Asymptotic Cellular Complexity

Bruno Durand and Victor Poupet*

Laboratoire d'informatique fondamentale de Marseille (LIF)
Aix-Marseille Université, CNRS

Abstract. We show here how to construct a cellular automaton whose asymptotic set (the set of configurations it converges to) is maximally complex: it contains only configurations of maximal Kolmogorov complexity. This cellular automaton hence exhibits the most complex possible asymptotic behavior.

1 Introduction

Attractors are one of the key concepts in the study of dynamical systems. Discrete complex systems have the same concern and their asymptotic behavior has been the subject matter of many publications in theoretical computer science.

Cellular automata form the simplest of these complex systems and can produce complex behaviors from a finitely defined local transition rule. Many different classifications have been proposed, several of which based on their asymptotic behavior [12, 4, 8, 2].

Litterature usually considers *limit sets* as attractors; they have many interesting properties (topological and algorithmic). However they contain some transient configurations. In this article we work both on limit sets and on a tighter definition: the *asymptotic set*. It was recently proved by Cervelle [3] that some cellular automata have linearly complex asymptotic sets but the theoretical bound on the complexity of two-dimensional cellular automata is quadratic. We show here that an optimal quadratic bound can indeed be achieved.

Our construction is inspired by the proof of a complexity result on tilings of the plane [5]. Nevertheless, there is a major difference: tilings are "static" models and only linear complexity can be obtained. Taking advantage of the dynamical aspect of cellular automata gives an improvement in the complexity to n^2.

The article is organized as follows: Section 2 recalls some definitions and properties about cellular automata, asymptotic behavior and Kolmogorov complexity. Section 3 states the main theorem and proves it by detailing the construction of a cellular automaton of high complexity. Section 4 discusses possible extensions and improvements of the result.

2 Definitions

Definition 1 (Cellular Automaton). *A cellular automaton (CA) is a discrete dynamical system defined by a quadruple* $\mathcal{A} = (d, \mathcal{Q}, V, \delta)$ *where* $d \in \mathbb{N}$ *is*

* {bruno.durand,victor.poupet}@lif.univ-mrs.fr

the dimension of the CA, $\mathcal{Q}$ is a finite set called set of states*, $V \subseteq \mathbb{Z}^d$ is a finite set called* neighborhood *and $\delta : \mathcal{Q}^V \to \mathcal{Q}$ is the local transition function of the automaton.*

For a given automaton $\mathcal{A} = (d, \mathcal{Q}, V = \{v_1, \ldots, v_n\}, \delta)$, we call *configuration* any mapping $\mathfrak{C} : \mathbb{Z}^d \to \mathcal{Q}$. Elements of $\mathbb{Z}^d$ are called *cells*. Given a configuration $\mathfrak{C}$, we say that a cell c is in state q if $\mathfrak{C}(c) = q$. From the local function δ we define a global function $\Delta : \mathcal{Q}^{\mathbb{Z}^d} \to \mathcal{Q}^{\mathbb{Z}^d}$ associating any configuration C with the configuration $\mathfrak{C}'$ such that

$$\forall x \in \mathbb{Z}^d, \mathfrak{C}'(x) = \delta(\mathfrak{C}(x + v_1), \ldots, \mathfrak{C}(x + v_n))$$

In this article we mainly consider two-dimensional CA ($d = 2$). We work on the Moore neighborhood (9 nearest neighbors), although the choice of the neighborhood has little relevance to our construction.

