Complexity Steering in Cellular Automata

Bar Y. Peled
Avishy Y. Carmi
Department of Mechanical Engineering
Ben-Gurion University of the Negev

A cellular automaton is presented whose governing rule is that the Kolmogorov complexity of a cell's neighborhood may not increase when the cell's present value is substituted for its future value. Using an approximation of this two-dimensional Kolmogorov complexity, the underlying automaton is shown to be capable of simulating binary logic circuits. A similar automaton whose rule permits at times the increase of a cell's neighborhood complexity is shown to produce animated entities that can be used as information carriers akin to gliders in Conway's Game of Life. The element that repeatedly generates gliders, the glider gun, is constructed in this automaton using a number of self-replicating mechanisms. Moreover, gliders' annihilation and creation allow constructing logic gates as well as data encoding mechanisms.

Keywords: cellular automata; Kolmogorov complexity; universal computation; self-replication; negentropy

1. Introduction

In natural and artificial systems, pattern formation and order are the imprints of a computational process. We ask whether this process can be reversed—can the enforcement of order or a pattern bring about computation? What sort of computation can be realized, for example, by not allowing the local patterns in cellular automata to get any more complex than they already are? Can any computation be realized in such a manner? This work is an attempt to answer some of these questions.

A cellular automaton is a discrete dynamical system that was originally conceived in the late 1940s by John von Neumann and Stanislaw Ulam. They incorporated the cellular automaton model into von Neumann's idea of a universal constructor [1]. Cellular automata exhibit a new way of thinking about how little complexity can achieve interesting behavior, which can be emulated in diverse physical and biological phenomena [2]. They have played a significant role in computation, as they have proven to be Turing universal [3, 4]. The idea that the universe in itself is a cellular automaton inspired Zuse's *Calculating Space* [5], a precursor of digital physics.

https://doi.org/10.25088/ComplexSystems.27.2.159

A typical cellular automaton consists of a grid of cells, each storing a value and a set of rules according to which their values change. Although the underlying rules may be simple, the behavior of the automaton as a whole may quickly become complex, a trait that allows such automata to emulate diverse physical and biological phenomena. There are many known cellular automata whose chaotic behavior has been intensively studied. Perhaps the most famous are the Game of Life and rule 110. It has been shown that both of them can realize any computation an ordinary computer can perform [4].

Von Neumann's universal constructor used thousands of cells and 29 states to self-reproduce. Its complexity was later reduced in [6]. Langton constructed a self-replicating automaton known as Langton's loops [7]. His model is less complex than the previous models due to von Neumann and Codd. Langton's model was further simplified in [8, 9]. The constructions of the smallest computationally universal cellular automata have been considered in [2, 10–12]. Reversible cellular automata have been studied in [13]. The links between dynamical and computational properties of cellular automata have been investigated by Di Lena and Margara [14].

In this paper, we show that computation in cellular automata can be realized by exerting local order, where "order" is mathematically defined in terms of Kolmogorov complexity. In particular, we formulate a cellular automaton that employs the following rule: A cell's value is changed from y to x if the Kolmogorov complexity of its present Moore neighborhood is smaller with x than with y. Using an approximation of this two-dimensional Kolmogorov complexity, we show that the underlying automaton can compute Boolean functions. In particular, it can realize a universal set of Boolean gates, AND and NOT, as well as wire elements for transferring information from a given cell to any other cell.

The expressiveness of a cellular automaton employing the "nowhere increasing" complexity rule is rather limited. The initial grid that encodes the logic circuit is gradually obliterated during the automaton evolution. This implies that similar constructions may not be used for recursive computations. Nevertheless, a similar automaton with an alternating rule where the cell's neighborhood complexity may at times increase is shown to produce animated entities that can be used as information carriers akin to gliders in the Game of Life [15]. It is further shown how glider guns, logic gates and data encoding mechanisms can be realized in this automaton.

This paper is organized as follows. Section 2 is a brief introduction to the notion of Kolmogorov complexity. The nowhere increasing complexity cellular automaton is described in Section 3. It is shown to realize logic gates in Section 4. The alternating, nowhere increasing/decreasing cellular automaton is presented and analyzed in Section 5.

The evolution of complexity during the automaton's operation is discussed in Section 6. Concluding remarks are offered in Section 7.

2. Kolmogorov Complexity

The Kolmogorov complexity $K(s)$ of an object or a string s is a measure of the computational resources that are needed to generate s [16, 17]. Informally, $K(s)$ is the length of the shortest program that produces s as an output and then halts. The string $(01)^n$, for example, can be described as "n repetitions of 01." This string is $2n$ digits long, while its description contains around $\log_2(n)$ binary digits, that is, the length of the binary representation of n. On the other hand, seemingly random occurrences of zeros and ones will generally not admit a description shorter than their own length.

The Kolmogorov complexity can be used to define the measure

$$2^{-K(s)}. \tag{1}$$

If $K(s)$ is the length of a program to a universal prefix Turing machine that produces s and then halts, then by Kraft's inequality (1) may be interpreted as an unnormalized measure of probability over all such programs [17].

