Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 225 (2009) 201–220

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Surprising Areas in the Quest for Small Universal Devices

Maurice Margenstern[1],[2]

LITA, EA 3097 and IUT de Metz
Université Paul Verlaine – Metz
Metz, France

Abstract

In this paper, we study a few points indicated in the talk which we presented at MFCSIT meeting in Cork. Our study concerns three main areas: Turing machines, cellular automata and hyperbolic cellular automata. The common thread is the quest for small universal devices. It leads from properties belonging to the classical domain up to results on super-Turing computations.

Keywords: Turing machines, cellular automata, hyperbolic geometry, universality, super-Turing computations.

1 Introduction

The quest for small universal devices is a source of inspiration for a lot of works in mathematical logic and theoretical computer sciences, mainly that later field.

In this paper, we shall consider three main areas: Turing machines, cellular automata and hyperbolic cellular automata. The connection from Turing machines to cellular automata is rather natural. A way to represent the computations of a Turing machine is to embed its tape in an infinite cellular automaton on the line. As usual, we require that at the initial time all the cells of the cellular automaton, except finitely many of them are in the quiescent state. The connection with hyperbolic geometry is more surprising. However, as the last section will show, it gives a natural framework to gather classical Turing computations with super-Turing possibilities.

[1] The author is especially in deb to Damien Woods for his invitation at MFCSIT'06. He is very much in debt to the University of Cork, especially to the organizers Tony Seda and Damien Woods for the support of his participation.

[2] Email: margens@univ-metz.fr

doi:10.1016/j.entcs.2008.12.075

2 Turing machines

In this section, we shall start from the famous Rogozhin's Pleiads. This is the figure which we obtain by considering squares in the upper right-hand corner of the plane whose coordinates of the centre are natural numbers ℓ and s. Consider that s is the number of states of a Turing machine, halting state not included, and that ℓ is the number of letters in the alphabet of the Turing machine. Then the square represents the set $M(\ell, s)$ of all deterministic Turing machines whose program can be written in a table with s rows and ℓ columns. We put states as ordinates and letters as abscissas. The Pleiads is obtained by giving a special colour to the squares $\ell \times s$ with the smallest as possible s and ℓ so that $M(\ell, s)$ contains a universal machine.

2.1 Universal Turing machines

The Pleiads was created by Yurii Rogozhin in 1982. Yurii improved the Pleiads several times, see [30]. The most recent improvements were performed by Claudio Baiocchi, who constructed a universal Turing machines with two letters and 18 states, a significant improvement,

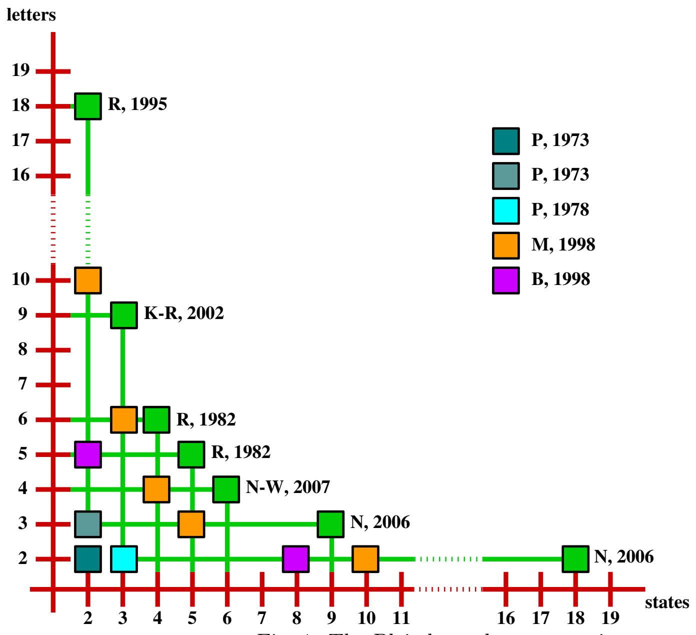


Fig. 1. The Pleiads at the present time.

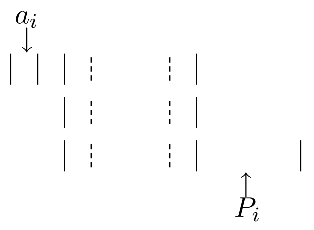
Another universal Turing machine with 3 letters and 9 states and another one with 2 letters and 18 states were recently obtained by Turlough Neary, see [24]. Very recently, the same author with Damien Woods have obtained a universal Turing machine with 4 letters and 6 states, see [25].

We may very sketchily remember the standard technique of these machines. It consists in simulating a 2-tag-system. This notion, which comes from works of

E. Post and is quoted in the present form by M. Minsky, see [20], consists in the following.

A p**-tag system** consists of an alphabet A, a positive number p, and a **mapping** $a_i \mapsto P_i$ mapping from A into A^*. We say that P_i is the **production** associated to a_i. In fact, a p-tag system is a particular case of **canonical system** introduced by E. Post, see [29]. Post himself studied a few tag-systems and M. Minsky popularized the device in [20] as well as its use in the simulation of Turing machines.

A p-tag system defines a computation on words of the alphabet A. One step of the computation consists in taking a_i, the first letter of the submitted word w. Then erase the first p letters of w and denote by w' what remains from w after this operation. Next, append to w' the word P_i, the production associated to a_i. The newly obtained word is $w'P_i$ which is the submitted word for the next step of the computation. The initial word may be any word on A. The computation halts either because the submitted word has less than p letters or because its first letter is a **halting** letter: when a_i is a halting letter, the above described operation applies once again and the computation halts on the word which is obtained. By definition, this word is the result of the computation of the tag-system on the initial word w. Below, Figure 2 illustrates the application of a 2-tag system on a given word.



consider tag-system on {a,b,c}:

a $\longrightarrow$ b
b $\longrightarrow$ bc
c $\longrightarrow$!

application on bbb:

```
bb|b
  |bb|c
  |   |cb|c
  |   |   |c!
```

Fig. 2. On the left-hand side: one step of the computation. On the right-hand side: application on an exampl e.

In what follows, for easily understood technical reasons, we assume that $p = 2$ and we shall only speak of 2-tag systems. When we shall omit the '2', we are dealing with 2-tag systems.

The simulation of the computation of a tag-system by a Turing machine is based on a compact **location technique** in terms of the length of the program of the simulating Turing machine. This idea comes from Watanabe, in the fifties, see [34].

