I F I G
Research
Report

Institut für Informatik

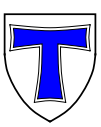
Massively Parallel Fault Tolerant Computations on Syntactical Patterns

Martin Kutrib Jan Thomas Löwe

IFIG Research Report 0102
March 2001

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 0102, March 2001

Massively Parallel Fault Tolerant Computations on Syntactical Patterns

Jan Thomas Löwe[1] Martin Kutrib[2]

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

Abstract. The general capabilities of reliable computations in linear cellular arrays are investigated in terms of syntactical pattern recognition. We consider defects of the processing elements themselves and defects of their communication links. In particular, a processing element (cell) is assumed to behave as follows. Dependent on the result of a self-diagnosis it stores its working state locally such that it becomes visible to the neighbors. A defective cell cannot modify information but is able to transmit it unchanged with unit speed. Cells with link failures are not able to receive information via at most one of their both links to adjacent cells. Moreover, static and dynamic defects are distinguished.
It is shown that fault tolerant real-time 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 is proved that all failures can be compensated as long as the number of adjacent defective cells is bounded.
In case of arrays with link failures it is shown that the sets of patterns that are reliably recognizable are strictly in between the sets of (intact) one-way and (intact) two-way arrays.

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

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

Copyright © 2001 by the authors

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 [19] 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 nature developed mechanisms which invalids the errors, with other words, 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, homogenously structured computers are the cellular arrays. Such devices of interconnected parallel acting finite state machines have been studied from various points of view.

In [4, 5] reliable arrays are constructed under the assumption that a cell (and not its links) at each time step fails with a constant probability. Moreover, such a failure does not incapacitate the cell permanently, but only violates its rule of operation in the step when it occurs. Under the same 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 [6, 14] where encodings are established that allow the correction of so-called K-separated misoperations, in [9, 10, 17, 20] where the famous firing squad synchronization problem is considered in defective cellular arrays, and in terms of interacting automata with nonuniform delay in [7, 11] where the synchronization of the networks is the main object either.

Here we are interested in more general computations. In terms of pattern recognition the general capabilities of reliable computations are considered. Since cellular arrays have intensively been 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.

In the sequel we distinguish three different types of defects.

Static defects are the main object of Section 3. It is assumed that each cell has a self-diagnosis circuit 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 cannot modify information but is able to transmit it with unit speed. Otherwise the parallel computation would be broken into two non interacting parts and, therefore, would become impossible at all.

In Section 4 the defects are generalized. In cellular arrays with dynamic defects it may happen that a cell becomes defective at any time. The formalization of the corresponding arrays includes also the possibility to repair a cell dynamically.

The remaining sections concern another natural type of defects. Not the cells themselves cause the misoperation but their communication links. It is assumed that a defective cell is not able to receive information via at most one of its both links to adjacent cells. The corresponding model is introduced in Section 5 in more detail. In Section 6 it is shown that the real-time arrays with link failures are able to reliably recognize a wider range of sets of patterns than intact one-way arrays. In order to prove this result some auxiliary algorithmic subroutines are given. Section 7 concludes the investigations by showing that the devices with link failures are strictly weaker than two-way arrays. Hence, link failures cannot be compensated in general but, on the other hand, do not decrease the computing power to that one of 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 Preliminaries

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 notion 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 $|w|$. The set 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 each symbol of the input string $w = x_1 \cdots x_n$ is fed to one cell: $c_{0,w}(i) = x_i$, $1 \leq i \leq n$. 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:

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

for CAs and

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

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 $\mathcal{M}$ at time step $i \in \mathbb{N}$, if $c_i(1) \in F$ for the configuration $c_i = \Delta^{[i]}(c_{0,w})$.*
2. *$L(\mathcal{M}) = \{w \in A^+ \mid \mathcal{M}$ recognizes w at some time step$\}$ 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. If $\mathcal{M}$ can recognize all $w \in L(\mathcal{M})$ within at most $t(|w|)$ time steps, then $\mathcal{M}$ 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 $\mathscr{L}_t(\text{CA})$ ($\mathscr{L}_t(\text{OCA})$). 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 $\mathscr{L}_{rt}((\text{O})\text{CA})$ and $\mathscr{L}_{lt}((\text{O})\text{CA})$. In the sequel we will use corresponding notations for other types of recognizers.

