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

Some relations between massively parallel arrays

Thomas Buchholz Martin Kutrib

Report 9603 December 1996

JUSTUS-LIEBIG-
UNIVERSITÄT
GIESSEN

Some relations between massively parallel arrays

Thomas Buchholz and Martin Kutrib

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

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

December 1996

Abstract

Relations between various models for massively parallel computers are investigated. These are arrays of finite-state machines – eventually augmented by pushdown storage – operating synchronously. The architectures differ mainly in how the input is supplied and how the single nodes are interconnected. The comparisons are made in terms of their capabilities to time-construct and time-compute functions. That means given an constant input of length n a distinguished cell has to enter distinguished states after $f(1), \ldots, f(n)$ respectively $f(n)$ time steps.

1 Introduction

A common distinguished feature (if not the substantial one) of massively parallel computers is their large number of processing elements. On the other hand, they differ heavily in e.g. how the single nodes are interconnected and in how the input is supplied. In order to compare the capabilities of various architectures in an analytical way one has to abstract from the physical conditions and go over to model the real devices. A widely accepted model for a single processing element is a finite-state machine [1, 4, 6]. Here we restrict our considerations to (one-dimensional) arrays of processing elements which operate synchronously by the dictate of a global clock. The interconnection is homogeneous which may be expressed by a so-called interconnection pattern: If the nodes are identified by integers, then in case of one-way information flow each node n is connected to the node $n+1$ and in case of two-way information flow each node is connected to the nodes $n+1$ and $n-1$ (i.e. its (one or two) immediate neighbors only).

Another question of layout is how the input gets to the nodes. In terms of concept there are two extreme input modes to which we are restricting. The parallel input mode (i.e. each node gets its own input during initialization) and the sequential input mode (i.e. a distinguished node gets the whole input successively). Other input modes are investigated e.g. in [10, 14].

Massively parallel programming is often done under heavily utilization of the concept of signals. They provide a powerful tool to encode and propagate information through the network [12]. The time computability [2] in a network – in some sense a generalization of signals – can be regarded as a higher programming concept which allows modularization techniques at algorithm design. It may be used for example to realize stable configurations or self terminating programs. Moreover, the capabilities to time-compute functions can show us the computation power and the limitations of a network.

The object of the present paper is to compare the different models, including two with pushdown storage augmented processing elements, in terms of their capabilities of time-computation and time-construction as well as formal language processing. Several relationships between different capabilities are established.

2 The models

Linear arrays of finite-state machines (eventually augmented by pushdown storage) are understood as models for massively parallel computers. For our convenience we identify the single nodes, sometimes called cells, by natural numbers. The number of nodes in a network depends on the size of the input.

The state transition function is applied to all cells synchronously at discrete time steps. It depends on the state of the cell itself and on the states of the nodes the cell is connected to, sometimes called its neighbors, and eventually on inputs.

In cases where pushdown storage is available each node is additionally allowed to access its top of stack item. The built-in operations are as usual `top`, `pop` and `push`.

2.1 Interconnection structure and input mode

Mainly, we distinguish two different interconnection patterns that are related to one-way and two-way information flow through the network. In case of one-way information flow each cell is connected to its right neighbor only (i.e. it receives the actual state of its right neighbor at every time step), thus transmission is from right to left. In case of two-way information flow each cell is connected to its both immediate neighbors.

For simplicity we assume that the border node(s) are initialized especially such that their states indicate their distinguished position.

Mainly, we distinguish two different input modes, the parallel and the sequential one.

Under parallel input mode all cells fetch their whole input at initialization (observe, that the number of nodes depends on the input size), thus it is reflected by their states at time step 0. Moreover, the state transition does not depend on inputs. Such arrays are commonly called *cellular arrays* or *cellular automata* (CA for short).

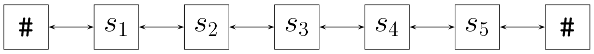

Figure 1: A cellular array.

For one-way arrays we use the abbreviation OCA.

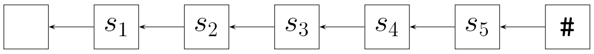

Figure 2: A one-way cellular array.

In case of sequential input mode the whole input to the network is supplied successively to a distinguished cell. For several reasons we use the left border cell as input cell for two-way arrays and the right border cell for one-way arrays. In both cases the states of the nodes are initially set to a special state, the so-called quiescent state. The state transition of the input cells depends besides on the actual states of the neighborhood additionally on input items, such that one of them is consumed at every time step. At the end of the input we assume there is a special item which is never consumed. Such arrays are commonly called *iterative arrays* (IA and OIA for short).

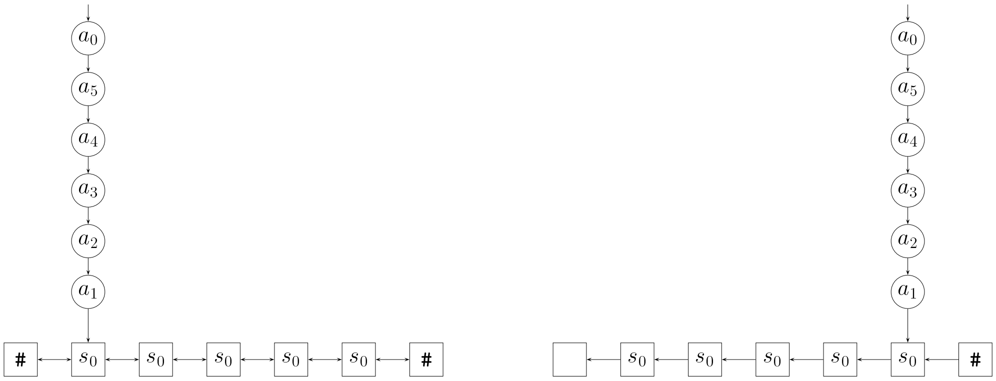

Figure 3: Two-way and one-way iterative arrays.

If there is pushdown storage available we call the corresponding machines *push-down cellular arrays* (PDCA, OPDCA).

In the sequel we always assume to have a network of n nodes if the input consists of n items.

More formally a cellular array is a system $(S, \sigma, \#)$ where S is the finite nonempty set of states (of the cells) and $\sigma : S^3 \longrightarrow S$ is the local state transition function which ensures that a cell is in the boundary state # at time step t iff it is at time step $t+1$.

The local transition function induces a length preserving mapping $\mathcal{T} : S^+ \longrightarrow S^+$ according to: $\forall\, n \in \mathbb{N}, i \in \{1, \ldots, n\} : \forall\, s_i \in S$:

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

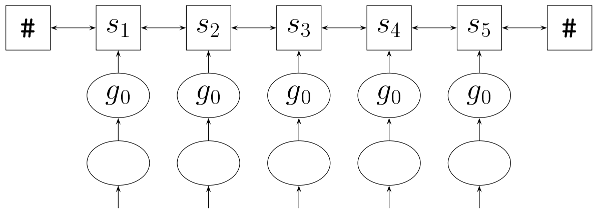

Figure 4: A two-way pushdown cellular array.

