A Universal Cellular Automaton on the Ternary Heptagrid

Maurice Margenstern\(^1\) and Yu Song\(^2\)

Laboratoire d’Informatique Théorique et Appliquée, EA 3097, Université de Metz, I.U.T. de Metz, Département d’Informatique, Île du Saulcy, 57045 Metz Cedex, France

Abstract

In this paper, we construct the first weakly universal cellular automaton on the ternary heptagrid. It requires six states only. It provides a universal automaton with less states than in the case of the pentagrid where the best result is nine states, a result also recently established by the authors.

Keywords: Cellular automata, hyperbolic plane, tessellations, universality

1 Introduction

As indicated in the abstract, this paper is the first result about a universal cellular automaton on the ternary heptagrid. Although the somehow bigger number of neighbours for one cell is an argument in favor for a smaller number of states than in the case of the pentagrid, this was never proved before. Also note that a direct translation of the result obtained in the pentagrid, using the continuous transformation constructed in [6,7] would certainly produce an automaton with at least the same number of states as in the pentagrid. This is why proving a better result needs a definite amount of work. As the reader will see, it is not at all trivial. Moreover, as this is performed in another grid which has its own properties, this result cannot be simply considered as an improvement of [10].

Our simulation uses the same model of a railway circuit as performed in [1] and in [10]. We refer the reader to [3,1] for the simulation of a register machine by a railway circuit. In order to help the reader to better understand the paper, we

---

\(^1\) margens@univ-metz.fr
\(^2\) sophia_41520@hotmail.com
sketch out the main lines of this simulation in Section 2. In Section 3, we remind the
reader about hyperbolic geometry and cellular automata on the ternary heptagrid.
Still in Section 3, we give the general features of the implementation of a railway
circuit in the ternary heptagrid and in Section 4, we study it very precisely. In
Section 5, we give the format of the rules and the transition table of the automaton
whose action is described in Section 4. We also indicate how a computer program
contributed to the construction of the table.

2 The railway circuit

As initially devised in [11] and then mentioned in [3,1,10], the circuit uses tracks
represented by lines and quarters of circles and switches. There are three kinds of
switches: the fixed, the memory and the flip-flop switches. They are represented
by the schemes given in Fig. 1.

![Figure 1](image)

**Figure 1** The three kinds of switches. From left to right: fixed, flip-flop and mem-
ory switches.

Note that a switch is an oriented structure: on one side, it has a single track \( u \)
and, on the other side, it has two tracks \( a \) and \( b \). This defines two ways of
crossing a switch. Call the way from \( u \) to \( a \) or \( b \) active. Call the other way, from \( a \)
or \( b \) to \( u \) passive. The names comes from the fact that in a passive way, the switch
plays no role on the trajectory of the locomotive. On the contrary, in an active
crossing, the switch indicates which track between \( a \) and \( b \) will be followed by the
locomotive after running on \( u \): the new track is called the selected track.

![Figure 2](image)

**Figure 2** The elementary circuit.

With the help of these three kind of switches, we define an elementary circuit
as in [11], which exactly contains one bit of information. The circuit is illustrated by
Fig. 2, above. It can be remarked that the working of the circuit strongly depends on
how the locomotive enters it. If the locomotive enters the circuit through $E$, it leaves the circuit through $O_1$ or $O_2$, depending on the selected track of the memory switch which stands near $E$. If the locomotive enters through $U$, the application of the given definitions shows that the selected track at the switches near $E$ and $U$ are both changed: the switch at $U$ is a flip-flop which is changed by the very active passage of the locomotive and the switch at $E$ is a memory one which is changed because it is passively crossed by the locomotive and through the non-selected track. The just described actions of the locomotive correspond to a read and a write operation on the bit contained by the circuit which consists of the configurations of the switches at $E$ and at $U$. It is assumed that the write operation is triggered when we know that we have to change the bit which we wish to rewrite.

From this element, it is easy to devise circuits which represent different parts of a register machine. As an example, Fig. 3 illustrates an implementation of a unit of a register.

