I F I G
RESEARCH
REPORT

Institut für Informatik

Automata Arrays and Context-Free Languages

Martin Kutrib

IFIG Research Report 9907
October 1999

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

Justus-Liebig-
Universität
Giessen

IFIG Research Report
IFIG Research Report 9907, October 1999

Automata Arrays and Context-Free Languages

Martin Kutrib[1]

Institute of Informatics, University of Giessen

Arndtstr. 2, D-35392 Giessen, Germany

Abstract. From a biological point of view automata arrays have been employed by John von Neumann in order to solve the logical problem of nontrivial self-reproduction.
From a computer scientific point of view they are a model for massively parallel computing systems.
Here we are dealing with automata arrays as acceptors for formal languages. Our focus of investigations concerns their capabilities to accept the classical linguistic languages. While there are simple relations to the regular and context-sensitive ones here we shed some light on the relations to the context-free languages and some of their important subfamilies.

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

[1]E-mail: kutrib@informatik.uni-giessen.de

1 Introduction

One of the cornerstones in the theory of automata arrays is the early result of John von Neumann who solved the logical problem of nontrivial self-reproduction. From this biological point of view he employed a mathematical device which is a multitude of interconnected automata operating in parallel to form a larger automaton, a macroautomaton built by microautomata. He established that it is logically possible for such a nontrivial computing device to replicate itself ad infinitum [16].

Nowadays the so-called polyautomata theory has several branches. One important field is the modelling of natural phenomena, for example in physics, biology or chemistry. Another one deals with certain types of polyautomata as computational models, in particular as acceptors for formal languages. Those automata arrays are object of the present article.

A lot of investigations have been done in order to explore the computing capabilities of such devices but there are still some unanswered basic questions. Usually these investigations are in terms and with the methods of automata and complexity theory. We are particularly interested in the relations between some of the most important automata array language families and the classical linguistic language families. It has turned out that the interesting families include the regular languages and are contained in the deterministic context-sensitive languages such that this interest is synonymous with the interest in the relations to the context-free languages and their subfamilies.

2 Automata arrays

We denote the rational numbers by $\mathbb{Q}$, the integers by $\mathbb{Z}$, the positive integers $\{1, 2, ...\}$ by $\mathbb{N}$, the set $\mathbb{N} \cup \{0\}$ by $\mathbb{N}_0$ and the powerset of a set S by 2^S. The empty word is denoted by ε and the reversal of a word w by w^R. For the length of w we write $|w|$. We use $\subseteq$ for inclusions and $\subset$ if the inclusion is strict. For a function f we denote its i-fold composition by $f^{[i]}$, $i \in \mathbb{N}$.

The specification of an automata array includes the type and specification of the microautomata (cells), their interconnection scheme (which can imply a dimension to the system), a local transition function and the input and output modes. One-dimensional synchronous devices with nearest neighbor connections whose cells are (nondeterministic) finite automata are commonly called *cellular arrays* ((N)CA) resp. *iterative arrays* ((N)IA) in case of parallel resp. sequential input mode. A connection to the nearest neighbor to the right only yields one-way information flow through the array. The corresponding devices are denoted by (N)OCAs resp. (N)OIAs.

For convenience we identify the cells of one-dimensional arrays by positive integers. The state transition depends on the current state of a cell and the current state(s) of its neighbor(s). The transition function is applied to all cells synchronously at discrete time steps. With an eye towards formal language processing we define formally:

Definition 1 *A* nondeterministic (two-way) cellular array *(*NCA*) is a system* $\langle S, \delta_{nd}, \#, A, F \rangle$*, where*

1. S *is the finite, nonempty set of* cell states,
2. $A \subseteq S$ *is the nonempty set of* input symbols,
3. $F \subseteq S$ *is the nonempty set of* accepting states,
4. $\# \notin S$ *is the* boundary state,
5. $\delta_{nd} : \left(S \cup \{\#\}\right)^3 \to 2^S \setminus \{\emptyset\}$ *is the (nondeterministic)* local transition function.

If the state set is a Cartesian product of some smaller sets $S = S_1 \times S_2 \times \cdots \times S_k$ we will use the notion *register* for the single parts of a state. The concatenation of one of the registers of all cells respectively forms a *track*.

Let $\mathcal{M} = \langle S, \delta_{nd}, \#, A, F \rangle$ be an NCA. A *configuration* of $\mathcal{M}$ at some time $t \geq 0$ is a description of its global state which is actually a mapping $c_t : [1, \dots, n] \to S$ for $n \in \mathbb{N}$. The configuration at time 0 is defined by the initial sequence of states. For a given input word $w = a_1 \cdots a_n \in A^+$ we set $c_{0,w}(i) := a_i$, $1 \leq i \leq n$. During its course of computation an NCA steps nondeterministically through a sequence of configurations whereby successor configurations are chosen according to the global transition function Δ_{nd}:

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

$$
\begin{aligned}
& c_{t+1} \in \Delta_{nd}(c_t) \iff \\
& \qquad c_{t+1}(1) \in \delta_{nd}\big(\#, c_t(1), c_t(2)\big) \\
& \qquad c_{t+1}(i) \in \delta_{nd}\big(c_t(i-1), c_t(i), c_t(i+1)\big), i \in \{2, \dots, n-1\} \\
& \qquad c_{t+1}(n) \in \delta_{nd}\big(c_t(n-1), c_t(n), \#\big)
\end{aligned}
$$

Thus, Δ_{nd} is induced by δ_{nd}.

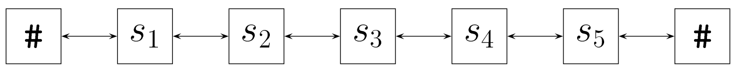


Figure 1: A (two-way) cellular array.

