A strongly universal cellular automaton on the heptagrid with seven states

Maurice Margenstern
February 9, 2021

Abstract
In this paper, we prove that there is a strongly universal cellular automaton on the heptagrid with seven states which is rotation invariant. This improves a previous paper of the author where the automaton required ten states.

1 Introduction
The first paper about a universal cellular automaton in the pentagrid, the tessellation \( \{5, 4\} \) of the hyperbolic plane, was [1]. This cellular automaton was also rotation invariant, at each step of the computation, the set of non quiescent states had infinitely many cycles: we shall say that it is a truly planar cellular automaton. But, the neighbourhood was à la von Neumann and it had 22 states. This result was improved by a cellular automaton with 9 states in [9]. Recently, I improved this result with 5 states, see [6]. A bit later, I proved that in the heptagrid, the tessellation \( \{7, 3\} \) of the hyperbolic plane, there is a weakly universal cellular automaton with three states which is rotation invariant and which is truly planar, [7]. Later, I improved the result down to two states but the rules are no more rotation invariant. In the present paper, I construct a cellular automaton with seven states which is rotation invariant and truly planar.

In the result of [4], the automaton evolves on a construction which follows a line, constructing it step by step, starting from the finite configuration. On the line, that automaton simulates a Turing machine which, in its turn, simulates a tag systems known to be universal when the deleting number is at least 2, see [10]. In the present paper we follow a different strategy. We explain it in Section 2. In Section 3 we describe in detail the structures which are involved in the new construction. In Section 4 we give the rules which were checked by a computer program which also computed many figures of the paper.

All these sections constitute the proof of the following result:

**Theorem 1** There is a strongly universal cellular automaton on the heptagrid which is rotation invariant, truly planar and which has seven states.
The minimal introduction to hyperbolic geometry needed to understand the paper can be found in [5]. We assume the reader is a bit familiar with that approach so that Sections 2 and 3 rely on it with no further reference.

2 Hyperbolic railway and register machines

In the papers were I constructed weakly universal cellular automata in the pentagrid and in the heptagrid, I used the model devised by Ian Stewart in [11] to simulate the computation of a register machine. That model makes use of a circuit constituted by switches connected by tracks to simulate the computation. The complexity of the tracks used in those papers made it impossible to start from a finite configuration within a small number of states. However, the line constructed in [4] made use of a comparatively small number of states, a part of which being also used for the simulation of the particular Turing machine used in that paper.

As we start from a finite configuration, the idea is to implement the data stored in a register as simply as possible. Here, the content of the register which is supposed to be a natural number, is represented by that number written in unary. If the content at time $t$ is $N$, it is represented by $N$ cells in an appropriate state stored along a line of the hyperbolic plane. As seen in Figure 7, such a line is easily characterised in the heptagrid. For the rest of the computation, our simulation follows the same lines as in the quoted papers: the instructions of the register machine and the access to the register are managed in the same way up to the adaptation of the tracks we shall soon consider.

2.1 The railway model

As already mentioned, the railway model was introduced by Ian Stewart in [11]. It was initially devised to simulate a Turing machine, but I used it to simulate a register machine. For technical reasons, that latter model is easier to simulate as it is known that two registers are enough to simulate any Turing machine on $\{0, 1\}$, see [10].

The model consists of a railway circuit on which a single locomotive is running. The circuit contains crosses and switches and the state of all switches of the circuit considered at a given top of the clock constitutes the configuration of the circuit at that time. The circuit consists of tracks which are represented by assembling segments of straight lines, either horizontal or vertical ones and quarters of a circle. That representation lies in the Euclidean plane. In the hyperbolic one, things are necessarily different. We turn to that point in Section 3.

It contains crosses which allows tracks to intersect. It also contain switches in order to make suitable figures, in particular cycles. There are three kinds of switches illustrated by Figure 11. From left to right in the figure, we can see the fixed switch, the flip-flop and the memory switch. We can see that a switch realizes the intersection of three tracks, we shall denote them by $a$, $b$ and $c$. Say that $a$ is the single track which arrives at the switch and that $b$ and $c$
are those from which the locomotive may leave the switch, either through \( b \) or through \( c \). The track taken by the locomotive to leave the switch is called the selected track. When the locomotive arrives to the track through \( a \) and leaves it through the selected track, we say that it is an active crossing. If the locomotive arrives through \( b \) or through \( c \), leaving the switch through \( a \), we say that it is a passive crossing. In the figure, each kind of switch is represented in two forms. Both of them will be used in our further illustrations.

![Figure 1](image1)

**Figure 1** The switches used in the railway circuit of the model.

In the fixed switch, the selected track is determined once and for all. The switch can be crossed both actively or passively. The flip-flop may be crossed actively only. When it leaves the switch, the selected track changes: the previous non-selected track becomes selected and the previously selected becomes non-selected. The memory switch can also be crossed either actively or passively. The selected track is determined by the last passive crossing. These switches are exactly those defined in [11]. We also take from paper [11] the configuration illustrated by Figure 2 which contains one bit of information.

![Figure 2](image2)

**Figure 2** The basic element containing one bit of information.

This configuration, called the basic element, is used as follows. If the locomotive arrives through \( R \), it goes either through track 0 or through track 1. If the locomotive goes through the track \( \alpha \) with \( \alpha \in \{0, 1\} \), it leaves the configuration through \( E_\alpha \). We say that the locomotive has read \( \alpha \). If the locomotive arrives through \( W \), it goes through track 0 or 1. Now, if it arrives through the track \( \alpha \), it passively crosses \( R \) through the track \( \beta \), with \( \beta \in \{0, 1\} \) and \( \alpha + \beta = 1 \). Eventually, the locomotive leaves the element through \( E \). Now, as the locomotive crossed \( W \), the selected track at \( W \) is now \( \beta \). Consequently, we can say that the selected tracks at \( R \) and \( W \) are always the same when the locomotive is arriving at the configuration and also when it leaves it. Accordingly, the locomotive may read 0 or 1 and also it may rewrite \( \alpha \) into \( \beta \).
2.2 Register machine

Many instructions arrive to a register. In [5] for instance, we described two dispatching structures: one of them to gather the instructions to increment that register and the other to gather the instructions to decrement that register. The structure has a single output way to the register according or the type of instructions arriving to it. On the way back, the structure sends out the locomotive to the corresponding sending instructions. Figure 3 displays both structures: on the upper picture, the dispatcher of instructions for incrementing that register; on the lower picture it performs the same for the instructions which decrement the same register.

![Diagram of Register Machine](image_url)

Figure 3 Gathering instructions on their way to a register and dispatching the return locomotive to the appropriate way on its way back.

We can see on the figure the role played by the basic unit to register the returning path by setting both basic units to 1 when the locomotive arrives from an instruction to decrement the register. It is simpler for the instructions which increment the register: a memory switch is enough for that purpose. For an instruction which decrements the register it is less simple as far as two cases may occur: the case when it was possible to decrement the register and the opposite case when the register was already zero at the arrival of the locomotive. It is the reason why two units are used for each arrival from an instruction which decrements the register. Both units are marked to 1 when the locomotive arrives from the instruction. One way is used when the register was actually decremented the other way when it could not be. One unit is read by the loco-
motive when it could decrement the register, the other one is read when it could not do it. When the unique couple of units set to 1 is detected, the locomotive resets both units to 0 and it goes to the appropriate destination.

However, in the register, when the operation was applied the locomotive goes back to the instructions through the same way. This is way that way goes through a special unit which remembers whether the locomotive coming to the register has to increment it or to decrement it. The unit is 0 for an instruction which increments the register, it was set to 1 in the other case. And so, on the way back, the locomotive knows where it has to go further. Note that the test to zero leads to the gathering units directly.

Figure 4 displays the circuit of that selection.

![Figure 4](image)

**Figure 4** The unit which remembers whether the instruction was to decrement the register.

### 2.3 Hyperbolic implementation of the model

The model of computation described in Sub-section 2.1 is described in [11] as a circuit lying in the Euclidean plane. It cannot been transported into the hyperbolic plane without detailed explanations. There are two steps in this translation. First, we need to have a global view of how to implement the circuitry which will simulate the computation. This is what is discussed in Sub-section 2.4. Once this question is solved, we can look at the details which is the purpose of Sub-section 2.5.

### 2.4 Global view

In order to fix things, we consider the Poincaré’s disc model as the frame of our investigation. We take this model for two reasons. The first one is that it preserves angles, which allow us to see something. The second reason is that the model gives us a global view on the hyperbolic plane. However, we must be aware of two big restrictions. The first restriction is that the model imposes a distortion on distances: when we are close to the centre of the disc, the distances
are small and the closer we go to the border, the bigger are the distances. Also, second restriction, there is no similarity in the hyperbolic geometry. There, distances are absolute. This means that if we change our unit to measure distances by another one, the new distances are not simply multiplied by a simple factor.