Figure 3 Here, we have two consecutive units of a register. A register contains infinitely many copies of units. Note the tracks $i$, $d$, $r$, $j_1$ and $j_2$. For incrementing, the locomotive arrives at a unit through $i$ and it leaves the unit through $r$. For decrementing, it arrives though $d$ and it leaves also through $r$ if decrementing the register was possible, otherwise, it leaves through $j_1$ or $j_2$.

As indicated by its name, the fixed switch is left unchanged by the passage of the locomotive. It always remains in the same position: when actively crossed by the locomotive, the switch always sends it onto the same track. The flip-flop switch is assumed to be crossed actively only. Now, after each crossing by the locomotive, it changes the selected track. The memory switch can be crossed by the locomotive actively and passively. In an active passage, the locomotive is sent onto the selected track. Now, the selected track is defined by the track of the last passive crossing by the locomotive. Of course, at initial time, the selected track is fixed.

Other parts of the needed circuitry are described in [3,1]. The main idea in these different parts is to organize the circuit in possibly visiting several elementary circuits which represent the bits of a configuration which allow the whole system to remember the last visit of the locomotive. The use of this technique is needed for the following two operations.

When the locomotive arrives to a register $R$, it arrives either to increment $R$ or to decrement it. As can be seen on Fig. 3, when the instruction is performed, the locomotive goes back from the register by the same track. Accordingly, we need
somewhere to keep track of the fact whether the locomotive incremented $R$ or it decremented $R$. This is one type of control. The other control comes from the fact that several instructions usually apply to the same register. Again, when the locomotive goes back from $R$, in general it goes back to perform a new instruction which depends on the one it has just performed on $R$. Again this can be controlled by what we called the selector in \cite{3,1}.

At last, the dispatching of the locomotive on the right track for the next instruction is performed by the sequencer, a circuit whose main structure looks like its implementation in the classical models of cellular automata such as the game of life or the billiard ball model. The reader is referred to the already quoted papers for full details on the circuit. Remember that this implementation is performed in the Euclidean plane, as clear from Fig. 4 which illustrates the case of a few lines of a program of a register machine.

![Diagram](image)

**Figure 4** An example of the implementation of a small program of a register machine. On the left-hand side of the figure, the part of the sequencer. It can be noticed how the tracks are attached to each instruction of the program. Note that there are four decrementing instructions for $W$: this is why a selector gathers the arriving tracks before sending the locomotive to the control of the register. On the way back, the locomotive is sent on the right track.

Now, we turn to the implementation in the hyperbolic plane, which first requires some features of hyperbolic geometry.

### 3 Implementation in the hyperbolic plane

Hyperbolic geometry appeared in the first half of the 19th century, in the last attempts to prove the famous parallel axiom of Euclid’s *Elements* from the remaining ones. Independently, Lobachevsky and Bolyai discovered a new geometry by assuming that in the plane, from a point out of a given line, there are at least two lines which are parallel to the given line. Later, models of the new geometry were...
found, in particular Poincaré’s model, which is the frame of all this study.

In this model, the hyperbolic plane is the set of points which lie in the open unit disc of the Euclidean plane whose border is the unit circle. The lines of the hyperbolic plane in Poincaré’s disc model are either the trace of diametral lines or the trace of circles which are orthogonal to the unit circle, see Fig. 5. We say that the considered lines or circles support the hyperbolic line, simply line for short, when there is no ambiguity. Fig. 5 illustrates the notion of parallel and non-secant lines in this setting.

The angle between two $h$-lines are defined as the Euclidean angle between the tangents to their support. The reason for choosing the Poincaré’s model is that hyperbolic angles between $h$-lines are, in a natural way, the Euclidean angle between the corresponding supports. In particular, orthogonal circles support perpendicular $h$-lines.

![Figure 5](image_url)

**Figure 5** The lines $p$ and $q$ are parallel to the line $\ell$, with points at infinity $P$ and $Q$, on the border of the unit disc. The $h$-line $m$ is non-secant with $\ell$: it can be seen that there are infinitely many such lines.

