A Strongly Universal Cellular Automaton on the Heptagrid with Six States

Maurice Margenstern

In this paper, we prove that there is a strongly universal cellular automaton on the heptagrid with six states that is rotation invariant. This improves a previous paper of the author that discusses a cellular automaton with seven states. Here, the structures are slightly simpler and the number of rules is reduced.

Keywords: universal cellular automaton; hyperbolic plane; heptagrid; Poincaré disc

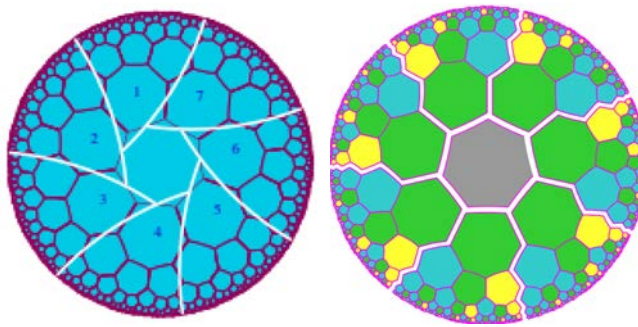
1. Introduction

The author has studied the possibility of constructing universal cellular automata in tilings of the hyperbolic plane in many papers; a few papers are about tilings in the hyperbolic three-dimensional space. Most often, the constructed cellular automaton was weakly universal. By *weakly universal*, we mean that the automaton is able to simulate a universal device starting from an infinite initial configuration. However, the initial configuration should not be arbitrary. It was the case that the automaton was periodic outside a large enough circle; in fact, it was periodic outside such a circle in two different directions, since the simulated device was a two-registered machine. In almost all papers, the considered tiling of the hyperbolic plane was either the pentagrid or the heptagrid, that is, the tessellations $\{5, 4\}$, $\{7, 3\}$, respectively. Both tessellations only live in the hyperbolic plane. In the pentagrid, the basic tile is a regular convex pentagon with right angles. In the heptagrid, it is a regular convex heptagon with the angle $2\pi/3$ between consecutive sides. Figure 1 provides a representation of the heptagrid in the Poincaré disc model of the hyperbolic plane.

We can see seven tiles in Figure 1(a) that are numbered counterclockwise from 1 up to 7; those tiles are the neighbors of a tile that we call the *central tile* for convenience. Indeed, there is no central tile in the heptagrid, as there is no central point in the hyperbolic plane. We can see the disc model as a window over the hyperbolic plane, as if we were flying over that plane in an abstract spacecraft. The center of the circle is the point on which our attention is focused, while the circle itself is our horizon. Accordingly, the central tile is the tile that

https://doi.org/10.25088/ComplexSystems.33.1.1

is central with respect to the area under our consideration. Figure 1(a) also shows seven arcs of a circle. The arcs represent straight lines in the hyperbolic plane. Those lines have the property that they cross midpoints of consecutive sides of tiles in the tiling. If two consecutive such midpoints on a line ℓ belong to the same tile T, it is not the case for the third midpoint on ℓ. That latter point belongs to a neighbor U of T: we say that two tiles of the tiling are *neighbors* if and only if they have a common side. In that tiling, if two tiles share a vertex, they also share a side. That property is entailed by the angle $2/3$. Lines possessing that latter property are called *midpoint lines*. Still in that figure, we can see that two midpoint lines meeting at the midpoint of a side of tile 2 cross midpoints of sides of tile 1. The rays that are thus delimited define a sharp angle S, and we call the set of tiles whose center is inside S the *sector headed by 1*, and tile 1 is called the *head* of that sector. Similarly, we can define sectors i with $i \in \{2 \,..7\}$, each of them having the tile i as its head.



(a) (b)

Figure 1. (a) The heptagrid; (b) the tree structure of the sectors around a central tile indicated by (a).

In Figure 1(b), we can see the central tile and the seven sectors that surround it: they are defined by the midpoint lines shown in Figure 1(a). In each sector, we have three kinds of tiles: green, yellow and blue. If we look from the head of a sector to the tiles close to the horizon, the head of the sector is green and it has three neighbors in the sector: blue, green and yellow, while turning counterclockwise around the head. Blue tiles have two neighbors, which are blue and green in that order, and green and yellow tiles have three neighbors, which are blue, green and yellow in that order. These indications define rules from which we can build a tree that spans the tiles of the sector. More indications on that topic and on the way to locate the tiles can be found in [1, 2].

Now that the global setting is given, we proceed as follows: Section 2 indicates the main lines of the implementation, which is precisely described in Section 2.2. Finally, Section 3 gives us the rules followed by the automaton. We refer the reader to [3] for figures that illustrate the application of the rules. Those figures were established from pieces of figures drawn by a computer program that applied the rules of the automaton to an appropriate window in each of the configurations described in Section 2.2. The computer program also checked that the set of rules is coherent and that rules are pairwise independent with respect to rotation invariance.

That allowed us to prove the following property.

Theorem 1. There is a strongly universal cellular automaton on the heptagrid that is rotation invariant and truly planar and has six states. The automaton makes use of 510 rules.

In [4] the automaton required 1119 rules, and in [5] it required 486 rules with seven states.

2. Main Lines of the Computation

The first paper about a universal cellular automaton in the pentagrid, the tessellation {5, 4} of the hyperbolic plane, was [6]. This cellular automaton was also rotation invariant and at each step of the computation, the set of nonquiescent states had infinitely many cycles: we will say that it is a truly planar cellular automaton. That automaton had 22 states. That result was improved by a cellular automaton with nine states in [7]. Recently, it was improved with five states; see [8]. A bit later, the author proved that in the heptagrid, the tessellation {7, 3} of the hyperbolic plane, there is a weakly universal cellular automaton with three states that is rotation invariant and truly planar [9]. Later, the author improved the result down to two states, but the rules are not rotation invariant; see [10]. In [11], the author constructs three cellular automata that are strongly universal and rotation invariant: one in the pentagrid, one in the heptagrid and one in the tessellation {5, 3, 4} of the hyperbolic three-dimensional space. By strongly universal, we mean that the initial configuration is finite, that is, it lies within a large enough circle.

In the present paper, we borrow the ideas of [3]. For illustrations and detailed analysis, refer to [3]. We will provide such an analysis for a particular configuration in Sections 2.2 and 3.

Our automaton simulates a two-registered machine, which is enough to entail the strong universality, because a two-registered machine can simulate any Turing machine, as proved in [12]. In our simulation, each register is constructed by following a line. At each

time of the computation, the register is a finite segment of the followed line. One element is appended to that segment each time it is incremented by the application of an appropriate instruction of the register machine program. Symmetrically, one element is removed from the segment each time it is decremented, with the exception of the case when the register is empty.

The simulation is based on the railway model devised in [13], revisited by the implementations given in the author's paper [3]. Section 2.1 describes the main structures of the model. In Section 2.2, we indicate the new features used in the present simulation. Those new features, introduced in [4], allowed us to reduce the number of auxiliary structures used to simulate the apparatus needed by the railway model as described in [13].

2.1 The Railway Model

The railway model of [13] lives in the Euclidean plane. It consists of *tracks*, *crossings* and *switches*, and the configuration of all switches at time t defines the configuration of the computation at that time. There are three kinds of switches, illustrated by Figure 2. The changes of the switch configurations are performed by a locomotive that runs over the circuit defined by the tracks and their connections organized by the switches.

A switch gathers three tracks a, b and c at a point. In an active crossing, the locomotive goes from a to either b or c. In a passive crossing, it goes either from b or c to a.

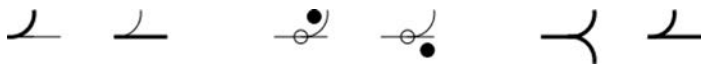


Figure 2. The switches used in the railway circuit of the model. On the left, the fixed switch; in the middle, the flip-flop switch; on the right, the memory switch. In the flip-flop switch, the bullet indicates which track has to be taken.

In the fixed switch, the locomotive always goes from a to the same track: either b or c. In any switch, the track followed by the locomotive in an active crossing is called the *selected* track. The passive crossing of the fixed switch is possible. The flip-flop switch is always crossed actively. The crossing of a locomotive entails the change of its selected track. The memory switch can be crossed actively or passively. Now, the selected track is the track taken by the locomotive in the last passive crossing.

As an example, Figure 3 shows the circuit that stores a one-bit unit of information. The locomotive may enter the circuit either through the gate R or through the gate W.

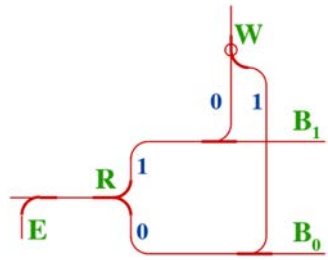


Figure 3. The basic element containing one bit of information.

If the locomotive enters through the gate R where a memory switch sits, it goes either through the track marked with 1 or through the track marked with 0. When it crosses the switch through track 1, 0, it leaves the unit through the gate B_1, B_0, respectively. Note that there are fixed switches on both ways, sending the locomotive to the appropriate gate B_i. If the locomotive enters the unit through the gate W, it is sent to the gate R, either through track 0 or track 1 from W. Accordingly, the locomotive arrives at R, where it crosses the switch passively, leaving the unit through the gate E thanks to a fixed switch leading to that latter gate. When the locomotive takes track 0, 1 from W, the switch after that selects track 1, 0, respectively, and the locomotive arrives at R through track 1, 0 of R. The tracks are numbered according to the value stored in the unit. By definition, the unit is 0, 1 when both tracks from W and from R are 0, 1, respectively. So that, as seen from that study, the entry through R performs a reading of the unit, while the entry through W changes the unit from 0 to 1 or from 1 to 0: the entry through W should be used when it is needed to change the content of the unit and only in that case. The structure works like a memory that can be read or rewritten. It is the reason why it is called the *one-bit memory*.

