# *One Dimensional Cellular Automata:* THE TOTALISTIC APPROACH

JONAKI B GHOSH & ROHIT ADSULE

The topic of cellular automata has many interesting and wide ranging applications to real life problems emerging from areas such as image processing, cryptography, neural networks, developing electronic devices and modelling biological systems. In fact cellular automata can be a powerful tool for modelling many kinds of systems. In the March 2018 issue of At Right Angles we had introduced the basic ideas which form the foundation of the *Elementary Cellular Automata* (ECA) as defined by Stephen Wolfram. The reader is urged to go through the article before reading this.

The topic of Cellular Automata lends itself to interesting investigations which are well within the reach of high school students. We had illustrated the simple and yet powerful ideas in the previous article where we had described and analysed the behaviour of the 256 ECAs. In this article we shall provide a brief recap for the first time reader before moving on to the concept of Totalistic Cellular Automata.

Briefly defined, a *cellular automaton* is a collection of cells on a grid of a specified shape that evolves through discrete time steps according to a set of rules based on the state (or color) of the neighbouring cells. Cellular Automata may be one, two or three – dimensional. In this article and in the earlier one we have limited ourselves to exploring the one dimensional cellular automata on a grid of square cells where each row of the grid represents a generation or an iteration of the automata.

*Keywords: cellular automata, grids, colour, state, neighbouring, rule, pattern, generation, iteration*

The defining characteristics of a cellular automaton are

i. A grid of cells

ii. Each cell has a *state* – dead or alive. Cells which are alive may be coloured black or numbered 1 and cells which are dead may be numbered 0 and are white.

iii. Each cell in the grid has a *neighbourhood*. A neighbourhood of a given cell is a set of cells which are adjacent to it. This may be chosen in various ways. E.g., if we consider a linear grid of square cells, then the neighbourhood of each cell will be the two adjacent cells – one to its left and the other to its right.

iv. Finally every cellular automaton must have a *defining rule* based on which it grows and evolves in discrete time steps. For example, in a square grid, each row of cells may be considered as a separate generation of cells. Thus the first row is the initial generation (or generation 0) where each cell has a state (0 or 1). The state of each cell in the second row must be a function of its neighbouring cells in the row above it (that is the initial row). This may be written as

    (Cell $\text{state}_t$) = f(Neighbouring Cell $\text{state}_{t-1}$)

To begin with let us consider a linear grid of 8 cells where every cell has state 0 except the 5th cell which has a state 1.

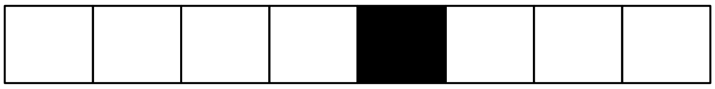

Figure 1. A linear grid of 8 cells where the 5th cell is a live cell.

This linear grid of square cells will be referred to as **generation 0** (or row 0). The states of cells in **generation 1**(that is row 1) will be determined by the neighbourhood of each cell in the row 0 which comprises of the three cells just above it. Clearly the states of these three cells may be any one of the following

000 001 010 100 011 101 110 111

The fact that in any three – cell neighbourhood, there are three cells each with state value either 0 or 1 implies that there are $2^3 = 8$ ways of colouring these cells. Thus there are 8 neighbourhood configurations described by the triples of 0's and 1's as shown above. However conventionally, while defining an ECA, these neighbourhoods are taken in the following specific order.

111 110 101 100 011 010 001 000

Each of these configurations will determine the state of the middle cell of the three cells just below it in the next row, which may again be either 0 or 1. However the state of the leftmost corner cell in row 1 will be determined by the state of the cell just above it in row 0, its right neighbour and the last cell in row 0. Similarly, the state of the rightmost corner cell in row 1 will be determined by the state of the cell just above it in row 0, its left neighbour and the first cell in row 0.

