moves $\{\xi(t+1) - \xi(t)\}_{t \in \mathbb{N}}$ is ultimately periodic.
A signal ξ is leftward resp. rightward if for all $0 \leq t < l_\xi : \xi(t) \geq \xi(t+1)$ resp. $\xi(t) \leq \xi(t+1)$.

A realization in bounded (O)CAs requires that the signal does not get out of the borders of the automaton. To avoid an overloading with signal definitions one can imagine that signals are dropped while crossing the border cells.

Under these assumptions if a single cell n is able to recognize the appropriate time step t, finite or basic signals can be realized at time t in cell n by CAs independently of any other computation. If it is additionally leftward it is realizable by OCAs, too. For the converse it can easily be shown that signals which are realizable in (O)CAs independently of any other computation must be finite or basic. On the other hand there are realizable signals which are neither finite nor basic. In general it will be necessary to use some auxiliary signals for their realization.

Let us consider two examples.
The signal $\xi : \mathbb{N}_0 \longrightarrow \mathbb{Z}, \quad n \mapsto -\lfloor \frac{n}{3} \rfloor$ is basic since the sequence of its elementary moves $0, 0, -1, 0, 0, -1, \ldots$ is periodic. It moves with *speed* $\frac{1}{3}$ leftwards (cf. figure 3).

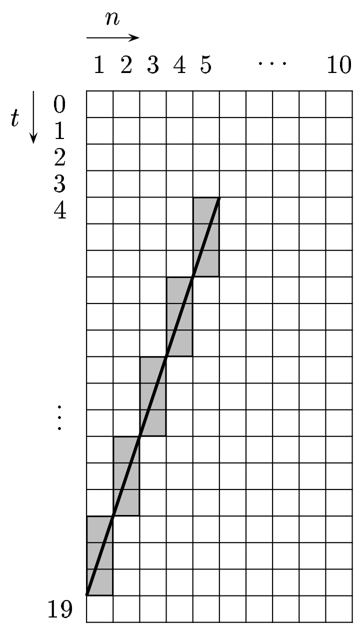

Figure 3: OCA-realization of ξ at time 4 in cell 5. At time 19 ξ is dropped.

Obviously, the signal $\xi : \mathbb{N}_0 \longrightarrow \mathbb{Z}$,

$$\xi(0) := 0, \qquad \xi(n) := \frac{1}{4} \cdot 2^{\lceil \log_2 n \rceil} - \left| n - \frac{3}{4} \cdot 2^{\lceil \log_2 n \rceil} \right| + 1$$

is not basic. We use the auxiliary signal $n \mapsto \lfloor \frac{n}{3} \rfloor$ for its realization in a CA (cf. figure 4). ξ is not OCA-realizable.

The main interest in the present paper is focussed on the time computability of functions. That is what we really want:

Definition 5 A function $f : \mathbb{N} \to \mathbb{N}$ is *(O)CA-time-computable* iff there exist a(n) (O)CA $M = (S, \sigma, \#)$, a distinguished state $s_0 \in S$ and a set of *final states* $F \subseteq S$ such that $f(n)$ is the smallest natural number for which $\pi_1(\mathcal{T}^{f(n)}(s_0^n)) \in F$.

We denote the family of OCA-time-computable resp. CA-time-computable functions by $\mathscr{C}(\mathrm{OCA})$ resp. $\mathscr{C}(\mathrm{CA})$.

The concepts of time computability and signals are basically different. For OCAs there is the following relationship. Every strictly increasing time-computable function describes a realizable signal. For the converse consider the realizable constant signal $n \mapsto 1$ which leads not to a function in $\mathscr{C}(\mathrm{OCA})$. On the

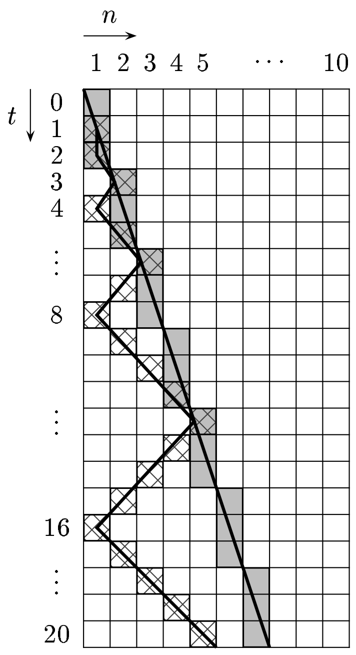

Figure 4: CA-realization of ξ at time 0 in cell 1.

other hand we have to require strictly increasing functions since it will be seen that, for example, $n \mapsto \begin{cases} n & \text{if } n \text{ is even} \\ 2n & \text{otherwise} \end{cases}$ and $n \mapsto \begin{cases} n+1 & \text{if } n \text{ is even} \\ n & \text{otherwise} \end{cases}$ are OCA-time-computable but do not describe realizable signals.

3 Examples for OCA time computability

In the sequel f always denotes a mapping from $\mathbb{N}$ to $\mathbb{N}$ and the mapping $n \mapsto n$ is denoted by id. For all $k \in \mathbb{N}$ we will use the notion k for the constant function $\mathrm{k}(n) = k$.

In the present section we give some examples of OCA-time-computable functions. Besides the results are interesting of its own they will be used in later sections to prove further properties. Rather than giving it in a formal way, which would be tedious and hard to read, our constructions are somewhat informal.

At first we investigate the world below id. As we will later see in theorem 19 all OCA-time-computable functions below id are ultimately constant.

Lemma 6 $\forall\, k \in \mathbb{N} : \mathrm{k} \in \mathscr{C}(\mathrm{OCA})$

Proof. For a given k we can construct an OCA which does the job:
$S := \{s_0, \dots, s_k\} \cup \{\texttt{\#}\}$ and $\forall\, s_i, s_j \in S \setminus \{\texttt{\#}\} : \sigma(s_i, s_j) := s_{(i+1) \bmod (k+1)}$. The set of final states is $\{s_k\}$. Obviously for all $n \in \mathbb{N} : \pi_1(\mathcal{T}^k(s_0^n)) = s_k$ from which the lemma follows. □

The next question is concerned with the function id itself or, more general, the functions $k \cdot \mathrm{id}$ for $k \in \mathbb{N}$.

Lemma 7 $\forall\, k \in \mathbb{N} : k \cdot \mathrm{id} \in \mathscr{C}(\mathrm{OCA})$

Proof. Since in an OCA the right border cell n can identify itself the leftward basic signal $m \mapsto -\lfloor \frac{m}{k} \rfloor$ for all $k \in \mathbb{N}$ can be realized at time 1 in cell n. Now simply each cell passed through enters a final state. □

The following two lemmas have previously been shown in terms of signals by Mazoyer and Terrier [10] for two-way unbounded CAs. Their constructions use only leftward signals such that the proofs can easily be adapted.

Lemma 8 $\forall\, k \geq 2 : k \cdot \mathrm{id} + \lfloor \sqrt{\mathrm{id}} \rfloor \in \mathscr{C}(\mathrm{OCA})$

For the investigation of general properties of OCA-time-computable functions the function $\mathrm{id} + \lfloor \log \rfloor$ becomes important. With an eye towards later extensions we adapt and recall the proof that it belongs to $\mathscr{C}(\mathrm{OCA})$ from [10].

Lemma 9 $\forall\, b \geq 2 : \mathrm{id} + \lfloor \log_b \rfloor \in \mathscr{C}(\mathrm{OCA})$

Proof. In order to proof the lemma we give a construction of a corresponding OCA for $b = 2$ (cf. figure 5). The construction for another basis is straightforward.

At initial time the OCA generates at the right border cell (which can identify itself) a leftward signal. The signal moves with speed 1 and strikes out each other cell passed through. Moreover, in the first cell which is not struck out it generates a new signal. This one is one time step delayed and behaves as the first one. It proceeds with speed 1 to the left and strikes out each other non struck out cell. If it passes through a cell which will not be struck out it generates again a new signal which behaves identical. Since the number of cells left decreases with each signal the algorithm terminates.