Definition 2 (Spreading State). *A CA $\mathcal{A} = (d, \mathcal{Q}, V, \delta)$ is said to have a* spreading state *q_s if, as soon as one of the neighbors of a cell is in state q_s at time t, then it becomes in state q_s at time $(t + 1)$.*

Definition 3 (Asymptotic Set). *The* asymptotic set *of a CA $\mathcal{A}$ is the union of the accumulation points*[1] *of all its orbits:*

$$\mathfrak{U}(\mathcal{A}) = \bigcup_{\mathfrak{C} \in \mathcal{Q}^{\mathbb{Z}^d}} \bigcap_{n \in \mathbb{N}} \overline{\{\Delta^i(\mathfrak{C})\}}_{i \geq n}$$

Informally, a configuration $\mathfrak{C}$ is in the asymptotic set of $\mathcal{A}$ if there is a configuration $\mathfrak{C}_0$ such that the orbit of $\mathfrak{C}_0$ has infinitely many configurations that coincide with $\mathfrak{C}$ on arbitrarily large areas around the origin.

This definition is different from the standard definition of a *limit set* $\Lambda(\mathcal{A}) = \bigcap_{n \in \mathbb{N}} \Delta^n(\mathcal{Q}^{\mathbb{Z}^d})$. The limit set has nice topological properties (in particular it is a closed set) but contains, in addition to all the asymptotic configurations, some that disappear after some time[2]. The asymptotic set more accurately captures the asymptotic behavior of the automaton.

In the sequel, we use the following two lemmas (the proofs are very straightforward from the definitions):

Lemma 1. *There is always a uniform configuration in the asymptotic set of a cellular automaton. Moreover, $\Delta(\mathfrak{U}) = \mathfrak{U}$.*

Lemma 2. *If a cellular automaton has a spreading state $\square$, the uniformly $\square$ configuration is in the asymptotic set of the automaton. Moreover, the $\square$ state does not appear in any other configuration of the asymptotic set.*

[1] We consider the usual topology induced by the Cantor distance: the distance between two configurations $\mathfrak{C}_1$ and $\mathfrak{C}_2$ is 2^{-k} where $k = \min\{\|x\| \mid \mathfrak{C}_1(x) \neq \mathfrak{C}_2(x)\}$.

[2] Consider for instance a one-dimensional CA with two states 0 and 1 for which each cell takes the maximal of the states in its neighborhood. Finite segments of 0 disappear over time but configurations of the form ${}^{\omega}10^n1^{\omega}$ have pre-images of any order and are hence in the limit set.

Definition 4 (Kolmogorov Complexity). *Given a recursive function f, the Kolmogorov complexity relative to f of a string $x \in \{0,1\}^*$ is defined as $K_f(x) = \min\{|y| \mid f(y) = x\}$ (it is infinite if the set $\{|y| \mid f(y) = x\}$ is empty).*

The idea is then to drop the recursive function f. This can be done by using the founding Kolmogorov theorem: there exists a recursive function U (called additively optimal) such that for any recursive function f, there is a constant $c_f \in \mathbb{N}$ such that for any string $x \in \{0,1\}^*$ we have $K_U(x) \leq K_f(x) + c_f$. The Kolmogorov complexity of a string x is then denoted as $K(x) = K_U(x)$ for some additively optimal U.

Informally, the Kolmogorov complexity of an object is its shortest possible description (after choosing the description language). In the following we will talk about Kolmogorov complexity of square patterns over a finite alphabet (not necessarily $\{0,1\}$). It is defined by choosing an encoding of these patterns into words over $\{0,1\}$ and considering the Kolmogorov complexity of the resulting encoding. The chosen encoding does not matter since it only affects the result up to an additive constant.

We will use the following well known properties of the Kolmogorov complexity:

Proposition 1. *There exists a constant c such that any string x has complexity less than $(|x|+c)$ and any square pattern of dimension $n \times n$ over a finite alphabet $\mathcal{Q}$ has complexity at most $(\log_2(|\mathcal{Q}|).n^2 + c)$.*

For all $\rho \in \mathbb{R}$ it is possible to enumerate all strings x such that $K(x) \leq \rho.|x|$.

