I F I G
Research
Report

Institut für Informatik

Massively Parallel Pattern Recognition with Link Failures

Martin Kutrib, Jan-Thomas Löwe

IFIG Research Report 0003
June 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 0003, June 2000

Massively Parallel Pattern Recognition with Link Failures

Martin Kutrib[1], Jan-Thomas Löwe[2]

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

Abstract. The capabilities of reliable computations in linear cellular arrays with communication failures are investigated in terms of pattern recognition.
The defective processing elements (cells) that cause the misoperations are assumed to behave as follows. Dependent on the result of a self-diagnosis of their communication links they store their working state locally such that it becomes visible to the neighbors. A defective cell is not able to receive information via one of its both links to adjacent cells. The self-diagnosis is run once before the actual computation. Subsequently no more failures may occur in order to obtain a valid computation.
We center our attention to patterns that are recognizable very fast, i.e. in real-time. It is well-known that real-time one-way arrays are strictly less powerful than real-time two-way arrays, but there is only little known on the range between these two devices. Here it is shown that the sets of patterns reliably recognizable by real-time arrays with link failures are strictly in between the sets of (intact) one-way and (intact) two-way arrays. Hence, the failures cannot be compensated in general but, on the other hand, do not decrease the computing power to that one of one-way arrays.

CR Subject Classification (1998): F.1, F.4.3, B.6.1, E.1, B.8.1, C.4

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

1 Introduction

Nowadays it becomes possible to build massively parallel computing systems that consist of hundred thousands of processing elements. Each single component is subject to failure such that the probability of misoperations and loss of function of the whole system increases with the number of its elements. It was von Neumann [14] who first stated the problem of building reliable systems out of unreliable components. Biological systems may serve as good examples. Due to the necessity to function normally even in case of certain failures of their components the nature developed mechanisms which invalids the errors, they are working in some sense fault tolerant. Error detecting and correcting components should not be global to the whole system because they themselves are subject to failure. Therefore the fault tolerance has to be a design feature of the single elements.

A model for massively parallel, homogeneously structured computers are the cellular arrays. Such devices of interconnected parallel acting finite state machines have been studied from various points of view. Under the constraint that cells themselves (and not their links) fail (i.e. they cannot process information but are still able to transmit it unchanged with unit speed) fault tolerant computations have been investigated, e.g. in [3, 11] where encodings are established that allow the correction of so-called K-separated misoperations, in [7, 8, 13, 15] where the famous firing squad synchronization problem is considered in defective cellular arrays, and in terms of interacting automata with nonuniform delay in [4, 9] where the synchronization of the networks is the main object either.

Recently, this approach has been generalized to more general computations and dynamic defects. In [6] it has been shown that fault tolerant recognition capabilities of two-way arrays with static defects are characterizable by intact one-way arrays and that one-way arrays are fault tolerant per se. For arrays with dynamic defects it was proved that the failures can be compensated as long as the number of adjacent defective cells is bounded. Arbitrary large defective regions (and thus fault tolerant computations) lead to a dramatically decrease of computing power. The recognizable patterns are those of a single processing element, the regular ones.

Here we are interested in another natural type of defects. Not the cells themselves cause the misoperations but their communication links. It is assumed that each cell has a self-diagnosis circuit for its links which is run once before the actual computation. The results are stored locally in the cells and subsequently no new defects may occur. Otherwise the whole computation would become invalid. A defective cell is not able to receive information via at most one of its both links to adjacent cells. Otherwise the parallel computation would be broken into two non-interacting parts and, therefore, would become impossible at all.

In terms of pattern recognition the general capabilities of reliable computations are considered. Since cellular arrays have been intensively investigated from a language theoretic point of view, pattern recognition (or language acceptance) establishes the connection to the known results and, thus, inheres the possibility to compare the fault tolerant capabilities to the non fault tolerant ones.

The model in question is introduced in Section 3 in more detail. In Section 4 it is shown that the real-time arrays with defects are able to reliably recognize a wider range of sets of patterns than intact one-way arrays. In order to show this result some algorithmic subroutines for time-constructions and various counters are given. Section 5 concludes the investigations by showing that the devices with failures are strictly weaker than two-way arrays. Hence, the failures cannot be compensated in general but, on the other hand, do not decrease the computing power to that one of purely one-way arrays.

In the following section we define the basic notions and recall the underlying intact cellular arrays and their mode of pattern recognition.

