A New Universal Cellular Automaton Discovered by Evolutionary Algorithms

Emmanuel Sapin[1], Olivier Bailleux[1], Jean-Jacques Chabrier[1], and Pierre Collet[2]

[1] Université de Bourgogne, 9 av. A. Savary, B.P. 47870, 21078 Dijon Cedex, France
emmanuelsapin@hotmail.com {olivier.bailleux,jjchab}@u-bourgogne.fr

[2] Laboratoire d'Informatique du Littoral, ULCO, Calais, France
Pierre.Collet@Univ-Littoral.Fr

Abstract. In *Twenty Problems in the Theory of Cellular Automata*, Stephen Wolfram asks "how common computational universality and undecidability [are] in cellular automata." This papers provides elements of answer, as it describes how another universal cellular automaton than the Game of Life (*Life*) was sought *and found* using evolutionary algorithms. This paper includes a demonstration that consists in showing that the presented R automaton can both implement any logic circuit (logic universality) and a simulation of *Life* (universality in the Turing sense). All the elements of the evolutionary algorithms that were used to find R are provided for replicability, as well as the analytical description in R of a cell of *Life*.

1 Introduction

Cellular automata are discrete systems [1] in which a population of cells evolves from generation to generation on the basis of local transitions rules. They can simulate simplified "forms of life" [2,3] or physical systems with discrete time, space and local interactions [4,5,6].

In [7], Wolfram studies the space $\mathcal{I}$ of isotropic two states 2D automata with a transition rule that takes into account the eight neighbours of a cell, to determine the cell's state at the next generation. He talks of a special automaton of $\mathcal{I}$ called *Game of Life* (hereafter referred to as *Life*) that was discovered by Conway in 1970 and popularised by Gardner in [2]. In [8], Conway, Berlekamp and Guy show that *Life* allows to compute any function calculable by a Turing machine. Their demonstration of the universality of *Life* uses *gliders*, *glider guns* and *eaters*. Gliders are patterns which, when evolving alone, periodically recover their original shape after some shift in space. A glider gun emits a stream of gliders that can be used to carry information. An eater absorbs gliders and, along with stream collisions, allows to create and combine logic gates into logic circuits. In [9], Rendell gives an explicit proof of the universality of *Life* by showing a direct simulation of counter machines.

K. Deb et al. (Eds.): GECCO 2004, LNCS 3102, pp. 175–187, 2004.

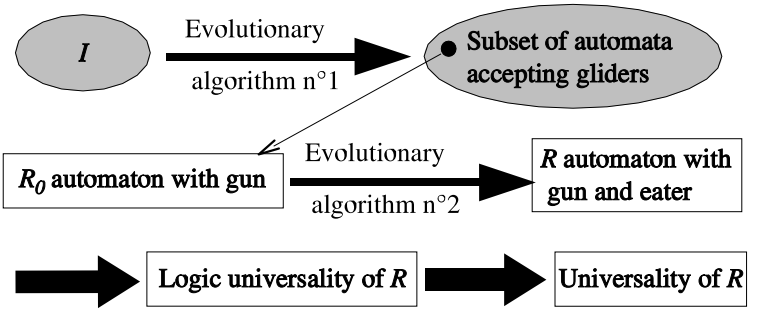

Fig. 1. Research approach used to find new universal automata.

In [10], Wolfram lists twenty problems on the theory of cellular automata. The sixteenth is introduced by the question "*How common are computational universality and undecidability in cellular automata ?*"

Up to now, *Life* was the only known dynamical universal automaton of $\mathcal{I}$: Margolus's cellular automaton billard-ball model [11] is universal but its transition rules take into account the parity of the generation number meaning that it is not in $\mathcal{I}$. Then, universal automata with more than two states [12,13] or more than two dimensions [14] are not in $\mathcal{I}$ either. Finally, Banks's 2D 2-state cellular automaton [15] is not dynamically universal because its wires are fixed.

This paper describes how another universal automaton of $\mathcal{I}$ was sought *and found* thanks to evolutionary algorithms. Section 2 describes the search process, that is developed in sections 3 and 4. Section 5 describes an analytical way to represent C.A.s, while sections 6 and 7 describe how patterns can be assembled into logic gates and how glider streams can be redirected and synchronised, in order to create logic circuits. Finally, section 8 presents a simulation of *Life* using the R automaton found in sections 3 and 4, and section 9 summarizes the presented results and discusses directions for future research.

