ARBEITSGRUPPE INFORMATIK
UNIVERSITÄT GIESSEN
ARNDTSTR. 2, D-35392 GIESSEN, GERMANY

On stack-augmented polyautomata

Martin Kutrib

Report 9501 July 1995

JUSTUS-LIEBIG-
UNIVERSITÄT
GIESSEN

On stack-augmented polyautomata

Martin Kutrib
AG Informatik, University of Giessen
Arndtstr. 2, D-35392 Giessen, Germany
kutrib@informatik.uni-giessen.de

July 1995

Abstract

A stack augmented generalization of cellular automata, the pushdown cellular automata, are investigated. We are studying the language accepting capabilities of such devices. Closure properties of real-time, linear-time and unrestricted time language families are shown. The relationships of these families with each other and to languages of sequential automata are considered.

1 Introduction

Linear arrays of automata can be understood as models for massively parallel computers. Various types of finite state machine arrays have been studied for a long time (see e.g. [2, 3, 4, 6, 7, 8, 14, 15, 17, 20, 23, 24, 25]). Mainly they differ in how the automata are interconnected and in how the input is supplied. Here we are investigating arrays with a very simple interconnection pattern. Each node is connected to its (one or two) immediate neighbors only. They are usually called cellular automata (CA) if the input is supplied in parallel and iterative arrays (IA) in case of sequential input to a designated single automaton. We will use the notion one-way (OCA) if the single automata (cells) are connected to its left immediate neighbor only. Moreover, our cells are not finite state machines. We study linear arrays of deterministic pushdown automata. Accordingly our models are called pushdown cellular automata (PDCA) resp. iterative pushdown arrays (IPDA). On a first glance the automata are more complex than their "classical" counterpart. On the other hand if computational universality should be obtained in the classical case we have to consider infinite arrays, the so-called cellular spaces or unbounded cellular automata, whereas in case of pushdown cellular automata arrays of at least length two are sufficient.

We establish the relationships to various other models by studying the power of the arrays as language acceptors. We focus our attention on real-time, linear-time and unrestricted time computations.

It is known that the real-time OCA languages form a proper subset of the real-time CA languages [20] which again are equal to the linear-time OCA languages [4]. On the other hand it is a long standing open problem whether the real-time CAs are less powerful than the linear-time CAs or not [2]. Closely related to that problem are the open problems of whether the real-time CA languages are closed under reversal or concatenation [21]. In this coherency Ibarra and Jiang [14] have shown that the closure under reversal would imply the closure under concatenation and that the concatenation of two real-time CA languages is a linear-time language.

In case of one-way information flow the closure of real-time OCA languages under reversal was shown in [4], whereas Terrier [23] proved that these languages are not closed under concatenation. The relationships to iterative arrays are as follows: The real-time IAs are less powerful than the real-time CAs [14] and the linear-time IA languages are equal to the linear-time CA languages. Cole [6] proved that the real-time IA languages are neither closed under reversal nor under concatenation. Additionally, it is known that the real-time IAs are incomparable to the real-time OCAs [20, 4].

Smith [21] raised the still open problem whether the context-free languages are contained in the real-time CA languages. The question was answered for several subsets of the context-free languages. E.g. it was shown in [13] that every semi-linear language is real-time acceptable by CAs. Furthermore it is known that the real-time OCA languages [23] and the real-time IA languages [6] are incomparable to the context-free languages.

Here we will show a number of results about the capabilities, relationships to other types of acceptors and closure properties of PDCAs, OPDCAs and IPDAs. Our paper is organized as follows:

Additional to the present it consists of four sections. In section 2 preliminary some definitions of acceptors are reviewed and notations are declared. Section 3 is divided into two subsections. The first one defines formally the model pushdown cellular automaton and how it can accept languages. The second one gives some important technical results needed in later sections. In section 4 we compare pushdown cellular automata to their classical counterparts as well as to sequential Turing machines and linear bounded automata. In section 5 various closure properties of the considered language families are obtained.

2 Preliminaries

Although our main interest in the present paper is on pushdown cellular automata we will review the notions (one-way) cellular automaton and iterative array shortly. One of the reasons to do so is that we want to compare pushdown cellular automata with such devices. Another one is that pushdown cellular automata are in some sense generalizations of classical cellular automata.

A *cellular automaton* (CA) is a linear array of identical finite state machines, sometimes called cells, each of which is connected to its both immediate neighbors. All cells work synchronously at discrete time steps. With an eye towards language recognition more formally a cellular automaton is a system $(S, \sigma, \#, F)$, where S is the finite nonempty set of states, $F \subseteq S$ is the set of final states, $\sigma : S^3 \longrightarrow S$ is the local 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 the following:
$\forall n \in \mathbb{N}, i \in \{1, 2, \ldots, n\} : \forall s_i \in S :$

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

Let $M = (S, \sigma, \#, F)$ be a CA, $L \subseteq A^+$ a formal language and $t : \mathbb{N} \longrightarrow \mathbb{N}$ a function. L is accepted by M in time t (with respect to F) iff $A \subseteq S$ and

$$L = \{a_1 \cdots a_n \mid \pi_n(\mathcal{T}^{t(n)}(a_1 \cdots a_n)) \in F \\ \wedge \forall t' < t(n) : \pi_n(\mathcal{T}^{t'}(a_1 \cdots a_n)) \notin F\}$$

$\pi_i(a_1 \cdots a_n) := a_i$ selects the ith component of $a_1 \cdots a_n$ and $\mathcal{T}^k$ denotes the k-fold composition of $\mathcal{T}$.

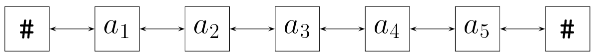

Figure 1: A cellular automaton.

If we restrict the flow of information to one-way (e.g. from left to right), the resulting device is an *one-way cellular automaton* (OCA). I.e. the next state of each cell depends on the state of the cell itself and the state of its left neighbor only.

Iterative arrays (IA) consist of an infinite array of cells. They differ from CAs in how the input symbols are supplied. Whereas in CAs we have a parallel

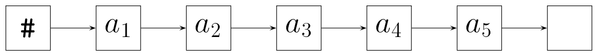

Figure 2: An one-way cellular automaton.

input mode the symbols are fed serially to a distinguished cell in iterative arrays. (E.g. this could be the cell at the origin if we identify the cells by integers.) Formally an iterative array is a system $(S, A, \sigma, F, s_0, a_0)$, where S and F are as for CAs. A is the finite nonempty set of input symbols containing a_0 the end-of-input symbol. The local transition function σ maps from $S^3 \cup (S^3 \times A)$ (depending on whether the cell is the distinguished one or not) to S. It satisfies $\sigma(s_0, s_0, s_0) = s_0$. Due to this property $s_0 \in S$ is called the quiescent state in which all cells are at time step 0. We assume that at the end of the input the symbol a_0 appears infinitely often. The global transition $\mathcal{T}$ now maps from A^+S^* to A^+S^+ as follows. For $n > 0$ and $m \geq 0$ let $a_1 \cdots a_n b_{-m} \cdots b_0 \cdots b_m$ be a word with symbols $a_i \in A$, $1 \leq i \leq n$, and $b_j \in S$, $-m \leq j \leq m$.

$$\mathcal{T}(a_1 \cdots a_n b_{-m} \cdots b_0 \cdots b_m) :=$$
$$\begin{cases} a_1 \cdots a_{n-1}\sigma(s_0, s_0, b_0)\sigma(s_0, b_0, s_0, a_n)\sigma(b_0, s_0, s_0) & \text{if } m = 0 \\ a_1 \cdots a_{n-1}\sigma(s_0, s_0, b_{-m}) \cdots \sigma(b_{-1}, b_0, b_1, a_n) \cdots \sigma(b_m, s_0, s_0) & \text{if } m > 0 \end{cases}$$

A language $L \subseteq (A \setminus \{a_0\})^+$ is accepted by an IA in time t (with respect to F) iff

$$L = \{a_1 \cdots a_n \mid \pi_0(\mathcal{T}^{t(n)}(a_0^{t(n)-n} a_n \cdots a_1 s_0)) \in F \\ \wedge \forall t' < t(n) : \pi_0(\mathcal{T}^{t'}(a_0^{t(n)-n} a_n \cdots a_1 s_0)) \notin F\}$$

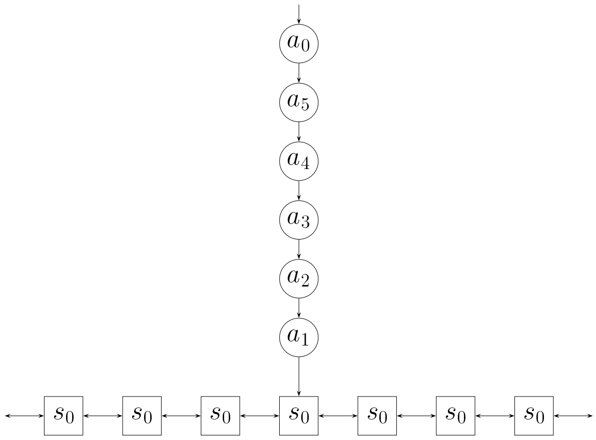

Figure 3: An iterative array.