2 Basic notions

We denote the integers by $\mathbb{Z}$, the positive integers $\{1, 2, \cdots\}$ by $\mathbb{N}$ and the set $\mathbb{N} \cup \{0\}$ by $\mathbb{N}_0$. $X_1 \times \cdots \times X_d$ denotes the Cartesian product of the sets $X_1, \ldots, X_d$. If $X_1 = \cdots = X_d$ we use the notation X_1^d alternatively. We use $\subseteq$ for inclusions and $\subset$ if the inclusion is strict. Let M be some set and $f : M \to M$ be a function, then we denote the i-fold composition of f by $f^{[i]}$, $i \in \mathbb{N}$.

A two-way resp. one-way cellular array is a linear array of identical finite state machines, sometimes called cells, which are connected to their both nearest neighbors resp. to their nearest neighbor to the right. The array is bounded by cells in a distinguished so-called boundary state. For convenience we identify the cells by positive integers. The state transition depends on the current state of each cell and the current state(s) of its neighbor(s). The transition function is applied to all cells synchronously at discrete time steps. Formally:

Definition 1 *A* two-way cellular array (CA) *is a system* $\langle S, \delta, \#, A \rangle$*, where*

1. S *is the finite, nonempty set of* cell states,
2. $\# \notin S$ *is the* boundary state,
3. $A \subseteq S$ *is the set of* input symbols,
4. $\delta : (S \cup \{\#\})^3 \to S$ *is the* local transition function.

If the flow of information is restricted to one-way (i.e. from right to left) the resulting device is a *one-way cellular array* (OCA) and the local transition function maps from $(S \cup \{\#\})^2$ to S.

A *configuration* of a cellular array at some time $t \geq 0$ is a description of its global state, which is actually a mapping $c_t : [1, \ldots, n] \to S$ for $n \in \mathbb{N}$.

The data on which the cellular arrays operate are patterns built from input symbols. Since here we are studying one-dimensional arrays only the input data are finite strings (or words). The set of strings of length n built from symbols from a set A is denoted by A^n, the set of all such finite strings by A^*. We denote the *empty string* by ε and the *reversal of a string* w by w^R. For its length we write $|w|$. A^+ is defined to be $A^* \setminus \{\varepsilon\}$.

In the sequel we are interested in the subsets of strings that are recognizable by cellular arrays. In order to establish the connection to formal language theory we call such a subset a *formal language*. Moreover, sets L and L' are considered to be equal if they differ at most by the empty word, i.e. $L \setminus \{\varepsilon\} = L' \setminus \{\varepsilon\}$.

Now we are prepared to describe the computations of (O)CAs. The operation starts in the so-called *initial configuration* $c_{0,w}$ at time 0 where one symbol of the input string $w = x_1 \cdots x_n$ is fed to one cell, respectively: $c_{0,w}(i) = x_i$, $1 \leq i \leq n$. During a computation the (O)CA steps through a sequence of configurations whereby successor configurations are computed according to the global transition function Δ: Let c_t, $t \geq 0$, be a configuration, then its successor configuration is as follows:

$$\begin{aligned} c_{t+1} = \Delta(c_t) \iff & \\ & c_{t+1}(1) = \delta\big(\#, c_t(1), c_t(2)\big) \\ & c_{t+1}(i) = \delta\big(c_t(i-1), c_t(i), c_t(i+1)\big), i \in \{2, \dots, n-1\} \\ & c_{t+1}(n) = \delta\big(c_t(n-1), c_t(n), \#\big) \end{aligned}$$

for CAs and

$$\begin{aligned} c_{t+1} = \Delta(c_t) \iff & \\ & c_{t+1}(i) = \delta\big(c_t(i), c_t(i+1)\big), i \in \{1, \dots, n-1\} \\ & c_{t+1}(n) = \delta\big(c_t(n), \#\big) \end{aligned}$$

for OCAs. Thus, Δ is induced by δ.

An input string w is recognized by an (O)CA if at some time i during its course of computation the leftmost cell enters a final state from the *set of final states* $F \subseteq S$.