2 Rationale

Figure 1 shows how new universal automata are sought. A first evolutionary algorithm is used to find in $\mathcal{I}$ a subset of automata accepting gliders.

In this subset, some automata accept glider guns. An automaton, called R_0, is chosen among them, on which attempts are made to demonstrate its universality.

In [8], Conway *et al.* used an eater to simulate a `NAND` gate. This observation is used as a starting point and an eater is sought among automata accepting the glider gun found in R_0, to be used as a building block to implement a `NAND` gate. This eater is found in an automaton called R, thanks to a second evolutionary algorithm.

The ability to create and assemble `NAND` gates into any logic circuit is then shown, which demonstrates the logic universality of R. Finally, *Life* is simulated

in R as a proof that R is universal in the Turing sense, since *Life* was shown to be able to implement registers cf. [8] and counter machines cf. [9].

3 Finding an Automaton Accepting Gliders

The search space of the evolutionary algorithm is the set $\mathcal{I}$ of 2 states 2D automata described in the introduction. An automaton can be described by telling what will become of a cell in the next generation, depending on its neighbours.

If symmetric and rotated neighbourhoods are considered as having an identical effect on a cell (isotropic automata), there are "only" 102 possible different neighbourhoods for one cell (and 2^{102} different automata). Therefore, an individual can be coded as a bitstring of 102 booleans (cf. fig. 2).

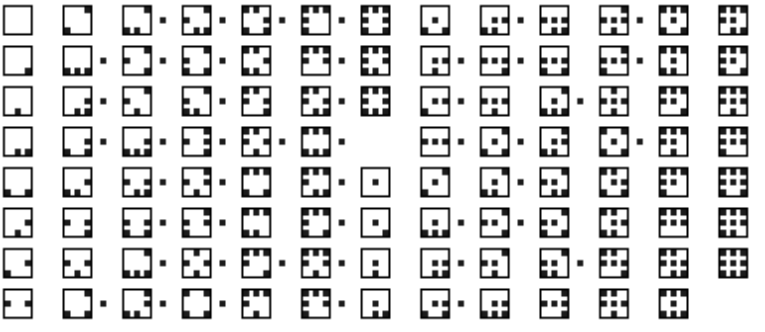

Fig. 2. A black cell on the right of the neighbourhood indicates a future central cell. The R automaton, can then be represented by 00000000011100011101111111111111110010010111111111000000000111011111011100001000101101000000000000000000000.

3.1 Description of the Evolutionary Algorithm That Searches Gliders

Fitness function. In order to find automata that accept gliders, the evolutionary algorithm attempts to maximise the number of gliders $\times$ the number of periodic patterns that appear during the evolution of a random configuration of cells by the tested automaton. (A more detailed description of the gliders and periodic patterns detector (inspired by Bays [16]) can be found in [17].)

Initialisation. The 102 bits of each individual are initialised at random.

Genetic operators. The mutation function simply consists in mutating one bit among 102, while the recombination is a single point crossover with a locus situated exactly on the middle of the genotype. This locus was chosen since the first 51 neighbourhoods determine the birth of cells, while the other 51 determine how they survive or die.

Evolution engine. It is very close to a $(\mu+\lambda)$ Evolution Strategy, although on a bitstring individual, and therefore without adaptive mutation: the population is made of 20 parents that are selected randomly to create 20 children by mutation only and 10 other by recombination. As in a straight ES+, the 20 best individuals among the 20 parents + 30 children are selected to create the next generation.

Stopping criterion. The algorithm stops after 200 generations, which, on a 800Mhz PC Athlon, takes around 20 minutes to complete.

3.2 The R_0 Automaton: An Experimental Result

The algorithm described above provided several automata accepting gliders. Among them, some would surprisingly generate glider guns for nearly every evolution of a random cell configuration.

One of them (called R_0) is picked up as a potential candidate for a universal automaton. Its 102 different neighbourhoods can be visually presented as in fig. 2 and fig. 3 shows the glider gun G_0 that appears spontaneously.