3 Static Defects

Now we are going to explore some general recognition capabilities of CAs that contain some defective cells. The defects are in some sense static [17]: It is assumed that each cell has a self-diagnosis circuit which is run once before the actual computation. The result of that diagnosis is stored in a special register of each cell 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 cell? It is reasonable to require that a defective cell cannot modify information. On the other hand, it must be able to transmit information in order to avoid the parallel computation being broken into two not interacting lines and, thus, being impossible at all.

The speed of information transmission is one cell per time step. Another point of view on such devices is to define a transmission delay between every two adjacent cells and to allow nonuniform delays [7, 11]. Now the number of defective cells between two intact ones determine the corresponding delay.

Since the self-diagnosis is run before the actual computation we may assume that defective cells do not fetch an input symbol. Nevertheless, real-time is the minimal possible time needed for non-trivial computations and, consequently, is defined to be the number of all cells in the array. In order to obtain a computation result here we require the leftmost cell not to be defective. Later on we can omit this assumption.

Formally we denote CAs with static defects by SD-CA and the corresponding language families by $\mathscr{L}_t$(SD-CA).

Considering the general real-time recognition capabilities of SD-CAs the best case is trivial. It occurs when all the cells are intact: The capabilities are those of CAs. On the other hand, fault tolerant computations are concerned with the worst case (with respect to our assumptions on the model). The next two results show that in such cases the capabilities can be characterized by intact OCAs from what follows that the bidirectionality of the information flow gets lost.

Theorem 3 *If a set is fault tolerant real-time recognizable by a* SD-CA, *then it is real-time recognizable by an* OCA.

Proof. Let $\mathcal{D}$ be a SD-CA and let $k \in \mathbb{N}$ be an arbitrary positive integer. Set the number of cells of $\mathcal{D}$ to $n = 2^k - 1$. For the mapping $f : \mathbb{Z} \to \mathbb{Z}$, $f(z) = n - 2^z + 2$ holds: $\forall\, z \in \{1, \ldots, k\} : f(z) \in \{1, \ldots, n\}$.

Now assume the cells at the positions $f(i)$, $1 \leq i \leq k$, are intact ones and all the other cells are defective (cf. Figure 1). In between the cells $f(i)$ and $f(i+1)$, $1 \leq i \leq k-1$, there are $f(i) - f(i+1) - 1 = (n - 2^i + 2) - (n - 2^{i+1} + 2) - 1 = 2^{i+1} - 2^i - 1 = 2^i - 1$ defective ones.

During a real-time computation the states of a cell $f(i)$ at time $t \geq 2^i$ cannot influence the overall computation result. The states would reach the leftmost

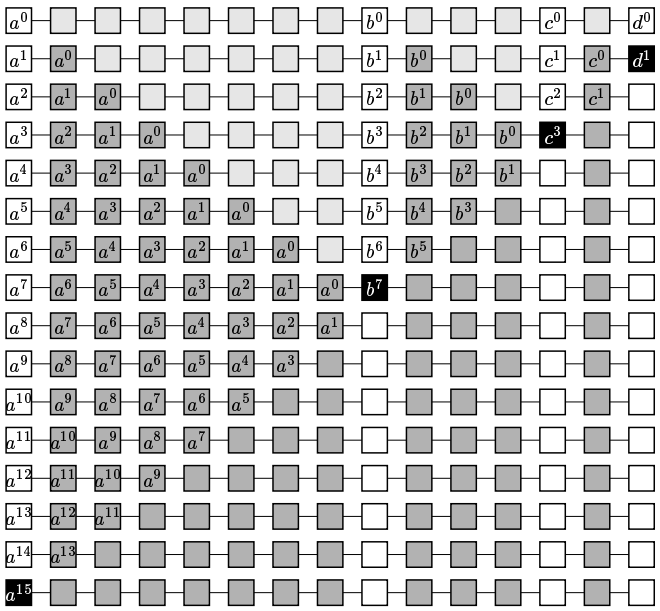


Figure 1: One-way information flow in SD-CAs.

cell after another $f(i) - 1 = (n - 2^i + 2) - 1 = 2^k - 1 - 2^i + 1 = 2^k - 2^i$ time steps. This gives the arrival time $2^i + 2^k - 2^i = 2^k = n + 1$, which is greater than real-time.