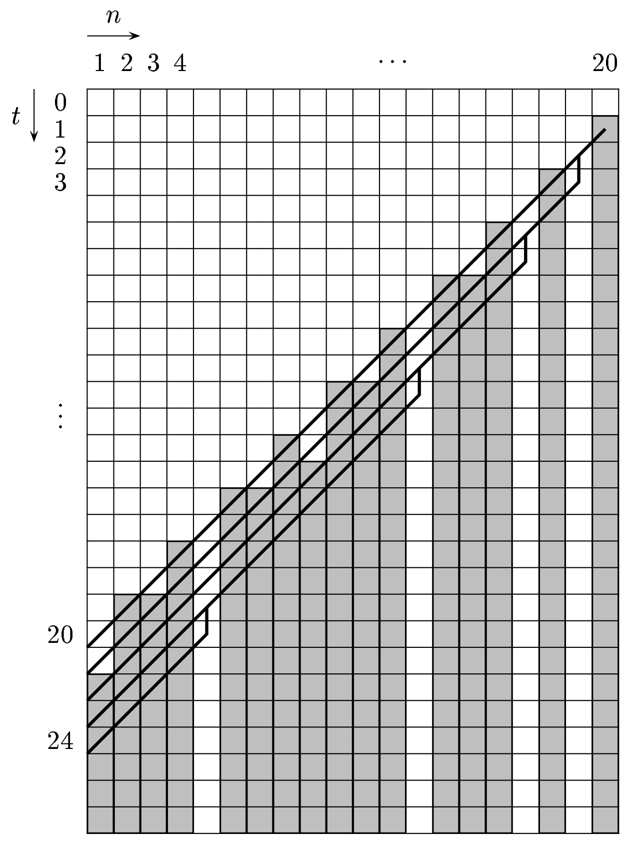

Figure 5: Example computation to lemma 9. $n = 20$, $f(n) = 24$.

As long as the signals have not struck out any cell on its way to the left they cause the cells passed through to enter a final state. Consequently, the unique signal which does not find any cell to strike out sets the left border cell into a final state for the first time.

Let $s(n)$ be the number of signals on input length n. Obviously, $s(n) = \lfloor 1 + \log_2 n \rfloor$. Since the first signal arrives at the left border at time n and every new signal is delayed by one time step, the total computation time and, hence, $f(n)$ is $n + \lfloor \log_2 n \rfloor$. $\square$

Lemma 10 $\forall\, b_1 \geq 2, b_2 \geq 2:$

$$f(n) := \begin{cases} 2 \cdot n + \lfloor \log_{b_1} \log_{b_2} n \rfloor & \text{if } n \geq b_2 \\ 2 \cdot n & \text{otherwise} \end{cases} \in \mathscr{C}(\text{OCA}).$$

Proof. The construction bases on that of the previous lemma. For each

$b_1 \geq 2, b_2 \geq 2$ there is an $n_0 \in \mathbb{N}$ such that $\forall\, n \geq n_0 : 2n + \lfloor \log_{b_1} \log_{b_2} n \rfloor \geq n + \lfloor log_{b_2} n \rfloor$. Because the number of concerned cells ($\leq n_0$) depends on b_1 and b_2 only and is finite, details to handle it especially are omitted.

An OCA for $2 \cdot \mathrm{id} + \lfloor \log_{b_1} \log_{b_2} \rfloor$ simulates one for $\mathrm{id} + \lfloor \log_{b_2} \rfloor$ as shown in lemma 9. Additionally it initiates at time step 0 a leftward signal with speed 1/2 that operates as a corresponding leftward signal for $\mathrm{id} + \lfloor \log_{b_1} \rfloor$ would do with the exceptions that only the $\lfloor \log_{b_2} \rfloor$ cells not struck out by the first task are considered and all further generated signals are propagating with speed 1/2. Altogether that leftward signal would need $2n$ time steps to reach the left border, but is delayed by $\lfloor \log_{b_1} \lfloor \log_{b_2} n \rfloor \rfloor = \lfloor \log_{b_1} \log_{b_2} n \rfloor$ time steps. □

The polynomials form a wide class of important and interesting functions. Unfortunately, we do not know whether e.g. id^2 is OCA-time-computable. But we have a result for its order of magnitude.

Theorem 11 $\forall\, k \in \mathbb{N} : \exists\, f \in \mathscr{C}(\mathrm{OCA}) : f \in \Theta(\mathrm{id}^k)$

Proof. At first we will give the construction of an appropriate OCA and afterwards we are going to analyze its behavior.

Let $m := 2^k$ and $S := \{s_0, \ldots, s_{m-1}\} \cup \{\#\}$.

$$\forall\, s_i, s_j \in S : \sigma(s_i, s_j) := \begin{cases} s_{(i+1) \bmod m} & \text{if } s_j \in \{s_{m-1}, \#\} \\ s_i & \text{otherwise} \end{cases}$$

The set of final states is $\{s_{m-1}\}$.

Let for each cell i the value $g(i)$ be defined as $g(i) := \lfloor \log_2(i) \rfloor + 1$. Observe, that the cells are grouped together by their corresponding value g, and that the number of members of each group is exactly the sum of the members of all former groups plus one. In the following we will identify each group by its value of g.

We claim that the behavior of a cell i is periodic with period length $m^{g(i)}$ and that it runs through the final state at time $t_i = m^{g(i)} - m^{g(i)-1} + t_{i-2^{g(i)-1}}$ for the first time. Moreover, if i is one less than a power of 2 (i.e. the last member in a group) it runs through the final state exactly once at the end of every period.

In the following table the period length, the group number, the number of final state occurrence during one period and the time step the final state will be entered firstly for the first 15 cells are depicted.

i	$g(i)$	period length	$\#\ s_{m-1}$ in period	firstly final at time
1	1	m	1	$m-1$
2	2	m^2	m	m^2-m
3	2	m^2	1	$m^2-m+m-1$
4	3	m^3	m^2	m^3-m^2
5	3	m^3	m	m^3-m^2+m-1
6	3	m^3	m	$m^3-m^2+m^2-m$
7	3	m^3	1	$m^3-m^2+m^2-1$
8	4	m^4	m^3	m^4-m^3
9	4	m^4	m^2	m^4-m^3+m-1
10	4	m^4	m^2	$m^4-m^3+m^2-m$
11	4	m^4	m	$m^4-m^3+m^2-1$
12	4	m^4	m^2	$m^4-m^3+m^3-m^2$
13	4	m^4	m	$m^4-m^3+m^3-m^2+m-1$
14	4	m^4	m	$m^4-m^3+m^3-m$
15	4	m^4	1	$m^4-m^3+m^3-1$

To our convenience we assume $t_0 = 0$. We will conclude inductively.

In the first group $g = 1$ we have just cell 1. Clearly, since its one and only neighbor is in the border state, from the definition of the local transformation it follows that its behavior is the repeated sequence $s_0 s_1 \cdots s_{m-1}$. Hence, its period length is m and it firstly changes to the final state at time $m^1 - m^0 + t_0 = m-1$. Moreover, cell 1 is the last cell of group 1 and it passes through the final state exactly at the end of every period.

Up to now we have shown that all the cells of the first group meet the claim.

Let us assume now that all cells of a group g meet the claim. We have to show that all the cells of group $g+1$ meet the claim as well.

From the known behavior of the last cell of group g we derive the behavior of the first cell of group $g+1$. It is the repeated sequence

$$\overbrace{s_0 \cdots s_0}^{m^g} \overbrace{s_1 \cdots s_1}^{m^g} \quad \cdots \quad \overbrace{s_{m-1} \cdots s_{m-1}}^{m^g}$$

since the last cell passes through the state s_{m-1} every m^g time steps which leads to a state change of the first cell. Consequently the claim holds.

The second cell of group $g+1$ remains in state s_0 for the first $m^{g+1} - m^g$ time steps. From that time on it passes during m^g time steps through the sequence

$s_0 \cdots s_{m-1}$ exactly m^{g-1} times. Afterwards its behavior repeats. Consequently, its period length is m^{g+1} and it is final at time $m^{g+1} - m^g + m - 1$ at the first, from which the claim follows.

In other words the behavior of the second cell consists of $m^{g+1} - m^g$ time steps s_0 and afterwards m^{g-1} times the behavior of the very first cell. Thus, all other cells of the group behavior starts with $m^{g+1} - m^g$ times s_0 and subsequently it is for m^g time steps identical to the behavior of a corresponding earlier cell. I.e. the second cell corresponds to the very first cell, the third cell to the very second cell and so on. Therefore, the last cell of the group corresponds to the last cell of the predecessor group, from which immediately follows that all cells of the group $g+1$ have period length m^{g+1} and that the last cell of that group passes through the final state exactly at the end of every period. Furthermore, each cell $i \in \{2^g, \cdots, 2^{g+1}-1\}$ of the group will firstly be final after $m^{g+1} - m^g$ time steps plus the final time t_{i-2^g} of the corresponding cell, which proves the claim.