Kolmogorov complexity is an uncomputable function—there is no program that takes a string s as an input and produces the number $K(s)$ [17]. Normally, $K(s)$ is approximated using known compression techniques; see, for example, [18]. But equation (1) can also be used to estimate $K(s)$, assuming one can simulate a large number of Turing machines that produce s. Some of these machines will halt and some will not. Counting the number of them that produce s and then halt gives a number $m(s)$. The Kolmogorov complexity is then approximated, up to a constant, as $-\log_2 m(s)$ [19, 20].

In this paper, we are interested in the Kolmogorov complexity of a two-dimensional object, a 3×3 binary matrix. As explained in Section 3, such a matrix describes the Moore neighborhood of a cell. The complexity of all these 512 binary matrices has been recently approximated in [20]. Simulating a large number of two-dimensional Turing machines, the probability (1) was approximated for any given matrix configuration. An estimate of K was then obtained, as just described. These approximations can be found at
github.com/algorithmicnaturelab/OACC/blob/master/data/K-3x3.csv.

3. The Cellular Automaton

A cellular automaton is defined by a finite number of "colored" cells together with a set of rules that specify how to manipulate their

https://doi.org/10.25088/ComplexSystems.27.2.159

colors. The cells of a standard automaton contain binary values, either "0" (black) or "1" (white). The purpose of the rules is to determine the state (color) of a cell at time $t+1$ based on the values of its neighbors, the set of cells in its vicinity at time t. The automaton evolves by using the rules to determine the next value for each cell in the grid at time t. The new cell values thus obtained make the grid at time $t+1$.

Let us denote as $c_{ij}(t) \in \{0, 1\}$ the value at time t of the cell whose coordinates are (i, j). At each time step, a cell updates its value according to the following rule.

Rule 1. A cell's value is changed from c to $1-c$ if the Kolmogorov complexity of its present Moore neighborhood is smaller with $1-c$ than with c.

This rule is mathematically expressed by:

$$c_{ij}(t+1) = \begin{cases} c_{ij}(t), & K_{ij}(t) \le K'_{ij}(t) \\ 1 - c_{ij}(t), & \text{otherwise.} \end{cases} \tag{2}$$

Here, $K_{ij}(t)$ is the Kolmogorov complexity at time t of the Moore neighborhood of $c_{ij}(t)$, a 3×3 pattern composed of the cell at location (i, j) together with eight other cells that surround it. The Kolmogorov complexity at time t of the same pattern in which the cell at (i, j) is flipped is represented by $K'_{ij}(t)$. For example, K and K' may be evaluated for the pair in Figure 1.

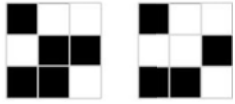


Figure 1. Moore neighborhoods for K and K'.

A pseudocode for a cellular automaton employing the rule depicted in Figure 1 is provided in Algorithm 1.

Syntax: $c_{ij}(t+1) = \mathbf{CA}\downarrow(c_{ij}(t))$
Input: $c_{ij}(t)$, $i = 1, \ldots, N$, $j = 1, \ldots, M$ (grid at time t)
Output: $c_{ij}(t+1)$, $i = 1, \ldots, N$, $j = 1, \ldots, M$ (grid at time $t+1$)
for $i = 2 : N-1$ **do**
  for $j = 2 : M-1$ **do**
    Let A be the Moore neighborhood of $c_{ij}(t)$.
    Obtain $K_{ij}(t)$ using A from the lookup table [20].
    Flip the value of the middle cell in A and similarly obtain $K'_{ij}(t)$.

```
            if K_ij(t) ≤ K'_ij(t) then
                c_ij(t + 1) = c_ij(t)
            else
                c_ij(t + 1) = 1 − c_ij(t)
            end if
        end for
    end for
```

Algorithm 1. Nowhere increasing Kolmogorov complexity cellular automaton.

4. Computational Capabilities

In what follows, the underlying automaton is shown to realize the universal set of gates, NOT and AND, together with wire elements for connecting them. By a gate or a wire, we mean a grid whose cells store initial values, some of which represent inputs and some of which represent outputs. Iterating the automaton rule in Figure 1 where K and K' are approximated as in [20] changes the cells' values. This procedure is reiterated until all cell values no longer change or oscillate indefinitely. The output cells then store the outcome of the computation.

A *wire element* transfers information between cells in the grid. Its basic form is shown in Figure 2. In this figure, the input $x \in \{0, 1\}$ to the wire is specified by a single black cell in the block of white cells just above the wire's upper end. The wire's other end is connected to another block of white cells. The initial grids of the automaton for two different inputs x are shown in the leftmost column. In the upper-left frame, the black cell in the center of the white block represents