The idea consists in encoding the program of a tag-system as a string in which the productions are associated to the letters of the alphabet by their order in the string. Productions are marked by a P-delimiter and, inside the encoding of a production, the encodings of the letters are also marked by another delimiter, the ℓ-delimiter. The productions are encoded as **mirrors** of the codes of the constituting letters and they are put in the order of the letters starting from a fixed square of the Turing tape and going to the left. Let c_0 be the square to the left of which the encodings of the productions are written on the tape of the Turing machine. Then, to find the rightmost cell c of the encoding of a production, it is enough that the letter is encoded by a number of 1's, for instance, which is the number of P-delimiters and ℓ-delimiters which stand on the tape between c_0 and c. For technical reasons, ℓ- and P- are considered as special words and the number of 1's is the sum of the lengths of all delimiters to cross between c_0 and c: this avoid to waste time, in the program, to check, after each delimiter, whether it is an ℓ- or a P-one.

Let $\mathbf{C}_i$ be the encoding of $\mathbf{P}_i$ for $1 \leq i \leq n$, where n is the number of the production, then

$$\Big| \; \mathbf{C}_n \; \Big| \ldots \Big| \; \mathbf{C}_1 \; \Big|$$

encodes the set of productions. Now, if a_i is encoded by 1^{k_i}, then :

$$k_{i+1} = \Big(\sum_{j \leq i} k_j \Big) + d_i, \quad 1 \leq i \leq n$$

where d_i is the sum of the lengths of the **markers** contained in $\mathbf{C}_i$. Without loss of generality, we may assume that $a_{n+1} \rightarrow !$ is the unique halting production of the system. It is worth noticing that very recently, Damien Woods and Turlough Neary proved that the small universal Turing machines based on 2-tag-systems can perform the simulation in polynomial time, see [35], a result which actually opens new avenues.

2.2 Among the 2×2 *machines*

As can be seen in Figure 1, very little is known about the decidability of the halting problem for Turing machines. The up to now known results are marked with a blue box in the figure. If there is a very readable and rather short proof that 2×2 machines have a decidable halting problem, see [27], the only known proof of the same result for machines with 2 letters and 3 states is very long and sophisticated, see [28]. As for machines with 3 letters and 2 states, the result is claimed in [27] where a proof is announced which was never published.

The study of 2×2 machines also interested other peoples and it is worth to mention another paper, see [11], published after [27], but following a completely different approach motivated by the characterization of the language produced by these machines. In [11], the paper proves that the set of words accepted by the machines with two states and two letters and a halting instruction are regular, except one case only, up to symmetries, in which the set is a deterministic context-free language. The hypothesis of a halting instruction is clear: otherwise the machine trivially never halts.

In [19], I succeeded to find a very particular machine with 4 instructions with a very surprising property: on appropriate words of length n, the time spent by the machine on this word, without going out of it is quasi maximal as this time is $2^n - 2n$.

	x_1	x_2	y_1	y_2
α	$R\,\beta$	$y_2\,L$		
β			$x_2\,R$	$y_1\,L\,\alpha$

Table 1
A 4-letter program for the jewel.

Table 1 gives a program of this machine which we called the **jewel** and which can be presented in a few versions, always with 4 instructions, but depending on the number of letters: 2, 3 or even 4. Here we present the 4-letter version: It is easy to transform this program into a 2×2 machine: identify x_1 and $y1$ with 0 and x_2 and y_2 with 1.

The execution of the jewel on the word $x_1 y_2 (y_2)^\infty$ with the head on x_1 under the state α never stops. The jewel has the intermediate configurations $\alpha x_1 (y_1)^n (y_2)^\infty$ at the time $t_n = 2^{n+2} - 2(n+2)$. Moreover, between t_n and t_{n+1}, the head of the machine never reads the symbol which stands on the right-hand side of the symbol y_2 which appeared at the time t_n.

Below, Figure 3 gives an execution of the jewel on the word $y_1 y_1 y_1 y_2$ with the head on the leftmost y_1 under the state α.

α	y_1	y_1	y_1	y_2	t	α	y_1	y_1	y_2	y_1	$t{+}16$
x_1	β				$t{+}1$	x_1	β				$t{+}17$
	x_2	x_2	x_2	β	$t{+}4$		x_2	x_2	β		$t{+}19$
			α	y_1	$t{+}5$			α	y_1		$t{+}20$
α	y_2	y_2	y_2		$t{+}8$	α	y_2	y_2			$t{+}22$
x_1	β				$t{+}9$	x_1	β				$t{+}23$
α	y_1				$t{+}10$	α	y_1				$t{+}24$
x_1	β				$t{+}11$	x_1	β				$t{+}25$
	x_2	β			$t{+}12$		x_2	β			$t{+}26$
	α	y_1			$t{+}13$		α	y_1			$t{+}27$
α	y_2				$t{+}14$	α	y_2				$t{+}28$
x_1	β				$t{+}15$	x_1	β				$t{+}29$
α	y_1				$t{+}16$	α	y_1	y_1	y_1	y_1	$t{+}30$

Fig. 3. A time table of an execution of the jewel.

It is worth noticing that the behaviour of the jewel is not that of a counter. By the way, the measure which we indicated for the time of execution indicates that it works slightly faster than a counter.

The search goes on for small universal Turing machines. After the paper of Claudio Baiocchi and the improvement obtained by Yurii Rogozhin and Manfred Kudlek, see [12] and also the already quoted new machines obtained by Turlough Neary and Damien Woods, [24,25], another remarkable result was that of

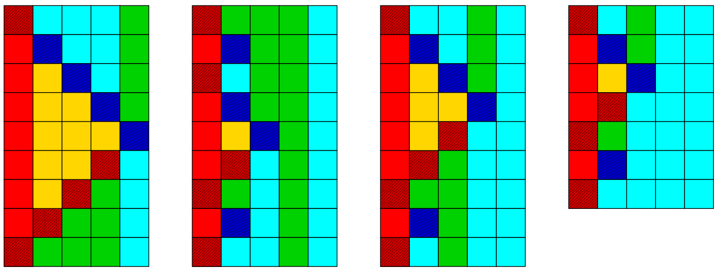


Fig. 4. Colour execution of the jewel: the colours represent both the scanned letters and the state of the head when it scans the letter.

Matthew Cook, [4], on the rule 110, which concerns cellular automata on the line with two states only. This leads us to the quest in the world of cellular automata.

3 Cellular automata

Cellular automata are a popular model of computation which is used far beyond the community of computer science. The activity of cellular automata from the point of view of computer science is so important that there is now a whole journal devoted to this topic. And a special issue of this journal could be devoted only to results on decidability and undecidability in the field of cellular automata.

