

An algorithm for building intrinsically universal cellular automata in hyperbolic spaces

Maurice Margenstern
Université Paul Verlaine – Metz
LITA, EA 3097, UFR-MIM
Île du Saulcy, 57045 METZ Cédex, FRANCE
Email: margens@univ-metz.fr

Abstract—Many proofs about universality are performed by reduction to a well known universal process. An intrinsically universal cellular automaton on a given space X must be able to 'directly' simulate any cellular automaton on X , starting from a finite configuration. Examples of such automata are known for the line and for planar CA's in the Euclidean case. In this paper, we give the construction of such an intrinsically universal cellular automaton for the hyperbolic plane. The construction is valid for infinitely many grids of the hyperbolic plane and also for the dodecahedral rectangular grid of the hyperbolic 3D space.

I. INTRODUCTION

In many cases, **indirect** proofs of universality proofs for cellular automata are based on the simulation of a well known universal device. Also, when initial configurations are infinite, the result is usually called a **weak** universality result. The first proofs of universality for cellular automata were mostly indirect proofs. Also, results of universality with a small number of states are very often weak universality results. By contrast, results about a **direct** simulation of cellular automata by a single one is called **intrinsic** universality. In this paper, we shall also restrict the simulation to cellular automata which work starting from an initial **finite** configuration. The initial configuration of the simulated cellular automaton as well as that of the simulating one must be both finite.

Recently, there was a new interest to intrinsic universality see, for instance [3], [10], [11].

In this paper, we deal with this problem on a hyperbolic planar grid. We shall prove the following result:

Theorem 1: *There is a cellular automaton on the pentagrid, working on finite configurations, which is able to simulate any cellular automaton on the pentagrid working on finite configurations, starting from an appropriate encoding of the simulated CA and of its initial configuration.*

As there is no similarity in the hyperbolic plane, solutions of the Euclidean case cannot be simply transported. We have to consider the problem from scratch.

In the second section, we remind very sketchily Poincaré's disc model of the hyperbolic plane. Then, also sketchily we remind a few properties of the pentagrid and other grids of the hyperbolic plane. Then, in section 3, we give the main tools of the proof and then, in section 4, we give the main lines of the proof, focusing on the key points.

II. TESSELLATIONS IN THE HYPERBOLIC PLANE

A. The hyperbolic plane

In Poincaré's disc model, the hyperbolic plane is the set of points which lie in the open unit disc of the Euclidean plane.

The lines of the hyperbolic plane in Poincaré's disc model are the trace of either diametral lines or circles which are orthogonal to the unit circle. We say that the considered lines or circles **support** the hyperbolic line, ***h*-line** for short, and sometimes simply **line** when there is no ambiguity.

Poincaré's unit disc model of the hyperbolic plane makes an intensive use of some properties of the Euclidean geometry of circles, see [4] for an elementary presentation of the properties which are needed for this paper.

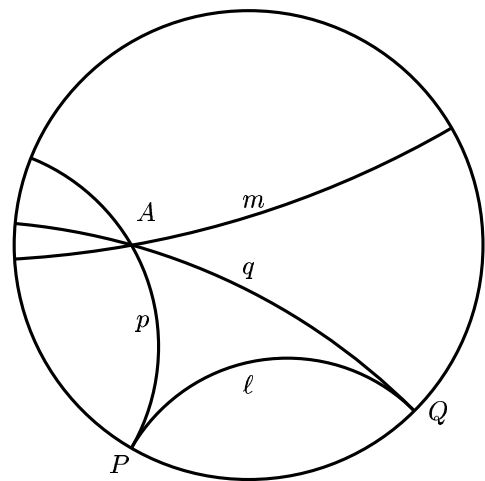


Figure 1: *The lines p and q are parallel to the line l , with points at infinity P and Q . The h -line m is non-secant with l .*

Consider the points of the unit circle as **points at infinity** for the hyperbolic plane. A ***h*-line** defines two points at infinity by the intersection of its Euclidean support with the unit circle which are called points at infinity of the ***h*-line**. Any ***h*-line** has exactly two points at infinity. Two points at infinity define a unique ***h*-line** passing through them. A point at infinity and a point in the hyperbolic plane uniquely define a ***h*-line**.

B. The pentagrid

A well known consequence of an important theorem proved by Poincaré, see [12], [1], [6] is that there are infinitely many tilings of the hyperbolic plane generated by **tessellation**: this means that the tiles are obtained by the reflection of an initial regular polygon in its sides and, recursively, of the images in their sides.