Fig. 3. An evolution of a random configuration of cell by R_0 showing G_0.

4 Looking for an "Eater" in Automata Accepting G_0

An eater is now needed in order to simulate a `NAND` logic gate, be it only because the glider gun of rule R_0 produces 3 too many glider streams that need to be suppressed to avoid interactions with other parts of a simulated logic circuit.

Ideal eaters are periodic patterns that, after the absorption of a glider, can resume their original shape and position, quickly enough to absorb another arriving glider (cf. fig. 4).

As no eater was found in R_0, a second evolutionary algorithm was elaborated to automatically find an eater in the space of all automata accepting the glider gun G_0. This space was determined, by finding deterministically which of the 102

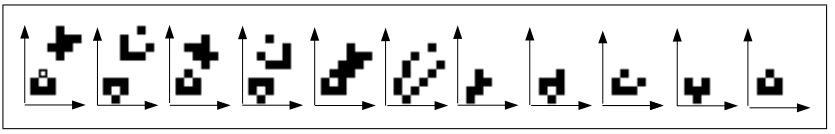

Fig. 4. Eater of R in action, found by a second evolutionary algorithm.

neighbourhoods of automaton R_0 were needed for gun G_0 to operate normally. It turns out that G_0 uses 81 different neigbourhoods, meaning that the output of the 21 other neighbourhoods could be changed in order to find an automaton R that would both accept the glider gun G_0 *and* an eater (cf. [15]).

An eater being a periodic pattern, a collection of 10 small periodic patterns of R_0 appearing frequently *and using only* the 81 neighbourhoods necessary for G_0 were chosen. Those periodic patterns were therefore sure to appear in all of the 2^{21} automata implementing G_0.

Finally, in order to find an eater, one needs to perform on the established collection of periodic patterns what could be called a crash test: each periodic pattern is positioned in front of a stream of gliders, and its fitness is simply the number of crashes it survives.

The number of possibilities being quite large (10 patterns to be tested in different relative positions with reference to the stream of gliders among 2^{21} different automata), a second evolutionary algorithm was therefore elaborated, with the following characteristics:

Individual structure. An individual is made of:

1. a bitstring of 21 bits determining one automaton among 2^{21},
2. an integer between 1 and 10, describing one pattern among the 10 periodic patterns common to the 81 neighbourhoods needed by G_0,
3. the relative position of the pattern relatively to the stream of gliders, coded by two integers, x and y, varying between $[-8, 8]$ and $[0, 1]$.

Individuals are initialised with R_0, a random integer between 1 and 10 and randomly within their interval for x and y.

Fitness function. Number of gliders stopped by an individual.

Genetic operators. The only operator is a mutator, since no really "intelligent" recombination function could be elaborated. The mutator is therefore called on all created offspring and can either choose any pattern among the 10 available, or mutate one bit in the bitstring, or move the position of the pattern by ± 1 within the defined boundaries for x and y.

Evolution Engine. It is this time closer to an Evolutionary Programming Engine, since it has no crossover, although the EP tournament was not implemented. 30 children are created by mutation of 20 parents selected uniformly. Among the 50 resulting individuals, the best 20 individuals are selected to create the next generation.

Stopping criterion. Discovery of an eater that would survive 50 000 collisions.

This algorithm allowed to discover the automaton R described in fig. 2, accepting *both* the glider gun G_0 and the eater shown in fig. 4.

Interestingly enough, other runs took between 1 and 20 minutes to complete, to always find the same eater pattern, although with different automata.

5 Analytical Description of a CA

Binary numbers can be implemented as a finite stream of gliders, where gliders represent 1s and missing gliders represent 0s.

The next step needed to prove the logic universality of R is to find a configuration of glider guns and eaters that can simulate a `NAND` gate on two streams of gliders representing two binary numbers.

Unfortunately, the cellular autamaton implementing a `NAND` gate would be too difficult to represent and explain by showing groups of cells on a grid, let alone a CA implementing a cell of *Life*. Therefore, a much clearer analytical description was needed, that should also allow replicability of the contents of this paper.

In order to simplify the representation of a CA, one can replace its building blocks by an analytical description, made of a letter, referring to the pattern, followed by three parameters (D, x, y) where D denotes a direction (North, East, South, West) and x, y the coordinates of a specific cell of the pattern (cf. [18]). An arrow is added in graphic descriptions to help visualising the CA.