### 3.1 The ternary heptagrid

Remember that in the Euclidean plane and up to similarities, there are only three kinds of tilings based on the recursive replication of a regular polygon by reflection in its sides and of the images in their sides. In the hyperbolic plane, where the notion of similarity is meaningless, there are infinitely many such tilings. In this paper, we consider the smallest regular polygon defined by the property that three copies of it can be put around a vertex in order to cover a neighbourhood of the vertex with no overlapping. This tiling is called the **ternary heptagrid**, see Fig. 6 and 7 for an illustrative representation. Here, we give a rough explanation of these objects, referring to [7] and to [5] for more details and references.

The left-hand side of Fig. 6 illustrates the ternary heptagrid. But, besides the occurrence of a lot of symmetries, nothing can be grasped on the structure of the tiling from this mere picture. The right-hand side picture of Fig. 6 illustrates the main tool to make the structure visible. There, we can see two lines which we call mid-point lines as they join mid-points of edges of heptagons belonging to the
tiling. On the figure, a half of each line is drawn with a thicker stroke. It is a ray issued from the common point of these lines: here, a mid-point of an edge of the central heptagon of the figure. We shall say a ray of mid-points. These two rays define an angle, and the set of tiles whose all mid-points of the edges fall inside the angle is called a sector.

Fig. 6 and 7 sketchily remember that the tiling is spanned by a generating tree. In fact, as can be noticed on both the right-hand side of Fig. 6 and the left-hand side of Fig. 7, the set of tiles constituting a sector is spanned by a Fibonacci tree.

Now, as indicated in Fig. 7, seven sectors around a central tile allow us to exactly cover the hyperbolic plane with the ternary heptagrid which is the tessellation obtained from the regular heptagon described above and easily seen on the figures.

In the left-hand side picture of Fig. 7, we represent the sectors in terms of tiles. The tiles are in bijection with the tree which is represented on the right-hand side part of the figure. This allows to define the coordinates in a sector of the ternary heptagrid, see [7]. We number the nodes of the tree, starting from the root and going on, level by level and, on each level, from the left to the right. Then, we represent each number in the basis defined by the Fibonacci sequence with \( f_1 = 1, f_2 = 2 \), taking the maximal representation, see [2,7].

**Figure 6** On the left: the tiling; on the right: the delimitation of the sectors which are spanned by a tree. Note the rays of mid-points. They are issued from the same point: a mid-point of an edge of the central cell of the figure.

**Figure 7** On the left: seven sector around a central tile; on the right: the representations of the numbers attached to the nodes of the Fibonacci tree.

One of the reasons to use this system of coordinates is that from any cell, we
can find out the coordinates of its neighbours in linear time with respect to the coordinate of the cell. Also in linear time from the coordinate of the cell, we can compute the path which goes from the central cell to the cell. These properties are established in [4,7] and they rely on a particular property of the coordinates in the tree which allow to compute the coordinate of the father of a node in constant time from the coordinate of the node. In the paper, the coordinate of a cell is of the form $\nu(\sigma)$ where $\sigma$ is the number of the sector where the cell is and $\nu$ is its number in the Fibonacci tree which spans the sector. Now, as the system of coordinates is fixed, we can turn to the application to the implementation of cellular automata on the ternary heptagrid, we shall say heptagrid for short.

### 3.2 Cellular automata on the ternary heptagrid

A cellular automaton on the heptagrid is defined by a local transition function which can be put in form of a table. Each row of the table defines a rule and the table has nine columns numbered from 0 to 8, each entry of the table containing a state of the automaton. On each row, column 0 contains the state of the cell to which the rule applies. The rule applies because columns 1 to 7 contain the states of the neighbours of the cell defined in the following way. For the central cell, its neighbour 1 is fixed once and for all. For another cell, its neighbour 1 is its father. In all cases, the other neighbours are increasingly numbered from 2 to 7 while counter-clockwise turning around the cell starting from side 1. The representation mentioned in Subsection 3.1 allows to find the coordinates of the neighbours from that of the coordinate of the cell in linear time. The list of states on a row, from column 0 to 7 is called the context of a rule. It is required that two different rules have different contexts. We say that the cellular automaton is deterministic. As there is a single row to which a rule can be applied to a given cell, the state of column 8 defines the new state of the cell. The local transition function is the function which transforms the state of a cell into its new one, also depending on the states of the neighbours as just mentioned.

