CELLULAR AUTOMATA AS DETERMINISTIC FINITE-STATE MACHINES

Benjamin Phillips , Brandon Packard
Pennsylvania Western University - Clarion
phi53809@pennwest.edu, bpackard@pennwest.edu

ABSTRACT

The analysis of cellular automata has been long hindered by computational irreducibility. We can understand the simplistic rules that generate the larger emergent phenomena, but we struggle to find methods of analysis that allow us to predict the behavior of these cellular automata.

In this paper, we seek to generate a new methodology using the structures of finite-state machines in order to numerically analyze cellular automata by their structures rather than by their behaviors. We do this by taking each state of the automata as a collection of binary values, and feeding them into counting machines with final states "Alive" and "Dead" to generate the next state.

1 Introduction

The notion of emergence is one that resonates deeply within many fields. The observation of complicated phenomena arising from simplistic rule sets has a profound impact on the empirically reductive methods that philosophers and scientists have employed for centuries. Emergent phenomena are demonstrated evidences of a more holistic science where one cannot, by looking at the cogs, pipes, and wires of a machine, tell you what wonders it might perform.

The analysis of emergent phenomena generates a problem. The complexity of these systems make them almost impossible to predict a far future state from the given rules.

We see this same sort of problem in the world of chaos theory. Even with a known initial state, and a deterministic machine, our predictive capabilities falter the further we go from that initial state. Pendulums are a perfect example of this type of behavior. With a single pendulum in a vacuum we can predict its motion with almost perfect accuracy for the rest of time knowing only its length, the gravitational acceleration, and the initial angle. However, when you place a second pendulum, with its axis as the bottom of the first, the behavior becomes incredibly complex. With only one more set of variables, it is almost impossible to accurately determine the behavior of the double pendulum.

Finding methodologies to prevent the need for approximate measurements is critical for finding the long term behavior patterns of emergent, or chaotic, phenomena.

In this paper, we have attempted to generate a possible means of mathematically analyzing one of the most beautiful representatives of emergence: cellular automata.

2 Related Work

Other papers have worked on defining the properties of emergence within cellular automata. Hanson in [1] describes an emergent phenomenon as a property "that arises out of the system's own dynamical behavior, as apposed to being introduced from outside".

Hanson's analysis involves taking a collection of rules and pulling out given emergent phenomena such as what he calls , "synchronization".

The paper "Visualizing Computation in Large-Scale Cellular Automata"[2] delves deep into this topic from a slightly different angle. They aim to analyze automata through the use of the space-time diagrams of different rules. They utilized a process known as "course-graining" to determine which automata exhibited "interesting behaviors at multiple scales." Another paper "Problife: A Probabilistic Game of Life" [3] delves into the topic of probabilistic cellular automata that we describe in the "Reasons for the Method" section. They, rather than utilizing the method described in this paper, use the strategy of giving each cell in the automata a continuous value [0,1] rather than a discrete binary value.

Each of these papers attempts to tackle the structure and behavior of cellular automata. Our is, however, unique in its methodology.

3 Cellular Automata

A cellular automaton is a discrete structure that adapts over a given unit time according to some set of rules. It could be more generally described as a collection of entities that respond to their environment, evolving in specific ways given

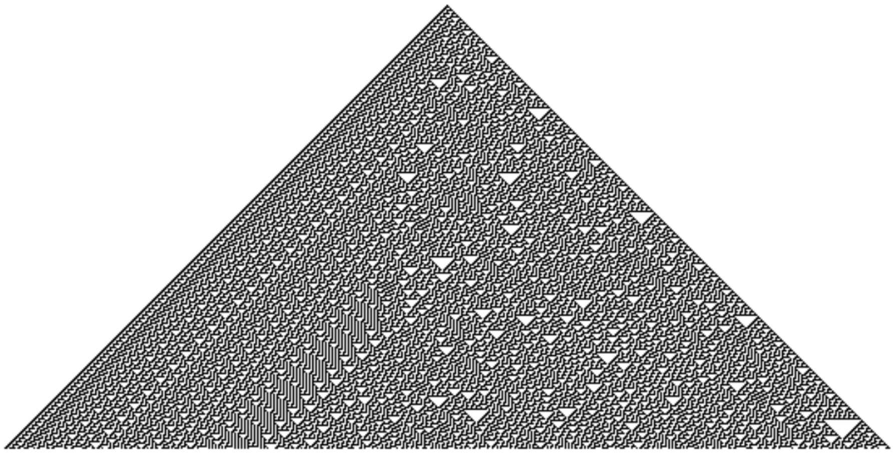


Figure 1: The resulting pattern of Rule 30.