In case of one-way information flow we have $\sigma : S^2 \longrightarrow S$ and

$$\begin{aligned} \mathcal{T}(s_1) &:= \sigma(s_1, \#) \\ \mathcal{T}(s_1 \cdots s_n) &:= \sigma(s_1, s_2) \cdots \sigma(s_n, \#) \end{aligned}$$

In iterative arrays we have to be prepared with inputs.
The system $(S, A, \sigma, \#, s_0, a_0)$ is an iterative array if S is as for CA, A is the finite nonempty set of input symbols containing a_0 the end-of-input symbol. The local transition function maps from $S^3 \cup (S^3 \times A)$ (depending on whether the cell fetches the input or not) to S. It satisfies $\sigma(s_l, s_0, s_r) = s_0$ iff s_l and s_r belong to $\{s_0, \#\}$. Due to this property s_0 is called the quiescent state in which all cells are at time step 0.

We assume that at the end of input the symbol a_0 appears infinitely often. The global transition function now maps from A^+S^+ to A^+S^+ as follows:
For $n, m > 0, i \in \{1, \ldots, n\}, j \in \{1, \ldots, m\} : \forall\, a_i \in A, s_j \in S :$

$$\begin{aligned} \mathcal{T}(a_m \cdots a_1, s_1) &:= (a_m \cdots a_2, \sigma(\#, s_1, \#, a_1)) \\ \mathcal{T}(a_m \cdots a_1, s_1 \cdots s_n) &:= (a_m \cdots a_2, \sigma(\#, s_1, s_2, a_1) \cdots \sigma(s_{n-1}, s_n, \#)) \end{aligned}$$

In case of one-way iterative arrays we have $\sigma : S^2 \cup (S^2 \times A) \longrightarrow S$ and

$$\begin{aligned} \mathcal{T}(a_m \cdots a_1, s_1) &:= (a_m \cdots a_2, \sigma(s_1, \#, a_1)) \\ \mathcal{T}(a_m \cdots a_1, s_1 \cdots s_n) &:= (a_m \cdots a_2, \sigma(s_1, s_2) \cdots \sigma(s_n, \#, a_1)) \end{aligned}$$

Observe, in iterative arrays all cells are initially quiescent.

If the single nodes are pushdown storage augmented the model PDCA is a system $(S, \Gamma, \sigma, \#, g_0)$, where additionally to CA Γ is the finite, nonempty set of stack symbols containing g_0 the bottom-of-stack symbol. σ maps from $S^3 \times \Gamma$ to $S \times \Gamma^*$ from which follows that the symbol at the top of the stack is erased at reading (but may be pushed again). σ has to ensure that g_0 appears at each cell exactly once (i.e. at the bottom of its stack).

The definitions of OPDCA are straightforward and omitted here, but may be found in [8].

2.2 Computations

The following sections are devoted to the investigation and comparison of three types of computations in the previously described models.

A function f which maps the natural numbers to the natural numbers is said to be time-computed by some device if the input of n identical items leads to a computation such that the configuration at time step $f(n)$ is distinguished. The distinction is done by means of a set of final states into which the leftmost cell of the network has to switch. The input item is denoted by q. $\pi_i(s_1 \cdots s_n) := s_i$ selects the ith component of $s_1 \cdots s_n$ and $\pi_{i,j}$ is an abbreviation for $\pi_i(\pi_j)$.

Definition 1 Let POLY be an i) (O)CA, ii) (O)IA, iii) (O)PDCA.
A function $f : \mathbb{N} \longrightarrow \mathbb{N}$ is *POLY-time-computable* iff there exists a POLY with state set S, a distinguished state $q \in S$ or input symbol $q \in A$ and a set of final states $F \subseteq S \setminus \{s_0\}$ such that $f(n)$ is the smallest natural number for which
i) $\pi_1(\mathcal{T}^{f(n)}(q^n)) \in F$, ii) $\pi_1(\mathcal{T}^{f(n)}(q^n, s_0^n)) \in F$, iii) $\pi_{1,1}(\mathcal{T}^{f(n)}((q, g_0)^n)) \in F$.

We denote the family of POLY-time-computable functions by $\mathscr{C}(\text{POLY})$.

For time-computation a network has to compute the time step which corresponds to the value of the function on the length of input. Another computation, the time constructibility of functions, requires the network to compute all values of the function up to the input length. For that we need to restrict to strictly increasing functions.

Definition 2 Let POLY be an i) (O)CA, ii) (O)IA, iii) (O)PDCA.
A strictly increasing function $f : \mathbb{N} \longrightarrow \mathbb{N}$ is *POLY-time-constructible* iff there exists a POLY with state set S and a set of final states $F \subseteq S$ such that for all $t \in \{1, \ldots, f(n)\}$:

$$\begin{array}{rl} \text{i)} & \pi_1(\mathcal{T}^t(q^n)) \in F \iff t \in \bigcup_{i=1}^n \{f(i)\} \\ \text{ii)} & \pi_1(\mathcal{T}^t(q^n, s_0^n)) \in F \iff t \in \bigcup_{i=1}^n \{f(i)\} \\ \text{iii)} & \pi_{1,1}(\mathcal{T}^t((q, g_0)^n)) \in F \iff t \in \bigcup_{i=1}^n \{f(i)\} \end{array}$$

The family of POLY-time-constructible functions is denoted by $\mathscr{F}(\text{POLY})$.

Our third type of computation is the most classical one when comparisons between different models are made, it is formal language processing. Although in principle the family of languages acceptable by a specific network may be considered of its own it is usual to bound some resource e.g. the time.

Definition 3 Let A be an alphabet and POLY be an i) (O)CA, ii) (O)IA, iii) (O)PDCA with state set S.

a) A word $w = w_1 \cdots w_n \in A^+$ is accepted by POLY 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 that a cell once entered a final state remains in a final state and

$$\begin{array}{rl} \text{i)} & \pi_1(\mathcal{T}^{n_0}(w_1 \cdots w_n)) \in F \\ \text{ii)} & \pi_1(\mathcal{T}^{n_0}(w_1 \cdots w_n, s_0^n)) \in F \\ \text{iii)} & \pi_{1,1}(\mathcal{T}^{n_0}((w_1, g_0) \cdots (w_n, g_0))) \in F \end{array}$$

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

The family of all languages which can be accepted by POLY with time complexity $t(n)$ is denoted by $\mathscr{L}_{t(n)}(\text{POLY})$. In this connection the identity id is denoted real-time and aliased to rt.

3 Time computability

Our first results in this section establish relationships between function time-computation capabilities under parallel and sequential input mode.

