Linear functional classes over cellular automata

Anaël Grandjean

École normale supérieure de Lyon
15 parvis René Descartes - BP 7000, 69 342 Lyon Cedex 07, FRANCE

GREYC, Université de Caen Basse-Normandie, CNRS,
Campus Côte de Nacre, Boulevard du Maréchal Juin, 14 000 Caen, France
`anael.grandjean@ens-lyon.fr`

Gaétan Richard   Véronique Terrier

GREYC, Université de Caen Basse-Normandie, CNRS,
Campus Côte de Nacre, Boulevard du Maréchal Juin, 14 000 Caen, France
`gaetan.richard@unicaen.fr`   `veronique.terrier@unicaen.fr`

Cellular automata are a discrete dynamical system which models massively parallel computation. Much attention is devoted to computations with small time complexity for which the parallelism may provide further possibilities. In this paper, we investigate the ability of cellular automata related to functional computation. We introduce several functional classes of low time complexity which contain "natural" problems. We examine their inclusion relationships and emphasize that several questions arising from this functional framework are related to current ones coming from the recognition context. We also provide a negative result which explicits limits on the information transmission whose consequences go beyond the functional point of view.

Introduction

Introduced in the forties to model self-replication [7], cellular automata are a discrete dynamical model composed of an infinite line of cells endowed with a state chosen among a finite alphabet. Dynamic is achieved in discrete time by applying uniformly and synchronously a local rule to each cell.

From the dynamical point of view, this system has been widely used to model phenomena issued from different fields of research. It is often cited as a representative of complex systems — systems that can exhibit a complex behavior even starting from simple rules. Thanks to its simple formal definition, many results have also been achieved over its dynamics (see [3]).

This model has also been studied *per se* as a theoretical model of massively parallel computation. For this purpose, one usually gives as input a *finite* word over the line of cells and waits until some predefined state occurs in the evolution to determine whether this word is accepted or not. The number of steps needed is considered as the *time*. Among all possible complexity classes, the mainly studied and interesting ones are real time — the minimal time needed to take into account the whole information — and linear time [8]. Central questions concerning their computational power and their limits remain unanswered.

To tackle such problems, a possible way is to extend the study of the model to the functional point of view. A first significant step was made by M. Kutrib and A. Malcher who have investigated iterative arrays (a variant of cellular automata) as transducers [5] and reported several interesting results. For the device they considered, both input and output modes are sequential. In this paper, we study the case where input and output are fed and retrieved in parallel and examine the corresponding small complexities classes.

After specifying different possible definitions of functional classes in Section 1, we give several meaningful examples in Section 2 along with some generic framework to build such algorithms. We present some closure properties, a linear acceleration algorithm, the basic relationships between classes and some links with classical questions on CA recognition ability in Section 3. In Section 4, we prove separation results over the classes by providing one specific impossibility of behavior for cellular automata. The latter result (Theorem 4) is of interest in itself and opens new perspectives to achieve negative results over cellular automata.

1 Definitions

Basically, a cellular automata (CA) is a one-dimensional array of finite automata (the cells) indexed by $\mathbb{Z}$. The cells range over a finite set S, *the set of states*, and evolve synchronously in discrete time steps. At each step, the state of each cell changes according to its own state and the states of its nearest neighbors. All cells have the same *local transition rule* f. Formally denoting (c,t) the cell c at time t and $\langle c,t \rangle$ its state, we have: $\langle c,t+1 \rangle = f(\langle c-1,t \rangle, \langle c,t \rangle, \langle c+1,t \rangle)$.

A *configuration* is the sequence of cell states at a given time. To represent the evolution of a cellular automaton starting from a given configuration, one convenient representation is a *space-time diagram* which consists in piling-up the configurations at successive time steps.

Viewed as a computational model, CA operate on finite words. Although different alternatives may be relevant, we make the following choices for the rest of the paper. First, we only consider parallel input mode. That means the input sequence w is supplied at initial time to the array: $\langle i,0 \rangle = w_i$ for $0 \leq i < |w|$. Second, we will assume that the computation is linearly bounded in space: only a fixed number of cells, equal to the length of the input, are active. In practice, when the input length is n, the cells not in range $\{0, \cdots, n-1\}$ will remain in a persistent state # during all the computation. Actually, this bound coincides with the space consumed by small time computations, i.e., those computations which attract our attention.

We also need to specify the output mode. Obviously, it depends on whether we are looking at recognition or functional computation. With its output of yes/no type, the recognition case is the simplest one. Before examining how the output could be retrieved in the functional case, we first recall the definitions related to recognition.

To use cellular automata as a recognizer, two subsets of the states set S are specified: the set of accepting states and the set of rejecting states. The cell indexed by 0 is chosen to be the output cell which determines the acceptance. So an input word $w \in S^*$ is said to be accepted (resp. rejected) in time $t \in \mathbb{N}$ if the cell 0 enters an accepting state (resp. a rejecting state) at time t; and for all time less than t, the output cell is neither in an accepting nor in a rejecting state. The language *recognized* by the automaton is the set of words it eventually accepts. A CA works within time $\tau : \mathbb{N} \to \mathbb{N}$ if every word w is accepted or rejected in time $\tau(|w|)$. Among the time complexities, the small ones — real time and linear time — are of major of interest.

Definition 1. A language $L \subset S^*$ is recognized in *real time* (resp. *linear time*) if it corresponds to the set of words w recognized by a cellular automaton in time $|w|$ (resp. $k|w|$ for some $k \in \mathbb{Q}$ with $k > 1$).

Since they have been introduced by A. R. Smith [8], a lot of work has been done to study these complexity classes in order to better understand parallel feature. Despite a number of interesting results (see [9] for a survey), the basic question whether real time and linear time classes differ or not, is still open.

One can also take as alternative definition the case where the output cell is located in the middle of the input word. The minimal time for the output cell to know the whole input, is reduced to half of the input length. In fact, this notion corresponds to real time CA restricted to one-way communication which is known to be strictly less powerful than real time CA with two-way communication.

Let us come now to the topic of our paper: the functional issue. When trying to formally define functional variant, one problem arises. For the real time complexity, the minimal time to obtain the whole information on the word differs according to the position of the cell inside the word. This gives birth to two variations over real functional time according whether we require the output to be synchronized. The resulting classes depicted in Figure 1 are defined as follows:

Definition 2. A function $\phi : I^* \to O^*$ is called *computable in strict real time* if there exists a cellular automaton (S, f) and a projection $\pi : S \to O$ such that, on any input word $w \in I^*$, we have for any $i \in [\![0, \frac{|w|-1}{2}]\!]$, $\pi(\langle i, |w| - i \rangle) = \phi(w)_i$ and for any $i \in [\![\frac{|w|-1}{2}, |w| - 1]\!]$, $\pi(\langle i, i+1 \rangle) = \phi(w)_i$.

