IFIG
RESEARCH
REPORT

INSTITUT FÜR INFORMATIK

Efficient Universal Pushdown Cellular Automata and their Application to Complexity

Martin Kutrib

IFIG Research Report 0004

August 2000

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 0004, August 2000

Efficient Universal Pushdown Cellular Automata and their Application to Complexity

Martin Kutrib[1]

Institute of Informatics, University of Giessen
Arndtstr. 2, D-35392 Giessen, Germany

Abstract. In order to obtain universal classical cellular automata an infinite space is required. Therefore, the number of required processors depends on the length of input data and, additionally, may increase during the computation. On the other hand, Turing machines are universal devices which have one processor only and additionally an infinite storage tape.
Here an in some sense intermediate model is studied. The pushdown cellular automata are a stack augmented generalization of classical cellular automata. They form a massively parallel universal model where the number of processors is bounded by the length of input data.
Efficient universal pushdown cellular automata and their efficiently verifiable encodings are proposed. They are applied to computational complexity, and tight time and stack-space hierarchies are shown.

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

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

1 Introduction

Arrays of automata can be understood as models for massively parallel computers. By treating them as acceptors for formal languages their computational power can be compared with other parallel and sequential computer models. Under these aspects the automata arrays and various modifications have been studied for a long time. Especially investigations concerning universality (often combined with other properties) have been done e.g. in [1, 5, 10, 11, 13, 14]. A state-of-the-art survey on universality and decidability versus undecidability in cellular automata and several other models of discrete computations can be found in [9].

Due to the historical precedent for a fixed amount of memory per cell (unbounded) cellular automata have to be defined over an infinite space in order to obtain computational universality. Therefore, the number of required processors depends on the length of input data and, additionally, may increase during the computation. From a more practical point of view an infinite number of processors seems to be fairly unrealistic. On the other hand Turing acceptors are computationally universal devices which have one processor only and additionally an infinite storage tape. For this reason and due to the possible speed-up gained in parallelism we investigate the pushdown cellular automata PDCA where each cell is now a deterministic pushdown automaton [7, 8]. So we obtain a computationally universal computer model where the number of processors is bounded by the length of input data. Furthermore, in our opinion the assumption of arbitrary large pushdown memory is less problematical than the assumption of an arbitrary number of processors.

Clearly, a PDCA with at least two cells is sufficient in order to obtain an universal device. But with an eye towards applications e.g. in diagonalization proofs here we are interested in efficient universal PDCAs.

The basic notions and the model in question are introduced in the next section. Section 3 is devoted to the design of an efficiently verifiable encoding of PDCAs. Given the encoding of an arbitrary PDCA $\mathcal{M}$ and the encoding of an input word w a universal PDCA has to simulate the behavior of $\mathcal{M}$ on input w. Since in general w is independent of $\mathcal{M}$ at first it will be necessary to create the encoding of the initial configuration of $\mathcal{M}$ with respect to w. Subject of Section 4 are encodings of configurations. The efficient universal PDCAs are presented in Section 5. The crucial point is the bounded number of cells such that the encodings have to be stored into the stacks. This fact causes additional expenses of time. On the other hand, the time complexity must not exceed a certain magnitude in order to obtain tight hierarchies. Finally, in Section 6 the universal PDCAs are applied and tight time and stack-space hierarchies are shown.

2 Basic Notions

We denote the integers by $\mathbb{Z}$, the positive integers $\{1, 2, ...\}$ by $\mathbb{N}$ and the set $\mathbb{N} \cup \{0\}$ by $\mathbb{N}_0$. The empty word is denoted by λ and the reversal of a word w by w^R. For the length of w we write $|w|$. We use $\subseteq$ for inclusions and $\subset$ if the inclusion is strict. For a function $f : \mathbb{N}_0 \to \mathbb{N}$ we denote its i-fold composition by $f^{[i]}$, $i \in \mathbb{N}$. $[i](x_1 \cdots x_n) = x_i$ selects the ith component of a word or a vector.

A pushdown cellular automaton is a linear array of identical deterministic pushdown automata, sometimes called cells, where each of them is connected to its both nearest neighbors (one to the right and one to the left). For convenience we identify the cells by positive integers. They operate synchronously at discrete time steps. The state transition of a cell depends on the current states of its both neighbors, the current state of the cell itself and the current symbol at the top of its stack. With an eye towards language recognition we provide accepting and rejecting states. More formally:

Definition 1 *A pushdown cellular automaton (PDCA) is a system* $\langle S, G, \delta_s, \delta_p, \#, \perp, A, F_+, F_- \rangle$, *where*

1. *S is the finite, nonempty set of* states,
2. *G is the finite, nonempty set of* stack symbols,
3. *$\# \notin S$ is the* boundary state,
4. *$\perp \in G$ is the* bottom-of-stack symbol,
5. *A is the finite, nonempty set of* input symbols,
6. *F_+ and F_-, $F_+ \cap F_- = \emptyset$, are the sets of* accepting *and* rejecting *states, respectively,*
7. *$\delta_s : (S \cup \{\#\})^3 \times G \to S$ is the* local state transition function,
8. *$\delta_p : (S \cup \{\#\})^3 \times G \to \{\lambda\} \cup G \cup G^2$ is the* local stack transition function *satisfying $\forall s_1, s_2, s_3 \in S, g \in G \setminus \{\perp\}$:*

$$\delta_p(s_1, s_2, s_3, \perp) \in \{g'\perp \mid g' \in (G \setminus \{\perp\}) \cup \{\lambda\}\} \text{ and}$$
$$\delta_p(s_1, s_2, s_3, g) \in \{\lambda\} \cup (G \setminus \{\perp\}) \cup (G \setminus \{\perp\})^2.$$

The condition on the local stack transition function ensures that the bottom-of-stack symbol appears at each cell exactly once (i.e. at the bottom of its stack). At every transition step each cell consumes the symbol at the top of its stack (if it is not empty) and pushes at most two new symbols onto it. Note that the restriction of pushing at most two symbols at every time step neither reduces the computation power nor slows down the computation itself [4].