An important case in the study of cellular automata is what are called rotation invariant cellular automata. To define this notion, we consider the following transformation on the rules. Say that the context of a rule is the rotated image of another one if and only if both contexts have the same state in column 0 and if one context is obtained from the other by a circular permutation on the contents of columns 1 to 7. Now, a cellular automaton is rotation invariant if and only if its table of transition $T$ possesses the following properties:

- for each row $\rho$ of $T$, $T$ also contains six rules exactly whose contexts are the rotated image of that of $\rho$ and whose new state is that of $\rho$;
- if $\rho_1$ and $\rho_2$ are two rules of $T$ whose contexts are the rotated image of each other, then their column 8 contains the same state.

The name of rotation invariance comes from the fact that a rotation around a tile $T$ leaving the heptagrid globally invariant is characterized by a circular permutation on the neighbours of $T$ defined as above.

Note that the universal cellular automata devised in [1,10] are rotation invariant.
while the one of [9] is not. For the question of rotation invariance for cellular automata on the heptagrid, we refer the reader to [8].

Now, we can turn to the simulation of the railway circuit by a cellular automaton.

### 3.3 The implementation of the railway circuit

In [1], the various elements of the circuit mentioned in [3] are implemented. In fact, the paper does not give an exact description of the implementation: it only gives the guidelines, but with enough details, so that an exact implementation is useless. In this paper, we take the same implementation exactly. This is why we do not repeat it in this paper and we refer the reader to [1] for further details. Just to help the reader to better see things, Fig. 8 provides a simplified illustration of the implementation of the example given by Fig. 4.

![Figure 8](image.png)

**Figure 8** The implementation, on the heptagrid, of the example of Fig. 4. In sector 1, also overlapping onto sector 2, the sequencing of the instructions of the program of the register machine. In sector 3, we can see the first register and, in sector 5, the second one. For simplicity, the figure represents two registers only.

Note the instructions which arrive to the control of the register through tracks in the shape of an arc of circle. Also note the return from the controller of the register when decrementing a register fails, because its content was zero.

Here, we just mention the general principle of motion of the locomotive which is again taken in the new setting which we define in the next sub-section.

The tracks are implemented by portions of paths between two switches. Such a portion consists in a finite sequence of tiles which are called blue ones. We require that each blue tile which is in contact neither with a switch nor a crossing has two blue neighbours exactly, where a neighbour of a tile $T$ is another tile sharing an edge with $T$. Its other neighbours are in the quiescent state.

From now on, we call tiles cells. Most cells are in a quiescent state which we also call the blank. In the following figures of the paper, it is represented by a light blue colour and it is not marked by a letter, contrary to the cells which are under
another state. The state of the cells which are on a blue tile is also blue, unless the locomotive is present on the cell. The locomotive is represented by two contiguous cells on a path. One cell is green, also represented by the letter G in the figures, the other is red, represented by the letter R. The green cell represents the front of the locomotive and the red cell represents its rear.

Now, the motion of the locomotive on a track is represented by Fig. 9. The easy rules can be found in [1].

Fig. 9 is a space-time diagram of the evolution of the states of the cells which are along a track.

![Figure 9](image)

**Figure 9** The motion of the locomotive on its tracks: first approximation.

In the heptagrid, the rules become a bit more complex. However, we can formulate them in a concise way with the help of Fig. 9. Table 1 gives a synthetic representation of the concerned rules. It is assumed that there are exactly five blank neighbours of the cell $c$. Then, the table gives the new state of $c$ depending on the state of $c$ and the states of the other non-blank neighbours.

<table>
<thead>
<tr>
<th>state</th>
<th>number of neighbours in B</th>
<th>G</th>
<th>R</th>
<th>new state</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>2</td>
<td>_</td>
<td>_</td>
<td>B</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>1</td>
<td>_</td>
<td>G</td>
</tr>
<tr>
<td>G</td>
<td>1</td>
<td>_</td>
<td>1</td>
<td>R</td>
</tr>
<tr>
<td>R</td>
<td>1</td>
<td>1</td>
<td>_</td>
<td>B</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>_</td>
<td>1</td>
<td>B</td>
</tr>
</tbody>
</table>

Table 1 Table for the definition of the rules of the basic motion.