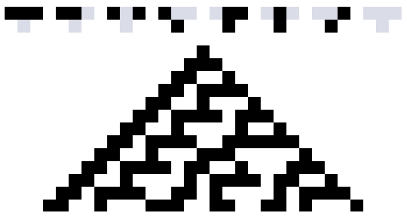


Figure 2: Rule 30.

particular parameters. This can occur in n-many dimensions, although we usually observe it in one or two.

The image in Figure 2 is of a one dimensional cellular automata known as rule 30. It was discovered by the renowned scientist Stephen Wolfram in 1983 [4]. It is one of 256 total rules in one dimension [5]. It evolves using the rules labeled above the structure. It is a one-dimensional automata that takes the initial row and scans three squares of it at a time. We can call these sections of three squares kernels. It applies the rules to each kernel and generates the new row. Over some number of steps, this rule generates the pyramidal shape.

We can see this behavior by looking at the very top of the pyramid. As we scan across the top row, we take the kernels containing the dark square. There is one kernel on the far right, one dead center, and one on the far left. Each of these, as can be seen by the rules, generates a dark square directly below the first row. Thus we now have a row with three dark squares. Keep repeating this process and you get immensely complex behavior, as can be seen in Figure 1. The behavior of this automata is so seemingly complex that Wolfram has offered large cash prizes if anyone can find a pattern in the center column of the pyramid.

Now we can look at the sort of quintessential cellular automata, often referred to as Conway's Game of Life. It is in two dimensions, but follows similarly simple rules. Just remember that rather than observing a kernel of three squares, now our kernel is nine squares. The rules, as stated by Conway, are:

1. Birth rule: An empty, or "dead," cell with precisely three "live" neighbors (full cells) becomes live.
2. Death rule: A live cell with zero or one neighbors dies of isolation; a live cell with four or more neighbors dies of overcrowding.
3. Survival rule: A live cell with two or three neighbors remains alive. [6]

These rules, despite their simplicity, lead to beautiful, almost life-like structures. Below there is a series of images representing concurrent stages of arguably the most famous structure in the game of life: the glider. It has the fascinating property that given an infinite grid, it will glide diagonally downward, step by step, forever.

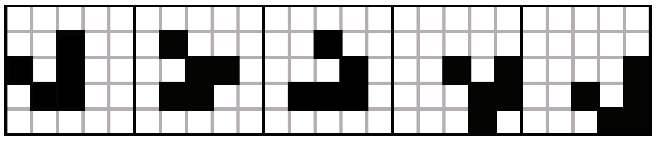


Figure 3: The stages of Conway's Glider

The especially fascinating thing about the Game of Life, is that its success is uncontested. It holds a fascinating uniqueness in its stability and controlled evolution. Why is this? What is so special about those three rules? The answer is that we don't quite know. Computational irreducibility is a brick wall that stands firmly in the way of comprehending any sort of conclusion regarding its behavior. However, in this paper we devise a method to form a model that is analogous to the automata itself whilst being more systemically analyzable.

This methodology, however, requires some notation.

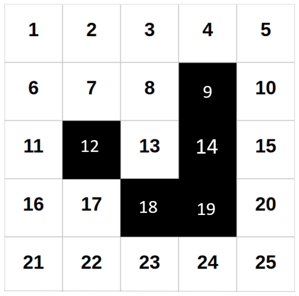


Figure 4: The initial glider phase with numerical assignments.

4 Binary Expansion, and Dependency Form

To begin, we need a way of numerically describing the state of a cellular automaton without directly using images. Ironically, we will demonstrate this strategy using images. The general sense of the method is that we are going to take a region within the automata space at state s and assign binary values to each discrete "chunk" of that space.

We will use those values as the inputs of a finite-state machine that will then churn out the next state s' after t = 1.