The families of languages acceptable by CAs (OCAs, IAs) in time $t(n)$ are denoted by $\mathscr{L}_{t(n)}(\mathrm{CA})$ ($\mathscr{L}_{t(n)}(\mathrm{OCA})$, $\mathscr{L}_{t(n)}(\mathrm{IA})$). Of special interest are the real-time languages which can be accepted within time $rt(n) := n$ and the linear-time languages $\mathscr{L}_{lt} := \bigcup_{k\in\mathbb{N}} \mathscr{L}_{k\cdot n}$. In case of unrestricted computation time the subscript is omitted. We denote the regular languages by $\mathscr{L}_3$, the context-free languages by $\mathscr{L}_2$ and the deterministic context-sensitive languages by $\mathscr{L}_1$.

3 Some technical results on pushdown cellular automata

3.1 The model

In this subsection we will formally define what pushdown cellular automata are. As mentioned above the main idea is to replace the finite automata which are actually the cells by deterministic pushdown automata.

Definition 3.1
A *pushdown cellular automaton* (PDCA) is a system $(S, \Gamma, \sigma, \#, F, g_0)$, where

a) S is the finite, nonempty set of *states*,
b) Γ is the finite, nonempty set of *stack symbols*,
c) $\# \in S$ is the *boundary state*,
d) $F \subseteq S$ is the set of *final states*,
e) $g_0 \in \Gamma$ is the *bottom of stack symbol* and
f) $\sigma : S^3 \times \Gamma \longrightarrow S \times \Gamma^*$ is the *local transition function* satisfying
  i) $\forall s_1, s_2, s_3 \in S, g \in \Gamma : \; \pi_1(\sigma(s_1, s_2, s_3, g)) = \# \iff s_2 = \#$
  ii) $\forall s_1, s_2, s_3, s_4 \in S, g \in \Gamma : \; \sigma(s_1, s_2, s_3, g) = (s_4, \gamma) \Longrightarrow$
  $\Big(\gamma \in (\Gamma \setminus \{g_0\})^* \wedge g \neq g_0\Big) \vee \Big(\gamma = \gamma' g_0 \wedge \gamma' \in (\Gamma \setminus \{g_0\})^* \wedge g = g_0\Big)$

The second condition on the local transition ensures that the bottom of stack symbol appears at each cell exactly once (i.e. at the bottom of its stack). At each transition step each cell consumes the symbol at the top of its stack (if it is not empty) and pushes a possibly empty string of stack symbols onto it. Observe that a restriction of pushing at most two symbols at each time step would neither reduce the computation power nor slowdown the computation itself. We call PDCAs with this property *stack normalized*.

The length preserving global transition $\mathcal{T} : (S \times \Gamma^+)^+ \longrightarrow (S \times \Gamma^+)^+$ is induced by σ as follows:

Let $\bar{\sigma} : S^3 \times \Gamma^+ \longrightarrow S \times \Gamma^+$ be defined as
$\bar{\sigma}(s_1, s_2, s_3, g_m \cdots g_0) := \Big(\pi_1(\sigma(s_1, s_2, s_3, g_m)), \pi_2(\sigma(s_1, s_2, s_3, g_m))g_{m-1} \cdots g_0\Big)$,
then $\forall n \in \mathbb{N}, i \in \{1, \ldots, n\} : \forall s_i \in S, \ \gamma_i \in \Gamma^+ :$

$$\begin{aligned} \mathcal{T}((s_1, \gamma_1)) &:= \bar{\sigma}(\#, s_1, \#, \gamma_1) \\ \mathcal{T}((s_1, \gamma_1) \cdots (s_n, \gamma_n)) &:= \bar{\sigma}(\#, s_1, s_2, \gamma_1)\bar{\sigma}(s_1, s_2, s_3, \gamma_2) \cdots \bar{\sigma}(s_{n-1}, s_n, \#, \gamma_n) \end{aligned}$$

Initially all stacks are empty. We often refer to configurations c_i of PDCAs at time steps $i \geq 0$. With $c_0 := (a_1, g_0) \cdots (a_n, g_0)$ we define $c_{i+1} := \mathcal{T}(c_i)$ and $c_i(j) := \pi_j(c_i)$.

Definition 3.2 Let $M = (S, \Gamma, \sigma, \#, F, g_0)$ be a PDCA, $L \subseteq A^+$ a formal language and $t : \mathbb{N} \longrightarrow \mathbb{N}$ a function.
L is accepted by M in time t (with respect to F) iff $A \subseteq S$ and

$$\begin{aligned} L = \{a_1 \cdots a_n \mid \pi_1\Big(\pi_n(\mathcal{T}^{t(n)}((a_1, g_0) \cdots (a_n, g_0)))\Big) \in F \\ \wedge \forall t' < t(n) : \pi_1\Big(\pi_n(\mathcal{T}^{t'}((a_1, g_0) \cdots (a_n, g_0)))\Big) \notin F\} \end{aligned}$$

The definitions for *one-way pushdown cellular automata* (OPDCA) and *iterative pushdown arrays* (IPDA) are straightforward and omitted here.

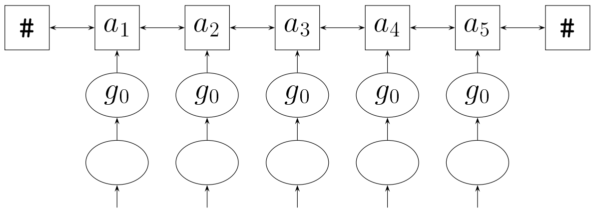

Figure 4: A pushdown cellular automaton.

3.2 Technical results

In this subsection we will prove some technical results meaningly on real-time OPDCAs. This includes a property of unary languages and some example languages. The lemmas are used in later sections to prove closure properties and establish the relationships to other types of acceptors.

It is well-known that a formal language over an one-letter alphabet is context-free iff it is regular [9, 19]. In [20] Seidel has shown: if such a language can be accepted by an OCA in real-time, then it is regular, too. We now turn to prove a similar result for OPDCAs, from which follows that in case of unary languages a single finite state machine has the same accepting power as a real-time OPDCA.

Lemma 3.1
A formal language over an one-letter alphabet belongs to $\mathscr{L}_{rt}(\text{OPDCA})$ iff it is regular.

Proof. Let L be a language over $A = \{\mathtt{a}\}$ and $M = (S, \Gamma, \sigma, \mathtt{\#}, F, g_0)$ an OPDCA accepting it in real-time. At first we are giving the construction of a deterministic pushdown automaton (PDA) $B = (S', A, \Gamma, \delta, F', s_0, g_0)$ which accepts L with respect to $F' \subseteq S'$. $s_0 \in S'$ denotes the starting state of B and its (partial) transition function δ maps from $S' \times (A \cup \{\varepsilon\}) \times \Gamma$ to $S' \times \Gamma^*$, where additionally to the condition on the bottom of stack symbol the following holds: If for any $s \in S'$ and $g \in \Gamma$ the value $\delta(s, \varepsilon, g)$ is defined then $\delta(s, \mathtt{a}, g)$ is undefined. (ε denotes the empty word.)

At each time step the configuration of a PDA is a triple (s, w, γ), where $s \in S'$ is the actual state, $w \in A^*$ the remaining input and $\gamma \in \Gamma^+$ the content of the stack. The automaton computes the configuration $(s', w, \beta\gamma)$ from $(s, w_1 w, g\gamma)$ iff $\delta(s, w_1, g) = (s', \beta)$. We write $(s, w_1 w, g\gamma) \vdash (s', w, \beta\gamma)$ for short.
Construction:
Let $\mathtt{1} \notin S$ be a symbol not belonging to S.

$$S' := (S \cup \{\mathtt{1}\})^2, \qquad s_0 := (\mathtt{1}, \mathtt{1}), \qquad F' := \{s \in S' \mid \pi_2(s) \in F\},$$

δ is defined as follows:
$\delta(s_0, \mathtt{a}, g_0) := \Big(\big(\mathtt{a},\ \ \pi_1(\sigma((\mathtt{\#}, \mathtt{a}), g_0))\big),\ \ g_0\Big)$
Let si be an abbreviation for $\sigma((s, s), g)$. $\forall s, \tilde{s} \in S, g \in \Gamma :$
$\delta((s, \tilde{s}), \mathtt{a}, g) := \Big(\big(\pi_1(si),\ \ \pi_1(\sigma((\tilde{s}, \pi_1(si)), \pi_2(si)))\big),\ \ \pi_2(si)\Big)$
Correctness:
We claim: An input is accepted by M iff it is by B. Since B is a PDA L is then context-free and as stated above regular.