Here, after a small introduction to remember the reader the main definitions, I just pick up three results.

3.1 Formal definitions

A **cellular automaton** on the line $\mathcal{C}$ consists of set of **cells** indexed by an interval $[a..b]$ of $\mathbb{Z}$, possibly $\mathbb{Z}$ itself, a finite automaton $\mathcal{A}$ whose alphabet is $Q \times Q \times Q$, where Q are the states of $\mathcal{A}$, a fixed in advance state of Q, q_0, called the **quiescent** state and a mapping δ from $Q \times Q \times Q$ into Q, called the **transition function** of $\mathcal{C}$, such that $\delta(q_0, q_0, q_0) = q_0$.

The cellular automaton performs its computation at ticks of a given clock. At each tick, we shall say at time t, all the cells of $\mathcal{C}$ simultaneously update their state according to the transition function, so that at time $t{+}1$ we have, for each $c \in [a..b]$: $q(c(t{+}1)) = \delta(q(c{-}1,t), q(c,t), q(c{+}1,t))$, where $c(t)$ is the content of the cell c at time t.

The neighbourhood of a cell c, c is called the **address** of the cell, consists of the cells with addresses $c{-}1$, c and $c{+}1$ respectively, the left-hand side neighbour, the central cell, the right-hand side neighbour.

The **configuration** at time t is the sequence of $c(t)$'s at the same time t, where the sequence is ordered according to the addresses of the cells.

One step of the **computation** of $\mathcal{C}$ consists in performing the above indicated updating of the states of the cells. A **computation** itself is a sequence of steps starting from an **initial configuration** at the initial time t_0, usually $t_0 = 0$, until the halting of the computation is met which means that there is a first time t_h such that the configuration at time t_h is identical to the configuration at time $t_h{+}1$. The initial configuration is **finite** which means that all cells of $\mathcal{C}$ are in the quiescent state except, possibly, finitely many of them. The configuration at time t where $t_0 \leq t \leq t_h$ is called the **current** configuration at time t and, most often t is omitted. The configuration at t_h is called the **final** configuration.

As this is the case for Turing machines, the computation of a cellular automaton may never halt.

3.2 Cellular automata on the line

If we restrict our attention to cellular automata on the line with two states, call them 0 and 1, it is possible to encode the transition function as follows. The possible configurations of the neighbourhood of a cell are of the form $\epsilon_1\,\epsilon_2\,\epsilon_3$, with $\epsilon_i \in \{0,1\}$ and so, we can interpret this as a number in [0..7]. Accordingly, the transition function appears as a mapping from [0..7] into $\{0,1\}$ and so, there are $2^8 = 256$ such functions. Ordering the configurations of the neighbourhood from 0 to 7, the values of the function can be read as the digit of a number in [0..255]. This is why, in the study of these cellular automata, a cellular automaton is characterized by the rule n where n is the number in [0..255] which encodes its transition function.

Among these cellular automata which behave very differently, one of them, the rule 110, was proved to be weakly universal by Matthew Cook, see [4]. By weakly universal, we mean that the initial configuration of the automaton is infinite with the restriction that it is ultimately periodic. This is a rather simple and natural extension of the notion of finite configuration, but it makes a huge difference. By a straightforward corollary of a result by Codd on cellular automata in the plane, see the next sub-section, cellular automata on the line with two states and starting from a finite configuration have a decidable halting problem. Accordingly, they cannot be universal. Nevertheless, Cook's result is very striking and impressive: it needs the study of large configurations during long periods of time. Cook proved that the rule 110 is able to simulate a special kind of 2-tag-systems called **cyclic** tag-systems: we refer the reader to [4], as even the flavour of the proof cannot be given here.

Coming back to universality problems, it is not difficult to prove that:

For any Turing machine M with p symbols and q states, there is a cellular automaton $\mathcal{C}$ with $p{+}2q$ states which simulates M

Accordingly, not only cellular automata are universal but they have this property with already a rather low number of states: using a universal TM with 4 states and 6 symbols, there is a universal CA on the line with 14 states.

A better result was obtained by : Lindgren and Nordhal in 1990, see [13], with 11 states. In the same paper, the authors lower the result to 9 states under the assumption of an initial ultimately periodic configuration. In 2002, Ollinger improved this result downto 6 states, see [26], using a very tricky simulation of boolean circuits, allowing to also simulate computations starting from an initial infinite configuration.

3.3 Cellular automata in the plane

The definition is an extension of the linear case. The set of cells is of the form $[a..b] \times [a..b] \subseteq \mathbb{Z}^2$. The transition function is a mapping δ from Q^V into Q where Q is the set of states of the finite automaton $\mathcal{A}$ assigned to each cell and V is a **neighbourhood** of (0,0), *i.e.* V is a finite subset of $\mathbb{Z}^2$. There is a quiescent state q_0 and the transition function satisfies $\delta\Big(\prod\limits_{i=1}^{|V|} q_0\Big) = q_0$.

The computation is defined in the same terms as in the case of cellular automata on the line. The difference is on the formulation of the neighbourhood. Here, the condition on the transition function δ writes as:

$$\delta(\prod_{(u,v)\in(c,d)+V} q(u,v,t)),$$

where $q(c,d,t)$ is the state of the cell (c,d) at time t. Note that in this definition, we assume that the neighbours of a cell are a uniform structure whatever the choice of V, fixed once for all for the considered automaton.

Traditionally, two kinds of neighbourhoods are used for planar cellular automata. They are easily described in terms of Cartesian coordinates. For the von Neumann neighbourhood, we have:

$$V = \{(0,0),(-1,0),(0,1),(1,0),(0,-1)\},$$

while, for the Moore neighbourhood we have:

$$V = \{(0,0),(-1,0),(-1,1),(0,1),(1,1),(1,0),(1,-1),(0,-1),(-1,-1)\},$$

see Figure 5 for an illustration.

