

Cellular Automata

Robert Thompson

Experimental Mathematics
Spring 2009

Contents

Introduction	2
The Approach	2
The Code for Generating the Automaton	2
A Simple Start	2
The Wolfram Rules	3
Interesting Rules	4
The Zero Rule—Rule 0	4
The One Rule—Rule 255	5
The Rule of Identity—Rule 204	6
The Rule of Inversion—Rule 51	7
Interesting Initial States	
Alternating 1's and 0's	7
Complicated Rules	9
Further Exploration	9
Lists of even length versus odd length	9
'Square' Automata	10
More Complicated Rules	10
Appendix I: The Code	11
Appendix II: Table of Interesting Rules	13
References	13

Introduction

A cellular automaton consists of a regular grid of cells, each cell in one of a finite number of states. For my project, I chose to examine cellular automata that—based upon inputs of sets of colors, initial cell-states, and rules—output a set number of levels of the resulting automaton. Then, after examination of the automata, I consider any patterns that may arise.

The Approach

With regards to figuring out how to approach the project in using Maple as an aide, some challenges arose. First of all, I wanted to be able to represent the cellular automata with actual graphics as opposed to strings of numbers. Further, to create subsequent rows of these graphics, I had to manipulate the data in such a way so as to create an easier means of plotting the data.

The Code for Generating the Automaton

What follows is a description of the code that was created for the sake of the project, what inputs are needed, and what outputs are generated. For the following code to function properly, the “plots” package for Maple must be loaded by using the command **with(plots)**. The code itself is attached to the end of this document.

acell($i, j, cellColor$) colors the i^{th} element of the j^{th} row from a set of values and colors it based upon the input color.

automaton(M) takes a matrix of values and outputs a set of colored cells based upon those values.

For the project, only the values of 0 and 1 were set for colors white and black respectively. However, plenty more colors may be coded in by appending the following code to **automaton**(M):

```
...
  if M[i,j]=n then coloredM[i,j]:=acell(i,-j,colorOfChoice):
  fi:
...
```

where n is the number that will be colored by *colorOfChoice*.

A Simple Start

What I wanted to do first was simply see how the automaton would look after I input my numerical values. So, I created a simple piece of test code that would do just that.

automatonrow(L) takes a list of values and outputs a string of colored cells based upon those values.

Consider, for example, if we had the input list $[1, 0, 1, 0, 1, 0, 1, 0]$. The corresponding ‘coloring’ is displayed:



Figure 1: Colored automaton $[1, 0, 1, 0, 1, 0, 1, 0]$.

The Wolfram Rules

Stephen Wolfram began working with cellular automata in the early 1980s, and created a rule system that is considered the standard today [4]. The rules he defined were that the value of a site in an automaton is defined by the site itself and its two nearest neighbors in the previous time step. This idea may be represented by the following functional notation:

$$a_i^{t+1} = F(a_{i-1}^t, a_i^t, a_{i+1}^t)$$

where a_i^{t+1} is the cell whose value is to be determined, and F is the function that operates on the three values of the previous time iteration, $a_{i-1}^t, a_i^t, a_{i+1}^t$. How this function is determined is by means of a binary number representation— with regards to this project and a majority of other research, from the values of 0 to 255 in binary, for simple rules of two colors. Consider a number in binary whose digits have been put into a list of length 8 that we will call B :

$$B = [b_8, b_7, b_6, b_5, b_4, b_3, b_2, b_1]$$

where b_i is the i^{th} value for the corresponding digit place in binary. Then, the Wolfram rules function is more specifically defined as follows:

$$F(x, y, z) = b_i, 4x + 2y + z + 1 = i$$

where F is the function applied on a combination of respective 1s and 0s, yielding the i^{th} position value for the rule. More explicitly shown:

$a_{i-1}^t, a_i^t, a_{i+1}^t$	1, 1, 1	1, 1, 0	1, 0, 1	1, 0, 0	0, 1, 1	0, 1, 0	0, 0, 1	0, 0, 0
a_i^{t+1}	b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1

To generate the rule set, instead of programming each individual rule into Maple, I wrote a procedure that would calculate them. First, a number from 0 to 255 would be input; then, that number would be converted to binary, and that binary representation put into list form. Finally, the rules would be assigned, and then applied to a given starting state and a desired number of rows.