Conversely, the cell $f(i)$ computes all its states up to time $t \leq 2^i - 1$ independently of the states of its intact neighbors to the left: The nearest intact neighbor to the left is cell $f(i+1)$ and there are $2^i - 1$ defective cells in between $f(i+1)$ and $f(i)$.

Up to now we have shown that the information flow in $\mathcal{D}$ is one-way. But compared to OCAs the cells in $\mathcal{D}$ are performing more state changes. It remains to show that this does not lead to stronger capabilities.

Let i be some intact cell of $\mathcal{D}$. As long as it operates independently on its intact neighbors it runs through state cycles provided that the adjacent defective regions are long enough. Let $s_0 s_1 \cdots s_j s_{j+1} \cdots s_{j+k} s_j \cdots$ be such a cycle. Now one can always enlarge the lengths of the defective regions such that they correspond to $j + p \cdot (k+1)$, $p \in \mathbb{N}_0$.

Therefore, during their isolated computations the cells run through complete cycles. Obviously, such a behavior can be simulated by the cells of an OCA since the cycle lengths are bounded by the number of states of $\mathcal{D}$. □

In order to obtain the characterization of real-time SD-CAs by real-time OCAs we need the converse of Theorem 3.

Theorem 4 *If a set is real-time recognizable by an* OCA*, then it is fault tolerant real-time recognizable by a* SD-CA*.*

Proof. The idea of the simulation is depicted in Figure 2. Each cell of a SD-CA that simulates a given OCA waits for the first information from its

right intact neighbor. The waiting period is signaled to its left intact neighbor by signals labeled $*$. This information leads to a waiting period of the left intact neighbor. Each intact cell performs a simulation step when it receives a non-waiting signal.

It follows that a cell sends exactly as many waiting signals to the left as are defective cells located to its right. Therefore, the leftmost cell needs exactly one simulation step for each intact cell and one waiting step for each defective cell and, thus, computes the result in real-time. □

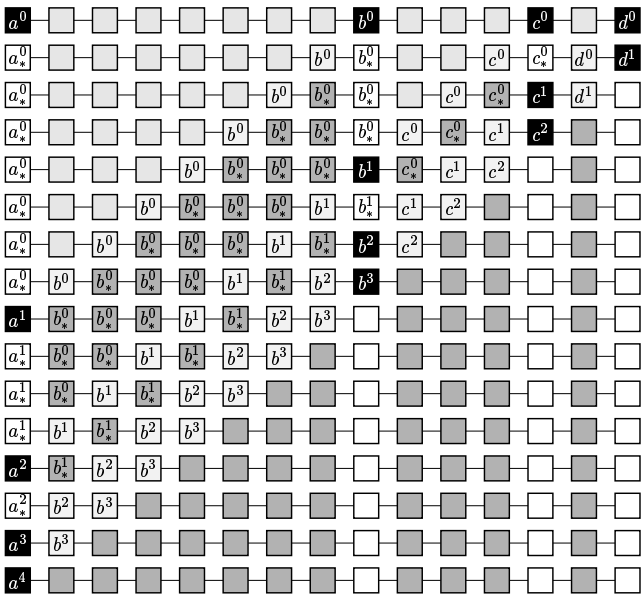


Figure 2: OCA simulation by SD-CAs.

The following corollary formalizes the characterization:

Corollary 5 $\mathscr{L}_{rt}(\text{SD-CA}) = \mathscr{L}_{rt}(\text{OCA})$

From the previous results follows the interesting fact that OCAs are per se fault tolerant. Additional defective cells do not decrease the recognition capabilities.

Corollary 6 $\mathscr{L}_{rt}(\text{SD-OCA}) = \mathscr{L}_{rt}(\text{OCA})$

It is often useful to have examples for string sets not recognizable by a certain device.

Example 7 Neither the set of duplicated strings $\{ww \mid w \in A^+\}$ nor the set of strings whose lengths are a power of 2 $\{w \mid w \in A^+ \text{ and } |w| = 2^i, i \in \mathbb{N}\}$ are fault tolerant real-time recognizable by SD-CAs.

(It has been shown in [13] resp. [15] that they do not belong to the family $\mathscr{L}_{rt}(\text{OCA})$.)