Let $\mathcal{M} = \langle S, G, \delta_s, \delta_p, \#, \perp, A, F_+, F_- \rangle$ be a PDCA. A configuration of $\mathcal{M}$ at some time $t \geq 0$ is a description of its global state which is actually a mapping $c_t : [1, \ldots, n] \to S \times G^+$ for $n \in \mathbb{N}$. The configuration at time 0 is defined by the initial sequence of states and empty stacks. For a given input $w = a_1 \cdots a_n \in A^+$ we set $c_0(i) = (a_i, \perp)$,$1 \leq i \leq n$. Subsequent configurations are computed according to the global transition function Δ: Let $n \in \mathbb{N}$ be an arbitrary positive integer and c and c' be two configurations defined by $(s_1, p_{1,1} \cdots p_{1,m_1}) \cdots (s_n, p_{n,1} \cdots p_{n,m_n})$ and $(s_1, p'_1) \cdots (s_n, p'_n)$, then

$$c' = \Delta(c) \iff$$

$$\begin{aligned}
s'_1 &= \delta_s\big(\#, s_1, s_2, p_{1,1}\big) \\
s'_i &= \delta_s\big(s_{i-1}, s_i, s_{i+1}, p_{i,1}\big), 2 \leq i \leq n-1 \\
s'_n &= \delta_s\big(s_{n-1}, s_n, \#, p_{n,1}\big) \\
p'_1 &= \delta_p\big(\#, s_1, s_2, p_{1,1}\big) p_{1,2} \cdots p_{1,m_1} \\
p'_i &= \delta_p\big(s_{i-1}, s_i, s_{i+1}, p_{i,1}\big) p_{i,2} \cdots p_{i,m_i}, 2 \leq i \leq n-1 \\
p'_n &= \delta_p\big(s_{n-1}, s_n, \#, p_{n,1}\big) p_{n,2} \cdots p_{n,m_n}
\end{aligned}$$

Thus, the global transition function Δ is induced by δ_s and δ_p.

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 a register of all cells forms a *track*.

3 Encoding of Pushdown Cellular Automata

Needless to say, in general it is possible to encode PDCAs and their configurations with any nonempty alphabet. But with an eye towards applications in formal language recognition, we are interested in efficiently verifiable encodings that have to be chosen with respect to the processing universal PDCA. Later on, the encodings in combination with the universal PDCA will determine the tightness of the hierarchies of language families.

Let $\text{bin} : \mathbb{N}_0 \to \{0,1\}^+$ be the mapping that maps a natural number to its binary representation without leading zeroes. Then the binary representation with leading zeroes is for all $k \geq 1$ defined by $\text{bin}_k : \mathbb{N}_0 \to \{0,1\}^+$, $n \mapsto 0^{k-|\text{bin}(n)|}\text{bin}(n)$ if $k \geq |\text{bin}(n)|$. $\text{bin}_k(n)$ is undefined for $k < |\text{bin}(n)|$.

S and G are finite nonempty sets and we can assume total orderings on their elements: $S = \{s_1, \ldots, s_{|S|}\}$ and $G = \{g_1, \ldots, g_{|G|}\}$. W.l.o.g. let s_0 be the boundary state # and g_1 be the bottom-of-stack symbol $\perp$. The different beginnings of the numberings have been chosen since for stack operations the empty word λ has to be encoded in addition. The ordering of the state set implies orderings of F_+ and F_-.

The state and stack transition functions can be represented as a table with seven columns. Each row is as follows:

$$s_i \quad s_j \quad s_k \quad g_l \quad \delta_s(s_i, s_j, s_k, g_l) \quad [1](\delta_p(s_i, s_j, s_k, g_l)) \quad [2](\delta_p(s_i, s_j, s_k, g_l))$$

We assume that the rows are in lexicographic order. If for certain s_i, s_j, s_k, g_l the corresponding row is missing then $\delta_s(s_i, s_j, s_k, g_l)$ is defined to be s_j and

$\delta_p(s_i, s_j, s_k, g_l)$ is defined to be g_l (i.e. neither the state nor the stack content changes).

Let $k = |\text{bin}(\max\{|G|, |S|\})|$ and $C = \{\mathtt{0}, \mathtt{1}, \mathtt{[}, \mathtt{]}, \mathtt{+}, \mathtt{-}, \mathtt{b}\}$.

1. The states, stack symbols and the empty word are encoded by

$$code_S : S \cup \{\mathtt{\#}\} \cup G \cup \{\lambda\} \to C^{k+3}$$

$$p \mapsto \begin{cases} \mathtt{[}\text{bin}_k(i)\mathtt{+]} & \text{if } p = s_i \in S \wedge s_i \in F_+ \\ \mathtt{[}\text{bin}_k(i)\mathtt{-]} & \text{if } p = s_i \in S \wedge s_i \in F_- \\ \mathtt{[}\text{bin}_k(i)\mathtt{b]} & \text{if } p = g_i \in G \vee p = s_i \in S \setminus (F_+ \cup F_-) \\ \mathtt{[}\text{bin}_k(0)\mathtt{b]} & \text{if } p = \lambda \vee p = \mathtt{\#} \end{cases}$$