If the flow of information is restricted to one-way, the resulting device is a *nondeterministic one-way cellular array* (NOCA). I.e. the next state of each cell depends on the state of the cell itself and the state of its immediate neighbor to the right.

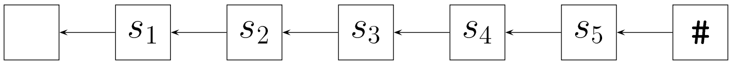


Figure 2: A one-way cellular array.

An NCA (NOCA) is *deterministic* if $\delta_{nd}(s_1, s_2, s_3)$ $(\delta_{nd}(s_1, s_2))$ is a singleton for all states $s_1, s_2, s_3 \in S \cup \{\#\}$. Deterministic cellular arrays are denoted by CA resp. OCA.

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

Definition 2 *Let $\mathcal{M} = \langle S, \delta_{nd}, \#, A, F \rangle$ be an* NCA *or an* NOCA*.*

1. *A word $w \in A^+$ is accepted by $\mathcal{M}$ if there exists a time step $i \in \mathbb{N}$ such that $c_i(1) \in F$ holds for a configuration $c_i \in \Delta_{nd}^{[i]}(c_{0,w})$.*
2. *$L(\mathcal{M}) = \{w \in A^+ \mid w$ is accepted by $\mathcal{M}\}$ is the language accepted by $\mathcal{M}$.*
3. *Let $t : \mathbb{N} \to \mathbb{N}$, $t(n) \geq n$, be a mapping and i_w be the minimal time step at which $\mathcal{M}$ accepts a $w \in L(\mathcal{M})$. If all $w \in L(\mathcal{M})$ are accepted within $i_w \leq t(|w|)$ time steps, then L is said to be of* time complexity t*.*

The family of all languages acceptable by NCAs (NOCAs) with time complexity t is denoted by $\mathscr{L}_t(\text{NCA})$ $(\mathscr{L}_t(\text{NOCA}))$. If t equals the *identity function* $id(n) = n$, acceptance is said to be in *real-time* and we write $\mathscr{L}_{rt}(\text{NCA})$ $(\mathscr{L}_{rt}(\text{NOCA}))$. In the sequel we will use a corresponding notion for other types of acceptors. The *linear-time* languages $\mathscr{L}_{lt}(\text{NCA})$ $(\mathscr{L}_{lt}(\text{NOCA}))$ are defined according to $\mathscr{L}_{lt}(\text{NCA}) := \bigcup_{k \in \mathbb{Q},\, k \geq 1} \mathscr{L}_{k \cdot id}(\text{NCA})$.

There is a natural way to restrict the nondeterminism of the arrays. One can limit the number of allowed nondeterministic state transitions. For this purpose a *deterministic local transition* $\delta_d : \left(S \cup \{\#\}\right)^3 \to S$ is provided and the global transition induced by δ_d is denoted by Δ_d.

Let $g : \mathbb{N} \to \mathbb{N}_0$ be a mapping that gives the number of allowed nondeterministic transitions dependent on the length of the input. The resulting system $\langle S, \delta_{nd}, \delta_d, s_0, \#, A, F \rangle$ is a gG-CA (gG-OCA) if starting with the initial configuration $c_{0,w}$ (for some $w \in A^+$) the possible configurations at some time i are computed by the global transition as follows:

$$
\begin{cases}
\{c_{0,w}\} & \text{if } i = 0 \\
\Delta_{nd}^{[i]}(c_{0,w}) & \text{if } i \leq g(|w|) \\
\displaystyle\bigcup_{c' \in \Delta_{nd}^{[g(|w|)]}(c_{0,w})} \Delta_d^{[t-g(|w|)]}(c') & \text{otherwise}
\end{cases}
$$

Observe that all nondeterministic transitions have to be applied before the deterministic ones and that for some $s_1, s_2, s_3 \in S \cup \{\#\}$ the nondeterministic part may contain the deterministic one: $\delta_d(s_1, s_2, s_3) \in \delta_{nd}(s_1, s_2, s_3)$.

In cellular arrays the input mode is called parallel. One can suppose that all cells fetch their input symbol during a pre-initial step. Another natural way to supply the input is the so-called sequential input mode. Now the distinguished cell at the origin, the so-called *communication cell*, fetches the input sequence symbol by symbol. In order to define such iterative arrays formally we have to provide an initial (quiescent) state for the cells. Moreover, due to historical reasons the space (number of cells) of iterative arrays is not bounded. We assume that once the whole input is consumed an end-of-input symbol is supplied permanently.

Definition 3 *An iterative array* (IA) *is a system* $\langle S, \delta, \delta_0, s_0, \#, A, F\rangle$, *where*

1. *S is the finite, nonempty set of* cell states,
2. *A is the finite, nonempty set of* input symbols,
3. *$F \subseteq S$ is the set of* accepting states,
4. *$s_0 \in S$ is the* initial (quiescent) state,
5. *$\# \notin A$ is the* end-of-input symbol,
6. *$\delta : S^3 \to S$ is the* local transition function for non-communication cells *satisfying* $\delta(s_0, s_0, s_0) = s_0$,
7. *$\delta_0 : S^3 \times (A \cup \{\#\}) \to S$ is the* local transition function for the communication cell.

Here iterative arrays are defined as deterministic devices. The reason is that real- resp. linear-time nondeterministic IAs have exactly the same accepting power as real- resp. linear-time nondeterministic CAs.

