Cellular Automata and Computational Universality

August 20, 2018

Thomas Archbold
University of Warwick
T.Archbold@warwick.ac.uk

Abstract

Cellular automata are discrete models with the ability to not only give rise to beautiful, intricate patterns, but also to be used as powerful tools of computation, with applications in cryptography, error-correction coding, and simulation of computer processors, to name a few. They also raise profound questions about the nature of our reality, asking whether our universe could be one such automaton. This paper provides a discussion into these automata, exploring the various power and limitations of a number of specific rule sets, covering John Conway's well-known "Game of Life" to the more obscure "Langton's ant" and "Wireworld". In particular, it explores the notion of computational universality, or Turing completeness, an automaton's ability to simulate any conceivable computation, and considers their potential in the context of solving two specific problems, the Firing Squad Synchronisation Problem and the Majority Problem. Existing solutions to these problems are explored, and their existing avenues for optimisation are discussed. In order to fully appreciate the complex structures that can arise from such simple beginnings, this project also presents software to visualise and probe further into the nature of the automata highlighted.

1 Introduction

A cellular automaton is a discrete model of computation which consists of a finite collection of "cells", each in one of a finite number of states. The state of each of these cells may evolve over the progression of time in discrete steps, and may do so according to a deterministic set of rules based on the states of neighbouring cells. Their inherently discrete nature allows for strong analogies to be made with digital computers, and gifts them the ability to simulate digital processes and the potential to solve problems in this area.

Consider the cellular automaton defined by the simple rule:

$$a_i^{t+1} = a_{i-1}^t + a_{i+1}^t \mod 2 \tag{1}$$

For any automaton, in each time step the rule is applied to all cells in the automaton instantaneously and simultaneously. In this case, our set of states is $0, 1$, and for each cell we look at the cells immediately preceding and succeeding it: if exactly one of them is in state 1, then this cell will be in state 1 in the next time step. Otherwise it will be in state 0.

1.1 Self-similarity

Figure 1 shows a visualisation of Eq. 1. A cell is filled white if it has the value 1, and black otherwise. Note that each new "line" of the visualisation represents the automaton in the next successive time-step from the one before; this automaton is one-dimensional, and so we may represent it in its entirety as a single row of cells. This pattern exhibits two important

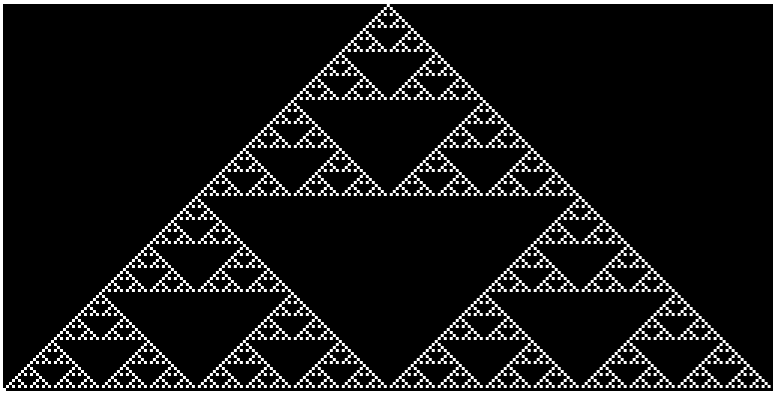


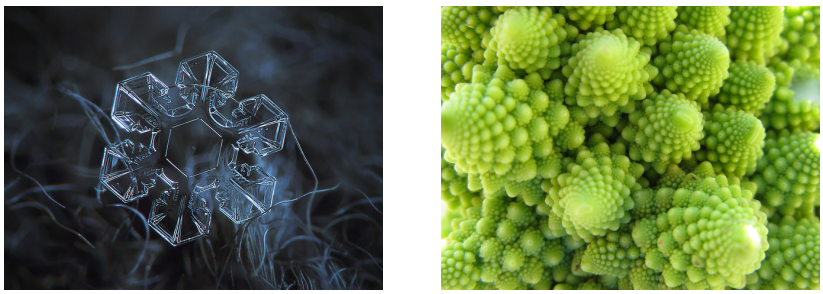
Figure 1: Pattern generated from applying Eq. 1 over 129 generations

characteristics, the first of which is "self-similarity", meaning that portions of the pattern generated are indistinguishable from the whole when magnified. As Wolfram states, in this way "the pattern is therefore invariant under rescaling... and may be characterised by a fractal dimension" [1]. One such definition of this is the Hausdorff-Besicovitch dimension [2].