2. An ordered subset $F = \{f_1, \ldots, f_m\} \subseteq S$ is encoded by

$$code_F(F) = code_S(f_1) \cdots code_S(f_m)$$

3. $code_r$ encodes the rows of the transition table

$$\begin{aligned} code_r\ (s_i, s_j, s_k, g_l, s_m, g_n, g_o) = \\ &\big([1](code_S(s_i)), [1](code_S(s_j)), \ldots, [1](code_S(g_o))\big) \\ &\quad \vdots \\ &\big([k+3](code_S(s_i)), [k+3](code_S(s_j)), \ldots, [k+3]code_S(g_o))\big) \end{aligned}$$

4. Consequently, the whole table with m rows is encoded as follows

$$code_\delta = code_r(r_1) \cdots code_r(r_m)$$

5. Hence, a PDCA $\mathcal{M}$ is encoded by

$$\begin{aligned} code\ (\mathcal{M}) = \\ &\mathtt{[}^7\big([1](code_S(s_{|S|})), [1](code_S(g_{|G|}))\big) \\ &\quad \vdots \\ &\quad \big([k+3](code_S(s_{|S|})), [k+3](code_S(g_{|G|}))\big) \\ &\quad code_S(g_{|G|}) code_F(F_+) code_F(F_-) code_\delta(\delta)\mathtt{]}^7 \end{aligned}$$

For easier reading we regard the encoding as a word over C^7 (i.e. all the not used registers are filled with blanks).

Example 2 $S = \{s_1, s_2, s_3, s_4\}$, $G = \{\perp, g_2\}$, $F_+ = \{s_2\}$, $F_- = \{s_3, s_4\}$ and $\delta_s(s_1, s_1, s_0, g_1) = s_2$ and $\delta_p(s_1, s_1, s_0, g_1) = \perp$ define the only row of the transition table. The encoding is as follows:

It is not hard but a technical challenge to prove that the language $\{w \in (C^7)^+ \mid \exists\, \text{PDCA}\ \mathcal{M} : w = code(\mathcal{M})\}$ of all PDCA encodings is recognizable by some PDCA in real-time.

[																									[	0	0	1	b	]	]
[																									[	0	0	1	b	]	]
[																									[	0	0	0	b	]	]
[																									[	0	0	1	b	]	]
[																									[	0	1	0	+	]	]
[	[	1	0	0	-	]																			[	0	0	1	b	]	]
[	[	0	1	0	b	]	[	0	1	0	+	]	[	0	1	1	-	]	[	1	0	0	-	]	[	0	0	0	b	]	]

Figure 1: Example encoding of a PDCA.

4 Encoding of Configurations

By designing an efficiently verifiable encoding of PDCAs we have done the first step towards an efficient universal PDCA. Given the encoding of an arbitrary PDCA $\mathcal{M}$ and the encoding of an input word w a universal PDCA has to simulate the behavior of $\mathcal{M}$ on input w. Since in general w is independent of $\mathcal{M}$ at first it will be necessary to create the encoding of the initial configuration of $\mathcal{M}$ with respect to w. Since the universal PDCA has to handle PDCAs with arbitrarily large state sets the space requirement may be arbitrarily large during the simulation. Since the number of available cells is bounded by the length of the input the encodings have to be stored into the stacks.

The following lemma solves a pattern transformation problem that will be utilized for the generation of initial configurations.

Lemma 3 *Let A be an alphabet, $\mathtt{b} \notin A$, $w = a_0 \cdots a_{n-1} \in A^+$ and $k \in \mathbb{N}$. Then there exists a* PDCA *that transforms a configuration $c_0(i) = a_i$, $0 \leq i \leq n-1$, and $c_0(i) = \mathtt{b}$, $n \leq i \leq n+k$, into the configuration $c_t(i) = a_{i \bmod n}$, $0 \leq i \leq n+k$, within $t \leq 4(n+k)$ time steps.*

Proof. Assume for a moment n divides k. The PDCA $\mathcal{M}$ has three tracks where the result is created at the first one. W.l.o.g. we may assume that initially the first and second track each contain $w\mathtt{b}^k$.

The content of the second track is shifted to the right as follows: a_{n-1} moves with speed $\frac{1}{2}$. The symbols at the left of a_{n-1} are moving right whenever their right neighbor becomes free (i.e. the corresponding symbol has moved to the right one time step before).

During the first transition the cell that contains a_{n-1} identifies itself and marks itself by an $*$ on its third track. At time 1 cell 0 establishes a signal r that moves to the right with speed 1. If the signal r passes through the cell marked by $*$, a signal s is established that moves to the right with speed $\frac{1}{3}$.

Signal r meets the moving symbol a_{n-1} after $2n$ time steps in cell $2n-1$. This cell is also marked by an $*$. Moreover, it reflects the signal r and establishes an additional signal l that moves to the left with speed 1.

Signal l causes all the cells passed through to copy the (moving) content of the second track to the first track if the first track is still empty. The moving process on the second track is not affected.

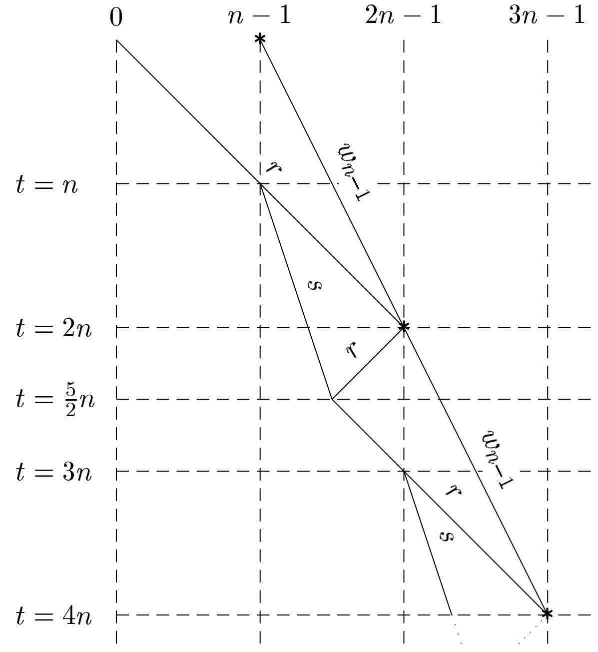

Figure 2: Signals in the proof of Lemma 3.