Definition 3. A function $\phi : I^* \to O^*$ is said *computable in synchronous real time* if there exists a cellular automaton (S, f) and a projection $\pi : S \to O$ such that, on any input word $w \in I^*$, we have $\pi(\langle i, |w| \rangle) = \phi(w)_i$ for any $i \in [\![0, |w| - 1]\!]$.
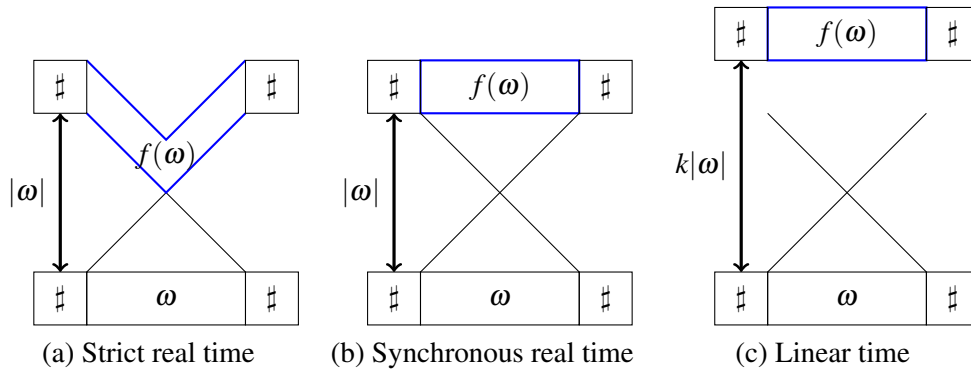

Figure 1: Functional classes over cellular automata

We will also consider linear time complexity (in this case, the two output variants are equivalent as proved in proposition 12).

Definition 4. A function $\phi : I^* \to O^*$ is said *computable in linear time* if there exists $k \in \mathbb{Q}$ with $k > 1$, a cellular automaton (S, f) and a projection $\pi : S \to O$ such that, on any input word $w \in I^*$, $\pi(\langle i, k|w| \rangle) = \phi(w)_i$ for any $i \in [\![0, |w| - 1]\!]$.

In the above definitions, we refer to sets of sites on which the outputs are displayed. In each case, the sequence of sites may be distinguished either, for strict real time, in making use of two signals initiated from each input extremity or, for synchronous time, by the way of *Firing Squad Synchronization* solutions (see [1, 6]). We also observe that, for both the real time complexities, it is possible to answer one step sooner by anticipating what would happen when the end of the input is reached. But the drawback is that we lose the capacity to explicit the set of output sites by marking them.

In addition, one can note that linear time is not affected by changing the input or output mode from parallel to sequential whereas this is not the case for (both) real time.

2   Examples

To start the study, we shall give several meaningful examples illustrating the range of functional classes and present some interesting (generic) algorithms.

One first easy remark is that functional classes are a generalization of detection.

Lemma 1. *If $L \subset I^*$ is a language recognizable in real (resp. linear) time then the function $f : I^* \to \{1,0\}^*$ which is defined by $f(w) = 10^{|w|-1}$ if $w \in L$ and $f(w) = 00^{|w|-1}$ otherwise is computable in strict real time (resp. linear time).*

Proof. It is sufficient to take the recognition automaton and send all output sites to 0 but the first one. □

Let us first look at some simple examples which still use the power of parallelism.

Example 2. *The mirror is computable in synchronous real time.*

Proof. Two possibilities are depicted in Figure 2. The left algorithm has been exhibited by M. Kutrib in [4]. The right one is a variant with symmetric features. □
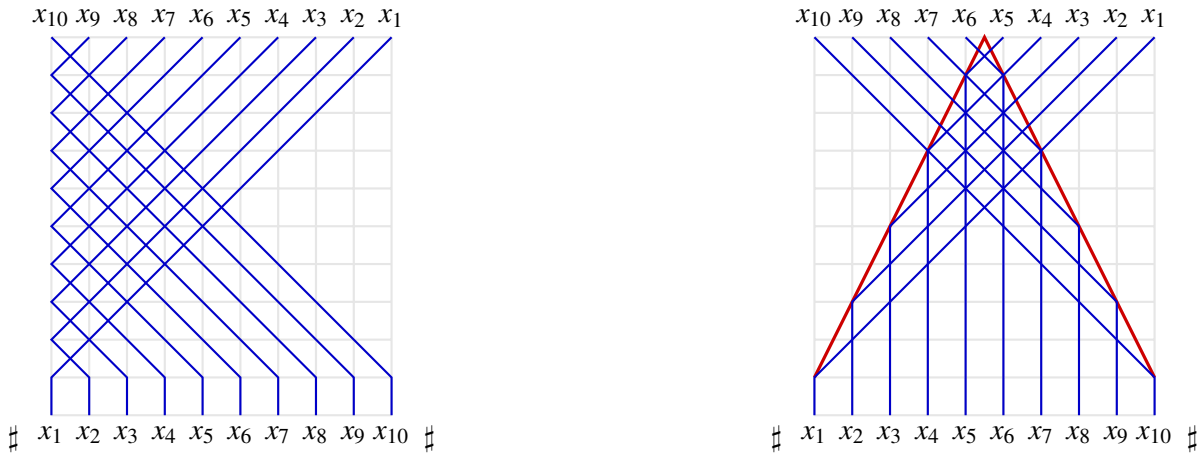

Figure 2: Computing the mirror in synchronous real time

Example 3. *The functions $f(u_1 \cdots u_n) = u_{n/2+1} \cdots u_n u_1 \cdots u_{n/2}$ and $g(u_1 \cdots u_n) = u_{n/2} \cdots u_1 u_n \cdots u_{n/2}$ are computable in strict real-time.*

Proof. The flow of data through the space time-diagram is depicted for the two functions f and g in Figure 3. For the construction of g, the space can be reduced to the bounded space of n cells in folding the figure.