Proof of the claim.
'$\Longrightarrow$' Let for $n \in \mathbb{N}$ $w = \mathtt{a}^n$ be a word accepted by M in real-time. Assume until time step n M computes the following sequence of configurations:

$$\begin{aligned} t = 0 : &\quad (\mathtt{a}, g_0)^n \\ 0 < t < n : &\quad (r_{t_1}, \gamma_{t_1}) \cdots (r_{t_t}, \gamma_{t_t})(s_t, \beta_t)^{(n-t)} \\ t = n : &\quad (r_{n_1}, \gamma_{n_1}) \cdots (r_{n_n}, \gamma_{n_n}) \end{aligned}$$

For $1 \leq t \leq n$ (r_{t_t}, γ_{t_t}) and (s_t, β_t) can be specified by

$$\begin{aligned}(r_{t_t}, \gamma_{t_t}) &= \begin{cases} \sigma((\#, \mathtt{a}), g_0) & \text{if } t = 1 \\ \sigma((r_{t-1_{t-1}}, s_{t-1}), \beta_{t-1}) & \text{otherwise} \end{cases} \\ (s_t, \beta_t) &= \begin{cases} \sigma((\mathtt{a}, \mathtt{a}), g_0) & \text{if } t = 1 \\ \sigma((s_{t-1}, s_{t-1}), \beta_{t-1}) & \text{otherwise} \end{cases}\end{aligned}$$

Since w is accepted by M it must hold $r_{n_n} \in F$.

Given w as input to B from the construction of δ we obtain:
$((\mathtt{1}, \mathtt{1}), \mathtt{a}^n, g_0) \vdash ((\mathtt{a}, r_{1_1}), \mathtt{a}^{(n-1)}, g_0) \vdash ((s_1, r_{2_2}), \mathtt{a}^{(n-2)}, \beta_1) \vdash^{n-2}$
$((s_{n-1}, r_{n_n}), \varepsilon, \beta_{n-1})$

After n time steps B stops since δ is not defined for empty input. From $\pi_2(s_{n-1}, r_{n_n}) = r_{n_n} \in F$ follows $(s_{n-1}, r_{n_n}) \in F'$. Consequently w is accepted by B.

'$\Longleftarrow$' Let $w = \mathtt{a}^n$ be a word accepted by B. Assume under input w automaton B computes the following sequence of configurations:
$((\mathtt{1}, \mathtt{1}), \mathtt{a}^n, g_0) \vdash ((s_1, s'_1), \mathtt{a}^{(n-1)}, \gamma_1) \vdash ((s_2, s'_2), \mathtt{a}^{(n-2)}, \gamma_2) \vdash^{n-2}$
$((s_n, s'_n), \varepsilon, \gamma_n)$, where $s'_n \in F'$.
Since $\delta((\mathtt{1}, \mathtt{1}), \mathtt{a}, g_0) = ((\mathtt{a}, \pi_1(\sigma((\#, \mathtt{a}), g_0))), g_0)$ we obtain $\gamma_1 = g_0$, $s_1 = \mathtt{a}$ and $s'_1 = \pi_1(\sigma((\#, \mathtt{a}), g_0))$.
Because of the construction of δ we obtain from
$\delta(((s_t, s'_t), \mathtt{a}, \gamma_t)) = ((s_{t+1}, s'_{t+1}), \gamma_{t+1})$ for $t > 1$: $s_{t+1} = \pi_1(\sigma((s_t, s_t), \gamma_t))$, $s'_{t+1} = \pi_1(\sigma((s'_t, s_{t+1}), \gamma_{t+1}))$ and $\gamma_{t+1} = \pi_2(\sigma((s_t, s_t), \gamma_t))$.

Consequently,
$s_2 = \pi_1(\sigma((\mathtt{a}, \mathtt{a}), g_0))$, $s'_2 = \pi_1(\sigma((s'_1, s_2), \gamma_2))$ and $\gamma_2 = \pi_2(\sigma((\mathtt{a}, \mathtt{a}), g_0))$.
Therefore, M will compute the configurations as follows.

$$\begin{aligned} t = 0 : &\quad (\mathtt{a}, g_0)^n \\ t = 1 : &\quad (s'_1, \ )(s_2, \gamma_2)^{(n-1)} \\ t = 2 : &\quad (\ ,\ )(s'_2, \ )(s_3, \gamma_3)^{(n-2)} \\ &\quad \vdots \\ t = n-1 : &\quad (\ ,\ ) \cdots (s'_{n-1}, \ )(s_n, \gamma_n) \\ t = n : &\quad (\ ,\ ) \cdots (s'_n, \ ) \end{aligned}$$

Since s'_n belongs to F' the OPDCA M accepts in real-time. From '$\Longleftarrow$' and '$\Longrightarrow$' the assertion follows. □

The next lemmas show that several languages are belonging to $\mathscr{L}_{rt}(\text{OPDCA})$. Due to lack of space and for readability the presentation of proofs is somewhat informal. It takes advantage from the well-known concept of propagating pulses or signals [22] and of building the state set from some smaller state sets assuming the control units of the single cells consist of separate registers. The ith register of all cells together then are called ith *track*.

Lemma 3.2 $\{(\mathtt{ab})^{n^2} \mid n \in \mathbb{N}\} \in \mathscr{L}_{rt}(\text{OPDCA})$

Proof. The algorithm is based on the fact that the distances between two consecutive square numbers are growing by two from number to number. The leftmost cell initiates a right propagating signal at time step 1. On its way to the right it checks whether the input is of the form $(\mathtt{ab})^+$.

For the present we are assuming that at time step $t = 2k^2$, $k \geq 1$, the stack of each a-cell contains $(k+1)^2 - k^2$ symbols whereas the stack of each b-cell is empty. The automaton continually performs the following task: The stack content of each a-cell is successively transfered to the stack of the right neighboring b-cell. This is done in $(k+1)^2 - k^2$ time steps. Subsequently during another $(k+1)^2 - k^2$ time steps the stack content of each b-cell is transfered to the stack of its right neighbor (an a-cell), whereby two additional symbols are pushed. Therefore, time step $2k^2 + 2((k+1)^2 - k^2) = 2(k+1)^2$ is the first time after $t = 2k^2$ at which the stacks of the b-cells get empty. On the other hand, the OPDCA can easily be constructed such that our assumption holds for $k = 1$. A cell enters an accepting state if and only if it is a b-cell which receives the r-signal for the first time at a time step at which is stack gets empty. □

Using a similar technique we can show that there are languages in $\mathscr{L}_{rt}(\text{OPDCA})$ the words of which are characterizable by 'exponential' lengths.

Lemma 3.3 $\{(\mathtt{ab})^{2^n} \mid n \in \mathbb{N}\} \in \mathscr{L}_{rt}(\text{OPDCA})$

Proof. The proof is only a slight modification of the previous one. In this case we are assuming that at time 2^k, $k \geq 1$, the stack of each a-cell contains 2^{k-1} symbols whereas the stack of each b-cell is empty. Now the stack content of the a-cell is successively moved to the b-cells (in 2^{k-1} time steps). When the b-cells subsequently transfer their stack symbols to the a-cells (in another 2^{k-1} time steps), these push two symbols for every received symbol. Therefore, at time $2^k + 2 \cdot 2^{k-1} = 2^{k+1}$ a b-cell can enter an accepting state if it receives the r-signal for the first time. □

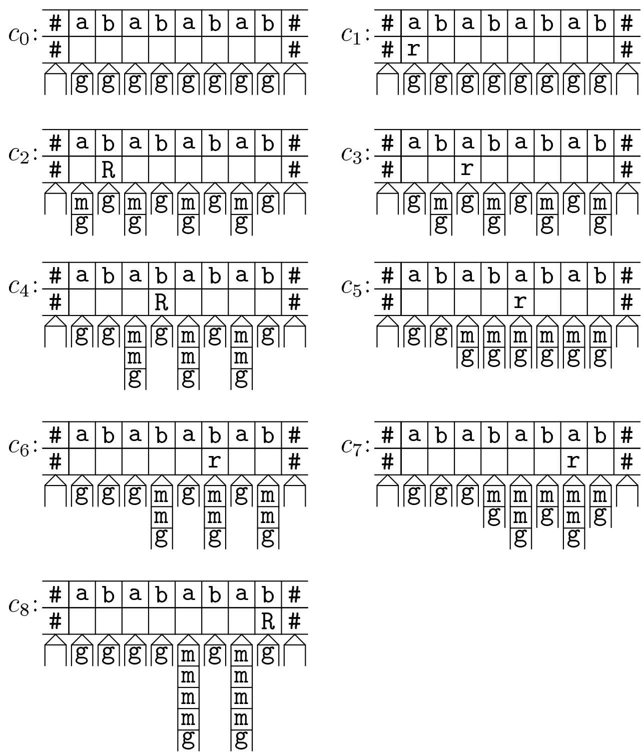

Figure 5: Excerpt of a sequence of configurations to lemma 3.3. R marks an accepting state.