A good image is the following one. We have to see Poincaré’s disc as a window over the hyperbolic plane, as flying over that plane in some spacecraft. Accordingly, the centre of Poincaré’s disc is not the centre of the hyperbolic plane: that plane has no centre. The centre of the disc is the centre of our window in the spacecraft: it is the point of the plane on which our attention is focusing. But what is close to the border is not affected by changing the point at which we are looking. If we change of unit, this amounts to change our altitude over the hyperbolic plane in our image. We shall have a larger view close to the centre but what is close to the border still remains out of our view.

Figure 5 shows how to implement a two-registered machine.

\begin{figure}[h]
\centering
\includegraphics[width=0.6\textwidth]{figure5.png}
\caption{A global look at our implementation. For the sake of simplicity, the represented machines has six instructions and two registers. We have to imagine other circles inside the three circles from which are drawn the arcs representing the instructions going to the registers. Blue arcs represent incrementing instructions, brown ones represent those that decrement registers.}
\end{figure}

In Sub-section 2.5 we explain the particular features of Figure 5, we know turn to that sub-section.

Our implementation is not directly performed in the hyperbolic plane, but in a tiling of this plane, more precisely the \textbf{heptagrid}, the tessellation \{7,3\} of the hyperbolic plane which is illustrated by the left-hand side part of Figure 6.

It is important to know that the tiling can be generated by a tree. The tree can be seen on each sector of the right-hand side picture of Figure 6 by the following property. Green and yellow tiles have three sons: blue, green and yellow; while blue tiles have only two sons: blue and green. On the left-hand side picture of Figure 6 we can see the \textbf{lines of the heptagrid}: they are the lines which joins mid-points of sides meeting at a vertex. On that figure, we
especially drawn the lines which delimit a sector. Remember that, in Poincaré’s
disc model, lines are represented by traces in the disc of diameters or circles
which are orthogonal to the border of the disc. A particular tree plays a key
role. It generates the tiles which are inscribed in a sector of the hyperbolic
plane which is delimited by two lines as previously defined. There are seven
such sectors around the central tile of Figures 6.

Figure 6  To left: the heptagrid; to right: showing the sectors around a central tile.

2.5 Basic principles

Presently, we turn to the implementation of the railway circuit in the heptagrid.
Picture 7 illustrates the way we implement a register. The leftmost picture
indicates a way which can be used to implement lines with tiles. However, such
a line of tiles in that way requires a two-celled locomotive in order to define the
direction of its motion. A possible solution would be to reinforce the line as
indicated on the central picture of the figure. However, we need another way
for the return of the locomotive to the instructions once it had performed the
operation involved by the just executed instruction. The solution is given by
the rightmost picture. The blue line defines the content of the register. The
tiles below the line, in yellow on the picture, are used for the execution of the
instruction. The tiles above the line, in orange on the picture, are devoted to
the return of the locomotive.
Our first focus will be the implementation of the tracks. Without tracks, there is no way to convey information between the parts involved in the simulation of the register machine. This is why any project for constructing a universal cellular automaton must first solve the question of the circuit implementing the circulation of information.

At this point, we have to remind that we are looking after a cellular automaton whose rules are rotation invariant: the new state of a rule is the same if we perform a circular permutation on the states of the neighbours. It appeared, during my search of a solution, that using lines as described by the central picture of Figure 7 a confusion could occur with rules in another situation. This is why I had to devise another solution for the tracks which consists in taking arcs of 'circles'.

In the tiling, a path between the tiles $A$ and $B$ is a finite sequence $\{T_i\}_{i \in [0..n]}$ of tiles such that $T_0 = A$, $T_n = B$ and $T_i$ shares a side with $T_{i+1}$ for all $i$ with $0 \leq i < n$. In such a situation, $n$ is called the length of the path. In the tiling, the distance between two tiles $A$ and $B$ is the length of a shortest path between $A$ and $B$. We call circle in the tiling, a set of tiles whose distance to a fixed tile, called its centre is constant. It will turn out that the rules for the motion of the locomotive along an arc of a circle are rotation invariant, keeping this property with the rules we shall construct for the other structures.

Among the details we have now to look at, the first task in the implementation deals with the tracks. The main reason is that without tracks, our locomotive can go nowhere, so that there cannot be any simulation. The second reason is that it may happen that we can construct the switches of the circuit with say $k$ states but that we need a $k+1^{th}$ one to build the tracks. Subsection 2.6 is devoted to that problem. Once this problem is temporarily solved, we shall see the main features of the crossings and of each kind of switches in Sub-sections 3.1, 3.1.1, 4.3 and 3.2.

### 2.6 The tracks

As already mentioned, the tracks are implemented by following arcs of circles. Such tracks are illustrated by Figure 8. As can be seen on the figure, we define a track by a first arc at distance $k$ from a tile which plays the role of the centre of the circle supporting the arc and there is a second arc at distance $k+1$. As
shown also by the figure, on the left-hand side picture there are blue tiles and mauve ones. On the right-hand side picture, the mauve tiles are in contact with orange ones. Those two colours indicate the direction of the motion performed by the locomotive. The locomotive is a single-tiled element. The blue tiles support a track on which the motion is counter-clockwise while the motion is clockwise on a track supported by orange tiles.

![Figure 8](image1.png)

**Figure 8** Implementation of tracks in the heptagrid. On the left-hand side, for a counter-clockwise motion along the arc. On the right-hand side, for a clockwise motion. The yellow tile is the centre of the circles supporting the arcs. The blue and orange tiles are at distance 4 from the centre. The mauve tiles are at distance 5.

![Figure 9](image2.png)

**Figure 9** To left: the notion of a zig-zag line in green following a straight line, in purple. To right: its implementation in the heptagrid.

On the further figures of this section, we shall represent tracks by arcs of circles. Blue arcs are counter-clockwise run, while red arcs are run in a clockwise motion. This convention allows us to indicate the directions without using arrows. As we shall have some discussion about delays, we need a which allow us to tune the length of the path covered by the locomotive. Note that if the radius of an arc \( A_0 \) is \( n \) and its angle is \( \alpha \), the length of the arc of the same angle and whose radius is \( n+1 \) is more than twice the length of \( A_0 \). The shortest path from a tile to another one is not an arc of a circle, especially if the distance is big. An approximation of that shortest path can be realized by a sequence
of arcs which we call a zig-zag line illustrated by Figure 9. The length of the shortest path following the line is multiplied by 4 in the zig-zag line illustrated on the right-hand side picture of Figure 9.

3 The scenario of the simulation

With only three states at my disposal, I do not know how to directly implement switches where the needed sensors and controls are immediate neighbours of the cell where the tree tracks arriving at the switch meet. The simplest solution for that purpose is to devote a state for the tracks, different from the blank and then to give a series of states for each type of switch and for each step required for the crossing of the switch by the locomotive. This may require more states than the 22 states of the automaton in [1].

Here, we again take the idea of a special colour for the tracks, the mauve one. Now, we take a single-celled locomotive, we shall say a simple one. As we need to use opposite directions, we have two different colours for the support of the track as mentioned in the caption of Figure 8. Note that the support is already needed for the motion of the locomotive in order to observe the rotation invariance of the rules. Also, we indicated that there are instructions which decrement a register and others which increment it. We have a single track on the register in order to perform the instruction and a single one to allow the locomotive to return to the instructions of the machine. The distinction is obtained by defining two colours. The locomotive is green when it has to increment a register and it is red when it has to decrement it. The colours for the supports of the tracks are blue and orange. Accordingly we shall speak of a blue and an orange path.

Here too, the crossing is performed by a most expensive structure, the round-about described in Sub-section 3.1. Switches are decomposed into simpler structures we shall indicate later. Although we follow the mainlines of [7], we use here specific structures. All structures described in the present section are under the idle configuration which is their configuration at the starting time of the computation.

3.1 The round-about

The round-about replaces the crossing, a railway structure, by a structure inspired by road traffic. At a round-bout where two roads are crossing, if you want to keep the direction arriving at the round-about, you need to leave the round-about at the second road. Figure 10 illustrates this features.

As mentioned in the legend of the figure, when the locomotive arrives through B, A, it leaves the round-about through gate 2, 3 respectively. At A and B, we have a special structure which we call the doubler which transforms the simple locomotive, into a double one which constitute of two contiguous cells of the same colour. In the points 1, 2 and 3, the structure, which we call the selector,
sends the locomotive further if it is a double one and sends it on the appropriate track if it is a single one.

Figure 10 Implementation scheme for the round-about. A locomotive arriving from B leaves the round-about through 2. If it arrives from A, it leaves through 3. Note the fixed switch f in between the structures 1 and 2 allowing the track coming from A to reach the round-about.

The auxiliary structures we need are the fixed switch, see Sub-section 3.1.1, the fork, see Sub-section 3.1.2, the doubler, see Sub-section 3.1.3 and the selector, see Sub-section 3.1.4.