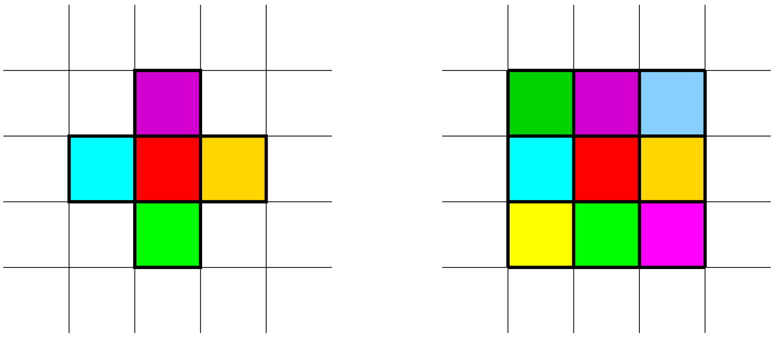


Fig. 5. On the left-hand side: the von Neumann neighbourhood. On the right-hand side, the Moore neighbourhood.

The difference is important: as Codd proved in the already quoted result of [3], the halting of a cellular automaton on the plane with two states and von Neumann neighbourhood is decidable. Accordingly, there is no universal cellular automaton in the plane with two states and in the case of the von Neumann neighbourhood. However, there is such a universal automaton, with the same neighbourhood, starting from 3 states, a result of Serizawa in 1987, see [31]. Now, with the Moore neighbourhood, two states are enough to construct a universal cellular automaton: it is a result of Berlekamp, Conway and Gay who prove the universality of the famous **game of life**, see [2].

Here, we just recall the rules of the game of life:

1 living cell + at most one living neighbour ⇒ death from loneliness,
2 living cell + 2 or 3 living neighbours ⇒ remains alive,
3 living cell + more than 3 living neighbours ⇒ death by overcrowding,
4 dead cell + exactly 3 living neighbours ⇒ birth.

The proof is based on the simulation of a register machine thanks to the implementation of logical gates by a suitable management of collisions between **gliders** which are a moving finite set of cells whose motion and transformation are periodic. It is interesting to notice that such gliders appear spontaneously in simulations starting from a large enough random configuration. Also, the simulation relies on special configurations which produce gliders, they are called the **guns** or which annihilate them, they are called **eaters**. We refer the reader to [2] for more results.

3.4 Reversible cellular automata

The notion of reversible computations introduced by Bennett in the seventies, was motivated by considerations coming from physics. The idea being that processing reversible computations in the reverse way would avoid heat dissipation.

In terms of cellular automata, the reversibility means that there is a way to perform the reverse computation and that this way is provided again by a cellular automaton.

What can be bijective in the computation of a cellular automaton? It is plain that the transition function cannot be bijective as there cannot be a bijection from Q^V into Q as Q is finite set. But we can consider the **global transition function** which is the transformation induced on the set of all possible current configurations of a cellular automaton by the application of one step of computation using the transition function which, in this context, is called the **local rule**. As the set of configurations is $Q^{\mathbb{Z}}$ in the case of the line and $Q^{\mathbb{Z}\times\mathbb{Z}}$ in the case of the plane, and as the global transition function maps this set into itself, the question of the bijectivity makes sense.

Accordingly, if $\mathcal{C}$ is a cellular automaton and if Φ denotes its global transition function, we have the following results:

$\mathcal{C}$ reversible $\Leftrightarrow$ Φ bijective (Hedlund, 1969, [7])

Φ surjective $\Leftrightarrow$ Φ_F injective (Moore (1962), [21] and Myhill (1963), [23])

And so, for cellular automata we have:

$$\text{injectivity} \Leftrightarrow \text{bijectivity} \Leftrightarrow \text{reversibility}$$

It is important to know whether such a property can be recognized by an algorithm. Here we have a sharp difference between the line and the plane:

> *There is a polynomial algorithm to test whether a cellular automaton on the line is reversible or not* (Amoroso-Patt, 1972, [1]).
>
> *The reversibility of cellular automata in the plane is undecidable* (Jarkko Kari, 1990, [10]).

An interesting feature is that several authors tried to obtain as small as possible universal reversible cellular automata. It is impossible to quote all papers in this line. We shall simply quote a work by Imai and Morita, see [9], constructing a reversible universal cellular automaton on the grid of the plane obtained from the equilateral triangle. The pictures obtained by this cellular automaton are really very nice.

We shall just mention two points.

First, most of these reversible cellular automata which are universal are weakly universal: they start from an infinite configuration. Second, they use different kinds of partitioning of the cells of the cellular automata, a notion due to Kenichi Morita, see [22], in order to obtain a bijective **local rule**, which gives an easy sufficient condition for bijectivity. In the square grid, if a cell is cut into four parts, and if the local rule only takes into account the part of the state of a neighbour which is in contact with the cell, then we get a mapping from Q_1^4 into Q_1^4 where Q_1 is the set of the possible values of the parts.

It can be wondered whether such models are necessarily weakly universal?

3.5 The BBM model

In this paper, we prove that this is the case for a very well known such model, the **billiard ball model**, **BBM** model for short, introduced by Toffoli in the early seventies, see [33].

The model takes place in $\mathbb{Z}^2$ where the cells are grouped by 4=2×2. There will be two kind of such **blocks** of 4 cells:

blue blocks for even coordinates

red blocks for odd coordinates

and at even times, the rules of Figure 6 are applied to the blue blocks and, at odd times, they are applied to the red ones.

Below, the figures illustrating some basic patterns also illustrate the way the rules are working. The rules are clearly bijective and also, they are **conservative**: the number of black states is unchanged.

First, we have the basic signal: the motion of a particle, see the left-hand side part of Figure 7. In fact, the figure represents the motion of a **half-particle**. A **particle** consists of two half-particles moving on the same line, separated by a block. On the right-hand side part of Figure 7, we can see how such a particle bounces on the group of half-particles constituting the mirror which remains globally

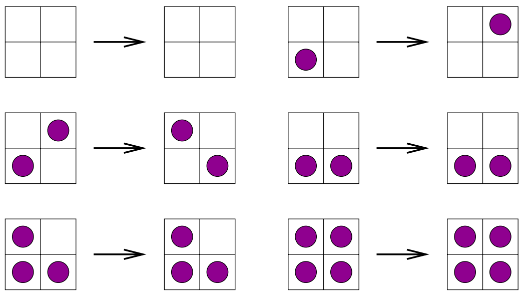


Fig. 6. The rules of the BBM.

unchanged.