Lemma 3.4 $L = \{\mathtt{a}^{2^{2^n}}\mathtt{b}^n \mid n \in \mathbb{N}\} \in \mathscr{L}_{rt}(\text{OPDCA})$

Proof. At first the construction of an OPDCA M, which accepts L in real-time is given. Subsequently we will prove its correctness.
$S := \{\mathtt{\#}, \mathtt{e}, \mathtt{1}, \mathtt{0}, *, !, \mathtt{T}, \mathtt{F}\} \times \{\mathtt{\#}, \mathtt{e}, \mathtt{pu}, \mathtt{po}\} \times \{\mathtt{\#}, \mathtt{a}, \mathtt{b}, \mathtt{r}\} \times \{\mathtt{\#}, \mathtt{e}, \mathtt{0}, \mathtt{1}, \mathtt{2}, \mathtt{3}, \mathtt{4}\}$,
$\Gamma := \{\mathtt{m}, \mathtt{g}\}$, $g_0 := \mathtt{g}$, boundary state: $(\mathtt{\#}, \mathtt{\#}, \mathtt{\#}, \mathtt{\#})$, $F := \{s \in S \mid \pi_1(s) = \mathtt{T}\}$.

Let $w = w_1 \cdots w_n$ be an input word, then we define the initial configuration for $i \in \{1, \ldots, n\}$ as $c_0(i) := ((\mathtt{e}, \mathtt{e}, w_i, \mathtt{e}), \mathtt{g})$.

The local transition function is defined as follows

$\sigma(((p_1, p_2, p_3, p_4), (q_1, q_2, q_3, q_4)), h) = ((q_1', q_2', q_3', q_4'), \gamma)$, where:

$$q_1' := \begin{cases} \mathtt{F} & \text{if } p_3 = \mathtt{b} \wedge q_3 = \mathtt{a} \\ & \quad \vee p_1 = \mathtt{F} \vee q_1 = \mathtt{F} \\ & \quad \vee p_3 = \mathtt{\#} \wedge q_3 = \mathtt{b} \\ \mathtt{1} & \text{if } q_1 = \mathtt{e} \wedge q_3 = \mathtt{b} \wedge p_3 = \mathtt{a} \\ \mathtt{0} & \text{if } q_1 \in \{*, !\} \wedge p_1 \neq \mathtt{F} \\ ! & \text{if } q_1 = \mathtt{1} \wedge p_1 \neq \mathtt{F} \wedge \\ & \quad (q_4 = \mathtt{3} \vee q_2 = \mathtt{po} \wedge p_1 = * \wedge h = \mathtt{g}) \\ * & \text{if } q_1 = \mathtt{0} \wedge p_1 \neq \mathtt{F} \wedge \\ & \quad (q_4 = \mathtt{3} \vee q_2 = \mathtt{po} \wedge p_1 = * \wedge h = \mathtt{g}) \\ \mathtt{T} & \text{if } q_1 \neq \mathtt{F} \wedge p_1 \neq \mathtt{F} \wedge \\ & \quad (q_1 = ! \wedge p_3 = \mathtt{r} \wedge q_3 = \mathtt{a} \vee q_1 = \mathtt{T}) \\ q_1 & \text{otherwise} \end{cases}$$

$$q_2' := \begin{cases} \mathtt{pu} & \text{if } q_2 = \mathtt{e} \wedge q_3 = \mathtt{b} \\ & \quad \vee p_1 \in \{*, !\} \wedge q_2 = \mathtt{po} \wedge h = \mathtt{g} \\ \mathtt{po} & \text{if } q_2 = \mathtt{pu} \wedge p_1 \in \{*, !\} \\ q_2 & \text{otherwise} \end{cases}$$

$$q_3' := \begin{cases} \mathtt{r} & \text{if } q_3 = \mathtt{r} \vee p_3 = \mathtt{\#} \\ p_3 & \text{otherwise} \end{cases}$$

$$q_4' := \begin{cases} \mathtt{0} & \text{if } q_4 = \mathtt{e} \wedge (q_3 = \mathtt{a} \vee q_3 = \mathtt{b} \wedge p_3 = \mathtt{b}) \\ \mathtt{1} & \text{if } q_4 = \mathtt{e} \wedge q_3 = \mathtt{b} \wedge p_3 = \mathtt{a} \\ & \quad \vee q_4 = \mathtt{4} \\ \mathtt{2} & \text{if } q_4 = \mathtt{1} \\ \mathtt{3} & \text{if } q_4 = \mathtt{2} \\ \mathtt{4} & \text{if } q_4 = \mathtt{3} \\ q_4 & \text{otherwise} \end{cases}$$

$$\gamma := \begin{cases} \mathtt{m}h & \text{if } q_1 \in \{\mathtt{0}, \mathtt{1}\} \wedge q_2 = \mathtt{pu} \wedge p_1 \notin \{*, !\} \wedge q_4 = \mathtt{0} \\ \varepsilon & \text{if } q_2 = \mathtt{po} \wedge p_1 \in \{*, !\} \wedge h \neq \mathtt{g} \\ h & \text{otherwise} \end{cases}$$

Observe, on the first track F is computed if the input string on the third track contains the substring ba. Subsequently the symbol F would be sent to the right border one cell per time step. Since the corresponding states are not accepting the automaton accepts input of structure $\mathtt{a}^+\mathtt{b}^*$ only.

During the computation the content of the third track is shifted to the right at each time step, whereby the leftmost cell writes an r into its third register respectively.

We assume the cells containing an initial b in their third registers are numbered $1, 2, \ldots$ from left to right.

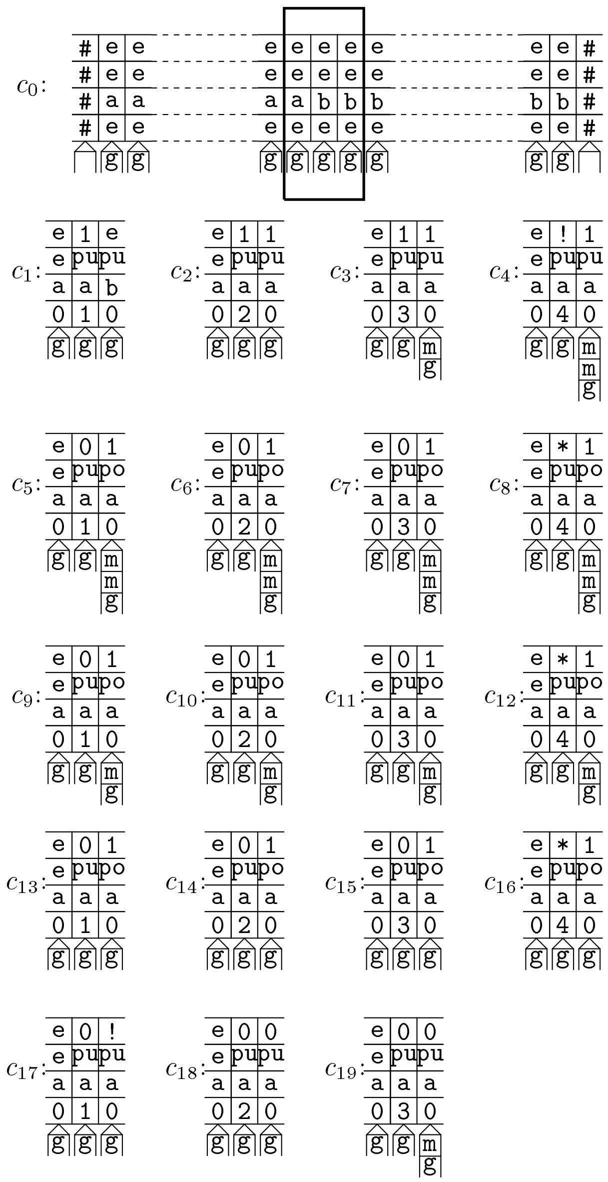

Figure 6: Excerpt of a sequence of configurations to lemma 3.4.