Definition 5 (ρ-complexity). *Given a constant $\rho > 0$, a square pattern of size $n \times n$ on an alphabet $\mathcal{Q}$ is said to be* ρ-complex *if its Kolmogorov complexity is greater than $\rho.n^2$.*

A configuration $\mathfrak{C} \in \mathcal{Q}^{\mathbb{Z}^2}$ of the plane is ρ-complex if there exists a constant n_0 such that all square patterns of size $n \times n$ for $n \geq n_0$ are ρ-complex.

Square patterns and configurations that are not ρ-complex are said to be ρ-simple.

3 Main Theorem

This whole section is dedicated to the proof of the following theorem:

Theorem 1. *For any constant $0 < \rho < 1$ there exists a CA $\mathcal{A}$ whose asymptotic set contains infinitely many ρ-complex configurations and only one ρ-simple configuration.*

For practical and readability reasons, we will not give here a complete formal proof of the theorem. We will instead describe the construction of a cellular automaton that verifies the announced property and show how to solve the different difficulties that arise. A good understanding of the main ideas should be enough to convince the reader that a complete formal proof can indeed be derived.

Intuitively, for this constructed CA, all configurations of the asymptotic set are complex, except the unavoidable constant one. Its limit set however has more than these configurations as it can also have "hybrid" configurations that are a reunion of complex areas (areas that can be extended into ρ-complex configurations) and uniform areas.

3.1 Layered Structure

About Layers. The automaton will be described as a finite superposition of "layers". Saying that a cellular automaton has n layers means that its set of states is the cartesian product of n finite sets of "sub-states" ($\mathcal{Q} = \mathcal{Q}_1 \times \mathcal{Q}_2 \times \ldots \times \mathcal{Q}_n$). Layers evolve simultaneously. Some layers can work independently of the others (the sub-state of a cell only depends on the corresponding sub-states of its neighbors) while some will evolve differently according to what lies on the other layers.

Layers are mainly used to simplify the description of a cellular automaton: we sometimes need to perform different computations on the same set of cells, in which case we use a layer for each of the computations. If there are only finitely many computations to perform on each cell (and the bound is uniform) the set of states of the automaton thus constructed by layers remains finite.

We use three main layers and a spreading state $\square$. The set of states of the automaton is therefore $\mathcal{Q} = \{\square\} \cup (\mathcal{Q}_B \times \mathcal{Q}_R \times \mathcal{Q}_C)$ where $\mathcal{Q}_B$, $\mathcal{Q}_R$ and $\mathcal{Q}_C$ are the finite sets of sub-states corresponding to each layer.

The Bitmap Layer. The first layer is a very simple one. It contains only one bit ($\mathcal{Q}_B = \{0, 1\}$) on each cell and this bit does not change when the automaton evolves (except if the $\square$ state propagates over all layers of course). It is nonetheless very important to our automaton because it is where the complexity will be. The whole automaton will then be designed to search for ρ-simple square patterns on this layer. If one is found then a $\square$ state will appear and the configuration will not be in the asymptotic set.

Conversely, if none of the patterns is ρ-simple then no $\square$ state will appear and there will be a configuration in the asymptotic set whose bitmap layer is the one considered. Such a configuration is ρ-complex no matter what information lies on the other layers.

The Robinson Layer. The second layer is a Robinson tiling. This means that each cell of the automaton is associated to one of the Wang tiles of the considered set. $\mathcal{Q}_R$ is therefore a set of Wang tiles. This layer does not change over time either, but each cell checks locally the correctness of the tiling. If a cell sees an error in its surrounding, a $\square$ state is generated and, as seen before, the configuration is not in the asymptotic set. The asymptotic set can hence only contain configurations for which the Robinson layer contains a valid tiling of the plane. Because this layer does not change over time, we will use the properties of the Robinson tiling to define square areas on which a computation can be realized.

The Computation Layer. This last layer is the one where the real computation takes place. It will use the squares of the Robinson layer to define areas on which separate computations will enumerate ρ-simple patterns and look for them on the bitmap layer. When a ρ-simple pattern is found, a spreading state $\square$ is produced and the whole configuration is thus removed from the asymptotic set.