A configuration of an IA at some time $t \geq 0$ is a pair (w_t, c_t) where $w_t \in A^*$ is the remaining input sequence and $c_t : \mathbb{Z} \to S$ is a function that maps the single cells to their current states. The configuration (w_0, c_0) at time 0 is defined by the input word w_0 and the mapping $c_0(i) = s_0$, $i \in \mathbb{Z}$, while subsequent configurations are chosen according to the global transition function Δ: Let (w_t, c_t), $t \geq 0$, be a configuration then its successor configuration (w_{t+1}, c_{t+1}) is as follows:

$$\begin{aligned}(w_{t+1}, c_{t+1}) = \Delta\big((w_t, c_t)\big) \iff & \\ & c_{t+1}(i) = \delta\big(c_t(i-1), c_t(i), c_t(i+1)\big), i \in \mathbb{Z} \setminus \{0\}, \\ & c_{t+1}(0) = \delta_0\big(c_t(-1), c_t(0), c_t(1), a\big)\end{aligned}$$

where $a = \#$, $w_{t+1} = \varepsilon$ if $w_t = \varepsilon$, and $a = a_1$, $w_{t+1} = a_2 \cdots a_n$ if $w_t = a_1 \cdots a_n$. Thus, the global transition function Δ is induced by δ and δ_0.

An input word w is accepted by an IA if at some time i during its course of computation the communication cell becomes accepting.

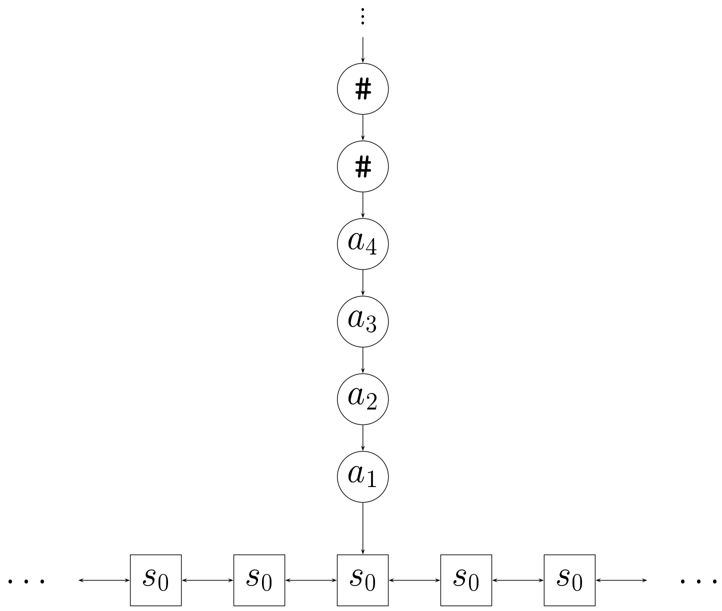


Figure 3: An iterative array.

Definition 4 *Let* $\mathcal{M} = \langle S, \delta, \delta_0, s_0, \#, A, F\rangle$ *be an* IA.

1. *A word* $w \in A^*$ *is* accepted *by* $\mathcal{M}$ *if there exists a time step* $i \in \mathbb{N}$ *such that* $c_i(0) \in F$ *holds for* $(w_i, c_i) = \Delta^{[i]}((w, c_0))$.
2. $L(\mathcal{M}) = \{w \in A^* \mid w$ *is accepted by* $\mathcal{M}\}$ *is the* language accepted *by* $\mathcal{M}$.
3. *Let* $t : \mathbb{N}_0 \to \mathbb{N}$, $t(n) \geq n+1$, *be a mapping and* i_w *be the minimal time step at which* $\mathcal{M}$ *accepts a* $w \in L(\mathcal{M})$. *If all* $w \in L(\mathcal{M})$ *are accepted within* $i_w \leq t(|w|)$ *time steps, then* L *is said to be of* time complexity t.

Since for nontrivial computations an IA needs to read at least one end-of-input symbol we define the time complexity $n+1$ to be real-time. Again, we denote the family of all languages acceptable by IAs with time complexity t by $\mathscr{L}_t(\text{IA})$.

Throughout the article the regular, deterministic context-free, context-free resp. deterministic context-sensitive languages are denoted by $\mathscr{L}_3$, $\mathscr{L}_{2,det}$, $\mathscr{L}_2$ resp. $\mathscr{L}_{1,det}$.

3 Relations between automata arrays and the position of $\mathscr{L}_2$

In order to compare the context-free languages and some of their important subclasses to the language families definable by time-bounded automata arrays it is convenient to recall some of the known results. Figure 4 summarizes inclusions between real-time and unlimited-time (O)CAs, 1G-OCAs, IAs and regular languages.

$$\begin{tikzcd}[column sep=small]
 & \mathscr{L}_{rt}(\mathrm{IA}) \arrow[dr, phantom, sloped, "\subset"] & \\
\mathscr{L}_3 \arrow[ur, phantom, sloped, "\subset"] \arrow[dr, phantom, sloped, "\subset"] & & \mathscr{L}_{rt}(\mathrm{CA}) \subseteq \mathscr{L}_{rt}(\text{1G-OCA}) \subseteq \mathscr{L}(\mathrm{OCA}) \subseteq \mathscr{L}(\mathrm{CA}) \\
 & \mathscr{L}_{rt}(\mathrm{OCA}) \arrow[ur, phantom, sloped, "\subset"] &
\end{tikzcd}$$

Figure 4: Hierarchy of language families.

At the left end of the hierarchy are the regular languages, which are strictly included in all of the considered array language families. It is known that the families $\mathscr{L}_{rt}(\mathrm{OCA})$ and $\mathscr{L}_{rt}(\mathrm{IA})$ are incomparable but both are strictly included in the real-time CA languages. Unfortunately, these are the only inclusions known to be strict. The right hand end of the (shown part of the) hierarchy is marked by the unlimited-time CA languages. These are in turn identical to the deterministic context-sensitive languages. Thus, we have a hierarchy between the linguistic families $\mathscr{L}_3$ and $\mathscr{L}_{1,det}$ and, naturally, the question for the position of $\mathscr{L}_2$ arises.

Since $\mathscr{L}_{rt}(\mathrm{IA})$ as well as $\mathscr{L}_{rt}(\mathrm{OCA})$ are containing noncontext-free languages the strict inclusion

$$\mathscr{L}_2 \subset \mathscr{L}(\mathrm{CA})$$

follows immediately.