The way to assign values is as shown in Figure 4. Take a given grid of pixels, with dimensions LxW and label them in ascending order going left-to-right, top-to-bottom. Then going through in that order, depending on the state of each given cell, assign it a 1 or a 0. A light square is assigned a 0, and a dark square is assigned a 1. So for the image above: cell one is a 0, cell 2 is a 0 ... cell 9 is a 1, cell 10 is a 0, etc. The total state of the grid is then the binary sequence generated by that assignment process. For the glider in this grid we have a state sequence of: 0000000010010100011000000 or in decimal: 75968. This is the binary form of the automata at state "s" or time-step 0. This number is useful on its own, as it is a unique number that we can assign any glider at this stage in a 5x5 space (in which a glider exists at all times). Instead of glider in pose "x" we can say the glider in state: 75968.

Now we need to find a way to take this state sequence and utilize it to generate the next step of this automata. The way to do this is to find the "dependency sequences" of each cell in the grid. What we mean by this is that we need to take each cell and generate the binary sequence corresponding to itself and the eight cells around it. We do this as such: take the 3x3 kernel with cell 1 at its center. Since cell 1 is at the edge of our grid, we imagine empty cells in all places where the cell state is unknown.

We take this kernel and perform the same binary sequencing procedure we did with the larger grid, going from left to right row by row, assigning a 1 for dark cells and a 0 for light cells.

Thus cell 1 is assigned the sequence 000000000, cell 2 would be assigned 000000000, cell 3, 000000001, cell 4, 000000010...etc. These are the expanded forms of each cell.

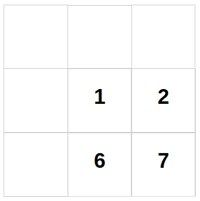


Figure 5: The grid describing the dependency of cell 1.

To take these expanded sequences into dependency sequences we simply take the binary value of the middle cell of each kernel and place it in front of the other eight. So cell 9 for example, with expanded form 000010010, would become: 100000010. The reasoning for this format will soon be apparent. Thus, the total dependency form of our 5x5 grid is:

1. 000000000
2. 000000000
3. 000000001
4. 000000010
5. 000000100
6. 000000001
7. 000000010
8. 000001101
9. 100000010
10.
11. 000010100
12. 000001000
13. 100000001
14. 000111011
15. 101000110
16. 010010100
17. 000100000
18. 001001000
19. 110101000
20. 101010000
21. 010010000
22. 000000000
23. 000100000
24. 001100000
25. 011000000
26. 010000000

The importance of these 9-bit sequences is that they are the language of a deterministic finite automata that we will use to describe Conway's Game of Life. It is graphically designed as such:

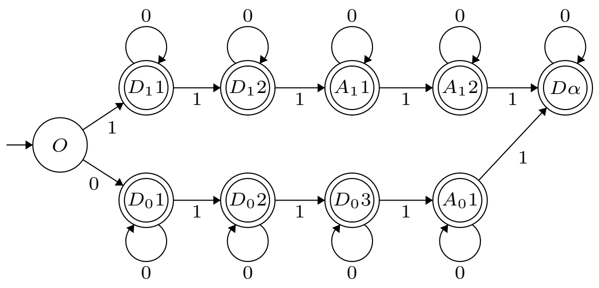


Figure 6: A Finite-State-Machine describing Conway's Game.

With state table:

	0	1
O	$D_0 1$	$D_1 1$
$D_0 1$	$D_0 1$	$D_0 2$
$D_0 2$	$D_0 2$	$D_0 3$
$D_0 3$	$D_0 3$	$A_0 1$
$A_0 1$	$A_0 1$	$D\alpha$
$D_1 1$	$D11$	$D12$
$D_1 2$	$D12$	$A11$
$A_1 1$	$A_1 1$	$A_1 2$
$A_1 2$	$A_1 2$	Dα
Dα	$\emptyset$	$\emptyset$

If we take our dependency sequences as strings and input them into our automaton, it returns either an A-state (Alive-state) or a D-state (Dead-state). We can demonstrate this with cells 1 and 19.