Cell 1 can identify itself. During the computation it cyclically writes the symbols `1`, `2`, `3` and `4` into its fourth register. At time 0 the first registers of all cells $i \geq 1$ contain the symbol `e`, which is replaced by `1` at time i. We call this the *activation* of cell i. Furthermore, at time step 1 all cells write the symbol `pu` into their second registers.

We claim: if the input string is sufficiently long, then

$$\forall n, i \in \mathbb{N} : \left(t = i - 1 + n2^{2^i} \iff \pi_1(\pi_1(c_t(i))) = \begin{cases} ! & \text{if } n = 1 \\ * & \text{if } n > 1 \end{cases} \right)$$

holds.

Proof of the claim. $i = 1$:
Cell 1 generates the symbol `1` on the first and fourth track respectively. Subsequently it writes cyclically `2`, `3`, `4` and `1` in its fourth register. At time step 4 the symbol !, at time step 5 the symbol `0` is written on the first track. During the remaining computation the first register is filled with $*$ iff the fourth one is filled with `4`.

$i \longrightarrow i + 1$:
At time $i + 1$ cell $i + 1$ is activated and $c_{i+1}(i + 1) = ((\mathtt{1}, \mathtt{pu}, \mathtt{a}, \mathtt{0}), \mathtt{g})$ holds. Subsequently at each time step a symbol `m` is additionally pushed onto the stack until cell i writes a ! or $*$ into its first register. This happens at time $i - 1 + 2^{2^i}$ at the first. Therefore, the stack of cell $i+1$ contains $i-1+2^{2^i}-(i+1) = 2^{2^i}-2$ symbols `m`.

At the next time step ($t = i + 2^{2^i}$) cell $i + 1$ writes the symbol `po` into its second register preserving the stack. From now on a stack symbol is erased iff the first register of cell i contains a $*$. This happens every 2^{2^i} time steps. After popping $2^{2^i} - 2$ symbols the stack gets empty. Subsequently cell $i + 1$ waits further 2^{2^i} time steps with an empty stack until cell i has generated another $*$, which happens at time step $i-1+2^{2^i}+(2^{2^i}-2)2^{2^i}+2^{2^i} = i-1+2^{2^{(i+1)}}$. Since now cell $i+1$ writes a ! into its first register (time $t = i + 2^{2^{(i+1)}}$) our assertion follows for $n = 1$:
$c_{i+2^{2^{(i+1)}}}(i + 1) = ((!, \mathtt{pu}, \mathtt{a}, \mathtt{0}), \mathtt{g})$ and $c_{i+1+2^{2^{(i+1)}}}(i + 1) = ((\mathtt{0}, \mathtt{pu}, \mathtt{a}, \mathtt{0}), \mathtt{g})$.
But the state differs from the state at time $i+1$ in that the symbol `0` is on the first track instead of symbol `1`. This causes the generation of ! instead of $*$, and the whole cycle will be repeated.

It remains to show how an input string is accepted. The symbol `T` is generated by one of the cells iff its first register contains a ! exactly at that time step the leftmost `a` passes through its third register. Let $\mathtt{a}^m\mathtt{b}^n$ be the input string. The symbol ! is generated by cell i at time $i-1+2^{2^i}$ once. The leftmost `a` needs $m + i - 1$ time steps to reach cell i. Hence, the input is accepted iff $m = 2^{2^i}$ and i is the number of the rightmost cell holds. □

The next lemma concerns CA, OCA and IA languages. Since it does not hold for OPDCAs it is an important tool for proving results in the next section.

Lemma 3.5 Let w be an arbitrary but fixed word over an alphabet A, $\mathtt{a}$ the symbol of a singleton and $f : \mathbb{N} \longrightarrow \mathbb{N}$ a mapping. For POLY $\in$ {CA, OCA, IA} holds
$L_1 = \{w^{f(n)} \mid n \in \mathbb{N}\} \in \mathscr{L}_{rt}(\text{POLY}) \Longrightarrow L_2 = \{\mathtt{a}^{f(n)} \mid n \in \mathbb{N}\} \in \mathscr{L}_{rt}(\text{POLY})$

Proof. Without loss of generality let $w = w_1 \cdots w_k$. From an acceptor M for L_1 we construct an acceptor M' for L_2 by splitting the finite control of each cell into k separate registers from which we are initially assume that they contain the symbols w_1 to w_k. Since originally all cells get the same input symbol we are allowed to do so. Now the recognizer for L_1 can be simulated, whereby each cell simulates k cells in its k registers. If $M = (S, \sigma, \#, F)$ is the recognizer for L_1 we (partially) construct $M' = (S', \sigma', \#', F')$ as follows: $S' := S^k$, $\#' := \#^k$, $F' := \{s' \in S' \mid \pi_k(s') \in F\}$.

Up to now we did not consider the speed of simulation. Since we have to achieve real-time but internally deal with k-fold input lengths we have to speed up the internal simulation k times. Of course, this is possible since the state after k time steps is unambiguous determined by the states of the k nearest neighbors (in any possible direction) and each cell can derive these (internal) states by inspecting the registers of its immediate neighbors. □

Since in the lemma above w is a fixed word we can also prove its reversal.

4 Relations to other language families

We are now investigating the relationships to several real-time language families and language families acceptable without any time restriction. The accepting devices are sequential machines as well as various parallel automata.

Trivially, $\mathscr{L}_{rt}(\text{OPDCA}) \subseteq \mathscr{L}_{rt}(\text{PDCA})$ holds. The next lemma shows that the inclusion is a proper one.

Theorem 4.1 $\mathscr{L}_{rt}(\text{OPDCA}) \subset \mathscr{L}_{rt}(\text{PDCA})$

Proof. It is sufficient to show that the inclusion is proper. The language $L = \{\mathtt{a}^p \mid p \in \mathbb{N} \text{ is prime}\}$ is a real-time IA language [8]. Due to $\mathscr{L}_{rt}(\text{IA}) \subset \mathscr{L}_{rt}(\text{CA})$ [4] it is a real-time CA language and for structural reasons a real-time PDCA language. From lemma 3.1 it follows that L does not belong to $\mathscr{L}_{rt}(\text{OPDCA})$ since it is not regular.

□

In the previous theorem it was shown that restricting the communication to one-way in case of pushdown cellular automata the accepting power is reduced. The next theorem shows that reducing structural properties instead (i.e. removing the stacks) would reduce the accepting power, too.

Theorem 4.2 $\mathscr{L}_{rt}(\text{OCA}) \subset \mathscr{L}_{rt}(\text{OPDCA})$

Proof. Again, it suffices to show that the inclusion is proper. The languages from lemma 3.2 and lemma 3.3 are real-time OPDCA languages. From lemma 3.5 it follows that they are not real-time OCA languages since otherwise $\{\mathtt{a}^{n^2} \mid n \in \mathbb{N}\}$ and $\{\mathtt{a}^{2^n} \mid n \in \mathbb{N}\}$ would be regular. □

Next we compare real-time OPDCAs to sequential automata.

Theorem 4.3 $\mathscr{L}_{rt}(\text{OPDCA}) \subset \mathscr{L}_1$

Proof. Let M be an OPDCA accepting a language L in real-time. Assume M is stack normalized and the cells are numbered from left to right. The proof turns on constructing a (deterministic) linear bounded automaton (LBA) B that accepts L, too. In [18, 16] it was shown that L then is a context-sensitive language.

A linear bounded automaton is simply an one-tape Turing machine the tape of which is restricted to the portion containing the input string. For a formal definition see e.g. [12, 19].

The LBA B uses four tracks; first it simulates one step of cell 1 of M, second two steps of cell 2 and so on. Obviously, it is possible to determine the OPDCA states $\pi_1(c_0(i+1)), \dots, \pi_1(c_{i+1}(i+1))$ provided its states $\pi_1(c_0(i)), \dots, \pi_1(c_i(i))$ are known.

On the first track B stores the input word, on the second it simulates the stack of the actually simulated cell. Since M is stack-normalized during the computation the content of stacks will not exceed the track capacity.

On the third track the sequence of states which would be computed by the actually simulated cell are stored. Since M works in real-time they fit onto the track.

After the LBA has computed the first i states of cell i it copies the content of the third track onto the fourth one, erases the third one and starts the simulation of cell $i+1$.

Finally, B accepts its input w if $\pi_1(c_{|w|}(|w|)) \in F$ holds, i.e. the OPDCA M would have accepted w, too.