In 1988 the position could be shifted downwards to the unlimited-time one-way languages. The inclusion

$$\mathscr{L}_2 \subset \mathscr{L}(\mathrm{OCA})$$

has been shown in [3].

Due to the hierarchical result $\mathscr{L}_{rt}(\text{1G-OCA}) \subseteq \mathscr{L}(\mathrm{OCA})$ [1] the following theorem is a further improvement.

Theorem 5 $\mathscr{L}_2 \subset \mathscr{L}_{rt}(\text{1G-OCA})$

Proof. From [1] the equality $\mathscr{L}_{rt}(\text{1G-OCA}) = \mathscr{L}_{rt}(\text{1G-CA})$ is known. Therefore, we may prove the theorem by showing how an 1G-CA can simulate an ε-free nondeterministic pushdown automaton (pda) in real-time.

The leftmost cell of the 1G-CA can identify itself (by the # on its left). It simulates the state transitions of the pda.

During the first time step all the other cells are guessing a state transition of the pda nondeterministically. These guesses as well as the input symbols are successively shifted to the left on some additional tracks. Thereby at every time step the leftmost cell receives an input symbol and a previously guessed state transition. Thus, simply by applying the transition to the current situation it simulates a nondeterministic transition of the pda deterministically. It remains to show how to simulate the stack.

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

Each cell has three registers that can store one stack symbol respectively. They are numbered 1, 2 and 3 downward the stack. The third register is used as a buffer. The cells preferably fill two of their registers with symbols. In order to reach that charge they behave according to the following rules (cf. Figure 5):

Figure 5: Example for pushdown store simulation by a two-way device.

• If all three registers of the left (upward) neighbor of a cell are filled, then it stores the symbol from the buffer register of the neighbor in its own first register. The content of the first and second register is shifted to the second resp. third one. Otherwise the content of the buffer is deleted.

• If the buffer of the left neighbor of a cell is empty, its own second register is empty and the first register of its right neighbor is filled, then it stores the symbol from the first register of the right neighbor in its own second register.

• If just one register of the left neighbor of a cell is filled, then the symbol in the cell's own first register is deleted, whereby the content of the second register is shifted to the first one.

Eventually, the actions have to be superimposed and it can easily be verified that the simulation works correctly. Thus, as shown 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. □

The next step of positioning the family $\mathscr{L}_2$ would be to prove or disprove the inclusion $\mathscr{L}_2 \subset \mathscr{L}_{rt}(\mathrm{CA})$. The problem whether or not the real-time CA languages are containing the context-free ones was raised in [13] and is still open. It is related to the open question whether or not sequential one-tape Turing machines are able to accept the context-free languages in square-time. A proof for the inclusion would imply the existence of square-time Turing machines.

4 Iterative arrays

Now we turn to the comparison of $\mathscr{L}_2$ and the language families at the lower end of the hierarchy. The present section is devoted to real-time iterative arrays. (In case of linear-time the accepting capabilities of IAs and CAs are identical.) In [9] it has been proved that iterative arrays can accept the context-free languages in square-time:

$$\mathscr{L}_2 \subset \mathscr{L}_{n^2}(\mathrm{IA})$$

In the same paper it has been shown that the time complexity can be reduced to linear-time if the cells are arranged in two dimensions:

$$\mathscr{L}_2 \subset \mathscr{L}_{lt}(\text{2D-IA})$$

About seventeen years later the second result could be improved to one-way information flow [2]:

$$\mathscr{L}_2 \subset \mathscr{L}_{lt}(\text{2D-OIA})$$

Moreover, the corresponding 2D-OIAs are not only acceptors but parsers for the context-free languages.

With the help of a specific witness language the incomparability of $\mathscr{L}_2$ and $\mathscr{L}_{rt}(\mathrm{IA})$ has been established in [5]. In the following we are going to explore the boundary between acceptable and nonacceptable context-free languages in real-time IAs.

Lemma 6 *Every ε-free (real-time) deterministic context-free language belongs to $\mathscr{L}_{rt}(\mathrm{IA})$.*

Proof. Using the method referred to in the proof of Theorem 5 one can construct an iterative array that simulates a deterministic pushdown automaton. The communication cell controls the stack, fetches the input and simulates the state transition.

Since per assumption the pda performs no ε-transitions the IA operates in real-time. $\square$

Once the stack simulation principle was known the previous lemma became straightforward. But nevertheless it seems to mark a sharp boundary what becomes clear by the next theorem.

Theorem 7 *There exists a deterministic, linear context-free language that does not belong to* $\mathscr{L}_{rt}(\text{IA})$.

Proof. We are considering the configuration of an IA $\langle S, \delta, \delta_0, s_0, \#, A, F\rangle$ at time $n-m$ where n denotes the length of the input and $m < n$. The remaining computation of the communication cell depends on the last m input symbols and the states of the cells $-m-1, \ldots, 0, \ldots, m+1$ only. Due to their distance from the communication cell all the other cells cannot influence the overall computation result. So we are concerned with at most $|S|^{2\cdot(m+1)+1}$ different situations.

Now let $L = \{\$x_k\$\cdots\$x_1¢y_1\$\cdots\$y_k\$ \mid x_i^R = y_iz_i, x_i, y_i, z_i \in \{0,1\}^*\}$. Obviously L is a deterministic, linear context-free language.

For each different two prefix-words w and w' of the form $\$x_k\$\cdots\$x_1¢$ with $|x_i| = k$ there exists at least one $1 \le j \le k$ such that $x_j \ne x'_j$. It follows $w\$^{j-1}x_j^R\$^{k-j+1} \in L$ and $w'\$^{j-1}x_j^R\$^{k-j+1} \notin L$. There exist 2^{k^2} different prefix-words of the given form.