We will see how to combine one-bit memories in the next subsection, where we introduce several changes to the original setting for the reasons indicated there.

2.2 Tuning the Railway Model

We first look at the implementation of the tracks in Section 2.2.1 and how it is possible to define the crossing of two tracks. In Section 2.2.3, we see how the switches are implemented. In Section 2.2.4, we see how the one-bit memory is implemented in the new context and then, in Section 2.2.5, how we use it in various places.

2.2.1 The Tracks

The tracks play a key role in the computation, as important as instructions and registers: indeed, they convey information without which any computation is impossible.

https://doi.org/10.25088/ComplexSystems.33.1.1

As in [5], the tracks are one-way. But as in [6], we define the tracks as a particular color. However, since the locomotive goes from the program to the registers and from there back to the program, it should be possible for the locomotive to move in opposite directions in some sense. But we keep to the one-way constraint, since we impose a passive crossing for the fixed switch. As in [4], the memory switch is split into two different structures: one of them for an active passage, the other for the passive passage.

Consider four tiles pairwise adjacent, the third one being not in contact with the first one. The motion from the first tile to the third one can be symbolized as follows:

FWWW RFWW WRFW WWRF WWWR

and the reverse motion is given by:

WWWF WWFR WFRW FRWW RWWW

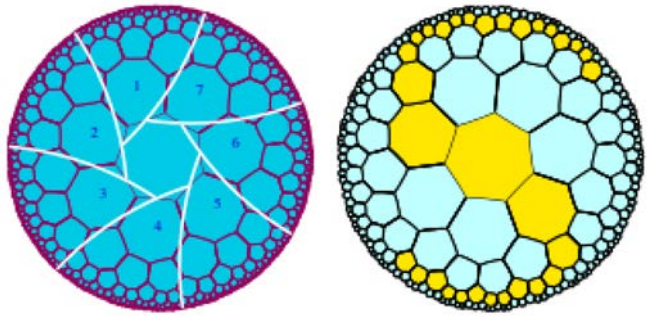
From that observation, we can deduce that a uniform color for the tracks can accept a single direction, although it can accept both of them, since the locomotive consists of two contiguous cells, the front and the rear. The single color allows us a rather free use of the tracks. We choose that they follow segments of lines or arcs of circles. In [3], we display figures illustrating that point.

Consider two tiles U and V. A *path* from U to V is a finite sequence of tiles $\{T_i\}_{i\in[0..n]}$, where $T_0 = U$, $T_n = V$ and, for any i in $[1\,..n-1]$, T_i and T_{i+1} share a common side. In that case, $n-1$ is the *length* of the path. The *distance* from U to V is the smallest length of the paths from U to V. A *circle* around T of radius n is the set of tiles U whose distance to T is n. An arc of circle is a path from a tile U of a circle C to another tile V of C, all of whose tiles also belong to C.

The color of the tracks is fixed as Y. However, we will indicate a track with respect to the color of the locomotive that runs on that track. A B-, G-track is run by a locomotive whose front is B, G, respectively. We also say B-, G-locomotives, respectively. Paths from a part of the circuit to another part consist of pieces of tracks connected together. On most parts of the circuit, the locomotive is B. Later, we will say that B and G are *opposite colors*.

Figure 4(b) represents a track used to test the implementation of the tracks. Figure 4(a) reproduces the sectors illustrated in Figure 1 in order to locate the tiles of the tracks. From top to bottom, those tiles are (7,7) up to (7,12), then (1,5) to (1,12), then (2,2), (2,1), (0,0), (5,1), (5,3), (5,7), (5,6), (5,5), and then (4,12) down to (4,5). The locomotives, both B and G, can run on that track from (7,7) down to (4,5) and also in the reverse order, from (4,5) up to (7,7): it has been tested by a computer program and the reader is referred to [3] for appropriate figures.

We now turn to Section 2.2.2, where we describe the auxiliary structures, the passive fixed switch used to implement the crossings and then the remaining switches, whose implementation is discussed in Section 2.2.2.



(a) (b)

Figure 4. (a) The sectors in order to locate the tiles of the tracks. (b) The tracks used for testing the implementation.

2.2.2 Auxiliary Structures and Crossings

The auxiliary structures we need are the fork, the changer and the filter. The changer and the filter are implemented in two versions: B and G. The B-, G-changer transforms a G-, B-locomotive, respectively, into a B-, G-locomotive, respectively. A filter has a color, either B or G. The filter lets the locomotive pass if it is of the same color and destroys it if it has an opposite color.

Figure 5 illustrates the *idle configuration* of those structures. We define an idle configuration of a structure to be a structure contained in a disc of radius 2 around the central tile, in which disc there is no locomotive.

In the changers, a track crosses the figure through tiles (7,21), (7,8), (7,3), (7,1), 0, (3,1), (3,3), (3,8) and (3,21). In (1,1) we have a B- or a G-tile and in (1,3) and (1,4) we have a Y-tile that fixes the tile in (1,1). In the filters, the track crosses the figure through the tiles (6,20), (6,7), (6,2), (5,1), (4,1), (4,2), (4,5) and (4,13). Tile 0 is B or G. It has R-tiles at (2,1) and (7,1), and there are two Y-tiles at (1,1) and (1,3) whose role is depicted later.

At last, the fork gathers three tracks around a central Y-tile: the arriving track passing through (5,21), (5,8), (5,3) and (5,1), the track leaving to the left through (2,1), (2,4), (2,12) and (2,33) and the track leaving to the right through (7,1), (7,3), (7,8) and (7,21).

A computer program checked the motion of the locomotive through each of those structures and checked that the structures worked as expected.

https://doi.org/10.25088/ComplexSystems.33.1.1

The fixed switch is crossed passively only in the present implementation, following [4].

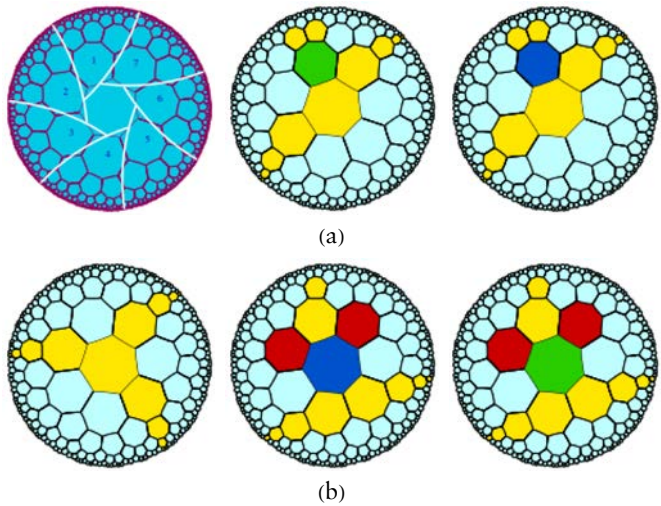


Figure 5. (a) From left to right: the sectors, the changer from B to G, the changer from G to B. (b) From left to right: the fork, the filter for B, the filter for G.

Figure 6 illustrates the idle configuration of a fixed switch.

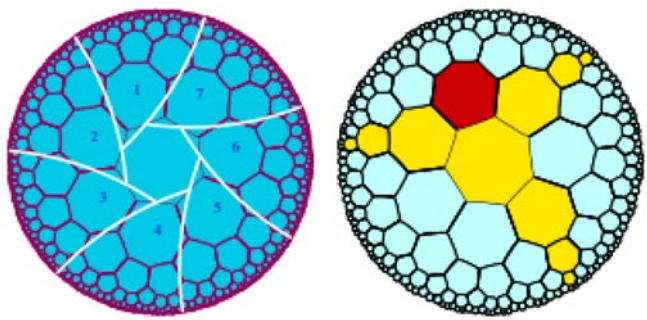


Figure 6.

Note the occurrence of an R-tile at (1,1). It allows the passage of a locomotive along one of its sides while preventing the front of the locomotive from entering the branch that starts from the side where the other branch of the switch arrives.

The fork, the changers, the filters and the fixed switch can be assembled into a structure that performs the working of a crossing. That organization is illustrated by Figure 7.

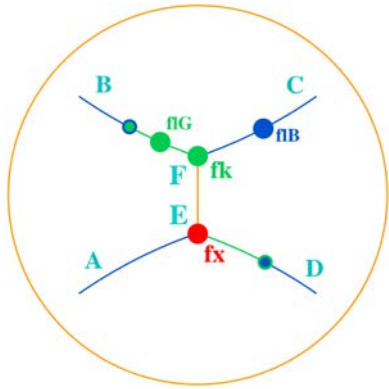


Figure 7. Organization of a crossing.

A locomotive arrives at *E*. If it comes from *A*, *D* it goes to *C*, *B*, respectively. If the locomotive has a given color, it is assumed that it keeps its color along *C* where, after *E*, there is a filter that lets that locomotive pass. If the locomotive arrives from *D*, before reaching *E*, a changer transforms that locomotive into a new locomotive with the opposite color. That new locomotive is supposed to go to *D*, where an appropriate filter is sitting. At *E*, we have a passive fixed switch that allows the locomotive, whatever its color is, to reach the fork sitting at *F*. That fork sends a copy of that locomotive, with the same color, on each branch, to *B* and to *C*. Now, the filter only allows a locomotive to pass that is the same color as the filter. Since the locomotive coming from *D* has changed its color before reaching *E*, the opposite changer allows that locomotive to recover the initial color it had on track *D*. Of course, if the locomotives arriving from *A* and *D* have opposite colors, there is no need of a changer on *D* before *E* and there is no need of a changer on *B* or *C*, provided that the filters have the appropriate opposite colors.