Up to now we obtained $\mathscr{L}_{rt}(\text{OPDCA}) \subseteq \mathscr{L}_1$. But from lemma 3.1 and the proof of theorem 4.1 we know that the (deterministic) context-sensitive language $\{\mathtt{a}^p \mid p \in \mathbb{N} \text{ is prime}\}$ does not belong to $\mathscr{L}_{rt}(\text{OPDCA})$ and, hence, that the inclusion is a proper one. □

Obviously, every regular language can be accepted by an OCA in real-time. So, besides others, we have $\mathscr{L}_3 \subset \mathscr{L}_{rt}(\text{OCA}) \subset \mathscr{L}_{rt}(\text{OPDCA}) \subset \mathscr{L}_1$.

An online Turing machine (TM) consists of a control unit (i.e. a deterministic finite automaton) and a read/write head which operates on an infinite storage tape. Initially the storage tape is filled with blank symbols. At each time step the finite control fetches an input symbol, reads the symbol on the actual head position, (possibly) writes a new symbol to the actual head position, changes its internal state and moves the head one square to the right or left or not at all. An input of length n is accepted by a TM in real-time, if the machine stops after n steps in an accepting state.

Theorem 4.4 There exists a language not acceptable in real-time by any OPDCA, which is a real-time TM language.

Proof. $\{\mathtt{a}^{n^2} \mid n \in \mathbb{N}\}$ is a real-time online TM language. From lemma 3.1 it follows that L cannot be accepted by any OPDCA in real-time since it is not regular. □

Theorem 4.5 There exists a language not acceptable in real-time by any online TM, which is a real-time OPDCA language.

Proof. Hartmanis and Stearns [11] have proved that the language $\{yx\mathtt{d}zx^R \mid x \in \{\mathtt{0},\mathtt{1}\}^*, y, z \in \{\varepsilon\} \cup \{\{\mathtt{0},\mathtt{1},\mathtt{d}\}^*\mathtt{d}\}\}$ cannot be accepted in real-time by any online TM. Dyer [7] has shown that it belongs to $\mathscr{L}_{rt}(\text{OCA})$. The assertion then follows from theorem 4.2. □

Corollary 4.1 $\mathscr{L}_{rt}(\text{OPDCA})$ and $\mathscr{L}_{rt}(\text{TM})$ are incomparable.

The relationship between $\mathscr{L}_{rt}(\text{CA})$ and $\mathscr{L}_{rt}(\text{OPDCA})$ is not completely known. It is an open problem whether the inclusion of the following theorem is a proper one or whether both families are incomparable.

Theorem 4.6 There is a language not accepted in real-time by any OPDCA which is a real-time CA language.

Proof. $\{\mathtt{a}^{2^n} \mid n \in \mathbb{N}\}$ is a real-time IA language and because of $\mathscr{L}_{rt}(\text{IA}) \subset \mathscr{L}_{rt}(\text{CA})$ it is a real-time CA language [4]. Due to lemma 3.1 it does not belong to $\mathscr{L}_{rt}(\text{OPDCA})$ since it is not regular. $\square$

Since $\mathscr{L}_{rt}(\text{CA}) = \mathscr{L}_{2n}(\text{OCA})$ [4] and $\mathscr{L}_{2n}(\text{OCA}) \subseteq \mathscr{L}_{2n}(\text{OPDCA})$ we have:

Corollary 4.2 $\mathscr{L}_{rt}(\text{OPDCA}) \subset \mathscr{L}_{2n}(\text{OPDCA}) \subseteq \mathscr{L}\ (\text{OPDCA})$

Cole [6] has shown that the context-free language $\{vv' \mid v, v' \in \{\mathtt{0},\mathtt{1}\}^* \wedge v' = v'^R \wedge |v'| \geq 3\}$ is not acceptable in real-time by any n-dimensional iterative array. The generalization of iterative arrays to n dimensions is straightforward. We can utilize this result for proving:

Theorem 4.7 $\mathscr{L}_{rt}(\text{IPDA})$ and the context-free languages are incomparable.

Proof. As mentioned above $L = \{\mathtt{a}^{2^n} \mid n \in \mathbb{N}\}$ is a real-time IA and therefore a real-time IPDA language. But L is not context-free.

On the other hand we may regard an (1-dimensional) IPDA as a restricted 2-dimensional IA as follows. The finite control of cell i of the IPDA is simulated by cell $(i, 0)$ of the IA. For all $i \in \mathbb{N}$ the IA cells $(i, j), j < 0$, are idle and the IA cells $(i, j), j \geq 0$, are simulating the stack of IPDA cell i. From Cole's result we conclude that there is a context-free language not belonging to $\mathscr{L}_{rt}(\text{IPDA})$. $\square$

It should be stated that we can also prove the previous theorem by adapting Cole's argumentation directly to IPDAs.

Theorem 4.8 $\mathscr{L}_{rt}(\text{IPDA})$ and $\mathscr{L}_{rt}(\text{OCA})$ are incomparable.

Proof. Again, the real-time IPDA language $\{\mathtt{a}^{2^n} \mid n \in \mathbb{N}\}$ is not a real-time OCA language.

Dyer [7] has shown that $\{vv' \mid v, v' \in \{\mathtt{0},\mathtt{1}\}^* \wedge v' = v'^R \wedge |v'| \geq 3\} \in \mathscr{L}_{rt}(\text{OCA})$ which does not belong to n-dimensional $\mathscr{L}_{rt}(\text{IA})$. Consequently it is not real-time acceptable by IPDAs. $\square$

With lemma 3.1 and theorem 4.2 it follows immediately:

Corollary 4.3 $\mathscr{L}_{rt}(\text{IPDA})$ and $\mathscr{L}_{rt}(\text{OPDCA})$ are incomparable.

Corollary 4.4 $\mathscr{L}_{rt}(\text{IA})$ and $\mathscr{L}_{rt}(\text{OPDCA})$ are incomparable.

Theorem 4.9 $\mathscr{L}_{rt}(\text{IPDA}) \subset \mathscr{L}\ (\text{IPDA})$

Proof. If we consider iterative pushdown arrays without any time restriction arbitrary online Turing machines can be simulated. We conclude $\mathscr{L}$ (IPDA) equals the family of recursively enumerable languages. □

Since a time unrestricted PDCA with at least two cells can perform any computation of a two-stack automaton it accepts exactly the recursively enumerable languages, too.

Corollary 4.5 $\mathscr{L}$ (IPDA) $= \mathscr{L}$ (PDCA)

With corollary 4.2 we obtain $\mathscr{L}_{rt}(\text{OPDCA}) \subset \mathscr{L}$ (OPDCA) $\subseteq \mathscr{L}$ (PDCA). It is an open problem whether the second inclusion is a proper one. For structural reasons we get $\mathscr{L}$ (OCA) $\subseteq \mathscr{L}$ (OPDCA) $\subseteq \mathscr{L}$ (PDCA). From the fact $\mathscr{L}$ (OCA) $\subseteq \mathscr{L}$ (CA) $= \mathscr{L}_1$ [21] it can easily be seen that at least one of the inclusions is a proper one, but up to now we do not know which of them.

5 Closure properties

Splitting the finite control of cells into two separate registers which simulate one specific acceptor respectively, it is easily seen that $\mathscr{L}_{rt}(\text{OCA})$ is closed under intersection, union and set difference. To construct an acceptor for the complement it suffices to send a signal with suitable speed from left to right which causes the right border cell to accept if the input would not accepted and vice versa. Hence, $\mathscr{L}_{rt}(\text{OCA})$ is closed under complement. Since the two track technique is not applicable to pushdown cellular automata (we cannot simulate two stacks by just one, otherwise the context-free languages would be closed under e.g. intersection) it seems to be even hard to prove closure or non-closure under boolean operations.

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

In order to proof that $\mathscr{L}_{rt}(\text{OPDCA})$ is not closed under reversal we consider the stack depth of single cells.

Definition 5.1 Let M be an OPDCA. The *stack depth* at time t for all cells i is defined according to $sd(i,t) = |\pi_2(c_t(i))|$.
The time steps at which cell i has stack depth u are $sd^{-1}(i,u) = \{t \mid sd(i,t) = u\}$.

We are interested in the behavior of a single pushdown cell fetching a constant input. In what follows we assume OPDCAs are stack normalized.

Lemma 5.1 Let i be a single pushdown cell fetching an infinite constant input. If the stack depth of i is unbounded then $\forall u \in \mathbb{N} : |sd^{-1}(i,u)| < \infty$ holds.