On the other hand, at time $n-2k$ the IA can distinguish at most $|S|^{2\cdot(2k+1)+1} = |S|^{4k+3}$ different situations. For k large enough we can conclude $|S|^{4k+3} < 2^{k^2}$ and, thus, after processing two different words w and w' the situation must be identical. We obtain $w\$^{j-1}x_j^R\$^{k-j+1} \in L \iff w'\$^{j-1}x_j^R\$^{k-j+1} \in L$, a contradiction. $\square$

By the previous theorem neither the deterministic nor the linear context-free languages are subfamilies of $\mathscr{L}_{rt}(\text{IA})$.

5 One-way cellular arrays

Now we are going to investigate the relations between real-time OCAs and some important subfamilies of $\mathscr{L}_2$. The presented results will prove the incomparability of $\mathscr{L}_{rt}(\text{OCA})$ and $\mathscr{L}_{rt}(\text{IA})$ in terms of context-free languages.

The reason why we restrict our considerations to real-time OCAs is due to the known relation $\mathscr{L}_{lt}(\text{OCA}) = \mathscr{L}^R(\text{CA})$ [4, 15] and the fact that the real-time CAs are object of the next section.

Some subfamilies of $\mathscr{L}_2$ are known to be strictly included in $\mathscr{L}_{rt}(\text{OCA})$. E.g. the Dyck languages [11] and the bracketed context-free languages [6]. In the following we deal with linear languages. In [12] it has been shown that they are acceptable by OCAs in real-time.

Theorem 8 *Every linear context-free language belongs to* $\mathscr{L}_{rt}(\text{OCA})$.

Proof. Let $G = (N, T, X_0, P)$ be a linear context-free grammar ([10]). W.l.o.g. we may assume that all productions are of the form $X \longrightarrow a$, $X \longrightarrow Ya$ or $X \longrightarrow aY$, where $X, Y \in N$ are nonterminals and $a \in T$ is a terminal.

Let $w = a_1 a_2 \cdots a_n$ be a word in $L(G)$. If $X \Rightarrow^* a_i \cdots a_j$ then there must exist a Y such that either $Y \Rightarrow^* a_i \cdots a_{j-1} \wedge X \Rightarrow Ya_j$ or $Y \Rightarrow^* a_{i+1} \cdots a_j \wedge X \Rightarrow a_i Y$. Basing on this fact we define sets of nonterminals for the given word w as follows:

$N(i,i) = \{X \in N \mid (X \to a_i) \in P\}, 1 \leq i \leq n$
$N(i,j) = N_1(i,j) \cup N_2(i,j), 1 \leq i < j \leq n$, where
$N_1(i,j) = \{X \in N \mid (X \to Ya_j) \in P \wedge Y \in N(i, j-1)\}$
$N_2(i,j) = \{X \in N \mid (X \to a_i Y) \in P \wedge Y \in N(i+1, j)\}$.

It holds $w \in L(G) \iff X_0 \in N(1,n)$.

A corresponding OCA behaves as follows. During the first transition a cell i computes $N(i,i)$ and (a_i, a_i). In subsequent transitions, say at time $k+1$, it computes $N(i, i+k)$ and (a_i, a_{i+k}) if $k+1 \geq i$ and keeps its state otherwise. The new values can be determined by the state of the cell itself and the state of its right neighbor (i.e. $N(i, i+k-1)$, (a_i, a_{i+k-1}) and $N(i+1, i+k)$, (a_{i+1}, a_{i+k})). Thus, the leftmost cell computes $N(1,n)$ at time step n. □

With Theorem 7 we can conclude that there exists a language in $\mathscr{L}_{rt}(\text{OCA}) \setminus \mathscr{L}_{rt}(\text{IA})$. Again, the question arises whether we can relax the condition of Theorem 8 to larger subfamilies. An answer follows from the results given in [14] where by counting arguments the closure of $\mathscr{L}_{rt}(\text{OCA})$ under concatenation has been shown to be negative. In the proof the language $L = \{0^i 1^i \mid i \in \mathbb{N}\} \cup \{0^i 1x01^i \mid x \in \{0,1\}^*, i \in \mathbb{N}\}$ has been used as witness. Clearly, L is a linear context-free language and, thus, belongs to $\mathscr{L}_{rt}(\text{OCA})$. But for the concatenation $L_1 = L \cdot L$ it holds $L_1 \notin \mathscr{L}_{rt}(\text{OCA})$. Since L_1 is the concatenation of two linear languages it is a 2-linear language.

Corollary 9 *There exists a 2-linear context-free language that does not belong to* $\mathscr{L}_{rt}(\text{OCA})$.

On the other hand, one can show $L_1 \in \mathscr{L}_{rt}(\text{IA})$ such that there exists a context-free language in $\mathscr{L}_{rt}(\text{IA}) \setminus \mathscr{L}_{rt}(\text{OCA})$ and, hence, with the previous results the incomparability of $\mathscr{L}_{rt}(\text{IA})$ and $\mathscr{L}_{rt}(\text{OCA})$ follows in terms of context-free languages.

Although $\mathscr{L}_{rt}(\text{OCA})$ does not contain the 2-linear languages, real-time OCAs are powerful devices. For example, the non-Parikh-linear language $\{(0^i 1)^* \mid i \in \mathbb{N}_0\}$ [11] and the inherently ambiguous language $\{0^i 1^j 2^k \mid i = j \vee j = k,\ i, j, k \in \mathbb{N}\}$ [13] are belonging to $\mathscr{L}_{rt}(\text{OCA})$.