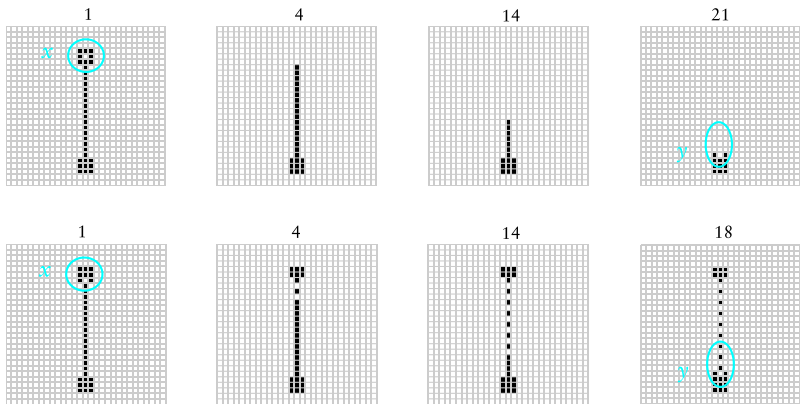


Figure 2. Wire element. The grid size is 30×30.

https://doi.org/10.25088/ComplexSystems.27.2.159

the input "0." Similarly, the input "1" in the lower-left frame is represented by a black cell just below the center of this white block. Injecting "0" to the wire completely destroys it, as seen by the upper sequence of images showing different times during the evolution. Injecting "1," the images in the lower row show that a propagating sequence of alternating black and white cells comes out of the upper white block all the way down. These two behaviors are interpreted as a wire carrying either "0" or "1"; that is, the content of the wire can be read off at the vicinity of y.

A *NOT gate* takes inputs x and returns $y = 1 - x$. Its realization is shown in Figure 3. The upper row in this figure shows the initial grid for this gate with different inputs, $x = 1$ and $x = 0$. The respective outputs $y = 0$ and $y = 1$ in the lower row are obtained after several iterations of the automaton rule.

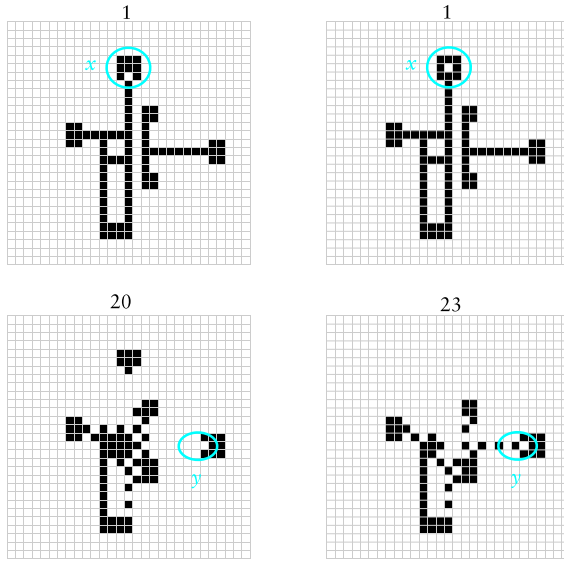


Figure 3. NOT gate. The upper row shows the initial grid for the two inputs, $x = 1$ (left) and $x = 0$ (right). The respective final grid for each input is shown in the lower row. The grid size is 30×30.

An *AND gate* takes inputs x and y and returns $z = xy$; that is, only when $x = y = 1$ does the gate return $z = 1$. Its realization is shown in Figure 4. The upper row in this figure shows the initial grid for this gate with different inputs (x, y), that is, $(0, 0)$, $(0, 1)$, $(1, 0)$ and $(1, 1)$. The respective outputs $z = 0$, $z = 0$, $z = 0$ and $z = 1$ in the lower row are obtained after several iterations of the automaton rule.

An OR gate may be constructed out of an AND and three NOTs; that is, $z = 1 - (1 - x)(1 - y)$, so that $z = 1$ if at least one of the

inputs, x or y, equals one. The realization of this gate is shown in Figure 5. The upper row in this picture shows the initial grid for this gate with different inputs (x, y), that is, $(0, 0)$, $(1, 0)$, $(0, 1)$ and $(1, 1)$. The respective outputs $z = 0$, $z = 1$, $z = 1$ and $z = 1$ in the lower row are obtained after several iterations of the automaton rule. The realization in Figure 6 is that of an XOR gate, for which the output is $z = \left(1 + (-1)^{xy}\right)/2$.

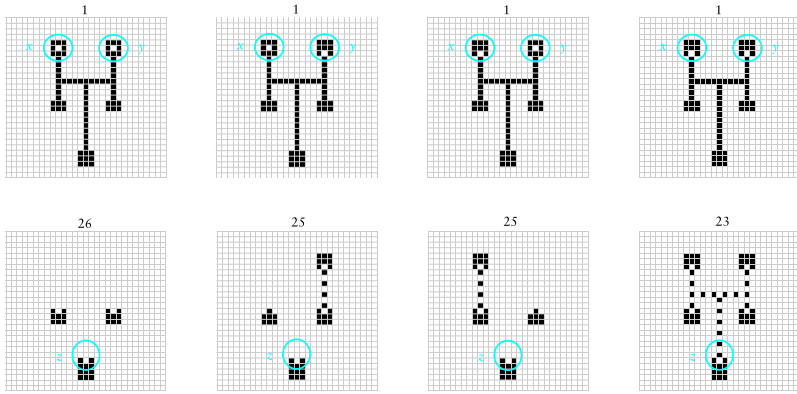


Figure 4. AND gate. The upper row shows the initial grid for the four input combinations, from left to right (0, 0), (0, 1), (1, 0) and (1, 1). The respective final grid for each input is shown in the lower row. The grid size is 30×30.