Below, we represent a quarter of the pentagrid,



Figure 2: The pentagrid and its tree-like construction.

The pentagrid is constructed in this way from the regular pentagon with right angles as vertex angles. As shown in [5], [4], it can be constructed by a tree which we call Fibonacci tree. This name comes from the fact that the number of the nodes of the tree which are at the same level n is f_{2n+1} , the odd term of rank n in the Fibonacci sequence. As we shall intensively use Fibonacci trees in the sequel, we remember that such a tree, [5], [4], is generated by the following rules:

$$B \rightarrow BW, W \rightarrow BWW$$

where B denotes **black** nodes, *i.e.* nodes with two sons and W denotes **white** nodes, *i.e.* nodes with three sons. Above, in figure 2, black nodes are defined by a red arc while white nodes are defined by blue and green arcs of the Fibonacci tree.

III. THE MAIN TOOLS

Remember that the problem is to simulate **any** cellular automaton on the pentagrid by a cellular automaton also defined on the pentagrid. Also, both the simulated and the simulating cellular automata have to start their computation from a finite configuration. Later on, we say CA as an abbreviation of cellular automaton.

A. Encoding other CA's: the scaled trees

The first task is to define the encoding of the simulated CA A . The problem is that the simulating CA U has a finite number of states, say n_U and U may be able to simulate any A , whatever the number of states n_A is. Also, U must be able to

encode any finite configuration in the pentagrid as well as any set of rules for A , of course, within the limit of n_A states.

Looking again at the pentagrid, we can notice that infinitely many other Fibonacci trees can be constructed by using the following process which is illustrated by figure 3, below.

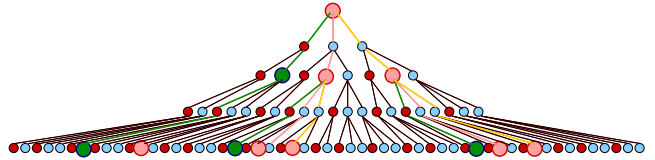


Figure 3: An example of scaling with a factor 2.

The rigorous definition starts by the definition of the balls:

Definition 1: Define the **distance** $d(c_1, c_2)$ between cells c_1 and c_2 by:

$$d(T_1, T_2) = \#\{T; T \text{ in shortest path from } T_1 \text{ to } T_2\}$$

Then, we call **ball** B_n of **radius** n around c_0 the set: $\{c; d(c_0, c) \leq n\}$ and we denote by ∂B_n the **border** of B_n , *i.e.* the set $\{c; d(c_0, c) = n\}$.

Now we can define a **scaled tree** by factor k with $k \in \mathbb{N}$, $k > 1$ by the following process:

- select 3 white nodes in the quarter of ∂B_k : the leftmost will be the **black** scaled node; the others will be the **white** scaled nodes;
- mark the branches to the scaled nodes;
- for each selected node ν , recursively repeat with the B_k around ν , selecting two nodes if ν is black and three nodes if ν is white.

Then, we assemble 5 scaled trees around a central cell. Below, figure 5 gives an illustration of such assembled trees.

Now, we can proceed with the encoding of A .

First, n_A is encoded in unary by n_A marked cells in some B_m . The smallest m is $O(\log n_A)$.

Next, we have to encode the transition table of A . For this purpose, remember that the format of a rule is the following: let s_0 be the current state of the cell, s_f be the state of its father, and s_i be the state of the i^{th} neighbour, counted counter-clockwise from the father. Then, a rule of the table is of the form $s_0, s_f, s_2, s_3, s_4, s_5 \rightarrow s$. Let us call $\{s_0, s_f, s_2, s_3, s_4, s_5\}$ the **context** and s the **result** or the **new state**.

Taking this format into account, we use a scaled tree to dispatch the rules. The first level contains n_A nodes, each one representing a state by its position with respect to the leftmost node. This level is associated with the current state of the cell. From each node of the first level, we have a subtree which has a similar structure: the first level has n_A nodes now representing the possible state of the father. This is repeated for the other four neighbours of the cell. At last, on the place of each node ν of the last level, we have a ball B_m around ν which contains the code of the new state for the rule whose context is defined by the path going from the root of the scaled tree to ν .

An example of such a tree is given below, by figure 4.