3.1 One-way devices

What are the differences between OCA and OIA? Conceptually there are two ones: The cells in OIA are initially quiescent. Therefore each node i has to be activated (at time step $n-i+1$ at the earliest), whereas in OCA all nodes are activated at time step 1.

On the other hand, in OIA the rightmost cell gets an input, which leads to the advantage that it can recognize the time step n (i.e. the length of the input). Since the single input symbols are always identical (i.e. the symbol q) it is the only advantage.

The following lemma is concerned with the question what would happen if the nodes of an OCA were initially quiescent.

Lemma 4 Let $f \geq id$ be an OCA-time-computable function, then it can be time-computed by an OCA where all cells except the rightmost one are initially quiescent.

Proof. Assume M is an OCA which time computes f and let n be large enough. Consider the state transitions of the leftmost cell. It starts in state q. Subsequently, it gets its own state (i.e. q) as input from its right neighbor if $n > 2$. Subsequently, it gets its own state (i.e. $\sigma(q,q)$) as input from its right neighbor if $n > 3$ and so on.

Since the nodes are finite-state machines its behavior will become periodic if $n > |S|$, say at time step p.

Let us have a look on the space-time-diagram of an activation where the states $r_i, i \geq 1$, appear on the diagonal (cf. figure 5).

Now an OCA with the required initialization which simulates M can be constructed as follows:

The nodes which are placed at the diagonal of the space-time diagram simulate the computation which takes place on the diagonal of M and additionally (in an additional register) the computation of the leftmost cell of M. Since they are on the diagonal they are active already (cf. figure 6). The computation at the right of the diagonal and after time step n can be simulated directly.

Since $f \geq id$ the lemma follows. $\square$

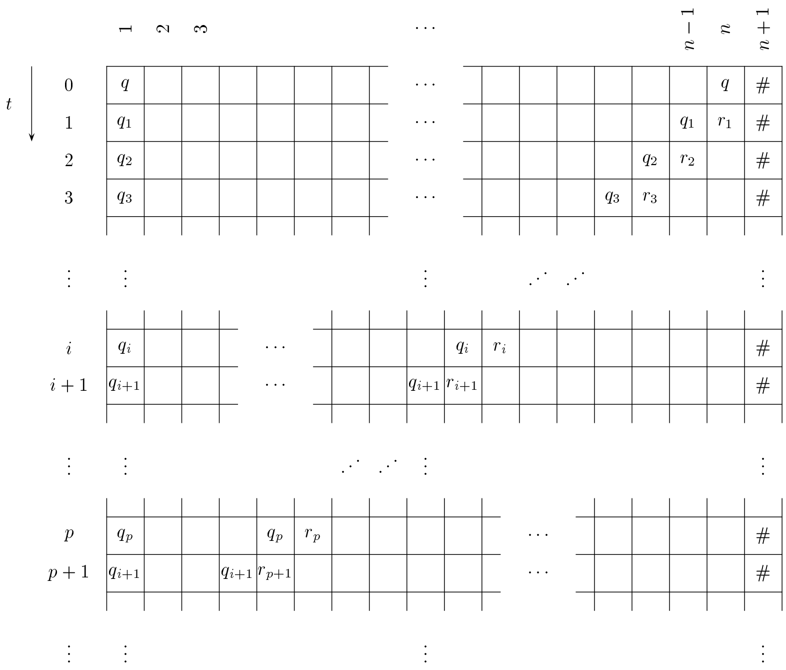

Figure 5: Space-time-diagram of an activation.

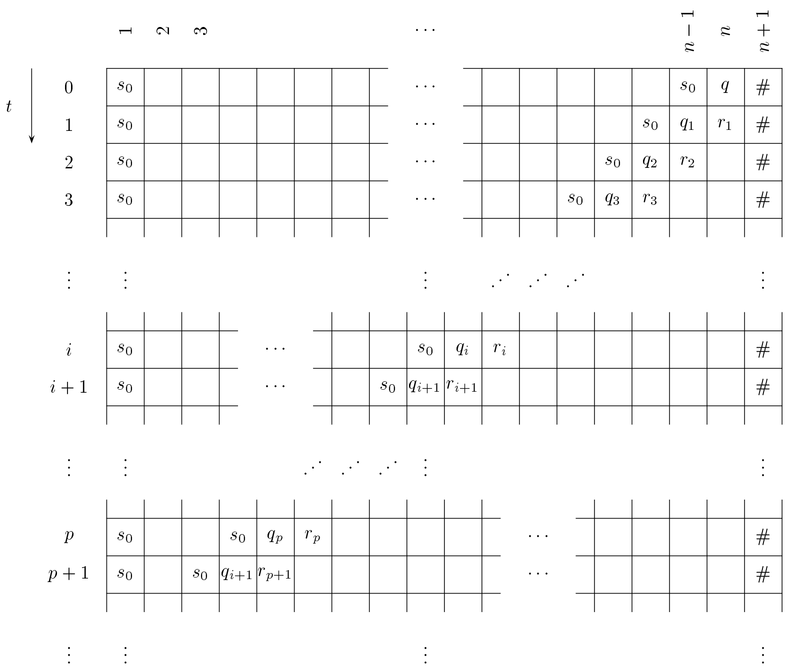

Figure 6: Space-time-diagram of a simulation.

When we are going to weigh the advantages and disadvantages of the two models under consideration we have to take account of the domains the functions are from.

Lemma 5 Let $f : \mathbb{N} \longrightarrow \mathbb{N}$ be a function where $f \not\geq id$, then $f \notin \mathscr{C}(\mathrm{OIA})$.

Proof. The lemma follows immediately from the fact that the leftmost cell is quiescent up to time n and the quiescent state cannot be final. □

On the other hand, there are functions below id (e.g. the constant functions) which belong to $\mathscr{C}(\mathrm{OCA})$ [2].

Theorem 6 Let $f : \mathbb{N} \longrightarrow \mathbb{N}$ be a function where $f \geq id$. $f \in \mathscr{C}(\mathrm{OCA}) \Longrightarrow f \in \mathscr{C}(\mathrm{OIA})$.

Proof. From an OCA which time-computes a function $f \geq id$ we can construct one all cells of which except the rightmost one are initially quiescent, which time computes f, too (cf. lemma 4). From such an OCA it is a little step to obtain an OIA for f since the rightmost cell is the input cell which is activated at time step 1. □

For the inversion of the theorem under the same assumptions one can consider the domain of functions where the input of an OIA leads not to an advantage. Obviously, these are the functions between id and $2id$.

Theorem 7 Let $f : \mathbb{N} \longrightarrow \mathbb{N}$ be a function where $id \leq f \leq 2id$. $f \in \mathscr{C}(\mathrm{OCA}) \iff f \in \mathscr{C}(\mathrm{OIA})$.

Proof. The part "$\Longrightarrow$" has been shown in theorem 6.