Cell 1: 000000000. This returns $D_0 1$

Cell 19: 101010000. This returns $A_1 1$

The subscript to the right of "D" and "A" is used to notate the original value of the square, which decides the branch of the machine it uses. The number on the right is the "nth" alive-state or the "ith" dead-state of each branch. The next step in the process is to take these generated states, and return them back into a state sequence by reassigning them a binary digit from their state. The rule will be that A-states are assigned a 1, and D-states are assigned a 0. So for our sequence we get the list:

1. 000000000$\rightarrow D_0 1 \rightarrow$ 0
2. 000000000$\rightarrow D_0 1 \rightarrow$ 0
3. 000000001$\rightarrow D_0 2 \rightarrow$ 0
4. 000000010$\rightarrow D_0 2 \rightarrow$ 0
5. 000000100$\rightarrow D_0 2 \rightarrow$ 0
6. 000000001$\rightarrow D_0 2 \rightarrow$ 0
7. 000000010$\rightarrow D_0 2 \rightarrow$ 0
8. 000001101$\rightarrow A_0 1 \rightarrow$ 1
9. 100000010$\rightarrow D_1 2 \rightarrow$ 0
10. 000010100$\rightarrow D_0 3 \rightarrow$ 0
11. 000001000$\rightarrow D_0 1 \rightarrow$ 0
12. 100000001$\rightarrow D_1 2 \rightarrow$ 0
13. 000111011$\rightarrow D\alpha \rightarrow$ 0
14. 101000110$\rightarrow A_1 2 \rightarrow$1
15. 010010100$\rightarrow A_0 1 \rightarrow$1
16. 000100000$\rightarrow D_0 1 \rightarrow$0
17. 001001000$\rightarrow D_0 2 \rightarrow$0
18. 110101000$\rightarrow A_1 2 \rightarrow$1
19. 101010000$\rightarrow A_1 1 \rightarrow$1
20. 010010000$\rightarrow D_0 3 \rightarrow$0
21. 000000000$\rightarrow D_0 1 \rightarrow$0
22. 000100000$\rightarrow D_0 1 \rightarrow$0
23. 001100000$\rightarrow D_0 2 \rightarrow$0
24. 011000000$\rightarrow D_0 2 \rightarrow$0
25. 010000000$\rightarrow D_0 1 \rightarrow$0

This process gives us the following binary state sequence: 0000000100000110011000000 or in decimal: 67168. This represents the state "s" of the automaton, or t = 1. Drawing it out we get this:

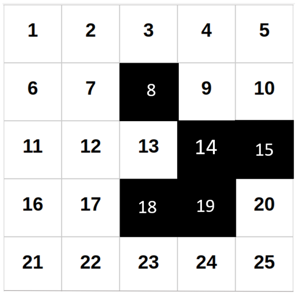
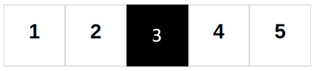


Figure 7: The second phase of our glider with numerical assignments.

This is exactly what our previously shown diagram said was the next step of the glider.

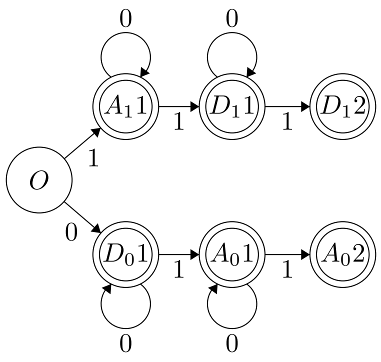
This process also works equally well for one-dimensional automata like Rule 30. The only difference is that given the position dependency of Rule 30, when we take it to expanded form, we will not place the center square at the front of the string. However, the process will look the same.



We first start with our initial state sequence, in this case; it is simply 00100. Each kernel in this case will be three bits. Taking our state sequence into our expanded list we get:

1. 000
2. 001
3. 010
4. 100
5. 000

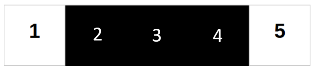
We then use this automata:



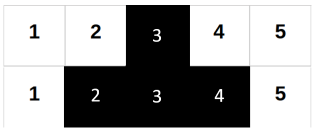
Which gives the new states:

1. 000 $\rightarrow D_0 1 \rightarrow 0$
2. 001 $\rightarrow A_0 1 \rightarrow 1$
3. 010 $\rightarrow A_0 1 \rightarrow 1$
4. 100 $\rightarrow A_1 1 \rightarrow 1$
5. 000 $\rightarrow D_0 1 \rightarrow 0$

Drawing this new frame we get:



This new row looks especially familiar when we place it underneath the previous frame as such:



5 Reasons for this Representation

The study of cellular automata goes beyond the theoretical.

5.1 Reason 1: Modeling Real World Phenomena

Cellular automata can be Turing complete systems that act as analogous representations of various real phenomena [7].