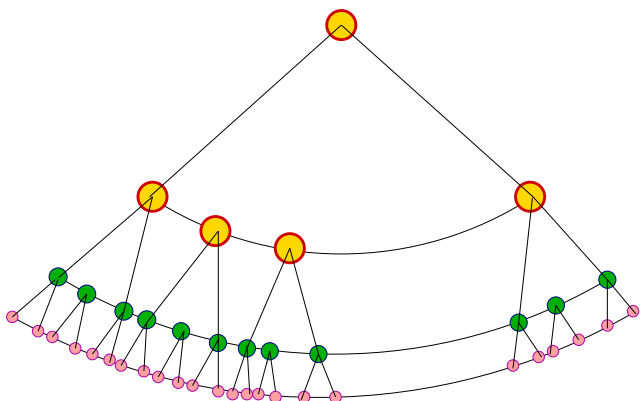


Figure 4: An example of a tree dispatching the table of transitions of A .

B. The layers

Now, we introduce another ingredient of the proof: the notion of **layers** which are the analog for CA's of **tracks** for Turing machines. Once for all, we fix s copies of the pentagrid and we imagine that we put them one upon another. We may assume that the cells exactly coincide from one copy to another.

In each copy, we may assume that the cells are addressed by five **standard** Fibonacci trees, each one reproducing the coordinates of [4]. We may also assume that each cell can see all cells of other layers with the same address as well as the cells of all the layers of its five neighbours.

The layers will enable us to devote some of them to a part of the cycle the repetition of which is the basis of the simulation of A by U .

In this section, we show how the layers will be used to organize the **initial configuration**.

We take k such that B_k around O , the central cell, contains both the table of rules of A and the initial configuration of A . Next, we denote by F a Fibonacci tree scaled by $2k$. From the choice of k , for any ν, μ in F , $\nu \neq \mu$, B_k around ν does not intersect B_k around μ . We call **simul-nodes** the nodes of F as they are devoted to the simulation of the A -cells.

Within B_k , we have five basic layers:

- L_0 : propagation of the standard Fibonacci tree structure
- L_1 : propagation of the structure of F
- L_2 : propagation of the border of the configuration if needed; L_2 contains the border of the configuration
- L_3 : for each B_k around a simul-node ν of F , in a B_a around ν with $a < k$, the state of the simulated cell of A
- L_4 : for each B_k around a simul-node of F : a copy of the table of rules

Later, other layers will be used for other stages of the computation.

IV. THE PROOF: U SIMULATING A

A. Propagation of the trees

Below, table 1 indicates the rules for propagating the structure of five assembled standard Fibonacci trees which is used in L_0 .

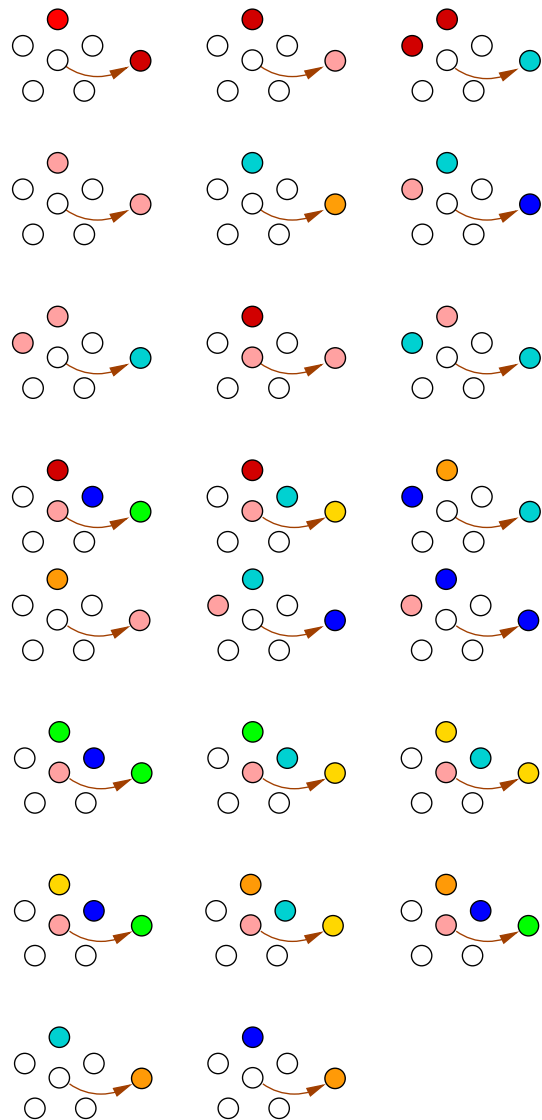


Table 1: Table of the rules for the simultaneous propagation of five standard Fibonacci trees.