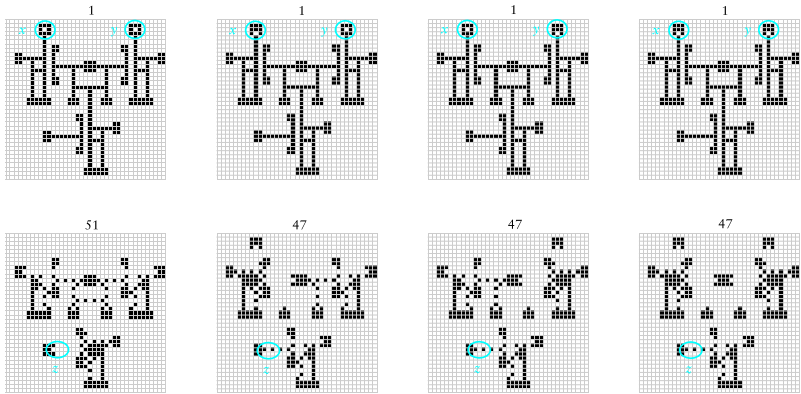


Figure 5. OR gate. The upper row shows the initial grid for the four input combinations, from left to right (0, 0), (1, 0), (0, 1) and (1, 1). The respective final grid for each input is shown in the lower row. The grid size is 40×40.

https://doi.org/10.25088/ComplexSystems.27.2.159

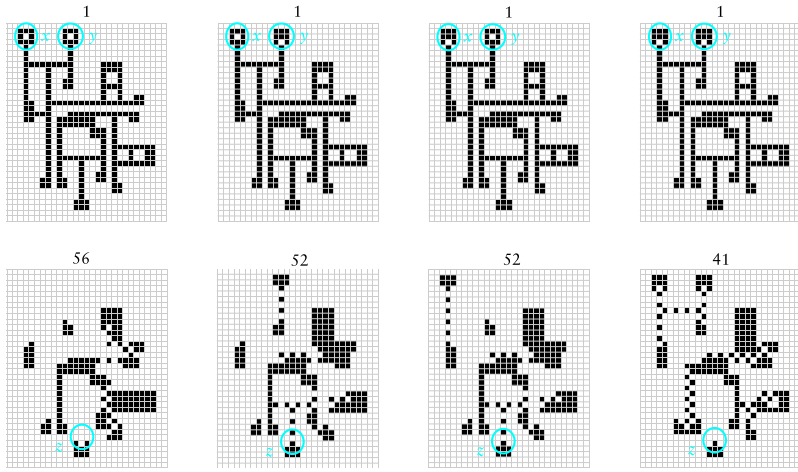


Figure 6. XOR gate. The upper row shows the initial grid for the four input combinations, from left to right (0, 0), (0, 1), (1, 0) and (1, 1). The respective final grid for each input is shown in the lower row. The grid size is 37×30.

5. Alternating Rules and Gliders

Gliders are animated entities that emerge in the grid during the automaton evolution. In terms of computation, such patterns are instrumental for carrying information across the grid. The Game of Life-based Turing machine, for example, heavily relies on gliders to realize its logic and memory parts [21].

For reasons mentioned in the introduction, we suspect that the preceding cellular automaton cannot produce gliders. We were able, however, to generate gliders with an automaton whose rule permits at times the increase of a cell's Kolmogorov complexity. One cycle of this automaton is as follows. In the beginning of a cycle it employs two rules to obtain the grid in the next time step:

Rule 2. Nothing comes out of nothing—do nothing to a (blank) cell whose Moore neighborhood vanishes.

Rule 3. A cell's value is changed from c to $1 - c$ if the Kolmogorov complexity of its present Moore neighborhood is larger with $1 - c$ than with c.

A single cycle of this automaton starts with a single iteration of Algorithm 2. For the next few time steps, the automaton operates as described in Algorithm 1; that is, it employs the "nowhere increasing" complexity rule. It proceeds in this way until the pair of grids, the

recent one at an odd time step and the one two time steps back, are the same. This cycle is repeated indefinitely. A pseudocode for this automaton is given in Algorithm 3.

Syntax: $c_{ij}(t+1) = \mathbf{CA}\uparrow(c_{ij}(t))$
Input: $c_{ij}(t)$, $i = 1, \ldots, N$, $j = 1, \ldots, M$ (grid at time t)
Output: $c_{ij}(t+1)$, $i = 1, \ldots, N$, $j = 1, \ldots, M$ (grid at time $t+1$)
for $i = 2 : N - 1$ **do**
  for $j = 2 : M - 1$ **do**
    Let A be the Moore neighborhood of $c_{ij}(t)$.
    if A does not zeros **then**
      Obtain $K_{ij}(t)$ using A from the lookup table [20].
      Flip the value of the middle cell in A and similarly obtain $K'_{ij}(t)$.
      if $K_{ij}(t) \geq K'_{ij}(t)$ **then**
        $c_{ij}(t+1) = c_{ij}(t)$
      else
        $c_{ij}(t+1) = 1 - c_{ij}(t)$
      end if
    end if
  end for
end for

Algorithm 2. Nowhere decreasing Kolmogorov complexity cellular automaton.