3.1.1 The fixed switch

As the tracks are one-way and as an active fixed switch always sends the locomotive in the same direction, there is no need of the other direction: there is no active fixed switch. Now, passive fixed switch are still needed as just seen in the previous paragraph. Figure 11 illustrates the passive fixed switch when there is no locomotive around. We can see that it consists of elements of the tracks which are simply assembled in the appropriate way in order to drive the locomotive to the bottom direction in the picture, whatever upper side the locomotive arrived at the switch. From our description of the working of the round-about, a passive fixed switch must be crossed by a double locomotive as well as a simple one. Also, it must be crossed by such locomotives whatever their colour.

Later, in Section 4 we shall check that the structure illustrated by Figure 11 allows these crossings.
3.1.2 The fork

The fork is a structure which allows us to get two simple locomotives from a simple one. It also allows us to get two double locomotives from a double one. The structure of the structure is illustrated by Figure 12. As shown there, two arcs, one blue, the other red, abut a same tile to which a third one arrives. The arriving locomotive is duplicated on the orange and the blue paths.

Figure 12 The fork in the heptagrid.

The fork is used later in several structures. We shall see it, namely, in the doubler to which we turn now in Sub-subsection 3.1.3. We can see on Figure 12
that the exiting paths, the red one and the blue one, have supporting tiles which belong to the same circle.

3.1.3 The doubler

The doubler is illustrated by Figure 13.

The left-hand side picture of the figure explains the structure of the doubler. The picture shows us a circle whose radius is 3. The supported track lies on a circle of radius 4.

Figure 13 To left: explanation of the implementation. To right: the doubler.

On a circle of radius 4, there are $7 \times 21 = 147$ tiles. Indeed, it was proved that within a sector, from one border line to the other, the number of tiles at the distance $k$ is $f_{2k-1}$ where $f_n$ is the Fibonacci sequence defined by the initial condition $f_0 = f_1 = 1$, see [3, 2]. Accordingly, an arc ending on the borders of a sector with radius 4 contains 21 tiles exactly. It can be checked in [3]. As the number of tiles is odd and as the duplication occurs at a tile $T$ of the circle, we remain with an even number of tiles so that each path on each side contains the same number of tiles, namely 73 of them. That same number of tiles means that both simple locomotives created at the same time $t$ at both neighbours of $T$ on the circle arrive at neighbouring tiles $U$ and $V$ so that, starting from time $t$, they constitute a double locomotive. The exit tracks allows that double locomotive to go further on its track.

That argument explains why the radius is important. The number of tiles on the circle which supports the tracks should be odd and the smallest case avoiding complications is radius 4. This is why the left-hand side picture of Figure 13 explains the right-hand side picture.
3.1.4 The selector

In Figure 14, we have an illustration of the structures involved in the selector. There are three structures: a fork at $F_k$, a first controlling device at $S_I$ and a second controlling device at $D_l$.

On $F_k$, the fork which replicates the arriving locomotive on both paths exiting from that point. If the locomotive is simple, its copy on the blue path goes out through $S_I$ which let it got. However, on the orange path, the other copy does not cross $D_l$ which stops a simple locomotive. If the locomotive is double, its copy on the orange path goes out through $D_l$ which let it go. The other copy on the blue path does not cross $S_I$ which stops a double locomotive.

![Diagram of the selector](image)

**Figure 14** Scheme of the selector. Above, the fork. To left and to right the controlling devices.

On Figure 15, we indicate the implementation of the control structures of the selector. On the left-hand side of the figure, we have the controlling device: it let pass the appropriate locomotive. We note that both controllers have a similar structure and that the converter is quite different.

According to what we said in the description of a round-about, the red path after $S_I$ leads to the tracks which continue the track through which the locomotive arrived at the round-about. On the other side of the scheme, the blue path after $D_l$ leads to the tracks of the round-about which goes to the next selector. However, after $D_l$, we can see another structure on the tracks denoted by $Cds$. It transforms the double locomotive into a simple one, so that at the next selector, the simple locomotive of the right colour is sent on the appropriate tracks. Note that the controller structures must be placed on a red path.
3.2 Flip-flop and memory switch: the use of controlling structures

In this Sub-section, we look at the decomposition of two active switches: the flip-flop and the active part of the memory switch. In both cases, we can split the working of the switch by separating at two different stages, the bifurcation and the fact that the passage in one direction is forbidden. It is not mandatory that both acts should happen simultaneously, as might be suggested by the definition of Sub-section 2.1. A fork may dispatch two simple locomotives and, later, on one track, a controller let the locomotive go on its way on the track and, on the other track, another controller stops the locomotive and destroys it.

The difference between the switches is the way in which the change of selection is performed. Figure 16 illustrates how to assemble forks and controllers in order to obtain either a flip-flop, left-hand side of the figure, or an active memory switch, right-hand side of the picture. The difference is important: the flip-flop immediately changes the selection once it was crossed by the locomotive. The active memory switch changes the selection if and only if the passive switch ordered it to do so through a signal sent to the active switch. This is realized in Figure 16.

Indeed, in the active memory switch, when the simple locomotive arrives at the fork $C$, it is duplicated into two simple locomotives, each one following its own path. One of these locomotives goes on its way to the controller, and the other is now a third locomotive which is sent to a second fork, labelled with $A$ in the figure. When it is crossed by that locomotive, $A$ sends two locomotives: one to the other controller and one to a third fork $S$. The controllers are blue and red. The blue one let the locomotive go its way. The red controller stops the locomotive and kills it. The locomotives sent by $S$ go both to a controller, one to the blue controller, the other to the mauve one. Now, when a locomotive sent by $S$ arrives at a controller, it changes the blue one into a mauve one and the mauve one into a blue one, so that what should be performed by a flip-flop
is indeed performed. It is enough to manage things in such a way that the locomotives arriving to the controllers from $S$ arrive later than those sent by $C$ and by $A$.

Figure 16 To left: the flip-flop. To right: the active memory switch.

Figure 17 illustrates the working of the memory switch. As in [8], the memory switch is split into two parts: the active one for an active crossing and a passive one for the passive crossing of the switch. Consider an active crossing. As shown on the left-hand side part of Figure 17 the locomotive arriving to $C$ is duplicated by a fork and each copy of the locomotive is sent to a controller: one of them is $L$, the other is $R$. One of them let the locomotive go, whatever its colour, the other controller stops the locomotive by killing it.

At present, consider a passive crossing, illustrated by the right-hand side picture of Figure 17. The locomotive arrives either to $P$ or to $Q$. Assume that it arrives to $Q$. A fork at $Q$ sends a copy of the locomotive to $F$ which is a fixed switch. The fork sends the other copy to $R$, a controller. If the corresponding track on the active switch can be crossed, the controller stops that copy of the locomotive. If the corresponding track is barred on the active switch, the controller let the copy of the locomotive go. The copy goes then to $S$ and then to $U$. There, a fork sends a copy of the locomotive to from which the $S$ of the active switch which is also a fork. The other locomotive sent from $U$ goes to $T$ where another fork sends to copies to $L$ and $R$ in order to change the function of those controllers which will thus be in accordance with those of the active switch which will be changed accordingly when the locomotive sent from $U$ will reach them. The fork at $P$ and $L$ play symmetrical roles when the locomotive arrives to the passive switch from the other side. Of course, the controllers at $L$ and $R$ play opposite roles at any given time.

We can see on Figure 18 that an auxiliary track abut the structure without crossing it. It is the end of the paths taken by the locomotive-signal sent from $U$ both in the passive and in the active switches. Those signals change the controller. If the signal arrives at a blue controller, it becomes a mauve one, if the signal arrives at a mauve controller, it becomes blue. The blue controller let the locomotive go while the mauve one stops it. The controllers at the active
switch are the same structures as those indicate in Figure 18. Simply, they are fixed on opposite functions: on the active switch, the blue controller is on the track to which an arriving locomotive at C can go, see the left-hand side picture of Figure 17, and the mauve controller is on the track whose access is forbidden. On the passive switch, the mauve, blue controller is set on the track which corresponds to the track of the active switch on which the blue, mauve controller lies respectively.

![Image](image-url)

**Figure 17** Memory switch: to left, the active switch; to right the passive one.

![Image](image-url)

**Figure 18** To left: the controller which let the locomotive go as a signal to the active switch. To right, the controller which stops the locomotive when there is no change to perform.

Note that the controlling structures of Figure 18 have a common point with those of Figure 15. The structures of Figure 18 are programmable versions of those of Figure 15.

### 3.3 Register and instructions

Thanks to the tracks and the connected structures, switches and control devices, the locomotive can go from the instructions to the register and then, from there
back to the instructions.

The structures we considered up to now allow us to revisit the schemes we indicated in Section 2. We start with the one-bit memory which is the content of Figure 2. In Figure 19, we remind that latter scheme and we present how we implement it with the structures defined up to now.