Initially, cells are quiescent except O which is red. First row: the five cells surrounding O become light red. Then a cell which sees one red cell becomes pink and a cell which sees two red cells becomes light blue.

Next rows: when information is sufficient, pink cells become orange if their father in the tree is blue and they become green or yellow when the father is white also depending on their rank as sons of this father. Also, dark blue cells correspond to black sons of a black node while simple blue is for black sons of a white node.

The propagation is stopped by the cells which are on the border ∂B_k . For this purpose, we call **silver** the colour of the cells of the border.

The action of the border is simple: when a white cell of L_0 happens to see a silver cell in L_2 , it remains white, whatever the colours of the neighbours are. The rules of table 1 easily allow to implement this property.

Next, we turn to the description of the propagation of F . Consider figure 5, below. The filled disc is B_k , which contains the initial configuration and its border defines the first row in the assembling of five copies of F . The first row contains the root of each copy of F . The process of constructing the next row which corresponds to the sons of the root is illustrated by figure 6, below.

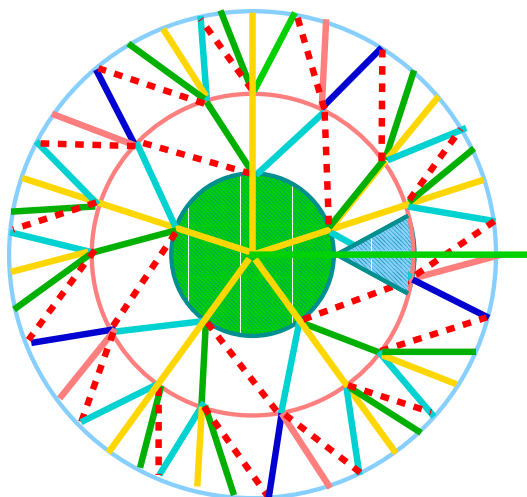


Figure 5: The propagation of the scaled tree via the 'big' rings.

To construct the second level, consider all the branches starting from the central cell and reaching a silver cell on the first row. On each branch, the central cell sends a synchronizing signal which starts at the same time on each branch. At synchronization time, the central cell sends a signal to the end of the branch at speed 1 and the end of the branch sends a signal further, at speed $\frac{2}{3}$. This allows to obtain the distance $2k$ between two consecutive levels of the F -trees. Notice that the new level is again a border. It is again constituted of silver cells and it lies on L_2 .

Below, figure 6 illustrates a copying process where the length of the initial segment is multiplied by 2.

Next, for $m \geq 2$, the construction of the level $m+1$ makes use of the levels $m-1$ and m . It also uses a synchronization of the branches between the levels $m-1$ and m which involves a synchronization from the central cell up to the level m . Now, at synchronization, we again have a signal at speed 1 starting from the level $m-1$ but we now have a signal at speed $\frac{1}{2}$ sent from the level m . The reason is that now, the distance between the levels $m-1$ and m is already $2k$.

The propagation of F involves three layers, L_0 , L_1 and L_2 . First, on L_2 , the new border is put on the second row of F by the process illustrated by figure 6. Then, on L_0 , the standard Fibonacci tree is propagated up to the new border. When this

is completed, F is propagated on L_1 by one more level marked by the new border on L_2 .

W	W	W	A	A	A	B	W	W	W	W	W	W	W	W	W	W	W	W	W
W	W	W	H	A	A	B	F	W	W	W	W	W	W	W	W	W	W	W	W
W	W	W	H	H	A	B	D	G	W	W	W	W	W	W	W	W	W	W	W
W	W	W	H	H	H	B	E	D	W	W	W	W	W	W	W	W	W	W	W
W	W	W	H	H	H	Z	E	F	W	W	W	W	W	W	W	W	W	W	W
W	W	W	H	H	H	E	S	E	D	G	W	W	W	W	W	W	W	W	W
W	W	W	H	H	E	E	S	S	E	D	W	W	W	W	W	W	W	W	W
W	W	W	H	E	E	E	S	S	S	E	F	W	W	W	W	W	W	W	W
W	W	W	E	E	E	E	S	S	S	S	D	G	W	W	W	W	W	W	W
W	W	W	E	E	E	E	S	S	S	S	S	D	W	W	W	W	W	W	W
W	W	W	E	E	E	E	S	S	S	S	S	S	F	W	W	W	W	W	W
W	W	W	E	E	E	E	S	S	S	S	S	S	S	G	W	W	W	W	W
W	W	W	E	E	E	E	S	S	S	S	S	S	S	S	W	W	W	W	W

Figure 6: Replicating an information. Note that the signal at speed $\frac{2}{3}$ involves 5 states: D, E, F, G and W, which is the blank.