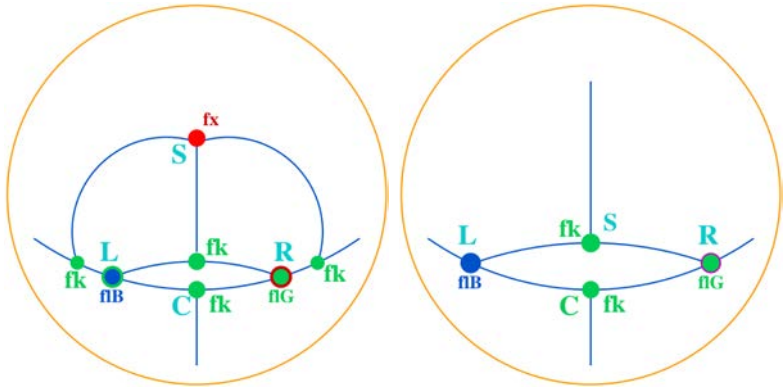
2.2.3 The Switches

This subsubsection deals with the flip-flop and memory switches.

A flip-flop changes the selection after an active crossing, and there is never a passive crossing through a flip-flop switch. In a memory switch, its passive part defines the selection of its active part. Accordingly, we need two kinds of structures: one structure that lets the locomotive go and another that kills it. In the flip-flop, the change is made

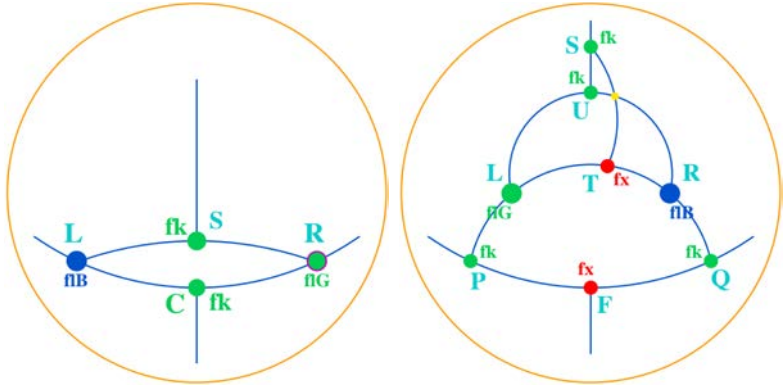
by the crossing of the switch; in the memory switch, the change is triggered by the crossing of the passive switch. Basically, it is the same constraint. Accordingly, the controlling structure in the flip-flop and in the memory switches should be *programmable*, a point that we explain further.

Figure 8 illustrates the implementation of the flip-flop in 8(a), and the active memory switch in 8(b). Figure 9(a) reminds us of the active memory switch, and 9(b) shows us the passive memory switch.



(a) (b)

Figure 8. (a) The flip-flop. (b) The active memory switch.



(a) (b)

Figure 9. Memory switch: (a) the active switch; (b) the passive one.

In the flip-flop, the simple locomotive arrives at *C*. There, a *fork* sends two simple locomotives: one toward *L*, the other toward *R*. The

locomotive that is sent to R meets a filter that is sitting there. If the filter has the color of the locomotive, it goes on. Otherwise, the locomotive is destroyed. Symmetrically, another filter is sitting at L. Now, the filter at R and the filter at L have opposite colors. Accordingly, the locomotive goes further on the side of the switch where the filter has the same color. That property defines the selected track: it is the leaving track on which the filter has the color of the locomotive arriving at C. When the locomotive goes on along the selected track, at some distance after the filter, it meets a fork: one branch of the fork lets the locomotive go on along the selected track; the other sends the locomotive to S. There, a fixed switch is sitting, and any locomotive arriving there passively crosses the switch. From S, the locomotive arrives at a fork that sends a copy of it to the filter at L and another copy to the filter at R. Those copies change the filter to the opposite color, so that now, the selected track of the switch is changed, which conforms with the definition of a flip-flop switch. Note that the change of color in a filter is performed by a B-locomotive, so that a changer is possibly put on the track leaving S and before the arrival to the fork leading to R and to L.

Note that the track from L to R is a segment of line, while the track from the fork placed after the filter to S is an arc of a circle. The change of the filters must occur after the passage of the locomotive through the filter of the selected track. It is enough to define the circle supporting that arc with a radius equal to the distance from C to L, making the angle of the arc at least $\pi/2$: the perimeter of a circle is an exponential of its radius.

The active memory switch is simpler in its working because when the locomotive arrives at C, only the arcs from C to L and from C to R are concerned. We may assume that a passive crossing occurs when the locomotive is not involved in the active switch.

The structure of the passive switch is very different. The simple locomotive arrives either at P or at Q. Assume that it arrives at P. The case of an arrival at Q is symmetrically dealt with. A fork at P sends a locomotive to F, a fixed switch that lets the locomotive leave the switch. The other locomotive is sent to L. There, if the filter in the active part lets the locomotive go, the filter at L stops the locomotive, so that the memory switch does not change the selection. If the filter of the active part stops the locomotive, the filter of the passive switch lets the locomotive go. That locomotive goes further to S via T. There, at S, a fixed switch sends it to U, where a fork sends a copy of the locomotive to the fork S of the active switch and according to the scheme of the active switch, that action will exchange the role of its filters. The second locomotive sent by the fork at S goes to U, where another fork duplicates that locomotive, sending one copy to L and the other to R, both copies exchanging the roles of the corresponding

https://doi.org/10.25088/ComplexSystems.33.1.1

filters. Accordingly, both parts of the memory switch operate as necessary. Here too, we have to make sure that the arrival at L and R of the locomotives sent from T occurs later at the permissive controller than the locomotive arriving from P or from Q. Figure 9 ensures that it is easy to fulfill that condition, using an arc of a circle with a large enough radius.

Note that the separation of the two parts of the memory switch allows using one passive part to change several active parts, a situation we will soon encounter.

2.2.4 The One-Bit Memory

We are now ready to implement the one-bit memory in our setting. The implementation is illustrated by Figure 10, which reminds us of the Euclidean implementation in 10(a). We adopt the same notations as in the Euclidean picture in order to facilitate the comparison for the reader.

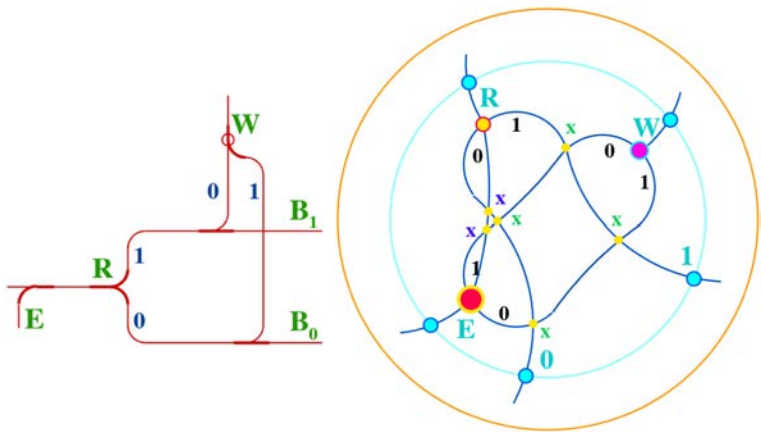


Figure 10. (a) The scheme of Figure 3; (b) its implementation in our setting. In the hyperbolic picture, the colors at W, R and E, namely yellow, light green and light purple, respectively represent a flip-flop switch, an active memory switch and a passive memory switch. The yellow points with an x letter indicate crossings.

Here again, a simple locomotive is involved. If it arrives at the blue disc labeled with R, it goes to the red disc labeled with the same letter, and then it exits through the gate 0 or through the gate 1, according to the track selected by the active memory switch.

If the locomotive arrives at the blue W, it is sent to the red W, where a flip-flop switch sends the locomotive to the red E either through track 0 or track 1, depending on what is selected by the

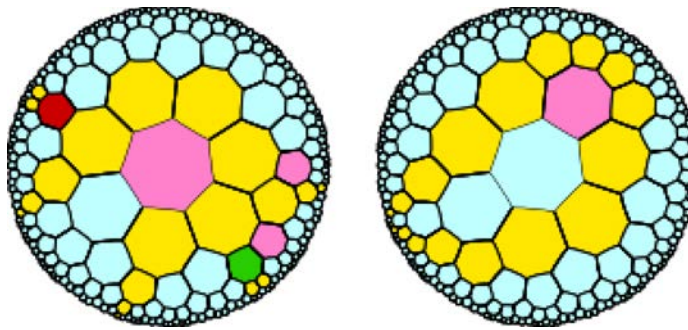
switch at the red R. If the track is i, then it arrives at the red E through its track $1 - i$. Now, from what we have seen in Figure 9, the red E sends a locomotive to the red R, making the switch select the values that are compatible with those at the red E. But from the red E, another simple locomotive is sent out of the switch through the blue gate E. Note that the memory switch of Figure 10(a) is split into two parts in Figure 10(b). There, the active part sits at R, while the passive one sits at E. Note that there is a path joining E to R.

Since we later use the one-bit memory several times, we represent it by a blue disc with five gates labeled with R, E, W, 0 and 1.

2.2.5 A Register and Its Discriminating Structures

Figure 11 illustrates the implementation of a register in our setting. As can be seen, we have a segment of line enclosed in a path surrounding it.

The value of the register is the number of W-tiles along that line from tile 0 in Figure 11(a) up to tile 0 of Figure 11(b), both tiles included. Denoting the register by $\mathcal{R}$, we denote the W-cells representing its content n by $\mathcal{R}(m)$ with $0 \leq m < n$. By construction, for any n, we have that $\mathcal{R}(n)$ is M. In particular, when the register is empty, whose value is 0 by definition, $\mathcal{R}(0)$ is M.



(a) (b)

Figure 11. (a) The beginning of a register when $n = 0$; (b), its end when $n > 0$. Both ends are marked, since they play an important role.