Note. While Wolfram usually considers an infinite plane of his automata, I am considering a finite plane where the first element of a row is the right-hand neighbor of the last element of a row, and the last element of the row is considered the left-hand neighbor of the first element of a row.

Interesting Rules

What follows are some interesting Wolfram rules that I have noticed through the course of this project.

The Zero Rule—Rule 0

The name of the rule is relatively self-explanatory; that is, no matter what the initial state may be, the output will be a series of ‘empty’ rows, or rows with just zero-values. For example:

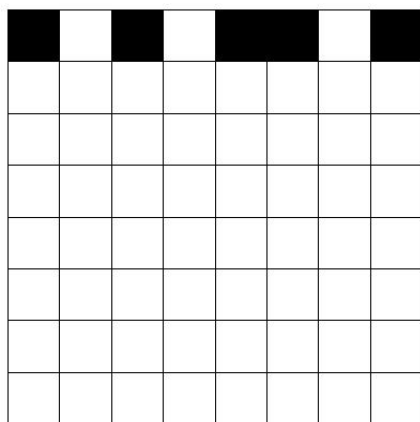


Figure 2: An example of the ‘zero rule’ applied.

Therefore, every corresponding value to the previous iteration will be a zero:

$$\frac{a_{i-1}^t, a_i^t, a_{i+1}^t}{a_i^{t+1}} \quad | \quad 1, 1, 1 \quad | \quad 1, 1, 0 \quad | \quad 1, 0, 1 \quad | \quad 1, 0, 0 \quad | \quad 0, 1, 1 \quad | \quad 0, 1, 0 \quad | \quad 0, 0, 1 \quad | \quad 0, 0, 0$$

The One Rule—Rule 255

Similar to the ‘zero rule,’ the ‘one rule’ holds to its title—that is, no matter what the initial state may be, the output will be a series of completely filled rows, or rows with just one-values. For example:

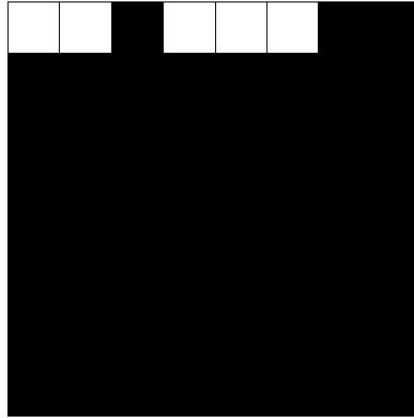


Figure 3: An example of the ‘one rule’ applied.

Once again, a table of values will perhaps demonstrate this better:

$a_{i-1}^t, a_i^t, a_{i+1}^t$	1, 1, 1	1, 1, 0	1, 0, 1	1, 0, 0	0, 1, 1	0, 1, 0	0, 0, 1	0, 0, 0
a_i^{t+1}	1	1	1	1	1	1	1	1

The Rule of Identity—Rule 204

Rule 204 of the Wolfram rule set may be thought of as the ‘rule of identity.’ This is because no matter what the neighboring cells for a cell in its previous iteration, that cell will remain the same value. In ‘picture form,’ this corresponds to a series of black and white columns. An example of the ‘rule of identity’ applied:

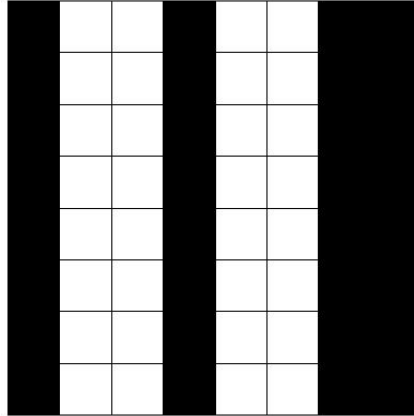


Figure 4: An example of the ‘rule of identity’ applied.