Below, figure 7 illustrates the selection of F -nodes, the simul-nodes, among those of the standard Fibonacci tree. Using the colours of table 1, coloured paths define the corresponding F -nodes by the intersection of the path with a level. The intersection gives rise to new branches, depending on the colour of the new simul-node. There is a great freedom to choose the nodes of the path. When possible, the path always chooses the central son of a white node.

A last point can now be fixed. On figure 5, we notice dotted red edges. They are exactly what has to be appended to the F -tree in order to obtain the connection of each cell to its neighbours.

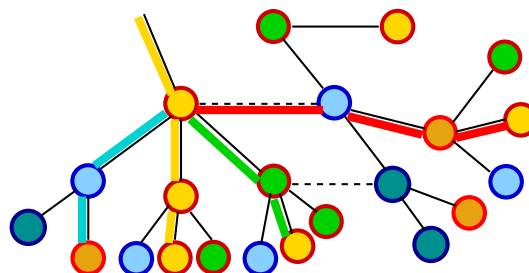


Figure 7: Propagation of the F -structure. In blue, propagation for a black node. In green, yellow and orange, propagation of white nodes.

Below, figure 8 indicates how these dotted edges are implemented. They follow a branch of a standard Fibonacci tree in order to go from one F -level to the next one. Once the new F -level is reached, the signal goes on on the level, always counter-clockwise, until it reaches a simul-node.

We have to notice an important feature about the dotted arcs. The length of a path along such an arc contains a constant segment of length $2k$ for going from one level to the next one. Then it contains a certain length ℓ on the new level until the new simul-node is reached. As the rank of the level increases,

ℓ also increases and the growth is exponential in the rank or, which is the same, in the distance to O .

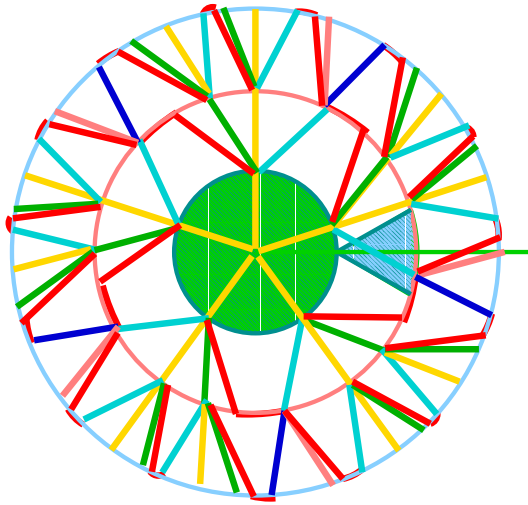


Figure 8: The F -trees, equipped with the connections of each simul-node with all its simul-neighbours.

B. The simulation

The simulation of A by U is split into **cycles**. Each cycle reproduce the execution of one step according to the computation of A . The beginning of a cycle is marked by a special synchronizing time which will be the **top** of the clock of the simulating process. Due to the remark about the dotted edges, the time between two tops of the clock is exponential with respect to the time of U .

A cycle itself can be split into **stages** according to the following scheme.

- 1 - copying the states of the simul-neighbours of the current simul-cell;
- 2 - finding the new state in the table of rules;
- 3 - on the layer L_3 , copying the new state in the simul-cell;
- 4 - waiting for the signal of the new cycle.

A bit later, we shall more precisely indicate which algorithms are involved by the first three stages. Now, we shall consider the last stage and explain why it is needed to wait.

The first reason is the delay imposed by the dotted connections which are different from those of the simul-neighbours which are connected to the current simul-node by a branch of the standard Fibonacci tree.

The second reason is that before turning to the next cycle, the simulation control has to check whether the new configuration is identical with the current configuration. For this purpose, the computation uses a sixth layer, L_5 , which contains a copy of the current configuration. When the test is performed, if the configurations is different, the new configuration is copied on L_5 and the computation enters a new cycle. If the configurations are identical, then a terminating signal is sent. Also, in the case of a difference between the configurations, it

may happen that a cell of the border becomes active, in which case a new level must be created on the five F -trees.