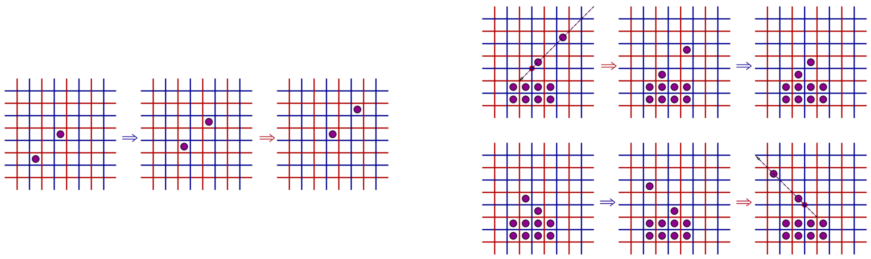


Fig. 7. On the left-hand side: the motion of a half-particle. On the right-hand side, a particle, bouncing on a mirror.

With such mirrors, it is possible to construct the elements of a circuit where a particle simulates the transmission of information in the circuits of a real computer. For reversibility, the logical gates have a special form. Their basic element is a **Fredkin** gate which performs a reversible computation on three bits, x, y and c, returning $cx+\overline{c}y$, $\overline{c}x+y$ and c, where $\overline{a}$ is $\neg a$. We refer the reader to [33], [5] and [22].

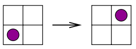
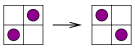
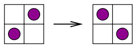
Now, we turn to the result of this subsection:

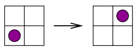
Theorem 3.1 (Margenstern) *The BBM computations starting from finite configurations are decidable.*

Proof. As the starting configuration is finite, it can be inserted in a square C_R centered at $(0,0)$ with sides, parallel to the diagonals, R being the length of the sides.

Consider C_{2R}, also centered at $(0,0)$, defined as C_R, but with a length $2R$ for the sides

After $2^{4R^2}+1$, steps of computation, either at least one cell is on ∂C_{2R} or there is outside a 1 for the first time, or two configurations happen to be identical. In the latter case, the motion is periodic within C_{2R}.

In the former case, only the rule  or the rule  was able to perform this change on ∂C_{2R}. But the rule  could not be applied.

Otherwise, there was previously a 1 on the border, as it is a diagonal. Accordingly, only the rule  was used.

Now, from this time we have a moving half-particle which goes at infinity, out of C_{2R}. From the point of view of the halting problem, we may ignore such particles, and so, we can repeat the argument, this time to C_{4R} in place of C_{2R}, the 'removed' particle being put aside of our considerations.

Now, it is easy to characterize the computation: a certain number N of half-particles goes to infinity and what remains, M of them, periodically evolves within some fixed square. Note that $N+M$ is constant and there are cases when $N = 0$ and others when $M = 0$. □

Thanks to Jérôme Durand-Lose for his kind attention to this result.

4 Hyperbolic cellular automata

As indicated at the beginning of the previous section, cellular automata is a very wide world. It is also an expanding universe. Up to some recent time, when people studied cellular automata on a larger support than a line, it was mainly in Euclidean spaces. Mostly, people looked at what happens in the plane. There are a few studies in $3D$ also due to the capability of cellular automata to simulate $3D$ diffusion-reaction processes, in particular of gases.

In 1999, cellular automata attacked the hyperbolic plane. This was performed by a paper which I wrote with Kenichi Morita, see [18]. However, if an important tool was found in this paper, a good location system was still missing. I found it in 1999, published in 2000, see [14]. This 2000 paper really opened new avenues. Since this time, more than fifty papers appeared in this new field to which I succeeded to attract a few colleagues.

4.1 Hyperbolic geometry

Hyperbolic geometry arose from a 20 century effort to deduce the famous axiom of parallels of Euclidean geometry from the other axioms. This effort resulted in early 19th century by the simultaneous discovery, by Lobachevsky and Bolyai of the hyperbolic geometry where all axioms of the Euclidean geometry hold except the parallel one. In this geometry, a new axiom holds telling that by a point A out of a line ℓ there are exactly two lines which are parallel to ℓ.

There is no room here to give further details about the history of this discovery nor about the hyperbolic geometry itself. Fortunately, models of this geometry inside the Euclidean one were found in the second half of the 19th century. We shall work inside one of them: the **Poincaré's disc model** which is represented in the left-hand side part of Figure 8.

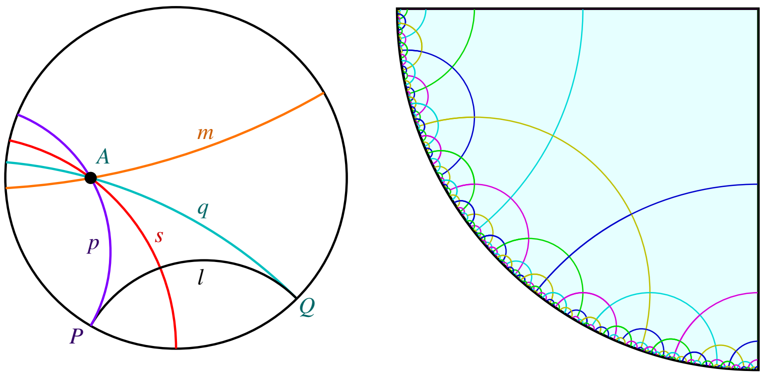


Fig. 8. On the left-hand side: a view on Poincaré's disc model and the illustration in this model of the hyperbolic axiom on parallels: the lines are the trace inside ∂U, the border of the unit disc U of diameters and of circles which are orthogonal to ∂U. In this figure, ℓ and p are parallel as well as ℓ and q: they have a common point at infinity, *i.e.* on ∂U; s cuts ℓ inside U and m is **non-secant** with ℓ: it does not meet it neither in U nor outside. On the right-hand side: the South-West quarter of the pentagrid.

4.2 The pentagrid

The major feature of hyperbolic geometry is that the space is determined by underlying **trees**, see an example in the left-hand side part of Figure 9.

I found a method, see [18], which allows us to do so for the **pentagrid**, *i.e.* the tiling obtained by replicating a regular rectangular pentagon of the hyperbolic plane by reflections in its edges and of the images in their edges. More precisely, the method defines a tree which spans the tiling in a quarter of it by an adequate splitting of the quarter, see [18,14]. Here is a simple figure to indicate how to build the tree which is called **Fibonacci tree** because there are exactly f_{2n+1} nodes of the tree which are at distance n from the root. Note that the branching of the tree is 2 or 3: nodes with 2 sons are called **black** and nodes with 3 sons are called **white**.

There is a very good surprise: if we number the nodes as indicated in the right-hand side of Figure 9 and if we write them in the standard Fibonacci sequence with $f_0 = f_1 = 1$ and the condition that consecutive 1's are ruled out, we find a unique representation for each positive number which we call **standard** and we have the following property:

In the Fibonacci tree, for each node, let n be its number and $\alpha_k..\alpha_0$ be the standard representation of n. Then among the sons of n in the tree, there is a single node with number m such that the standard representation of m is $\alpha_k..\alpha_0 00$. Moreover, a rule defines the position of this node which is called the **preferred son**. *If n is a black node, its preferred son is also black. If n is white, its preferred son is white and it is its middle son.* (Margenstern, [14]).