In order to prove the theorem we have to consider the function f which maps $i \mapsto t_i$. From $t_0 = 0$ we derive $f(0) = 0$ and, clearly, $f(i+1) \geq f(i)$ holds for all $i > 0$. Let l be defined as $\lceil \log_2 n \rceil$, then the value $n' := 2^{\lceil \log_2 n \rceil} = 2^l$ is bounded by $n \leq n' \leq 2n$.

$$\begin{aligned}
f(n) &\leq f(n') \\
&= m^{\lfloor \log_2 n' \rfloor + 1} - m^{\lfloor \log_2 n' \rfloor} + f(n' - 2^{\lfloor \log_2 n' \rfloor}) \\
&= m^{l+1} - m^l + f(0) \\
&\leq m^{l+1} = (2^k)^{l+1} \\
&= (2^l)^k \cdot 2^k \\
&= (n')^k \cdot 2^k \\
&\leq (2n)^k \cdot 2^k \\
&= 2^{k+1} \cdot n^k
\end{aligned}$$

$$\begin{aligned}
f(n) &= m^{\lfloor \log_2 n \rfloor + 1} - m^{\lfloor \log_2 n \rfloor} + f(n - 2^{\lfloor \log_2 n \rfloor}) \\
&\geq m^{\lfloor \log_2 n \rfloor + 1} - m^{\lfloor \log_2 n \rfloor} \\
&= (2^k)^{\lfloor \log_2 n \rfloor + 1} - (2^k)^{\lfloor \log_2 n \rfloor} \\
&= (2^{\lfloor \log_2 n \rfloor + 1})^k - (2^{\lfloor \log_2 n \rfloor})^k \\
&= (2^{\lfloor \log_2 n \rfloor} \cdot 2)^k - (2^{\lfloor \log_2 n \rfloor})^k \\
&= (2^k - 1) \cdot (2^{\lfloor \log_2 n \rfloor})^k
\end{aligned}$$

$$\begin{aligned}
&\geq (2^k - 1)\cdot (2^{(\log_2 n)-1})^k \\
&= (2^k - 1)\cdot (\frac{n}{2})^k = \frac{2^k-1}{2^k}\cdot n^k \\
&= (1-\frac{1}{2^k})\cdot n^k
\end{aligned}$$

It follows $f \in \mathcal{O}(\mathrm{id}^k)$ and $f \in \Omega(\mathrm{id}^k)$ and, consequently $f \in \Theta(\mathrm{id}^k)$. □

Beyond the polynomials there are exponential functions.

Lemma 12 $\forall\, b \geq 2 \in \mathbb{N} : b^{\mathrm{id}} + \mathrm{id} \in \mathscr{C}(\mathrm{OCA})$

Proof. For the construction we can simply setup a $b$ary counter where the rightmost cell hosts the least significant bit. A cell is in a final state at the first time iff its behavior becomes periodic, i.e. if it changes its state in such a manner that a carry over occurs at its position.

Observe, that a carry over from the rightmost cell to the leftmost cell needs exactly n time steps to travel through the counter. Therefore, the leftmost cell enters a final state at time $b^n + n$ for the first time (not at time step b^n). □

4 Formal language characterization of $\mathscr{C}$(OCA)

Now we are going to characterize the family $\mathscr{C}$(OCA) in terms of a sub-family of the real-time OCA languages. The main result of this section becomes important in so far as it can not be adapted to two-way cellular automata [2].

Definition 13

a) A word $w \in A^+$ is accepted by an OCA $M = (S, \sigma, \#)$ in $n_0 \in \mathbb{N}$ time steps $\iff A \subseteq S$ and there exists a set of final states $F \subseteq S$ with the property $(\forall\, t \in \mathbb{N} : \forall\, 1 \leq i \leq n : c_t(i) \in F \Longrightarrow c_{t+1}(i) \in F)$ and $\pi_1(\mathcal{T}^{n_0}(w)) \in F$.

b) A formal language $L \subseteq A^+$ is accepted by an OCA M with time complexity $t : \mathbb{N} \longrightarrow \mathbb{N} \iff L = \{w \mid w \text{ is accepted by } M \text{ in } t(|w|) \text{ time steps}\}$.

The family of all languages which can be accepted by an OCA in real-time (i.e. $t = \mathrm{id}$) is denoted by $\mathscr{L}_{rt}(\mathrm{OCA})$. The real-time OCA languages are well investigated. E.g. it is known that they are properly contained in the real-time CA languages, that they are closed under boolean operations [4] and reversal

[3] but are not closed under concatenation [13]. All real-time OCA languages over an one-letter alphabet are known to be regular [11].

Let $w = w_1 \cdots w_n$ be a word over an arbitrary alphabet, then

$$P(w) := \bigcup_{i=1}^{n} w_1 \cdots w_i$$

is the set of prefixes of w.

Let $K \subseteq \mathbb{N}$ be a subset of the natural numbers such that the language $\{\mathtt{a}^k \mid k \in K\}$ is regular. For a given formal language L we define the K-prefix language of L as

$$L_K := \{w \mid |P(w) \cap L| \in K\}$$

Let for an arbitrary natural number $k \in \mathbb{N}_0$ the set K be $\{k\}$ resp. $\{k, k+1, k+2, \ldots\}$. For convenience we denote the corresponding languages L_K by $L_{\overline{k}}$ resp. $L_{\overline{k}}^{>}$. (e.g. $L_{\overline{0}}$ is a subset of the complement of L, $L_{\overline{1}}^{>}$ is a superset of L and $L_{\overline{1}} \cap L$ consists of words from L where all other prefixes are not belonging to L).

On a first glance it seems to be harder to recognize L_K than L but for real-time OCA languages it is not.

Lemma 14 Every K-prefix language L_K of a real-time OCA language L belongs to $\mathscr{L}_{rt}(\mathrm{OCA})$.

Proof. Let $w = w_1 \cdots w_n$ be a word from an $L \in \mathscr{L}_{rt}(\mathrm{OCA})$. We are considering the computation of a real-time OCA-recognizer $M = (S, \sigma, \#)$ for L on all prefixes of w and denote the configurations at time t on input $w_1 \cdots w_j$, $j \leq n$, by c_t^j.

At first we give the following construction of an OCA: $M' = (S \times S, \sigma', (\#, \#))$, where $c'_0(i) := (w_i, \#)$
and $c'_{t+1}(i) = \Big(\sigma\Big(\pi_1(c'_t(i)), \pi_1(c'_t(i+1))\Big), \sigma\Big(\pi_1(c'_t(i)), \pi_2(c'_t(i+1))\Big)\Big)$.

We claim that for all time steps $1 \leq t \leq n$ and all cells $1 \leq i \leq n+1-t$: $c'_t(i) = (c_t^n(i), c_t^{i+t-1}(i))$.

From the definition of c'_0 and σ' it follows $c'_1(i) = (c_1^n(i), c_1^i(i))$ for $i \leq n$, which meets the claim.

Now, to prove the claim we have to show $\sigma(c_t^n(i), c_t^{i+t}(i+1)) = c_{t+1}^{i+t}(i)$.

Since the only difference in two computations, say on $w_1 \cdots w_j$ and $w_1 \cdots w_{j+1}$, is due to the influence of the right border state on the diagonal, one can observe that $c_t^j(i) = c_t^n(i)$ for $j \leq n$ and $1 \leq i \leq j - t$.

Especially, the observation holds for $i + t \leq j$ such that we can conclude $\sigma(c_t^n(i), c_t^{i+t}(i+1)) = \sigma(c_t^{i+t}(i), c_t^{i+t}(i+1)) = c_{t+1}^{i+t}(i)$ and, hence $c'_{t+1}(i) = \left(c_{t+1}^n(i), c_{t+1}^{i+t}(i)\right)$ which proves the claim.

From the claim it follows that the second components of the sequence of the states the first cell passed through are $\{c_t^t(1)\}_{1 \leq t \leq n}$. These are the states in which the first cell would be after real-time on input of prefix $w_1 \cdots w_t$. To prove the lemma the OCA can easily be extended in such a manner that the first cell simulates one step of a finite-state acceptor for the language $\{\mathtt{a}^k \mid k \in K\}$ every time it recognizes an accepted prefix. □

Lemma 15 Let $f : \mathbb{N} \longrightarrow \mathbb{N}_0$ be a function, $k \in \mathbb{N}$ and $L := \{\mathtt{a}^n\mathtt{b}^{f(n)} \mid n \in \mathbb{N}\} \in \mathscr{L}_{rt}(\text{OCA})$. Then $L' := \{\mathtt{a}^n\mathtt{b}^{f(n+k)} \mid n \in \mathbb{N}\}$ belongs to $\mathscr{L}_{rt}(\text{OCA})$, too.