To perform all these tasks, the processing of the stages is performed according to the following protocol:

A synchronization signal indicates the beginning of the first stage. This means that at the same time, each simul-cell starts its work. It sends a copy of its own states to all its neighbouring simul-cells. Taking advantage of the synchronization signal, the states which are on the level L_3 move to the needed cells, following the paths which lie on L_1 .

When the neighbouring states arrive to a simul-cell, they are led by appropriate paths to the copy of the transition table which is present within the B_k around each cell. The path leads each copy on the level corresponding to the simul-neighbour from which it comes. From the simul-cell, a copy of the current state also arrives to the area of the transition table at the first level of this tree. The copy of a state exactly contains its number in unary. Accordingly, this allows to locate which node on the tree of the transition table corresponds to the current state of the simul-cell. Then, the located tree becomes **enlighten**, which allows the different copies of the state to find with a similar process where they have to propagate the enlightenment. When this is completed, the new state is located. At this moment, a synchronization signal is sent from the centre of the ball which contains the new state. This allows to start the copying process of the new state on its new place, in the simul-cell. When this is completed, the simul-cell sends a signal **completed** to its simul-father and the whole B_k around the simul-cell enters a waiting state.

Before looking at this process and at the comparison with the current configuration stored on L_5 , note that the above described process needs no particular synchronization: the first enlightenment in the table of transitions is performed while the copies of states of the neighbouring simul-cell move to the considered simul-cell c . The distance between levels being twice the radius of the B_k around c , a copy of the current state of c has time to reach the needed position before a copy of a neighbouring state arrives. The same can be performed for the coming copies by appropriate paths of suited lengths.

The signal **completed** sent by a simul-cell is collected by its simul-father. When such a father notices that all its sons and itself have completed their computation, it sends also a signal **completed** to its father. Accordingly, when such a signal reaches the central cell, the simulation of one step in A -computation is completed. But, at the same time, the central cell may have received the information that a simul-cell of the border is no more blank, as we may assume that in the current configuration, all the simul-cells of the border are blank. If this is the case, the central cell knows that the new configuration is different from the current one and it sends a synchronization signal **repeat**. If not, it sends a signal for synchronization **compare**. At the time of synchronization of **compare**, each simul-cell compares itself with its "brother" on the level L_5 . The cell sends the result to its father as soon as it received the result of its simul-sons. And so, we have again the final result of the comparison at the central cell. Depending on the

result, it sends either a synchronization signal `repeat` or a synchronization signal `stop`. The effect of the signal `stop` is that all cells follow a new transition table: it remains in the same state, whatever the states of its neighbours are.

On another hand, the signal `repeat` entails an immediate copy of the new configuration on L_5 and the start of a new cycle.

C. Tuning a few points

In order to complete the proof, we give an additional information on three points of the above process. We indicate how a state is copied, when and how the border is extended and, at last, how the transition table is copied.

1) *Copying a state:* We already indicated that the current state of a simul-cell ν is stored in a B_m around ν . The storage of the state is illustrated in figure 9, below. A path π is wrapped inside the B_m in a spiral, as indicated. Although π may be computed by an appropriate CA, we assume that it is given in any B_m devoted to represent a state. Note that for any cell on π which is not among its ends, exactly two cells among its neighbours are on π . Three colours are used in order to define the motion of the states. For instance, we may decide that the colours are, in this order from the centre of B_m to the other end of π : green, blue and red.

At synchronization time, the cells constituting the current state move all together to their goal. The order of the three colours allows to define the direction of the move at speed 1. Five auxiliary layers are used for this purpose.

At the target point, the first cell in motion meets the centre of the ball in which the state has to wrap around. It stops at this point and the other marks go on their motion along the path. They 'walk' over the already arrived marks until they meet a free place on the path. The free place is materialized by a blank cell on an appropriate layer.

2) *Extending the border:* We already know that the cells of the border are blank in the current configuration. This means that the value of k is chosen in such a way that this condition is satisfied at initial time. As the cell is on the border, it receives no information from outside the configuration. But the simul-cell knows that it is on the border thanks to the silver mark. And so, when the enlightening process starts in its table of transition, the information corresponding to the outside neighbours is supplied by the cell itself. Accordingly, there is no difficulty here.