At time $\frac{5}{2}n$ the signals r and s meet in cell $\frac{3}{2}n-1$. Now signal s is dropped and signal r is reflected again. When r arrives at the cell $2n-1$ (which is marked by $*$) at time $3n$ the whole process is repeated.

If n does not divide k the PDCA simply uses two more tracks that are the folded extensions of the tracks one and two.

The PDCA needs $2n$ time steps for every concatenation of w and in addition less than $2n+k$ time steps for the last signal l to get back to the left. It follows

$$t \leq \left\lceil \frac{k}{n} \right\rceil \cdot 2n + \left\lceil \frac{k}{n} \right\rceil \cdot n + n \leq \left(\frac{k}{n} + 1 \right) \cdot 3n + n \leq 3k + 4n \leq 4(n+k)$$

□

The encoding of a single cell in a configuration is an element from $\{\mathtt{b}\} \times ((C^5)^k)^+$ and, thus, well-suited for later processing. The first component is the current state, the second one the stack content, which in term consists of the current states of the cell itself and of its neighbors and the current stack content itself.

Let $k = |code_S(s)|$ be the length of the codes for states. For a cell i at time t let $c_t(i-1) = (s_h, p_{h,1} \cdots p_{h,m_h})$, $c_t(i) = (s_i, p_{i,1} \cdots p_{i,m_i})$ and $c_t(i+1) = (s_j, p_{j,1} \cdots p_{j,m_j})$. Then the encoding of cell i at time t is

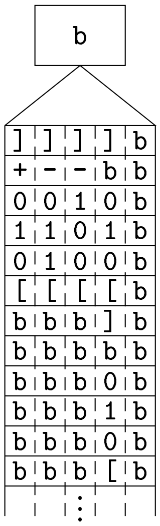

Figure 3: Example encoding of a cell.

$$
\begin{array}{l}
\mathit{code}_c\ (i,t) = \\
\quad \mathtt{b},\ \big([k](\mathit{code}_S(s_h)), [k](\mathit{code}_S(s_i)), [k](\mathit{code}_S(s_j)), [k](\mathit{code}_S(p_{i,1})), \mathtt{b}\big) \\
\qquad \vdots \\
\qquad \big([1](\mathit{code}_S(s_h)), [1](\mathit{code}_S(s_i)), [1](\mathit{code}_S(s_j)), [1](\mathit{code}_S(p_{i,1})), \mathtt{b}\big) \\
\qquad \big(\mathtt{b}, \mathtt{b}, \mathtt{b}, [k](\mathit{code}_S(p_{i,2})), \mathtt{b}\big) \\
\qquad \vdots \\
\qquad \big(\mathtt{b}, \mathtt{b}, \mathtt{b}, [1](\mathit{code}_S(p_{i,2})), \mathtt{b}\big) \\
\qquad \big(\mathtt{b}, \mathtt{b}, \mathtt{b}, [k](\mathit{code}_S(p_{i,3})), \mathtt{b}\big) \\
\qquad \vdots \\
\qquad \big(\mathtt{b}, \mathtt{b}, \mathtt{b}, [1](\mathit{code}_S(p_{i,m_i})), \mathtt{b}\big)
\end{array}
$$

Consequently, the encoding of a configuration c_t is the concatenation of the encodings of the cells:

$$\mathit{code}_c(c_t) = \mathit{code}_c(1,t)\mathit{code}_c(2,t)\cdots\mathit{code}_c(n,t)$$

Example 4 $S = \{s_1, s_2, s_3, s_4\}$, $G = \{\perp, g_2\}$, $F_+ = \{s_2\}$, $F_- = \{s_3, s_4\}$. Let $c_t(i-1) = (s_2, g_2\perp)$, $c_t(i) = (s_3, g_2g_2\perp)$ and $c_t(i+1) = (s_4, \perp)$ then $\mathit{code}_c(i,t)$ is

```
b,
(],],],],b)(+,-,-,b,b)(0,0,1,0,b)(1,1,0,1,b)(0,1,0,0,b)([,[,[,[,b)
(b,b,b,],b)(b,b,b,b,b)(b,b,b,0,b)(b,b,b,1,b)(b,b,b,0,b)(b,b,b,[,b)
(b,b,b,],b)(b,b,b,b,b)(b,b,b,0,b)(b,b,b,0,b)(b,b,b,0,b)(b,b,b,[,b)
```

It will be stored in a single cell and its stack as depicted in Figure 3.

5 Universal Pushdown Cellular Automata

This section is devoted to the construction of an efficient universal PDCA. The crucial point is the bounded number of cells such that the encodings have to be stored into the stacks. This fact causes additional expenses of time. On the other hand, the time complexity must not exceed a certain magnitude in order to obtain tight hierarchies.