From this property, we derive an algorithm to find the path from a node to the root which is linear in the length of the standard representation of the number n of the node. We also find the standard representation of the neighbours of the node,

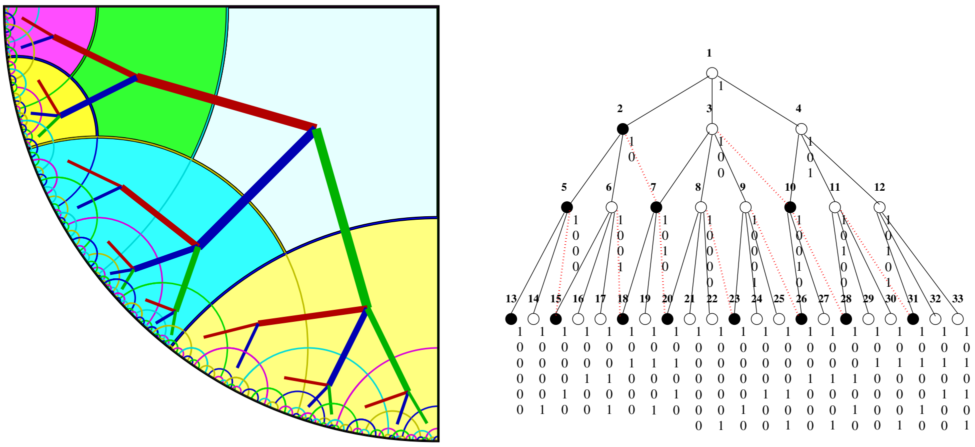


Fig. 9. On the left-hand side: the spanning tree generated by an appropriate splitting. On the right-hand side: the Fibonacci tree with the magic property of the preferred son.

i.e. the nodes which share an edge with n. Later, we shall often denote a node by its number and we shall call **coordinate** its standard representation. We shall often identify the number with the coordinate.

4.3 Universal hyperbolic cellular automata

I have three results in this area. One is a cellular automaton in the hyperbolic plane which is weakly universal. It has 22 states and it was published in [8]. Later, I obtained a cellular automaton in the hyperbolic $3D$ space. It is also weakly universal and it has 5 states, see [16]. This cellular automaton is in some sense a small one. I have also a result on an **intrinsically** universal cellular automaton in the hyperbolic plane, see [17]. This means that the cellular automaton directly simulates a cellular automaton of the hyperbolic plane.

Both solutions in the plane and in the space use the simulation of a register machine by the railway simulation introduced by Ian Stewart in [32]. A single locomotive runs over a railway circuit with tracks and switch points. There are three switch points, see the left-hand side part of Figure 10: the fixed one, the flip-flop and the memory one. The fixed switch point is neutral when the locomotive crosses it in the passive way. In the active way, it always send the locomotive on the same selected tracks. The flip-flop cannot be crossed passively. Each active passage changes the selected track. The memory switch point sends the locomotive on the selected tracks defined by the last passive passage. Thanks to the switches, the tracks can be organized into an **element** which is a read/write unit for a single bit of information given by the positions of the two switches. To read the content of the unit, the locomotive enters it through E. To change the content of the unit, 0 to 1 or 1 to 0, the locomotive enters through U. Then, it is possible to assemble the elements into the units of a register, see Figure 11, then into the control to the access of the register and to the return to the instructions in order to simulate a program.

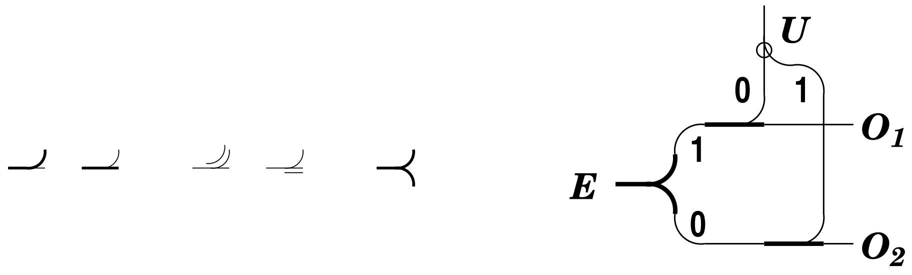


Fig. 10. On the left-hand side: the switch-points: fixed, flip-flop, memory. On the right-hand side: organizing them into a read/write unit.

In the right-hand side of Figure 11, the elements are organized in a few instructions programming the addition of X and Y into Z, with saving the content of X and Y. The figure gives a hint to the organization of any program of a register machine.

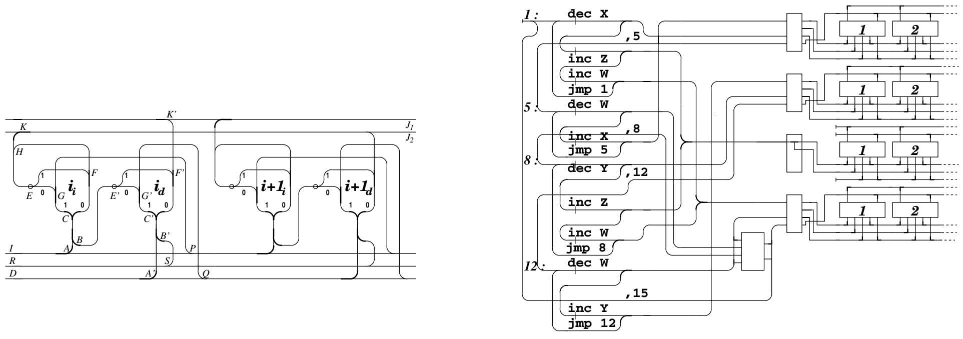


Fig. 11. On the left-hand side: the unit of a register. On the right-hand side: implementation of a small program.

To devise the hyperbolic $3D$ solution, the main important point is to simulate the switches as the tracks and the motion of the locomotive on them is easily dealt with, see [16]. Figure 12 represents the crossing of the switch from the non-selected track.

The locomotive consists of two cells: the front, in green and the rear, in red. The closest neighbours of the cell which implements the switch detect the arrival of the locomotive before it enters the switch. This allows to perform the preliminary changes in order to trigger the change of the selected track once the locomotive leaves the switch.