The input to the OIA is homogeneous (i.e. the single input symbols are not distinguishable). Therefore, only the length of the input may influence the computation. The last symbol is fed to the rightmost cell at time step n and may affect the computation of the leftmost cell at time step $2n$ at the earliest. We conclude that the part "$\Longleftarrow$" holds for $id \leq f < 2id$ and, trivially, $2id$ belongs to $\mathscr{C}(\mathrm{OIA})$.
Together with the fact that $2id$ belongs to $\mathscr{C}(\mathrm{OCA})$ [2] the theorem follows. □

Altogether, in the domain below id the OCA are more powerful, in the domain between id and $2id$ both architectures are evenly matched. It is not known whether in the domain beyond $2id$ the OIA are more powerful than OCA or both models are equivalent.

On the other hand, $2^{id} + id$ belongs to $\mathscr{C}(\mathrm{OCA})$ whereas 2^{id} does not seem so. It can easily be proved that if a function $f \geq 3id$ belongs to $\mathscr{C}(\mathrm{OCA})$, then $f - id$ belongs to $\mathscr{C}(\mathrm{OIA})$. Therefore 2^{id} belongs to $\mathscr{C}(\mathrm{OIA})$ which suggests $\mathscr{C}(\mathrm{OCA}) \subset \mathscr{C}(\mathrm{OIA})$ for functions beyond $2id$.

Now we are interested in the influence of pushdown storage augmentation of one-way devices.

In [2] the family $\mathscr{C}$(OCA) has been characterized in terms of formal languages. It has been shown that for a function $f : \mathbb{N} \longrightarrow \mathbb{N}$, where $f \geq id$, it holds $f \in \mathscr{C}(\text{OCA}) \iff \{\mathtt{a}^n\mathtt{b}^{f(n)-n} \mid n \in \mathbb{N}\} \in \mathscr{L}_{rt}(\text{OCA})$.

In what follows we prove a similar formal language characterization for the family $\mathscr{L}$ (OPDCA).

Lemma 8 Let $f : \mathbb{N} \longrightarrow \mathbb{N}$ be a function where $f \geq id$. $f \in \mathscr{C}(\text{OPDCA}) \iff L_f := \{\mathtt{a}^n\mathtt{b}^{f(n)-n} \mid n \in \mathbb{N}\} \in \mathscr{L}_{rt}(\text{OPDCA})$.

Proof. "$\Longrightarrow$" On an input w of the form $\mathtt{a}^n\mathtt{b}^m$ a corresponding real-time OPDCA acceptor for L_f may work as follows.

The a-cells simply simulate the time-computation of f. At time step 1 the rightmost cell (which can identify itself) sends a 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.

"$\Longleftarrow$" In [8] it was shown that a unary real-time OPDCA language is always a regular language. Therefore, the b-cells of a given real-time OPDCA acceptor for L_f could be replaced by a single finite-state machine fetching the input symbols sequentially such that the sequence of its states reflects the sequence of states of the leftmost b-cell. Furthermore, the finite-state machine can, at every time step, compute two states (in separate registers) one of which under the assumption that there has been an input symbol and the other under the assumption the last input symbol has been consumed one time step before.

An OPDCA time-computer for f may work as follows.

All the cells simulate a-cells of the acceptor. The rightmost cell additionally to its own work simulates the finite-state machine. Due to the described property each simulating cell has to do the job of an a-cell twice (in a parallel manner). It computes the state of the corresponding a-cell as well under the assumption that there is further work as under the assumption the last time step of the computation was done.

A cell becomes final when – under assumption the computation is finished – it would accept the input for the first time.

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

The following lemma gives an upper bound for time-computable functions in various models.

Lemma 9 Let POLY $\in$ {OCA, CA, OIA, IA}. If f belongs to $\mathscr{C}$(POLY) 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 POLY-time-computable by an POLY with state set S 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 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 time computability of f. $\square$

Example 10 Let POLY $\in$ {OCA, CA, OIA, IA}. Neither $f(n) = 2^{2^n}$ nor $f(n) = 2^{2^n} + n$ is POLY-time-computable since the series $\{\frac{2^{2^n}}{k^n}\}_{n\in\mathbb{N}}$ is unbounded for all $k \in \mathbb{N}$.

Now we are prepared to show that in case of one-way devices pushdown storage augmentation leads to strictly more powerful architectures.

Theorem 11 $\mathscr{C}$(OCA) $\subset \mathscr{C}$(OPDCA)

Proof. For structural reasons there is an inclusion between the families.

Consider the function $2^{2^{id}} + id$. In [8] it has been shown that $\{\mathtt{a}^n\mathtt{b}^{2^{2^n}} \mid n \in \mathbb{N}\} \in \mathscr{L}_{rt}$(OPDCA). Therefore, $2^{2^{id}} + id$ is OPDCA-time-computable (cf. lemma 8). But due to example 10 it does not belong to $\mathscr{C}$(OCA). $\square$

Theorem 12 $\exists\, f \in \mathscr{C}(\text{OPDCA}) : f \notin \mathscr{C}(\text{OIA})$

Proof. The proof of the theorem 11 can easily be adapted. $\square$

It is not known whether both families are incomparable.

3.2 Two-way devices

The fact that two-way devices can simulate a pushdown storage in real-time is often a useful tool for the modular design of algorithms. The principle of such a simulation is shown in the following.

Assume without loss of generality that at each time step at most one symbol from a nonempty finite stack alphabet is pushed onto or popped from the stack.

Assume furthermore the top of the stack is represented by the leftmost cell of the array.

Each cell has three registers in each of which one stack symbol can be stored. They are numbered 1, 2 and 3 downward the stack. The third register is used as a buffer. Each cell preferably fills two of its registers with symbols. In order to reach that charge it behaves according to the following rules (cf. figure 7):

– If all three registers of its left (upward) neighbor are filled then it takes over the symbol from the neighbors buffer and stores it in its first register. The contents of the first and second register is shifted to the second resp. third one.
– If there is filled just one register of its left neighbor then the symbol in the first register is deleted, whereby the content of the second and third register is shifted to the first resp. second on. The deleted symbol is taken over by the neighbor and stored in its lowest empty register.

Eventually two of the actions have to be superimposed and it can easily be verified that the simulation works correct. By storing two symbols into one cell and using a buffer the delay is avoided which is needed by the lower cells to react to operations applied to the top of the stack.