Proof. From a real-time acceptor for L we derive one for L' as follows:
The rightmost a-cell can identify itself. It simply simulates besides its own work another k virtual a-cells which would be located to its right. Altogether L' simulates the computation of L on input $\mathtt{a}^{n+k}\mathtt{b}^{f(n+k)}$. To achieve real-time on input $\mathtt{a}^n\mathtt{b}^{f(n+k)}$ L' must be faster for k time steps. E.g. the speed-up is done by a maximum speed leftward signal, which is generated at the right border at time step 1. When this signal arrives at the rightmost a-cell the cell simulates $k+1$ steps of its own work (with help of the virtual cells) at once and stores the results in k additional registers. Afterwards its left neighbor recognizes that fact and simulates also $k+1$ steps to get synchronized again and so on to the first cell. □

Lemma 16 Let $f : \mathbb{N} \longrightarrow \mathbb{N}_0$ be a function, $k \in \mathbb{N}$ and $L := \{\mathtt{a}^n\mathtt{b}^{f(n)} \mid n \in \mathbb{N}\} \in \mathscr{L}_{rt}(\text{OCA})$. Then $L' := \{\mathtt{a}^{n+k}\mathtt{b}^{f(n)} \mid n \in \mathbb{N}\}$ belongs to $\mathscr{L}_{rt}(\text{OCA})$, too.

Proof. The proof is similar to the previous one, except that the k rightmost a-cells of L' do not simulate anything but transmit the state of their right neighbor to their left neighbor respectively. □

Now we shed light on the relationship between OCA time computability and OCA real-time recognizability.

Theorem 17 Let $f : \mathbb{N} \longrightarrow \mathbb{N}$ be a function with $f \geq \mathrm{id}$.
Then $f \in \mathscr{C}(\mathrm{OCA}) \iff \{\mathtt{a}^n \mathtt{b}^{f(n)-n} \mid n \in \mathbb{N}\} \in \mathscr{L}_{rt}(\mathrm{OCA})$ holds.

Proof. Let L denote the language $\{\mathtt{a}^n \mathtt{b}^{f(n)-n} \mid n \in \mathbb{N}\}$ for a function $f \geq \mathrm{id}$ and $f \in \mathscr{C}(\mathrm{OCA})$.

"$\Longrightarrow$" On an input w of the form $\mathtt{a}^+\mathtt{b}^+$ a corresponding real-time acceptor for L works as follows.
The $\mathtt{a}$-cells simply simulate the time-computation of f.
At time step 1 the rightmost $\mathtt{b}$-cell (which can identify itself) sends a maximum speed signal to the left. The input is accepted iff that signal arrives in the leftmost cell exactly at the time step the time-computation in that cell becomes final.

Since the running time of the signal is $n + m$ time steps, the input would be accepted if $n + m = f(n) \Longrightarrow m = f(n) - n$ holds.

The leftward signal can easily be extended such that it recognizes whether the input meets the form $\mathtt{a}^+\mathtt{b}^+$. Otherwise an error signal is propagated to the left.

"$\Longleftarrow$" Let us assume that there is a real-time acceptor for L. Due to lemma 14 then there exists a real-time acceptor for the 1-prefix-language $L_1^{\geq}$ of L, say M, from which we construct an OCA M' which time-computes f.

All the cells of M' simulate the $\mathtt{a}$-cells of M.

Suppose the rightmost cell additionally to its own work simulates another (virtual) cell which would be located to its right. Since the rightmost cell can identify itself this behavior can be achieved in another register.

That (virtual) cell simulates the behavior of the leftmost $\mathtt{b}$-cell of M under the assumption there are infinitely many $\mathtt{b}$-cells. To achieve that behavior we can simply define $c'_0(v) = \mathtt{b}$ and $\sigma'(c'_t(v), \#)$ to be $\sigma(c'_t(v), c'_t(v))$, where v is the virtual cell. M' becomes final exactly at the time step M would have recognized the first prefix.

Obviously, $\mathtt{a}^n\mathtt{b}^m$ is in L iff $m = f(n) - n$ holds. This means that M' will be final after $n + m = n + f(n) - n = f(n)$ time steps, from which the time computability follows. $\square$

Corollary 18 Let $L \in \mathscr{L}_{rt}(\mathrm{OCA})$ be a language with the property that for all $n \in \mathbb{N}$ there exists an $m \in \mathbb{N}_0$ such that $\mathtt{a}^n\mathtt{b}^m \in L$ holds. Then the function $F + \mathrm{id}$ belongs to $\mathscr{C}(\mathrm{OCA})$, where $F : \mathbb{N} \longrightarrow \mathbb{N}_0$ with $F(n) := \min\{m \in \mathbb{N}_0 \mid \mathtt{a}^n\mathtt{b}^m \in L\}$.

Proof. The required property ensures that F, and hence $F + \mathrm{id}$, is totally defined. Since $L \in \mathscr{L}_{rt}(\mathrm{OCA})$ and $L' := \{\mathtt{a}^n\mathtt{b}^m \mid n, m \in \mathbb{N}\} \in \mathscr{L}_{rt}(\mathrm{OCA})$ and $\mathscr{L}_{rt}(\mathrm{OCA})$ is closed under intersection it follows that $L'' := (L \cap L')$ belongs to $\mathscr{L}_{rt}(\mathrm{OCA})$. From lemma 14 it follows that $L_1''^{=}$ belongs to $\mathscr{L}_{rt}(\mathrm{OCA})$ and $L_1''^{=} \cap L$ does, too. This implies that for the words $\mathtt{a}^n\mathtt{b}^m$ from $L_1''^{=} \cap L$ the number m is minimal with respect to n. Otherwise there would be another prefix of the form $\mathtt{a}^+\mathtt{b}^+$ belonging to L and, hence, the word could not belong to $L_1''^{=} \cap L$.

Then the proof follows with theorem 17. □

5 The range of OCA-time-computable functions

In the present section we give some results that characterize the range of the family $\mathscr{C}(\mathrm{OCA})$.

Due to the following theorem all OCA-time-computable functions below id are ultimately constant. The converse, namely all constant functions are OCA-time-computable, has been shown in lemma 6.

Theorem 19 Let $f \in \mathscr{C}(\mathrm{OCA})$ and suppose there is an $m \in \mathbb{N}$ for which $f(m) < m$ holds. Then there exist $k \in \mathbb{N}$ and $n_0 \in \mathbb{N}$ such that for all $n \geq n_0$ it holds $f(n) = k$.

Proof. Let f be an OCA-time-computable function for which $f(m) < m$ and let $M = (S, \sigma, \mathtt{\#})$ be a corresponding OCA. Define $k := f(m)$ and $n_0 := m$. To prove the theorem it suffices to show that $\forall\, n > m : \pi_1(\mathcal{T}^k(s_0^n)) \in F$.

Consider the space-time-diagram $\{\mathcal{T}^t(s_0^m)\}_{t\in\mathbb{N}}$ of the computation of M on s_0^m which is depicted in figure 6.

Due to our assumptions s_k must be a final state. Since the OCA is a deterministic device for all $n > m$ the space-time-diagram is as depicted in figure 7.

Because s_k is final we can conclude $f(n) = k$ and, hence, the theorem follows. □

As we have seen in lemma 7 the function id is OCA-time-computable. Consequently we have a gap below id. For example neither $\lfloor \log \rfloor$ nor $\lfloor \sqrt{} \rfloor$ are belonging to $\mathscr{C}(\mathrm{OCA})$. Now the question arises whether there is a similar gap beyond id.

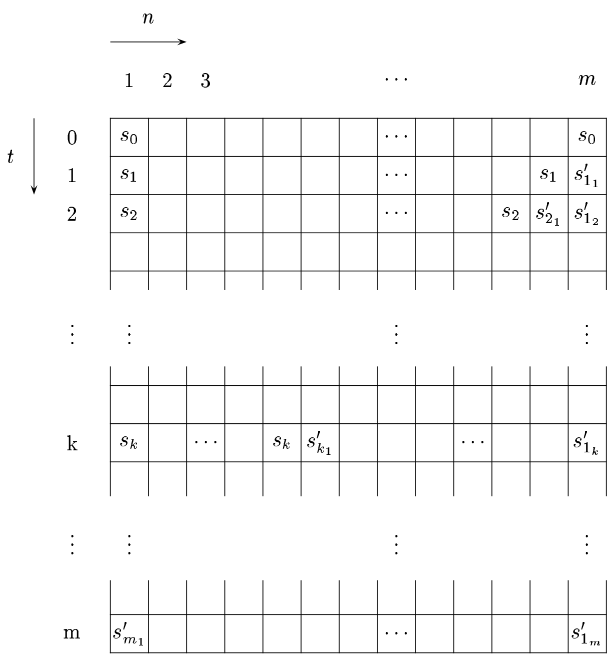

Figure 6: Space-time-diagram to theorem 19. The computation of M on s_0^m.