3.2 The Robinson Hierarchy

Properties of the Robinson Tiling. Fully describing the construction of a Robinson tiling and proving its properties is way beyond the scope of this article. Extensive literature has been written on the subject and any reader unfamiliar with the techniques but eager to learn more about it should refer to the books and articles presented in the bibliography [10, 6, 1]. We will only focus on the general properties of valid tilings without discussing how such properties are enforced locally by a set of Wang tiles.

The construction we will be using here is the "strong Robinson tiling" presented in [9]. Using this construction we can add decorations to the tiles (we "draw" lines on each tile) such that any valid tiling of the plane exhibits a self-similar structure of squares of different sizes. More precisely, in such strong Robinson tilings, all squares of a given size are aligned, while in the original Robinson tiling, it is not always the case because of the degenerate case where a fracture line appears.

Robinson Squares. The size of a square is the length of its sides. Any valid tiling has squares of size 4^n for all n. Squares of the same size are regularly arranged on a grid (rows and columns of aligned squares). The space between two adjacent squares of size 4^n is exactly 4^n. Moreover, grids of squares of different sizes are placed the ones relatively to the others in such a way that each square of size 4^{n+1} contains four squares of size 4^n. Figure 1 illustrates a portion of a valid Robinson tiling (filled and hatched areas should be ignored for now).

Infinite Squares. An important property of the Robinson squares that we will be using later is that each row and each column of the plane contains sides of squares of exactly one size: two squares of different sizes cannot have aligned sides and there are square sides on every row and column. A special case is when all squares are placed in such a way that a line or column has no finite-sized square side on it. In this case, an "infinite square" fills the empty row or column: if there was only an empty column (or an empty row), a bi-infinite line goes through it: it is the side of an infinite square. If both a line and a column are left empty, two perpendicular semi-infinite lines form the angle of an infinite square.

Computation. Robinson squares will be required to perform some computational tasks. The computation will take place on the lower side of the square.

For practical reasons, we will ignore squares that are too small to start a correct computation. We will therefore only consider squares of size greater than $s_0 = 4^{k_0}$ for some s_0 that should be chosen after explaining the whole construction to see how much space a segment needs to correctly start its computation. From now on, Robinson squares of size 4^{k_0+n} will be called squares of level n (or simply n-squares). Squares of level 0 are the smallest ones that we will consider.

The Life Cycle of a Robinson Square. Each Robinson square (large enough since we are ignoring the smaller ones) will perform a computation on its lower segment. Because this segment is finite, the duration of a computation before entering a loop is also limited. To make sure that each square does its verification correctly, we will reset the whole computation periodically. To do so, we implement a binary counter on the lower side of each square that starts on the bottom left corner and grows towards the bottom right one. When the counter reaches the bottom right corner of the square, a signal resets the counter and the computation that takes place on the square.

Incrementation signals are sent by all bottom left angles of all Robinson squares. Any Robinson square of size s will be reset by its counter after at most 2^{s+1} steps no matter what computation it was initially performing (if any). From there we know that the square behaves as intended from a freshly initialized configuration. The duration of a cycle is exponential in the size of the square, and we will show that it is sufficient to perform the tasks that we need, which require a polynomial time.

3.3 Reading the Bitmap Layer

Some cells of the plane are inside infinitely many Robinson squares. For this reason we cannot allow each square of any possible size to access directly the bitmap layer because this would lead to an unbounded number of competitive requests on some cells. Instead, only the smallest considered Robinson squares (level 0) will access directly the bitmap layer. All other squares will query recursively the squares of lower level to obtain the required information.