Note that the composition of these two functions is the mirror function. □

Example 4. *The function $h(\flat^n u_1 \cdots u_n) = u_n \cdots u_1 \flat^n$ is computable in strict real-time.*

Proof. See Figure 4. Knowing the middle of the word at time 1 allows to set up a signal that serves as a reflector. □
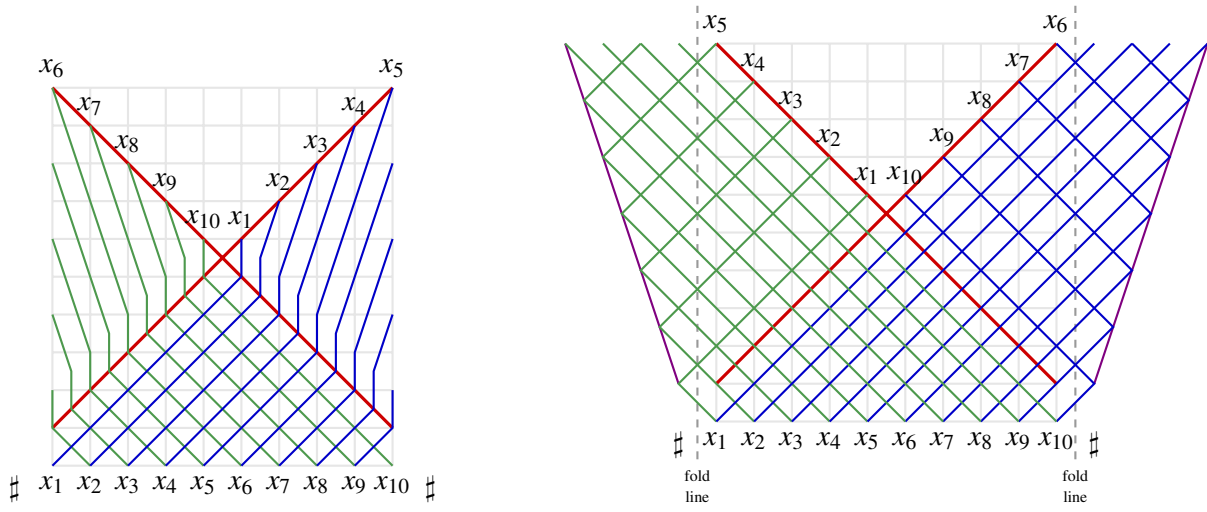
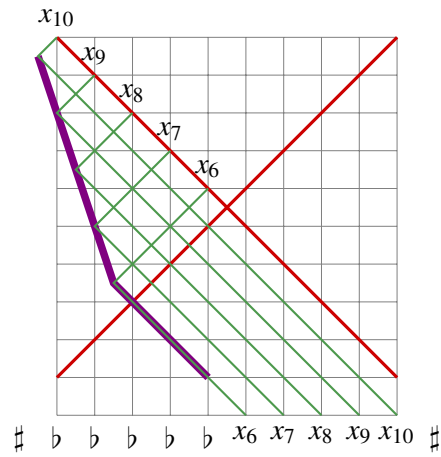Figure 3: Computing f and g in strict real time

Figure 4: Computing h in strict real time

In a recent result, T. Worsch and H. Nishio have proved that sorting binary numbers of the same size is computable in synchronous real time [10]. The algorithm is based on an odd-even sort and uses some clever adaptations to achieve synchronous real time. Here, we shall present a new algorithm to sort in linear time. Its interest lies in the "generic" method used which applies to several different problems. The basic idea is to build a *assembly line*: the input will be traveling along some path where *agents* will act on it. We shall give three significant examples taking advantage of this method: sorting a sequence, reordering a cycle in a graph, and marking the connected components of an undirected graph.

Lemma 5. *Sorting a sequence can be done in linear time.*

Algorithm. To do this, we use two layers of states. The lower layer (in black) will serve to transmit information whereas the upper layer (in blue) will stay still and serve as an agent.

The basic scheme is the following: the lower level travels to the left. If it is greater than the agent (in the upper level), they swap place. If the upper level is empty, the lower information becomes the agent (see Figure 5).

Once the end of the word reaches an agent, it indicates the end of computation for this agent. The algorithm is ended by shifting the result (in green) back in place (not depicted in the figure).

Figure 5: Scheme of sorting algorithm

Here, the behavior is depicted using integers to underline the scheme. Practically, those numbers are supposed to be encoded in binary with a fixed number of cells and require thus some fixed size. Every elementary transition of the scheme can be done in linear time with respect to the size of integers. Since the number of steps of the scheme is proportional to the number of integers the resulting algorithm is linear in the size of the input.

$$\square$$

The previous algorithm is not a surprising result and can be probably presented in a different way. However, the generic idea of the method can be adapted to the problem of *edge reordering of path*: given a sequence of edges that form a path in a random order, we want to reconstruct the order of nodes in the path. For example, the input $(6,12)(2,6)(1,11)(12,7)(8,1)(7,8)$ should output $2,6,12,7,8,1,11$. Intuitively, this problem can be seen as a sorting problem in which the order is given by local constraints. Using similar method as previously, the problem can be solved in linear time.

Lemma 6. *Edge reordering of a path can be done in linear time.*

Algorithm. The algorithm also uses two layered states as previously.

In the first step, elements of both layer are triplets representing the beginning, the end and the length of one path (for example, $(3,5)_2$ represent the path of length 2, going from edge 3 to edge 5). Initially, all edges correspond to path of length 1. The basic operation of an agent consists in merging once the two paths present in the cell and storing the relative position of the gluing element with respect to the start of the newly created path. Once all information are gone through, the result consists of a unique element designating the path.

In a second step, the first vertex of the path is send backwards. Each agent waits for its reference and after seeing it, can put the second vertex at the correct position using the length as a counter.

The correctness of the algorithm lies in the following properties: during the first step, any agent is removing reference to exactly one vertex (and all references to this vertex since it is guaranteed to only appear twice in the input). The resulting flow transmitted to its left neighbour is a valid data. Moreover, if the flow returning from the left neighbour is correct, then the missing vertex is added at the correct position. Since the last agent exists and does its job correctly, a recurrence can prove the correctness of the algorithm.

Figure 6: Edge reordering along path

$\square$

At last, we want to present a third use of the generic method. In this last case, the problem is: given a sequence of edges of an undirected graph, can we output the same sequence where every edge is marked by a unique identifier per connected component.

Remark 7. *Connected components marking in undirected graphs is computable in linear time.*

Algorithm. This algorithm (depicted in Figure 7) is a variation of the previous one and associates to each vertex the label of the smallest vertex in its connected component. In a first step, the agent (given a edge) only replaces any occurrence of the greater vertex by the smaller one. During the second step, when seeing the result for the smallest one, it duplicate it for the greater vertex. The proof of correctness work

as in the previous case, each agent "suppress" one vertex and send valid data to its left neighbour in the first step. In the second step, it "add" this vertex correctly.