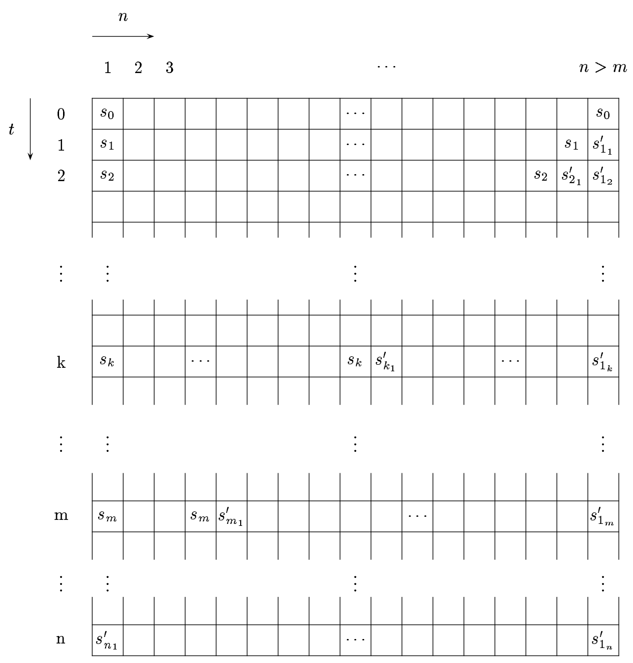
n
1 2 3 ⋯ $n > m$
t
0 1 2 k m n
s_0 ⋯ s_0
s_1 ⋯ s_1 s'_{1_1}
s_2 ⋯ s_2 s'_{2_1} s'_{1_2}
s_k ⋯ s_k s'_{k_1} ⋯ s'_{1_k}
s_m s_m s'_{m_1} ⋯ s'_{1_m}
s'_{n_1} ⋯ s'_{1_n}

Figure 7: Space-time-diagrams to theorem 19. The computation of M on $s_0^n, n > m$.

Lemma 20 Let $f \geq \mathrm{id}$ be an OCA-time-computable function which satisfies $\forall\, b \in \mathbb{N} : f \not\geq \mathrm{id} + \lfloor \log_b \rfloor$. Then there exists a $k \in \mathbb{N}_0$ such that $f(n) = n + k$ for infinitely many $n \in \mathbb{N}$.

Proof. Suppose $f \geq \mathrm{id}$ is OCA-time-computable by $M = (S, \sigma, \#)$ and satisfies $\forall\, b \in \mathbb{N} : f \not\geq \mathrm{id} + \lfloor \log_b \rfloor$. Then especially there exists an $n_0 \in \mathbb{N}$ such that $n_0 \leq f(n_0) < n_0 + \lfloor \log_{(|S|+1)^2}(n_0) \rfloor$. We conclude that $\lfloor \log_{(|S|+1)^2}(n_0) \rfloor \geq 1$ and, therefore, $n_0 > |S|^2$ and $n_0^{\frac{1}{2}} > |S|$ holds.

Assume for the rest of the proof that the cells are numbered from right to left in ascending order. Considering the computation of M we denote the evolution

$$c_{n-1}(n) c_n(n) c_{n+1}(n) \cdots c_{n+\lfloor \log_{|S|^2} n_0 \rfloor - 1}(n)$$

of a single cell n from time $n-1$ to $n + \lfloor \log_{|S|^2} n_0 \rfloor - 1$ by e_n. Observe, that the lengths of all the evolutions are identical (i.e.: $\lfloor \log_{|S|^2} n_0 \rfloor + 1$).

In total there are

$$\begin{aligned} & |S|^{\lfloor \log_{|S|^2} n_0 \rfloor + 1} \leq |S|^{1 + \log_{|S|^2} n_0} \\ = \; & |S|^{1 + \frac{1}{2} \cdot \log_{|S|} n_0} = |S|^{\frac{1}{2} \cdot \log_{|S|} n_0} \cdot |S| \\ = \; & |S| \cdot \left(|S|^{\log_{|S|} n_0} \right)^{\frac{1}{2}} \\ = \; & |S| \cdot n_0^{\frac{1}{2}} < n_0 \end{aligned}$$

different evolutions of M. Hence at least two evolutions from $e_1, \ldots, e_{n_0}$, say e_i and e_j $(i < j)$ are identical. Since M is a deterministic device and the initial input consists of identical states s_0 the evolution e_n determines the evolution e_{n+1} uniquely. Therefore $e_i = e_j$ implies $e_{n_0 - (j-i)} = e_{n_0}$ and inductively $\forall\, l \geq -1 : e_{n_0 + l(j-i)} = e_{n_0}$. Define $k := f(n_0) - n_0$. Since $e_{n_0 + l(j-i)} = e_{n_0}$ cell $n_0 + l(j-i)$ enters a final state at time $n_0 + l(j-i) + k$ for the first time which marks $f(n_0 + l(j-i))$. It follows $f(n_0 + l(j-i)) - (n_0 + l(j-i)) = k$ which proves the lemma. $\square$

The following theorem which follows immediately from the previous lemma shows that there is a gap beyond id, but it is in some sense smaller than that one below id. Consider for example $\mathrm{id} + \lfloor \log \rfloor$ which is OCA-time-computable and $\mathrm{id} + \lfloor \log \log \rfloor$ which is not.

Theorem 21 Let $f \geq \mathrm{id}$ be a function. If f belongs to $\mathscr{C}(\mathrm{OCA})$ then either $f \geq \mathrm{id} + \lfloor \log_b \rfloor$ for some $b \in \mathbb{N}$ or there exists a $k \in \mathbb{N}_0$ such that $f(n) = n + k$ for infinitely many $n \in \mathbb{N}$.

In the previous theorem our assumptions on f are weak. If we require additional properties we can prove stronger results:

Theorem 22 Let $f \geq \mathrm{id}$ be an *increasing* function. If f belongs to $\mathscr{C}(\mathrm{OCA})$ then either $f \geq \mathrm{id} + \lfloor \log_b \rfloor$ for some $b \in \mathbb{N}$ or $\{f(n) - n\}_{n\in\mathbb{N}}$ is ultimately periodic.

Proof. In the proof of lemma 20 it has been shown that $\forall\, l \geq -1 : f(n_0+l(j-i)) = n_0+l(j-i)+k$. Since f is increasing all the values $f(n_0-(j-i)), \dots, f(n_0)$ are between $n_0-(j-i)+k$ and n_0+k. Thus, all the cells $m \in \{n_0-(j-i), \dots, n_0\}$ are final at times $n_0 - (j-i) + k \leq t_m \leq n_0 + k$. Since $n_0 + k = f(n_0) < n_0 + \lfloor \log_{|S|^2}(n_0) \rfloor$ all the cells m enter a final state during their evolution e_m. The evolutions $\forall\, l \geq -1 : e_{n_0+l(j-i)}$ are identical to e_{n_0}. On the other hand e_{n+1} is totally determined by e_n. Therefore the evolution e_p of an arbitrary cell $p \geq j$ is identical to the evolutions $\forall\, l \geq -1 : e_{p+l(j-i)}$. It follows that the series of evolutions $\{e_p\}_{p\in\mathbb{N}}$ is periodic with period length $j-i-1$. Since every evolution e_p contains a final state the series $\{f(p)-p\}_{p\in\mathbb{N}}$ is periodic with period length $j-i-1$, too. □

The difference between the both previous results is illustrated by the following example.

Example 23 Let $f : \mathbb{N} \longrightarrow \mathbb{N}$ be defined as

$$f(n) := \begin{cases} 2^n + n & \text{if } n \text{ is even} \\ n & \text{otherwise} \end{cases}.$$

Since an OCA can recognize in real-time whether its input is of even length or not (see theorem 28, too) in combining lemma 7 and lemma 12 it can easily be seen that f is OCA-time-computable. Of course, f is not increasing. Therefore one can find a $k \in \mathbb{N}_0$ (i.e. $k = 0$) such that $f(n) - n = k$ for infinitely many $n \in \mathbb{N}$ (i.e. for all odd $n \in \mathbb{N}$) but $\{f(n) - n\}_{n\in\mathbb{N}}$ is not periodic since $\{f(2n) - 2n\}_{n\in\mathbb{N}} = \{2^{2n}\}_{n\in\mathbb{N}}$ is unbounded.