6 Two-way cellular arrays

For structural reasons $\mathscr{L}_{rt}(\mathrm{IA}) \subseteq \mathscr{L}_{rt}(\mathrm{CA})$ and $\mathscr{L}_{rt}(\mathrm{OCA}) \subseteq \mathscr{L}_{rt}(\mathrm{CA})$ holds. Since $\mathscr{L}_{rt}(\mathrm{IA})$ and $\mathscr{L}_{rt}(\mathrm{OCA})$ are incomparable both inclusions must be strict. Unfortunately, this proof of the strictness, although given in terms of context-free languages, gives us no context-free subfamily contained in $\mathscr{L}_{rt}(\mathrm{CA})$.

Theorem 10 $\mathscr{L}_{2,det} \subset \mathscr{L}_{rt}(\mathrm{CA})$

Proof. Here we cannot use a stack simulation as in the proof of Lemma 6 because we are concerned with deterministic pushdown automata that are allowed to perform ε-transitions. But w.l.o.g. we may assume that a given deterministic pda $\mathcal{M}$ pushes at most $k \in \mathbb{N}$ symbols onto the stack at every non-ε-transition and erases exactly one symbol from the stack at every ε-transition [7]. Moreover, the first transition is a non-ε-transition.

Due to the equality $\mathscr{L}_{lt}(\mathrm{OCA}) = \mathscr{L}_{rt}^{R}(\mathrm{CA})$ it suffices to construct a $\big((k+1)\cdot n\big)$-time OCA $\mathcal{M}'$ that accepts the language $L^R(\mathcal{M})$.

Therefore, let S denote the state set of the pda $\mathcal{M}$, let G denote its set of stack symbols, A its set of input symbols, s_0 its initial state, $\perp \in G$ its bottom-of-stack symbol, F its set of accepting states and $\delta : S \times G \times \big(A \cup \{\varepsilon\}\big) \to S \times G^*$ its transition function.

Now define the OCA $\mathcal{M}' = \langle S', \delta', \#, A, F' \rangle$ as follows:

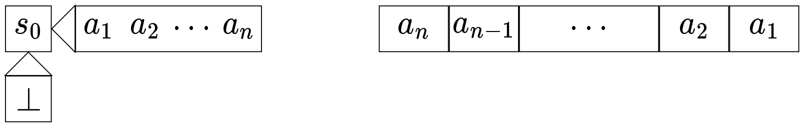
Each cell of $\mathcal{M}$ has $k+2$ registers: k registers for stack symbols of $\mathcal{M}$ that may be empty, one register for a finite counter and another one that can store an input symbol, a distinguished special symbol € or a state of $\mathcal{M}$. Accordingly, S' is defined to be $\big(S \cup A \cup \{€\}\big) \times \big\{0, \ldots, k\big\} \times \big(G^k \cup \{\varepsilon\}\big)$. δ' will be designed in such a way that at every time step at most one of the cells contains a symbol from S in its corresponding register. This symbol is the currently simulated state of $\mathcal{M}$. Accordingly, F' is defined to be $F \times \big\{0, \ldots, k\big\} \times \big(G^k \cup \{\varepsilon\}\big)$.

Let $a_1 \cdots a_n$ be an input of $\mathcal{M}$ then $\mathcal{M}'$ gets the reversal $a_n \cdots a_1$ as input. Initially the counter registers of $\mathcal{M}'$ are set to 0 and the k registers for stack symbols are empty.

Since the first transition of $\mathcal{M}'$ has to be a non-ε-transition and the rightmost cell of $\mathcal{M}'$ can identify itself, we define (cf. Figure 6):

$$\begin{aligned}
&\forall\, a \in A: \\
&\qquad \delta'\big((a, 0, \varepsilon), \#\big) = (s', 0, g_1 \cdots g_p \perp^i), \text{ where } i = k - p \\
&\qquad \text{iff } \delta(s_0, \perp, a) = (s', g_1 \cdots g_p)
\end{aligned}$$

From now on at every time step the cells of $\mathcal{M}'$ are divided into three segments. Exactly one of the cells has stored a symbol from S in its corresponding register.



$\delta(s_0, \perp, a_1) = (s', g_1 \cdots g_p)$

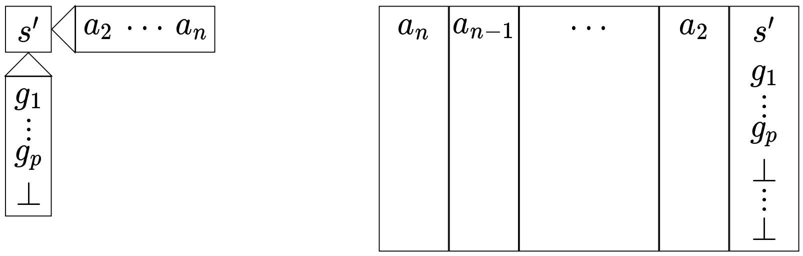


Figure 6: The initial step of the pda $\mathcal{M}$ (left) and the corresponding step of the OCA $\mathcal{M}'$ (right). Counters are not depicted.

The second segment is formed by the cells to the right of that cell. They are marked by the special symbol €. The cells to the left are the third segment. These are in their initial states (cf. Figures 6,7,8).