					7_1	5_1	6_2	4_1	
				7_1	5_1	6_2	4_1		
				7_1 5_1	6_2	4_1			
				7_1 5_1 6_2	4_1				
			5_1 5_1 6_2	6_2 $(1,7)$ 4_1	4_1				
			$(1,5)$	$(1,7)$					
		6_2 $(2,6)$ 4_1	4_1 $(1,5)$	$(1,7)$					
		$(2,6)$	$(1,5)$	$(1,7)$					
	4_1 $(1,4)$	$(2,6)$ $\#$	$(1,5)$	$(1,7)$					
		$(2,6)$ $(1,4)$	$(1,5)$ $\#$	$(1,7)$					
		$(2,6)$	$(4,5)$ $(1,4)$	$(1,7)$ $\#$					
		$(4,5)$ $(2,6)$	$(4,7)$ $(4,1)$	$\#$					
		$(4,5)$	$(4,7)$ $(2,6)$	$(4,1)$	$\#$				
			$(5,7)$ $(4,5)$	$(2,6)$	$(4,1)$	$\#$			
			$(5,7)$	$(4,5)$	$(2,6)$	$(4,1)$	$\#$		
		$\#$	$(5,7)$	$(4,5)$	$(2,6)$	$(4,1)$	$\#$		

Figure 7: Marking connected component

$\square$

One can note that since sorting cannot be done in linear time for most sequential computation models (such as Turing machines), most of the previous examples are known not to belong to any linear class for sequential computational models.

3 Constructive properties

In this section, we study some stability properties of our functional classes. We also present a speed-up result for the linear class and show that this class is robust.

First let us observe that, by definition of our complexity classes and their ability to explicitly mark the output sites, we can easily derive the following chain of inclusions.

Remark 8. *Strict real time is included in synchronized real time which is included in linear time.*

One usual question is how the functional classes behave with respect to operations on functions. A first result is that since we can construct the Cartesian product of cellular automata, those classes are stable by Cartesian product.

Remark 9. *If $f : I^* \to O^*$ and $g : I^* \to O'^*$ are computable in strict real time (resp. synchronous real time, linear time) so is the Cartesian product: $(f,g) : I^* \to O^* \times O'^*$ defined by $(f,g)(w) = (f(w),g(w))$.*

Another natural operation is the composition of two functions. In this case, since the sum of two linear functions is linear, it is not difficult to see that linear time is closed under composition.

Remark 10. *The set of functions computed in linear time is closed under composition.*

Now, let us go to some more technical stuff and look at the possibility of speed-up. The speed-up algorithm presented below will also serve to prove that strict and synchronous linear time are equivalent.

Proposition 11 (Linear acceleration). *If a function f is computable in time $n+t(n)$ then, for any positive ratio $r \in \mathbb{Q}^+$, f is also computable in time $n + \lceil rt(n) \rceil$.*

Proof. Let $\mathscr{A}$ be a CA which computes f in time $n+t(n)$. One will construct a CA $\mathscr{B}$ which computes f in time $n+rt(n)$. The CA $\mathscr{B}$ will simulate the behavior of $\mathscr{A}$ on any input w in achieving a geometric transformation of the space-time diagram of $\mathscr{A}$. It will be done in two steps. First the space-time diagram will be compressed in order to speed up the computation. Then a decompression will be achieved to retrieve the output. In both steps, we will freely make use of rational coordinates. To revert then to a discrete space-time diagram is the classic trick which consists in grouping and keeping some redundant information in each integer site.

First step To speed up the computation implies to scale the time axis by some ratio r and, due to neighborhood constraints, to scale likewise the space axis. However, a transformation scaling both in time and space axis is not feasible directly from the initial configuration. Here it will be achieved by the composition of two intermediate and symmetric directional scalings which fit the specificities of the device.

The first transformation takes advantage of the cell 0 which receives a border state from its left at step 1 and so knows that it is the leftmost active cell and has no information to expect from its left. Concretely, the transformation starts from the cell 0 at time 1 and scales the space-time area above the line $t = c+1$ by the ratio r in the diagonal direction (up-left to down-right). Figure 8 depicts the rational representation of the space-time diagram resulting from this transformation.
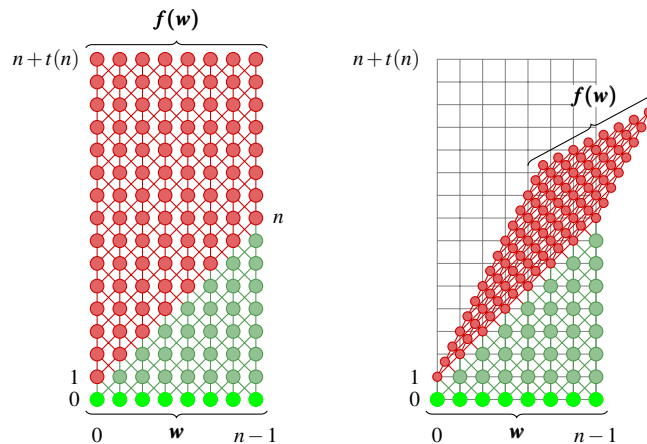

Figure 8: Scaling in the diagonal direction by a ratio $r = 1/2$

The origin of the transformation is the site $\langle 0,1 \rangle$ and its matrix is $\begin{pmatrix} (1+r)/2 & (1-r)/2 \\ (1-r)/2 & (1+r)/2 \end{pmatrix}$. It applies to all sites in the area $\{\langle c,t \rangle : 0 \le c < t\}$, i.e., the set of sites impacted by $\langle 0,1 \rangle$ with respect to the neighborhood. Observe that the transformation is workable. First, inside the scaled part, the initial

communication links $(-1,1)$, $(0,1)$ and $(1,1)$ are mapped into the links $(-r,r)$, $((1-r)/2,(1+r)/2)$ and $(1,1)$ which satisfy the dependency constraints. Second the initial communication links entering by the down right side of the scaled part are also mapped into links satisfying the dependency constraints. Third, no information is coming from its left side and thus leads to no constraint.

In a symmetric way, the second transformation starts from the right end of the input word at time 1 and scales the space-time area above the line $t+c=n$ (with n the size of the input w) by the ratio r in the anti-diagonal direction (up-right to down-left). The transformation $\begin{pmatrix} (1+r)/2 & -(1-r)/2 \\ -(1-r)/2 & (1+r)/2 \end{pmatrix}$ with as origin $\langle n-1, 1 \rangle$ applies to all sites in the area $\{\langle c,t \rangle : 0 \le n-1-c < t\}$.