If we require f to be strictly increasing the theorem can be strengthen furthermore. In the context of signals a similar result was found in [10].

Theorem 24 Let $f \geq \mathrm{id}$ be an *strictly increasing* function. If f belongs to $\mathscr{C}(\mathrm{OCA})$ then either $f \geq \mathrm{id} + \lfloor \log_b \rfloor$ for some $b \in \mathbb{N}$ or $\{f(n) - n\}_{n\in\mathbb{N}}$ is ultimately constant.

Proof. From theorem 22 we know that $\{f(n) - n\}_{n \in \mathbb{N}}$ is ultimately periodic, say with period length p. Suppose the periodicity starts at cell $n_0 \in \mathbb{N}$ and $f(n_0) - n_0 = k$. For an arbitrary cell $n \geq n_0$ we have $f(n) - n = f(n+p) - (n+p) \Longrightarrow f(n+p) - f(n) = p$. Thus there are exactly p values from $f(n)$ to $f(n+p-1)$ for the p cells n to $n+p-1$. Since f is strictly increasing it follows $\forall\, n > n_0 : f(n) = f(n-1) + 1$. Thus $f(n) = f(n_0) + n - n_0 = k + n$ from which $\forall\, n \geq n_0 : f(n) - n = k$ follows. $\square$

Again, to illustrate the difference between the two previous results we give an example:

Example 25 Let $k \in \mathbb{N}$ be a constant and $f : \mathbb{N} \longrightarrow \mathbb{N}$ be defined as

$$f(n) := \begin{cases} n + k & \text{if } n \text{ is odd} \\ n + 1 + k & \text{otherwise} \end{cases} .$$

f belongs to $\mathscr{C}(\text{OCA})$. Since f is increasing but not strictly increasing the series $\{f(n) - n\}_{n \in \mathbb{N}}$ (i.e. $k, k+1, k, k+1, \ldots$) is periodic but not constant.

As we have seen the constant functions form a lower bound of the range of OCA-time-computable functions. As we will see in the following theorem the exponential functions form an upper bound.

Theorem 26 If f belongs to $\mathscr{C}(\text{OCA})$ then there exists a $k \in \mathbb{N}$ such that $\lim_{n \to \infty} \frac{f(n)}{k^n} = 0$.

Proof. It suffices to show that there exists a k such that the series $\{\frac{f(n)}{k^n}\}_{n \in \mathbb{N}}$ is bounded by a constant k', because $2k$ fulfills the requirements:

$$\frac{f(n)}{(2k)^n} = \frac{1}{2^n} \frac{f(n)}{k^n} \leq \frac{1}{2^n} k'$$

and $\lim_{n \to \infty} \frac{1}{2^n} k' = 0$.

Now we conclude indirectly. Suppose there is a function f which is OCA-time-computable by an OCA $M = (S, \sigma, \#)$ and for all $k \in \mathbb{N}$ the series $\{\frac{f(n)}{k^n}\}_{n \in \mathbb{N}}$ is unbounded. Then especially there exists an $n_0 \in \mathbb{N}$ such that $f(n_0) > |S|^{n_0}$. On input $s_0^{n_0}$ the computation of M will be cyclically at most after $|S|^{n_0}$ time steps at the latest (there are at most $|S|^{n_0}$ different configurations of length n_0). But because $f(n_0) > |S|^{n_0}$ the leftmost cell can never enter a final state, which leads to a contradiction to the OCA time computability of f. $\square$

Example 27 $f(n) = 2^{2^n}$ is not OCA-time-computable since the series $\{\frac{2^{2^n}}{k^n}\}_{n \in \mathbb{N}}$ is unbounded for all $k \in \mathbb{N}$.

6 Closure properties

Besides closure properties are interesting of its own they are a powerful tool for proving time computability and properties concerned with.

Our first result in the present section is a little bit unusual closure property: It is the closure under regular selection. Theorem 28 ensures the OCA time computability of the functions in example 23 and 25 and generalizes their structure.

Theorem 28 Let $\{i_j\}_{j\in\mathbb{N}}$ be an ultimately periodic sequence of natural numbers and let $\max\{i_j \mid j \in \mathbb{N}\} = n_0$. If for $1 \leq l \leq n_0$ there are OCA-time-computable functions $f_l \geq \text{id}$, then $f : \mathbb{N} \longrightarrow \mathbb{N}$, $n \mapsto f_{i_n}(n)$ is OCA-time-computable.

Proof. Due to the fact that the sequence $\{i_j\}_{j\in\mathbb{N}}$ is ultimately periodic it must be bounded. Therefore the maximum always exists.

An OCA which time-computes f is constructed as follows:
At time step 1 a leftward signal with speed 1 is generated in the rightmost cell. It is realized with states $\{1, 2, \dots, n_0\}$ such that on its way it runs exactly through the sequence $\{i_j\}_{j\in\mathbb{N}}$. Therefore each cell n knows in real-time to which value i_n it corresponds.

In parallel each cell time-computes all of the finite number of functions f_l, $1 \leq l \leq n_0$ in separate registers. Since all functions are greater or equal to the identity the cells are able to select the required computation after receipt of the leftward signal. □

Speeding up or slowing down the time-computation by a certain well specified amount of time may be expressed in terms of closure of $\mathscr{C}(\text{OCA})$ under the operations addition/multiplication and subtraction/division.

The following theorems show that $\mathscr{C}(\text{OCA})$ is in some sense closed under subtraction resp. division by a constant but not by arbitrary functions (from $\mathscr{C}(\text{OCA})$), respectively.

Theorem 29 Let $f + \text{id}$ be OCA-time-computable then $\forall\, k \in \mathbb{N} : \max\{f - k, 0\} + \text{id} \in \mathscr{C}(\text{OCA})$.

Proof. An OCA M' for $\max\{f - \text{k}, 0\} + \text{id}$ is constructed from an OCA M for $f + \text{id}$ as follows:

The rightmost cell of M' knows the state of its right neighbor in advance (i.e. the boundary state #). Therefore, it can simulate the first $k+1$ time steps of the rightmost cell of M at the first time step and store the results in $k+1$ registers. Subsequently it simulates one transition of M at every time step and stores that result and the previous k results in its registers. At the second time step cell 2 behaves as cell 1 and so on. Thus, at the nth time step cell n has completed the $(n+k)$th time step of M and will subsequently complete the $(n+k+1)$th time step and so on. □

Because of theorem 19 we have to ensure that the function in the previous theorem is at least the identity. On the other hand, we know that the constants are the only OCA-time-computable functions below id and, of course, we can speed up a constant by a lower constant.

Theorem 30 Let for a $k \in \mathbb{N}$ the function $f > k$ be constant, then $f - \mathrm{k}$ is OCA-time-computable.

Proof. $f - \mathrm{k}$ is itself a constant function which belongs to $\mathscr{C}(\mathrm{OCA})$ (cf. lemma 6). □

Theorem 31 $\mathscr{C}(\mathrm{OCA})$ is not closed under (arbitrary) subtraction.

Proof. Due to lemma 7 $f := \mathrm{id}$ belongs to $\mathscr{C}(\mathrm{OCA})$ and due to lemma 10 $g := 2 \cdot \mathrm{id} + \lfloor \log\log \rfloor$ does. But from theorem 22 we conclude that $g - f = \mathrm{id} + \lfloor \log\log \rfloor$ is not OCA-time-computable. □

The family $\mathscr{C}(\mathrm{OCA})$ is not closed under subtraction even if we assume $f = f' + \mathrm{id} \in \mathscr{C}(\mathrm{OCA})$ and $g = g' + \mathrm{id} \in \mathscr{C}(\mathrm{OCA})$ and require that $f' - g' + \mathrm{id}$ belongs to $\mathscr{C}(\mathrm{OCA})$. This fact follows from the fact that $2 \cdot \mathrm{id} + \lfloor \log\log \rfloor + \mathrm{id}$ (cf. corollary 35) and $2 \cdot \mathrm{id} + \mathrm{id}$ are OCA-time-computable but $\lfloor \log\log \rfloor + \mathrm{id}$ is not.

Theorem 32 Let $f + \mathrm{id}$ be OCA-time-computable then $\forall\, k \in \mathbb{N} : \mathrm{id} + \lfloor \frac{f}{k} \rfloor \in \mathscr{C}(\mathrm{OCA})$.