**Figure 19** To left, the scheme of Figure 2; to right, its implementation in our setting. In the hyperbolic picture, the colours 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 green points with an x letter indicate crossings.

In both figures, the content of the memory is defined by the position of the switches at W, at R and at E. There are five gates in both structures: W, R, E, 0 and 1 in the right-hand side picture of the figure. The main difference with the left-hand side picture is that in that latter case, some portions of the tracks are two-ways: the tracks between R and the fixed switch in between R and E1 on one hand and in between R and E2 on the other hand can be crossed in both directions by the locomotive. In the setting of the present paper, tracks are one-way. The direction is indicated by the colour of the support. In our schemes, red tracks are clockwise run while blue ones are counter-clockwise run. In both pictures, there are two possible entries: R and W. Entering through R does not change the structure. In the left-hand side scheme, a memory switch stands at R. In the right-hand side picture, at R, we have an active memory switch. If the position of the switch at R is 0, the exit is through gate 0, E0 in the right-, left-hand side picture respectively. If the position is 1, the exit is through gate 1, E1 respectively. If the locomotive enters the structure through gate W, as a flip-flop switch is placed at that point, the position of the switch is changed by the passage of the locomotive. Then, the locomotive exits through gate E after it has passively crossed the switch at E, R in the right-, left-hand side picture respectively. This also changes the selected track in that switch. Accordingly, if the locomotive enters the structure through W, the positions
of the switches are changed to the opposite position. It is also the case in the right-hand side picture for the switch at $R$ as its selected track is defined by the track through which the locomotive passed at $E$. On the right-hand side picture of the figure, we can see a track going from $E$ to $R$ which represent the structures we have seen for the memory switch in Sub section 3.2 allowing the passive switch to change the position at the active switch when it is required.

In later Figures, we represent the one-bit memory unit as a circle with five gates, $W$, $R$, $E$, 0 and 1, two gates for entrance, $R$ and $W$, and three exits, $E$, 0 and 1.

![Figure 20](image)

**Figure 20** A one-bit memory to remember whether the last instruction was to decrement or to increment the register which was just operated. The conventions for colours are those of Figure 19.

Figure 20 illustrates a structure belonging to a register which allows the locomotive to return to another memory structure, that for the incrementing instructions and that for the decrementing ones. The structure is based on a one-bit memory: 0 if the instruction incremented the register, 1 if it decremented it. The idle configuration is 0. When the locomotive goes to increment a register, it does not visit the structure which remains 0. When the locomotive goes to decrement a register, it enters the structure through its $W$-gate so that when it leaves it through the $E$-gate, it goes to the register. When it goes back from the register, it enters the structure through its $R$-gate: if it reads 0, it is sent to the memory structure for incrementing instructions which will send back the locomotive to the right instruction. If the locomotive reads 1 it leaves the structure through its 1-gate and it is sent to the $W$-gate of the structure, resetting it to 0. Now, the locomotive is not sent again to the register. On its way to decrement the register after crossing the structure, the locomotive crossed a flip-flop $ff$ whose 0-branch sent it to the register. And so, when it again meets the flip-flop which is now indicating its 1-branch, the locomotive is sent to the memory structure of the decrementing instructions and the flip-flop
again indicates its 0-branch which is its idle configuration.

**Figure 21** The unit in the memory structure of a register for the return from a decrementing operation to the right place in the instructions. The conventions for colours are those of Figure 19 and of Figure 20.

Figure 21 illustrates a unit of the memory structure which ensures a returning locomotive which decremented a register to go back to the appropriate place among the instructions. Such a memory structure contains as many units as there are instructions in the program which decrement that register. As we can see, the unit contains two one-bit memories denoted by D and Z in the program, so that we shall denote their W-, R, E, 0- and 1-gates by W_D, W_Z, R_D, R_Z, E_D, E_Z, B0_D, B0_Z, and B1_D, B1_Z respectively. The tracks which connect E_D to W_Z shows us that both D and Z are simultaneously either 0 or 1. The reason why we need two connected one-bit memory is that when the locomotive reaches the register in order to decrement it, it may happen that the operation cannot be performed because the register is empty. In that case at which we look a bit further, the locomotive is sent to the memory structure on track D if the register was actually decremented and on another track, say Z, if the register was empty so that the locomotive could not decrement it. The distinction is important as the next instruction depends on which the case was. When it goes to decrement a register, the locomotive first goes to the memory structure. It enters the D memory through W_D. Note that the locomotive enters into one
unit only so that all other units remain 0. As it enters the unit through \(W_D\) we know that both \(D\) and \(Z\) become 1. Next, from \(E_Z\), the locomotive goes to \(F\) at which a flip-flop switch stands which indicates its 0-branch. That branch is a track leading to the register.

When the locomotive succeeded to decrement the register, it goes back to the memory along the \(D\)-track. The track enters a unit through \(R_D\). If the locomotive reads 0 it is sent through \(B0_D\) to the \(R_D\) of the next unit. So that it goes from one unit to the next one until it reads 1 in the \(D\) memory of a unit. A similar motion happens for the locomotive if it arrives to the memory through the \(Z\)-track: it enters the unit through \(R_Z\) and, if it reads 0, it is sent through \(B0_Z\) to the \(R_Z\) of the next unit.

Consider the case when the locomotive arrives to a unit containing 1. First, let us look the case when the locomotive arrives to \(R_D\). As it read 1, it goes through \(B1_D\) which sends it to \(P\) at which a passive memory switch stands. It arrives to the switch through a track named \(d\). Accordingly, another locomotive is sent through \(P\) which reaches \(A\) at which an active memory switch stands. The locomotive make the active switch indicate its branch also named \(d\). From \(P\) the former locomotive is sent to \(W_D\) so that it will set \(D\) and \(Z\) to 0. Now, from \(E_Z\) the locomotive passes a new time through \(F\) so, that it takes the 1-branch of the flip-flop switch which stands there and makes the flip-flop to indicate its 0-branch again. On the 1-branch, the locomotive goes from \(F\) to \(A\) where the active memory switch indicates the \(d\)-track, leading the locomotive to the right place in the program.

Now, consider the case when the locomotive reads 1 in the unit after it arrived through \(R_Z\). Then it goes out through \(B1_Z\) which sends it to \(P\) at which a passive memory switch stands. It arrives to the switch through a track named \(z\). The memory switch at \(P\) sends another locomotive to \(A\) as seen previously but this time the flip-flop switch at \(A\) is made to indicate its \(z\) branch. Meanwhile, the former locomotive goes from \(P\) to \(W_D\) so that it resets both memories \(D\) and \(Z\) to 0. That locomotive also goes through \(E_Z\) which sends it to \(F\). The flip-flop switch sitting there indicates its 1-branch so that the locomotive goes to \(A\) and the switch again indicates its 0-branch. At \(A\) the locomotive is sent through the track named \(z\) to the right place in the instruction part of our implementation.

On the same line, we have to make a bit more precise the memory structure used for defining the right place in the program for the return of a locomotive after it has incremented a register. There is such a structure for each register. Each memory structure consists of as many units as there are instructions which increment the considered register. Figure 22 illustrates such a unit. It makes use of a one bit memory. In the idle configuration, each unit contains 0. When the locomotive arrives from a given instruction which increments the register, it enters the appropriate unit of the memory structure of that register through the \(E\)-gate. Accordingly, from now on, that unit contains 1. As it exits through the \(E\)-gate, the locomotive arrives at \(A\) to a flip-flop switch which sends it on its 0-branch leading to the register, and the flip-flop switch indicates its 1-branch. When the locomotive returns from the register, it arrives to a unit through a track which leads to the \(R\)-gate. If the locomotive reads 1 there, it is sent
through B0 to the R-gate of the next unit. Accordingly the locomotive arrives to the unique unit containing 1. As it reads 1, it leaves the one-bit memory through B1 which sends the locomotive to the W-gate through a fixed switch. Accordingly, the locomotive changes the one-bit memory to 0 and it leaves that memory through the E-gate. Accordingly, it is sent again to A. The flip-flop switch which sits there, sends the locomotive to its 1-branch leading to the right instruction of the program and the flip-flop switch again indicates its 0-branch, the direction to the registers.

Figure 22 The use of a one-bit memory in a unit for selecting the right place for an incrementing instruction on its returning way. The conventions for the switches and the crossings are those of Figure 19

It is time to look closer at what happens for a returning locomotive to the instructions. When it comes back from a register, the locomotive is red. If it arrives to an incrementing instruction or to a jump instruction, it must become green. If the jump instruction arrives to a decrementing instruction, the locomotive must become red. Accordingly, we need a structure which allows to convert a green locomotive to a red one and conversely. It is illustrated by Figure 23 on the left-hand side picture. The right-hand side picture reminds us the converter of a double locomotive to a simple one.