Definition 2 *Let $\mathcal{M} = \langle S, \delta, \#, A \rangle$ be an* (O)CA *and $F \subseteq S$ be a set of final states.*

1. *An input $w \in A^*$ is* recognized *by $\mathcal{M}$ if it is the empty string or if there exists a time step $i \in \mathbb{N}$ such that $c_i(1) \in F$ holds for the configuration $c_i = \Delta^{[i]}(c_{0,w})$.*
2. *$L(\mathcal{M}) = \{w \in A^* \mid w$ is recognized by $\mathcal{M}\}$ is the* set of strings (language) recognized *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 $\mathcal{M}$ recognizes $w \in L(\mathcal{M})$. If all $w \in L(\mathcal{M})$ are recognized within $i_w \leq t(|w|)$ time steps, then L is said to be of* time complexity *t.*

The family of all sets which are recognizable by some CA (OCA) with time complexity t is denoted by $\mathscr{L}_t(\mathrm{CA})$ ($\mathscr{L}_t(\mathrm{OCA})$). If t equals the identity function $id(n) = n$ recognition is said to be in *real-time*, and if t is equal to $k \cdot id$ for an arbitrary rational number $k \geq 1$ then recognition is carried out in *linear-time*. Correspondingly, we write $\mathscr{L}_{rt}((\mathrm{O})\mathrm{CA})$ and $\mathscr{L}_{lt}((\mathrm{O})\mathrm{CA})$. In the sequel we will use corresponding notations for other types of recognizers.

3 Devices with link failures

In [6] it has been shown for CAs with defective cells that in case of large adjacent defective regions the bidirectional information flow gets lost. This means that the fault tolerant computation capabilities of two-way arrays are those of one-way arrays. The observation gives rise to investigate cellular arrays with defective links for their own. In order to explore the corresponding general reliable recognition capabilities we have to take a closer look on the device in question.

The defects are in some sense static [13]: It is assumed that each cell has a self-diagnosis circuit for its links which is run once before the actual computation. The result of that diagnosis is indicated by the states of the cells such that intact cells can detect defective neighbors. Moreover (and this is the static part), it is assumed that during the actual computation no new defects may occur. Otherwise the whole computation would become invalid. What is the effect of a defective link? Suppose that two adjacent cells are interconnected by two unidirectional links, respectively. On one link information is transmitted from right to left and on the other one from left to right. Now, if both links are failing, then the parallel computation would be broken into two not interacting lines and, thus, would be impossible at all. Therefore, it is reasonable to require that at least one of the links between two cells does not fail, respectively.

Suppose for a moment that there exists a cell that, due to a link failure, cannot receive information from its right neighbor. This would imply that the overall computation result (indicated by the leftmost cell) is obtained with no regard to the input data to the right of that defective cell. So all reliable computations would be trivial. In order to avoid this problem we extend the hardware such that if a cell detects a right to left link failure it is able to reverse the direction of the other (intact) link. Thereby we are always concerned with defective links that cannot transmit information from left to right.

Another point of view on such devices is that some of the cells of a two-way array behave like cells of a one-way array. Sometimes in the sequel we will call them OCA-cells.

The result of the self-diagnosis is indicated by the states of the cells. Therefore we have a partitioned state set.

Definition 3 *A* cellular array with defective links *(mO-CA) is a system* $\langle S, \delta_i, \delta_d, \#, A, m \rangle$*, where*

1. $S = S_i \cup S_d$ *is the partitioned, finite, nonempty set of* cell states *satisfying* $S_i \cap S_d = \emptyset$ *and* $S_d = \{s' \mid s \in S_i\}$,
2. $\# \notin S$ *is the* boundary state,
3. $A \subseteq S_i$ *is the set of* input symbols,
4. $m \in \mathbb{N}_0$ *is an upper bound for the* number of link failures,
5. $\delta_i : (S \cup \{\#\})^3 \to S_i$ *is the* local transition function for intact cells,
6. $\delta_d : (S \cup \{\#\})^2 \to S_d$ *is the* local transition function for defective cells.

Considering the general real-time recognition capabilities of mO-CAs the best case is trivial. It occurs when all the cells are intact: The capabilities are those of CAs. On the other hand, reliable computations are concerned with the worst case (with respect to our assumptions on the model). In particular, the recognition process has to compute the correct result for all distributions of the at most m defective links. In advance it is, of course, not known which of the links will fail. Therefore, for mO-CAs we have a set of admissible start configurations as follows.