Now, as illustrated in Figure 9, the composition of these two symmetric scalings sharing the same ratio r, results in an uniform scaling with origin $\langle (n-1)/2, (n+1)/2 \rangle$ and ratio r:

$$\begin{pmatrix} (1+r)/2 & (1-r)/2 \\ (1-r)/2 & (1+r)/2 \end{pmatrix} \begin{pmatrix} (1+r)/2 & -(1-r)/2 \\ -(1-r)/2 & (1+r)/2 \end{pmatrix} = \begin{pmatrix} r & 0 \\ 0 & r \end{pmatrix}.$$

It applies to the sites $\{\langle c,t \rangle : 0 \le c < n \text{ and } |c-(n-1)/2| < t-(n-1)/2\}$. The uniform scaling arising from two feasible transformations is workable as well while respecting the neighborhood constraints. Then a simple calculation can confirm that the initial space-time diagram with the output written on cells 0 to $n-1$ at time $n+t(n)$ is scaled into a diagram where the output is written on the segment of cells $[(1-r)(n-1)/2, (1+r)(n-1)/2]$ at time $1+(1+r)(n-1)/2+rt(n)$.
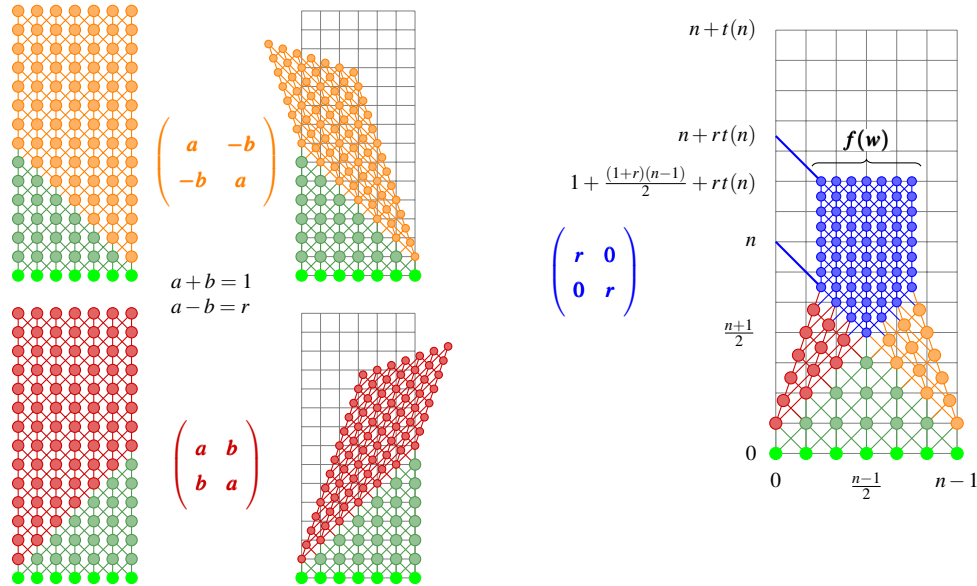

Figure 9: Uniform scaling as composition of two symmetric scalings with a ratio $r = 1/2$

Second step Finally, the compressed output has to be decompressed. The geometric construction, depicted in Figure 10, makes use of two signals initiated from each extremity of the compressed output
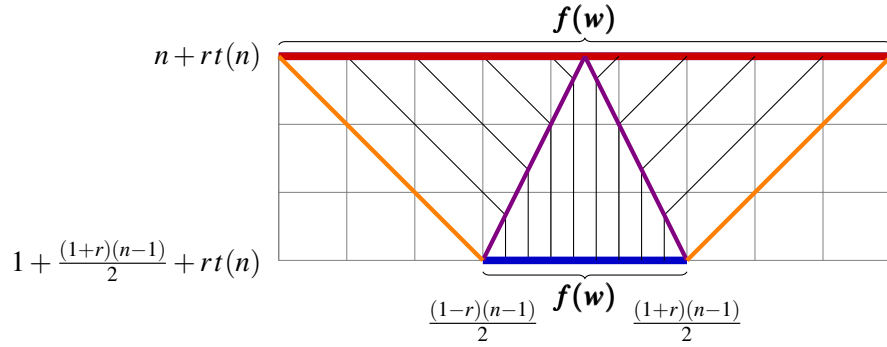
Figure 10: Decompression step (with as ratio $r = 1/3$)

and run respectively to the left and to the right with a slope $1/r - 1$. The decompression is halted by a modified firing squad which marks times $n + rt(n)$.

$\square$

In fact, when looking more in detail at this proof, it can be seen that the result applies not only in (synchronous) linear time but also in the linear extension of strict real time.

Proposition 12. *The linear extension of the strict real time is equivalent to linear time.*

To get a comprehensive picture of functional classes, one main question is to determine the power of synchronous real time, notably in comparison with synchronous linear time. Unfortunately we fail to give a concrete answer. This is not surprising, we face the same difficulty to separate real time and linear time in the case of recognition CA and even though the equality seems unlikely. The next proposition makes explicit the links between the functional and recognition contexts regarding to real time ability.

Proposition 13. *The following assertions are equivalent:*

1. *The composition of any two strict real time functions is computable in synchronous real time.*

2. *The synchronous real time functional class is closed under composition.*

3. *The synchronous real time and synchronous linear time functional classes are equivalent.*

4. *The real time and linear time recognition classes are equivalent.*

5. *The real time recognition class is closed under reversal.*

Proof. The equivalence $(4) \Leftrightarrow (5)$ is far to be trivial: it is a meaningful result established by O. H. Ibarra and T. Jiang [2]. The implication $(3) \Rightarrow (2)$ is straightforward as linear time functional class is closed under composition. The implication $(2) \Rightarrow (1)$ is immediate. The implications $(1) \Rightarrow (5)$ and $(4) \Rightarrow (3)$ are presented below in Lemma 14 and Lemma 15. $\square$

Lemma 14. *If the composition of any two strict real time functions is computable in synchronous real time then the real time recognition class is closed under reversal.*

Proof. Let L be a language recognized in real time and its reverse denoted by L^R. According to Lemma 1, a CA $\mathscr{A}$ computes in strict real time the function f defined by $f(w) = 10^{|w|-1}$ if $w \in L$ and $f(w) = 0^{|w|}$ otherwise. And its mirror CA computes in strict real time the function g defined by $g(w) = 0^{|w|-1}1$ if $w \in L^R$ and $g(w) = 0^{|w|}$ otherwise. Independently the function h defined by $h(x_1, \cdots, x_n) = x_n 0^{n-1}$ is also computable in strict real time. The composition of h and g corresponds to the function $i = h \circ g$ defined by $i(w) = 10^{|w|-1}$ if $w \in L^R$ and $i(w) = 0^{|w|}$ otherwise. If the hypothesis holds then i is computable in synchronous real time and it follows that L^R is recognized in real time. $\square$

Lemma 15. *If the real time and linear time recognition classes are equivalent then the synchronous real time and synchronous linear time functional classes are equivalent.*

Proof. The proof proceeds in two steps. First, making use of the hypothesis, we will show that if a function f is computable in linear time then there exists a CA which for any input w yields every i-th output symbol $f(w)_i$ on the site $\langle 0, |w| + i \rangle$. Second, a compression will allow to get the first half of the output at synchronous real time. As regards the second half of the output, these two steps will be applied in a symmetric way.