The following first construction yields a PDCA $\mathcal{U}$ that, given the encoding of a PDCA $\mathcal{M}$ and the encoding of a configuration c_t of $\mathcal{M}$, computes the encoding of the successor configuration c_{t+1} of $\mathcal{M}$. Thus, $\mathcal{U}$ transforms a configuration

$$c'_u(i) = \begin{cases} [i](code(\mathcal{M})), \perp & \text{if } 1 \leq i \leq |code(\mathcal{M})| \\ code_c(i - |code(\mathcal{M})|, t)\perp & \text{if } |code(\mathcal{M})| < i \leq |code(\mathcal{M})| + |code_c(c_t)| \end{cases}$$

within some r time steps into the configuration

$$c'_{u+r}(i) = \begin{cases} [i](code(\mathcal{M})), \perp & \text{if } 1 \leq i \leq |code(\mathcal{M})| \\ code_c(i - |code(\mathcal{M})|, t+1)\perp & \text{if } |code(\mathcal{M})| < i \leq |code(\mathcal{M})| + |code_c(c_t)| \end{cases}$$

$\mathcal{U}$ simulates one transition step of $\mathcal{M}$ in three phases. During the *presimulation phase* some of the tracks are initialized. The successor states and stack symbols of $\mathcal{M}$ are computed during the *simulation phase*. Subsequently, during the *postsimulation phase* the valid configuration c'_{u+r} is computed such that the next step of $\mathcal{M}$ can be simulated.

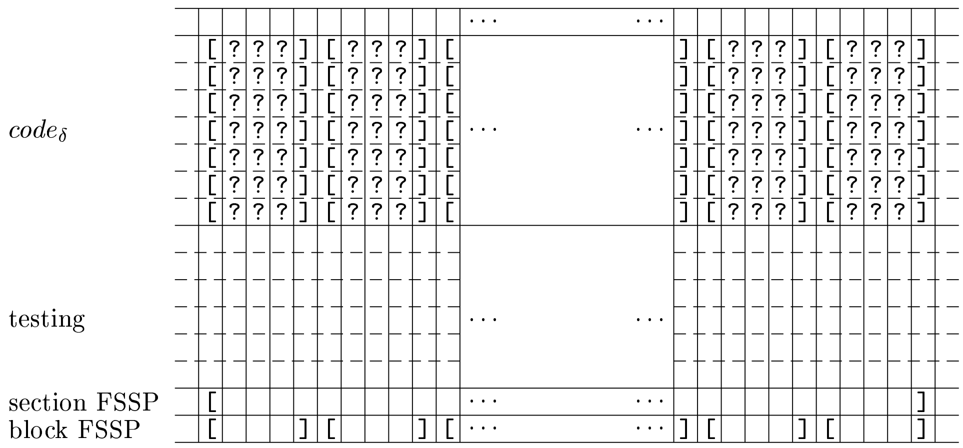

Figure 4: An inner section after the presimulation phase.

Presimulation phase: $\mathcal{U}$ has five main tracks which may be divided into subtracks. If in the sequel the lengths of some blocks do not divide the number of cells or if the construction uses a constant number of cells beyond the borders it is assumed that there exist another five main tracks that are used as the folded extensions of the corresponding tracks in order to cope with that (constant) space problem.

The cell that contains the last symbol of $code(\mathcal{M})$ can identify itself by the situation $]^7]^7$. The same holds for the cell that contains the first symbol of $code_\delta(\delta)$ by the situation $][^7$.

The cells that contain $code_c(c_t)$ are divided into sections of length $|code_\delta(\delta)|$ such that $code_\delta(\delta)$ is available on track 2 in each section. Each section is divided into blocks of length $|code_S(s)|$, lets say of length k. The result of the presimulation phase is depicted in Figure 4.

By Lemma 3 the sections can be created by concatenations of $code_\delta(\delta)$. During that process the inscriptions of track 4 and 5 can also be generated.

At the end of the presimulation phase a global FSSP is started on the first track such that all the cells synchronously start the simulation phase. The presimulation phase needs $t \leq 4(|code(\mathcal{M})| + |code_c(c_t)|) + 2(|code(\mathcal{M})| + |code_c(c_t)|) = 6(|code(\mathcal{M})| + |code_c(c_t)|)$ time steps.

Simulation phase: In the sequel all signals and labellings are realized on the first main track (resp. its subtracks). The simulation is performed in all sections in parallel. Therefore, it suffices to explain the process for one section.

On the fifth track modified FSSPs are performed. [and] are generals that synchronize the cells in between them every k (i.e. $code_S(s)$) time steps. After each other synchronization the process is one time step delayed.

During such a cycle all cells of $code_c(c_t)$ with $]^7$ on their second track are working as follows (cf. Figure 5 for the general behavior).

During each time step the top-of-stack symbol is copied onto five subtracks of the third track. At the same time the contents of the tracks 2 and 3 are shifted to the right. Since on the second track there is the encoding of one of the rows of the transition table the cell can successively test whether the row matches its current situation.

After k time steps the FSSP fires and the contents of the second and third track now are successively shifted to the left. Therefore, the cell can push its old state and top-of-stack symbol back into its stack if the row did not match the current situation, or the new state and top-of-stack symbol(s) otherwise. The test is finished when the FSSP fires again. During the delay step the contents of track 2 and 5 are shifted one cell to the right and the next cells will be tested during the next $2k$ time steps.

The process has to be repeated until all cells of the section have been tested with all blocks of $code_\delta(\delta)$. This needs $|code_\delta(\delta)| \cdot (2k+1)$ time steps.

In order to recognize the end of the simulation phase on track 4 a FSSP is performed that synchronizes the whole section. One transition of the FSSP is computed at every time the block FSSP has finished a test cycle. Thus, the section is exactly synchronized after $|code_\delta(\delta)| \cdot (2k+1)$ time steps.

Postsimulation phase: At the beginning of this phase the situation at the top of the stacks is as depicted in Figure 6.