For an input string $w = x_1 \cdots x_n \in A^n$ the configuration $c_{0,w}$ is an admissible start configuration of a mO-CA if there exists a set $D \subseteq \{1, \dots, n\}$ of defective cells, $|D| \leq m$, such that $c_{0,w}(i) = x_i \in S_i$ if $i \in \{1, \dots, n\} \setminus D$ and $c_{0,w}(i) = x'_i \in S_d$ if $i \in D$.

For a clear understanding we define the global transition function Δ of mO-CAs as follows: Let c_t, $t \geq 0$, be a configuration of a mO-CA with defective cells D, then its successor configuration is as follows:

$$c_{t+1} = \Delta(c_t) \iff$$

$$\begin{cases} c_{t+1}(1) = \delta_i\big(\#, c_t(1), c_t(2)\big) & \text{if } 1 \notin D \\ c_{t+1}(1) = \delta_d\big(c_t(1), c_t(2)\big) & \text{if } 1 \in D \end{cases}$$

$$\begin{cases} c_{t+1}(j) = \delta_i\big(c_t(j-1), c_t(j), c_t(j+1)\big) & \text{if } j \notin D, j \in \{2, \dots, n-1\} \\ c_{t+1}(j) = \delta_d\big(c_t(j), c_t(j+1)\big) & \text{if } j \in D, j \in \{2, \dots, n-1\} \end{cases}$$

$$\begin{cases} c_{t+1}(n) = \delta_i\big(c_t(n-1), c_t(n), \#\big) & \text{if } n \notin D \\ c_{t+1}(n) = \delta_d\big(c_t(n), \#\big) & \text{if } n \in D \end{cases}$$

Due to our definition of δ_i and δ_d once the computation has started the set D remains fixed, what meets the requirements of our model.

In the following we are going to explore some general recognition capabilities of mO-CAs. Do some link failures reduce the recognition power of intact CAs or is it possible to compensate the defects by modifications of the transition function as has been shown for CAs with defective cells [6]? Can mO-CAs recognize a wider range of string sets than intact OCAs?

4 mO-CAs are better than OCAs

The inclusions $\mathscr{L}_{rt}(\text{OCA}) \subseteq \mathscr{L}_{rt}(m\text{O-CA}) \subseteq \mathscr{L}_{rt}(\text{CA})$ are following immediately from the definitions. Our aim is to prove that both inclusions are strict.

4.1 Subroutines

In order to prove that real-time mO-CAs are more powerful than real-time OCAs we need some results concerning CAs and OCAs which will later on serve as subroutines of the general construction.

4.1.1 Time constructors

A strictly increasing mapping $f : \mathbb{N} \to \mathbb{N}$ is said to be *time constructible* if there exists a CA such that for an arbitrary initial configuration the leftmost cell enters a final state at and only at time steps $f(j)$, $1 \leq j \leq n$. A corresponding CA is called a *time constructor* for f. It is therefore able to distinguish the time steps $f(j)$.

The following lemma has been shown in [1].

Lemma 4 *The mapping $f(n) = 2^n$, $n \in \mathbb{N}$, is time constructible.*

Proof. The idea of the proof is depicted in Figure 1. At initial time the leftmost cell of a CA sends a signal with speed 1/3 to the right. At the next time step a second signal is established that runs with speed 1 and bounces between the slow signal and the left border cell. The leftmost cell enters a final state at every time step it receives the fast signal. The correctness of the construction is easily seen by induction. □

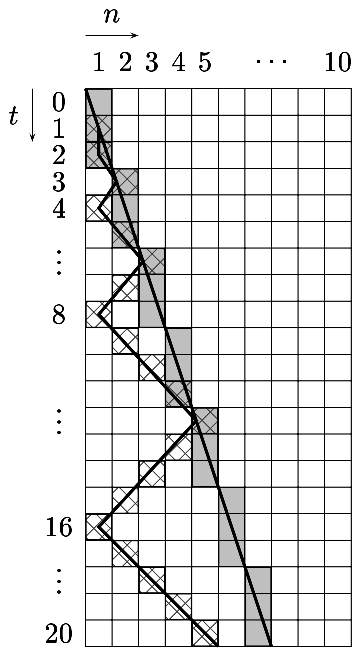

Figure 1: A time constructor for 2^n.

A general investigation of time constructible functions can be found, e.g. in [10, 2].

Actually, we will need a time constructor for the mapping 2^{2^n}. Fortunately, in [10] the closure of these functions under composition has been shown.