First step   Let f be a function computable in linear time. So there exists a CA $\mathscr{A}$ which on every input w outputs $f(w)$ at time $2|w|$ (See Figure 11(a)). In addition, the CA $\mathscr{A}$ can be completed in order to get the output $f(w)$ on the leftmost cell between times $2|w|$ and $4|w|$: from the time $2|w|$, each output symbol has to be sent at speed $1/2$ towards the leftmost cell (See Figure 11(b)). In particular the i-th output symbol $f(w)_i$ reaches the leftmost cell at time $2|w| + 2i - 2$. Now, let us translate what does it mean in term of recognition capacity. We introduce a padding symbol $\natural$ not belonging to the input alphabet I. For any symbol q of the output alphabet, consider the language $L_q = \{w\natural^i : f(w)_i = q\}$. Then the CA $\mathscr{A}$ provides the basic ingredients to recognize the language L_q and that in linear time (See Figure 11(c)).
Next, the assumption that real time and linear time are equivalent in the recognition framework, allows one to deduce the existence of a CA $\mathscr{A}_q$ which recognizes the language L_q in real time. Then consider the CA $\mathscr{B}$ resulting from the cross product of all the CA $\mathscr{A}_q$ where q ranges over the output alphabet. Such a CA $\mathscr{B}$ has everything it needs to produce, on input $w\natural^i$ the i-th output symbol $f(w)_i$ at time $|w| + i - 1$. See Figure 12. Moreover observe that the running of $\mathscr{B}$ on the input word $w\natural^{|w|}$ contains all the real-time evolutions on the prefixes of $w\natural^{|w|}$. That means $\mathscr{B}$ is able to produce the whole output $f(w)$ on the cell 0 between times $|w|$ and $2|w|$. Lastly, in order to $\mathscr{B}$ works in the same way on the input w clear of padding symbol, just fold the space-time diagram along the vertical line $c = |w| - 1/2$.

Second step   Now we will retrieve the first half of the output in synchronous real time. For this purpose, we will apply the compression presented in Proposition 11 on the CA $\mathscr{B}$, and thereby move the segment $[\langle 0, |w| \rangle, \cdots, \langle 0, |w| + (|w| - 1)/2 \rangle]$ which yields the first half of the output to the segment $[\langle 0, |w| \rangle, \cdots, \langle (|w| - 1)/2, |w| \rangle]$. Precisely, making use of the transformation $\begin{pmatrix} 2/3 & 1/3 \\ 1/3 & 2/3 \end{pmatrix}$ with origin $\langle 0, 1 \rangle$, each of the site $\langle 0, |w| + i \rangle$ carrying the i-th output symbol is mapped to the site $\langle (|w| - 1 + i)/3, 1 + 2(|w| - 1 + i)/3 \rangle$. See Figure 13(b). Next, as depicted in Figure 13(c), the data are transmitted at maximal speed to the left. Thus, for all indices i into the first half, i.e. $i \leq (|w| - 1)/2$, the i-th output symbol reaches after $(|w| - 1 - 2i)/3$ steps the site $\langle i, |w| \rangle$.
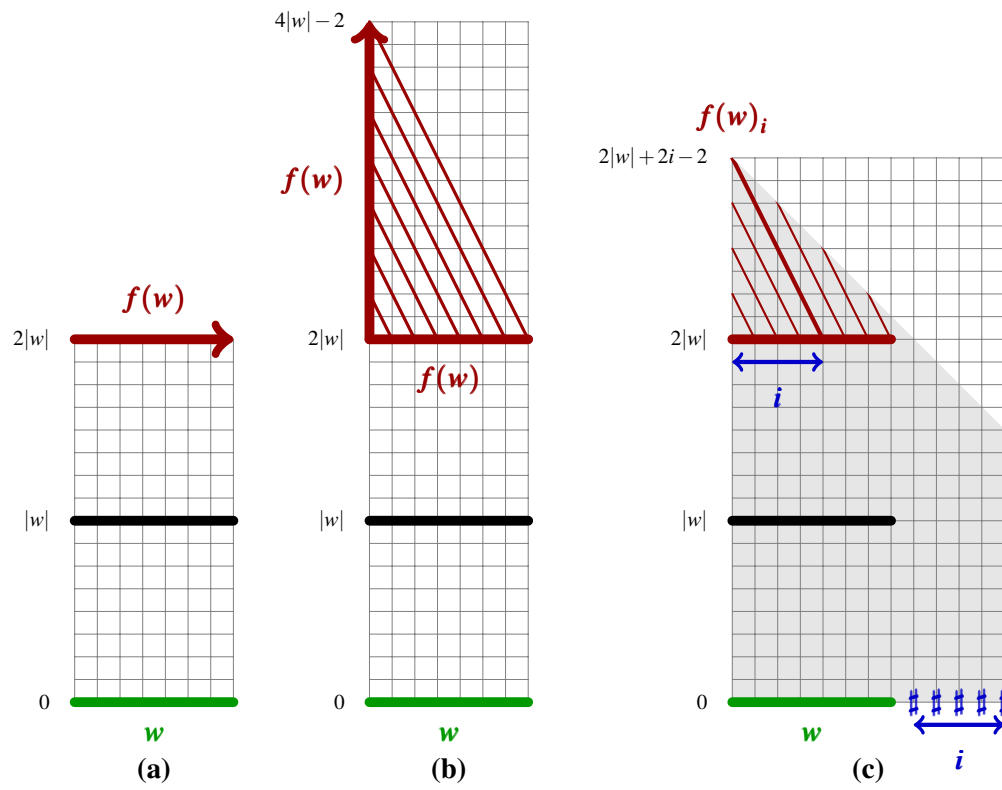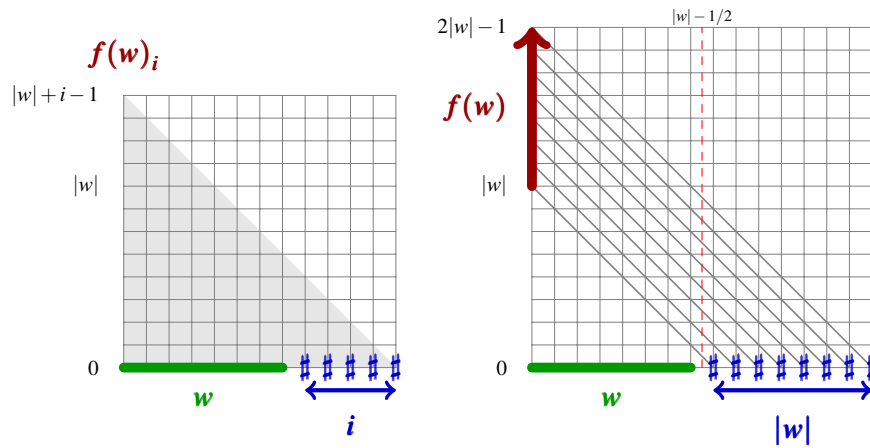
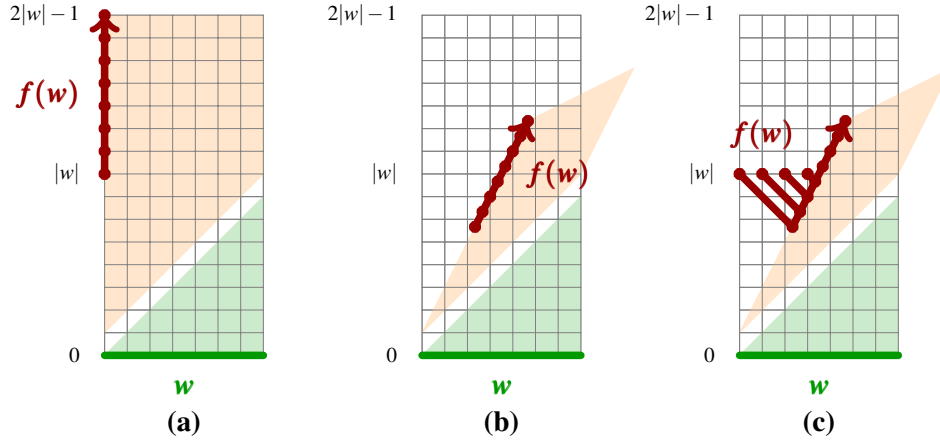Figure 11: The CA $\mathscr{A}$