Proof. Contrary to the assertion suppose there is a stack depth u which occurs infinitely often for cell i. Say, at time steps $t_{u_1} < t_{u_2} < \cdots < t_{u_r} < t_{u_{r+1}} < \cdots$. At the latest at time t_{u_r} with $u_r = |\Gamma|^u \cdot |S|$ there is a $t \in \{t_{u_1}, \ldots, t_{u_{r-1}}\}$ for which $c_t(i) = c_{t_{u_r}}(i)$ holds. Therefore, the behavior of cell i would become cyclically and, hence, the stack depth would be bounded by $\max\{sd(i,t) \mid t \leq t_{u_r}\}$ which leads to a contradiction. $\square$

Now we know that for such a cell i a stack depth u occurs lastly at a time $\max(sd^{-1}(i,u))$.

Lemma 5.2 Let $\max(sd^{-1}(i,u)) < t \leq \max(sd^{-1}(i,u+1))$, then $sd(i,t) \leq u + |S| \cdot |\Gamma|$ holds.

Proof. Contrary to the assertion assume there is a t for which $sd(i,t) > u + |\Gamma| \cdot |S|$ holds. Then for all u' with $u \leq u' \leq u + |S| \cdot |\Gamma|$ there is a maximal $t' \leq t$ for which $|\pi_2(c_{t'}(i))| = u' \wedge |\pi_2(c_{t'+1}(i))| > u'$. At constant input there are $|S| \cdot |\Gamma|$ different arguments to the local transformation and therefore, there are at least two of the time steps t' for which these arguments are equal. Since the t' were chosen maximal with respect to the stack depth, the behavior of the cell will become cyclically and the stack depth $u+1$ will not occur any more. This is a contradiction to the assertion $t \leq \max(sd^{-1}(i,u+1))$. $\square$

From the lemmas above we derive $\max(sd^{-1}(i,u+1)) - \max(sd^{-1}(i,u)) \leq |S| \cdot |\Gamma|^{|S| \cdot |\Gamma|}$. Therefore, the stack depth would cyclically grow because there are only $|S| \cdot |\Gamma|$ different constellations in which a stack depth u can occur the last time. That is starting at stack depth u the cell run through cycles in which the stack depth grow after the stack depth $u + |S| \cdot |\Gamma|$ has occured the last time.

Altogether we state that a single cell fetching a constant input will be cyclically at time step $|S| \cdot |\Gamma| \cdot |S| \cdot |\Gamma|^{|S| \cdot |\Gamma|}$ at the latest. Especially, the cycle length depends on S and Γ only.

Theorem 5.1 Let $f : \mathbb{N} \longrightarrow \mathbb{N}$ be a mapping for which

$$\lim_{n \to \infty} \frac{n^{(n+1)^2}}{f(n)} = 0,$$

holds, then $L = \{\mathtt{b}^n \mathtt{a}^{f(n)} \mid n \in \mathbb{N}\} \notin \mathscr{L}_{rt}(\text{OPDCA})$.

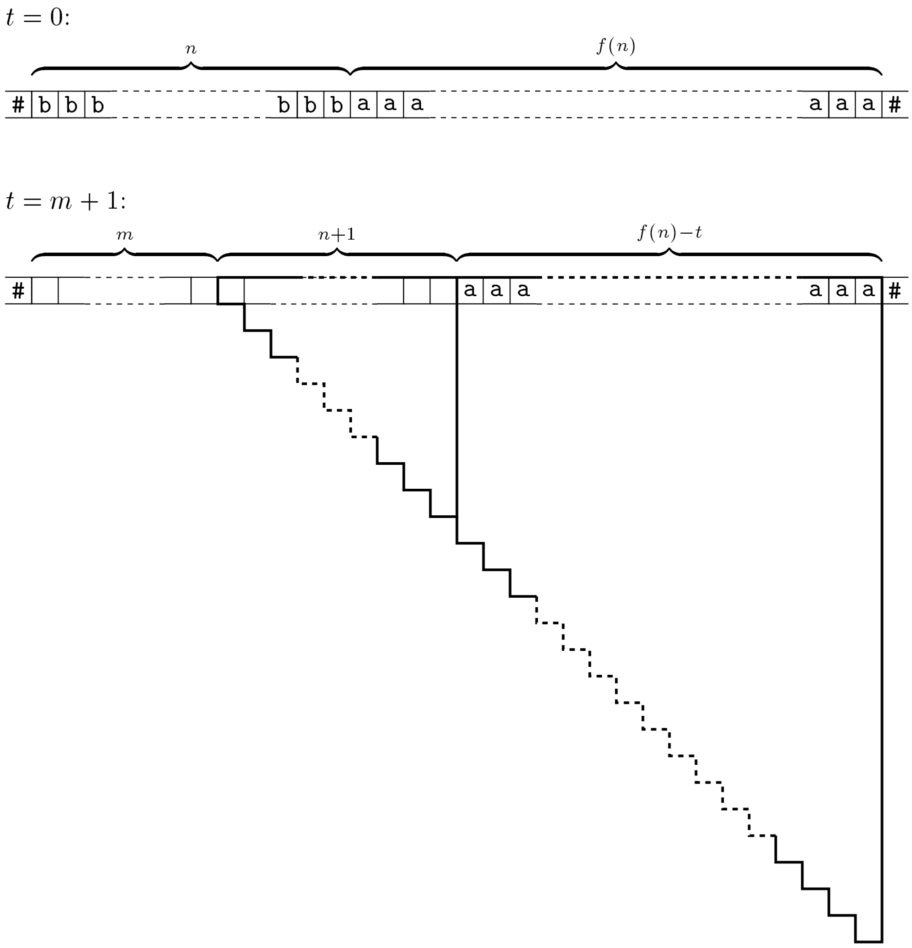

Figure 7: Configurations to the proof of theorem 5.1.

Proof. Contrary to the assertion assume there is an OPDCA M accepting L in real-time. An initial configuration and a configuration at time $t = m + 1 < n + f(n)$ are depicted in figure 7. The dark bordered areas contain the relevant information. The right $f(n) - (m+1)$ a-cells have to behave as shown above for cells with a constant input. Therefore, assuming a sufficiently long input the automata in that area will become cyclically at latest at time step $z(|S|, |\Gamma|)$, where z is a constant depending on $|S|$ and $|\Gamma|$. One can imagine the $n+1$ cells in the middle altogether form a super-cell which moves one cell to the right at each time step. The state set of the super-cell contains at most $|S|^{n+1} \cdot |\Gamma|^{n^2}$ states. Since on its way to the right the super-cell meets only cells which behave cyclically it will be cyclically at latest after $z'(n, |S|, |\Gamma|) = z(|S|, |\Gamma|) \cdot |S|^{n+1} \cdot$

$|\Gamma|^{n^2}$ time steps, too. From the assertion $\lim_{n\to\infty} \frac{n^{(n+1)^2}}{f(n)} = 0$ and

$$\begin{aligned} & z \cdot |S|^{n+1} \cdot |\Gamma|^{n^2} \leq z \cdot \max\{|S|, |\Gamma|\}^{n^2+n+1} = z \cdot \max\{|S|, |\Gamma|\}^{(n+1)^2-n} \\ & \leq z \cdot \max\{|S|, |\Gamma|\}^{(n+1)^2} \leq (z \cdot \max\{|S|, |\Gamma|\})^{(n+1)^2} \end{aligned}$$

it follows

$\forall |S|, |\Gamma| \in \mathbb{N} : \exists n \in \mathbb{N} : \forall i \in \mathbb{N} : f(n+i) > (z \cdot \max\{|S|, |\Gamma|\})^{(n+i+1)^2}$. Especially, $f(n+i) > z'(n+i, |S|, |\Gamma|)$. We conclude that for such inputs $\mathtt{b}^n\mathtt{a}^{f(n)} \in L$ the inputs $\mathtt{b}^n\mathtt{a}^{f(n)}\mathtt{a}^{z'(n,|S|,|\Gamma|)}$ would be accepted, too, from which a contradiction follows. □

Now we are prepared to prove the following.

Theorem 5.2 $\mathscr{L}_{rt}(\text{OPDCA})$ is not closed under reversal.

Proof. Due to lemma 3.4 $L = \{\mathtt{a}^{2^{2^n}}\mathtt{b}^n \mid n \in \mathbb{N}\} \in \mathscr{L}_{rt}(\text{OPDCA})$. $L^R = \{\mathtt{b}^n\mathtt{a}^{2^{2^n}} \mid n \in \mathbb{N}\} \notin \mathscr{L}_{rt}(\text{OPDCA})$ can be seen as follows: $n^{(n+1)^2} = 2^{(n+1)^2 \cdot \log_2 n}$ and $\lim_{n\to\infty} \frac{2^{(n+1)^2 \cdot \log_2 n}}{2^{2^n}} = 0$. From theorem 5.1 follows the assertion. □

Corollary 5.1 Neither $\{\mathtt{a}^{2^{2^n}}\mathtt{b}^n \mid n \in \mathbb{N}\}$ nor its reversal $\{\mathtt{b}^n\mathtt{a}^{2^{2^n}} \mid n \in \mathbb{N}\}$ are real-time OCA languages.