Several patterns and their analytical representation are described in this section, namely a glider stream, an eater, a glider gun and a large glider gun:

Glider stream. Fig. 5 shows a glider stream $S(E, x, y)$, where x and y are the coordinates of the white cell whence an arrow is shooting.

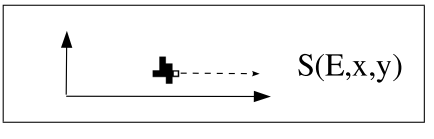

Fig. 5. A glider stream and its analytical representation $S(E, x, y)$.

Eater. The eater of fig. 4 can be identified as $E(N, x, y)$, where N denotes its northward orientation and x, y denote the position of the white cell.

Glider gun. The glider gun of fig. 3 is unfortunately not usable as is, because it shoots gliders spaced every nine cells only. The complex guns shown in fig. 6 shoot gliders spaced by 45 cells, which gives more slack to work on streams. Fig. 6 shows instances of this gun at generation 2, used later on in this paper, namely $Ga(S, x, y)$ and $Gb(S, x, y)$. The guns in other cardinal directions are obtained by rotation thanks to the isotropy of R.

Large glider gun. Another type of glider, called *large glider*, appears in R. A large gun (L) shooting a large glider every 45 cells, is made of two complex guns G, shooting their stream perpendicularly (cf. fig. 7).

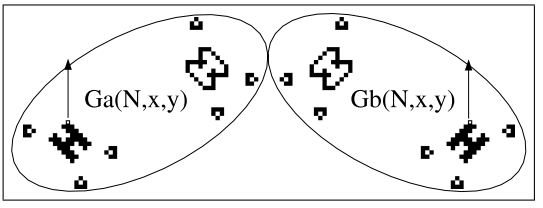

Fig. 6. Symmetric complex guns of R, viewed at generation 2. x, y are the coordinates of the white cell, whence an arrow is shooting.

When both streams collide, large gliders are created that sail on the direction of the stream of the top left gun. The large glider gun of fig. 7 will be referred to as $L(E, 16, 80)$, as it was decided that its coordinates would be those of the top left gun. The shape of a large glider is shown on the right of the gun.

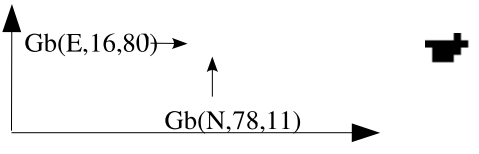

Fig. 7. Schematics of a large glider gun $L(E, 16, 80)$, made of two complex Gb guns.

6 Assembling Patterns into a `Not` Gate

[18] describes the simulation of an `AND` gate in R. Unfortunately, it is `NAND` gates that allow to build any logic circuits, and not `AND` gates. It is therefore very important to build a `NOT` gate for the R automaton to be logically universal. Before the description of the `NOT` gate begins, it is important to observe that:

1. The frontal collision of two identical large glider streams produces two orthogonal standard glider streams equal to the input streams. If only one output glider stream is needed, an eater can be placed in front of the second stream —cf. fig. 8 and $L(E, 16, 80)/L(W, 379, 80)/E(N, 217, 65)$ in fig. 9.
2. The collision between two standard gliders at a right angle, or between a standard glider and a large glider at a right angle destroys both gliders. Therefore, a glider gun (large or complex) G positioned so that it fires gliders at a right angle towards another stream of standard gliders A will complement the stream at a right angle: whenever a glider of stream A (symbolising 1) arrives at the collision point, the glider of stream A and the glider of G disappear (resulting in a 0). On the contrary, an absence of glider in stream A (symbolising 0) will let a glider of G get through the collision point (resulting in a 1) cf. second collision of fig. 8.