Next, Figure 24 shows us different configurations of a register. The rightmost picture of the figure illustrates a standard configuration, when some positive number is stored in the register. The leftmost picture illustrates an empty register, which means that its content is 0. The middle picture shows a register which contains 1 exactly. On the rightmost picture, we can see the structure of the register: the number stored in the structure is the number of blue central cells. As already mentioned, the yellow cells allows the locomotive to arrive at the end of the stored number in order to append a new blue cell or in order to erase the last one. The process of appending or of erasing has also to deal with the yellow and orange cells which sit around the blue part. A green
A red locomotive performs its action as a green cell which becomes a red locomotive again when the action is performed.

![Diagram of converters for the locomotive. The left side shows the converter from green to red and from red to green colours. The right side shows the converter from double to simple forms.](image)

Figure 23 Converters for the locomotive. To left: the converter from green to red and from red to green colours. To right, the converter from double to simple forms.

We conclude the subsection devoted to registers by showing two schemes on Figure 25. To left, we have the scheme of zero. As clear from Table 11, when the red locomotive checks that the register is empty, it sends two locomotive on the circuit. One locomotive takes the path to the structure which stand to the right of the figure, a structure which distinguish between instructions which incremented and those which decremented. The second locomotive goes to the instruction to be executed after a zero test. In the case when the locomotive realises that the register was empty, the first locomotive must be stopped. The scheme of that action is illustrated by the right-hand side picture of Figure 25.

![Diagram of registers. The left side shows the register when it is zero. The middle side shows the register when it contains 1 exactly. The right side shows the register when it has some number.](image)

Figure 24 The register: to left, when it is zero; in the middle, when it contains 1 exactly; to right, when it has some number.

On the way to the further treatment of the locomotive returning from zero, call it the zero-locomotive, that locomotive sends at \( I \) a first copy to a controller which will stop the locomotive returning through the standard way, say the nonzero-locomotive. On its way to the controller, the first copy sends a second one to the controller in order to let the next locomotive go. The paths are computed in such a way that the non-zero locomotive arrives to the controller in
between the arrival of the copies sent by the zero-locomotive. Indeed, increasing the radius of a circle by one multiply the circumference by at least two, so that it is easy to manage the circuit in order to obtain the expected action.

![Figure 25](image)

**Figure 25** To left: the test of zero for a register. To right, the discrimination between instructions which increment and those which decrement when the locomotive returns to the instructions. That structure uses the one-bit memory.

On the right-hand side picture, we can see the implementation of the structure which allows an instruction returning from a register to go back to the appropriate next instruction. It makes use of a one-bit memory, see the middle picture of the figure, which was set to 1 if a red locomotive crosses it on its way to the register. A green locomotive does not cross that unit, in that case, remains set to 0. The returning locomotive reads the memory. If it is 0, it know it goes to the selector of incrementing instructions in order to find out the appropriate next instruction. If the memory is set to 1, the locomotive crosses it again which automatically sets the unit to 0, next it goes to the selector of the decrementing instructions. The locomotive is set again to the unit through a flip-flop switch which was already crossed on the way to the register, so that the second crossing sets back the switch to the right position. Note that a zero-locomotive goes directly to the selector of decrementing instructions in order to go to the correct next instruction.

### 4 Rules and figures

The figures of Sections 2 and 3 help us to establish the rules. The rules and the figures were established with the help of a computer program which checked the rotation invariance of the rules and which wrote the PostScript files of the pictures from the computation of the application of the rules to the configurations of the various type of parts of the circuit. The computer program also established the traces of execution which allow the reader to check the application of the rules.
In comments of the figures, the central tile is denoted by 0(0). The cells which are neighbours of 0(0) by sharing an edge with that cell are numbered 1(i), with \( i \in \{1..7\} \), increasingly while counter-clockwise turning around 0(0), see the left-hand side part of Figure 6. In each sector delimited on that picture, the cell 1(i) has three sons numbered 2(i), 3(i) and 4(i), blue, green and yellow respectively in the right-hand side part of Figure 6 where the cell 1(i) is the closest green tile to the central tile. The cell 2(i) has two sons: the cells 5(i) and 6(i), blue and green respectively. The cell 3(i) has three sons numbered 7(i), 8(i) and 9(i), blue, green and yellow respectively on the picture while the cell 4(i) has also three sons numbered 10(i), 11(i) and 12(i), also blue, green and yellow respectively on the figure. For further numbers, the reader is invited to look at [3].

Later, we provide the reader with tables of rules and figures in following the various structures of the implementation, starting with the tracks, see Sub-section 4.1. Before turning to the study, let us mention notations and important properties for a better understanding of the rules and their application.

First, we establish the format we use to denote the rules. A rule has the form \( w_0w_1..w_7w_n \) where \( w_0 \) is the current state of the cell \( c \), \( w_i \) is the current state of the neighbour \( i \) of \( c \) and \( w_n \) is the next state of \( c \) once the rule has been applied. The rules must be rotation invariant which means that the rule \( w_0w_1..w_7w_n \) and the rule \( w_{\pi(1)}w_{\pi(2)}..w_{\pi(7)}w_n \) are the same for any circular permutation \( \pi \) on the set \([1..7]\). The cellular automaton have seven states, \( W, B, R, Y, G, O \) and \( M \) in that order. The states \( G \) and \( R \) are essentially used by the locomotives. Consider the rules \( w_0w_{\pi(1)}..w_{\pi(7)}w_n \) with \( \pi \) running over the seven circular permutations on \([1..7]\). We can order these rules according to the order we defined on the states as a rule can be read as a word. In the tables, we shall take the rule using the circular permutation which provides the smallest word.

There are two types of rules. Those which keep the structure invariant when the locomotive is far from them, and those which control the motion of the locomotive. We call conservative the rules of the former type and motion rules those of the latter one. Motion rules are applied to the cells of the tracks: the one on which the locomotive stand and also the neighbours which will be occupied by the locomotive at the next time. Among the conservative rules, there are rules in which the state of the cell does not change but the states of at least one neighbour is affected by the motion of the locomotive. Those rules are called witness rules. A cell to which a witness rule applies has at least one neighbour on which the locomotive may stand. The largest set of contiguous neighbours of a cell on which the locomotive may stand on each of them is called a window on the motion of the locomotive. As an example, such cells occur in the supports of the tracks, see Figure 8. As an example, on a blue path, many cells are applied the conservative rule \( BWMWMMMMBB \). The contiguous four \( M \) constitute a window, each cell of which is step after step occupied by the locomotive, giving rise to four conservative rules. We shall mention the witness rules among the conservative ones, but we shall not provide all the rules, indicating the window only as follow, taking the previous example: \( BWMWMMMMBB \). That notation replaces the rules \( BWLMWMBB, BWMLMMBB, BWMMMLMBB \) and \( BWMMMMMBB \) as well as the rules when
a double locomotive is passing through the window: \texttt{BWB:LM:BB} and \texttt{BWB:LM:BB}. In the previous rules, we used symbol \( L \) to replace \( G \) and \( R \). Note that in the case of several occurrences of \( L \) in a rule, all of them have to be replaced either by \( G \) or by \( R \). That notation will be used in the sequel in order to spare space. However, a single-symboled window will not use that notation. In the tables for the rules, in order to avoid fastidious repetitions, we shall use the notation displayed in \texttt{BW:B:MB:BB}. In those cases, the successive symbols of the window can be replaced by \( G \) or by \( R \), as we can do when \( L \) has to be replaced by those symbols in the seven previous rules.

We can do the same remarks about similar rules when green or red, simple or double locomotives run on the blue path and on the orange path. We leave that to the reader as an exercise.

### 4.1 Rules and figures for the tracks

Table 1 provides us conservative rules which are applied to the white cells which remain white in the computation and for cells which are completely surrounded by white cells.

| Table 1 | Conservative rules for white cells. |
|---------|-------------------------------------|
| 1 \texttt{WWWWWWWW} | 6 \texttt{WWWWWWG} | 11 \texttt{WWWWWWG} |
| 2 \texttt{BWWWWWWWB} | 7 \texttt{WWWWWWG} | 12 \texttt{WWWWWWG} |
| 3 \texttt{RWWWWWWWR} | 8 \texttt{WWWWWWG} | 13 \texttt{WWWWWWG} |
| 4 \texttt{YWWWWWWWY} | 9 \texttt{WWWWWWG} | |
| 5 \texttt{GWWWWWWWG} | 10 \texttt{WWWWWWG} | |

Table 2 gives the rules for ordinary tracks. By that expression, we mean mean tracks which are supported by a circle. Later, we look at rules which allow the locomotive to pass from an arc of a circle to that of another circle. As already mentioned, the simulation works on two kinds of tracks. We call \( B \)-, \( O \)-path a track whose support is an arc of a circle consisting of blue, orange cells respectively.