Any configuration which respects these constraints defines a rule of the cellular automaton accordingly. We call these rules the rules of the basic motion.

The switches organize the meeting of tracks according to the principles defined in the previous section. By definition, four paths meet at a crossing and three ones meet at a switch. As in [1,10], such a meeting is materialized by a tile which plays a special role and which we call the centre of the meeting. The other parts of the tracks abut to a neighbour of the centre. This neighbour is called the first cell of the considered track.

In the case of a crossing, we have four tracks attached to two different paths. As
in [10], one path is marked by another blue colour for its first cell, see the picture (a) of Fig 10. We take advantage of the ternary heptagrid to put in contact two tracks which arrive at a neighbour of the centre when they do not belong to the same path. This allows each track to know what it has to do when the locomotive is passing through the crossing.

In [10], we have more systematically followed the pattern of the basic motion than in [1], even in a context where the cell has less than five blank neighbours. This is the reason why we could reduce the number of states of a weakly universal cellular automaton on the pentagrid. Now, in the heptagrid, the advantage we have mentioned for the crossings can be maintained for the switches. For them, the simulation of [10] makes use of all the nine states. As here we have less states, we have to simulate the differences by different configurations of cells. This is what is performed for the three types of switches where these configurations allow to differentiate the first cell. For the centre, it is an additional neighbour with the same colour. For the cell itself, the additional neighbours allow to differentiate its behaviour depending on its place with respect to the tracks.

4 The simulation in the heptagrid

In this section and in the next one, we prove the following:

**Theorem 4.1** (Margenstern-Song) – There is a cellular automaton on the ternary heptagrid which is weakly universal and which has six states. Moreover, the rules of the cellular automaton are rotation invariant. The cellular automaton has an infinite initial configuration which is ultimately periodic along two different rays of mid-points $r_1$ and $r_2$ of the ternary heptagrid and finite in the complement of the parts attached to $r_1$ and $r_2$.

The configurations at a crossing and at the different kinds of switches are given in Fig. 10. Say that such a configuration is idle if the locomotive is not near its centre. As in [10], the centre of an idle configuration is blue. The difference between a crossing and the different switches are given by the configuration itself.

![Figure 10](image)

**Figure 10** The idle configurations. From left to right: crossing and then fixed, memory and flip-flop switches.

Now, we shall consider that the locomotive arrives at such a meeting. In the next sub-sections, we look at what happens in each case.
4.1 The crossings

There are four cases of crossing from the different sectors. We shall consider two cases only, from sectors 1 and 7 illustrated by Fig. 11 and 12 respectively. These pictures show that we need the six states for the crossing: W, B, G, R, B2 and G2. Five of them are enough for one path. The additional state, G2, is required to distinguish the arrival of the front of the locomotive from the other path.

Fig. 12 illustrates what happens when the locomotive comes from the other path. Note that the situation from the other tracks is the same as from the ones we have studied: the other tracks are obtained by a rotation around the centre.

The second path, from sectors 7 to 3, is signalized by the first cells of its tracks: there are in the state B2 while the similar cells for the other tracks are in the state B.

Figure 11 The locomotive goes through a crossing: here from sector 1 to sector 4.

Figure 12 The locomotive goes through a crossing: here from sector 7 to sector 3.

Without loss of generality, we may assume that all crossings are rotated images of the configuration described by picture (a) of Fig. 10. Note that we require the track to go along the rightmost branch of the Fibonacci tree spanning the corresponding sectors only up to the third level. On the complement of a ball of radius 4 around the centre of the crossing, the four paths may be very different. In particular, from the assumption of rotation invariance of the configurations, we may always number the sectors by counter-clockwise turning around the centre in such a way that the cells marked by B2 are in sectors 7 and 3.