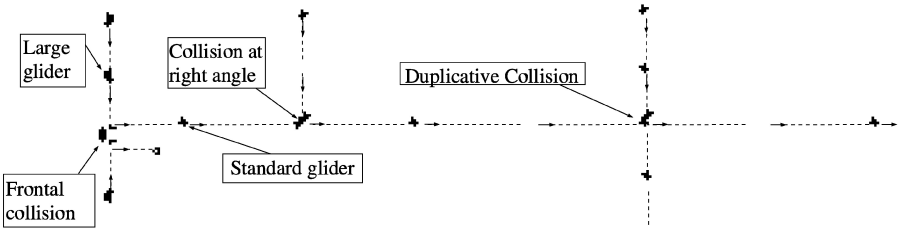

Fig. 8. This figure presents the three types of collisions used in this paper. 1's are represented by gliders, and 0's by an absence of gliders. The figure starts on the left with a frontal collision between two streams of large gliders, that creates a stream of standard gliders going towards the East. In this example, the stream then collides with a periodic stream of 101010... coming from the North at a 90 degrees angle, with the consequence that the information it carries is complemented and turned by 90 degrees into a horizontal eastward stream of 010101.... This stream finally hits another vertical stream made of ones coming from the North although slightly displaced, therefore creating a duplicating collision. The left to right 010101... stream goes through the duplicative collision unmodified, while a complemented stream 101010... is created and issued southwardly.

3. When the right angle stream of gliders is slightly displaced w.r.t. the initial stream, a collision of gliders does not result in a destruction of gliders. Instead, the initial glider goes through the collision, while the glider coming at a 90 degrees angle disappears, duplicating the stream into a complemented stream at a 90 degrees angle —cf. fig. 8 and $Gb(E, 179, 200)$ in fig. 9.

Thanks to these three observations, a `NOT` gate is built on fig. 9 below:

On this figure, the information stream A ($S(S, 210, 184)$) is shown as a dotted line. A complex glider gun $Gb(E, 179, 200)$ creates a complementary duplicate stream $\overline{A}$ towards the East. The two outputs are redirected by glider guns $Gb(W, 253, 142)$, $Ga(S, 159, 234)$ and $Ga(S, 273, 262)$ until they are vertical again. Then, they are complemented into large gliders by the two large glider guns $L(E, 16, 80)$ and $L(W, 379, 80)$. When the frontal collision between the two large glider streams occurs, a complementary stream $\overline{A}$ is created towards the same direction as the original A stream while the other one is "eaten" by $E(N, 217, 65)$.

7 Intersection and Synchronisation of Streams

In order to prove the logic universality of R, one needs to combine several `NAND` gates. This is possible if two streams in any position can be redirected in order to become the input streams of a `NAND` gate. This operation can be realised thanks to intersection and synchronisation patterns.

Intersection. Thanks to complex guns, gliders in a stream are separated by 45 cells. This means that it is possible to have two streams cross each other

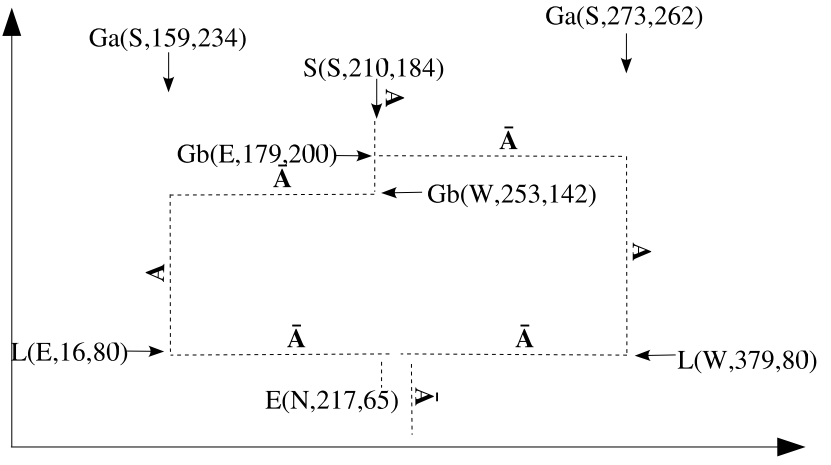

Fig. 9. Complementation of stream A: the analytical representation for the NOT gate is $\{L(E,16,80), Ga(S,159,234), Gb(E,179,200), S(S,210,184), E(N,217,65), Gb(W,253,142), Ga(S,273,262), L(W,379,80)\}$.

without any interference. If, for a synchronisation reason, interference cannot be avoided, [19] shows how a stream intersection can be realised by a cunning combination of `NAND` gates.