Some care is needed when the transition table indicates that the next state of this cell is no more blank. This means that the border must be shifted by one F -level further from the central cell. We already know the algorithm for extending all the F -trees by one level and we know that the border simul-cell sends an appropriate signal to the central cell. But this is performed only when the extension is completed and in this process, the main task is the copying of the transition table, to which we now turn.

3) *Copying the transition table:* The first task of this copying process is to copy the tree structure. The idea is to put all the nodes of the tree into a list in such a way that

we can easily restore the tree from the list. For this purpose, a signal spreads over the tree, duplicated according to the degree of branching at each node where this is the case. During this process, the signal gives a special colour to branching nodes, say green, and another to leaves, say blue. The completion of the process can be known by a reverse signal from the leaves to the root. The branching nodes have just to wait the return signals from all their upper branches before sending back their own return signal.

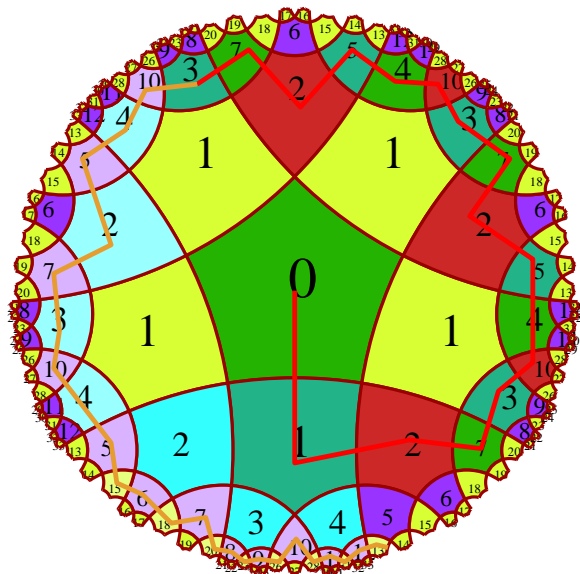


Figure 9: A current state wrapped in a B_m around the simul-cell. In this example it is state 17.

Below, figure 10 illustrates the marking of the tree.

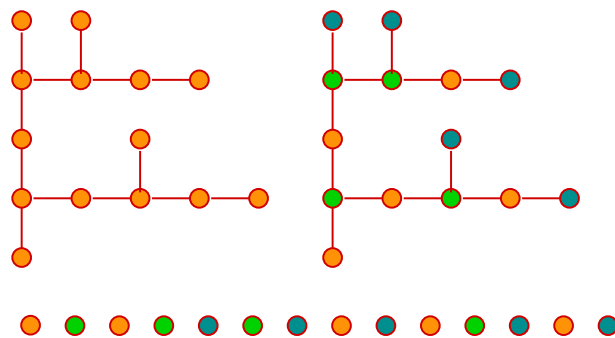


Figure 10: Marking a tree and the list obtained from the marking: on the right-hand side, the marking according the above rules. Below, the list: it contains first the whole leftmost branch, then what remains from the next branch to the right and so on.

Then, the branches go down one by one, from the leftmost to the rightmost. When the leftmost branch goes down, it constitutes the beginning of the list. When it passes the last branching point, the mark of the last branching point moves to the next leftmost branching point on the right. And this is repeated until there is no more branching point further. Then the mark goes backward down to the previous branching and

the search goes on recursively. The move of the mark is made in parallel with the motion of the branches. Note that the speed of this motion is only $\frac{1}{2}$ as there is a propagation of the start of the motion. This allows to construct the appropriate list.

The restoration of the tree from the list is a reverse process.

When the list arrives to the target, *i.e.* the root of the copy of the transition table, the first element stops at this point and the other marks 'walk' over already arrived elements, as in the process of copying a state. However, when the first leaf is reached, the progression of the list is blocked: this is also for this reason that we need a speed $\frac{1}{2}$. When the leaf is on its place, it sends back a return signal which will mark the first branching it meets. Then it goes down to the root in order to start again the progression of the list. The same process is repeated each time a leaf crosses the root.

Accordingly, we proved theorem 1.

D. Application to other tilings

It is not difficult to see that the same construction may be repeated *mutatis mutandis* for the ternary heptagrid, *i.e.* the tiling $\{7, 3\}$ of the hyperbolic plane. It can also be repeated in the same way for all the tilings $\{p, 4\}$ and $\{p+2, 3\}$ of the hyperbolic plane as, from [7] and [2], the same technique of localisation works in these tilings.