The register is the occasion to remind the reader of the fact that we have two kinds of locomotives. The B-locomotive is devoted to implementing the register, while the G-locomotive is devoted to decrementing the register. Figure 11(b) suggests that the Y-cells that run around the content of the register behave like a track that is outside a window around $\mathcal{R}(0)$ and outside a window containing the hat of the register, a structure we define in Section 2.2.5(c). The locomotive returning from the register has the same color as it had when entering

the register. However, once it has performed its operation, the locomotive does not remember which instruction of the program instructed it to execute that operation. Accordingly, we need two new structures outside a register: a structure to remember which instruction was required to increment the register, the $\mathbb{DD}_{\parallel}$-structure, see Section 2.2.5(a); and a structure to remember which instruction was required to decrement the register, the $\mathbb{DD}_{\mathbb{D}}$-structure, see Section 2.2.5(b). Note that the locomotive color is enough to characterize which operation it performed. Both structures are required for each register. Those structures also make use of one-bit memories.

2.2.5(a) Remembering the Incrementing Instruction

As can be guessed from Figure 11, when the locomotive returns from the register, and we only have two types of locomotives, the locomotive does not remember from which point of the program it was issued. In order to go back to that point to define the next instruction, the instruction must be marked in some way near the register itself. The $\mathbb{DD}_{\parallel}$-structure is devoted to playing that role. There is a $\mathbb{DD}_{\parallel}$-structure for each register $\mathcal{R}$. The structure contains as many units as there are instructions in the program to specifically increment $\mathcal{R}$. From each instruction I incrementing $\mathcal{R}$, a path goes from the place of I in the program to the unit of the $\mathbb{DD}_{\parallel}$ attached to $\mathcal{R}$, associated by that path to that unit. The unit itself contains a one-bit memory; see Figure 12.

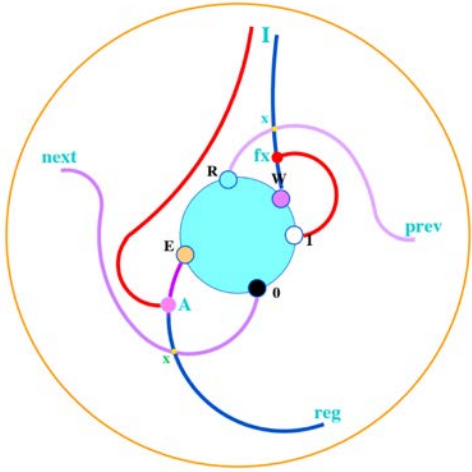


Figure 12. The unit of $\mathbb{DD}_{\parallel}$ devoted to an incrementing instruction.

At the initial configuration, in all units of each $\mathbb{DD}_{\parallel}$-structure, the one-bit memory contains 0. When the locomotive starts on its way to

perform an incrementing instruction I over the register $\mathcal{R}$, the track traveled by the locomotive arrives at the W-gate of the unit of the $\mathbb{DD}_{\|}$ attached to $\mathcal{R}$, which is associated with I. When the locomotive arrives at a W-gate, it rewrites the 0-bit of the one-bit memory to 1. Then, the locomotive exits the memory through its E-gate. It meets a flip-flop at A, which, from the initial configuration, sends it toward $\mathcal{R}$. After crossing that flip-flop switch, the selected track is now a track that goes back to the program, to the instruction that has to be executed after I has completed its operation on $\mathcal{R}$. That track is shown in orange in Figure 12. At that time, there is a single unit in that $\mathbb{DD}_{\|}$, whose one-bit memory contains 1.

When the locomotive returns from $\mathcal{R}$, it is a B-locomotive, and a fork combined with appropriate filters of opposite colors will send that locomotive to the $\mathbb{DD}_{\|}$ of $\mathcal{R}$.

The locomotive visits each unit of the $\mathbb{DD}_{\|}$-structure until it meets the single one whose one-bit memory contains 1. To do that, the locomotive enters the memory of the unit through its R-gate. If it reads 0, it is sent to the next unit. When it reads 1, it knows that the correct unit has been reached. Leaving the one-bit memory through the 1-gate, the locomotive is sent to the W-gate so that it rewrites the content of the one-bit memory from 1 to 0. Leaving the memory through the E-gate, the locomotive again meets A, where the selected track of the flip-flop sends it on the track leading to the correct place in the program. Once the flip-flop at A is crossed, the selected track again becomes the track leading to $\mathcal{R}$. Accordingly, when the locomotive leaves the $\mathbb{DD}_{\|}$-structure after preforming its operation, then $\mathbb{DD}_{\|}$ recovers its initial configuration.

2.2.5(b) Remembering the Decrementing Instruction

If the locomotive has to decrement the register $\mathcal{R}$, it visits a similar structure as $\mathbb{DD}_{\|}$, the $\mathbb{DD}_{\mathbb{D}}$-structure attached to $\mathcal{R}$. The principle of that structure is similar to that of $\mathbb{DD}_{\|}$. However a unit of $\mathbb{DD}_{\mathbb{D}}$ is more complex than a unit of $\mathbb{DD}_{\|}$. The reason comes from the difference between a decrementing instruction and an incrementing one. It is always possible to increment a register. It is not possible to decrement an empty register, because register machines deal only with natural numbers. When a decrementing locomotive arrives at $\mathcal{R}(0)$ when $\mathcal{R}$ is empty, it leaves $\mathcal{R}(0)$ through a different track than if it had successfully decremented it.

Accordingly, if the working of $\mathbb{DD}_{\mathbb{D}}$ is similar to that of $\mathbb{DD}_{\|}$ for an arriving locomotive from the program, it is not the case for a locomotive returning from the register. We leave the case of an arriving locomotive from the program to the reader.

Consider the case of a locomotive returning from $\mathcal{R}$. It comes from the D-track after a successful operation, or it comes through the

Z-track because it could not decrement an empty $\mathcal{R}$. Both tracks behave in the same way: they lead to the R-gate. If the locomotive reads 0, it goes to the next unit through the same kind of track as the one it used to arrive at the current unit. As can be seen, there are two possible exits for the locomotive entering the one-bit through the R-gate when it comes from the register. But in both cases, the exit again splits into two possibilities, depending on which track the locomotive used to enter the unit. In case of an exit to the next unit, the exit track is a D-, Z-track if it was, respectively, the case for the entering track. In case of an exit to the program, the exit track goes to the next instruction or to another one defined by the jump condition if the entering track was D or Z, respectively. The distinction is known at the entry of the unit; see point *F* in Figure 13. It is the reason why a passive memory switch is sitting at *F*. Since that difference has to be known at *B* and at *C*, there is at each of those points an active memory switch connected to the passive one at *F*. The connection to those active memory switches is established by a track from *F* to a fork before reaching *C*, which sends a copy to *C* and another copy to *B*. Accordingly, the whole structure performs what is expected from it in all situations.

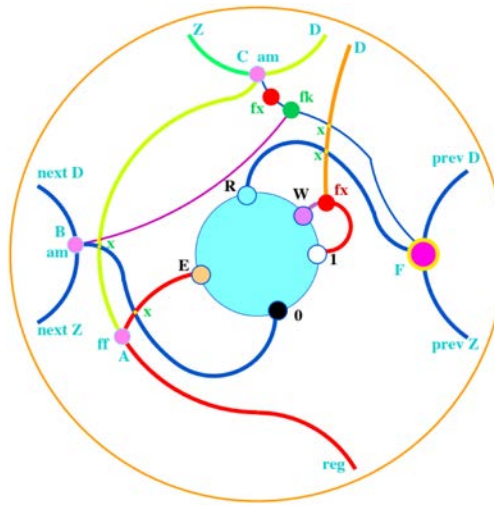


Figure 13. The unit of $\mathbb{DD}_{\mathbb{D}}$ devoted to a decrementing instruction.

Figure 14 illustrates the tracks followed by a locomotive returning from the register. Its color indicates which operation it performed. Coming from the register, it arrives at *A*, where a fork is sitting. One branch has a B-filter at *B*, meaning that the branch leads to $\mathbb{DD}_{\mathbb{I}}$. The other branch has a G-filter at *G*, so that the branch leads to $\mathbb{DD}_{\mathbb{D}}$.

Accordingly, the locomotive goes to the appropriate structure, which, as already seen, sends it back to the correct place in the program.

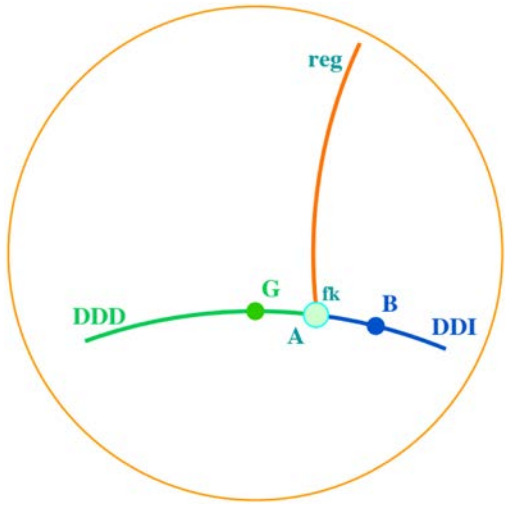


Figure 14. The structure sending a locomotive returning from the register to the appropriate $\mathbb{D}\mathbb{D}_{\mathbb{I}}$- or $\mathbb{D}\mathbb{D}_{\mathbb{D}}$-structure.