Visibility and Responsibility. Let us define the *visibility area* of a Robinson square of size s as the square surface of size $2s$ centered on it (see Figure 1). Because of the regular arrangement of Robinson squares, for each k, any bit of the bitmap layer is in the visibility area of exactly one k-square. Moreover, the visibility area of a k-square is exactly the union of that of the 16 closest $(k-1)$-squares.

Visibility areas are very convenient because, at a given level, they form an exact partition of the plane. They are, however, not sufficient for our purpose.

The problem comes from the special Robinson tilings described earlier that contain some "degenerate" squares of infinite size. This means that in some cases, some square patterns are not included in the visibility area of any finite Robinson square.

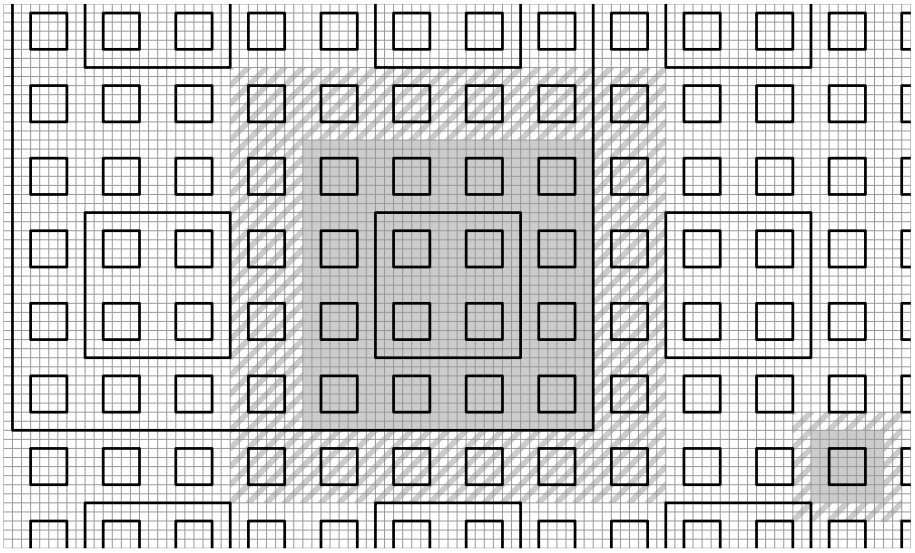

Fig. 1. The visibility area (grey) and the responsibility area (hatched) of two Robinson squares.

To solve this problem, we will have to widen the area over which a square looks for simple patterns. We define the *responsibility area* of a Robinson square of size s as the square surface of size $3s$ centered on it (see Figure 1). Of particular interest is the fact that the responsibility area of a k-square is exactly the disjoint union of the visibility areas of the 36 closest $(k-1)$-squares. Note that responsibility areas of neighboring squares of the same level overlap, but it will not be a problem because a given cell can be in at most 4 responsibility areas of squares of the same level.

Communication Channels. All Robinson squares will need to read the bits on the bitmap layer when looking for simple patterns in their responsibility area. Because only 0-squares can access those bits directly, we have to set up a communication system between squares of different levels.

The idea is that each time a k-square needs to know a bit value, it will ask the corresponding $(k-1)$-square (the one whose visibility area contains the bit), which will in turn ask a $(k-2)$-square, and so on until the request is finally made to a 0-square that can directly access the bit and answer the query. Answers will then be transmitted recursively to the upper levels so that the original k-square can have the needed information.

A given Robinson square of any level can only request bits inside its responsibility area. However, each request is addressed to the square whose visibility area contains the bit, so after the first step of a recursive request, all subsequent requests are made inside visibility areas only. It is enough that any given k-

square be able to request a bit to any of the 36 $(k-1)$-squares around itself. We will use communication channels (connected paths) from a square to another to transmit queries and answers. What we need to ensure is that there is a global upper bound on the number of communication channels that pass through a given cell.