To determine what rule maintained identity, I simply had to look at the function, and which ones retained the cell state. That is, for a cell a_i^t , its value is the same value as a_i^{t-1} , with no regards to a_{i-1}^{t-1} or a_{i+1}^{t-1} . So, using the table of values, and substituting appropriate values for b_i :

$a_{i-1}^t, a_i^t, a_{i+1}^t$	1, 1, 1	1, 1, 0	1, 0, 1	1, 0, 0	0, 1, 1	0, 1, 0	0, 0, 1	0, 0, 0
a_i^{t+1}	1	1	0	0	1	1	0	0

The Rule of Inversion—Rule 51

Rule 51 of the Wolfram rule set may be thought of as a ‘rule of inversion,’ as it takes the given value of a cell state and immediately proceeding is its opposite, thus creating opposites of each row, or alternating between the initial values and their opposites. Here is an example:

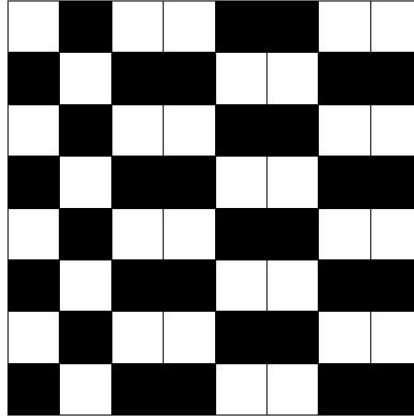


Figure 5: An example of the ‘rule of identity’ applied.

Once again, a table of values:

$a_{i-1}^t, a_i^t, a_{i+1}^t$	1, 1, 1	1, 1, 0	1, 0, 1	1, 0, 0	0, 1, 1	0, 1, 0	0, 0, 1	0, 0, 0
a_i^{t+1}	0	0	1	1	0	0	1	1

Interesting Initial States: Alternating 1’s and 0’s

If the initial state of the automaton is of the form $[1, 0, \dots, 1, 0]$ or of the form $[0, 1, \dots, 0, 1]$, then there are four previous states to consider: those where 0 has 1 as both its neighbors, or 1, 0, 1, where 1 has 0 as both its neighbors, or 0, 1, 0, and when the neighbors are the same as themselves, or 0, 0, 0 and 1, 1, 1. This is because, given that the initial states is solely an alternating sequence of 1’s and 0’s, one need only consider where the neighbors of a given cell are its opposites; if they are changed to being rows of all 1’s or all 0’s, that’s where the same-neighbors come into play.

Therefore, it follows that there are 16 types of rules to consider:

1, 1, 1	1, 0, 1	0, 1, 0	0, 0, 0
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

One way to think about these instances is if we let any number of this form be represented by $b_8b_7b_6b_5b_4b_3b_2b_1$, where b_i is the i^{th} digit of the binary representation of the number. So, the digits we are concerned with are those of b_1 , representing the place-holder for ‘1,’ b_3 , representing the place-holder for ‘4,’ b_6 , representing the place-holder for ‘32,’ and b_8 , representing the place-holder for ‘128.’ Hence, if the number is represented by $b_8b_7b_6b_5b_4b_3b_2b_1$, the following holds:

Rule (in binary)	Rule congruence
$0b_70b_5b_40b_20$	Zero-rule
$0b_70b_5b_40b_2b_1$	Alternating rows of white and black
$0b_70b_5b_41b_20$	Rule of Identity
$0b_70b_5b_41b_21$	Rule of Identity
$0b_71b_5b_40b_20$	Rule of Inversion
$0b_71b_5b_40b_21$	Rule of Inversion
$0b_71b_5b_41b_20$	Black row, followed by all white
$0b_71b_5b_41b_21$	Alternating rows of white and black
$1b_70b_5b_40b_20$	Zero-rule
$1b_70b_5b_40b_2b_1$	Alternating rows of white and black
$1b_70b_5b_41b_20$	Rule of Identity
$1b_70b_5b_41b_21$	Rule of Identity
$1b_71b_5b_40b_20$	Rule of Inversion
$1b_71b_5b_40b_21$	Rule of Inversion
$1b_71b_5b_41b_20$	One-rule
$1b_71b_5b_41b_21$	One-rule