2.2.5(c) Decrementing and Incrementing the Register

The operation performed by a locomotive inside the register is illustrated by Figure 15. The figure illustrates a standard situation for a nontrivial value of the content: the M-tile is at some distance from the beginning of the structure, depending on the initial content of the register, which may be empty. In the leftmost image of the figure, we can see the Y-cells 1(6), 1(7) and 1(1) around the M-cell at 0, which we call the *hat* of the register. In the rightmost image of the figure,

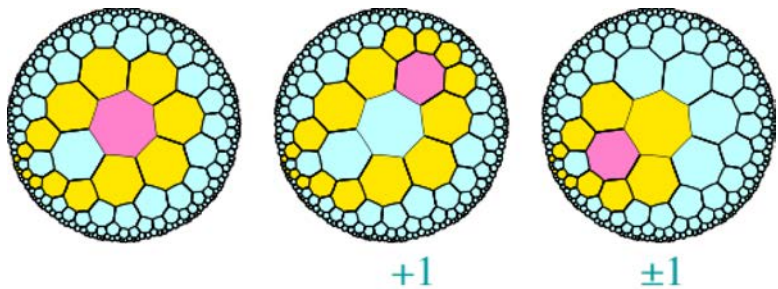


Figure 15. Left, the register before the operation. Right, the configuration after the operation.

https://doi.org/10.25088/ComplexSystems.33.1.1

we also see the hat around the M-cell at 1(3); it consists of the Y-cells 1(4), 0 and 1(2). In the second image from the left, illustrating the incrementing operation, the hat consists of four Y-cells around the M-cell at 1(7): 2(7), 3(7), 4(7) and 2(1).

Figure 15 illustrates how the operation is performed. The figure is split into two parts. To the left, the register before the operation; to the right, the register after it was performed. In that right-hand-side part, two possibilities: to the left, incrementing; to the right, decrementing as mentioned.

The action illustrated by Figure 15 has been tested by a computer program and can be checked in Figure 17 and in Table 1.

3. Rules

The definition of the rules gives us the opportunity to explicitly study the behavior of the automaton.

We start the section by introducing a formalism for writing schemes of rules that will allow us to better present some rules. Then, in each further subsection, we successively examine the cases studied in Section 2.2.

3.1 Fixing Notations

Since we assume the rules to be rotation invariant, we gather the rules having the same neighborhood up to rotation by taking the first one according to the following lexicographic order we set on the states: W, Y, B, G, R, M. From now on, when speaking about the cellular automaton, we will say cell in place of tiles. As an example, the central tile of a window will become the central cell of the window. The reason is that a cell is supported by a tile but it also contains the finite automaton, the same for all cells, which rules the change of cell at each tick of the discrete clock. All those conditions are just an application of the definition of a cellular automaton.

The first letter is the current state of the cell and the last one is the new cell obtained from the current state of the cell and the current states of its neighbors: if we order the neighbors according to the letters from 1 up to 7 from left to right in the word delimited between two commas, the i^{th} letter gives the state of the corresponding i^{th} neighbor of the cell. That latter word is the *neighborhood word*. However, rotation invariance entails that the rule is the same if we replace its neighborhood word by an image of that word under a circular permutation. The rules we present give the permutation that gives the smallest neighborhood word according to the lexicographic order on words.

Take as an example the motion rule on a segment of line for a blue locomotive, namely Y-BWWYWWY:B. It is the minimal form of rule 152 in Table 1, where it is displayed as Y-WWWYWWB:B. The rule replaces the following ones:

a	Y-WWWYWWY:Y	*e*	Y-BWWYWWW:B
b	Y-WWWBWWY:B	*f*	Y-WWYWWWB:B
c	Y-WWBWWYW:B	*g*	Y-WYWWWBW:B
d	Y-WBWWYWW:B	*h*	Y-YWWWBWW:B

Say that a rule is *conservative* if and only if the new state is the same as the current one and also if the states of the neighbors are also unchanged when the rules are applied. If the only state of the cell is unchanged, we speak of a *witness rule*. As an example, rule 7, namely W-WWWWWBR:W, is a witness rule if B- and R-cells in the neighborhood of the current cell to which the rule applies belong to a blue locomotive.

The rules dealing with the motion of a locomotive on tracks are given in Table 1 from rule 16 to rule 162. However, several rules among those are specific to the fork or to the fixed switch.

Note that among the mentioned rules, many of them deal with arcs of a circle. Together with rules applying to a segment of line, the arcs require additional rules. They have the form Y-WWWWFWY:F, where F stands for B or G, the front of a locomotive. We have the rules Y-WWWWFWY:F, the rule F-WWWWRWY:R, the rule R-WWWWYWF:Y and the rule Y-WWWWYWR:Y. We leave their identification in Table 1 as an exercise for the reader. For figures illustrating those motions, see [3].

3.2 Fixed Switch and Fork

We start with the fixed switch, only considering its passive version.

We have four cases: a blue locomotive coming from the left-hand-, right-hand-side branch and the similar two subcases with a green locomotive. See Figures 5 and 6 to follow the application of the rules given by Table 1, where the notations are those introduced in Section 3.1. We also use the same meta-symbol F to replace B and G. For a figure illustrating those motions, see [3] .

We just mention rules 119 and 120 for the fixed switch, namely Y-WWYWYRY:Y and R-WWWWYYY:R, which apply to the Y-cell where all tracks of the switch meet and the R-cell that is a neighbor of the previous Y-cell. That R-cell can see a Y-cell of each track meeting at the switch. Rules 121 and 130, namely Y-WWYRWWB:B and Y-WWYWWRB:Y, show us that the front of the locomotive goes to the leaving track and not to the other side of the switch when it comes from the left. Similarly, rules 146 and 151, namely Y-WWYWBRY:B and Y-WWYWWBR:Y, show us that the symmetric situation happens when the locomotive comes from the right. Note that the same rule 133,

https://doi.org/10.25088/ComplexSystems.33.1.1

namely Y-WWYWWRR:Y, holds for both sides when the rear of the locomotive is seen at the cell where all tracks meet. Those rules, except rule 133, are applied to a B-locomotive. Table 1 contains similar rules for a G-locomotive: it is left as an exercise for the reader to check it.

In the case of a fork, rule 98, namely Y-WWYWYWY:Y, gives the neighborhood of the Y-cell where all tracks meet. There is a W-cell that can see one Y-cell of each track meeting at the fork. For that cell, rule 57 is the conservative rule, namely W-WWWWYYY:W, already met in a W-cell witnessing a track along a circular arc. That W-cell is a good witness of the action of the fork: rule 105, namely W-WWWWBRB:W, indicates that the cell can see both fronts of the locomotives being created from the arriving one and rule 108, namely W-WWWWRYR:W, shows us that the cell can see the rears of both of the leaving locomotives. Similar rules hold for a G-locomotive; it is left to the reader as an exercise.

3.3 Changers and Filters

In the case of the changers, the converter contains a B- or a G-cell. It transforms a locomotive of the opposite color into a locomotive of its color. Looking at Figure 5, it is considered that the locomotive arrives from the right. Accordingly, the first cell of the track seeing the B-or the G-cell is cell 1(7). Rule 167, namely Y-WWBWWGY:G, tells us that the cell 1(7) takes the G-color of the changer when it can see the B-front of an approaching locomotive. Rule 184, namely Y-WWBYWWG:B, illustrates the opposite conversion. Note that rule 167 is obtained from rule 184 by exchanging the states B- and G-, but that change modifies the alphabetic order of the corresponding minimal rule. The rules can be checked in an appropriate figure of [3].

From the use of the filters in Section 2.2, we know that it was required that those structures can change their color: in other words, they must be programmable. Looking again at Figure 5, we can see two Y-cells at the top of the filters, those Y-cells being fitted with an R-cell at each side. Those two Y-cells can be seen as the end of a track arriving at the filter. We require that the filter be changed by a B-locomotive. Rule 243, namely Y-WWRBRWB:G, tells us that the last Y-cell of the arriving change track becomes G when it sees the front of the changing locomotive. Rule 245, namely B-WYYWRGR:G, tells us that see ing the change of the Y-cell to G, the color of the filter is changed. Rule 246, namely G-WWRBRWR:Y, says that the G-cell returns to Y-cell. Similar rules can be seen in Table 1 when the B-locomotive arrives at the G-filter in order to change it into a B-filter.

3.4 Registers

From Section 2.2.5, we know that when the register is empty, $\mathcal{R}(0)$ is M. From Figure 11, we know that the transformation from $\mathcal{R}(n)$ to

$\mathcal{R}(n+1)$ or $\mathcal{R}(n-1)$ entails much work. First, the M-cell moves from $\mathcal{R}(n)$ to $\mathcal{R}(n+1)$ or to $\mathcal{R}(n-1)$ which, in the latter case, assumes that $n > 0$. It also means that the previous hat is either destroyed or modified, and that a new hat is built from scratch or from already nonempty cells.

The case of $n = 0$ requires a special configuration for $\mathcal{R}(0)$ in order not to process the decrementing operation. Instead, the G-locomotive should go along a specific track, as mentioned in Section 2.2.5. Now, when a locomotive goes to $\mathcal{R}(n)$, in order to perform an operation when $n > 1$, it must cross $\mathcal{R}(0)$. Accordingly, there are two possible tracks followed by a locomotive leaving $\mathcal{R}(0)$, depending on whether it could not perform the operation or it could do it, since it has to increment $\mathcal{R}$ or it has to operate on $\mathcal{R}(n)$ for $n > 0$. Figure 11(b) is repeated in Figure 16 in order to better see the situation around $\mathcal{R}(0)$. The two tracks that can be followed by a locomotive leaving the $\mathcal{R}(0)$ fork from cell 1(5): one track goes on to 1(6), the other goes on to 2(6). For technical reasons, cell 2(6) has two M-neighbors: 3(6), shared with 1(6), and 4(5), shared with 1(5).

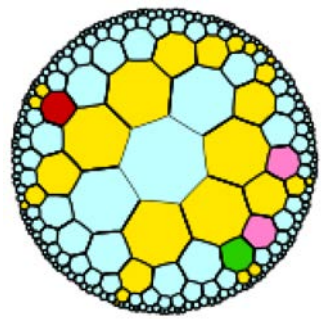


Figure 16. The idle configuration of $\mathcal{R}(0)$. Note that cell 0 is here W, since $n > 0$. Note the M-cells at 4(5) and at 3(6), the R-cell at 3(2) and the G-cell at 3(5).