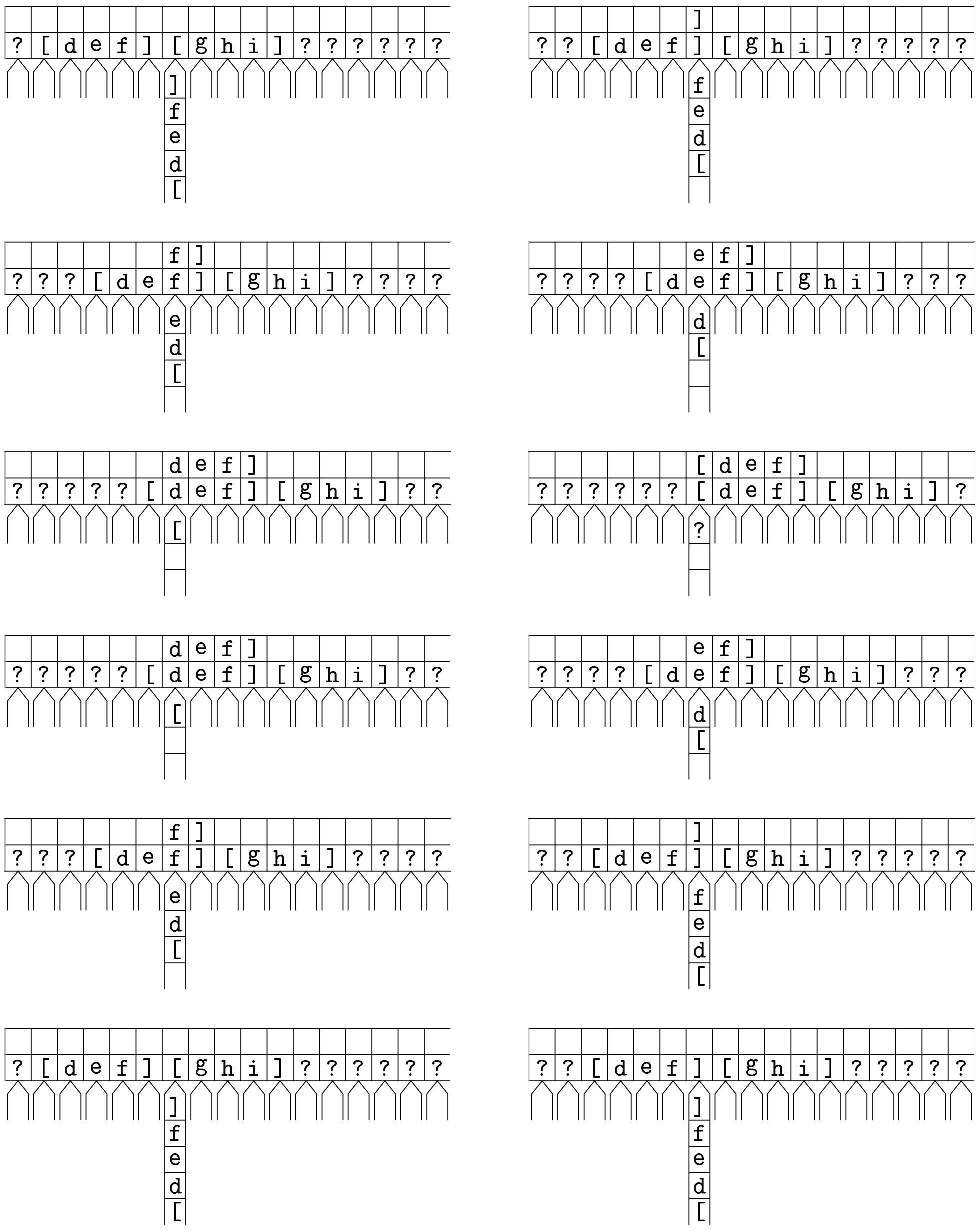

Figure 5: A test cycle in the simulation phase.

The process that updates the stack contents is similar to the simulation phase. During k time steps the contents of the top of the stack of three adjacent cells are successively copied onto the third track. Then during another $2k$ time steps the stack content of the inner cell is updated appropriately.

The postsimulation phase needs k such update cycles, hence, $k(3k+1)$ time steps.

Theorem 5 *$\mathcal{U}$ simulates l transitions of a* PDCA $\mathcal{M}$ *within* $t \leq 6n+7l|code_\delta(\delta)|^2$ *time steps, where n denotes the length of the input of $\mathcal{U}$.*

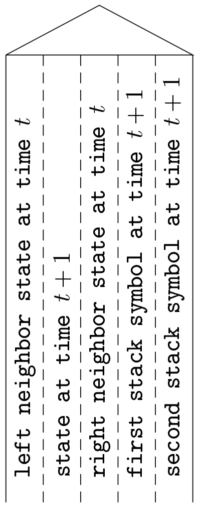

Figure 6: Stack content at the beginning of the postsimulation phase.

Proof. The theorem follows from the fact that the presimulation phase has only to be performed once. Let $k = |code_S(s)|$. From the construction we obtain:

$$\begin{aligned} t &\leq 6(|code(\mathcal{M})| + |code_c(c_t)|) + l(|code_\delta(\delta)| \cdot (2k+1) + k(3k+1)) \\ &\leq 6n + l(|code_\delta(\delta)| \cdot 2k + |code_\delta(\delta)| + 3k^2 + k) \\ &\leq 6n + l(2|code_\delta(\delta)|^2 + |code_\delta(\delta)| + 3|code_\delta(\delta)|^2 + |code_\delta(\delta)|) \\ &\leq 6n + l(5|code_\delta(\delta)|^2 + 2|code_\delta(\delta)|) \\ &\leq 6n + 7l|code_\delta(\delta)|^2 \end{aligned}$$

□

The PDCA $\mathcal{U}$ works fine if its input is the encoding of a PDCA $\mathcal{M}$ and the encoding of a configuration of $\mathcal{M}$. In the following this precondition is too restrictive since for diagonalization proofs we need to consider the behavior of PDCAs when they get their own encoding as input.