Corollary 5 *The mapping $f(n) = 2^{2^n}$, $n \in \mathbb{N}$, is time constructible.*

4.1.2 Binary OCA-counters

Here we need to set up some adjacent cells of an OCA as a binary counter. Actually, we are not interested in the value of the counter but in the time step at which it overflows. Due to the information flow the rightmost cell of the counter has to contain the least significant bit. Assume that this cell can identify itself. In order to realize such a simple counter every cell has three registers (cf. Figure 2). The third ones are working modulo 2. The second ones are signaling a carry-over to the left neighbor and the first ones are indicating whether the corresponding cell has generated no carry-over (0), one carry-over (1) or more than one carry-over (2) before. Now the whole counter can be tested by a leftmoving signal. If on its travel through the counter all the first registers are containing 0 and additionally both carry-over registers of the leftmost cell are containing 1, then it recognizes the desired time step. Observe that we need the second carry-over register in order to check that the counter produces an overflow for the first time.

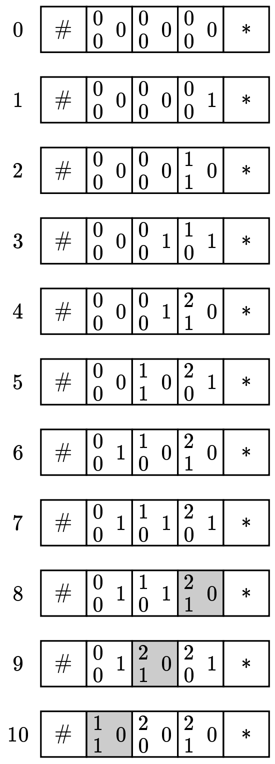

Figure 2: A binary OCA-counter.

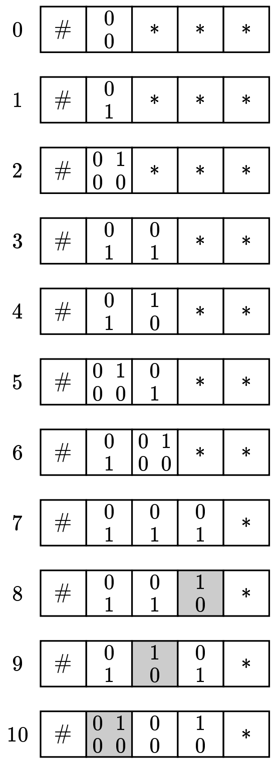

Figure 3: A binary CA-shift-right-counter.

4.1.3 Binary CA-shift-right counters

For this type of counter we need two-way information flow. It is set up in a single (the leftmost) cell of a CA. Since we require the least significant bit to be again the rightmost bit in the counter we have to extend it every time the counter produces an overflow. The principle is depicted in Figure 3.

Every cell has two registers. One for the corresponding digit and the other one for the indication of a carry-over. Due to the two-way information flow the leftmost cell can identify itself. Every time it generates a carry-over the counter has to be extended. For this purpose the leftmost cell simulates an additional cell to its left appropriately. This fact signals its right neighbor the need to extend the counter by one cell. The right neighbor reacts by simulating in addition the old process of the leftmost cell which now computes the new most significant bit. After the arrival of this extension signal at the rightmost cell of the counter the extension is physically performed by the first cell at the right of the counting cells which now computes the least significant bit.

Obviously, it can be checked again by a leftmoving signal whether the counter represents a power of 2 or not.

4.2 Proof of the strictness of the inclusion

Now we are prepared to prove the main result of this section.

Let a set of strings L be defined as follows:

$$L = \{b^n a^m \mid m = 2^{2^n} + 2^n, n \in \mathbb{N}\}$$

The easy part is to show that L does not belong to $\mathscr{L}_{rt}(\text{OCA})$.

Lemma 6 $L \notin \mathscr{L}_{rt}(\text{OCA})$

Proof. In [5] it has been shown that for a mapping $f : \mathbb{N} \to \mathbb{N}$ with the property

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

the set of strings $\{b^n a^{f(n)} \mid n \in \mathbb{N}\}$ does not belong to $\mathscr{L}_{rt}(\text{OCA})$. Applying the result to L it follows

$$\lim_{n\to\infty} \frac{n^{(n+1)^2}}{2^{2^n} + 2^n} = \lim_{n\to\infty} \frac{2^{(n+1)^2 \cdot \log_2(n)}}{2^{2^n} + 2^n} = 0$$

and therefore $L \notin \mathscr{L}_{rt}(\text{OCA})$. □

It remains to show that L is real-time recognizable by some mO-CA.