Let us now arbitrarily assign 0's and 1's to all 8 neighbourhood configurations as follows

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

Pictorially this may be represented as

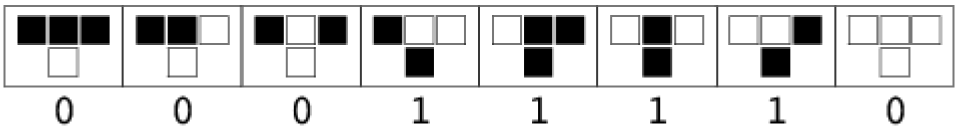

Figure 2. A rule set for a one-dimensional cellular automaton

This arbitrary assignment (also known as the *rule set*) will be the *defining rule* which will determine how this particular automaton will evolve. Note that this defining rule 00011110 may be treated as a binary number whose decimal representation may be obtained as follows

$$0 \times 2^7 + 0 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 30$$

This kind of a rule set generates an *elementary cellular automaton*. The ECA which evolves from this rule set is referred to as **Rule 30**. However a different assignment of 0's and 1's would lead to a different rule set and a different ECA. Since each of the 8 groups of three cells may be assigned 0 or 1, this leads to $2^8 = 256$ possible assignments. Thus, in all, there are 256 ECA rules.

In our previous article we had used Mathematica, a powerful Computer Algebra System and NICO an open source software to explore the 256 ECAs. Both Mathematica and NICO were used to obtain the graphic (pictorial) representations of all the 256 ECAs. We observed the evolutionary pattern of each ECA through the first 100 iterations and categorised the ECAs into specific classes based on the patterns manifested by them.

## A Totalistic Cellular Automaton

In this article we shall focus on the notion of generating a cellular automaton based on the totalistic approach. A *totalistic cellular automaton* is a cellular automaton in which the rules depend only on the total (or equivalently, the average) of the values of the cells in a neighbourhood. Wolfram introduced this idea in 1983. Like an ECA, the evolution of a one-dimensional totalistic cellular automaton can be completely described by a rule specifying the state a given cell will have in the next generation based on the *sum* of the values of the three cells consisting of the cell just above it in the grid, the one to its left and the one to its right.

Let us consider the case of a three cell neighbourhood (as described in the earlier section) where each cell has a value of 0 or 1.

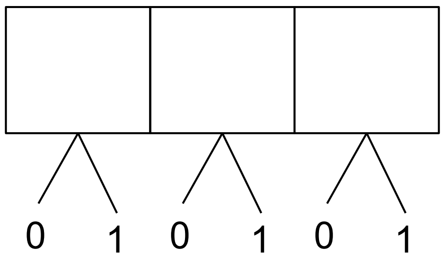

Figure 3. A three- cell neighbourhood

Note that each neighbourhood of three cells will have a total value of 0 when all the three cells have value 0 as shown.

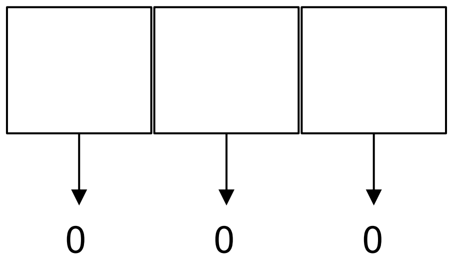

Figure 4. A three cell neighbourhood where all cells have state 0.

The total value will be 1 when one of the three cells has value 1 and the other two have value 0. This can happen in $3_{C_1} = 3$ ways.

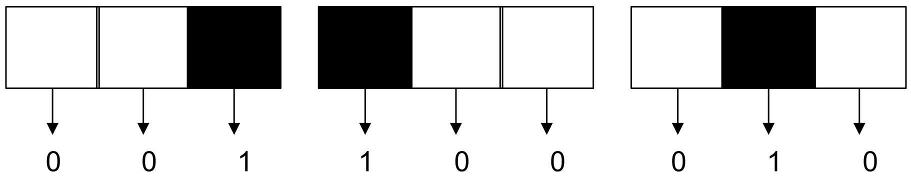

Figure 5. Three – cell neighbourhoods with total value equal to 1.

Similarly for a total value of 2, two out of the three cells must have value 1 which can happen in $3_{C_2} = 3$ ways.