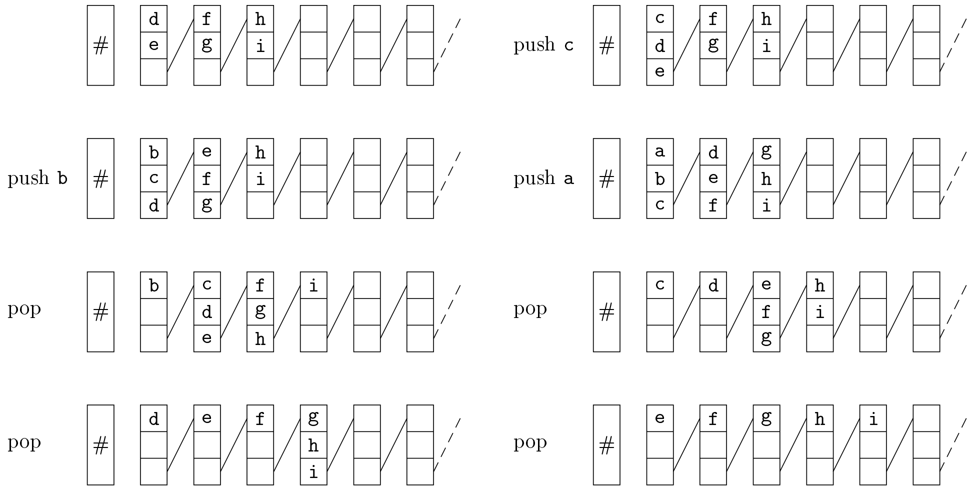

Figure 7: Example for pushdown store simulation by a two-way device.

The first two models we are going to compare are CA and IA because their relationships depend on the domain of considered functions.

Theorem 13 Let $f : \mathbb{N} \longrightarrow \mathbb{N}$ be a function. $f \in \mathscr{C}(\mathrm{IA}) \Longrightarrow f \in \mathscr{C}(\mathrm{CA})$

Proof. A cellular array which time-computes f has simply to simulate the iterative array. The only thing is to supply the input to the leftmost cell. This is done by setting up a signal at the right border at time step 1. The signal moves with speed 1 to the left. As long as it passes through the network the leftmost cell assumes that there is an input symbol. Obviously, at its arrival at the leftmost cell this one has simulated the input of exactly n symbols. $\square$

The families $\mathscr{C}(\mathrm{CA})$ and $\mathscr{C}(\mathrm{IA})$ coincide except for functions in the domain from id to $2id$. That gap corresponds to a gap of knowledge in this field.

Theorem 14 Let $f : \mathbb{N} \longrightarrow \mathbb{N}$ be a function, where $f \not\geq id$. $f \in \mathscr{C}(\mathrm{CA}) \iff f \in \mathscr{C}(\mathrm{IA})$

Proof. The part "$\Longleftarrow$" has been shown in the previous theorem.

The leftmost cell of a corresponding CA would be final without the influence of the size of the network, say n nodes, because a signal from the right border would need at least n time steps. Therefore one may enlarge the array without changing the first final time step and, hence, for all $n_0 \geq n$ the value $f(n_0)$ is identical to the value $f(n)$. Consequently, the time-computed function is ultimately constant. The fact that such functions are belonging to $\mathscr{C}(\mathrm{IA})$ is straightforward. □

Now we turn to the other side of the gap.
In order to prove the equivalence for functions beyond $3id$ we need two technical lemmas.

Lemma 15 Let $f : \mathbb{N} \longrightarrow \mathbb{N}$ be a function where $f \geq id$. $f \in \mathscr{C}(\mathrm{CA}) \Longrightarrow f + id \in \mathscr{C}(\mathrm{IA})$

Proof. Suppose the space-time-diagram of a time-computation in a CA and imagine that it is folded in such a manner that the right margin gets onto the left margin. Such a transformed diagram can be generated by the computation of a CA the cells of which consist of two registers such that a cell $i \leq \frac{n}{2}$ simulates the cells i and $n - i + 1$, respectively. Of course, such a CA uses only the left half of its cells.

At the embedding of that computation into an IA the problem of finding the center of the network arises. It can be solved by fetching the input during the first n time steps and propagating them to the right whereby each other marks the first non-marked node.

After n time steps the basic simulation of the CA cells 1 and n starts in cell 1. Since the initial input in the CA cells is homogeneous the whole computation of the CA can be simulated in that "folded" manner. □

Lemma 16 Let $f : \mathbb{N} \longrightarrow \mathbb{N}$ be a function where $f \geq 3id$. $f \in \mathscr{C}(\mathrm{IA}) \Longrightarrow f - id \in \mathscr{C}(\mathrm{IA})$

Proof. The simulation of the computation of the slow IA by the faster one is performed in three phases.

The first phase takes n time steps during which the input is fetched. Each third input symbol is propagated to the right whereby the first non-marked node is marked respectively. Additionally the nodes altogether simulate a pushdown store as described above. During the first phase $\frac{n}{3}$ tokens are pushed. At its end there has been no time step simulated.

The second phase takes another n time steps. It is started at the end of input and stopped by firing a time optimal FSSP at time step $2n$ which can be started at time step 2 on a separate track. By grouping three nodes to one (remember that there are $\frac{n}{3}$ nodes marked) the simulation can take place at a rate of triple speed. During the whole second phase that sped-up simulation is performed. To handle the input properly at every time step as long as tokens are available one of them is popped from the stack. Each token corresponds to three input symbols in the original computation. At the end of the second phase $3n$ time steps have been simulated.

When the FSSP fires the third phase starts in which simulation takes place at a rate of single speed until the computation is finished.

Altogether for all $m \geq 0$ at time step $2n + m$ the fast IA has simulated $3n + m$ time steps of the slow one which proves the lemma. $\square$

Now the lemmas are used to show that for $f \geq 2id$ the time-computing power of CA and IA is identical.

Theorem 17 Let $f : \mathbb{N} \longrightarrow \mathbb{N}$ be a function, where $f \geq 2id$. $f \in \mathscr{C}(\mathrm{CA}) \iff f \in \mathscr{C}(\mathrm{IA})$

Proof. The part "$\Longleftarrow$" has been shown in theorem 13.

Let $f \geq 2id$ be a CA-time-computable function. From lemma 15 we obtain $f+id \in \mathscr{C}(\mathrm{IA})$. It holds that $f+id \geq 3id$. Therefore, with lemma 16 $f+id-id = f$ belongs to $\mathscr{C}(\mathrm{IA})$ which proves the theorem. $\square$

In case of one-way information flow devices with pushdown storage have been shown to be more powerful than devices without this feature. In case of two-way information flow we obtain similar relations.

Theorem 18 $\mathscr{C}(\mathrm{CA}) \subset \mathscr{C}(\mathrm{PDCA})$

Proof. Again, for structural reasons it suffices to show that the inclusion is a proper one.