Syntax: $c_{ij}(t+s) = \mathbf{CA}\updownarrow(c_{ij}(t))$
Input: $c_{ij}(t)$, $i = 1, \ldots, N$, $j = 1, \ldots, M$ (grid at time t)
Output: $c_{ij}(t+s)$, $i = 1, \ldots, N$, $j = 1, \ldots, M$ (grid at time $t+s$)
$c_{ij}(t+1) = \mathbf{CA}\uparrow(c_{ij}(t))$
$s = 0$
while $\{\exists\, i, j c_{ij}(s) \neq c_{ij}(s-2)\} \vee \{s \text{ is even}\}$ **do**
  $c_{ij}(t+s+1) = \mathbf{CA}\downarrow(c_{ij}(t+s))$
  $s = s + 1$
end while

Algorithm 3. Alternating, nowhere decreasing/increasing cellular automaton.

The basic construction of a glider and its evolution in the course of two cycles of this automaton are shown in Figure 7.

https://doi.org/10.25088/ComplexSystems.27.2.159

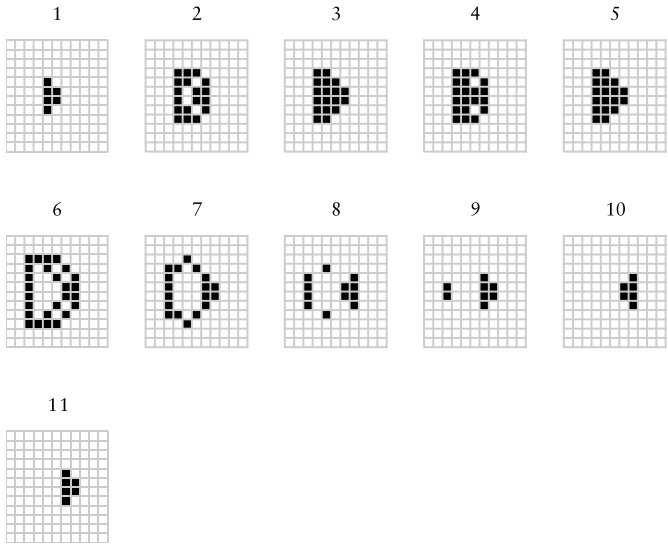


Figure 7. Evolution of a glider using the cellular automaton in Algorithm 3. The first cycle starts with the grid numbered 1 and concludes with the grid numbered 5. The second cycle starts with grid 5 and concludes with grid 13. The grid size is 12×12.

5.1 Glider Guns and Logic Gates

The glider gun in the Game of Life is the mechanism that generates gliders. A similar entity may be realized using the alternating automaton rules in Figure 1. Its construction makes use of another elementary unit. A *caterpillar*, as the name suggests, is an elongated entity that grows in the grid by replicating a single pattern every few time steps. A caterpillar extends indefinitely if no other entities interact with it in the grid. See Figure 8.

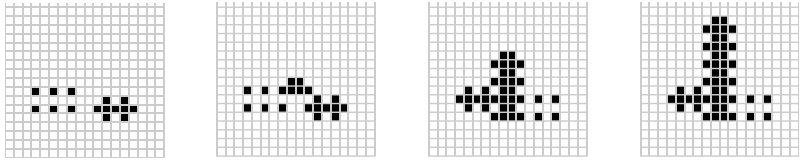


Figure 8. Evolution of a caterpillar using the cellular automaton in Algorithm 3.

Three caterpillars are used to construct a glider gun as in Figure 9. The two vertical caterpillars approach one another and cut a piece of the horizontal caterpillar once they meet. After that, the caterpillars are all annihilated, while the single glider that was cut out endures

(see Figure 10). This process is repeated indefinitely during the automaton evolution, and so gliders come out one after the other at a constant rate.

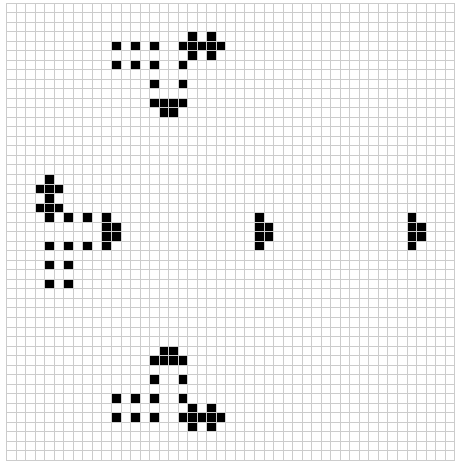


Figure 9. A glider gun is constructed using three caterpillars.

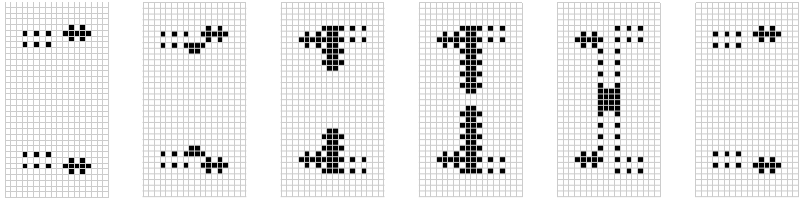


Figure 10. Once two caterpillars meet up, they are annihilated in the next time step.