Proof. Ibarra and Palis [7] have shown that for all $f : \mathbb{N} \longrightarrow \mathbb{N}_0$ the family $\mathscr{L}_{n+f(n)}(\mathrm{OCA})$ is identical to the families $\mathscr{L}_{n+\lfloor \frac{f(n)}{k} \rfloor}(\mathrm{OCA}), k \geq 1$. Since the time-computation of a function f on s_0^n may be regarded as recognizing the word s_0^n in $f(n)$ time steps that proof can easily be adapted. □

After asking for closure under subtraction and division we are interested in closure under addition and multiplication or in other words in slowing down the computation.

Theorem 33 Let f be OCA-time-computable then $\forall\, k \in \mathbb{N} : k \cdot f \in \mathscr{C}(\mathrm{OCA})$.

Proof. One can construct an OCA M' which time computes $k \cdot f$ from an OCA M for f.

Each cell of M' has two registers. One of them is a counter which counts the time steps modulo k. The other simply simulates one step of M at each time step the counter has the value $k - 1$. □

The following result about closure under addition is stronger than that on multiplication.

Theorem 34 Let $d : \mathbb{N} \longrightarrow \mathbb{Z}$ be a ultimately periodic function such that $f(n) := \sum_{i=1}^{n} d(i) \geq 0$ for all $n \in \mathbb{N}$. Then it holds $g \geq \mathrm{id} \wedge g \in \mathscr{C}(\mathrm{OCA}) \Longrightarrow g + f \in \mathscr{C}(\mathrm{OCA})$.

Proof. Without loss of generality we may assume that d is periodic. In the case it is ultimately periodic for all $n \geq n_0$ in the following construction the finite first n_0 cells can specially be handled.

Assume now the period length of d is p. From the requirement $f(n) \geq 0$ it follows that the sum over one period $s := \sum_{i=1}^{p} d(i) = f(p)$ is greater than or equal to 0.

An OCA can easily be constructed in such a manner that it marks every pth of its cells (i.e. marks the beginning of every period).

Let M be an OCA for g and let s^- be defined as $\sum_{i=1}^{p} d^-(i)$, where $d^-(i) := \begin{cases} |d(i)| & \text{if } d(i) < 0 \\ 0 & \text{otherwise} \end{cases}$.

An OCA M' which time-computes $f + g$ has to perform two main tasks: Basically a simulation of M sped-up by s^- time steps and, in parallel, each cell has to delay its final time by a certain amount of time.

The basically simulation of M is obtained by theorem 29 (Observe that $g + f \geq \mathrm{id}$).

Additionally in the first task every pth cell (which can be marked as mentioned before) has to delay the previous sped-up simulation by s time steps. (Observe,

s is a previously known fixed natural number.) Generally, delaying the computation can be done by adding some r registers that are initially empty and work as a queue in a first-in-first-out manner. The states leaving the queue are r time steps old, but after the first r time steps one state is leaving at every time step.

Up to now every cell $j = k \cdot p + l$, where $k \geq 0$ and $1 \leq l \leq p$ time-computes $g(j) - s^- + k \cdot s$.

The second task is to delay the final time by a certain amount of time. Concretely every cell $j = k \cdot p + l$ has to wait another $s^- + \sum_{i=1}^{l} d(i)$ time steps before changing to a final state. Remember, d is periodic. Consequently the requested behavior can be realized by letting the cells count up to the required value. Moreover, from the periodicity of d it follows $f(p) = s$ and $f(k \cdot p) = k \cdot s$, $k \geq 1$, and $f(k \cdot p) + f(l)$, $1 \leq l \leq p$, is equal to $f(k \cdot p + l)$.

Altogether the single tasks can be combined such that the resulting time steps are superimposed. Every cell $j = k \cdot p + l$, where $k \geq 0$ and $1 \leq l \leq p$ then will be final at time $g(j) - s^- + k \cdot s + s^- + \sum_{i=1}^{l} d(i) = g(j) + k \cdot s + \sum_{i=1}^{l} d(i) = g(j) + f(k \cdot p) + f(l) = g(j) + f(j)$, which proves the theorem. □

Unfortunately, the converse of the previous theorem does not hold. Otherwise we would have had the closure under subtraction of id. Consider for example $d(n) := 1$, such that $f(n)$ is the identity. d is periodic and $f \geq 0$ holds. But although $\mathrm{id} + \lfloor \log\log \rfloor + f$ belongs to $\mathscr{C}(\mathrm{OCA})$ (cf. lemma 10) $\mathrm{id} + \lfloor \log\log \rfloor$ does not.

On the other hand the result implies the closure under addition of id.

Corollary 35 Let g be an OCA-time-computable function then $g + \mathrm{id} \in \mathscr{C}(\mathrm{OCA})$.

The theorem holds even if the function f itself is ultimately periodic.

Corollary 36 Let $f : \mathbb{N} \longrightarrow \mathbb{N}_0$ be an ultimately periodic function. If $g \geq \mathrm{id}$ is OCA-time-computable then $g + f \in \mathscr{C}(\mathrm{OCA})$.

Proof. Define $d(1) := f(1)$ and $\forall\, i > 1 : d(i) := f(i) - f(i-1)$. It can easily be seen that $f(n) := \sum_{i=1}^{n} d(i)$ for all $n \geq 1$. Since f is a function from $\mathbb{N}$ to $\mathbb{N}_0$ it holds $f \geq 0$ and, hence, $\sum_{i=1}^{n} d(i) \geq 0$. □

Example 37 Every constant function k is periodic. Therefore, $\mathscr{C}(\mathrm{OCA})$ is closed under addition of constants: $f \in \mathscr{C}(\mathrm{OCA}) \Longrightarrow \forall\, k \in \mathbb{N} : f + \mathrm{k} \in \mathscr{C}(\mathrm{OCA})$.

Theorem 38 $\mathscr{C}(\mathrm{OCA})$ is not closed under composition.

Proof. $2^{\mathrm{id}} + \mathrm{id}$ belongs to $\mathscr{C}(\mathrm{OCA})$ and it holds $2^{2^{\mathrm{id}}+\mathrm{id}} + 2^{\mathrm{id}} + \mathrm{id} > 2^{2^{\mathrm{id}}}$. Therefore the theorem follows with theorem 26 and example 27. $\square$

The next theorem is related to the closure of $\mathscr{L}_{rt}(\mathrm{OCA})$ under union and intersection. Let for two functions f and g and all $n \in \mathbb{N}$ $\max\{f, g\}$ be defined according to $n \mapsto \begin{cases} f(n) & \text{if } f(n) \geq g(n) \\ g(n) & \text{otherwise} \end{cases}$. The definition of $\min\{f, g\}$ is straightforward.

Theorem 39 Let f and g be OCA-time-computable functions then $\max\{f, g\} \in \mathscr{C}(\mathrm{OCA})$ and $\min\{f, g\} \in \mathscr{C}(\mathrm{OCA})$.

Proof. An OCA for $\max\{f, g\}$ ($\min\{f, g\}$) on two tracks separately simulates the time-computation of f and g. The cells change to a final state when both (the first) of the simulations are (is) final. $\square$

The following theorems show in some sense the closure of $\mathscr{C}(\mathrm{OCA})$ against inversion. They are a powerful tool for proving time computability.

Theorem 40 Let $g : \mathbb{N} \longrightarrow \mathbb{N}_0$ be a function with the property that for all $m \in \mathbb{N}$ there exists an $n \in \mathbb{N}$ such that $g(n) = m$ holds. If $g + \mathrm{id}$ is OCA-time-computable then $F + \mathrm{id} \in \mathscr{C}(\mathrm{OCA})$, where $F : \mathbb{N} \longrightarrow \mathbb{N}$ with $F(m) := \min\{n \in \mathbb{N} \mid g(n) = m\}$.

Proof. Due to theorem 17 from the time computability of $g + \mathrm{id}$ it follows $L := \{\mathtt{a}^n\mathtt{b}^{g(n)} \mid n \in \mathbb{N}\} \in \mathscr{L}_{rt}(\mathrm{OCA})$. In [3] it has been shown that $\mathscr{L}_{rt}(\mathrm{OCA})$ is closed under reversal. Therefore L^R belongs to $\mathscr{L}_{rt}(\mathrm{OCA})$. Due to the required property there exists for all $m \in \mathbb{N}$ an $n \in \mathbb{N}$ such that $\mathtt{b}^m\mathtt{a}^n \in L^R$. From corollary 18 it follows the time computability of $F + \mathrm{id}$, where $F : \mathbb{N} \longrightarrow \mathbb{N}$ with $F(m) := \min\{n \in \mathbb{N} \mid \mathtt{b}^m\mathtt{a}^n \in L^R\}$. From $\forall\, n \in \mathbb{N} : \mathtt{b}^m\mathtt{a}^n \in L^R \iff m = g(n)$ we conclude $F(m) = \min\{n \in \mathbb{N} \mid g(n) = m\}$. $\square$

The following example is an improvement of lemma 12.