Theorem 7 $L \in \mathscr{L}_{rt}(\text{1O-CA})$

Proof. In the following a real-time 1O-CA $\mathcal{M}$ that recognizes L is constructed. On input data $b^n a^m$ we are concerned with three possible positions of the unique defective cell:

1. The position is within the b-cells.
2. The position is within the leftmost 2^n a-cells.
3. The position is at the right hand side of the 2^nth a-cell.

At the beginning of the computation $\mathcal{M}$ starts the following tasks in parallel on some tracks: The unique defective cell establishes a time constructor $\mathcal{M}_1$ for 2^{2^n} if it is an a-cell. The leftmost a-cell establishes another time constructor $\mathcal{M}_2$ for 2^{2^n} and, additionally, a binary shift-right counter $\mathcal{C}_1$ that counts the number of a's. The rightmost b-cell starts a binary OCA-counter $\mathcal{C}_2$ and, finally, the rightmost a-cell sends a stop signal with speed 1 to the left.

According to the three positions the following three processes are superimposed.

Case 3. (cf. Figure 4) The shift-right counter is increased by 1 at every time step until the stop signal $*$ arrives. Each overflow causes an incrementation (by 1) of the counter $\mathcal{C}_2$. Let i be the time step at which the stop signal arrives at the shift-right counter $\mathcal{C}_1$ and let l be the number of digits of $\mathcal{C}_1$. During the next l time steps the signal travels through the counter and tests whether its value is a power of 2, from which $i = 2^l$ follows. Subsequently, the signal tests during another n time steps whether the value of the binary counter $\mathcal{C}_2$ is exactly 2^n, from which $l = 2^n$ follows. If the tests are successful the input is accepted because the input string is of the form $b^n a^l a^i = b^n a^{2^n} a^{2^l} = b^n a^{2^n} a^{2^{2^n}}$ and thus belongs to L.

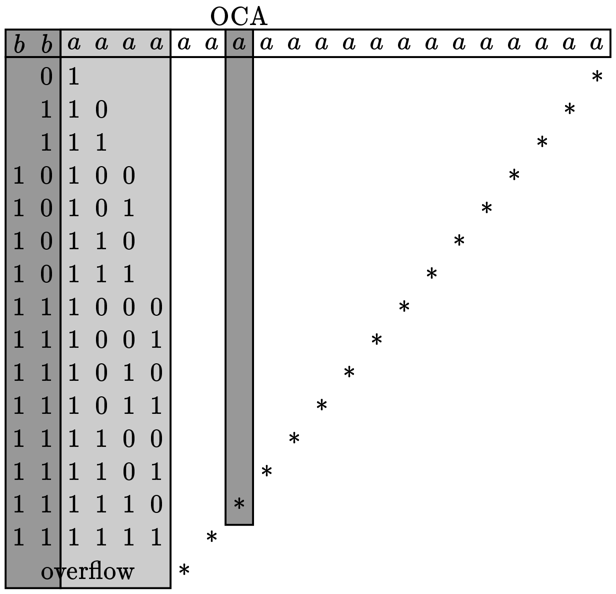

Figure 4: Example for case 3, $w = b^2 a^{2^{2^2}+2^2}$.

Case 2. (cf. Figure 5) In this case the space between the b's and the defective cell is too small for setting up an appropriate counter as shown for case 3. Here a second binary counter $\mathcal{C}_3$ within the b-cells is used. It is increased by 1 at every time step until it receives a signal from the defective cell and, thus, contains the number of cells between the b-cells and the defective cell. Its value x is conserved on an additional track. Moreover, at every time step at which the time constructor $\mathcal{M}_1$ marks the defective cell to be final, a signal is sent to the b-cells that causes them to reset the counter to the value x by copying the conserved value back to the counter track. After the reset the counter is increased by 1 at every time step respectively. The reset signals also mark an unmarked b-cell respectively.