As demonstrated in Figure 11, when two gliders meet, depending on their position, they may be annihilated in the next time step. Similar to the Game of Life, this feature may be utilized for constructing the universal set of gates, AND and NOT. See Figures 12 and 13. It can also be used for encoding data into a stream of gliders emanating from a glider gun.

A stream of gliders can be viewed as a stream of bits, where 0 and 1 are represented by the absence and presence of gliders at particular locations. Encoding bits into such a stream is done by placing caterpillars below or above it. This allows eliminating some gliders while leaving others, so as to produce a particular periodic pattern of 0s and 1s. See Figure 14.

https://doi.org/10.25088/ComplexSystems.27.2.159

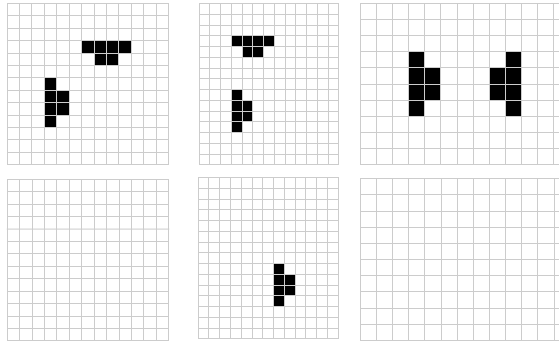


Figure 11. Annihilation configurations. The grids in the upper row show two gliders approaching one another from different directions. The corresponding grids after they meet up are shown in the bottom row.

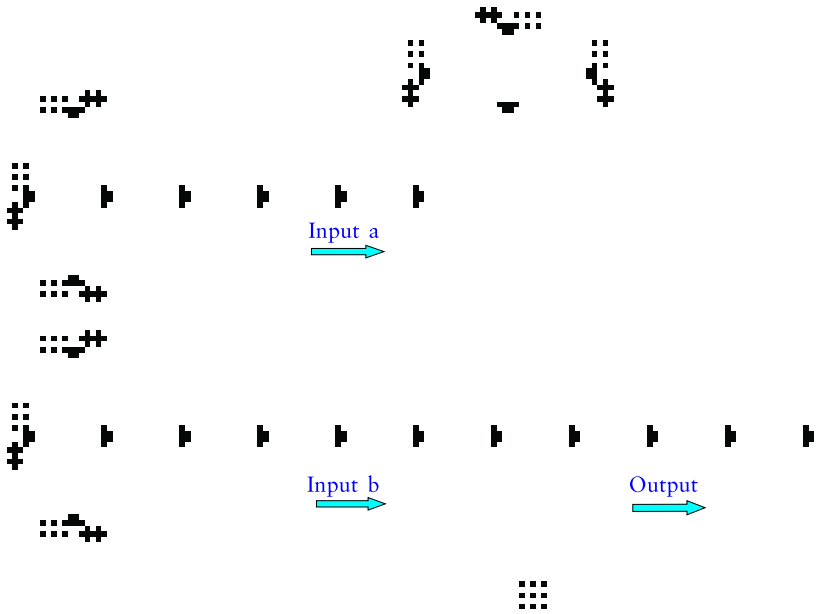


Figure 12. AND gate in the alternating increasing/decreasing complexity automaton.

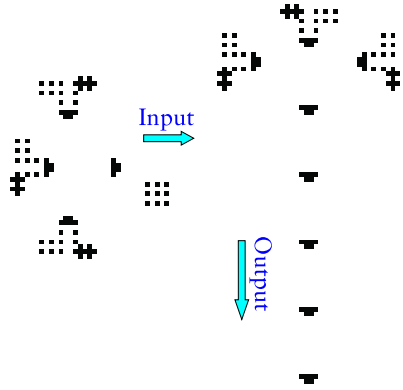


Figure 13. NOT gate in the alternating increasing/decreasing complexity automaton.

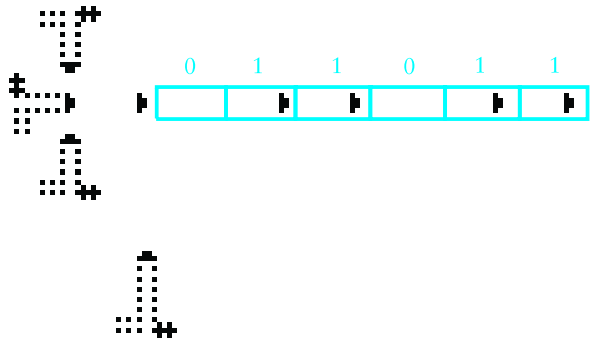


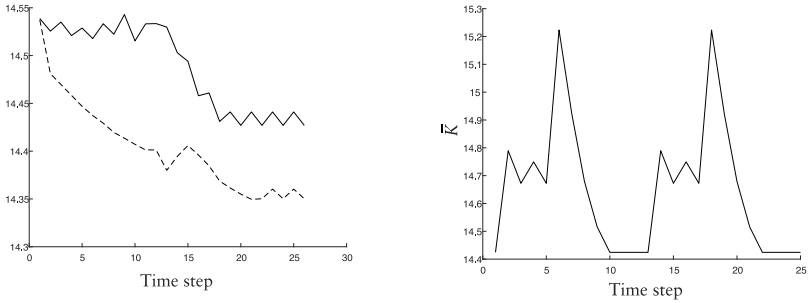
Figure 14. Encoding a periodic pattern 011 011... by a stream of gliders.