In the proof of theorem 11 it was shown that $2^{2^{id}} + id$ belongs to $\mathscr{C}(\mathrm{OPDCA})$ and, consequently, it belongs to $\mathscr{C}(\mathrm{PDCA})$. Due to example 10 it does not belong to $\mathscr{C}(\mathrm{CA})$. $\square$

Theorem 19 $\mathscr{C}(\mathrm{IA}) \subset \mathscr{C}(\mathrm{PDCA})$

Proof. From theorem 13 we know $\mathscr{C}(\mathrm{IA}) \subseteq \mathscr{C}(\mathrm{CA})$. The theorem follows with theorem 18. $\square$

3.3 One-way versus two-way devices

Previously, we have given formal language characterizations for $\mathscr{C}(\text{OCA})$ and $\mathscr{C}(\text{OPDCA})$. Unfortunately that characterization cannot be adapted to the family $\mathscr{C}(\text{CA})$ since, for example, the function $2^{2^{id}}$ does not belong to $\mathscr{C}(\text{CA})$ but the language $L_f := \{\mathtt{a}^n\mathtt{b}^{2^{2^n}-n} \mid n \in \mathbb{N}\}$ is real-time recognizable by CA [7]. These facts suggest that the families $\mathscr{C}(\text{OCA})$ and $\mathscr{C}(\text{CA})$ are different.

The following results show that, in some cases, two-way architectures have more computing power than one-way devices.

Theorem 20 There exists a function $f \in \mathscr{C}(\text{IA})$ which does not belong to $\mathscr{C}(\text{OIA})$.

Proof. Due to example 39 below the function $f := id + \lfloor \log\log\log \rfloor$ belongs to $\mathscr{C}(\text{IA})$. In [2] it has been shown that f is not OCA-time-computable. Due to theorem 7 then f is a witness for the properness of the inclusion. $\square$

Corollary 21 There exists a function $f \in \mathscr{C}(\text{IA})$ which does not belong to $\mathscr{C}(\text{OCA})$.

Proof. The corollary follows immediately from theorem 6 and theorem 20. $\square$

Corollary 22 $\mathscr{C}(\text{OCA}) \subset \mathscr{C}(\text{CA})$

Proof. The corollary follows immediately from theorem 13, corollary 21 and structural reasons. $\square$

Theorem 23 $\mathscr{C}(\text{OIA}) \subset \mathscr{C}(\text{CA})$

Proof. The proof corresponds to that of theorem 13. But instead of setting up a leftward signal at the right border a rightward signal is generated at the left border. $\square$

Corollary 24 $\mathscr{C}(\text{OCA}) \subset \mathscr{C}(\text{PDCA})$ and $\mathscr{C}(\text{OIA}) \subset \mathscr{C}(\text{PDCA})$

Up to now we have shown that the one-way devices OCA and OIA are strictly less powerful (in terms of time-computation) than IA, CA and PDCA. If the single nodes are augmented by pushdown storage their power is strengthen.

The next lemma proves that there is a gap between $id + k$ and $id + \lfloor \log\log \rfloor$ in the family $\mathscr{C}(\text{OPDCA})$. The gap is later used to obtain non OPDCA-time-computable functions.

Lemma 25 Let $f \geq id$ be an OPDCA-time-computable function which satisfies $\forall\, b \in \mathbb{N} : f \ngeq id + \lfloor \log_b \log_b \rfloor$. Then there exists a $k \in \mathbb{N}_0$ such that $f(n) = n + k$ for infinitely many $n \in \mathbb{N}$.

Proof. Suppose $f \geq id$ is OPDCA-time-computable by an OPDCA with state set S and stack symbols from Γ and satisfies $\forall\, b \in \mathbb{N} : f \not\geq id + \lfloor \log_b \log_b \rfloor$. Then especially there exists an $n_0 \in \mathbb{N}$ such that $n_0 \leq f(n_0) < n_0 + \lfloor \log_{x^2} \log_{x^2}(n_0) \rfloor$, where $x := \max\{|S|, |\Gamma|\}$.

Assume for the rest of the proof that the cells are numbered from right to left in ascending order. Considering the computation of the OPDCA we denote the state and the upper $\lfloor \log_{x^2} \log_{x^2}(n_0) \rfloor + 1$ stack symbols of a cell i at time t by $c_t(i)$ and the evolution

$$c_{i-1}(i)c_i(i)c_{i+1}(i)\cdots c_{i+\lfloor \log_{x^2} \log_{x^2}(n_0)\rfloor -1}(i)$$

by e_i.
Observe, that the lengths of all evolutions are identical ($\lfloor \log_{x^2} \log_{x^2}(n_0) \rfloor + 1$).
In total there are less than

$$\begin{aligned} & x^{\lfloor \log_{x^2} \log_{x^2}(n_0) \rfloor + 1} \cdot x^{((\lfloor \log_{x^2} \log_{x^2}(n_0) \rfloor + 1)^2)} \\ < \; & x^{\log_{x^2}(n_0)} \cdot x^{((\sqrt{\log_{x^2}(n_0)})^2)} \\ = \; & x^{\log_{x^2}(n_0)} \cdot x^{\log_{x^2}(n_0)} \\ = \; & x^{2 \cdot \frac{1}{2} \cdot \log_x(n_0)} \\ = \; & n_0 \end{aligned}$$

different evolutions of M. Hence, at least two of the evolutions from $e_1, \ldots, e_{n_0}$, say e_i and e_j $(i < j)$ are identical. Since the OPDCA is a deterministic device and the initial input consists of identical states q 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. □

By augmenting the models by pushdown storage their capabilities are extended such that the family $\mathscr{C}$(OPDCA) contains functions neither belonging to $\mathscr{C}$(IA) nor to $\mathscr{C}$(CA). But in turn these families contain functions not belonging to $\mathscr{C}$(OPDCA).

Theorem 26 There exists a function $f \in \mathscr{C}$(IA) which does not belong to $\mathscr{C}$(OPDCA).

Proof. Together with lemma 25 from example 39 we obtain a witness for the claim. □

Corollary 27 There exists a function $f \in \mathscr{C}$(CA) and $f \in \mathscr{C}$(PDCA) which does not belong to $\mathscr{C}$(OPDCA).

Theorem 28 $\mathscr{C}(\mathrm{CA})$ and $\mathscr{C}(\mathrm{IA})$ are incomparable to $\mathscr{C}(\mathrm{OPDCA})$.

Proof. From the proof of theorem 12 we know that $2^{2^{id}} + id$ belongs to $\mathscr{C}(\mathrm{OPDCA})$, but it does neither belong to $\mathscr{C}(\mathrm{CA})$ nor to $\mathscr{C}(\mathrm{IA})$, which has been shown by lemma 9. □

Theorem 29 $\mathscr{C}(\mathrm{OPDCA}) \subset \mathscr{C}(\mathrm{PDCA})$

Proof. The theorem follows from corollary 27 and structural reasons. □

4 Time constructibility

In the sixties Fischer [5] found an iterative array unbounded to the right (i.e. there are infinitely many nodes in the network) in which the leftmost cell switches to a final state exactly at every time step that is a prime number.