Appropriate figures in [3] show us the various motions of the locomotive depending on the operation it has to perform and, in the case of decrementing, depending on whether $\mathcal{R}$ is empty or not. Rule 455, namely Y-WGWGMYY:R, tells us in that case, 1(5) becomes R, a signal to 2(6) to remain Y; see rule 446, namely Y-WWYMYRM:Y. When $n = 0$, $\mathcal{R}(0)$ is M, so that rule 330 is the conservative rule of 2(6), namely Y-WWYMYYM:Y, a rule already appearing in Table 1 for the incrementing operation. When a G-locomotive appears when $n = 0$, the applied rule is rule 431, namely Y-WWYMYGM:G, which shows us that, in such a situation, 2(6) becomes G. Next, of course, rule 440 applies, which is G-WWYMYRM:R, and then rule 447 applies,

which is R-WGMYYMY:Y. A superfluous Y-neighbor occurs due to rule 310, namely W-WWWWWGM:Y, a rule used in an incrementing operation. That Y-cell becomes again W by rule 448, Y-WWWWWRM:W. The situation of an incrementing instruction is simpler: when $n = 0$, 1(5) becomes M since a B-locomotive is approaching. So that for 2(6), rule 345 applies, which is Y-WWYMYMM:Y. Accordingly, the locomotive does not take the branch starting from 2(6). When $n > 0$ and the operation is to increment the register, the situation is simpler: rule 261 applies, namely Y-WWYMYBM:Y, showing us that the presence of B does not prevent the Y-cell 2(6) from remaining Y: the presence of two M-neighbors explains that behavior.

The examination of the operations on a register will be completed with a look at how the hat is moved from one position to the other. We constrain that study to the situation when $n > 0$, with in fact $n \geq 2$.

First, we consider the case of incrementing the register. Consequently, the M-cell at $\mathcal{R}(n)$ is translated to $\mathcal{R}(n+1)$. The former hat partly remains as a part of the extended track and the new track has to be built. As soon as the M-cell at $\mathcal{R}(n)$ can see the approaching B-locomotive, the cell becomes B by rule 277, that is, M-WBYYYYY:B, which triggers the cells of the hat to become M for half of them while the other half becomes B. It involves rule 280, which is Y-WWWWYMB:M, and rule 282, namely Y-WWWWYBY:B, which is applied to two cells of the hat. That new configuration of the hat triggers the conversion of W-cells to G and then, from G to Y, completing the change of the W-cells that have to become Y. First, rule 292, that is, W-WWWWWBB:G, and rule 301, that is, W-WWWWWBM:G, are activated and then rule 304, that is, G-WWWWWRM:Y, and rule 308, that is, G-WWWWWMY:Y, complete that stage of the transformation. Next, while each G-cell becomes Y, it transforms a W-neighbor to Y, which completes the creation of the new hat: rule 310, that is, W-WWWWWGM:Y, and rule 309, that is, W-WWWWWMG:Y, perform that completion. In the meantime, rule 290, namely B-WYMMBBY:W, creates a W-cell at $\mathcal{R}(n)$, and rule 300, that is, B-WWWWBBM:M, places the M-signal at $\mathcal{R}(n+1)$.

Figure 17(a) shows the application of the above rules for incrementing a register when $n \geq 1$.

Now, consider the case of a decrementing operation. The M-signal is moved from $\mathcal{R}(n)$ to $\mathcal{R}(n-1)$ and the previous hat is destroyed. When the M-signal at $\mathcal{R}(n)$ can see the approaching G-locomotive, the cell becomes G, rule 387, namely M-WGYYYYY:G, which entails the destruction of the hat in two steps: half of it becomes G, while one Y-cell of the hat becomes W: rule 392, Y-WWWWYGY:G, which is applied to two Y-cells of the hat and rule 390, Y-WWWWYMG:W, for the erasing of one Y-cell of the hat. At the next time, those G-cells become

Y by rule 395, G-WWWWYGG:W, and by rule 400, G-WWWWGGY:W. The last Y-cell of the previous hat becomes W at that time by rule 399, Y-WWWWWGG:W. During those transformations, rule 394, G-WYWYGGY:Y, transforms $\mathcal{R}(n)$ into a Y-cell of the new hat and rule 398, W-WYYYGYY:M, installs the M-signal at $\mathcal{R}(n-1)$.

Figure 17(b) shows the application of those rules decrementing the register when $n \geq 2$.

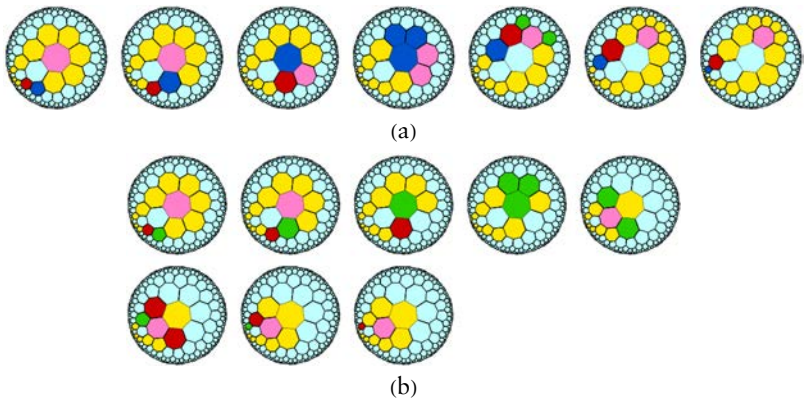


Figure 17. (a) Execution of an incrementing instruction when $n > 0$. (b) Execution of a decrementing instruction when $n > 1$.

Table 1 displays the whole set of rules used by the automaton. The same notation is used as in the previous parts of the paper. We remind the reader of the fact that in the table, the neighboring word is the minimal lexicographic order among the possible equivalent forms of the rule obtained through a circular permutation. However, in Figure 17, it is not difficult to check the rules indicated in the text. Note that in the text we considered rules applied to a few cells, while the rules of Table 1 were devised for all possible cells involved in the computation: cells directly impacted by the computation as well as witnessing cells.

4. Conclusion

We can see that we have 510 rules in Table 1, conservative, witness and motion rules. Note that the registers alone required 257 rules and 135 of those are required for the special cases when the content of the register is 0 or 1.

The open question remains as to whether it is possible to reduce the number of states. It is most probably the case if we relax the condition of rotation invariance.

https://doi.org/10.25088/ComplexSystems.33.1.1

Table 1. Table of the rules used by the cellular automaton.

preliminary rules			
1	W	WWWWWWW	W
2	W	WWWWWWY	W
3	Y	WWWWWWW	Y
4	W	WWWWWWB	W
5	W	WWWWWWG	W
6	W	WWWWWWR	W
7	W	WWWWWBR	W
8	W	WWWWWGR	W
9	W	WWWWWRB	W
10	W	WWWWWRG	W
11	Y	WWWYWWY	Y
12	W	WWWWWYY	W
13	W	WWWWWYB	W
14	W	WWWWWRY	W
15	Y	WWWWWWY	Y
tracks			
B-locomotive			
16	W	WWWWWBY	W
17	B	WWWRWWY	R
18	R	WWWYWWB	Y
19	W	WWWWWYR	W
20	Y	WWWWWWR	Y
21	Y	WWWYWWR	Y
22	Y	WWWWWWB	Y
G-locomotive			
23	W	WWWWWGY	W
24	G	WWWRWWY	R
25	R	WWWYWWG	Y
26	W	WWWWWYG	W
27	Y	WWWWWWG	Y
tracks on arc			
28	W	WWYYYYY	W
B-locomotive			
29	W	WWBYYYY	W
30	W	WWRBYYY	W
31	W	WWYRBYY	W
32	W	WWYYRBY	W
33	W	WWYYYRB	W
34	W	WWYYYYR	W
35	W	WWYYYYB	W
36	Y	WWWWBWY	B
37	W	WWYYYBR	W
38	B	WWWWRWY	R
39	W	WWYYBRY	W
40	R	WWWWYWB	Y
41	W	WWYBRYY	W
42	Y	WWWWYWR	Y
43	W	WWWYYYB	W
44	W	WWBRYYY	W
45	W	WWRYYYY	W
46	W	WWWYBRY	W
47	W	WWWBRYY	W
tracks on arc, continued			
B-locomotive, continued			
48	W	WWWRYYY	W
49	W	WWWWYRB	W
50	W	WWWBYYY	W
51	W	WWWYYYY	W
52	R	WWWWBWY	Y
53	B	WWWYWWR	R
54	Y	WWWWYWB	B
55	Y	WWWWYWY	Y
56	W	WWWWYYY	W
57	W	WWWWYYR	W
58	W	WWWRBYY	W
59	Y	WWWWRWY	Y
60	R	WWWBWWY	Y
61	W	WWWYRBY	W
62	Y	WWWRWWY	Y
63	B	WWWWYWR	R
64	Y	WWWYWWB	B
65	W	WWWYYRB	W
66	W	WWWYYYR	W
67	W	WWWWBYY	W
68	W	WWWWRBY	W
G-locomotive			
69	W	WWGYYYY	W
70	W	WWRGYYY	W
71	W	WWYRGYY	W
72	W	WWYYRGY	W
73	W	WWYYYRG	W
74	W	WWYYYYG	W
75	Y	WWWWGWY	G
76	W	WWYYYGR	W
77	G	WWWWRWY	R
78	W	WWYYGRY	W
79	R	WWWWYWG	Y
80	W	WWYGRYY	W
81	W	WWWGRYY	W
82	W	WWWYYYG	W
83	W	WWGRYYY	W
84	W	WWWYGRY	W
85	W	WWWWYRG	W
86	W	WWWGYYY	W
87	R	WWWWGWY	Y
88	G	WWWYWWR	R
89	Y	WWWWYWG	G
90	W	WWWRGYY	W
91	R	WWWGWWY	Y
92	W	WWWYRGY	W
93	G	WWWWYWR	R
94	Y	WWWYWWG	G
95	W	WWWYYRG	W
96	W	WWWWGYY	W
97	W	WWWWRGY	W

