Cellular Automata Lab 3 Report

Neo Nguyen

September 12, 2020

Abstract

In this experiment we were introduced to Wolfram elementary cellular automata and how it can generate interesting structures. We analysed 3 different set-ups that all generate the CA given by a specific ruleset, which resembles the same triangular fractals we previously examined.

1 Introduction and Theory

A cellular automaton is simply a grid of binary colored cells that evolve with each iteration, updating the current input as the previous generation. The coloring of each cell is decided upon certain rules that match the state of its neighbors against a binary number, which determines the value of the cell. In this lab, we are only working with one dimensional elementary CA which uses a single line of grid as a generation and the two nearest cells as the neighborhood of a cell. In the general case for this CA, each "neighborhood set" including a particular cell and its 2 neighbors may only have $2^3 = 8$ different possible values, since each particular cell only has 2 different possible values. We are interested in a rule set that maps each of these possible neighborhood sets onto a binary number, indicating whether that particular cell will be colored. Since we have 8 possible neighborhood sets, we will have $2^8 = 256$ possible binary rule sets for this kind of CA.

2 Rule 90

In this particular lab we are interested in rule 90, defined by:

current pattern	111	110	101	100	011	010	001	000
new state for center cell	0	1	0	1	1	0	1	0

Where 1 indicates a colored cell and 0 an empty white cell.
Looking closely at the rule set, it can be observed that the value of the new cell is equivalent to the logic expression "exclusive or" in respect to it's neighbors. In logic, the expression is equivalent to modulo 2 addition, which generates the modulo 2 version of pascals triangle when starting from a singular colored block in the first generation, namely the triangular

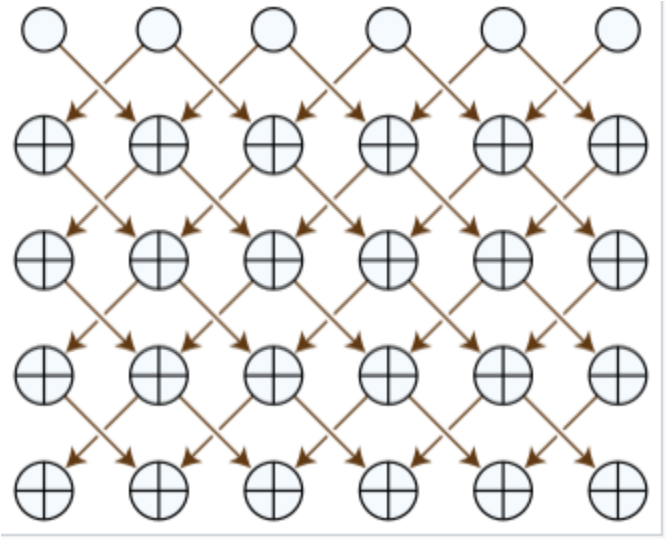
figure we have examined in the past.

3 Methods

Method 1: We start with a single colored block in the first generation. The excel sheet can then either be coded using the direct formula, examined in Method 3, or by simple evolutionary iteration. Using the XOR property of the rule 90, we take a cell in the first generation and examine its neighborhoods. From the ruleset, the only important values are the neighborhood values and if the are true in the XOR sense, then the block directly below that cell is colored. This process is repeated for every 3 blocks in the row and then throughout the generations.

Method 2: This method essentially follows the same iterative concept and uses a lookup python table to define the rule set. Through a basic algorithm, we fill a 100x100 matrix of initially empty values (except for the singular cell filled in the first generation), by matching the values of the neighboring cells to the lookup table, which tells us the resulting binary number.

Method 3: The method uses a condensed but equivalent version of the previous algorithms. As we have explored before, the rule can be logically expressed as the XOR function in respect to its neighbors, meaning it does not take into account what the value of the cell is. Due to this property, we can arrive at the following relationship graph:



The modular nature is displayed in the general formula:

```
X[i,j] = (rule/(2**(4*X[i-1,j-1] + 2*X[i-1,j] + X[i-1,j+1]))) % 2
```

Which is equivalent to the other methods.

4 Conclusions

We explored 3 different generation methods that work for every defined ruleset and generates the respective CA for it. Rather than use a more unintuitive formula in order to derive the structure, I personally believe the second method, using previous iterations of the model is a more effective and elaborating structure.