Taking into account that scaled trees also allowed us to implement transition tables which are not at all Fibonacci trees, we conclude that taking the initial parameter k big enough, we can implement any tree structure which would be fixed in advance. Accordingly:

Theorem 2: For any p, q such that $\frac{1}{p} + \frac{1}{q} < \frac{1}{2}$, there is a cellular automaton $U_{p,q}$ on the pentagrid, working on finite configurations, which is able to simulate any CA on the tiling $\{p, q\}$ of the hyperbolic plane, working on finite configurations.

From [8] it also follows that the same technique can be applied to the tiling $\{5, 3, 4\}$ of the hyperbolic 3D space.

Theorem 3: There is a cellular automaton U_3 on the pentagrid, working on finite configurations which is able to simulate any CA on the tiling $\{5, 3, 4\}$ of the hyperbolic 3D space, also working on finite configurations.

From [9], the same can be said of the tiling $\{5, 3, 3, 4\}$ of the hyperbolic 4D space.

Theorem 4: There is a cellular automaton U_4 on the pentagrid, working on finite configurations which is able to simulate any CA on the tiling $\{5, 3, 3, 4\}$ of the hyperbolic 4D space, also working on finite configurations.

V. CONCLUSION

The same technique could be applied to any combinatoric tiling as defined in [6], [9] which generalises the splitting technique indicated in section II.A. The spanning tree together with the language of the splitting defined in these papers allow to implement cellular automata in such a tiling, call them **abstract** cellular automata. Consequently, the scaled tree introduced in this paper can be used to construct an intrinsically universal cellular automaton on such a grid. Again applying the remark about the possibility to implement any tree in the pentagrid, we obtain that for any combinatoric tiling \mathcal{T} , there is a cellular automaton $U_{\mathcal{T}}$ on the pentagrid, working on finite configurations which is able to simulate any CA on \mathcal{T} , also working on finite configurations.

ACKNOWLEDGMENT

The author would like to thank Dr. Mark Burgin for inviting him to present this work at the session **Theoretical Foundations for Distributed and Concurrent Computations** at the 2006 International Conference on Foundations of Computer Science (**FCS'06**).

REFERENCES

- [1] C. Caratheodory, *Theory of Functions of Complex Variable*, vol. II, 177-184, Chelsea, NY, 1954.
- [2] K. Chelghoum, M. Margenstern, B. Martin, I. Pecci, Cellular automata in the hyperbolic plane: proposal for a new environment, *Lecture Notes in Computer Sciences*, **3305**, (2004), 678-687.
- [3] J. Durand-Lose, Intrinsic Universality of a 1-Dimensional Reversible Cellular Automaton, **STACS'1997**, (1997), 439-450.
- [4] M. Margenstern, New tools for cellular automata in the hyperbolic plane, *Journal of Universal Computer Science*, vol **6**, issue 12, (2000), 1226-1252.
- [5] M. Margenstern, K. Morita, NP problems are tractable in the space of cellular automata in the hyperbolic plane, *Theoretical Computer Science*, **259**, 99-128, (2001).
- [6] M. Margenstern, Revisiting Poincaré's theorem with the splitting method, talk at **Bolyai'200**, International Conference on Geometry and Topology, Cluj-Napoca, Romania, October, 1-3, 2002.
- [7] M. Margenstern, G. Skordev, Fibonacci Type Coding for the Regular Rectangular Tilings of the Hyperbolic Plane, *Journal of Universal Computer Science* **9**, N°5, (2003), 398-422.
- [8] M. Margenstern, G. Skordev, Tools for devising cellular automata in the hyperbolic 3D space, *Fundamenta Informaticae*, **58**, N°2, (2003), 369-398.
- [9] M. Margenstern, The tiling of the hyperbolic 4D space by the 120-cell is combinatoric, *Journal of Universal Computer Science*, **10**, N°9, (2004), 1212-1238.
- [10] N. Ollinger, The quest for small universal cellular automata, *Lecture Notes in Computer Sciences*, **2380**, (2002), 318-329.
- [11] N. Ollinger, The Intrinsic Universality Problem of One-Dimensional Cellular Automata, *Lecture Notes in Computer Sciences*, **2607**, (2003), 632-641.
- [12] Poincaré H., Théorie des groupes fuchsien. *Acta Mathematica*, **1**, 1-62, (1882).