The previous results imply another natural question. Is it possible to regain the recognition power of two-way CAs in fault tolerant SD-CA computations by increasing the computation time? How much additional time would be necessary? In [3, 18] it has been shown that an OCA can simulate a real-time CA computation in twice real-time if the input word is reversed. It is an open problem whether or not $\mathscr{L}_{rt}(\mathrm{CA})$ is closed under reversal. But nevertheless, a piece of good news is that only one additional time step for each intact cell is necessary in order to regain the computation power in a fault tolerant manner.

Theorem 8 *A set is recognizable by an* OCA *in twice real-time (and, thus, its reversal by an* CA *in real-time) if and only if it is fault tolerant recognizable by a* SD-CA *in real-time+m, where m denotes the number of intact cells.*

Proof. Let L be a set that is recognizable by an OCA in twice real-time and denote the length of an input word by n. Let us consider the situation in the proof of Theorem 4 at real-time: The corresponding SD-CA has simulated n steps of the corresponding OCA. But due to the previous delay, from now on no further delay of the intact cells is necessary. Thus, the next n steps of the OCA can be simulated in n steps by the SD-CA either. Since the number of intact cells is exactly n the only if part follows.

Now let L be a set that is recognizable by a SD-CA in time real-time+m. We follow the same idea as in the proof of Theorem 3. The distance between two intact cells has to be enlarged. If we number the intact cells from right to left by $1, 2, \ldots, m$ then the distance between cells i and $i+1$ has to be enlarged at least by $2^{i-1} \cdot (m-1)$ in order to prevent additional flow of information. □

One assumption on our model has been an intact leftmost cell. Due to Corollary 5 we can omit this requirement. Now the overall computation result is indicated by the leftmost intact cell of the one-way array which operates per se independently on its defective left neighbors.

4 Dynamic Defects

In the following cellular arrays with dynamic defects (DD-CA) are introduced. Dynamic defects can be seen as generalization of static defects. Now it becomes possible that cells fail at any time during the computation. Afterwards they behave as in the case of static defects.

In order to define DD-CAs more formally it is helpful to suppose that the state of a defective cell is a pair of states of an intact one. One component represents the information that is transmitted to the left and the other one the information that is transmitted to the right. By this formalization we obtain the type indication of the cells (defective or not) for free: Defective cells are always in states from S^2 and intact ones in states from S. A possible failure implies a weak kind of nondeterminism for the local transition function.

Definition 9 *A two-way cellular array with dynamic defects (*DD-CA*) is a system* $\langle S, \delta, \#, A\rangle$*, where*

1. S *is the finite, nonempty set of* cell states *which satisfies* $S \cap S^2 = \emptyset$,
2. $\# \notin S$ *is the* boundary state,
3. $A \subseteq S$ *is the set of* input symbols,
4. $\delta : (S \cup \{\#\} \cup S^2)^3 \to \{\{a, (b, c)\} \mid a, b, c \in S\}$ *is the* local transition function *which satisfies* $\delta(s_1, s_2, s_3) = \{s, (s_l, s_r)\}$ *with* $s \in S$, $(s_1 = s_l \in S \vee s_1 = (s_l, s'_r) \in S^2)$ *and* $(s_3 = s_r \in S \vee s_3 = (s'_l, s_r) \in S^2)$.

If a cell works fine the local transition function maps to a state from S. Otherwise it maps to a pair from S^2 indicating that the cell is now defective. The definition includes the possibility to repair a cell during the computation. In this case δ would map from a pair to a state from S. Note that the nondeterminism in a real computation is a determinism since the failure or repair of a cell is in some sense under the control of the outside world.

We assume that initially all cells are intact and as in the static case that the leftmost cell remains intact.

In the sequel we call an adjacent subarray of defective cells a *defective region.*

The next results show that dynamic defects can be compensated as long as the lengths of defective regions are bounded.

Theorem 10 *If a set is real-time recognizable by a* CA*, then it its real-time recognizable by a* DD-CA *if the lengths of its defective regions are bounded by some* $k \in \mathbb{N}_0$.

Proof. Assume for a moment that the lengths of the defective regions are exactly k. A DD-CA $\mathcal{D}$ that simulates a given CA $\langle S, \delta, \#, A\rangle$ has the state set $S' = S^{4k+1}$.