4.2 The switches

All switches are built according a similar pattern. They are already marked by the fact that the cell 2(1) is in a different state which is specific to each switch. This state is blank, B2 and G2 for the fixed, the memory and the flip-flop switch.
respectively. We can see that the different local configurations around the centre allow to have the cells (0), 1(1) and 1(7) play different roles, see Fig. 10, pictures (b), (c) and (d). Three tracks meet at a centre, the central cell of the pictures of Fig. 10, 13, 14 and 15. Here also, the track arriving to the switch goes along the rightmost branch of the Fibonacci tree spanning the corresponding sector, passing through cells 33, 13, 4 and 1. Moreover, the unique arriving track is in sector 4, and the two tracks emanating from the switch are in sectors 1 and 7. On the figures, the selected path is in sector 1. Of course, symmetric figures where the selected paths is in sector 7 are also possible for the memory and flip-flop switches. For the fixed one, this is not mandatory: indeed, a fixed switch to the other direction can be simulated by the fixed switch of Fig. 13 followed by a crossing which allows to exchange the directions between the tracks which go out from the switch. Now, any switch can be considered as a rotated image of the switches represented in Fig. 10, 13, 14 and 15 and their symmetrical counterparts. Now, for all switches, cell 1(4) has two neighbours permanently in the state $B_2$. This permanence is guaranteed by the two additional cells in the same state. These additional cells have a unique non-blank neighbour, so that this condition is easy to satisfy.

**Fixed switch**

Its working is illustrated by Fig. 13.

The fixed switch uses the rules of the crossings too. Contrarily to what we have in the pentagrid, here the fact that the first cells of the selected and non-selected tracks can see each other simplifies the situation a lot. Also note that the configuration of the central cell is clearly different from the one of a crossing.

![Fixed switch diagram](image)

**Figure 13** The locomotive crosses a fixed switch from the non-selected track, here in sector 7.

**Memory switch**

The memory switch is the most complicate item among the three kinds of switches.

The crossing from track 1 or 4 is not very difficult and it mainly uses the rules of a fixed switch, taking into account the difference of configuration of the central cell. However, specific rules are also in use due to the state $B_2$ which is permanently present in the cell 2(1).

The memory switch is characterized by the configuration defined by the cells 1(1), 2(1) and 1(7). One of the cells 1(1) and 1(7) has the same state as
the cell 2(1): it is the first cell of the track which is not selected by the switch. It constitutes a **mark** which changes after the visit of the locomotive from this track.

Fig. 14 illustrates the working of the switch when the locomotive comes from the non-selected track, here track 7. As the colours of the illustrations have been placed on the heptagrid from the execution of the cellular automaton by a computer program, the representation is exact. We can see that the mark is changed. The change of selected track occurs from picture \(c\) to picture \(d\) of the figure. It is triggered when the front of the locomotive is at the centre of the switch. At this moment, both cells 1(1) and 1(7) can see this state in cell 0 at the same time and the cell 1(1) can see the red state of the cell 1(7). This allows the mark to switch from one track to the other.

**Figure 14** The locomotive crosses a memory switch from the non-selected track, here in sector 7. Note the change of the selected track when the rear of the locomotive leaves the first cell of the non-selected track.

**Flip-flop switch**

Here, the switch is again recognized by the state at the cell 2(1): this time, \(G2\). Note that here, the transformation is easier than in the case of the pentagrid, see [10].

It can be remarked that the configuration around the central cell and around the cell 1(4) is exactly the same as the configurations of the same cells in a fixed switch. This means that the rules used for the motion of the locomotive actively arriving to a fixed switch are also in use here when the locomotive arrives to this switch. The situation changes when the locomotive arrives to the cells 1(1) or 1(7) only. Now there, the neighbouring of the cells 1(1) and 1(7) of a flip-flop switch are clearly different from the neighbouring cells in both the fixed and the memory switches.

On Fig. 15, we can see how the change of marks occurs to indicate the new selected track. Again, it makes use that the cells 1(1) and 1(7) can see each other until the very last moment: when the rear of the locomotive is already about to leave the cell 1(1).
leaves at the next neighbour such turns are required and they raised a problem which was solved, see [10]. Here by sides 4 and 5 and that those of a white node are given by sides 3, 4 and 5.