Table 1. Table of the rules used by the cellular automaton (*continued*).

fork				
98	Y	WWYWYWY	Y	
B-locomotive				
99	Y	WWBWYWY	B	
100	W	WWWWYYB	W	
101	B	WWRWYWY	R	
102	W	WWWWYBY	W	
103	W	WWWWYBR	W	
104	R	WWYWBWB	Y	
105	W	WWWWBRB	W	
106	W	WWWWBRY	W	
107	Y	WWYWRWR	Y	
108	W	WWWWRYR	W	
109	W	WWWWRYY	W	
110	R	WWWYWWY	Y	
G-locomotive				
111	Y	WWGWYWY	G	
112	W	WWWWYYG	W	
113	G	WWRWYWY	R	
114	W	WWWWYGY	W	
115	W	WWWWYGR	W	
116	R	WWYWGWG	Y	
117	W	WWWWGRG	W	
118	W	WWWWGRY	W	
fixed switch				
119	Y	WWYWYRY	Y	
120	R	WWWWYYY	R	
from left				
B-locomotive				
121	Y	WWYRWWB	B	
122	Y	WWYWWRY	Y	
123	Y	WWYWYRB	B	
124	R	WWWWBYY	R	
125	B	WWYRWWR	R	
126	B	WWYWYRR	R	
127	R	WWWWRBY	R	
128	R	WWYWWBR	Y	
129	W	WWWWWRR	W	
130	Y	WWYWWRB	Y	
131	R	WWBWYRY	Y	
132	R	WWWWYRY	R	
133	Y	WWYWWRR	Y	
134	Y	WWRWYRY	Y	
135	Y	WWYWWYR	Y	
G-locomotive				
136	Y	WWYRWWG	G	
137	Y	WWYWYRG	G	
138	R	WWWWGYY	R	
139	G	WWYRWWR	R	
140	G	WWYWYRR	R	
141	R	WWWWRGY	R	
142	R	WWYWWGR	Y	
from left (continued)				
G-locomotive				
143	Y	WWYWWRG	Y	
144	R	WWGWYRY	Y	
from right				
B-locomotive				
145	Y	WWBWWRY	B	
146	Y	WWYWBRY	B	
147	R	WWWWYYB	R	
148	B	WWRWWRY	R	
149	B	WWYWRRY	R	
150	R	WWWWYBR	R	
151	Y	WWYWWBR	Y	
152	Y	WWWBWWY	B	
153	R	WWYWWRB	Y	
G-locomotive				
154	Y	WWGWWRY	G	
155	Y	WWYWGRY	G	
156	R	WWWWYYG	R	
157	G	WWRWWRY	R	
158	G	WWYWRRY	R	
159	R	WWWWYGR	R	
160	Y	WWYWWGR	Y	
161	Y	WWWGWWY	G	
162	R	WWYWWRG	Y	
converters				
from B-locomotive to G-locomotive				
163	Y	WWWYGWY	Y	
164	G	WWYYWYY	G	
165	Y	WWWWWYG	Y	
166	Y	WWWWWGY	Y	
167	Y	WWBWWGY	G	
168	Y	WWWGGWY	G	
169	G	WWYGWYY	G	
170	W	WWWWYGG	W	
171	G	WWGYWWR	R	
172	G	WWWRGWY	R	
173	G	WWGRWYY	G	
174	R	WWYWWGG	Y	
175	R	WWWYGWG	Y	
176	G	WWRYWYY	G	
177	Y	WWWYGWR	Y	
178	W	WWWWRYG	W	
179	Y	WWYWWGY	Y	
from G-locomotive to B-locomotive				
180	Y	WWWYBWY	Y	
181	B	WWYYWYY	B	
182	Y	WWWWWYB	Y	
183	Y	WWWWWBY	Y	
184	Y	WWBYWWG	B	
185	Y	WWWBBWY	B	
186	B	WWYBWYY	B	

Table 1. Table of the rules used by the cellular automaton (*continued*).

converters, continued			
from G-locomotive to B-locomotive			
187	W	WWWWYBB	W
188	B	WWBYWWR	R
189	B	WWWRBWY	R
190	B	WWBRWYY	B
191	R	WWYWWBB	Y
192	R	WWWYBWB	Y
193	B	WWRYWYY	B
194	Y	WWWYBWR	Y
195	W	WWWWRYB	W
196	Y	WWYWWBY	Y
B-filter			
B-locomotive, OK			
197	B	WYYWRYR	B
198	Y	WWRBRWY	Y
199	R	WWWWWBY	R
200	W	WWWYYBR	W
201	Y	WWWBWBY	B
202	W	WWWRBYB	W
203	R	WWWWWYB	R
204	B	WYBWRYR	B
205	B	WWWRWBY	R
206	W	WWWRBBR	W
207	B	WBRWRYR	B
208	W	WWWYBBR	W
209	R	WWWYWBB	Y
210	W	WWWRBRY	W
211	B	WRYWRYR	B
212	W	WWWBRBR	W
213	Y	WWWYWBR	Y
214	W	WWWRYBR	W
215	Y	WWWYWBY	Y
G-locomotive, NO			
216	Y	WWWGWBY	Y
217	W	WWWRBYG	W
218	Y	WWWRWBY	Y
219	W	WWWRBYR	W
G-filter			
G-locomotive, OK			
220	G	WYYWRYR	G
221	Y	WWRGRWY	Y
222	R	WWWWWGY	R
223	W	WWWYYGR	W
224	Y	WWWGWGY	G
225	W	WWWRGYG	W
226	R	WWWWWYG	R
227	G	WYGWRYR	G
228	G	WWWRWGY	R
229	W	WWWRGGR	W
230	G	WGRWRYR	G
231	W	WWWYGGR	W
232	R	WWWYWGG	Y
G-filter, continued			
G-locomotive,OK			
233	W	WWWRGRY	W
234	G	WRYWRYR	G
235	W	WWWGRGR	W
236	Y	WWWYWGR	Y
237	W	WWWRYGR	W
238	Y	WWWYWGY	Y
B-locomotive, NO			
239	Y	WWWBWGY	Y
240	W	WWWRGYB	W
241	Y	WWWRWGY	Y
242	W	WWWRGYR	W
changing filters			
from B -t o G-			
243	Y	WWRBRWB	G
244	W	WWWWBYR	W
245	B	WYYWRGR	G
246	G	WWRBRWR	Y
247	W	WWWWRGR	W
248	R	WWWWWBG	R
249	R	WWWWWGB	R
from G- to B-			
250	Y	WWRGRWB	B
251	G	WYYWRBR	B
252	B	WWRGRWR	Y
253	W	WWWWRBR	W
registers incrementing across $\mathcal{R}(0)$ to $\mathcal{R}(n)$			
254	W	WWWBWYY	W
255	Y	WBWGMYY	B
256	B	WGMYYWR	R
257	W	WWWWGBR	W
258	G	WWYYWMB	G
259	M	WWWWYBG	M
260	Y	WWBYMWY	B
261	Y	WWYMYBM	Y
262	R	WYWGMYB	Y
263	B	WWRYMWY	R
264	Y	WWYMBRM	Y
265	M	WWWWBYY	M
266	W	WWWWYBM	W
267	R	WWYYMWB	Y
268	W	WWWWBRM	W
$n \to n+1$, $n > 0$			
269	M	WYYYYYY	M
270	Y	WWWWYMY	Y
271	Y	WWWYWMY	Y
272	W	WRBYMYY	W
273	W	WYYWYYR	W
274	W	WYYWYYY	W
275	W	WWWWYWY	W
276	Y	WWWYMWB	B
277	M	WBYYYYY	B

Table 1. Table of the rules used by the cellular automaton (*continued*).

$n \to n+1$, $n > 0$ continued			
278	W	WYRBMYY	W
279	B	WWWYMWR	R
280	Y	WWWWYMB	M
281	B	WRMYYYY	B
282	Y	WWWWYBY	B
283	W	WYYRBYY	W
284	R	WWWMBWY	Y
285	M	WWWWYBR	M
286	W	WWWWWMR	W
287	W	WWWWWWM	W
288	Y	WWWWYBM	M
289	W	WWWWWYM	W
290	B	WYMMBBY	W
291	B	WWWWYBB	R
292	W	WWWWWBB	G
293	Y	WWWYWBB	B
294	W	WYYYBYY	W
295	Y	WWWMBWY	Y
296	M	WWWWMBY	Y
297	W	WWWWWMY	W
298	M	WWWWBBM	Y
299	W	WWWWWMM	W
300	B	WWWWBBM	M
301	W	WWWWWBM	G
302	W	WYYYMRB	W
303	R	WWWBWMG	Y
304	G	WWWWWRM	Y
305	W	WYYYWBY	W
306	Y	WWWGMWY	Y
307	M	WWGRWYG	M
308	G	WWWWWMY	Y
309	W	WWWWWMG	Y
310	W	WWWWWGM	Y
311	W	WYYYMYR	W
312	Y	WWWRWMY	Y
313	Y	WWWWWYM	Y
314	W	WYYYWRB	W
315	Y	WWWYMWY	Y
316	M	WWYYWYY	M
317	W	WYYYMYY	W
318	W	WYYYWYR	W
319	W	WYYYWRY	W
$0 \to 1$			
320	Y	WYWMYWR	Y
321	R	WWYWWYY	R
322	Y	WWWWWYR	Y
323	Y	WWWWWRY	Y
324	W	WWWYMYY	W
325	Y	WWYMWWB	B
326	Y	WGMYYMY	Y
327	G	WWYYWMY	G
328	M	WWWWYYG	M
329	W	WWWWMGY	W
$0 \to 1$ continued			
330	Y	WWYMYYM	Y
331	Y	WWYWYYM	Y
332	M	WWWWYYY	M
333	W	WWWWMYY	W
334	W	WWWBMYY	W
335	B	WWYMWWR	R
336	Y	WGMYYMB	M
337	W	WWWWGYB	W
338	Y	WYWBYWR	Y
339	R	WWYWWMB	Y
340	M	WGMYYBR	M
341	W	WWWWGMR	W
342	G	WWYYWMM	G
343	M	WWWWYMG	M
344	Y	WWYBMYM	M
345	Y	WWYMYMM	Y
346	Y	WYWBBWR	B
347	W	WWWYBYY	W
348	Y	WWYWWMB	Y
349	M	WGMYMBY	Y
350	W	WWWWGMY	W
351	M	WWBBMYM	Y
352	Y	WWYMMMM	Y
353	M	WWWWMYY	M
354	B	WWRWRWY	R
355	R	WWYYWWB	R
356	W	WWWYWBY	W
357	Y	WYWGMYY	Y
358	Y	WYYMWGM	Y
359	W	WWWWGYM	W
360	R	WWYWRWB	Y
361	W	WWWWRRY	W
362	R	WWYYWWR	R
363	W	WWWWBRR	W
364	W	WWWYWRB	W
365	Y	WYYMWYM	Y
366	W	WWWWYYM	W
367	W	WWWYWYR	W
368	Y	WWYWRWY	Y
369	W	WWWYWYY	W
back to program			
370	Y	WWYYMWY	Y
371	W	WYYWRBY	W
372	W	WYYWBYY	W
373	W	WYYWYRB	W
374	Y	WWBWRWY	B
decrementing across $\mathcal{R}(0)$ to $\mathcal{R}(n)$			
375	W	WYYWRRY	W
376	Y	WWRYMWY	G
377	W	WYYWYRY	W
378	G	WWRYMWY	R
379	Y	WYWGMYR	Y
380	R	WWYYMWG	Y