Now some closure properties concerning homomorphisms are shown.

Theorem 5.3 $\mathscr{L}_{rt}(\text{OPDCA})$ is not closed under ε-free homomorphism.

Proof. Due to lemma 3.2 $L = \{(\mathtt{ab})^{n^2} \mid n \in \mathbb{N}\}$ belongs to $\mathscr{L}_{rt}(\text{OPDCA})$. We define an ε-free homomorphism h as $h(\varepsilon) := \varepsilon$, $h(\mathtt{a}) := h(\mathtt{b}) := \mathtt{a}$. Under h the homomorphic image of L is $\{\mathtt{a}^{2n^2} \mid n \in \mathbb{N}\}$ which is not regular and therefore not a member of $\mathscr{L}_{rt}(\text{OPDCA})$. □

Corollary 5.2 $\mathscr{L}_{rt}(\text{OPDCA})$ is not closed under arbitrary homomorphism and substitution.

Theorem 5.4 $\mathscr{L}_{rt}(\text{OPDCA})$ is not closed under inverse homomorphism.

Proof. Define $h(\varepsilon) := \varepsilon$ and $h(\mathtt{a}) := \mathtt{ab}$. Due to lemma 3.2 $L = \{(\mathtt{ab})^{n^2} \mid n \in \mathbb{N}\} \in \mathscr{L}_{rt}(\text{OPDCA})$. But $\{\mathtt{a}^{n^2} \mid n \in \mathbb{N}\}$ the image under homomorphism h of which is L is not a real-time OPDCA language. □

If the accepting node may take notice of the time step when the acceptance should take place, the corresponding language family is closed under complement.

Theorem 5.5 Let $k \in \mathbb{N}, k \geq 1$, be a constant and $t(n) := k \cdot n$; then $\mathscr{L}_{t(n)}(\text{OPDCA})$, $\mathscr{L}_{t(n)}(\text{PDCA})$ and $\mathscr{L}_{t(n)}(\text{IPDA})$ are closed under complement.

Proof. A corresponding PDCA or OPDCA acceptor sends at initial time a signal from the left border to the accepting node the speed of which is $\frac{1}{k}$. At its arrival the automaton accepts if the acceptor for the language rejects and vice versa.

An IA acceptor at first has to store its input additionally to its basic computation in n consecutive cells. Subsequently it can send a signal from the origin to the cell containing the nth input symbol. This altogether takes $2n$ time steps. From now on the array behaves as described for CAs except that the speed of the signal is $\frac{1}{k-2}$. For the special cases $k = 1, 2$ we can provide special mechanisms the details of which are omitted. $\square$

If we consider a pushdown cellular automata language and a language not concerned with a pushdown memory acceptor the two track technique can be used to show some closures. For example $\mathscr{L}_{rt}(\text{OPDCA})$ is closed under intersection, union and set difference with real-time OCA languages. The same holds for $\mathscr{L}_{rt}(\text{IPDA})$ and real-time IA languages and $\mathscr{L}_{rt}(\text{PDCA})$ and real-time CA languages. Furthermore it holds for the linear-time language families, too. In case of $\mathscr{L}_{lt}(\text{PDCA})$ and $\mathscr{L}_{lt}(\text{IPDA})$ we can generalize the results.

Theorem 5.6 $\mathscr{L}_{lt}(\text{PDCA})$ and $\mathscr{L}_{lt}(\text{IPDA})$ are closed under intersection, union and set difference, respectively.

Proof. Closure under union: The usual technique of simulating the two computations in parallel and combining both results with logical or cannot be applied. But in linear-time it is possible to compose both computations in a sequential manner [1]. After finishing the simulation of the first acceptor the result is stored in a special register. Subsequently the second acceptor is simulated.
Closure under intersection: Because $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$ and the closure under complement and union the assertion follows.
Closure under set difference: Because $L_2 \setminus L_1 = \overline{L_2} \cap L_1$ and the closure under complement and intersection the assertion follows. $\square$

Smith [21] has shown that $\mathscr{L}_{lt}(\text{CA})$ is closed under reversal.

Theorem 5.7 $\mathscr{L}_{lt}(\text{PDCA})$ and $\mathscr{L}_{lt}(\text{IPDA})$ are closed under reversal.

Proof. A corresponding acceptor at first reverses its input resp. reads its input into n consecutive cells and simulates the acceptor for the mirror image subsequently. □

The next result excludes the simultaneous closure under two operations.

Theorem 5.8 $\mathscr{L}_{lt}(\text{OPDCA})$ and $\mathscr{L}_{rt}(\text{PDCA})$ are not simultaneous closed under homomorphism and intersection.

Proof. The Dyck languages are real-time OCA languages [7] and therefore real-time OPDCA and real-time PDCA languages. Chomsky [5] has shown that every context-free language is the homomorphic image of the intersection of a regular language and a Dyck language.

Contrary to the assertion we assume $\mathscr{L}_{lt}(\text{OPDCA})$ and $\mathscr{L}_{rt}(\text{PDCA})$ are closed under intersection and homomorphism. Since they contain the regular as well as the Dyck languages they contain the context-free languages.

Ginsburg, Greibach and Harrison [10] have shown that every recursively enumerable language is the homomorphic image of the intersection of two context-free languages. Due to our assumption all recursively enumerable languages have to be contained in $\mathscr{L}_{rt}(\text{OPDCA})$ and $\mathscr{L}_{lt}(\text{PDCA})$ from which a contradiction follows. □

Theorem 5.9 $\mathscr{L}_{lt}(\text{PDCA})$ is not closed under homomorphism.

Proof. Since $\mathscr{L}_{lt}(\text{PDCA})$ contains the regular and Dyck languages and is closed under intersection it would contain all recursively enumerable languages if it would be closed under homomorphism. □

Corollary 5.3 $\mathscr{L}_{lt}(\text{PDCA})$ is not closed under substitution.

References

[1] Bleck, B. and Kröger, H. *Cellular algorithms*. In Evans, D. (ed.), *Advances in Parallel Computing Volumne 2*. JAI Press, London, 1992, pp. 115–143.

[2] Bucher, W. and Čulik II, K. *On real time and linear time cellular automata*. RAIRO Informatique Théorique et Applications. 18 (1984), 307–325.

[3] Chang, J. H., Ibarra, O. H. and Vergis, A. *On the power of one-way communication*. Journal of the ACM 35 (1988), 697–726.

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

[5] Chomsky, N. *Context-free grammars and pushdown storage*. Technical Report QPR 65, Massachusetts Institute of Technology, 1962.

[6] Cole, S. N. *Real-time computation by n-dimensional iterative arrays of finite-state machines*. IEEE Conference Record of 7^{th} Annual Symposium on Switching and Automata Theory, 1966, pp. 53–77.

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

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

[9] Ginsburg, S. and Rice, H. G. *Two families of languages related to ALGOL*. Journal of the ACM 9 (1962), 350–371.

[10] Ginsburg, S., Greibach, S. and Harrison, M. *One-way stack automata*. Journal of the ACM 14 (1967), 389–418.

[11] Hartmanis, J. and Stearns, R. E. *On the computational complexity of algorithms*. Transactions of the AMS 117 (1965), 285–306.

[12] Hopcroft, J. and Ullman, J. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, Massachusetts, 1979.

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

[14] Ibarra, O. H. and Jiang, T. *Relating the power of cellular arrays to their closure properties*. Theoretical Computer Science 57 (1988), 225–238.

[15] Kosaraju, S. R. *On some open problems in the theory of cellular automata*. IEEE Transactions on Computers C-23 (1974), 561–565.

[16] Kuroda, S. *Classes of languages and linear bounded automata*. Information and Control 7 (1964), 207–223.

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

[18] Landweber, P. *Three theorems on phrase structure grammars of type 1*. Information and Control 6 (1963), 131–136.

[19] Salomaa, A. *Formal Languages*. Academic Press, New York, 1973.

[20] 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.

[21] Smith III, A. R. *Real-time language recognition by one-dimensional cellular automata*. Journal of Computer and System Sciences 6 (1972), 233–253.

[22] Terrier, V. *Signals in linear cellular automata*. Proceedings of a Workshop on Cellular Automata, Espoo Finland, 1991.

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

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

[25] Yamada, H. and Amoroso, S. *Tessellation automata*. Information and Control 14 (1969), 299–317.