The general idea of the proof is depicted in Figure 3. As long as a cell does not detect a defective neighbor it stores the states of its neighbors and its own state in some of its additional registers as shown in the figure.

At time t the state of cell i might be as follows:

$$(\ldots, c_{t-1}(i-1), c_{t-1}(i), \underbrace{c_t(i)}_{\text{center}}, c_{t-1}(i), c_{t-1}(i+1), \ldots)$$

Assume now that the right neighbor of cell i becomes defective. Due to our assumption we know that there must exist a defective region of length k at the right of cell i. During the next k time steps cell i stores the received states and computes missing states from its register contents as shown in Figure 3.

Subsequently its state might be as follows

$$(\ldots, \underbrace{c_{t+k}(i)}_{\text{center}}, c_{t+k-1}(i), c_{t+k-1}(i+1), \ldots, c_{t+1}(i+k-2), c_t(i+k-1), c_t(i+k))$$

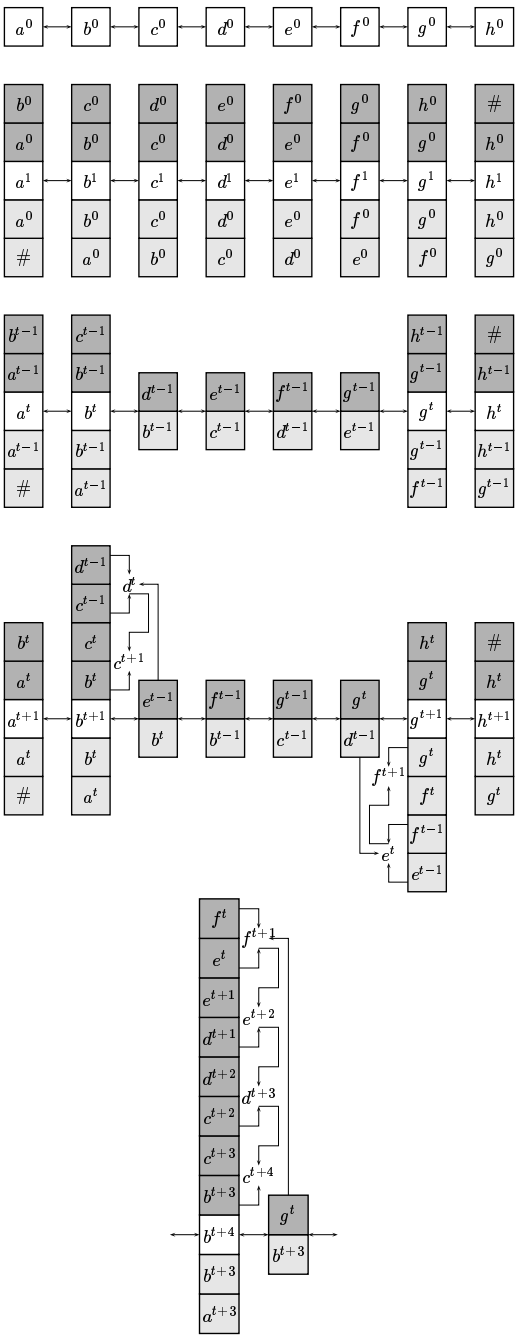


Figure 3: Compensation of $k = 4$ defects.

From now on cell i receives the states that the intact cell $i + k + 1$ has been in at time $t, t + 1, \ldots$ and is able to compute the necessary intermediate states from its register contents.

A crucial point is that the lengths of defective regions are fixed to k. Due to that assumption a cell i knows when it receives the valid states from its next intact neighbor $i+k+1$ or $i-k-1$. We can relax the assumption as required to

lengths of at most k cells by the following extension of the simulation. Each cell is equipped with a modulo k counter. Since the current value of the counter is part of the cell state it is also part of the transmitted information. A cell that stores received information in its additional registers stores also the received counter value. Now it can decide whether it receives the valid state from its next intact neighbor by comparing the received counter value to the latest stored counter value. If they are equal then the received information is from a defective cell, otherwise it is valid and the cell uses no more additional registers.

New failures in subsequent time steps can be detected by the same method. If the received counter value is equal to the latest stored counter value then additional cells have become defective. In such cases the cell uses correspondingly more additional registers in order to compensate the new defects.