It is interesting to notice that the rules are devised in a such a way that they do not rely on geometrical criteria to distinguish all the possible configurations of states by the neighbours. In fact, using the $3D$ and the number of connections allowed by

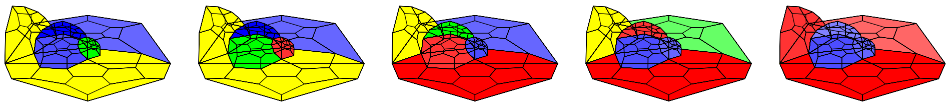


Fig. 12. The passive crossing of a memory switch through the non-selected track: the approach.

the hyperbolic space, we obtain the distinction only by the numbers with which each state occurs in the neighbours for the considered rule. This property was checked by a computer program.

Again, note that this is a weak universality result: the registers we need in this construction are infinite.

4.4 Far beyond the Turing barrier

As announced, in this section we show how hyperbolic geometry gives a new way to obtain super-Turing computations. In most models devised to obtain super-Turing capabilities without using real numbers, people manipulate time in some way. Here, we use the advantage given the geometry for space.

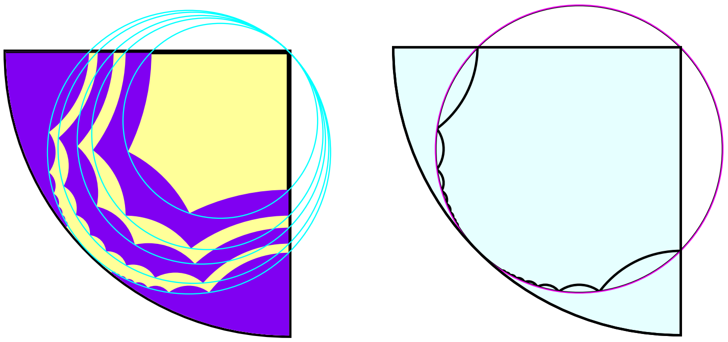


Fig. 13. Construction of an infinigon.

In the hyperbolic plane, there are infinitely many tilings even if we restrict our attention to those which are constructed by reflection from a regular polygon. Among these tilings, there are special ones which are a kind of limit of the others. Figure 13 shows how its right-hand side is obtained from the limit of rectangular regular polygons displayed in a suitable way in the left-hand side. The limit is circumscribed in a **horocycle** whose centre is called the **point at infinity** of the infinigon, see the right-hand side part of Figure 13.

Such objects which are called **infinigons** can be defined for any angle α at a vertex. For any angle, the length of the size of an edge of the infinigon is fixed. Now, it turns out that infinigons tile the hyperbolic plane when the angle is $\frac{2\pi}{k}$ with $k \geq 3$, k integer. What we obtain is called an **infinigrid**. In [15], I give an algorithmic construction of the infinigrids and some other geometric properties of the infinigons. It is very difficult to represent an infinigrid, as shown by Figure 14. A more abstract representation is probably more suggestive, see Figure 15. However, in this latter representation, it is assumed that a hierarchical structure is imposed on the infinigrid which, a priori, is completely uniform. However, it is not difficult, using either Figure 14 or 15, to define the **address** of a cell as a sequence $(a_1, ..., a_k)$ of non-negative integers, a_{2i} denoting the neighbours on the right-hand side of the point at infinity and a_{2i+1} those which are on the left-hand side.

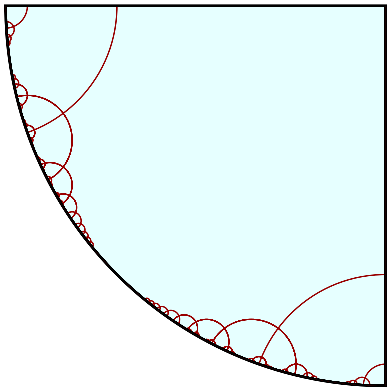


Fig. 14. An illustration of the infinigrid with the angle $\frac{\pi}{2}$.

Now, as indicated in [6], let us fix an infinigrid and consider that each cell is an infinigon. Of course, within the frame of computer science, we cannot accept that a cell has an infinite amount of information at its disposal. On the other hand, if a cell knows only a finite number of its neighbours, we get nothing more than the usual Turing power.

Accordingly, we give the cell just a *glimpse* at infinity. We mean the following. Let ν be a cell in this new setting. Assume that some neighbour μ of ν takes the state α. Then, we shall consider that ν knows that at least one of its neighbours is under the state α. It does not mean that ν knows that μ is under this state. It may even happen that ν does not know who is under the state α. Similarly, if no neighbour of ν is under the state α, then ν also knows this fact. This can be captured by the following notion. We admit that the transition function δ of the cellular automaton works on Q, the set of states and also on $\{0, 1\}^Q$, the set of subsets of Q. Defining $\mathbf{1}_{\mathbf{i}}(s, t)$ as 1 if there is a neighbour of the cell with address s which is under the state $\mathbf{i}$ at time t and 0 otherwise, then the updating of the cells can be formalized as follows:

$$s(t+1) = \delta(s(t) > \mathbf{1}_{\mathbf{1}}(s, t), \ldots, \mathbf{1}_{|Q|}(s, t)).$$

When this is the case, we say that the cellular automaton is **adapted** to the infinigrid.

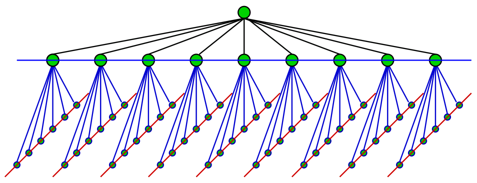


Fig. 15. Another representation of the infinigrids.

From this definition, we obtain the following first result:

There is a cellular automaton U, adapted to the infinigrid, such that, for any arithmetic formula F in Σ^0_n or in Π^0_n, U recognizes whether F is true or not. (Grigorieff-Margenstern, [6])

The proof, see [6], makes use of an infinite initial configuration. By changing a bit the above definition, we obtain a much stronger result where there is no need to an infinite initial configuration. The idea is to make the cells compute the initialization of the previous theorem, using the fact that a cell has infinitely many neighbours. Any of them can use the same information to contribute to the considered initialization.