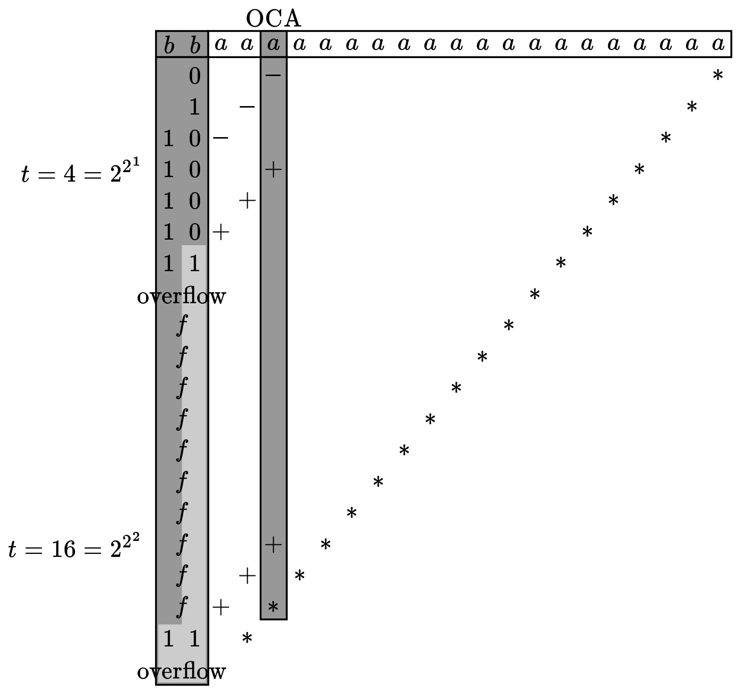
OCA
b b a a a a a a a a a a a a a a a a a a a a
$t = 4 = 2^{2^1}$
$t = 16 = 2^{2^2}$
overflow
overflow

Figure 5: Example for case 2, $w = b^2 a^{2^{2^2}+2^2}$.

The input is accepted if exactly at the arrival of the stop signal at the leftmost cell the counter overflows for the first time and all b-cells are marked: Let the last marking of $\mathcal{M}_1$ happen at time $i = 2^{2^r}$ for some $r \in \mathbb{N}$. The corresponding leftmoving signal arrives at time $2^{2^r} + x$ at the b-cells and resets the counter $\mathcal{C}_3$ to x. The stop signal arrives at time $2^{2^r} + x + s$, for some $s \in \mathbb{N}$, at the counter that has now the value $x + s$. Since the counter produces an overflow it holds $x + s = 2^n$. Moreover, since $\mathcal{M}_1$ has sent exactly r marking signals and all b-cells are marked it follows $r = n$. Therefore, the stop signal arrives at the rightmost b-cell at time $2^{2^r} + x + s = 2^{2^n} + 2^n$ and the input belongs to L.

Case 1. Since the binary counter $\mathcal{M}_3$ within the b-cells is an OCA-counter it works fine even if the defective cell is located within the b-cells. Case 1 is a straightforward adaption of case 2 (here $\mathcal{M}_2$ is used instead of $\mathcal{M}_1$). □

Corollary 8 $\mathscr{L}_{rt}(\text{OCA}) \subset \mathscr{L}_{rt}(\text{1O-CA})$

Without proof we present the following generalization:

Theorem 9 *Let $m \in \mathbb{N}$ be some constant then $L \in \mathscr{L}_{rt}(m\text{O-CA})$.*

Corollary 10 *Let $m \in \mathbb{N}$ be some constant then $\mathscr{L}_{rt}(\text{OCA}) \subset \mathscr{L}_{rt}(m\text{O-CA})$.*

5 CAs are better than mO-CAs

In order to complete the comparisons we have to prove that the computational power of real-time mO-CAs is strictly weaker than those of CAs. For this purpose we can adapt a method developed in [12] for proving that certain string sets do not belong to $\mathscr{L}_{rt}(\text{OCA})$. The basic idea in [12] is to define an equivalence relation on string sets and bound the number of distinguishable equivalence classes of real-time OCA computations.

Let $\mathcal{M} = \langle S, \delta, \#, A \rangle$ be an OCA and $X, Y \subseteq A^*$. Two strings $w, w' \in A^*$ are defined to be $(\mathcal{M}, X, Y)$-equivalent iff for all $x \in X$ and $y \in Y$ the leftmost $|x| + |y|$ states of the configurations $\Delta^{[|w|]}(c_{0,xwy})$ and $\Delta^{[|w'|]}(c_{0,xw'y})$ are equal (cf. Figure 6).