It remains to explain what happens if two defective regions are joint by failure of a single connecting intact cell. Up to now we have used the transmitted contents of the main registers only. But actually the whole state, i.e. all register contents, are transmitted. In the case in question the next intact cells to the left and right of the joint defective region can fill additional registers as desired. □

Corollary 11 *If a set is real-time recognizable by an* OCA, *then it is real-time recognizable by a* DD-OCA *if the lengths of its defective regions are bounded by some* $k \in \mathbb{N}_0$.

In order to provide evidence for general fault tolerant DD-CA computations we have to relax the assumption of bounded defective region lengths. We are again concerned with the worst case. The hardest scenario is as follows. Initially all cells are intact and thus fetching an input symbol. During the first time step all but the leftmost cell fail. (Needless to say, if the leftmost cell becomes also defective then nobody would expect a reasonable computation result.)

It is easy to see that in such cases the recognition capabilities of DD-CAs are those of a single cell, a finite state machine (see Figure 4).

Lemma 12 *If a set is fault tolerant recognizable by a* DD-CA, *then it is recognizable by a finite state machine and thus regular.*

Corollary 13 *If a set is fault tolerant recognizable by a* DD-OCA, *then it is recognizable by a finite state machine and thus regular.*

5 Devices with Link Failures

In Section 3 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

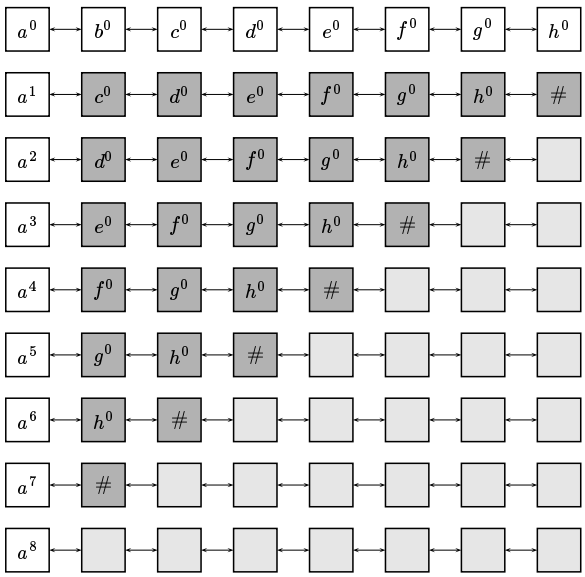


Figure 4: Worst case DD-CA computation.

reliable recognition capabilities we have to take a closer look on the device in question.

Again we assume that each cell has a self-diagnosis circuit for its links which is run once before the actual computation, and again the result of that diagnosis is indicated by the states of the cells such that intact cells can detect defective neighbors. What is the effect of a defective link? Suppose that each two adjacent cells are interconnected by two unidirectional links. 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.

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 14 *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.

A reliable 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 an mO-CA if there exists a set $D \subseteq \{1, \ldots, n\}$ of defective cells, $|D| \leq m$, such that $c_{0,w}(i) = x_i \in S_i$ if $i \in \{1, \ldots, n\} \setminus D$ 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 an mO-CA with defective cells D, then its successor configuration is as follows:

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

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 answer the following questions. 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 shown for CAs with defective cells? Can mO-CAs recognize a wider range of string sets than intact OCAs?

6 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.

6.1 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.

6.1.1 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 15 *The mapping $f(n) = 2^n$, $n \in \mathbb{N}$, is time constructible.*

Proof. The idea of the proof is depicted in Figure 5. 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. □

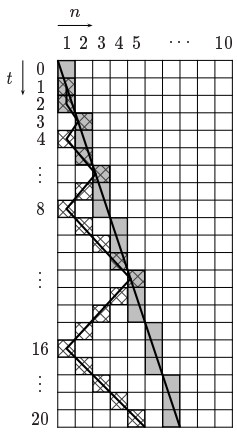


Figure 5: A time constructor for 2^n.

A general investigation of time constructible functions can be found, e.g. in [12, 2]. Especially, there exists a time constructor for 2^n that works with an area of at most n cells.

Actually, we will need a time constructor for the mapping 2^{2^n}. Fortunately, in [12] the closure of these functions under composition has been shown.