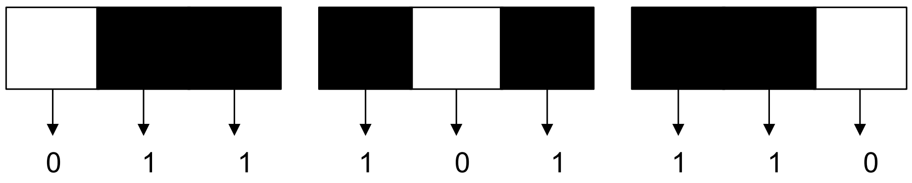

Figure 6. Three – cell neighbourhoods with total value equal to 2.

Finally the total value of 3 occurs when all cells have value 1 and this can occur in only 1 way.

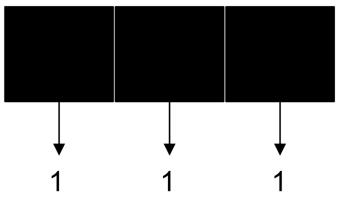

Figure 7. A three - cell neighbourhood with total value equal to 3.

Thus while we had $2^3 = 8$ neighbourhood possibilities in the ECAs, here we have only 4 neighbourhood possibilities – those with values 0, 1, 2 and 3. Further, a neighbourhood total of 0 may be assigned to a cell numbered 0 or 1. Similarly neighbourhood totals of 1, 2 and 3 can be also assigned to either 0 or 1. Thus there are only $2^4 = 16$ possible totalistic cellular automata (with three cell neighbourhoods) whereas there were $2^8 = 256$ ECAs! The totalistic approach considerably reduces the number of possible automata.

## A 5 – cell totalistic cellular automaton

In our project we decided to explore the case of an automaton with five cell neighbourhoods. A five cell neighbourhood can have $2^5 = 32$ possible colourings if each cell is assigned values 0 or 1. Some examples are shown in Figure 8. Further, all these 32 neighbourhoods can lead to a cell with state 0 or 1. Hence there will be $2^{32} = 4294967296$ cellular automata with five cell neighbourhoods. We realised that it would be too difficult to explore such a large number of cases.

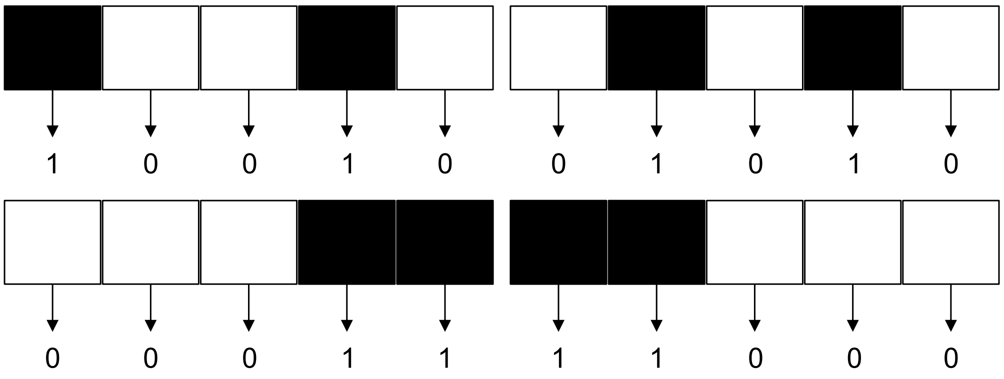

Figure 8. Five - cell neighbourhoods or a totalistic cellular automaton.

However if we go by the totalistic approach, the sum of the values of a 5 – cell neighbourhood can be 0, 1, 2, 3, 4 or 5, thus reducing the neighbourhood possibilities to 6 only! Each of these neighbourhoods can be assigned to 0 or 1 in the next generation of cells thus leading to $2^6 = 64$ possible automata rules. We have chosen to number the rules from 0 to 63. Thus rule 0 will represent the case when all total neighbourhood values (from 0 to 5) are assigned to 0. Similarly, rule 63 will represent the case when all total neighbourhood values (from 0 to 5) are assigned to 1. Since these two rules lead to all white cells or all black cells in subsequent generations, they may be referred to as trivial cases.