Complicated Rules

Thus far, conjectures and conclusions made about patterns and formulas that arise have been noted. However, something must be said of apparent randomness generated by a rule for an automata. A famous example is of rule 30; claimed by Wolfram to be his ‘favorite’ rule, rule produces complex, seemingly-random patterns. It is used as a random number generator for the computer algebra system ‘Mathematica,’ and has also been suggested as a use in cryptography.[2] An example is shown below:

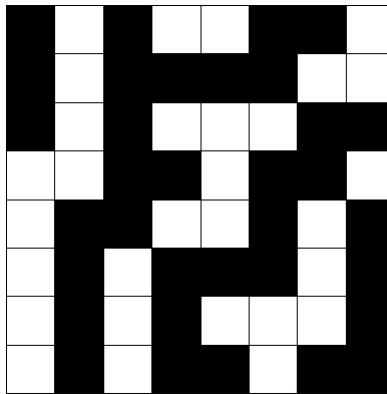


Figure 6: An example of the ‘rule 30’ applied.

Further Exploration

Lists of even length versus odd length

The differences in list length will obviously generate more complex automata; yet thus far, I have noticed that even incrementing a simple list by 1 element can have a drastic effect.

Notice below, in changing the list $[1, 0, 1, 0]$ to either $[1, 0, 1, 0, 1]$ or $[1, 0, 1, 0, 0]$, with rule 45 applied:

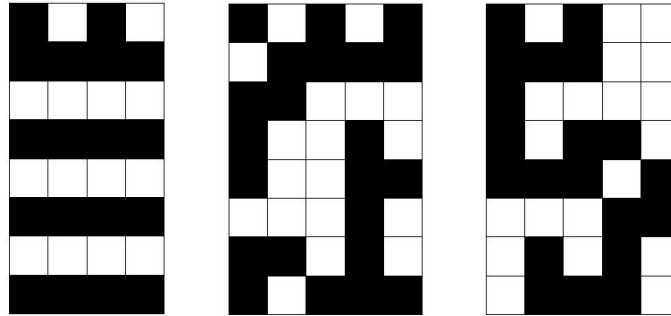


Figure 7: Rule 45 applied to (from left to right) $[1, 0, 1, 0]$, $[1, 0, 1, 0, 1]$, and $[1, 0, 1, 0, 0]$

‘Square’ Automata

The notion of a ‘square’ automaton is that, given an initial cell of length n , and $n - 1$ rows following, what cells corresponding to what rules cause the generated automaton to be a mirror image of itself along a diagonal? Along the horizontal? The horizontal between the two middle rows? Such an exploration would certainly yield interesting results.

More Complicated Rules

Certainly, complicated rules other than rule 30 exist (such as 86, 135, and 189[2]), but what is it that specifically makes them complicated? Are they more or less complicated based upon cell length, or would they be complicated in and of themselves?

Conclusion

Throughout the course of this semester, I have had the wonderful opportunity to work on this unique and challenging project; and, while several things were noted about cellular automata, clearly there is a vast amount of research that could still be performed. The use of a computer as an aide for the project was incredibly useful, and the project itself served to help me appreciate the ‘computer as crucible’ so much more. Having created code to generate cellular automata, with the ability to expand built-in, inspires me to keep pursuing the project and other ideas that the computer may help me to notice, assume, and solve.