A general investigation of that concept of its own was started by Mazoyer and Terrier [12]. According to the approach of Fischer they use an infinite halfline of nodes. All the nodes except the leftmost one are initially quiescent. No input is supplied to the network. They call their model *impulse cellular automaton* and the time constructible functions *Fischer's constructible*. Observe, that with regard to our notion in their model all values $f(n)$ have to be constructed since the number of nodes is infinite.

On the other hand, in impulse cellular automata there is no restriction on the available space (i.e. even for the construction of the value $f(n)$ more than n nodes may be used).

In order to incorporate some results from [12] we will consider infinite arrays, too, but for historical reasons we call them *cellular spaces* (CS for short).

The following theorem shows that with regard to time constructibility the computing power of CA and IA coincide.

Theorem 30 $\mathscr{F}(\mathrm{IA}) = \mathscr{F}(\mathrm{CA})$

Proof. Using the same argumentation as in theorem 13 one can show that $\mathscr{F}(\mathrm{IA}) \subseteq \mathscr{F}(\mathrm{CA})$, since the input in an IA may be simulated by use of an initial leftward signal generated at the right border in a CA.

For the inversion we have to deal with the possibility that the right border cell in a CA could influence the computation. But this possibility is invalid. Suppose there is a CA-time-constructor of size n, which marks the leftmost cell at time steps $f(1), \dots, f(n)$, respectively. The same CA-time-constructor on size $f(n)+1$ marks the cell at time steps $f(1), \dots, f(n), \dots, f(f(n)+1)$ but does it for time steps $f(1), \dots, f(n)$ without influence of any signal from the right border cell. Such a signal would need at least $f(n)+1$ time steps to reach the leftmost cell.

Because n was chosen arbitrarily the CA-time-constructor works independently of an influence of the rightmost cell and, thus, can be simulated by an IA-time-constructor.

□

As mentioned above the restriction in space is a hard one although for a function in $\mathscr{F}(\mathrm{CA})$ only the values up to $f(n)$ have to be constructed.

Theorem 31 $\mathscr{F}(\mathrm{CA}) \subset \mathscr{F}(\mathrm{CS})$

Proof. To prove the inclusion one cannot assert structural reasons since in cellular arrays there may be an initial signal generated at the right border, what cannot be done in cellular spaces. But fortunately from the proof of theorem 30 it can be seen that the advantage is not a really one.

It remains to show that the inclusion is a proper one. In [12] it has been shown that the factorials form a CS-time-constructible function, whereas from lemma 9 follows that $id! \notin \mathscr{C}(\mathrm{IA})$ and, hence, with corollary 33 $id! \notin \mathscr{F}(\mathrm{CA})$. □

Although in [12] the family $\mathscr{F}(\mathrm{CS})$ is under consideration for some of the functions it can easily be seen that they belong to $\mathscr{F}(\mathrm{CA})$, too, since the corresponding time-constructor is linear space bounded (e.g. the polynomials).

4.1 Relations between time constructibility and time computability

The next result shows that the domain of CA- and IA-time-computable functions is at least as rich as the domain of CA- and IA-time-constructible functions.

Theorem 32 $\mathscr{F}(\mathrm{IA}) \subset \mathscr{C}(\mathrm{IA})$

Proof. Once it is shown that there is an inclusion it is a proper one since time-constructible functions have to be strictly increasing but two-way time-computable function may be constant.

Let f be a function that is IA-time-constructible. An IA-time-computer for f has to perform two tasks.

One is to simulate the IA-time-constructor which can be done directly. The other one is to recognize the time step at which the leftmost cell is marked for the nth time because this is time step $f(n)$ at which the time-computer has to become final.

For that reason all nodes together simulate a pushdown store as described above. The stack is initially empty. Every input symbol fed is pushed onto the stack.

On the other hand, at every time step the leftmost cell is marked by the time-constructor one input symbol is popped from the stack. Since there are exactly n input symbols the leftmost cell is marked for the nth time when the stack gets empty. □

Corollary 33 $\mathscr{F}(\mathrm{CA}) \subset \mathscr{C}(\mathrm{IA})$

Corollary 34 $\mathscr{F}(\mathrm{IA}) \subset \mathscr{C}(\mathrm{CA})$ and $\mathscr{F}(\mathrm{CA}) \subset \mathscr{C}(\mathrm{CA})$

Proof. The corollary follows from theorem 13, theorem 32 and corollary 33. □

Due to the next theorem one can conclude from a time-computable function to another time-constructible one.

Theorem 35 Let f be a strictly increasing function, then $f \in \mathscr{C}(\mathrm{OCA}) \Longrightarrow f + id \in \mathscr{F}(\mathrm{CA})$.

Proof. Since a CA has two-way information flow it can easily simulate an OCA. But it can also simulate an OCA for which the information flow is from left to right instead from right to left. Evidently, one can easily construct such an OCA-time-computer from a given "normal" OCA-time-computer.

To prove the theorem we construct a CA-time-constructor for $f + id$ from an OCA-time-computer for f. The CA simulates an OCA-time-computer for f with reverse information flow. I.e. the cells i would become final respectively at time $f(i)$ firstly (f is strictly increasing). Exactly at that time step they send a signal to the left which marks the leftmost cell at time step $f(i) + i$, respectively. □

Corollary 36 Let f be a strictly increasing function, then $f \in \mathscr{C}(\mathrm{OCA}) \Longrightarrow f + id \in \mathscr{F}(\mathrm{IA})$.

The next two results allow in some sense the adaption of the results in [12] to the notion of time computability. The proofs are tedious and hard to read. They are omitted here but may be found in [3].

To a function $f : \mathbb{N} \longrightarrow \mathbb{N}$ we define a function $F : \mathbb{N} \longrightarrow \mathbb{N}$ according to $F(m) := \max\{n \mid f(n) \leq m\} \cup \{0\}$.

Theorem 37 $f \in \mathscr{F}(\mathrm{CS}) \iff F + 2id \in \mathscr{C}(\mathrm{OCA})$

Theorem 38 $f \in \mathscr{F}(\mathrm{CS}) \Longrightarrow F + id \in \mathscr{F}(\mathrm{CA})$

Example 39 In [12] it was shown that the function $2^{2^{2^{id}}}$ belongs to $\mathscr{F}(\mathrm{CS})$. With theorem 38 we conclude that $f := id + \lfloor \log \log \log \rfloor$ belongs to $\mathscr{F}(\mathrm{CA})$ which is identical to $\mathscr{F}(\mathrm{IA})$ (cf. theorem 30). Furthermore, from theorem 32 it follows $f \in \mathscr{C}(\mathrm{IA})$.

5 What is known?

Instead of comparing the time-computing power of several architectures one can ask for the hierarchical structure of the time-computable functions in a specific model. In figure 8 the known results for OCA (left part) and CA (right part) are summarized. It can be seen that there exist gaps between constant functions k and the function id. For example neither $\lfloor \log \rfloor$ nor $\lfloor \sqrt{\ } \rfloor$ is (O)CA-time-computable.