Synchronisation. It is important to be able to delay one stream w.r.t another, in order to synchronise them properly just before they enter a logic gate, for instance. This can be done precisely by diverting four times the stream to be delayed with orthogonal guns (cf. fig. 10).

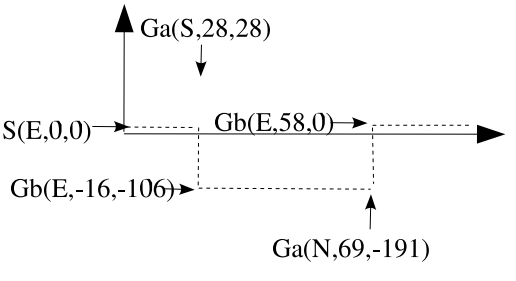

Fig. 10. Stream temporisation for synchronisation purposes.

Stream duplication, redirection, synchronisation and intersection allow to combine any number of `NAND` gates together, therefore proving the logic universality of R.

8 Simulation of *Life* in *R*

The previous sections have proved that R was universal in the logic sense (i.e.: it can implement any logic circuit). In order to show that R is universal *in the Turing sense*, one must find memory structures like registers within R. Since this was done in [8] and [9] for *Life*, finding a simulation of *Life* in R will prove the universality of R in the Turing sense.

To simulate *Life*, one must first find in R a simulation of a cell of *Life*, and then a way to tile a surface with any number of interconnected cells.

8.1 Simulation of a Cell of *Life*

Since it has been shown that any logic circuit can be implemented in R, a single cell of *Life* can be implemented as a boolean function computing the value of a cell S at generation $n+1$ from the value of its eight neighbours $C_1 \dots C_8$ at generation n.

The rules of *Life* are the following: a "living" cell dies at the next generation unless it has two or three neighbours. A dead cell comes alive at the next generation iff it has three neighbours in the current generation.

Supposing that the addition of $C_1 + \ldots + C_8$ gives a three bit number $n_2 n_1 n_0$, the rules of *Life* can be simply expressed by the formula $S_{n+1} = \overline{n_2}.n_1.(S_n + n_0)$, which can be translated into a combination of `NAND` gates. This function, implementing a cell of *Life*, is represented by the grey area of fig. 11 below, and analytically described in R in the appendix.

8.2 Interconnecting Cells of *Life*

In order to simulate *Life* in R, proof must now be given that it is possible to tile a surface with identical cells, each interconnected with their 8 neighbours.

All cells being identical, the inputs of a cell must physically correspond to the outputs of its neighbours. Therefore, the way a cell receives the state of its neighbours can be induced from the way it sends its own state to its neighbours, which is what is described below and in fig. 11.

It is straightforward for a cell to send its state to its cardinal neighbours C_2, C_4, C_5, C_7. Sending its state to neighbours C_1, C_3, C_6, C_8 is however more tricky, since those neighbours are situated diagonally. This is done by passing the information to their neighbours C_2 and C_7. Therefore, one can see in fig. 11 that the state S of the cell is sent three times to C_2 and three times to C_7, so that C_2 (resp. C_7) can keep one stream for its own use, and pass the two others to its horizontal neighbours, C_1 and C_3 (resp. C_6 and C_8).

S being itself a top neighbour of cell C_7, one sees how the state of C_7 is passed over to C_4 and C_5 in the same way that C_2 will pass over the information of the state of S to C_1 and C_3.