Since requests from a k-square are aimed at level $(k-1)$ squares, we will make the corresponding channels along the lines and columns on which level $(k-1)$ square sides lie. Figure 2 illustrates the communication channels used by a k-square to communicate with the necessary $(k-1)$-squares (the dark grey areas are the computing segments of the squares).

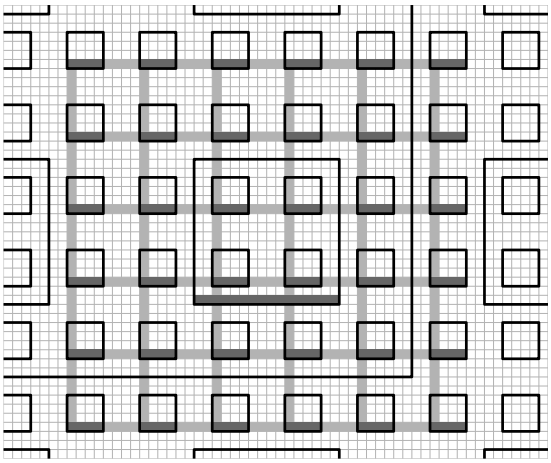

Fig. 2. The communication channels used by a level k Robinson square to communicate with the 36 closest level $(k-1)$ Robinson squares.

Because Robinson squares of different levels cannot have aligned sides, a line or column used in a channel network between a level k square and its surrounding level $(k-1)$ squares will not be used for channels connecting other levels of Robinson squares. As a consequence, any given cell of the plane lies on at most two levels of communication channels. Moreover a cell is in the responsibility area of at most 4 k-squares. These two last observations mean that a cell is on the communication network of at most 8 different Robinson squares (of any level). If we ensure that each Robinson square does at most one request at time, no more than 8 requests can pass on a single cell at the same time so the communications can be realized with a finite number of states.

Recursive Requests. We now have the necessary framework to explain in details how a high level square of size s can obtain the value of a bit on its responsibility area.

Coordinates of the required bits are considered relatively to the lower-left corner of the responsibility area of a square if they were generated by the square itself (during its computation it requires a given bit) and relatively to the lower left corner of the visibility area otherwise (when the request comes from a higher level square). This might seem unnecessarily complicated but by doing so we will only deal with positive coordinates and all modulo computations will be very easy to perform. Simply consider that the coordinates are with respect to the lowest and leftmost possible location of a bit.

All coordinates will be represented in binary form and we will assume that coordinates are padded with 0 so that they all have the maximum length (coordinates relative to the responsibility area are one bit longer than the ones relative to the visibility area).

When a square knows the coordinates of the bit it requires, it has to determine in which of the neighboring lower level squares' area of visibility the bit is, convert the coordinates relatively to that lower level square and send them to it. All these operations are very easy to handle because lower level squares are of size $s/4$, regularly arranged around the square of size s and their coordinate system "coincides" with that of the larger square.

This means that the most significant bits directly determine which lower square should be contacted, and the remaining bits represent the coordinates of the bit relatively to this lower square's visibility area. If the initial coordinates are relative to the visibility area, the two most significant bits determine which lower square to ask. If relative to the visibility area, the 3 first bits should be considered.

From these bits, a *path* is created that indicates where the destination square is. This path indicates how to travel through the communication network described earlier, from square to square, to reach the destination. It is a word on the aphabet $\{\uparrow, \downarrow, \rightarrow, \leftarrow\}$. After the path has been determined (in constant time since there are only finitely many possibilities), it is appended to the reduced coordinates (coordinates relative to the lower level destination square), to form a message that is sent through the communication network. This message is written on several cells: one bit of the coordinates or one letter of the path per cell.

When the destination square receives the message, it keeps the path, reads the coordinates, converts them and sends the next request (to the lower level square) the same way. When the answer to this request arrives, it can use the path (interpreted backwards) to send back the result bit to the higher level square.

Note that a given square will send at most one request at a time in a normal behavior. If however it is reset while a request has been sent, a new one might be sent. In this case the most recent request is the only one that is considered.