https://doi.org/10.25088/ComplexSystems.33.1.1

Table 1. Table of the rules used by the cellular automaton (*continued*).

decrementing across $\mathcal{R}(0)$ to $\mathcal{R}(n)$ continued			
381	Y	WWYMRYM	Y
382	M	WWWWRYY	M
383	W	WWWWGRM	W
384	Y	WWYYMWR	Y
$n \to n-1$, $n > 1$			
385	W	WRGYMYY	W
386	Y	WWWYMWG	G
387	M	WGYYYYY	G
388	W	WYRGMYY	W
389	G	WWWYMWR	R
390	Y	WWWWYMG	W
391	G	WYYYYWR	G
392	Y	WWWWYGY	G
393	W	WYYRGYY	W
394	G	WYWYGGY	Y
395	G	WWWWYGG	W
396	W	WWWWWGG	W
397	Y	WWWYWGG	G
398	W	WYYYGYY	M
399	Y	WWWWWGG	W
400	G	WWWWGGY	W
401	Y	WWWWGMG	Y
402	G	WWWWYMY	R
403	M	WYYGYGY	M
404	Y	WWWYWMG	G
405	Y	WWWWGMY	Y
406	Y	WWWWRMR	Y
407	R	WWWWGMY	Y
408	M	WYYRYRG	M
409	G	WWWYWMR	R
410	W	WYYYMGY	W
411	R	WWWWYMY	Y
412	Y	WWWWRMY	Y
413	M	WYYYYYR	M
414	R	WWWGWMY	Y
415	W	WYYYMRG	W
416	W	WYYYWGY	W
decrementing $n = 0$, failure			
417	Y	WWYMWWG	G
418	W	WWWGMYY	W
419	G	WWYMWWR	R
420	Y	WGMYYMG	G
421	W	WWWWGYG	W
422	G	WRGYYYY	M
423	Y	WYWGYWR	Y
424	R	WWYWYGG	Y
425	W	WWWWYRY	W
426	G	WGMYYGR	R
427	W	WWWWGGR	W
428	G	WWYYWMG	G
429	M	WWWWYGG	M
430	Y	WWYGGYM	Y
decrementing $n = 0$, failure continued			
431	Y	WWYMYGM	G
432	M	WYRYGGY	M
433	G	WWWWYMG	Y
434	Y	WYWMGWR	Y
435	Y	WWYWWRM	Y
436	R	WGMGYMY	Y
437	G	WWYYWMR	G
438	M	WWWWGRG	M
439	Y	WWGMRGM	Y
440	G	WWYMYRM	R
441	M	WWWWYGY	M
442	G	WWWWGMY	Y
443	Y	WWYWWYM	Y
444	Y	WGMRYMY	Y
445	M	WWWYRYG	M
446	Y	WWYMYRM	Y
447	R	WGMYYMY	Y
448	Y	WWWWWRM	W
449	Y	WWRMYYM	Y
450	M	WWWWYYR	M
451	R	WWWYWMY	Y
452	W	WWWWMRY	W
$1 \to 0$			
453	W	WGYYMYY	W
454	W	WWWGWYY	W
455	Y	WGWGMYY	R
456	W	WRRYMYY	W
457	W	WWWRWYY	W
458	R	WWWYWWR	Y
459	R	WGMYYWR	R
460	M	WWWWYRG	M
461	W	WWWWGRR	W
462	Y	WYMWRYM	G
463	Y	WWYWYRM	Y
464	W	WYYGMYY	W
465	R	WYWGMYG	Y
466	G	WYMWRYM	Y
467	Y	WWYWGRM	Y
468	Y	WWYMGRM	Y
469	M	WWWWGYY	M
470	W	WWWWYGM	W
471	Y	WYWGWYR	Y
472	Y	WWGWYYM	Y
473	G	WYWYYYY	Y
474	Y	WWWYMYG	G
475	G	WWWWYYG	W
476	Y	WYGGYMY	M
477	G	WWWWGYY	W
478	M	WYYYMGY	M
479	G	WWWWYMM	G
480	Y	WWMMYYM	Y
481	M	WWGMYWY	M

Table 1. Table of the rules used by the cellular automaton (*continued*).

$1 \to 0$ continued			
482	W	WWWWYMY	W
483	Y	WWWWWWM	W
484	G	WWWYMMY	M
485	Y	WWWWWGM	W
486	Y	WWYMMYY	Y
487	M	WWWYGMY	Y
488	Y	WWWWWMY	W
489	M	WYYYYMY	M
490	M	WWWWYMY	R
491	Y	WYWMMWR	G
492	W	WWWWRYM	W
493	Y	WWWWMMY	Y
494	M	WYYYYRG	M
495	G	WYWMRWR	R
496	R	WWYYWWG	R
497	W	WWWYMGY	W
498	R	WGWMYWR	Y
499	W	WWWYMRG	W
500	Y	WRWRWMY	Y
501	W	WWWYMYR	W

back to program			
502	W	WYYWRGY	W
503	W	WYYWGYY	W
504	W	WYYWYRG	W
505	Y	WWGWRWY	G
506	W	WYYYWRG	W
507	G	WWRWRWY	R
508	W	WWWYWGY	W
509	R	WWYWRWG	Y
510	W	WWWYWRG	W

References

[1] M. Margenstern, *Cellular Automata in Hyperbolic Spaces, Volume 1: Theory*, Collection: *Advances in Unconventional Computing and Cellular Automata* (A. Adamatzky, ed.), Philadelphia: Old City Publishing, Inc., 2007.

[2] M. Margenstern, "New Tools for Cellular Automata in the Hyperbolic Plane," *Journal of Universal Computer Science*, **6**(12), 2000 pp. 1226–1252. doi:10.3217/jucs-006-12-1226.

[3] M. Margenstern, "A Strongly Universal Weighted Cellular Automaton on the Heptagrid with Six States." arxiv.org/abs/2304.13575.

[4] M. Margenstern, "A Strongly Universal Cellular Automaton on the Heptagrid with Seven States." arxiv.org/abs/2102.04292.

[5] M. Margenstern, "A Strongly Universal Cellular Automaton on the Heptagrid with Seven States, New Proof." arxiv.org/abs/2304.13575v1.

[6] F. Herrmann and M. Margenstern, "A Universal Cellular Automaton in the Hyperbolic Plane," *Theoretical Computer Science*, **296**(2), 2003 pp. 327–364. doi:10.1016/S0304-3975(02)00660-6.

[7] M. Margenstern and Y. Song, "A New Universal Cellular Automaton on the Pentagrid," *Parallel Processing Letters*, **19**(2), 2009 pp. 227–246. doi:10.1142/S0129626409000195.

[8] M. Margenstern, "A Weakly Universal Cellular Automaton in the Pentagrid with Five States," *Computing with New Resources: Essays Dedicated to Jozef Gruska on the Occasion of His 80th Birthday* (C. S. Calude, R. Freivalds and I. Kazuo, eds.), Cham: Springer, 2014 pp. 99–113. doi:10.1007/978-3-319-13350-8_8.

[9] M. Margenstern, "A Weakly Universal Cellular Automaton in the Heptagrid with Three States." arxiv.org/abs/1410.1864.

[10] M. Margenstern, "A Weakly Universal Cellular Automaton in the Heptagrid of the Hyperbolic Plane," *Complex Systems*, **27**(4), 2018 pp. 315–354. doi:10.25088/ComplexSystems.27.4.315.

[11] M. Margenstern, "An Upper Bound on the Number of States for a Strongly Universal Hyperbolic Cellular Automaton on the Pentagrid," *Journées Automates Cellulaires 2010 (JAC2010)* Turku, Finland, Proceedings, Turku Center for Computer Science, 2010 pp. 168–179. hal.science/hal-00541965.

[12] M. L. Minsky, *Computation: Finite and Infinite Machines*, Englewood Cliffs, NJ: Prentice-Hall, 1967.

[13] I. Stewart, "A Subway Named Turing," *Scientific American*, **271**(3), 1994 p. 104. doi:10.1038/scientificamerican0994-104.