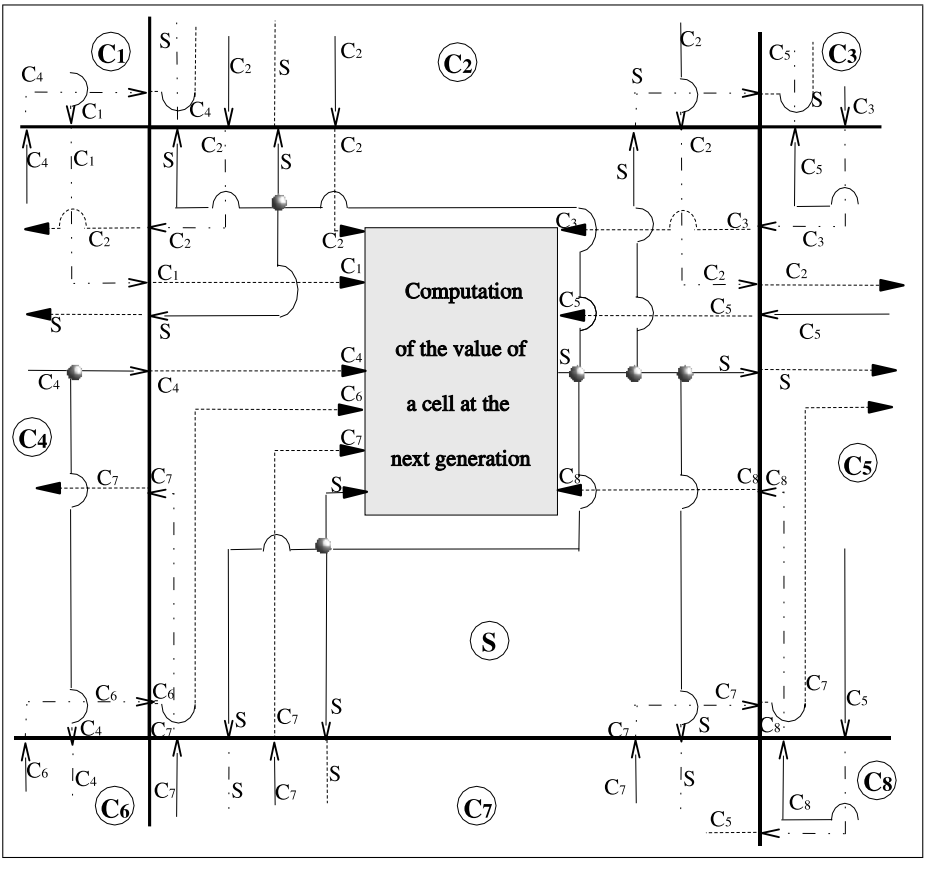

Fig. 11. Diagram of the simulation of a cell of *Life*.

9 Synthesis and Perspectives

An extensive bibliographic research seems to show that this paper actually presents the first proof that another 2D 2 state dynamical universal (in the Turing sense) automaton other than the famous *Life* exists in $\mathcal{I}$, therefore providing an element of answer to Wolfram's 16th problem.

Evolutionary algorithms played a key role in discovering gliders, and a rule R accepting a glider gun and eaters, in a very large search space.

Further goals are now to give a more complete answer to Wolfram's problem by finding whether other universal automata than *Life* and R exist, and how common they are. Then, another domain that seems worth exploring is how this approach could be extended to automata with more than 2 states.

Finally, the study of the construction of an automatic system of selection / discovery of such type of automata based on evolutionary algorithms is far more interesting, as it could lead to a new classification.

References

1. S. Wolfram. Universality and complexity in cellular automata. *In Physica D*, 10:1–35, 1984.
2. M. Gardner. The fantastic combinaisons of john conway's new solitaire game "life". *In Scientific American*, 1970.
3. M. Gardner. On cellular automata, self-reproduction, the garden of eden, and the game of life. *In Scientific American*, 224:112–118, 1971.
4. C. Dytham and B. Shorrocks. Selection, patches and genetic variation: A cellular automata modeling drosophila populations. *Evolutionary Ecology*, 6:342–351, 1992.
5. I. R. Epstein. Spiral waves in chemistry and biology. *In Science*, 252, 1991.
6. Ermentrout, G. Lotti, and L. Margara . Cellular automata approaches to biological modeling. *In Journal of Theoretical Biology*, 60:97–133, 1993.
7. S. Wolfram N.H. Packard. Two-dimensional cellular automata. *In Journal of Statistical Physics*, 38:901–946, 1985.
8. E. Berlekamp, J.H Conway, and R.Guy. Winning ways for your mathematical plays. *Academic press, New York*, 1982.
9. P. Rendell. Turing universaility of the game of life. *Andrew Adamatzky (ed.), Collision-Based Computing, Springer Verlag.*, 2002.
10. S. Wolfram. Twenty problems in the theory of cellular automata. *In Physica Scripta*, pages 170–183, 1985.
11. N. Margolus. Physics-like models of computation. *In Physica D*, 10:81–95, 1984.
12. K. Lindgren and M. Nordahl. Universal computation in simple one dimensional cellular automata. *In Complex Systems*, 4:299–318, 1990.
13. K. Morita Y. Tojima I. Katsunobo T. Ogiro. Universal computing in reversible and number-conserving two-dimensional cellular spaces. *Andrew Adamatzky (ed.), Collision-Based Computing, Springer Verlag.*, 2002.
14. A. Adamatzky. Universal dymical computation in multi-dimensional excitable lattices. *In International Journal of Theoretical Physics*, 37:3069–3108, 1998.
15. E. R. Banks. *Information and transmission in cellular automata.* PhD thesis, MIT, 1971.
16. C. Bays. Candidates for the game of life in three dimensions. *In Complex Systems*, 1:373–400, 1987.
17. E. Sapin, O. Bailleux, and J.J. Chabrier. Research of complex forms in the cellular automata by evolutionary algorithms. *In EA03.Lecture Notes in Computer Science*, 2936:373–400, 2004.
18. E. Sapin, O. Bailleux, and J.J. Chabrier. Research of a cellular automaton simulating logic gates by evolutionary algorithms. *In EuroGP03.Lecture Notes in Computer Science*, 2610:414–423, 2003.
19. A. Dewdney. The planiverse. *Poseidon Press*, 1984.