Corollary 16 *The mapping $f(n) = 2^{2^n}$, $n \in \mathbb{N}$, is time constructible. There exists a corresponding time constructor that works with an area of at most 2^n cells.*

6.1.2 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 6). 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.

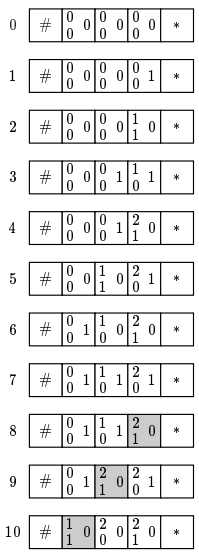


Figure 6: A binary OCA-counter.

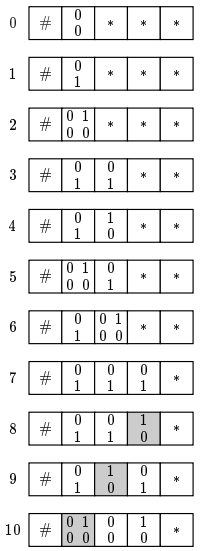


Figure 7: A binary CA-shift-right-counter.

6.1.3 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 the counter every time it produces an overflow. The principle is depicted in Figure 7.

Each 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.

6.2 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 17 $L \notin \mathscr{L}_{rt}(\text{OCA})$

Proof. In [8] 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 we obtain

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

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

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

Theorem 18 $L \in \mathscr{L}_{rt}(1\text{O-CA})$

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 8) 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.

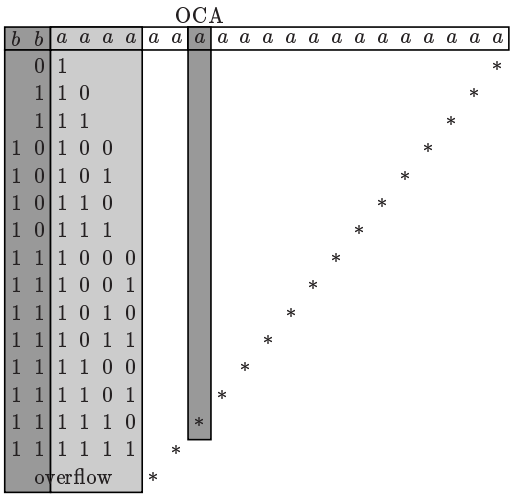


Figure 8: Example for case 3, $w = b^2 a^{2^{2^2}+2^2}$.

Case 2. (cf. Figure 9) 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. Each reset signal also marks an unmarked b-cell.

The input is accepted if exactly at the arrival of the stop signal at the leftmost

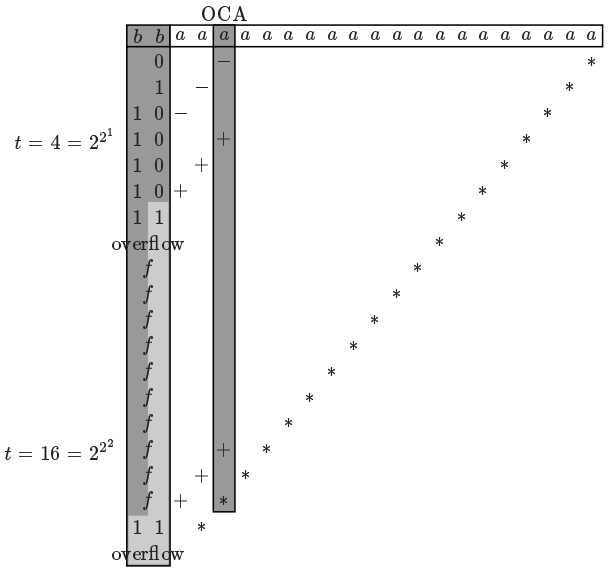


Figure 9: Example for case 2, $w = b^2 a^{2^{2^2}+2^2}$.

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 19 $\mathscr{L}_{rt}(\text{OCA}) \subset \mathscr{L}_{rt}(1\text{O-CA})$

This result can be generalized to devices with a bounded number of defective cells as follows.