Many computational systems, whether social, biological, physical or theoretical, are ripe with concepts well represented by the discrete deterministic structures of cellular automata. For example, cellular automata have been used to represent prokaryotic behaviors evidencing the idea that single-cell organisms behave algorithmically [8]. Some cellular automata also follow the epigenetic principle of evolutionary biology. [9]

Having a means by which we can analyze not just the behavior, but the structure, of these machines is critical to understanding why they so well represent real world phenomena.

5.2 Reason 2: Numerical Comparison of Automata

The finite-state representation allows us to directly compare automata. We can use the methods of automata theory (closure, language analysis, etc.) to determine the equivalency, similarity, or dissimilarity of each machine.

We can also see, in a more numerical way, the structures that lead cellular automata to behave as they do.

We could try and ascertain these notions by the behavior or rules alone, but we quickly run into the problem of computational irreducibility. If we knew how they worked by the rules alone, the notion of emergence would be a moot subject.

5.3 Reason 3: Dimensional Reductionism

For certain experimentation where we want to ascertain the behavior of higher dimensional automata, analyzing the behavior via trial and error/visually is not particularly helpful.

The numerical machinery of the finite-state version allows for analyzing automata of any dimension. We already saw with the Rule 30 automata, and the Game of Life, that increasing the dimension of the automata increases the number of states.

In this paper, we took our numbers back into the visual for the sake of demonstration, but there is no need to do this. The binary representation works just as well and for much less computational cost.

5.4 Reason 4: Ease of Experimentation

The finite-state form of the automata allows us to easily modify the rules for the sake of experimentation.

Say, for example, we want to determine what happens when we allow for internal entropy in the cellular automata (Maybe for representation of quantum phenomena, or social behaviors). All that needs done to the model is to replace the inputs with probabilities, and we have a Markov chain with which we can generate stochastic matrices to represent entropic behavior.

This is merely one instance of where the numerical, finite-state representation serves as a benefit to computer scientists trying to analyze cellular automata.

6 Conclusion

In review: we have a process that takes a given region of some discrete space. It analyzes the alive or dead nature of each unit within this space and assigns the proper binary values. We call this the binary expansion, and it represents the state of the system in the region at some time t.

From these values we generate a dependency form where each unit of the region is given a sequence denoting its relationships with the units around it. This sequence is run through a finite-state machine corresponding to the cellular automata we are analyzing.

For ones such as Conway's game, and Rule 30, these automata are simple counting machines with final states represented by a D or an A, which then represents the dead or alive state of the unit after computation.

We take this state and generate a new binary expansion representing the regions state at time t+1. We can either use this new expansion to run the process again, or we can assign the values to some visual medium as demonstrated in the paper.

We hope to use this representation of cellular automata as a means by which to analyze the birth/death rates of various rules, and determine which machines seem to overpopulate, underpopulate, or maintain some sort of homeostasis.

We also plan on using genetic algorithms to generate rule-sets that optimize certain behaviors such as stable structures (like gliders), exponential growth patterns, etc. We will then use the finite-state representation to compare these rule-sets, and attempt to find the underlying cause of emergent behaviors.

References

[1] J. E. Hanson, Cellular automata, emergent phenomena in, *Computational Complexity*, page 325–335, doi:10.1007/978-1-4614-1800-9_22.

[2] H. Cisneros, J. Sivic, T. Mikolov, Visualizing computation in large-scale cellular automata, 2021.

[3] S. Vandevelde, J. Vennekens, Problife: A probabilistic game of life, 2022.

[4] E. Weisstein, Rule 30.

[5] F. Berto, T. Jacopo, Cellular automata, 2012.

[6] S. Roberts, The lasting lessons of john conway's game of life, 2020.

[7] L. Milano, Y. Liu, X. Feng, D. Haase, Cellular automata.

[8] R. Bowness, M. A. Chaplain, G. G. Powathil, S. H. Gillespie, Modelling the effects of bacterial cell state and spatial location on tuberculosis treatment: Insights from a hybrid multiscale cellular automaton model, *Journal of Theoretical Biology*, *446*:(2018), 87–100, doi:10.1016/j.jtbi.2018.03.006.

[9] L. Caballero, B. Hodge, S. Hernandez, Conway's "game of life" and the epigenetic principle, 2016.