In $\mathscr{C}(\mathrm{OCA})$ there is another gap between $id + k$ and $id + \lfloor \log \rfloor$. The area below $id + \lfloor \log \rfloor$ is fuzzy since we can choose the base of the logarithm arbitrarily. It is not known whether there are more gaps up to the fuzzy upper bound which is marked by the exponential functions.

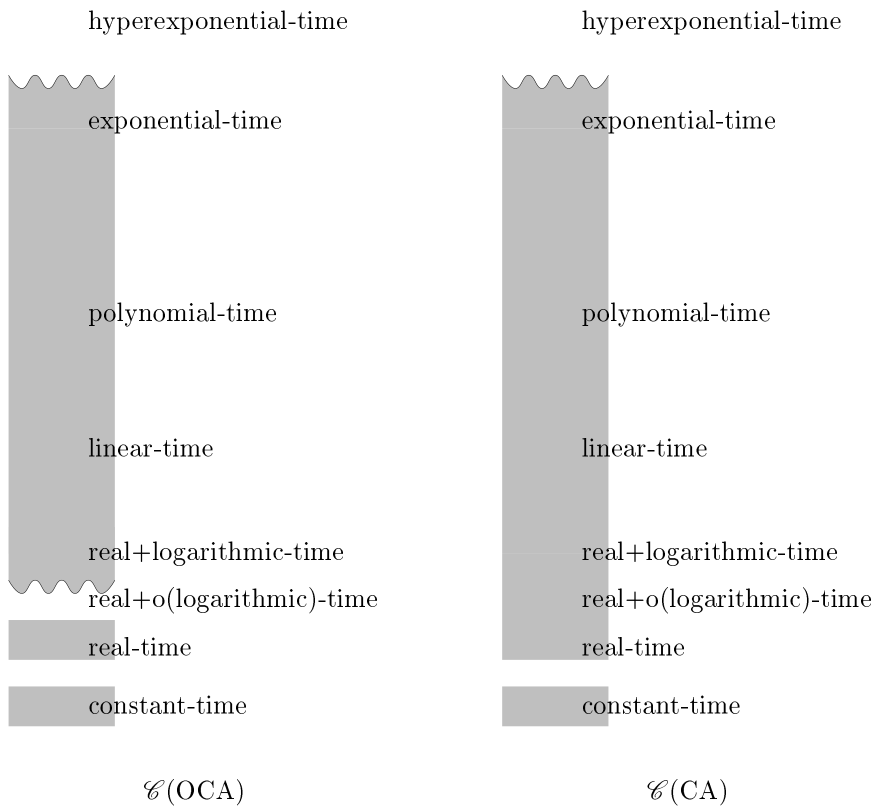

Figure 8: Hierarchical structure of $\mathscr{C}(\mathrm{OCA})$ and $\mathscr{C}(\mathrm{CA})$.

In the following diagram known relations and implicitly open problems concerning the relations between the families considered above are depicted. The dashed arrows show inclusions which may be proper or not, the single arc lines indicate that the connected families are incomparable or there is a proper inclusion between them.

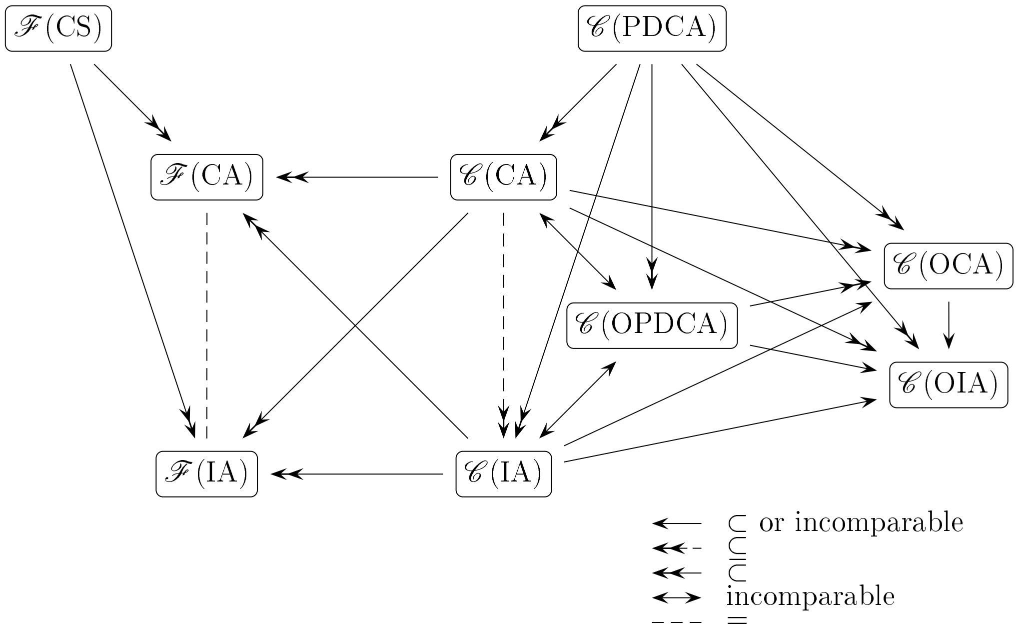

Figure 9: Relations between the considered families.

References

[1] Amoroso, S., Lieblein, E., and Yamada, H. *A unifying framework for the theory of iterative arrays of machines*. 1^{st} Ann. ACM Symposium on Theory of Computing (STOC '69), Marina del Ray, CA, 1969, pp. 259–269.

[2] Buchholz, Th. and Kutrib, M. *On time computability of functions in one-way cellular automata*. Report 9502, Arbeitsgruppe Informatik, Universität Gießen, Gießen, 1995.

[3] Buchholz, Th. and Kutrib, M. *On the power of one-way bounded cellular time computers*. Report 9604, Arbeitsgruppe Informatik, Universität Gießen, Gießen, 1996.

[4] Codd, E. F. *Cellular Automata*. Academic Press, New York, 1968.

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

[6] Hennie, F. *Iterative Arrays of Logical Circuits*. MIT Press, Cambridge, Mass., 1961.

[7] Kutrib, M. *A note on cellular real-time recognizability of languages definable by a family of functions*. Unpublished manuscript, 1995.

[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, pp. 420–429.

[10] Kutrib, M. and Worsch, Th. *Investigation of different input modes for cellular automata*. Parcella '94, Akademie Verlag, Berlin 1994, pp. 141–150.

[11] Mazoyer, J. *A six-state minimal time solution to the firing squad synchronization problem*. Theoretical Computer Science 50 (1987), 183–238.

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

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

[14] Vollmar, R. *Some remarks on pipeline processing by cellular automata*. Computers and Artificial Intelligence 6 (1987), 263–278.

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