Theorem 20 *Let $m \in \mathbb{N}$ be some constant then $L \in \mathscr{L}_{rt}(m\text{O-CA})$.*

Proof. If all defective cells are located within the b-cells, or within the first 2^n a-cells or within the last 2^{2^n} a-cells, then the proof follows from the three cases in the proof of Theorem 18. Otherwise we are concerned with two more cases:

Case 4: The leftmost defective a-cell is not within the first $\frac{1}{m}2^n$ a-cells. By standard compression techniques we can simulate m cells by one. Now the construction of Case 3 in the proof of Theorem 18 solves this case.

Case 5: The leftmost defective a-cell is within the first $\frac{1}{m}2^n$ a-cells. In order to adapt Case 2 of the previous proof due to Corollary 16 we may assume the

time constructor for 2^n works in at most n cells. By standard compression techniques we obtain at most $\frac{1}{m}2^n$ cells.

According to our assumption not all defective cells are within the first 2^n a-cells. So we have two defective cells with a distance of more than $\frac{1}{m}2^n$ intact cells. This allows us to use the compressed time constructor in that area. The remaining construction has been shown in Case 2.

At the very beginning of the computation it is not know which area of the possible m areas is the related one. In order to determine it we start time constructors at each defective cell. Now we are able to choose a correct track of a successful computation and the set is recognized if one of the m time constructors recognizes the input. □

Corollary 21 *Let $m \in \mathbb{N}$ be some constant then $\mathscr{L}_{rt}(\text{OCA}) \subset \mathscr{L}_{rt}(m\text{O-CA})$.*

7 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 [16] for proving that certain string sets do not belong to $\mathscr{L}_{rt}(\text{OCA})$. The basic idea in [16] 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 10).

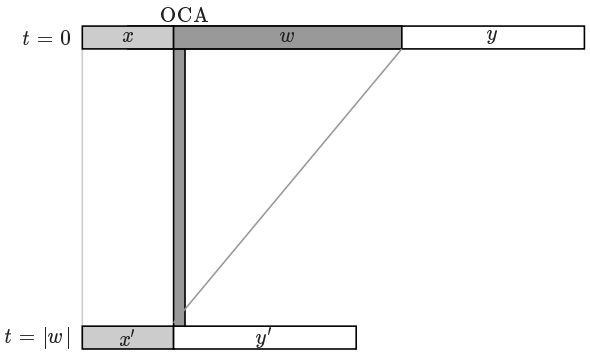


Figure 10: Principle of bounding real-time equivalence classes.

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 [16] to 1O-CAs immediately:

Lemma 22 *$\{uvu \mid u, v \in \{0,1\}^*, |u| > 1\} \notin \mathscr{L}_{rt}(1\text{O-CA})$ and $\{uvu \mid u, v \in \{0,1\}^*, |u| > 1\} \in \mathscr{L}_{rt}(\text{CA})$.*

Corollary 23 *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 24 *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] W. Bucher and K. Čulik II, On real time and linear time cellular automata, RAIRO Inform. Théor. 18 (1984) 307–325.

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

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

[4] P. Gács, Reliable computation with cellular automata, J. Comput. System Sci. 32 (1986) 15–78.

[5] P. Gács and J. Reif, A simple three-dimensional real-time reliable cellular array, J. Comput. System Sci. 36 (1988) 125–147.

[6] M. Harao and S. Noguchi, Fault tolerant cellular automata, J. Comput. System Sci. 11 (1975) 171–185.

[7] T. Jiang, The synchronization of nonuniform networks of finite automata, Inform. Comput. 97 (1992) 234–261.

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

[9] M. Kutrib and R. Vollmar, Minimal time synchronization in restricted defective cellular automata, J. Inform. Process. Cybern. EIK 27 (1991) 179–196.

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

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

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

[13] K. Nakamura, Real-time language recognition by one-way and two-way cellular automata, in: Proc. MFCS 1999, Lecture Notes in Computer Science, Vol. 1672 (Springer, Berlin, 1999) 220–230.

[14] H. Nishio and Y. Kobuchi, Fault tolerant cellular spaces, J. Comput. System Sci. 11 (1975) 150–170.

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

[16] V. Terrier, Language not recognizable in real time by one-way cellular automata, Theoret. Comput. Sci. 156 (1996) 281–287.

[17] H. Umeo, A fault-tolerant scheme for optimum-time firing squad synchronization, in: Proc. Parallel Computing: Trends and Applications (North-Holland, Amsterdam, 1994) 223–230.

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

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

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