Appendix I: The Code

```
with(plots):

acell:=(i,j,cellColor)->polygonplot([[1-j,-i],[-j,-i],[-j,1-i],[1-j,1-i]],
    color=cellColor,scaling=constrained,axes=None):

automaton:=proc(M) local rows, columns, i, j, coloredM:
    rows:=op(1,M)[1]:
    columns:=op(1,M)[2]:
    for i from 1 to rows do:
        for j from 1 to columns do:
            if M[i,j]=0 then coloredM[i,j]:=acell(i,-j,white):
            fi:
            if M[i,j]=1 then coloredM[i,j]:=acell(i,-j,black):
            fi:
        od:
    od:
    display(seq(seq(coloredM[a,b],b=1..columns),a=1..rows)):
end:

automatonrow:=proc(L) local M:
M:=LtoM(L):
automaton(M):
end:

LtoM:=proc(L) local M, i:
M:=Matrix(1,nops(L)):
    for i from 1 to nops(L) do:
        M[1,i]:=L[i]:
    od:
    return M:
end:

binarylist:=proc(b) local L:
    if b = 0 then return 0:
    fi:
    if b = 1 then return 1:
    fi:
    L:=[]:
    if type(b,odd) then L:=[op(binarylist((b-1)/10)),1]:
    fi:
    if type(b,even) then L:=[op(binarylist(b/10)),0]:
    fi:
    return L:
end:
```

```

rule:=proc(n) local bi,Rule,tempRule:
bi:=convert(n,binary):
Rule:=binarylist(bi):
tempRule:=Rule:
if nops(Rule) < 8 then Rule:=[seq(0(q), q=1..(8-nops(tempRule))), op(Rule)]:
fi:
return Rule:
end:

wolfram:=proc(S,r,n) local L,bi,a,b,c,M,i:
L:=rule(r):
M:=Matrix(n,nops(S)):
for i from 1 to nops(S) do:
M[1,i]:=S[i]:
od:
for a from 1 to n-1 do:
for b from 1 to nops(S) do:
if M[a,((b-2) mod nops(S))+1]=1 and M[a,b]=1 and M[a,(b mod nops(S)) +1]=1
then M[a+1,b]:=L[1]:
elif M[a,((b-2) mod nops(S))+1]=1 and M[a,b]=1 and M[a,(b mod nops(S)) +1]=0
then M[a+1,b]:=L[2]:
elif M[a,((b-2) mod nops(S))+1]=1 and M[a,b]=0 and M[a,(b mod nops(S)) +1]=1
then M[a+1,b]:=L[3]:
elif M[a,((b-2) mod nops(S))+1]=1 and M[a,b]=0 and M[a,(b mod nops(S)) +1]=0
then M[a+1,b]:=L[4]:
elif M[a,((b-2) mod nops(S))+1]=0 and M[a,b]=1 and M[a,(b mod nops(S)) +1]=1
then M[a+1,b]:=L[5]:
elif M[a,((b-2) mod nops(S))+1]=0 and M[a,b]=1 and M[a,(b mod nops(S)) +1]=0
then M[a+1,b]:=L[6]:
elif M[a,((b-2) mod nops(S))+1]=0 and M[a,b]=0 and M[a,(b mod nops(S)) +1]=1
then M[a+1,b]:=L[7]:
elif M[a,((b-2) mod nops(S))+1]=0 and M[a,b]=0 and M[a,(b mod nops(S)) +1]=0
then M[a+1,b]:=L[8]:
fi:
od:
od:
return M:
end:

```

Appendix II: Table of Interesting Rules

Rule	1,1,1	1,1,0	1,0,1	1,0,0	0,1,1	0,1,0	0,0,1	0,0,0	Special Designation
0	0	0	0	0	0	0	0	0	Zero-Rule
51	0	0	1	1	0	0	1	1	Rule of Inversion
204	1	1	0	0	1	1	0	0	Rule of Identity
255	1	1	1	1	1	1	1	1	One-rule

References

- [1] “Cellular automaton.” *Wikipedia, The Free Encyclopedia*. 17 Feb 2009, 12:05 UTC. 26 Feb 2009 http://en.wikipedia.org/wiki/Cellular_automaton.
- [2] “Rule 30.” *Wikipedia, The Free Encyclopedia*. 18 Apr 2009, 12:05 UTC. 3 May 2009 http://en.wikipedia.org/wiki/Rule_30.
- [3] Wolfram, Stephen. “Cellular Automata.” *Los Alamos Science* 9 (1983): 2-21. Stephen Wolfram. 1 Mar. 2007. 22 Apr. 2009 <http://www.stephenwolfram.com/publications/articles/ca/83-cellular/>.
- [4] Wolfram, Stephen. “Historical Notes: History of cellular automata.” *A New Kind of Science*. Wolfram Media, 2002. 876. Wolfram Science. 22 Apr. 2009 <http://www.wolframscience.com/reference/notes/876b>.