Figure 12: The CA $\mathscr{B}$

Figure 13: Transfer of the first half of the output

Finally, the two stages are achieved in a symmetric way (i.e., making use of a vertical symmetry axis) to reconstitute the second half of the output.

□

4   Limitations

Achieving negative results is often a more difficult task. Here, we shall present a result which makes explicit the impossibility of some information transfer in cellular automata. To do this we shall prove a lack of certain patterns in the space-time diagram.

To be more precise, if we look at a triangular extract from a space-time diagram, we know that states inside the *triangle* of size n are uniquely determined by its *base* $b = b_1 b_2 \ldots b_n$. The *height* of the triangle is of $h = \lceil n/2 \rceil$.

Now, let us take any arbitrary partition of the states set into two subsets $\pi : S \to \{0,1\}$.

Definition 5. A triangular extract of a space-time diagram is said *uniform* (with respect to a partition π) if for any row of the triangle, all states in the row are in the same part.

We can thus associate to any uniform triangle of size n a *characteristic word* $c = c_1 c_2 \ldots c_{\lceil n/2 \rceil} \in \{0,1\}^{\lceil n/2 \rceil}$ corresponding to the sequence of the parts attached to each row (see Figure 14).

Theorem 16. *For any partition π, there exists a characteristic word $c \in \{0,1\}^*$ such that no uniform triangle with this characteristic exists.*

Proof. Let us pose the sequence v_n defined by :

$$v_1 = \frac{|S|^4}{4} \text{ and } v_{n+1} = \frac{v_n^2}{2^{2^{n-1}}}$$

To show the theorem, let us prove that for uniform triangle of size 2^{n+1}, there exists a characteristic word c_n of size 2^n generated by at most v_n bases by recurrence over n. As v_n converges towards 0, this leads to the desired result.

For $n = 1$, a uniform triangle of height 2 has a base of size 4. There are $|S|^4$ different bases, $2^2 = 4$ different characteristics. There exists a characteristic word $c_0 \in \{0,1\}^2$ which has at most $|S|^4/4$ corresponding bases.

Let us take a uniform triangle of size 2^{n+1} and divide it into four smallest ones as depicted in Figure 14. By definition, A and B correspond to uniform triangles of size 2^n sharing the same characteristic. By recurrence, there exists a characteristic word c_n such that the number of different bases b_A and b_B are both less than v_n. Since $b_A b_B$ is the base of our triangle, we can deduce that any uniform triangle of size 2^{n+1} whose characteristic begins with c_n must have its base among the v_n^2 elements on the form $b_A b_B$. Since the number of characteristic words of size 2^n with prefix c_n (of length 2^{n-1}) is $2^{2^{n-1}}$, the average number of base per words is v_{n+1} and at least one word has thus less or equal bases than v_{n+1}
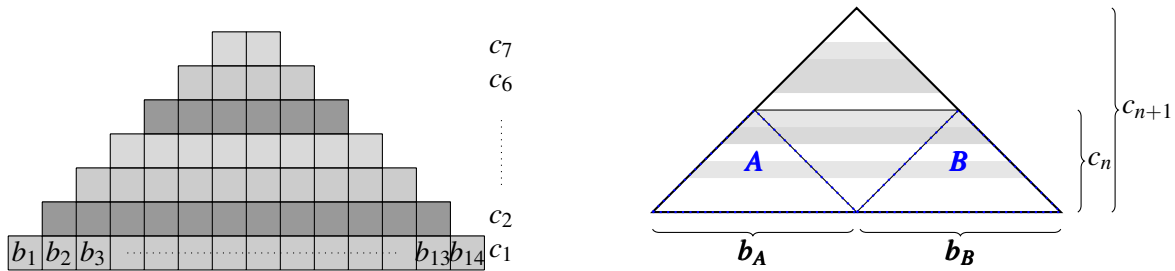

Figure 14: Induction over uniform triangle

Using this general result, we can prove that the mirror is not computable in strict real time.

Proposition 17. *Mirror is not computable is strict real time.*

Proof. By contradiction, assume that there exists a CA computing such a solution. Such CA works over each prefix of the input as depicted in the first row of Figure 15 and output the result at the specified position.