Figures 26 and 27 illustrate the application of the rules of Table 2 for \( B \)- and \( O \)-paths respectively. In the table, two rules are numbered by red digits. It means that the same rules will later be repeated with the same numbers. That convention will be used in the sequel.
Table 2  *Rules for ordinary tracks*

| Conservative rules | witness rules |
|--------------------|---------------|
| 14 WWWWBBMW | 18 WWWBBBW | 22 RBW:MMBM:BM |
| 15 WWWBMMB | 19 WWWBBBW | 23 RBWB:MMBM:BM |
| 16 WWWWOBMM | 20 WWWODOM | 24 OWO:MMOM:OM |
| 17 WWWMDOM | 21 WWWODOM | 25 OWO:MMOM:OM |

**Motion rules**

| B-path | O-path |
|--------|--------|
| 26 WWWWWWBL | 31 WWWWWWBL | 36 WWWWWWOL | 41 WWWWWWOL |
| 27 LWWWBBBM | 32 LWWWBBBM | 37 LWWWBBOM | 42 LWWWBBOM |
| 28 LWWWBBMM | 33 LWWWBBMM | 38 LWWWBWOL | 43 LWWWBWOL |
| 29 LWWWBBLL | 34 LWWWBBLL | 39 LWWWBLOL | 44 LWWWBLOL |
| 30 LWWWBLBM | 35 LWWWBLBM | 40 LWWWBLOM | 45 LWWWBLOM |

**Figure 26** Locomotives on a B-path. From top to bottom: simple green one, double green one, simple red one, double red one. On those pictures, the radius of the arc on which the tracks lie is 4.

As mentioned in the legend of the figures, the radius of the arc on which the mauve cells, those of the tracks, lie is 4. As the cells of the tracks have at least three white neighbours and at most two neighbours on their support, the identification of which rule is applied is easy.

The figures help us to see that, in Table 2, the conservative rules 14 to 17 concern the mauve cells of tracks, while the rules 18 to 21 concern the white cells which are inside the circle supporting the support of the tracks. The blue or orange cells of the support are applied one of the witness rules from 22 to 25. It is easy to check that the 45 meta-rules of Tables 1 and 2 represent 71 actual rules of the automaton.
Figure 27 Locomotives on a O-path. From top to bottom: simple green one, double green one, simple red one, double red one. On those pictures too, the radius of the arc on which the tracks lie is 4.

Table 3 Rules for the links between B- and O-paths.

| B → O | O → B | B → B | O → O |
|-------|-------|-------|-------|
| \( \text{conservative rules and witness rules} \) | \( \text{motion rules} \) |
| \( \text{conservative rules and witness rules} \) | \( \text{motion rules} \) |

| \( \text{conservative rules and witness rules} \) | \( \text{motion rules} \) |
|---|---|
| 46 \( \text{MMWBMWOOMM} \) | 52 \( \text{MMWBMWOOMM} \) | 59 \( \text{MMWBMWOOMM} \) | 66 \( \text{MMWBMWOOMM} \) |
| 47 \( \text{WB:MMMM:B} \) | 53 \( \text{WWWBBBBLL} \) | 60 \( \text{WWW:MMM:O} \) | 60 \( \text{WWW:MMM:O} \) |
| 48 \( \text{WWWAWMM} \) | 54 \( \text{WWWWWWOM} \) | 61 \( \text{WWW:MMM:O} \) | 68 \( \text{WWW:MMM:O} \) |
| 49 \( \text{WWWBB:MMM:O} \) | 55 \( \text{BBBB:MMMM:B} \) | 62 \( \text{WWW:MMM:O} \) | 69 \( \text{WWW:MMM:O} \) |
| 50 \( \text{WWWWWW:MMM:O} \) | 56 \( \text{WWW:MMM:BB} \) | 63 \( \text{WWW:MMM:BB} \) | 70 \( \text{WWW:MMM:BB} \) |
| 51 \( \text{WWWGGOOML} \) | 57 \( \text{WWW:MMM:BB} \) | 64 \( \text{WWW:MMM:BB} \) | 65 \( \text{WWW:MMM:BB} \) |
| \( \text{motion rules} \) | \( \text{motion rules} \) |
| 71 \( \text{MMWWWB:LL} \) | 76 \( \text{MMWWWB:LL} \) | 81 \( \text{MMWWWB:LL} \) | 86 \( \text{MMWWWB:LL} \) |
| 72 \( \text{LWWWB:LL} \) | 77 \( \text{LWWWB:LL} \) | 82 \( \text{LWWWB:LL} \) | 87 \( \text{LWWWB:LL} \) |
| 73 \( \text{MMWWWB:LL} \) | 78 \( \text{MMWWWB:LL} \) | 83 \( \text{MMWWWB:LL} \) | 88 \( \text{MMWWWB:LL} \) |
| 74 \( \text{LWWWB:LL} \) | 79 \( \text{LWWWB:LL} \) | 84 \( \text{LWWWB:LL} \) | 89 \( \text{LWWWB:LL} \) |
| 75 \( \text{LWWWB:LL} \) | 80 \( \text{LWWWB:LL} \) | 85 \( \text{LWWWB:LL} \) | 90 \( \text{LWWWB:LL} \) |

Now, as already noticed, the tracks are composed of arcs belonging to different circles with possibly different radiiuses. The link between an arc to another one is an important point which should not be forgotten. Table 3 provides us with the corresponding rules, conservative, witness and motion ones.

Figures 28, 29, 30 and 31 provide us with an illustration of how rules are applied to the cells of the different configurations.
As mentioned in the legends of the figures, the radiuses of the circle supporting the arcs may be different. Note that the link requires some adaptation between the arcs. The flexibility on the radiuses allow us to give the configurations the same picture around the mauve cell of the link as in the figures. Note that Figure 31 is different from Figure 30 not only by the change of colour but also by the occurrence of a mauve witness only present in Figure 31. That conditions is needed for compatibility of the rules.
Figure 30 Locomotives going from a B-path to another one. From top to bottom: simple green one, double green one, simple red one, double red one.

Figure 31 Locomotives going from a O-path to another one. From top to bottom: simple green one, double green one, simple red one, double red one.

We remain with the rules devoted to the zig-zag lines which we defined in Section 2. They are given by Table 4. Note that the rules for the O-path can be deduced from those for the B-paths. Also note that the connection between B, O-paths in the figures are a bit different from those of Figures 32. Indeed, the conservative rules 94 \text{WWBBWBMB} \text{M} and 59 \text{WWBBWBMB} \text{MB} together with the corresponding motion rules give all the possible connections between two B-paths as far as the situation \text{WWBBWBMB} \text{M} is excluded. A similar remark holds for O-paths.
Table 4 Rules for the zigzag-lines.

| conservative and witness rules | motion rules |
|---------------------------------|--------------|
| 91 WWWWMBW | 95 WWWWOOW | 99 WWWMBWLL | 104 WWWWOLL |
| 65 BWW:MMM::BB | 58 DWW:MMM::DD | 100 LWWWBBMM | 105 LWWWOWMM |
| 64 BWBB:MMM::B | 97 DWWD:MMM::D | 101 LWWWLBMM | 106 WWWLOWOMM |
| 94 WWWMBWM | 98 WWWWOOM | 102 LWWWBBL | 107 LWWWOWML |

Figure 32 Locomotives going along the epicycles of a zig-zag line.

4.2 Passive fixed switch

As noticed in Section 3 there is no active switch in our simulation as return paths are strongly separated from the direct ones. We need to separately check the crossing from both branches of the switch.
Table 5 provides us with the rules whose application is illustrated by Figures 33 and 34.

### Table 5  Rules for the passive fixed switch.

| conservative and witness rules |
|--------------------------------|
| 109 MWWGYOMMM 114 WWWWOMMW |
| 110 WWWBYYMP 60 WWWW:MMR:W |
| 111 MWWWYBMM 116 YW:MMW:YM |
| 112 MWWWKYYMM 117 YMB:MMW:YM |
| 113 MWWWYKYMM 118 OWY:MMW:OM |

### motion rules:

| from the left: | from the right: |
|----------------|-----------------|
| 119 MYMOLYM 124 WWWWYBLL 129 WWWWYLLL 134 WWWOMYLL 139 WWWOMYML | 142 WWWOMYML |
| 120 LWMOYMOL 125 LWWWWYBLL 130 LWWWWYLLL 135 LWWWWOMYL | 120 LWMOYMOL |
| 121 LWMOYMOL 126 LWWWWLYBLL 131 LWWWWLYLL 136 LWWWWOLYLM | 121 LWMOYMOL |
| 122 LWMOYOMOL 127 LWWWWYBLL 132 LWWWWYLLL 137 LWWWWOMYLM | 142 LWMOYMOL |
| 123 LWMOYOMOL 128 LWWWWLYBLL 133 LWWWWLYLL 138 LWWWWOLYLM | 123 LWMOYMOL |

#### Figure 33  Locomotives crossing a passive fixed switch through its left-hand side track. From top to bottom: simple green one, double green one, simple red one, double red one. Note the yellow cell which separates the B- and the O-paths.