The changes consist in appending to registers **a** and **x** to each cell. The regsiter **a** is read only and contains the address of the cell. The register **x** is read/write and the cell may perform simple operations on it: to copy the content of **a** and to compute with $+$, $-$, $/$, $*$, mod, sg, sg and $\{(n)_i\}_{i=1}^{|n|}$, these last operations allowing the cell to decode an integer interpreted as the code of a finite sequence of integers. We assume that each operation can be performed in 1 step, whatever the manipulated integers. We also assume that the automaton has two particular states **accept** and **reject**. If the cellular automaton is adapted to the infinigrid, we say that we have a **register cellular automaton**. The initial data is given through the root before the start of the computation. Then, we have:

There is a register cellular automaton U on the infinigrid such that U decides the truth of any Σ_n formula F in a time which is linear in the size of F. (Grigorieff-Margenstern, quoted papers).

References

[1] Amoroso S., Y. Patt, *Decision Procedures for Surjectivity and Injectivity of Parallel Maps for Tessellation Structures*, Journal of Computer and System Sciences, **6**, (1972), 448-464.

[2] Berlekamp E.R., J.H. Conway and R.K. Guy, "Winning Ways for your Mathematical Plays". Academic Press, London, 1982.

[3] Codd E.F., "Cellular automata", Academic Press, 1968.

[4] Cook M., *Universality in Elementary Cellular Automata*, Complex Systems, **15**(1), (2004), 1-40.

[5] J. Durand-Lose, *Representing Reversible Cellular Automata with Reversible Block Cellular Automata*, DM-CCG 2001, (2001), 145-154.

[6] Grigorieff S., M. Margenstern, *Register Cellular Automata in the Hyperbolic Plane*, Fundamenta Informaticae, **61**(1), (2004), 19-27.

[7] Hedlund G., *Endomorphisms and automorphisms of shift dynamical systems*, Math. Systems Theory, **3**, (1969), 320-375.

[8] Herrmann F., M. Margenstern, *A universal cellular automaton in the hyperbolic plane*, Theoretical Computer Science, **296**(2), (2003), 327-364.

[9] Imai, K., K Morita, *A computation-universal two-dimensional 8-state triangular reversible cellular automaton*, Theoretical Computer Science, **231**, (2000), 181-191.

[10] Kari J., *Reversibility of* 2D *cellular automata is undecidable*, Physica D, **45**, (1990), 379-385.

[11] Kudlek M., *Small Deterministic Turing Machines*, Theoretical Computer Science, **168-2**, (1996), 241-255.

[12] Kudlek M., Yu. Rogozhin, *A universal Turing machine with 3 states and 9 symbols*, Lecture Notes in Computer Science, **2295**, (2001), 311-318.

[13] Lindgren K., M.G. Nordahl, Universal computation in simple one-dimensional cellular automata. Complex Systems, **4**, 299–318, (1990).

[14] Margenstern M., *New tools for cellular automata in the hyperbolic plane*, Journal of Universal Computer Science, vol **6**(12), (2000), 1226–1252.

[15] Margenstern M., On the Infinigons of the Hyperbolic Plane, A combinatorial approach, *Fundamenta Informaticae*, **56**, N°3, (2003), 255-272.

[16] Margenstern M., *A universal cellular automaton with five states in the 3D hyperbolic space*, International Journal of Unconventional Computations, to appear.

[17] Margenstern M., *An algorithm for building intrinsically universal cellular automata in hyperbolic spaces*, Proceedings of FCS'06, (2006), 3-9, CSREA Press, ISBN: 1-60132-015-9.

[18] Margenstern M., K. Morita, *A Polynomial Solution for 3-SAT in the Space of Cellular Automata in the Hyperbolic Plane*, Journal of Universal Computations and Systems, **5**, (1999), 563-573.

[19] Margenstern M., L.M. Pavlotskaya, *On the optimal number of instructions for universal Turing machines connected with a finite automaton* International Journal of Algebra and Computation, **13**(2), (2003), 133-202.

[20] Minsky M.L., "Computation: Finite and Infinite Machines". Prentice Hall, Englewood Cliffs, N.J., 1967.

[21] Moore E.F., *Machine Models of Self-reproduction*, Proceedings of the Symposium in Applied Mathematics, **14**, (1962), 17-33.

[22] Morita K., *A simple construction method of a reversible finite automaton out of Fredkin gates, and its related model*, Transaction of the IEICE, **E**, (1990), 978–984.

[23] Myhill J., *The Converse to Moore's Garden-of-Eden Theorem*, Proceedings of the American Mathematical Society, **14**, (1963), 685-686.

[24] Neary T., *Small polynomial time universal Turing machines*, MFCSIT'06, Cork, August, 1-5, 2006.

[25] Neary T., Woods D., *Four Small Universal Turing Machines*, Lecture Notes in Computer Science, **4664**, (2007), Proceedings of **MCU'2007**, 242-254.

[26] Ollinger N., The Quest for Small Universal Cellular Automata, *Lecture Notes in Computer Science*, **ICALP'2002**, **2380**, (2002), 318-329.

[27] Pavlotskaya L.M., *Razreshimost' problemy ostanovki dlja nekotorykh klassov mashin T'juringa.* Matematicheskie Zametki, **13**, (6), (1973), 899-909, (transl. *Solvability of the halting problem for certain classes of Turing machines*, Notes of the Acad. Sci. USSR, 13 (6) Nov.1973, 537-541)

[28] Pavlotskaya L.M., *Dostatochnye uslovija razreshimosti problemy ostanovki dlja mashin T'juring*, Avtomaty i mashiny, (1978), 91-118. (Sufficient conditions for halting problem decidability of Turing machines) (in Russian)

[29] Post E.L., Recursive unsolvability of a problem of Thue. Journal of Symbolic Logic, **12**, 1-11, (1947).

[30] Rogozhin Yu.V., *Small universal Turing machines*, Theoretical Computer Science, **168-2**, (1996), 215-240.

[31] Serizawa T., Three-state Neumann neighbour cellular automata capable of constructing self-reproducing machines. Systems and Computers in Japan, **18**, 4, 33–40, (1987).

[32] Stewart I., *A subway named Turing*, Scientific American, **271#3**, (September 1994), 90-92.

[33] Toffoli T., Computation and Construction Universality of Reversible Cellular Automata, J. Comput. Syst. Sci., **15**, No.2, 213–231, (1977).

[34] Watanabe S., 5-*symbol* 8-*state and* 5-*symbol* 6-*state universal Turing machines*, Journal of the ACM, **8**(4), (1961), 476-483.

[35] Woods D., T. Neary, *On the time complexity of* 2-*tag-systems and small universal Turing machine*, **FOCS'2006**, Berkeley, October, 22-24, 2006.