Note that the sons of a black node in the Fibonacci tree are given numbers of the side increase when counter-clockwise turning around the tile which a side fixed once and for all. Once this side is fixed, all the others are also fixed: the other cells, side 1 is the side shared with the father. In the central cell, side 1 is the side shared with the father. Note that we could control the behaviour of the switches without any additional state, mainly because we succeeded to make the neighbouring around the key cells of each case different from their neighbouring in the other cases.

The tracks

In the case of the pentagrid, we had a problem of sharp turns on a track. To fix things, a sharp turn at the cell $c$ means that the track arrives at a neighbour $n$ and leaves at the next neighbour $n+1$ while turning around the cell. In the pentagrid, such turns are required and they raised a problem which was solved, see [10]. Here we have no more this problem: first, even if sharp turns occurred, this would raise no problem. But in fact, sharp turns can be avoided: in the previous setting, as neighbours $n$ and $n+1$ are contiguous, the track goes from $n$ to $n+1$ without passing through $c$. Also here, we do not need any additional state. Consequently, the simulation requires six states only.

5 The rules

The set of rules of the automaton is displayed in table 2. Here, we indicate how the rules where computed by a computer program and how the rules are represented and dispatched in the table.

5.1 The format of the rules and rotation invariance

Now, we complete the information given in Subsection 3.2 about the format of the rules.

Each cell has a numbering of its sides from 1 to 7. The central cell excepted, for the other cells, side 1 is the side shared with the father. In the central cell, side 1 is a side fixed once and for all. Once this side is fixed, all the others are also fixed: the numbers of the side increase when counter-clockwise turning around the tile which supports the cell. Note that the sons of a black node in the Fibonacci tree are given by sides 4 and 5 and that those of a white node are given by sides 3, 4 and 5.

Figure 15 The locomotive crosses a flip-flop switch, necessarily actively, here from sector 4. Note the change of the selected track when the rear of the locomotive leaves the switch.
A rule can be denoted as follows:

\[ \eta_0, \eta_1, \eta_2, \eta_3, \eta_4, \eta_5, \eta_6, \eta_7 \rightarrow \eta_0^1, \]

where \( \eta_0 \) is the state of the cell, \( \eta_i \) the state of its neighbour \( i \) and \( \eta_0^1 \) is its new state.

In Subsection 3.2, we defined the context of a rule which is the word \( \eta_0 \eta_1 \eta_2 \eta_3 \eta_4 \eta_5 \eta_6 \eta_7 \). In Table 2, we represent the rules by a word too. The just above form of the rule is represented by the following word: \( \eta_0 \eta_1 \eta_2 \eta_3 \eta_4 \eta_5 \eta_6 \eta_7 \eta_0^1 \), in the same notations as the rule. We can see that the context is the biggest proper prefix of this word.

To define the features of the computer program, it can be useful to devise a test in order to check the rotation invariance of the automaton. The notion of rotated form can be used for that purpose in the following way.

Consider a rule \((\eta)\) of the cellular automaton which we represent by the word \( \eta_0 \eta_1 \eta_2 \eta_3 \eta_4 \eta_5 \eta_6 \eta_7 \eta_0^1 \), as above indicated. Its rotated forms are defined by \( \eta_0 \eta_\pi(1) \eta_\pi(2) \eta_\pi(3) \eta_\pi(4) \eta_\pi(5) \eta_\pi(6) \eta_\pi(7) \eta_0^1 \), where \( \pi \) is a circular permutation on \( 1..7 \). We can see that the difference between the rotated forms of a rule and the rule itself lies in their contexts: they can be deduced from each other by a suitable circular permutation on \( \{1..7\} \). There are 7 rotated forms for each rule. Now, as we consider the contexts as words, we can order them lexicographically. And so, there is a minimal rotated form for each rule \((\eta)\): that whose context is the minimum of the contexts of the rotated forms of \((\eta)\) with respect to the lexicographical order which is a linear order. Denote \( \min(\eta) \) the minimal rotated form of the rule \((\eta)\). Remember that the current state of the cell is the first letter of the context.

Now, it is plain that we have the following property:

**Lemma 5.1** The set of rules of a cellular automaton on the heptagrid is invariant by rotation if and only if for any pair of rules \((\eta)\) and \((\epsilon)\) of the automaton, if \( \min(\eta) \) and \( \min(\epsilon) \) have equal contexts then \( \eta_0^1 = \epsilon_0^1 \).