Note that the rules allow both a simple and a double locomotive to cross the switch. From Figure 10, we know that from A a double locomotive passively crosses the fixed switch through its left-hand side branch while if the locomotive comes to the round-about from B, a double locomotive does the same through the right-hand side branch of the switch. Also, those locomotives may be either green or red.

The passive fixed switch occurs in the passive memory switch, in the one-bit memory and, consequently, in the discriminating structures which allow a
returning locomotive from a register to go back to the right next instruction.

![Image of locomotives crossing a switch](image)

**Figure 34** Locomotives crossing a passive fixed switch through its right-hand side track. From top to bottom: simple green one, double green one, simple red one, double red one. Note the yellow cell which separates the B- and the O-paths.

### 4.3 Fork and doubler

Presently, we deal with the rules managing the fork and the doubler. The rules are given by Table 6. Figure 35 illustrates the application of the rules managing the fork while Figure 36 illustrates the application of those managing the doubler.

| Table 6 | Rules for the fork and for the doubler. |
|---------|----------------------------------------|
| **fork** | **doubler** |
| 144     | **conservative and witness rules**      |
| 145     | WWWBOMBMM 60 WWW::MMM::B                 |
| 146     | WWWWBBBM 64 WWWB::MM::B                 |
|         | WWWOBBBBM 151 WWWO::MM::B               |
|         | WWWWBDDBB 152 WWWO::MM::B               |
|         | WWWWBBBM 156 WWWB::MM::B                |
|         | WWWWBBBM 157 WWWB::MM::B                |
| 151     | **motion rules**                         |
| 154     | wwwbmobll 166 wwwwboll                   |
| 155     | wwwwbombbl 167 wwwwbombbl                |
| 156     | wwwwbombbl 169 wwwwbombbl                |
|         | wwwwbombbl 170 wwwwbombbl                |
| 160     | wwwwbombbl 171 wwwwbombbl                |
| 172     | wwwwbombbl 172 wwwwbombbl                |
| 173     | wwwwbombbl 173 wwwwbombbl                |
| 174     | wwwwbombbl 174 wwwwbombbl                |
| Note that the fork can be crossed by a double locomotive: that happens when a double locomotive arrives to the selector in a round-about. Contrarily |
to that circumstance, the doubler receives only simple locomotives, however two of them at the same time. The synchronization explained in Sub-subsection 3.1.3 is illustrated on Figure 36 by the fact that the green locomotive is, on each side, at the same distance from the cell where the double locomotive is formed.

Figure 35 Locomotives crossing a fork. From top to bottom: simple green one, double green one, simple red one, double red one.

Figure 36 Locomotives crossing a doubler. Top row: a simple green locomotive becomes a double one. Bottom row: a simple red locomotive becomes a double one.

4.4 Selector

Here, we examine the rules managing the crossing of the selectors. As explained in Sub-subsection 3.1.4, a fork distributes the arriving locomotive into two copies. The latter ones are sent to controllers which work on opposite ways. Both of them are fixed. One selector let a simple locomotive go and kills a double locomotive. The other selector performs the opposite action: it let a double locomotive go and it kills a simple locomotive. Accordingly, a locomotive arriving to the fork leading to the controllers eventually goes on its way on a single track, as required in our scenario.
Table 7  Rules managing the controllers of the selector. We remind the reader that a meta-rule indicating a window replaces the rules with a locomotive at the different places it occupies, whether it is green, red, simple or double.

| blue controller | both controllers | mauve controller |
|-----------------|------------------|------------------|
| conservative and witness rules |
| 178 WWYBB:MM:W | 187 WMW:MM:OM   | 192 WYYYYYMBW   |
| 179 BYYYYBBB    | 46 WMW:MM:OM    | 193 WBHYYYWY    |
| 180 BRBBYYYYB   | 189 WWW:MM:BYW  | 194 YYYYWWY     |
| 181 YYYYWBBY    | 68 WWW:MM:DBW   | 195 YYYYWYY     |
| 182 YYYYYYBYY   | 191 WWY:WW:OM   | 196 YYYYYYW     |

| motion rules |
|--------------|
| 200 BWLLMBBYW| 204 MMLDOWM:71 MMBLWOM:72 216 BWLLMBMYY |
| 201 WWLLBBYB  | 206 LBMBWOM:72 LMBWOM:72   217 WWLLMBMY | |
| 202 WBBBELMLW | 206 MMBLWOM:73 MMBWODLW   218 BWLLMWWML |
| 203 WBBBMMMB  | 207 LWLDOOM:74 LWLWOM:75   219 WWBBMMLMB |
| 208 LWLWOM:75 LWMBWOM:75 LWMBWOM:75 198 WW:MM:BMYY |
| 214 MMLWOM:75 199 BWMMB:MM:BM |
| 215 LWWWWWMW |

Figure 37  Locomotives crossing the control of a selector when the control is open. From top to bottom: simple green one, simple red one, double green one, double red one. Note that the control which let a double locomotive go has not the same configuration than the one which let a simple locomotive go.

Note that in the table, the meta-rules are replaced by rules when the witness cell is changed by the passage of the locomotive, see rules 182 up to 185 for the...
blue controller and rules 198 up to 201 for the mauve controller. On the table, rule 186 shows us the importance of the order taken to choose the representing instruction among its rotated forms. It is different from the order we can see on rule 187. The reason is that $WL < WM$ whatever the value of $L$.

Figures 37 and 38 illustrate the application of the rules given by Table 7. We can check the key roles of the just mentioned rules together with rules 196 and 197. Figure 37 gathers the configurations when both controllers let the locomotive go: this is why on that figure, the top two rows show us the blue controller letting a simple locomotive go, whether it is green or red while the bottom two rows show us the mauve controller letting a double locomotive go.

![Figure 37](image1.png)

Figure 37 shows the opposite working of the controllers, when they kill the locomotive: a double one is killed by a blue controller, see the bottom two rows of the figure while a simple locomotive is killed a mauve controller, see the top two rows of the figure.

We conclude the present subsection with the rules allowing a double locomotive to be changed into a simple one. We remind the reader that Figure 15 indicates the need of such a structure as far as, according to what we just reminded the reader about the controllers of a selector, the controller which let a double locomotive go on its way does not change it. Table 8 gives the few rules which manage that structure together with the motion of the double locomotive
when it crosses it. Note that rule 213 is the key one which transforms the double locomotive into a simple one. Figure 39 shows us the application of the rules to the configuration which we have seen on the rightmost picture of Figure 15.

![Figure 39](image)

**Figure 39** Top row: a green double locomotive crosses the structure; bottom row: a red double locomotive does the same. In both cases, a simple locomotive with the same colour leaves the structure.

| Table 8 | Rules managing the conversion of a double locomotive into a simple one. |
|---------|-------------------------------------------------------------------|
|         | conservative and witness rules | motion rules |
| 222     | MWWWWWYBB               | 227 NWLNYYBL     |
| 223     | YWWWWWBM:MM:YY          | 228 LWLNYYBL     |
| 194     | WWYWWMMY                | 229 LWWWLYYBL    |
| 225     | WWMMYyEB                | 230 NWYYLNYL    |
| 226     | NWYyYyEB                | 231 LWYYLM     |

### 4.5 Controllers in the flip-flop and the memory switches

In the present sub-section, we consider the rules managing the controllers of the flip-flop switch and of the memory switch. Both switches use the same controllers but their place in the circuit with respect to the tracks they control is opposite. While the controllers of a selector are fixed, the controllers of those switches are in some sense programmable. The blue controller may become mauve and the mauve one may become blue. Also, those controllers can see a simple locomotive only. When a controller is blue, it lets the locomotive go, when it is mauve, it kills the locomotive. On Figure 40 we can see that the controllers under study are rather close to those we have studied for the selector. It is the reason why in Table 9 we do not reproduce a few conservative rules for yellow cells which have the same neighbourhood here and in that former case as, for instance rules 163 and 164.

In Table 9 the key rules are rules 229 and 230 on one side, changing the blue controller into a mauve one, and rules 233 and 234 on another side, changing the mauve controller into a blue one.
Table 9 *Rules which manage the change of colour in the controllers of the flip-flop switch and of the memory one.*

|   | 232 | 233 | 234 | 235 | 236 | 237 | 238 | 239 | 240 | 241 | 242 | 243 | 244 | 245 | 246 | 247 | 248 | 249 | 250 | 251 | 252 |
|---|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
|   | BWYYMBBB | BBBBBMMYYE | MNOYBBYM | WWWBYMBY | WWWYBMOM | MNOYMMYM | OWWYMM:DM | WLOYBBYL | WLOYBLYYY | LMOYBLYY | BBYYLBMB | BBBLLYYY | WWYYMM:BM | BBYYLM:BM | BBMLLYYB |

**blue to mauve**

**mauve to blue**

**conservative and witness rules**

**motion rules**

As the working of those controllers are the same as those of a selector, Table 9 does not remind us the rules associated with the motion of the locomotive on the ordinary tracks. The table gives the rules connected with the change of the colour only. First the conservative and witness rules which are specific to those controllers. Then, it gives the rules for the motion of the signal-locomotive and also the rules managing the change of colour of the controllers.