As an example, suppose we have a cube with some mass, and we wish to find out how the mass scales when we try to make the same cube out of smaller copies of the original. Intuitively, one way to do this require eight smaller cubes, each of whose side length is half that of the original cube. So with the scaling factor of $\frac{1}{2}$, the mass of each smaller cube is $\frac{1}{8} = (\frac{1}{2})^3$. So we arrive at the equation $N = S^D$, where N is the number of smaller copies that can be stuck together to make the original, S is the scaling factor of these smaller copies, and D is the dimension. In this example, we compute that the dimension of a cube is 3, which is obviously correct. Back to the dimension of the Sierpinski Triangle, we can see that each larger triangle is made up of three smaller triangles, each of whose side lengths are half that of the larger triangle. So we calculate the dimension as:

$$\begin{aligned} \tfrac{1}{3} &= (\tfrac{1}{2})^D \\ \log \tfrac{1}{3} &= D \log \tfrac{1}{2} \\ D &= \tfrac{\log 3}{\log 2} \approx 1.585 \end{aligned} \tag{2}$$

Many naturally-occurring systems exhibit fractal structures, from snowflakes to pine cones to Romanesco broccoli, and raises the possibility that these are generated through the evolution of some natural cellular automata or similar processes.



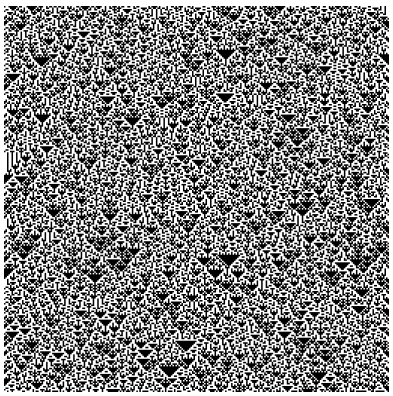
(a) Snowflake (b) Romanesco broccoli

Figure 2: Snowflakes and Romanesco broccoli exhibit fractal structures

1.2 Self-organisation

Second of the characteristics exhibited by this cellular automaton is "self-organisation", whereby from an initial disorderly or chaotic state there appear ordered structures seemingly spontaneously. This is illustrated in Fig. 3 below: one can clearly see the emergence of similar triangular structures in the cone snail's shell, appearing from a chaotic background. This starting chaos is modelled in the cellular automaton by randomly assigning each cell a value of 0 or 1.

(a) Shell of a cone snail



(b) Rule 90 with random starting configuration

Figure 3: Natural processes can exhibit structures found in automata

2 Classifying automata

In order to classify automata, we can take into account both the size of the neighbourhood each cell considers going into the next generation, as well as the number of possible states in which a cell may be. Call these N and Q respectively; in the case of Eq. 1, $N = 3$ and $Q = 2$, and so the total number of different "transitions" from one state to the next is $N^Q = 8$. We may write out all of these transitions as follows:

111	110	101	100	011	010	001	000
0	1	0	1	1	0	1	0

Reading the bottom row of this transition table, we get the binary number $01011010_{\text{bin}} = 90_{\text{dec}}$, allowing us to name this rule set Rule 90. Using this convention, we may now construct our own automata based solely on the knowledge of its name (at least for elementary cellular automata). For example, take Rule 110, introduced by Wolfram 1983 [3]. We first convert 110_{10} to binary, and then use this to fill in the table as above:

111	110	101	100	011	010	001	000		111	110	101	100	011	010	001	000
?	?	?	?	?	?	?	?	$\Rightarrow$	0	1	1	0	1	1	1	0

To convert this to a concrete set of rules to simulate in a computer programs (in a more elegant way than a collection of eight `if` statements), we can put the information we have gathered so far in a Karnaugh map to get a Boolean algebraic expression of the ruleset, as in the table below. We use A , B, and C to denote a_{i-1}, a_i, and a_{i+1}, at time t, respectively. From this we can reduce the ruleset to the Boolean expression in Eq. 3 by observation.

C \ AB	00	01	11	10
0	0	1	1	0
1	1	1	0	1

$$\begin{aligned} B^{t+1} &= \overline{A} \cdot C + B \cdot \overline{C} + \overline{B} \cdot C \\ &= \overline{A} \cdot C + B \oplus C \end{aligned} \tag{3}$$

The resulting visualisation in Fig. 4 exhibits similar characteristic of self-organisation, however it is hard to say whether it shows self-similarity as well.