As a non-trivial case, let us consider **rule 53** which has a binary representation 110101 (as it can be expressed in powers of 2 as $1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$).

In order to define rule 53, we shall place the binary digits of 53 in correspondence with the sum values starting with 5 till 0. This leads us to the following rule

| **Sum** | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|
| **Assignment of 0 or 1** | 1 | 1 | 0 | 1 | 0 | 1 |

Table 1. Rule 53 totalistic cellular automata rule

When applied on a grid of square cells with a single live cell in the centre of row 1, this rule leads to the following intricate and symmetric triangular pattern.

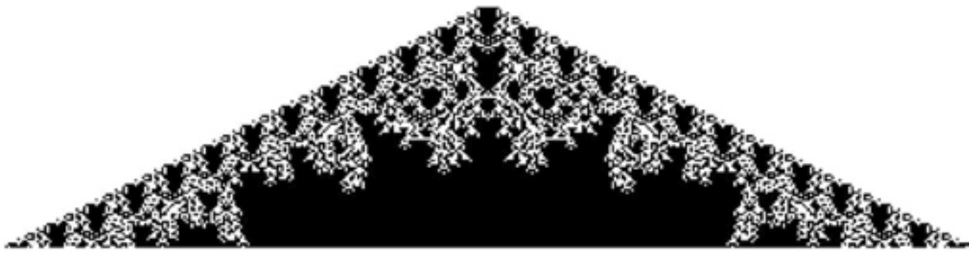

Figure 9: Rule 53

Table 2 lists out the defining rules of all 64 totalistic automata. Note that in any row of the table we will find the binary digits of the corresponding rule number in the columns numbered 5,4,3,2,1 and 0.

| **Sum** | **5** | **4** | **3** | **2** | **1** | **0** |
|---|---|---|---|---|---|---|
| Rule 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Rule 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| Rule 2 | 0 | 0 | 0 | 0 | 1 | 0 |
| Rule 3 | 0 | 0 | 0 | 0 | 1 | 1 |
| Rule 4 | 0 | 0 | 0 | 1 | 0 | 0 |
| Rule 5 | 0 | 0 | 0 | 1 | 0 | 1 |
| Rule 6 | 0 | 0 | 0 | 1 | 1 | 0 |
| Rule 7 | 0 | 0 | 0 | 1 | 1 | 1 |
| Rule 8 | 0 | 0 | 1 | 0 | 0 | 0 |
| Rule 9 | 0 | 0 | 1 | 0 | 0 | 1 |
| Rule 10 | 0 | 0 | 1 | 0 | 1 | 0 |
| Rule 11 | 0 | 0 | 1 | 0 | 1 | 1 |
| Rule 12 | 0 | 0 | 1 | 1 | 0 | 0 |
| Rule 13 | 0 | 0 | 1 | 1 | 0 | 1 |
| Rule 14 | 0 | 0 | 1 | 1 | 1 | 0 |
| Rule 15 | 0 | 0 | 1 | 1 | 1 | 1 |
| Rule 16 | 0 | 1 | 0 | 0 | 0 | 0 |
| Rule 17 | 0 | 1 | 0 | 0 | 0 | 1 |