Example 41 $\lfloor \log_b \rfloor + \mathrm{id}$ belongs to $\mathscr{C}(\mathrm{OCA})$. Therefore $b^{\mathrm{id}} + \mathrm{id}$ is OCA-time-computable, too.

In the previous theorem it was required that the function g is surjective (except for the value 0). Since, generally, the resulting function F has not to be surjective the theorem cannot be applied to its results again. Instead we have the following theorems for the inversion of the inverse.

Theorem 42 Let $f : \mathbb{N} \longrightarrow \mathbb{N}_0$ be a strictly increasing function. If $f + \mathrm{id}$ is OCA-time-computable then $F + \mathrm{id} \in \mathscr{C}(\mathrm{OCA})$, where $F : \mathbb{N} \longrightarrow \mathbb{N}$ with

$$F(m) := \begin{cases} \max\{n \in \mathbb{N} \mid f(n) \leq m\} & \text{if } m \geq f(1) \\ 0 & \text{otherwise} \end{cases} .$$

Proof. Since $f + \mathrm{id}$ is OCA-time-computable the language $L := \{\mathtt{a}^n\mathtt{b}^{f(n)} \mid n \in \mathbb{N}\}$ belongs to $\mathscr{L}_{rt}(\mathrm{OCA})$. From lemma 14 it follows that $L_1^= \in \mathscr{L}_{rt}(\mathrm{OCA})$. Due to lemma 15 $L' := \{\mathtt{a}^n\mathtt{b}^{f(n+1)} \mid n \in \mathbb{N}\}$ belongs to $\mathscr{L}_{rt}(\mathrm{OCA})$ and with lemma 14 $L_1'^{\geq}$ does. The family $\mathscr{L}_{rt}(\mathrm{OCA})$ is closed under set difference thus $L'' := L_1^= \setminus L_1'^{\geq}$ is a real-time OCA language. Because f is strictly increasing we have in other words $L'' = \{\mathtt{a}^n\mathtt{b}^m \mid n \in \mathbb{N} \wedge f(n) \leq m < f(n+1)\}$. Since $\mathscr{L}_{rt}(\mathrm{OCA})$ is closed under reversal the language $L''^R = \{\mathtt{b}^m\mathtt{a}^n \mid n \in \mathbb{N} \wedge f(n) \leq m < f(n+1)\}$ is real-time OCA-acceptable. To express the number of $\mathtt{a}$'s in terms of a function depending on m we have to be prepared to the cases where $m < f(1)$ since a function would not be defined on such arguments. But fortunately the corresponding set of words is finite and therefore is $L''' := \{\mathtt{b}^m\mathtt{a}^{F(m)} \mid m \in \mathbb{N}\}$ where

$$F(m) := \begin{cases} \max\{n \in \mathbb{N} \mid f(n) \leq m\} & \text{if } m \geq f(1) \\ 0 & \text{otherwise} \end{cases}$$

real-time OCA-acceptable.

From lemma 14 it follows that $F + \mathrm{id}$ is OCA-time-computable. □

The following example additionally gives insight in one of the problems concerned with the inversion of discrete functions. In most cases there are more than one inverse function depending on rounding or truncating the results. From lemma 12 and theorem 42 the lemma 9 can be improved.

Example 43 $b^{\mathrm{id}} + \mathrm{id}$ belongs to $\mathscr{C}(\mathrm{OCA})$. Therefore $\lfloor \log_b \rfloor + \mathrm{id}$ is OCA-time-computable, too.

In the following theorem a dual result yields to “rounded up” results.

Theorem 44 Let $f : \mathbb{N} \longrightarrow \mathbb{N}_0$ be a strictly increasing function. If $f + \mathrm{id}$ is OCA-time-computable then $F + \mathrm{id} \in \mathscr{C}(\mathrm{OCA})$, where $F : \mathbb{N} \longrightarrow \mathbb{N}$ with $F(m) := \min\{n \in \mathbb{N} \mid f(n) \geq m\}$.

Proof. Let $\bar{f} : \mathbb{N}_0 \longrightarrow \mathbb{N}_0$, where $\bar{f}(0) := 0$ and $\bar{f}(n) := f(n)$, $n \geq 1$ be an extension of f. Taking the denotations of the previous proof set $L' := \{\mathtt{a}^{n+1}\mathtt{b}^{f(n)} \mid n \in \mathbb{N}\}$ (cf. lemma 16) and $L'' := \Big((L_1'^{=} \setminus L') \setminus L_1^{=}\Big) \cup L$. In other words $L'' = \{\mathtt{a}^n\mathtt{b}^m \mid n \geq 2 \wedge \bar{f}(n-1) < m \leq \bar{f}(n)\}$ from which follows that $L''^R := \{\mathtt{b}^m\mathtt{a}^n \mid n \geq 2 \wedge \bar{f}(n-1) < m \leq \bar{f}(n)\}$ belongs to $\mathscr{L}_{rt}(\mathrm{OCA})$ and, hence, (observe that F is totally defined since f is strictly increasing) $L''' := \{\mathtt{b}^m\mathtt{a}^{F(m)} \mid m \in \mathbb{N}\}$ where $F(m) = \min\{n \in \mathbb{N} \mid f(n) \geq m\}$ belongs to $\mathscr{C}(\mathrm{OCA})$. $\square$

Example 45 $b^{\mathrm{id}} + \mathrm{id}$ belongs to $\mathscr{C}(\mathrm{OCA})$. Therefore $\lceil \log_b \rceil + \mathrm{id}$ is OCA-time-computable, too.

Example 46 From theorem 11 (and its proof) it follows that there exists an OCA-time-computable function $f \in \Theta(\mathrm{id}^2)$, which is strictly increasing. From theorem 34 one can obtain $f + \mathrm{id} \in \mathscr{C}(\mathrm{OCA})$ and therefore there exists an OCA-time-computable function in $\mathrm{id} + \Theta(\sqrt{\mathrm{id}})$.

Thus theorem 40 and on the other side theorem 42 and theorem 44 are inverting each other in some sense. The question after the fix-points is easily answered: One can show that the identity meets all the requirements and is a fix-point under the considered operations. On the other hand the identity can be shown to be the only fix-point.

References

[1] Balzer, R. M. *An 8-state minimal time solution to the firing squad synchronization problem.* Information and Control 10 (1967), 22–42.

[2] Buchholz, Th. and Kutrib, M. *Some relations between massively parallel models.* To appear.

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

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

[5] Fischer, P. C. *Generation of primes by a one-dimensional real-time iterative array.* Journal of the ACM 12 (1965), 388–394.

[6] Ibarra, O. H. and Jiang, T. *On one-way cellular arrays.* SIAM Journal on Computing 16 (1987), 1135–1154.

[7] Ibarra, O. H. and Palis, M. A. *Some results concerning linear iterative (systolic) arrays.* Journal of Parallel and Distributed Computing 2 (1985), 182–218.

[8] Kutrib, M. *On stack-augmented polyautomata.* Report 9501, Arbeitsgruppe Informatik, Universität Gießen, Gießen, 1995.

[9] Kutrib, M. and Richstein, J. *Real-time one-way pushdown cellular automata languages.* In Dassow, J., Rozenberg, G. and Salomaa, A. (eds.), *Developments in Language Theory II. At the Crossroads of Mathematics, Computer Science and Biology.* World Scientific Publishing, Singapore, 1996.

[10] Mazoyer, J. and Terrier, V. *Signals in one dimensional cellular automata.* Research Report RR 94-50, Ecole Normale Supérieure de Lyon, Lyon, 1994.

[11] 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.

[12] Terrier, V. *Signals in linear cellular automata.* Proc. Workshop on Cellular Automata, Centre of Scientific Computing, Espoo Finland, 1991.

[13] Terrier, V. *On real time one-way cellular array.* Theoretical Computer Science 141 (1995), 331–335.

[14] Vollmar, R. *Algorithmen in Zellularautomaten.* Teubner, Stuttgart, 1979.

[15] Waksman, A. *An optimum solution to the firing squad synchronization problem.* Information and Control 9 (1966), 66–78.