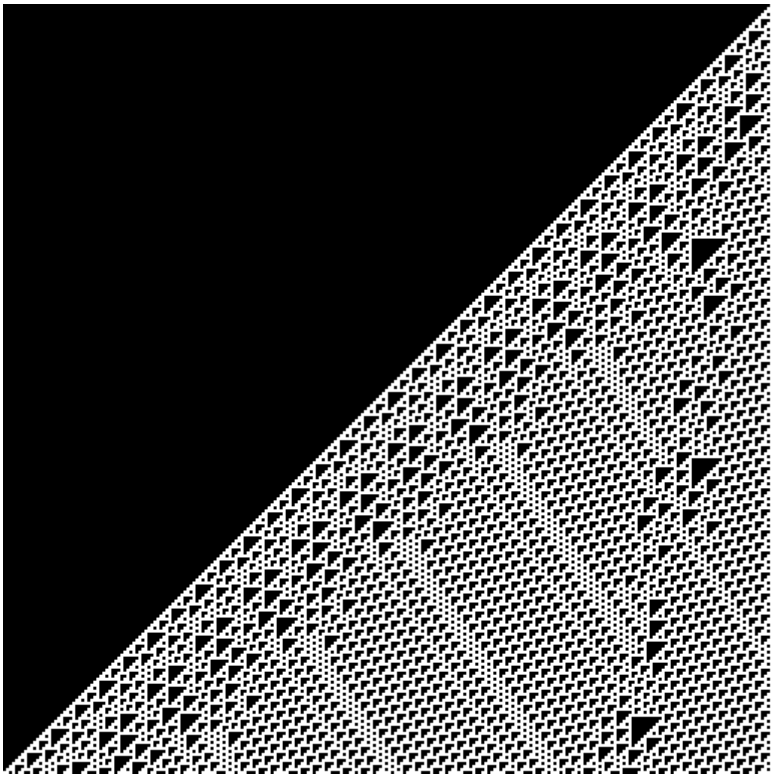


Figure 4: Rule 110 generated using Eq. 3

2.1 Elementary cellular automata

Elementary cellular automata are the simplest class of one-dimensional cellular automata; each cell may be in one of two states, 0 or 1, and rule sets only take into account the cell itself and its immediate neighbours (the nearest cell left and right of the cell being computed). Since there are a total of $2^3 = 8$ combinations of values for a neighbourhood of cells, and each of these neighbourhoods causes a cell to move to one of two states in the next generation, this means there are $2^8 = 256$ elementary cellular automata. Their evolution is often illustrated by each progressive "new line" showing the next generation of the automaton, as caused by applying the rule set to the current generation.

Two technically different automata may give rise to similar patterns, and indeed many automata are equivalent to each other thanks to simple transformations of theirunerlying geometry. The first of these transformations is reflection in the vertical axis; the result of applying

this is known as the mirror rule [4]. For example, take Rule 30. Each time some neighbourhood produces a live cell in Rule 30, let's take this neighbourhood and reflect it in the vertical plane, giving the table below:

111	110	101	100	011	010	001	000
0	0	0	1	1	1	1	0
0	1	0	1	0	1	1	0

The first line of the table shows the ruleset for Rule 30, while the line below gives that of our new ruleset, made by applying the mirror rule to Rule 30. Thus we get Rule 30's mirror rule Rule 01010110_{bin} = Rule 86_{dec}. Rules which generate the same ruleset under mirroring are called amphichiral, and of the 256 elementary cellular automata, 64 are so.

The second of these transformations is to exchange the roles of 0s and 1s in the ruleset definition, and doing so is applying the complement rule [4]. Again applying this to Rule 30, we get the table below (note that the roles of 0s and 1s are exchanged in both the top and bottom rows):

000	001	010	011	100	101	110	111
1	1	1	0	0	0	0	1

So we arrive at Rule $10000111_{\text{bin}} = 135_{\text{dec}}$ as the complement of Rule 30. There are 16 rules which are the same as their complement. Both of these rules can be applied at once to a ruleset, and the result is Rule 149 when applied to Rule 30. There are 16 rules which are the same as their mirrored complementary rules. There are 88 rules which are inequivalent under these transformations [4].

2.2 Wolfram classes

Wolfram gives the following classes for categorising the behaviours of different automata:

- Class One: rapidly converge to a uniform state e.g. Rule 0, 232
- Class Two: rapidly converge to a repetitive or stable state e.g. Rule 250
- Class Three: appear to remain in a random state e.g. Rule 150, 182
- Class Four: form areas of repetitive or stable states, but also structures which interact in complicated ways e.g. Rule 54

Two of these 256 rules are particularly special, namely Rule 54 and 110, both of which are Class 4 elementary cellular automata. Rule 110 has been proven to be universal, or Turing complete, meaning it is theorectically able to simulate any function. Meanwhile, it is so far unknown whether Rule 54 is capable of universal computation – interacting structures form, but it remains to been seen whether structures useful for computation can be formed. Computational universality is discussed more in-depth later on.

2.3 Two dimensional cellular automata

Two dimensional cellular automata become a bit more interesting than their dimensionally-deficient brethren. Rather than having a single row of cells to consider at each generation, now a two-dimensional plane of cells is used to compute the evolution of the automaton. Whereas elementary automata looked only at the neighbourhood of a_{i-1}, a_i, and a_{i+1} when applying the ruleset, two-dimensional automata have a wider choice. Two of the most common types of neighbourhood to consider are the Moore neighbourhood and the von Neumann neighbourhood.

References

[1] S. Wolfram, "Cellular automata," *Wolfram on Cellular Automata and Complexity*, p. 412, 1983.

[2] K. Clayton, "Basic concepts in nonlinear dynamics and chaos: Fractals and the fractal dimension." `https://www.vanderbilt.edu/AnS/psychology/cogsci/chaos/workshop/Workshop.html#Fractals`, 1996. From Society for Chaos Theory in Psychology and the Life Sciences annual meeting. Page accessed: 2018-08-03.

[3] S. Wolfram, "A new kind of science," 2002.

[4] "Elementary cellular automaton." `https://en.wikipedia.org/wiki/Elementary_cellular_automaton`. Page accessed: 2018-08-19.