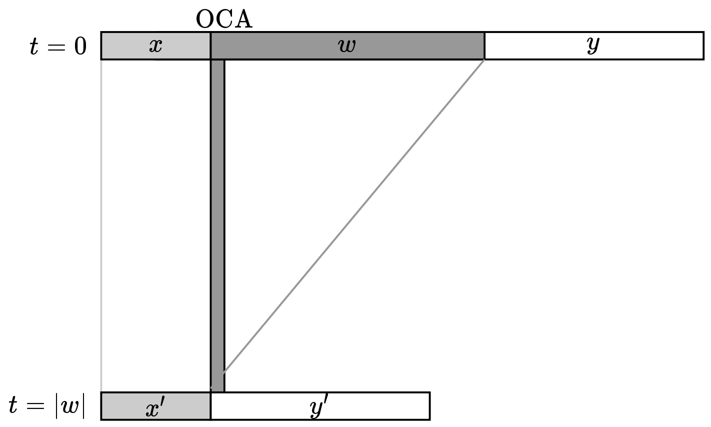

Figure 6: Principle of bounding real-time equivalence classes.

The observation is that the essential point of the upper bound on equivalence classes is due to the fact that the input sequences x and y are computational unrelated. Therefore, we can assume that the cell obtaining the first symbol of w resp. of w' as input is defective and so adapt the results in [12] to 1O-CAs immediately:

Lemma 11 *$\{uvu \mid u, v \in \{0,1\}^*, |u| > 1\} \notin \mathscr{L}_{rt}(\text{1O-CA})$ and $\{uvu \mid u, v \in \{0,1\}^*, |u| > 1\} \in \mathscr{L}_{rt}(\text{CA})$.*

Corollary 12 *Let $m \in \mathbb{N}$ be some constant then $\{uvu \mid u, v \in \{0,1\}^*, |u| > 1\} \notin \mathscr{L}_{rt}(m\text{O-CA})$.*

Corollary 13 *Let $m \in \mathbb{N}$ be some constant then $\mathscr{L}_{rt}(m\text{O-CA}) \subset \mathscr{L}_{rt}(\text{CA})$.*

Finally, it follows for a constant $m \in \mathbb{N}$:

$$\mathscr{L}_{rt}(\text{OCA}) \subset \mathscr{L}_{rt}(m\text{O-CA}) \subset \mathscr{L}_{rt}(\text{CA})$$

References

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

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

[3] Harao, M. and Noguchi, S. *Fault tolerant cellular automata.* Journal of Computer and System Sciences 11 (1975), 171–185.

[4] Jiang, T. *The synchronization of nonuniform networks of finite automata.* Information and Computation 97 (1992), 234–261.

[5] Kutrib, M. *Pushdown cellular automata.* Theoretical Computer Science 215 (1999), 239–261.

[6] Kutrib, M. and Löwe, J.-T. *Fault tolerant parallel pattern recognition.* Cellular Automata for Research and Industry (ACRI 2000), 2000, to appear.

[7] Kutrib, M. and Vollmar, R. *Minimal time synchronization in restricted defective cellular automata.* Journal of Information Processing and Cybernetics EIK 27 (1991), 179–196.

[8] Kutrib, M. and Vollmar, R. *The firing squad synchronization problem in defective cellular automata.* IEICE Transactions on Information and Systems E78-D (1995), 895–900.

[9] Mazoyer, J. *Synchronization of a line of finite automata with nonuniform delays.* Research Report TR 94-49, Ecole Normale Supérieure de Lyon, Lyon, 1994.

[10] Mazoyer, J. and Terrier, V. *Signals in one dimensional cellular automata.* Theoretical Computer Science 217 (1999), 53–80.

[11] Nishio, H. and Kobuchi, Y. *Fault tolerant cellular spaces.* Journal of Computer and System Sciences 11 (1975), 150–170.

[12] Terrier, V. *Language not recognizable in real time by one-way cellular automata.* Theoretical Computer Science 156 (1996), 281–287.

[13] Umeo, H. *A fault-tolerant scheme for optimum-time firing squad synchronization.* Parallel Computing: Trends and Applications, 1994, pp. 223–230.

[14] von Neumann, J. *Probabilistic logics and the synthesis of reliable organisms from unreliable components.* Automata Studies, Princeton University Press 34 (1956), 43–98.

[15] Yunès, J.-B. *Fault tolerant solutions to the firing squad synchronization problem.* Technical Report LITP 96/06, Institut Blaise Pascal, Paris, 1996.