Request Time. We will denote by $T(s)$ the maximum duration of a request from a square of size s, that is the maximum number of time steps from the time when the square has the coordinates of the bit it requires to the moment

when it receives the bit value. Coordinates handled by a square of size s are of logarithmic size in s. According to what has been previously described, a request is handled as follows:

1. The square converts the coordinates, creates the path and message: $O(\log s)$
2. The message is sent to the lower level square: $O(s)$
3. The lower level square requests the bit: $T(s/4)$
4. The bit is send back from the lower level square using the path: $O(s)$.

There is however a slight complication: a given square can receive multiple requests at the same time. If it is in the area of responsibility of multiple squares of upper level, each of those can send a direct request. Each square also "produces" its own requests (from its own computation) that it has to send. A Robinson square can therefore handle up to 5 simultaneous requests.

In order not to exponentially increase this number, only one request is transmitted to the lower level at a time, the others are delayed until the answer is obtained. By using a "first in first out" ordering, we get $T(s) \leq 5(T(s/4)+O(s))$.

Since requests from 0-squares are completed in constant time, we get $T(s) = O(s^2)$. All requests are handled in quadratic time.

3.4 A Day in the Life of a Robinson Square

Now that we know how a Robinson square can get the values on the bitmap layer, which is by far the most complicated task performed on the automaton, let us have a look at the main computation.

The square's task is to look for "simple" square patterns in its responsibility area. The lower side of the Robinson square acts like a Turing machine with a tape of finite size s.

The computation of a Robinson square consists merely in enumerating ρ-simple patterns and, each time one is found, check that it does not appear in its responsibility area. Checking if a pattern of size $n \times n$ appears in the responsibility area of a square of size s can be done in time $O(n^2.s^4)$ without any optimization (roughly s^2 possible positions for the pattern, each bit requires s^2 steps).

Correctness of the Construction. Because looking for a pattern takes a polynomial time in the size of both the Robinson square and the pattern, and because each Robinson square has an exponential time to do the computation before it is reset, for any ρ-simple square pattern all sufficiently large Robinson squares will have enough time to produce it and search for it in their responsibility area.

The Robinson tiling has the property that any finite set of cells is in the responsibility area of an arbitrarily large Robinson square (hence the necessary overlapping of the responsibility areas). This means that any ρ-simple square pattern present on the bitmap layer will eventually be found by a large enough Robinson square (one whose responsibility area contains the pattern). If this pattern is of size larger than n_0, a $\square$ state will appear and the configuration will

not appear in the asymptotic set. The construction therefore ensures that no configuration containing a ρ-simple pattern of size larger than n_0 on its bitmap layer is in the asymptotic set of the automaton (except for the all $\square$ configuration of course).

3.5 Infiniteness of the Asymptotic Set

Up to this point we have proved that the asymptotic set of the described automaton contained no ρ-simple configuration other than the uniform $\square$ configuration. It remains to be shown that the asymptotic set contains infinitely many ρ-complex configurations. We will use the following lemma proved in [11]:

Lemma 3. *For any $0 < \rho < 1$, there exists a ρ-complex configuration over the set of states $\{0, 1\}$.*

Consider now the evolution of the automaton from an initial configuration containing one of the complex configurations described in Lemma 3 on its bitmap layer, a valid Robinson tiling on its Robinson layer and empty initial computations on all Robinson squares. From such a configuration, all Robinson squares will properly perform their verification but none will find any simple pattern of size more than n_0. The automaton will evolve infinitely without producing any $\square$ state, and its bitmap layer will never be changed. This means that for each ρ-complex bitmap layer, there is a configuration in the asymptotic set of our automaton with this exact bitmap layer. There are hence infinitely many configurations in the asymptotic set. This completes the proof of Theorem 1

4 Extensions

4.1 Optimality

For any $0 < \rho < 1$, the described automaton can ensure that no square pattern of size $n \geq n_0$ has complexity lower than $\rho.n^2$. It is clear that quadratic complexity is optimal but if the CA has N states the theoretical maximal complexity of a square pattern is $\log_2(N).n^2$. It is however possible to modify the construction to approach this theoretical bound by increasing the alphabet on the bitmap layer. If we use 2^α letters on this layer, the Robinson layer does not change and the computation layer only needs $O(\alpha)$ more states.