| Sum | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|
| Rule 18 | 0 | 1 | 0 | 0 | 1 | 0 |
| Rule 19 | 0 | 1 | 0 | 0 | 1 | 1 |
| Rule 20 | 0 | 1 | 0 | 1 | 0 | 0 |
| Rule 21 | 0 | 1 | 0 | 1 | 0 | 1 |
| Rule 22 | 0 | 1 | 0 | 1 | 1 | 0 |
| Rule 23 | 0 | 1 | 0 | 1 | 1 | 1 |
| Rule 24 | 0 | 1 | 1 | 0 | 0 | 0 |
| Rule 25 | 0 | 1 | 1 | 0 | 0 | 1 |
| Rule 26 | 0 | 1 | 1 | 0 | 1 | 0 |
| Rule 27 | 0 | 1 | 1 | 0 | 1 | 1 |
| Rule 28 | 0 | 1 | 1 | 1 | 0 | 0 |
| Rule 29 | 0 | 1 | 1 | 1 | 0 | 1 |
| Rule 30 | 0 | 1 | 1 | 1 | 1 | 0 |
| Rule 31 | 0 | 1 | 1 | 1 | 1 | 1 |
| Rule 32 | 1 | 0 | 0 | 0 | 0 | 0 |
| Rule 33 | 1 | 0 | 0 | 0 | 0 | 1 |
| Rule 34 | 1 | 0 | 0 | 0 | 1 | 0 |
| Rule 35 | 1 | 0 | 0 | 0 | 1 | 1 |
| Rule 36 | 1 | 0 | 0 | 1 | 0 | 0 |
| Rule 37 | 1 | 0 | 0 | 1 | 0 | 1 |
| Rule 38 | 1 | 0 | 0 | 1 | 1 | 0 |
| Rule 39 | 1 | 0 | 0 | 1 | 1 | 1 |
| Rule 40 | 1 | 0 | 1 | 0 | 0 | 0 |
| Rule 41 | 1 | 0 | 1 | 0 | 0 | 1 |
| Rule 42 | 1 | 0 | 1 | 0 | 1 | 0 |
| Rule 43 | 1 | 0 | 1 | 0 | 1 | 1 |
| Rule 44 | 1 | 0 | 1 | 1 | 0 | 0 |
| Rule 45 | 1 | 0 | 1 | 1 | 0 | 1 |
| Rule 46 | 1 | 0 | 1 | 1 | 1 | 0 |
| Rule 47 | 1 | 0 | 1 | 1 | 1 | 1 |
| Rule 48 | 1 | 1 | 0 | 0 | 0 | 0 |
| Rule 49 | 1 | 1 | 0 | 0 | 0 | 1 |
| Rule 50 | 1 | 1 | 0 | 0 | 1 | 0 |
| Rule 51 | 1 | 1 | 0 | 0 | 1 | 1 |
| Rule 52 | 1 | 1 | 0 | 1 | 0 | 0 |
| Rule 53 | 1 | 1 | 0 | 1 | 0 | 1 |
| Rule 54 | 1 | 1 | 0 | 1 | 1 | 0 |
| Rule 55 | 1 | 1 | 0 | 1 | 1 | 1 |
| Rule 56 | 1 | 1 | 1 | 0 | 0 | 0 |
| Rule 57 | 1 | 1 | 1 | 0 | 0 | 1 |
| Rule 58 | 1 | 1 | 1 | 0 | 1 | 0 |
| Rule 59 | 1 | 1 | 1 | 0 | 1 | 1 |
| Rule 60 | 1 | 1 | 1 | 1 | 0 | 0 |
| Rule 61 | 1 | 1 | 1 | 1 | 0 | 1 |
| Rule 62 | 1 | 1 | 1 | 1 | 1 | 0 |
| Rule 63 | 1 | 1 | 1 | 1 | 1 | 1 |

Table 2.

**Classification of the 5 – cell totalistic cellular automata (TCA)**

We decided to explore all 64 TCAs by writing a program in Java (see Appendix I). The program produces the graphic image of the first 100 iterations of a specified rule number. After observing all 64 TCAs we tried to classify them based on their evolutionary behaviour. Interestingly, we found that the TCAs may be classified into four major categories which are very similar to the case of the ECAs. These categories are also mentioned in the research literature associated with cellular automata.

1. **Uniform:** All cells in the grid are either black or white.
2. **Repetitive:** These automata reveal regular alternating pattern or a block of cells which repeat themselves throughout the grid.
3. **Nested or Fractal-like:** These automata lead to Sierpinski triangle like fractal patterns exhibiting clear self- similarity or other nested patterns.
4. **Random or chaotic:** These are patterns which cannot be placed in any of the above three categories. There is no fixed pattern in these automata and their evolution is highly unpredictable.