6. Emergent Complexity

As neighborhoods overlap, the average Kolmogorov complexity in the grid, $\overline{K}$, may nevertheless increase in the automaton in Algorithm 1 and may decrease in the automaton in Algorithm 2.

The values of $\overline{K}$ during the operation of the NOT gate in Figure 3 are shown in Figure 15(a). The transition from the initial to the final grid, in which complexity is lower on the average, shows instances where the average complexity rises. The average complexity fluctuates in the course of a glider evolution, as shown in Figure 15(b).

https://doi.org/10.25088/ComplexSystems.27.2.159

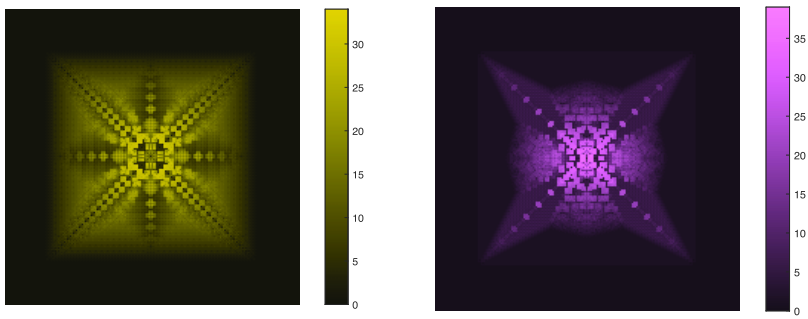


(a) (b)

Figure 15. (a) The average Kolmogorov complexity during the evolution of the NOT gate in Figure 3. The solid and dashed lines correspond, respectively, to the inputs 1 and 0. (b) The corresponding measure is shown for the glider in Figure 7.

Counting the instances where a neighborhood's complexity drops in the next time step yields the "flowers" in Figure 16. The four patterns are obtained using different initial conditions of the nowhere decreasing complexity automaton in Algorithm 2. Brighter pixels represent higher counts for the neighborhoods in the respective locations in the grid.

These examples demonstrate that locally enforcing lower complexity states potentially leads to otherwise untenable reduction rates in the grid's average complexity as neighborhoods overlap (interact). These very interactions are the reason for instantaneous increase in both local and average complexities [22]. It can be seen that two opposing factors are at work in this dynamical behavior; a "life" rule that tends to decrease local complexity at greater rates than otherwise possible, and interactions that at times deny such reduction rates.



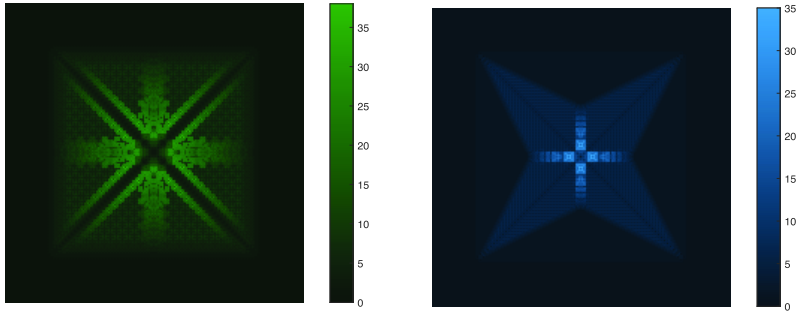


Figure 16. Kolmogorov complexity flowers.

7. Conclusion

This paper is an attempt to address the questions raised in Section 1. The answer we offer is only partial. One may wonder whether any computable function can be computed by a similar "nowhere increasing" cellular automaton, or in other words, whether such an automaton is Turing complete. For one reason, we think it is not. During its evolution, the initial grid on which the logical gate is encoded self-destructs. Therefore, outputs cannot be reused as inputs to the same logical gate. Although not proven, we suspect that this behavior hinders the construction of a memory device and thus also of a Turing-equivalent model of computation.

But the concept of using a measure of complexity to evolve is multifaceted. We have shown that a cellular automaton whose rule permits at times the increase of the cell's neighborhood complexity can produce gliders, glider guns, logic gates and data encoding mechanisms. The lesson learned from the Game of Life is that, apart from a memory unit, these are the basic ingredients in any computation, and so perhaps this automaton also is Turing complete.

As a final remark, we have used a particular measure of complexity of the 3×3 Moore neighborhood. Other complexity measures and neighborhood dimensions may similarly be used to evolve cellular automata with different computational capabilities and behavior. In this respect, it would be interesting to investigate the behavior of similar automata employing the approximate 4×4 Kolmogorov complexity lookup table in
github.com/algorithmicnaturelab/OACC/blob/master/data/K-4x4.csv.