Since the # symbol cannot influence any cell under the diagonal, the computation in the lower left triangle is the same for all cases. Therefore, if we use the classical trick of adding to each cell an additional layer which makes computation as if the input was finished, the resulting cellular automaton can do the partial computation indicated on the lower row of the figure.

Taking the projection π used for outputting the result, we can see that we could construct uniform triangles with any arbitrary characteristics contradicting the previous theorem.

This last result allows us to state a separation result between strict and synchronized real time. Note that there exist other ways to achieve such a result (using the separation of real time recognition with central output or left output for example).

Corollary 18. *Strict real time is strictly included in synchronous real time.*

As mirror can be achieved by composition of the two functions f and g given in Example 3 which both are computable in strict real time, it provides an explicit counter-example for the stability of this class under composition.

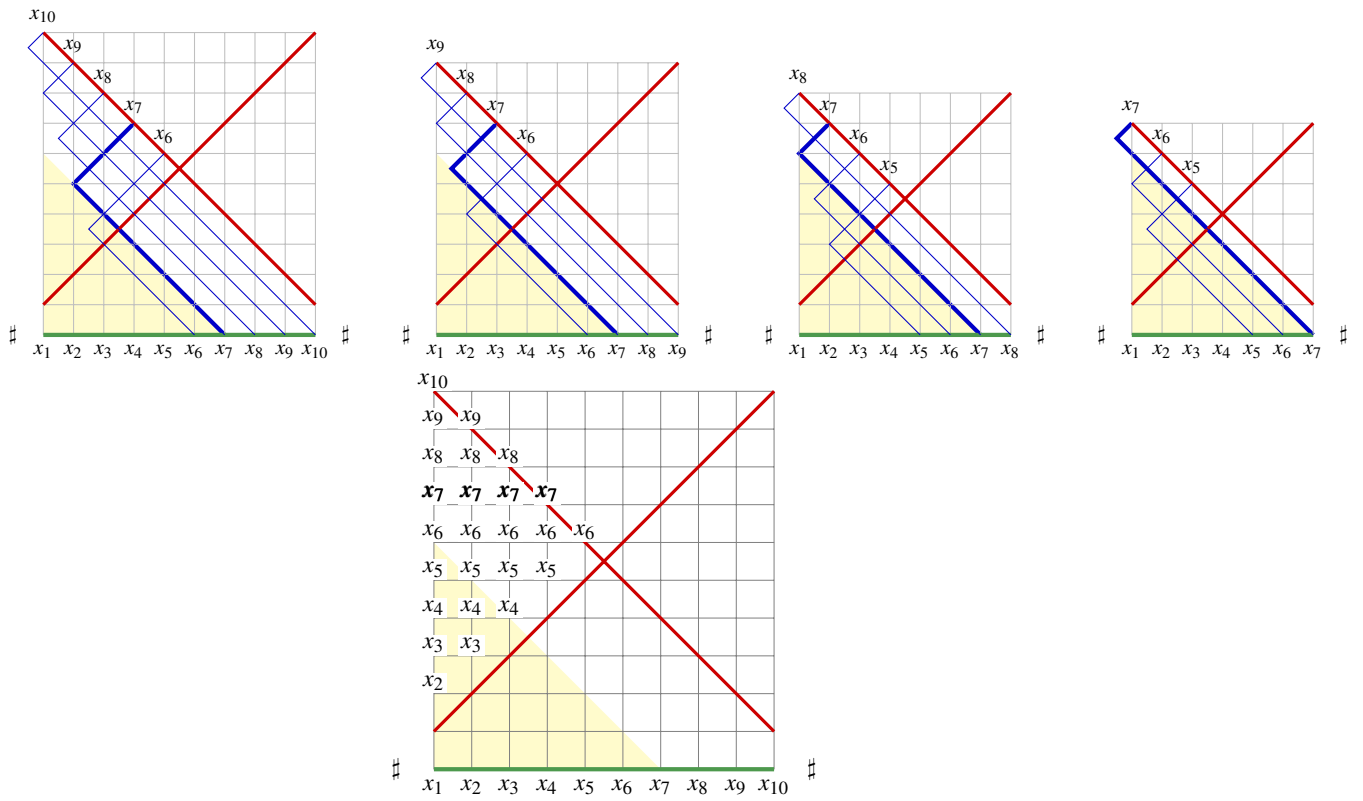**Corollary 19.** *Strict real time is not closed under composition.*

Figure 15: Triangles appearing in the space-time diagram of the mirror computation

5   Conclusion

The parallel functional classes provide an interesting perspective to study the question of small time complexities over cellular automata.

The three defined classes: strict real time, synchronous real time and linear time form a chain of inclusions and all contain "natural" problems. Not surprisingly, the functional classes are strongly linked with the recognition classes. In particular open questions about proper inclusions are correlated.

Moreover, the functional approach brings us new solvable problems for which solutions are interesting outside the strict range of functional classes (as the generic method used in several algorithms or the result over restriction on uniform triangles in space-time diagram).

References

[1] Karel Čulík II (1989): *Variations of the firing squad problem and applications. Information Processing Letters* 30(3), pp. 152 – 157.

[2] Oscar H. Ibarra & Tao Jiang (1988): *Relating the Power of Cellular Arrays to Their Closure Properties. Theoretical Computer Science* 57, pp. 225–238.

[3] Jarkko Kari (2005): *Theory of cellular automata: a survey.* Theoretical Computer Science 334(1-3), pp. 3 – 33.

[4] Martin Kutrib (2008): *Cellular Automata - A Computational Point of View.* In: *New Developments in Formal Languages and Applications*, pp. 183–227.

[5] Martin Kutrib & Andreas Malcher (2010): *Transductions Computed by Iterative Arrays.* In Jarkko Kari, editor: *Proceedings of JAC 2010*, TUCS, pp. 156–167.

[6] Jacques Mazoyer (1996): *On Optimal Solutions to the Firing Squad Synchronization Problem. Theoretical Computer Science* 168(2), pp. 367 – 404.

[7] John von Neumann (1966): *Theory of Self-Reproducing Automata.* University of Illinois Press, Champaign, IL, USA.

[8] Alvy R. Smith III (1971): *Cellular Automata Complexity Trade-Offs. Information and Control* 18(5), pp. 466 – 482.

[9] Véronique Terrier (2011): *Language Recognition by Cellular Automata.* In Jarkko Kari, editor: *Handbook of Natural Computing*, 1, Springer, pp. 123 – 158.

[10] Thomas Worsch & Hidenosuke Nishio (2010): *Real-time Sorting of Binary Numbers on One-dimensional CA.* In Jarkko Kari, editor: *Proceedings of JAC 2010*, TUCS, pp. 214–225.