Producing the simple patterns is almost identical, requests on the bitmap layer are done bit by bit (α requests for a single letter) so the overall test time remains polynomial. Lemma 3 can be extended to larger alphabets and we can therefore ensure a complexity of at least $\rho.\alpha.n^2$ for all sufficiently large square patterns.

The number of states of this new automaton is $N = N_K.(N_C + \alpha).2^\alpha$. Since $\lim_{\alpha\to\infty} \log N = \alpha$ we can approach the optimal complexity as much as needed.

4.2 Other Dimensions

It might be possible to obtain a similar result in one dimension (with a linear lower bound on the complexity of the patterns). A similarly layered construction could work but the Robinson structure and the communications are more problematic. By using the "North-West deterministic" version of the Robinson tiling by Kari and Papasoglu [7] we can have a layer of the automaton on which a Robinson structure appears on the space-time diagram. In this case the finite areas are constantly evolving (segments of increasing and decreasing size) which makes the computation much harder to define and perform. We are currently working on a one-dimensional solution but the details have not yet been completely solved and written.

As for higher dimensions, there are different ways to extend the Robinson structure to finite cubes in such a way that any finite volume is in the visibility area of a cube (using three-dimensional substitution systems for instance). Computations are performed as in the previous construction on a segment and verifications can still be done in polynomial time.

References

1. Allauzen, C., Durand, B.: Tiling Problems. In: The classical decision problem. Perspectives in Mathematical Logic. Springer-Verlag Berlin, Heidelberg (2001)
2. Cattaneo, G., Dennunzio, A., Margara, L.: Chaotic Subshifts and Related Languages Applications to one-dimensional Cellular Automata. Fundamenta Informaticae **52**(1-3) (2002) 39–80
3. Cervelle, J.: Complexité dynamique et algorithmique des automates cellulaires. Habilitation à diriger des recherches, Université Paris Est, Marne-la-Vallée (December 2007)
4. Culik II, K., Pachl, J.K., Yu, S.: On the Limit Sets of Cellular Automata. SIAM J. Comput. **18**(4) (1989) 831–842
5. Durand, B., Levin, L.A., Shen, A.: Complex tilings. Journal of Symbolic Logic **73**(2) (2008) 593–613
6. Johnson, A., Madden, K.: Putting the Pieces Together: Understanding Robinson's Nonperiodic Tilings. The College Mathematics Journal **28**(3) (May 1997) 172–181
7. Kari, J., Papasoglu, P.: Deterministic Aperiodic Tile Sets. Geometric And Functional Analysis **9** (1999) 353–369
8. Kurka, P.: Languages, equicontinuity and attractors in cellular automata. Ergodic Theory & Dynamical Systems **17** (1997) 417–433
9. Levin, L.A.: Aperiodic Tilings: Breaking Translational Symmetry. Comput. J. **48**(6) (2005) 642–645
10. Robinson, R.M.: Undecidability and Nonperiodicity for Tilings of the Plane. Inventiones Math. **12** (1971)
11. Rumyantsev, A.Y., Ushakov, M.A.: Forbidden Substrings, Kolmogorov Complexity and Almost Periodic Sequences. In: STACS 2006, 23rd Annual Symposium on Theoretical Aspects of Computer Science, Marseille, France, February 23-25, 2006, Proceedings. (2006) 396–407
12. Wolfram, S.: Universality and complexity in cellular automata. Physica D **10** (1984) 1–35