Figure 40 illustrates the working of the controllers. Figure 41 illustrates how

---

Figure 40 *Locomotives crossing the control of a passive memory switch when it let the locomotives go. From top to bottom: simple green one, simple red one, double green one, double red one. Note that the control works as that of the selector. However, its decorations are not the same and a path is getting out from the controller. See the use of that path on Figure 41.*
the change of colour is performed.

Figure 41 Signal-locomotives arriving at the control of a passive memory switch. From top to bottom: green signal to the blue control, red one to the same control, green signal to the mauve control, red one to that control. Compare with the working of the control on Figure 40.

4.6 Registers

We give the rules for the registers in the present sub-section. We distinguish three main cases: when a locomotive arrives and performs an instruction, the case when the arriving locomotive finds out an empty register, the case when the register contains value 1 and the locomotive arrives to perform an instruction. As we have many rules, we split the table into two ones: a small table, Table 10, for the conservative and for the witness rules, a much bigger one, Table 11, for the motion rules.

Table 10 Conservative and witness rules for a register.

| 253 | WWWWWYYW | 262 | WWWWBODW | 273 | WWW:MM:YY | 21 | WWWWOOOW |
| 254 | WWWWWYYW | 263 | WWWWMYW | 274 | WYY:MMMM:YY | 238 | WWWWWWY |
| 255 | WWWWW000 | 264 | WWWWWYYW | 275 | YWWOBYYT | 286 | B:YY:0000MB |
| 256 | EBYYB000 | 265 | WWWWW000 | 276 | WYYOBYYT | 287 | YWOYBM0Y |
| 257 | EBYYB000 | 266 | WWWWWBYW | 277 | YWWWBB0Y | 288 | YWWWBY0Y |
| 183 | WWWWBBYT | 267 | WWWWMY0W | 278 | WWW00MBY | 289 | 00000MMO |
| 259 | WWWWBB0Y | 268 | WWWWY00B | 279 | WWW00MB0 | 290 | 00000BB0 |
| 260 | WWWOB000 | 269 | WWY:MM:BB | 280 | 0BYY0000 | 291 | 0WWW0000 |
| 261 | WWWWBO00 | 270 | WWY:MM:Y0Y | 281 | 0WWY:MM0:00 | 292 | 0WWW0000 |
| 271 | WWWOB000 | 282 | WWWWBY00 | 272 | WWWWBY00 | 283 | WWY:MM:00 | 39 |
Table 11  Motion rules for a register. First, to increment or decrement the register. Then, the particular cases: when the register contains 0 and then when it contains 1.

| Incrementing | Decrementing |
|--------------|--------------|
| 0 → 1        | testing 0    |
| 1 → 2        | 1 → 0        |

In Table 11 we can see several rules whose number is red according to the convention we already mentioned. As an example, rule 315 transforms a white cell into a red one if that latter has two contiguous neighbours and in that order, the other neighbours being white. That rule applies three times and when the created this way red cell is superfluous, it is returned to white by rule 318. A few rules play the key role: rule 336 creates the 1 when an empty registered is visited by a green locomotive. Rule 359 triggers the locomotive
which will stop the returning red locomotive as the former one signals the zero-
state of the register and arrives to the memory of decrementing the register
by the way devoted to the zero-test. At last, Rule 397 starts the process of
returning the content of the register to 0 when its content is 1 and at that time,
a red locomotive arrives at the register.

Figure 42 Working on a register. Top row, a green locomotive arrives: it incre-
ments the register. Bottom row, a red locomotive arrives: it decrements the register.
As decrementing a register needs one step more than incrementing it, the illustration
of decrementing starts one step further after the arrival of the red locomotive to the
register.

Figure 43 Working on a register, special cases. First two rows the register is empty,
last two rows it contains value 1. Topmost row: a green locomotive arrives; the content
of the register grows from 0 to 1. Next row: a red locomotive arrives; it can see that
the register is empty, so that it triggers the locomotive exits through a different way.
Third row from the top, a green locomotive arrives: the content of the register grows
from 1 to 2. Last row, a red locomotive arrives: the content of the register goes from 1
to 0. Here too, there is a delay in the progression of the red locomotive when compared
with the arrival of the green one.
4.7 Changing the colour of the locomotive

We finish that section with the rules managing the change of colours for a locomotive. Such a transformation may be necessary when the locomotive goes back to the area of the simulation where the instructions are placed. On the way back from a register, the locomotive is always red, even in the case when it was not possible to decrement the register. If the next instruction increments a register, the locomotive must become green. So we need to convert a red locomotive to a green one. We have already mentioned that the occurrence of a double locomotive happens within a round-about only. Accordingly, the change of colour concerns simple locomotives only. That needs a few instructions gathered in Table 12. Their application is illustrated by Figure 44.

| conservative and witness rules | motion rules |
|-------------------------------|--------------|
| 263 WWWWYYYYW                 | 447 WWWWBBGRR|
| 442 WWWWYYYYW                 | 448 WWWWBBRBB|
| 443 YYYYYYYYY                 | 449 WYYGGRRRR|
| 444 YYYYYYYYY                 | 450 WYYYBRRYG|
| 445 WYYYYYYYYY                | 451 WYYYYBRYY|
| 446 WYYYYBBRYY                | 452 WYYYYBYRR|

Figure 44 Changing a red simple locomotive into a green one.

The last rule has number 452. However, 42 rules are repeated at least once, 32 of them exactly once. Accordingly, the total number of rules as indicated in the tables is 420. However, as already mentioned, many rules of the tables are schemes of rules, so that the actual number of rules is 732. Note that the total number of all possible different rules is \(7^9 = 40,353,607\) which is approximately reduced to \(7^8 = 5,764,801\) if we consider rules which are pairwise different under rotation invariance. It is interesting to note that many rules can be gathered according to their neighbourhood. The neighbourhood of a rule \(x_0x_1\ldots x_7\) is the word \(x_1\ldots x_7\). As an example, the neighbourhood \(WWWWW\) occurs in rules 306 and 333, each of them being repeated in the tables. Rule 306 changes \(\r\) into \(\alpha\) while rule 333 changes \(\w\) into \(\alpha\). In the same line, the neighbourhood \(WWWWW\) occurs in rules 310 and 311. In rule 311 it changes \(\w\) into \(\alpha\) while in rule 310 it remains \(\beta\) unchanged. Note that rule 311 entails parasitic occurrences of \(\alpha\) which are erased by \(327 \ WWWW\ YYY\). That latter feature reminds me a similarity with the correction of errors in the replication of DNA in natural processes.
Conclusion

There are several questions raised by this result. Is it possible to reduce the number of states in this context? The huge number of arbitrary rules seems to say that it could be possible. However, that would be at the cost at reducing several facilities we had here although the number of freedom is reduced, compared with what was previously achieved. Perhaps another model should be used for that purpose. What could be done in the pentagrid is also an open question.

References

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

[2] M. Margenstern, New Tools for Cellular Automata in the Hyperbolic Plane, *Journal of Universal Computer Science*, **6**(12), (2000), 1226-1252.

[3] M. Margenstern, *Cellular Automata in Hyperbolic Spaces*, vol. 1, *Theory, Collection: Advances in Unconventional Computing and Cellular Automata*, Editor: Andrew Adamatzky, Old City Publishing, Philadelphia, (2007), 422p.

[4] M. Margenstern, An Upper Bound on the Number of States for a Strongly Universal Hyperbolic Cellular Automaton on the Pentagrid, *JAC2010* Turku, Finland, (2010), *Proceedings*, Turku Center for Computer Science, 168-179.

[5] M. Margenstern, *Small Universal Cellular Automata in Hyperbolic Spaces: A Collection of Jewels*, Springer Verlag, (2013), 331p.

[6] M. Margenstern, A Weakly Universal Cellular Automaton in the Pentagrid with Five States, *Lecture Notes in Computer Science*, C.S. Calude et al. (Eds.): Gruska’s Festschrift, **8808**, (2014), 99-113.

[7] M. Margenstern, A weakly universal cellular automaton in the heptagrid with three states, *arXiv:1410.1864v1*, (2014), 27p.

[8] M. Margenstern, A Weakly Universal Cellular Automaton in the Heptagrid of the Hyperbolic Plane, *Complex Systems*, **27**(4), (2018), 315-354.

[9] M. Margenstern, Y. Song, A new universal cellular automaton on the pentagrid, *Parallel Processing Letters*, **19**(2), (2009), 227-246.

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

[11] I. Stewart, A Subway Named Turing, Mathematical Recreations in Scientific American, (1994), 90-92.