Every simulation of a transition of $\mathcal{M}$ takes two phases. During the first phase, which takes one time step only, the new current state and top-of-stack content of $\mathcal{M}$ is computed. (The first phase is indicated by 0 in the counter registers.) If $\mathcal{M}$ would perform a non-ε-transition (cf. Figure 7) the cell bordering the distinguished cell at the left hand side has the necessary information in order to perform the computation. We define:

$$\begin{aligned}
&\forall\, a \in A,\ s \in S,\ g_j \in G: \\
&\qquad \delta'\big((a, 0, \varepsilon), (s, 0, g_1 \cdots g_k)\big) = (s', i, g'_1 \cdots g'_p \perp^i),\ \text{where } i = k - p \\
&\qquad \text{iff}\ \ \delta(s, g_1, a) = (s', g'_1 \cdots g'_p)
\end{aligned}$$

All the other cells in the left segment keep their states:

$$\forall\, a, \tilde{a} \in A: \quad \delta'\big((a, 0, \varepsilon), (\tilde{a}, 0, \varepsilon)\big) = (a, 0, \varepsilon)$$

The distinguished cell observes that $\mathcal{M}$ would not perform an ε-transition. It changes to the special symbol €:

$\forall\, s \in S:$

$$\delta'\big((s, 0, g_1 \cdots g_k), \texttt{\#}\big) = (€, 0, g_2 \cdots g_k \perp) \quad \text{and}$$
$$\delta'\big((s, 0, g_1 \cdots g_k), (€, 0, g_{k+1} \cdots g_{2k})\big) = (€, 0, g_2 \cdots g_k g_{k+1})$$
$$\text{iff} \quad \delta(s, g_1, a) \text{ is defined for some } a \in A$$

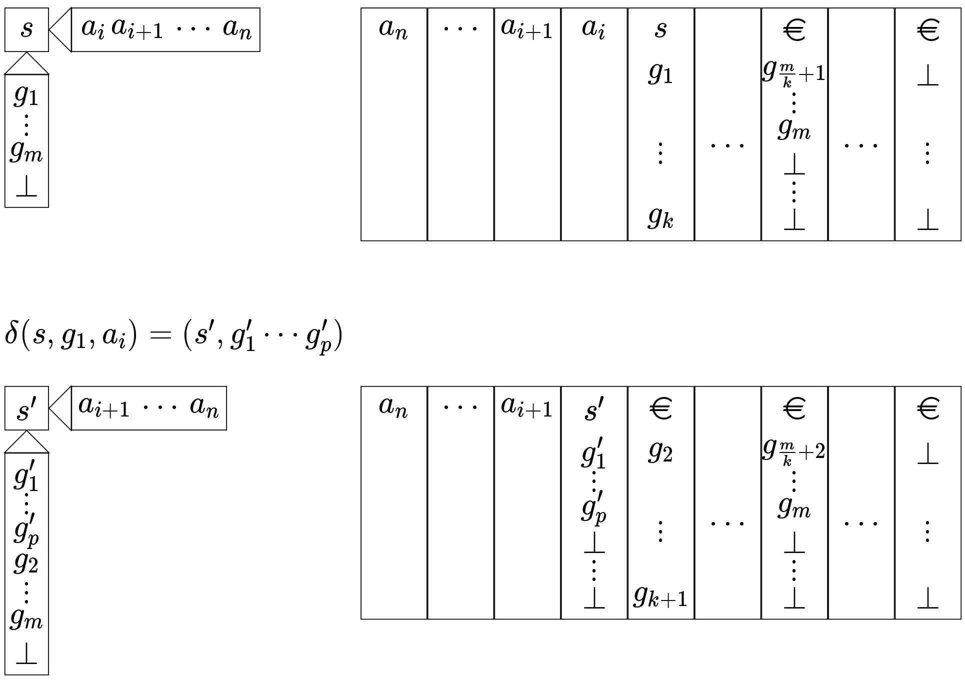


Figure 7: A non-ε-transition of the pda $\mathcal{M}$ (left) and the corresponding transition of the OCA $\mathcal{M}'$ (right). Counters are not depicted.

All cells that contain the special symbol € shift their contents of the k stack symbol registers at every time step by one register. The last register is filled with the symbol shifted out by the right neighbor.

$\forall\, g_j \in G:$

$$\delta'\big((€, 0, g_1 \cdots g_k), \texttt{\#}\big) = (€, 0, g_2 \cdots g_k \perp) \quad \text{and}$$
$$\delta'\big((€, 0, g_1 \cdots g_k), (€, 0, g_{k+1} \cdots g_{2k})\big) = (€, 0, g_2 \cdots g_k g_{k+1})$$

The purpose of the counter is to pack the stack symbols after a non-ε-transition. If the content of the counter is greater than 0, then the second phase is performed in the distinguished cell:

$$\begin{array}{l} \forall\, s \in S,\ g_j \in G : \\ \qquad \delta'\big((s, i, g_1 \cdots g_p \bot^i), (\text{€}, 0, g_{p+1} \cdots g_{p+k})\big) = (s, i-1, g_1 \cdots g_p g_{p+1} \bot^{i-1}) \\ \qquad \text{iff } \ i > 0 \end{array}$$

If the counter has been decreased to 0, then the next transition of $\mathcal{M}$ is simulated. If it is an ε-transition (cf. Figure 8) this fact can be recognized by the distinguished cell as well as by its left neighbor. Moreover, since during ε-transitions the top-of-stack symbol is erased we get the packing for free from the above described behavior.

$$\begin{array}{l} \forall\, a \in A,\ s \in S,\ g_j \in G : \\ \qquad \delta'\big((a, 0, \varepsilon), (s, i, g_1 \cdots g_k)\big) = (a, 0, \varepsilon) \\ \qquad \text{iff } \ i > 0 \text{ or } \delta(s, g_1, \varepsilon) \text{ is defined} \end{array}$$

$$\begin{array}{l} \forall\, a \in A,\ s \in S,\ g_j \in G : \\ \qquad \delta'\big((s, 0, g_1 \cdots g_k), \#\big) = (s', 0, g_2 \cdots g_k \bot) \ \text{ and} \\ \qquad \delta'\big((s, 0, g_1 \cdots g_k), (\text{€}, 0, g_{k+1} \cdots g_{2k})\big) = (s', 0, g_2 \cdots g_k g_{k+1}) \\ \qquad \text{iff } \ \delta(s, g_1, \varepsilon) = (s', \varepsilon) \end{array}$$