### 5.2 The program and the table of the rules

The computer program was written in ADA.

The program uploads the initial configuration of the crossings and of the switches from a file and puts the corresponding information into a table 0. In this table, each row represents a cell. The entries of the row indicate the coordinates of the neighbours of the cell as well as the states of the cell and of its neighbours. The program also contains a copy of table 0 with no state in the cells which we call table 1. The set of rules is in a file under an appropriate format, close to the one which was depicted in Subsection 5.1.

During the construction of the set of rules, the program works as follows. When we run the program, it reads the file of the rules which, initially contains the rule \( \text{W W W W W W W} \) which says that a cell in the quiescent state whose neighbours are all

**Table 2** The rules of the cellular automaton of Theorem 4.1:
quiescent remains quiescent. Starting from the central cell the program scans the
sectors one after the other and in each one, from the root to the last level, level by
level. For each cell \( c \), call \textbf{context} of the cell the sequence of states constituted of
the state of \( c \) and those of the neighbours of \( c \), starting from its neighbour 1 and
going on by counter-clockwise turning around \( c \). The program takes the context \( \kappa \)
of \( c \) in table 0. Then, it compares \( \kappa \) with the contexts of the rules of the file. If
it finds a match, it copies the new state of the rule at the address of \( c \) in table 1,
under column 0. If it does not find a match, it asks for a new rule which the user
writes into the file. To help the user, the program indicates the context of the cell.
The user enters the new state only. Then the program resumes its computation: it
reads again table 0 from the initial configuration and performs the computation as
far as possible. If it can compute the new state of all cells of table 0, it completes
table 1 by computing the new states of the neighbours of each cell. When this task
is over, the program copies table 1 onto table 0: a new step of the computation of
the cellular automaton can be processed. This cycle is repeated until no new rule
is required and until the fixed in advance number of steps is reached.

Now, when a new rule is entered by the user on a cell \( c \), it may happen that the
new rule is in conflict with the previously entered rules. This happens when there is
a rule \( \eta \) whose context is a rotated form of the context of \( c \), but the state suggested
by the user is not the new state of the rule. In this case, the program stops with
an error message which also displays the rule with which the program have found a
mismatch. If the rule constructed on the context of the cell and the state indicated
by the user is a rotated form of an already existing rule, it is appended to the set
of rules.

When the program can be run without asking a new rule nor indicating any
error, we know that the set of rules is computed.

The program also contributes to check that the rules are rotation invariant, using
lemma 5.1 which is very easy to implement: this is left to the reader. The same
property was used in [10]. In the display of the rules in Table 2, we assume that the
minimal rotated forms of the rules are pairwise distinct. This means that on the
seven rotated forms which should be present for each rule, we keep one form only
for the table. As we have 545 rules in the table, this is an important simplification.

Note that we have obtained 545 rules instead of 299 in the case of the pentagrid,
see [10]. In Table 2, the rules are displayed in the format indicated in Subsection 5.1.
Also, the rules are gathered according to the order in which they were entered in
the set of rules during the execution of the program. This order also corresponds to
the different configurations which we described in Section 4 and to the figures which
are displayed in this section as well as figures which are not displayed but which
correspond to the motions described in the section. The reader may check in the
table that for all crossings and switches, the motion of the locomotive was checked
for all possible arrivals. Also, the program have checked configurations which are
not represented in the figures of the paper: as an example, the picture (h) of Fig. 14
represents the idle configuration of the memory switch in which the selected track
is 7. But the figure represents a motion in which, initially, the selected track is 1.
No figure of the paper represents the motion of the locomotive when the selected track is 7. However, in order to get a correct set of rules, the program had to test this situation too. As the reader can check it on the table, all the needed tests were performed. Indeed, each test on a type of motion induced new rules where at least one of them was not a rotated form of a previously obtained rule.

This completes the proof of Theorem 4.1.

6 Conclusion

It is probably possible to do better, but it will require some effort and changes in the configurations themselves.

References