Appendix

This appendix contains an analytical description of a cell of the game of life in the R automaton, for replicability. Let us define X, Y and Z as:

$X(S, 212, 65) = \{Gb(E, 18, 91), Gb(N, 80, 25), Gb(S, 212, 65), Gb(W, 241, 36), Gb(W, 253, 142), Gb(N, 370, 23), Gb(W, 433, 93), Ga(S, 159, 234), Gb(E, 179, 200), Ga(S, 273, 262), S(S, l, 210, 184), E(E, 292, 159), E(S, 219, 78)\}$,

$Y(E,1,1)$=$\{Gb(E,1,1), Ga(S,99,189), Ga(S,173,520), Gb(W,184,104), E(E,183,484)\}$
$Z(E,1,1) = \{Y(E,1,1), X(S,425,155), Gb(E,721,91), Gb(W,843,213), Ga(S,749,299), Ga(S,839,630), Ga(S,1019,749), Gb(W,1246,261), E(E,1229,276), Gb(E,990,180), Gb(E,965,277), Ga(S,1071,350), Ga(S,1145,457), E(E,1161,394), Ga(S,423,455), E(S,427,-47)\}$.

A cell of the game of life can be described in R as:

LifeCell$(E,1259,-436) = \{Z(E,1,1), Z(E,-759,486), Z(E,-1519,971), Z(E,-2279,1456), Z(E,-3039,1941), Z(E,-3799,2426), Z(E,-4559,2911), Y(E,-269,-89), Y(E,-1028,397), Y(E,-1298,127), Y(E,-1787,883), Y(E,-2057,613), Y(E,-2327,343), Y(E,-2546,1369), Y(E,-2816,1099), Y(E,-3086,829), Y(E,-3356,559), Y(E,-3305,1855), Y(E,-3575,1585), Y(E,-3845,1315), Y(E,-4115,1045), Y(E,-4385,775), Y(E,-4064,2341), Y(E,-4334,2071), Y(E,-4604,1801), Y(E,-4874,1531), Y(E,-5144,1261), Y(E,-5414,991), Y(E,-4823,2827), Y(E,-5093,2557), Y(E,-5363,2287), Y(E,-5633,2017), Y(E,-5903,1747), Y(E,-6173,1477), Y(E,-6443,1207), Ga(S,843,33), Ga(S,933,123), Ga(S,1203,213), X(S,1205,-84), Gb(E,663,-147), Y(E,719,-437)\}$.

The input streams must arrive at coordinates –6227,988 for Sn, –5957,1258 for $C8$, –5687,1528 for $C7$, –5417,1798 for $C6$, –5147,2068 for $C5$, –4877,2338 for $C4$, –4607,2608 for $C3$, –4427,2788 for $C2$, –4337,2878 for $C1$.
The stream for $Sn+1$ comes out at coordinates 1259,–436 after 21600 generations.