The following construction yields a PDCA $\mathcal{V}$ that, given the encoding of a PDCA $\mathcal{M}$, computes the encoding $code_c(c_0)$ where c_0 is the initial configuration of $\mathcal{M}$ with input $code(\mathcal{M})$. Subsequently, $\mathcal{V}$ computes the encoding of the successor configuration c_1 of $\mathcal{M}$ and so on.

The first task of $\mathcal{V}$ is to compute the encoding of the initial configuration of $\mathcal{M}$. The second task is to simulate the universal PDCA $\mathcal{U}$ in order to compute encodings of successor configurations of $\mathcal{M}$.

The construction of $\mathcal{U}$ has been done under the assumption that $code(\mathcal{M})$ is located at the left of the encoding of the configuration. This situation can be emulated as follows. We assume that for each track there exists another one which is regarded as the extension of the track. Both tracks are connected at the left border. So it suffices to write the mirror image of $code(\mathcal{M})$ on the extension. This can be done in $|code(\mathcal{M})|$ time steps.

Subsequently, $\mathcal{V}$ simulates the presimulation phase of $\mathcal{U}$ where $code_\delta(\delta)$ is concatenated in order to initialize the sections. Parallel to this process all cells i compute $code_c(i, 0)$ which completes the first task of $\mathcal{V}$. This computation is now explained for one of the stack registers.

During the presimulation phase $code_\delta(\delta)$ is moved across the cells. Cell i with input $s_i \in C$ "knows" $\mathrm{bin}(i)$. When it receives a `]` it waits for $|\mathrm{bin}(i)|$ time steps and subsequently pushes `0`s into the stack until it receives a `[`. Now it pushes $\mathrm{bin}(i)$ into the stack.

If $code_F(F_+)$ and $code_F(F_-)$ are also moved across the cells, the encoding of s_i can be completed simply by successively testing whether s_i belongs to one of the sets and by pushing the appropriate symbol `+`, `-` or `b` followed by a `[`.

Theorem 6 *$\mathcal{V}$ simulates l transitions of a* PDCA *$\mathcal{M}$ that operates on its own encoding within $t \leq 13n + 7l|code_\delta(\delta)|^2$ time steps, where n denotes the length of the input of $\mathcal{V}$.*

Proof. From the construction follows: $\mathcal{U}$ needs $|code(\mathcal{M})| = n$ time steps to create the mirror image of $code(\mathcal{M})$. Subsequently, $\mathcal{V}$ simulates the presimulation phase of $\mathcal{U}$ in which additionally the encoding of the configuration is computed. Since we are concerned with extended tracks the time has to be doubled. Thus, this phase needs $12n$ time steps. After the presimulation phase $\mathcal{V}$ simulates $\mathcal{U}$ directly and the theorem follows. □

6 Tight Hierarchies of Language Families

Subject of this section are tight time and space hierarchies. Since the number of cells is fixed by the length of the input the space complexity is measured as stack-space.

Definition 7 *Let $\mathcal{M} = \langle S, G, \delta_s, \delta_p, \#, \perp, A, F_+, F_- \rangle$ be a* PDCA. *$F_+ \cup F_-$ is the set of final states.*

1. *A word $w \in A^+$ is* accepted *resp.* rejected *by $\mathcal{M}$ if at input w the leftmost cell of $\mathcal{M}$ becomes final and if its first final state is an accepting resp. rejecting state.*

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 function. A* PDCA *is said to be t*-time-bounded *or of* time complexity *t iff every input of length n after at most $t(n)$ time steps is accepted or rejected.*

4. *Let $g : \mathbb{N} \to \mathbb{N}$ be a function. A* PDCA *is said to be g*-stack-space-bounded *or of* space complexity *g iff every input of length n is accepted or rejected at some time t and for all $t' \leq t$ each of the stacks contains at most $g(n)$ symbols.*

The family of all languages which can be accepted by PDCAs with time complexity t resp. space complexity g is denoted by $\mathscr{L}_t(\text{PDCA})$ resp. ${}_g\mathscr{L}(\text{PDCA})$.

In order to prove infinite tight hierarchies in almost all cases honest resource bounding functions are required. Usually the notion "honest" is concretized in terms of the computability or constructibility of the function with respect to the device in question.

Definition 8 *A function $f : \mathbb{N} \to \mathbb{N}$ is* computable *if there exists a* PDCA *$\mathcal{M}$ with input alphabet A and constant stack-space complexity such that $\mathcal{M}$ recognizes all $w \in A^+$ in exactly $f(|w|)$ time steps.*

Thus, computability of f means that there exists a PDCA that for any input w from A^+ can distinguish the time step $f(|w|)$ without using its stack (i.e. a classical cellular automaton). As usual here we remark that the class of such functions is very rich [2, 3, 6, 12].

Now we are prepared to prove the tight time hierarchy.

Theorem 9 *Let $t : \mathbb{N} \to \mathbb{N}$ and $t' : \mathbb{N} \to \mathbb{N}$ be two functions. If t is computable and $\underline{\lim}_{n\to\infty} \frac{t'(n)}{t(n)} = 0$ then there exists a language L such that*

$$L \in \mathscr{L}_{t(n)}(\text{PDCA}) \setminus \mathscr{L}_{t'(n)}(\text{PDCA})$$

If additionally $\forall\, n \in \mathbb{N} : t'(n) \leq t(n)$ then $\mathscr{L}_{t'(n)}(\text{PDCA}) \subset \mathscr{L}_{t(n)}(\text{PDCA})$.