Acknowledgments

This research is supported by Israel Science Foundation Grant No. 1723/16.

References

[1] J. von Neumann, *Theory of Self-Reproducing Automata* (A. W. Burks, ed.), Urbana, IL: University of Illinois Press, 1966.

[2] S. Wolfram, *A New Kind of Science*, Champaign, IL: Wolfram Media, 2002. www.wolframscience.com.

[3] P. Rendell, "Turing Universality of the Game of Life," in *Collision-Based Computing* (A. Adamatzky, ed.), London: Springer, 2002 pp. 513–539.

[4] M. Cook, "Universality in Elementary Cellular Automata," *Complex Systems*, **15**(1), 2004 pp. 1–40. complex-systems.com/pdf/15-1-1.pdf.

[5] K. Zuse, "Rechnender Raum (Calculating Space)," 1st re-edition (A. German and H. Zenil, eds.), *A Computable Universe: Understanding & Exploring Nature as Computation,* London: World Scientific Publishing Company, 2012. www.mathrix.org/zenil/ZuseCalculatingSpace-GermanZenil.pdf.

[6] E. F. Codd, *Cellular Automata*, New York: Academic Press, 1968.

[7] C. G. Langton, "Self-Reproduction in Cellular Automata," *Physica D: Nonlinear Phenomena*, **10**(1–2), 1984 pp. 135–144. doi:10.1016/0167-2789(84)90256-2.

[8] J. Byl, "Self-Reproduction in Small Cellular Automata," *Physica D: Nonlinear Phenomena*, **34**(1–2), 1989 pp. 295–299. 10.1016/0167-2789(89)90242-X.

[9] J. A. Reggia, S. L. Armentrout, H.-H. Chou and Y. Peng, "Simple Systems That Exhibit Self-Directed Replication," *Science*, **259**(5099), 1993 pp. 1282–1287. doi:10.1126/science.259.5099.1282.

[10] K. Lindgren and M. G. Nordahl, "Universal Computation in Simple One-Dimensional Cellular Automata," *Complex Systems*, **4**(3), 1990 pp. 299–318. complex-systems.com/pdf/04-3-4.pdf.

[11] N. Ollinger, "Universalities in Cellular Automata: A (Short) Survey," in *Journées Automates Cellulaires* (B. Durand, ed.), Apr 2008, Uzès, France, 2008 pp. 102–118. core.ac.uk/download/pdf/52464175.pdf.

[12] T. Neary and D. Woods, "Four Small Universal Turing Machines," *Fundamenta Informaticae*, **91**(1), 2009 pp. 123–144.

[13] N. Margolus, "Physics-like Models of Computation," *Physica D: Nonlinear Phenomena*, **10**(1–2), 1984 pp. 81–95. doi:10.1016/0167-2789(84)90252-5.

[14] P. Di Lena and L. Margara, "Computational Complexity of Dynamical Systems: The Case of Cellular Automata," *Information and Computation*, **206**(9–10), 2008 pp. 1104–1116. doi:10.1016/j.ic.2008.03.012.

[15] J. T. Lizier, M. Prokopenko and A. Y. Zomaya, "Local Information Transfer as a Spatiotemporal Filter for Complex Systems," *Physical Review E*, **77**(2), 2008 026110. doi:10.1103/PhysRevE.77.026110.

[16] A. N. Kolmogorov, "Three Approaches to the Quantitative Definition of Information," *International Journal of Computer Mathematics*, **2**(1–4), 1968 pp. 157–168. doi:10.1080/00207166808803030.

[17] G. J. Chaitin, "On the Length of Programs for Computing Finite Binary Sequences: Statistical Considerations," *Journal of the ACM (JACM)*, **16**(1), 1969 pp. 145–159. doi:10.1145/321495.321506.

[18] É. Rivals, M. Dauchet, J. P. Delahaye and O. Delgrange, "Compression and Genetic Sequence Analysis," *Biochimie*, **78**(5), 1996 pp. 315–322. doi:10.1016/0300-9084(96)84763-8.

[19] J.-P. Delahaye and H. Zenil, "Numerical Evaluation of Algorithmic Complexity for Short Strings: A Glance into the Innermost Structure of Randomness," *Applied Mathematics and Computation*, **219**(1), 2012 pp. 63–77. doi:10.1016/j.amc.2011.10.006.

[20] H. Zenil, F. Soler-Toscano, J.-P. Delahaye and N. Gauvrit, "Two-Dimensional Kolmogorov Complexity and an Empirical Validation of the Coding Theorem Method by Compressibility," *PeerJ Computer Science*, **1**(e23), 2015. doi:10.7717/peerj-cs.23.

[21] P. Rendell, *Turing Machine Universality of the Game of Life*, Cham, Switzerland: Springer International Publishing, 2016 pp. 45–70.

[22] H. Zenil, N. A. Kiani and J. Tegnér, "Algorithmic Information Dynamics of Persistent Patterns and Colliding Particles in the Game of Life." arxiv.org/abs/1802.07181.