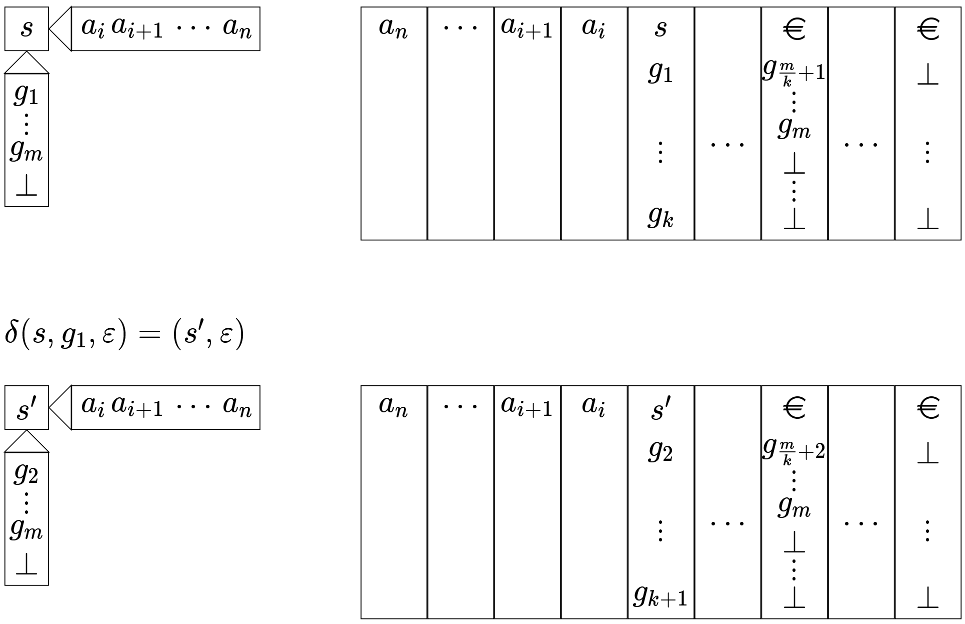


Figure 8: An ε-transition of the pda $\mathcal{M}$ (left) and the corresponding transition of the OCA $\mathcal{M}'$ (right). Counters are not depicted.

The OCA $\mathcal{M}'$ takes at most $(k+1)\cdot n$ time steps. It has to simulate n non-ε-transitions of $\mathcal{M}$. This takes n time steps. During these time steps at most $k \cdot n$ stack symbols can be generated. These are successively shifted out or erased by ε-transitions. So it takes at most another $k \cdot n$ time steps in order to erase them all. It follows that $\mathcal{M}'$ obeys the time complexity $(k+1)\cdot n$. $\square$

Combined with Theorem 7 the previous proof gives us the strictness of the inclusion $\mathscr{L}_{rt}(\mathrm{IA}) \subset \mathscr{L}_{rt}(\mathrm{CA})$ *and* a strictly in $\mathscr{L}_{rt}(\mathrm{CA})$ contained subfamily of $\mathscr{L}_2$.

Regarding $\mathscr{L}_{rt}(\mathrm{OCA})$ the following theorem works in some sense similarly.

Theorem 11 *Every metalinear language belongs to* $\mathscr{L}_{rt}(\mathrm{CA})$.

Proof. Let L be a metalinear language. Then there exists a $k \in \mathbb{N}$ such that L is k-linear. Therefore, we can represent L as the concatenation $L_1 \cdot L_2 \cdot \ldots \cdot L_k$ of k linear languages.

The family $\mathscr{L}_{rt}(\mathrm{OCA})$ is closed under reversal [11]. Combined with Theorem 8 we conclude that there exist real-time OCAs for each of the languages $L_1^R, \ldots, L_k^R$. Since the concatenation of a real-time and a linear-time OCA language is again a linear-time OCA language [8] we obtain $L_k^R \cdot \ldots \cdot L_1^R \in \mathscr{L}_{lt}(\mathrm{OCA})$. Due to the equality $\mathscr{L}_{lt}(\mathrm{OCA}) = \mathscr{L}_{rt}^R(\mathrm{CA})$ it follows $L_1 \cdot \ldots \cdot L_k = L \in \mathscr{L}_{rt}(\mathrm{CA})$. $\square$

References

[1] Buchholz, Th., Klein, A., and Kutrib, M. *On interacting automata with limited nondeterminism*. Fundamenta Informaticae (2000), to appear. IFIG Research Report 9806, Institute of Informatics, University of Giessen, Giessen, 1998.

[2] Chang, J. H., Ibarra, O. H., and Palis, M. A. *Parallel parsing on a one-way array of finite-state machines*. IEEE Trans. Comput. C-36 (1987), 64–75.

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

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

[5] Cole, S. N. *Real-time computation by n-dimensional iterative arrays of finite-state machines*. IEEE Trans. Comput. C-18 (1969), 349–365.

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

[7] Ginsburg, S. *The Mathematical Theory of Context-Free Languages*. McGraw Hill, New York, 1966.

[8] Ibarra, O. H. and Jiang, T. *On one-way cellular arrays*. SIAM J. Comput. 16 (1987), 1135–1154.

[9] Kosaraju, S. R. *Speed of recognition of context-free languages by array automata*. SIAM J. Comput. 4 (1975), 331–340.

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

[11] Seidel, S. R. *Language recognition and the synchronization of cellular automata*. Technical Report 79-02, Department of Computer Science, University of Iowa, Iowa City, 1979.

[12] Smith III, A. R. *Cellular automata and formal languages*. 11^{th} Ann. IEEE Symposium on Switching and Automata Theory, 1970, pp. 216–224.

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

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

[15] Umeo, H., Morita, K., and Sugata, K. *Deterministic one-way simulation of two-way real-time cellular automata and its related problems*. Inform. Process. Lett. 14 (1982), 158–161.

[16] von Neumann, J. *Theory of Self-Reproducing Automata*. Edited and completed by A. W. Burks. University of Illinois Press, Urbana, 1966.