Proof. Let $\mathcal{W}$ be a PDCA that works as follows. At first $\mathcal{W}$ checks whether or not its input belongs to the language $L' = \{uv \mid v \in \{\mathtt{0},\mathtt{1}\}^+ \wedge \exists\, \text{PDCA}\, \mathcal{M} : u = \mathit{code}(\mathcal{M})\}$. Since the cell that contains the last symbol of u can identify itself this verification needs at most $|uv|$ time steps.

Subsequently, $\mathcal{W}$ performs two tasks in parallel. One is to simulate the computation of $\mathcal{M}$ with input uv as has been shown by the construction for Theorems 5 and 6. The second one is to distinguish the time step $t(|uv|)$ since t is computable.

$\mathcal{W}$ rejects its input if $uv \notin L'$ or if after $t(|uv|)$ time steps the simulation of $\mathcal{M}$ has not produced a decision. If, on the other hand, $\mathcal{W}$ recognizes during $t(|uv|)$ time steps that $\mathcal{M}$ did its decision, then $\mathcal{W}$ rejects if $\mathcal{M}$ accepts and vice versa. Thus, $L(\mathcal{W}) \in \mathscr{L}_t(\text{PDCA})$.

Contrarily to the assertion we assume that there exists a PDCA $\mathcal{W}'$ with $u = \mathit{code}(\mathcal{W}')$ that recognizes $L(\mathcal{W})$ with time complexity t'. By Theorem 6 $\mathcal{W}$ needs $k_1|uv|+k_2t'(|uv|)k_3^2$ time steps in order to simulate $t'(|uv|)$ transitions of $\mathcal{W}'$. k_1, k_2 and k_3 are constants that depend on $\mathcal{W}'$. W.l.o.g. we may assume $t'(|uv|) \geq |uv|$. Therefore, the time complexity of $\mathcal{W}$ is $k_1|uv| + k_2t'(|uv|)k_3^2 \leq k_4t'(|uv|)$ for a suitable constant k_4. From the limes inferior we obtain a $v' \in \{\mathtt{0},\mathtt{1}\}^+$ such that $k_4t'(|uv'|) \leq t(|uv'|)$. Therefore, $\mathcal{W}$ can simulate $t'(|uv'|)$ transitions of $\mathcal{W}'$ within $t(|uv'|)$ time steps. If $\mathcal{W}'$ accepts the input uv' in $t'(|uv'|)$ time steps then $\mathcal{W}$ rejects. If $\mathcal{W}'$ rejects the input within $t'(|uv'|)$ time steps then $\mathcal{W}$ accepts. This is a contradiction to the assumption that $\mathcal{W}'$ accepts $L(\mathcal{W})$ with time complexity t'. □

Without proof we state the tight space hierarchy:

Theorem 10 *Let $g : \mathbb{N} \to \mathbb{N}$ and $g' : \mathbb{N} \to \mathbb{N}$ be two functions. If g is computable and $\underline{\lim}_{n\to\infty} \frac{g'(n)}{g(n)} = 0$ then there exists a language L such that*

$$L \in {}_{g(n)}\mathscr{L}(\mathrm{PDCA}) \setminus {}_{g'(n)}\mathscr{L}(\mathrm{PDCA})$$

If additionally $\forall\, n \in \mathbb{N} : g'(n) \leq g(n)$ then ${}_{g'(n)}\mathscr{L}(\mathrm{PDCA}) \subset {}_{g(n)}\mathscr{L}(\mathrm{PDCA})$.

References

[1] Albert and Čulik II, K. *A simple universal cellular automaton and its one-way and totalistic version*. Complex Systems 1 (1987), 1–16.

[2] Buchholz, Th. and Kutrib, M. *Some relations between massively parallel arrays*. Parallel Comput. 23 (1997), 1643–1662.

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

[4] Harrison, M. A. *Introduction to Formal Language Theory*. Addison-Wesley, Reading, 1978.

[5] Imai, K. and Morita, K. *A computation-universal two-dimensional 8-state triangular reversible cellular automaton*. Theoret. Comput. Sci. 231 (2000), 181–191.

[6] Iwamoto, C., Hatsuyama, T., Morita, K., and Imai, K. *On time-constructible functions in one-dimensional cellular automata*. Fundamentals of Computation Theory 1999, LNCS 1684, 1999, pp. 317–326.

[7] Kutrib, M. *Pushdown cellular automata*. Theoret. Comput. Sci. 215 (1999), 239–261.

[8] Kutrib, M. and Richstein, J. *Real-time one-way pushdown cellular automata languages*. Developments in Language Theory II. At the Crossroads of Mathematics, Computer Science and Biology, World Scientific, Singapore, 1996, pp. 420–429.

[9] Margenstern, M. *Frontier between decidability and undecidability: a survey*. Theoret. Comput. Sci. 231 (2000), 217–251.

[10] Martin, B. *Efficient unidimensional universal cellular automaton*. Mathematical Foundations of Computer Science 1992, LNCS 629, 1992, pp. 374–382.

[11] Martin, B. *A universal cellular automaton in quasi-linear time and its S–m–n form*. Theoret. Comput. Sci. 123 (1994), 199–237.

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

[13] Morita, K. *Computation-universality of one-dimensional one-way reversible cellular automata*. Inform. Process. Lett. 42 (1992), 325–329.

[14] Morita, K. and Ueno, S. *Computation-universal models of two-dimensional 16-state reversible cellular automata*. Trans. IEICE 75 (1992), 141.