The rule numbers which fit into the above categories are listed in the following table.

| Category | TCA rule number | Characteristics |
|---|---|---|
| Uniform | 0,4,8,12,16 (multiples of 4) | All cells are white (trivial case) |
| | 63 | All cells are black (trivial case) |
| | 54, 57, 58, 59, 61, 62 | Black cells with a border of black and white cells |
| Repetitive | 1,3,5,7,11,13,15,31 | Repetitive pattern of rows of black and white cells. 7,11,13,15 and 31 have a different configuration at the border. |
| | 19, 23 | Alternating rows of black and white with a central band pattern. |
| Nested | 2, 6, 14, 30, 33, 34 | Sierpinski triangle like structure |
| | 21, 25, 38, 42, 49 | Nested but not Sierpinski like |
| Random | 17, 29, 35, 47 | A generally random pattern with a border which gets wider as the iterations increase. |
| | 9, 27, 41, 45, 50, 51 | A generally random pattern with a uniform border which remains the same as the iterations increase. |
| Chaotic | 10, 18, 22, 26, 33, 43, 46 | A seemingly chaotic behaviour (although symmetrical) |
| All black with a fern-like repetitive pattern in the border | 53, 54 | It is difficult to place this in any of the above categories. |

Table 3. Classification of the 64 Totalistic Cellular Automata.

Here are some examples of TCAs which evolve from one live cell in the centre of the top row of the grid. One may note that all TCAs are symmetrical about the height of the triangular pattern. While describing them we will focus on the evolution of the cells in one half of the triangular pattern.

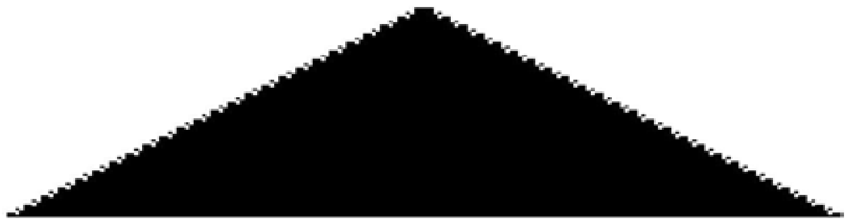
Figure 10. Rule 57: Uniform - a black triangle is formed with a border of white cells.

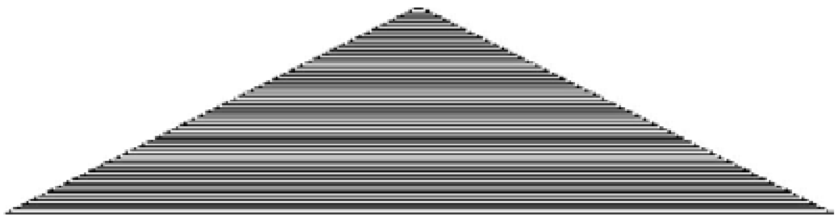
Figure 11. Rule 3: Repetitive pattern comprising rows of black and white cells

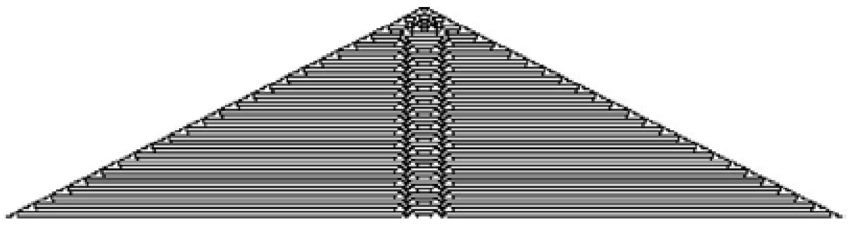
Figure 12. Rule 23: Alternating rows of black and white with a central band pattern.

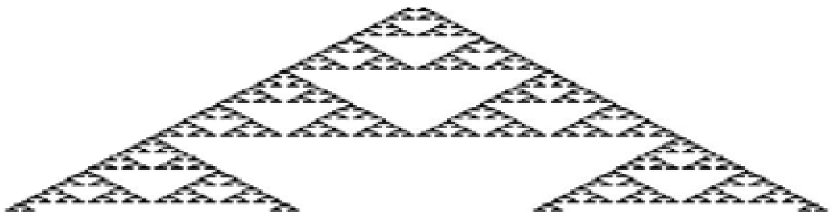
Figure 13. Rule 6: Sierpinski triangle like structure

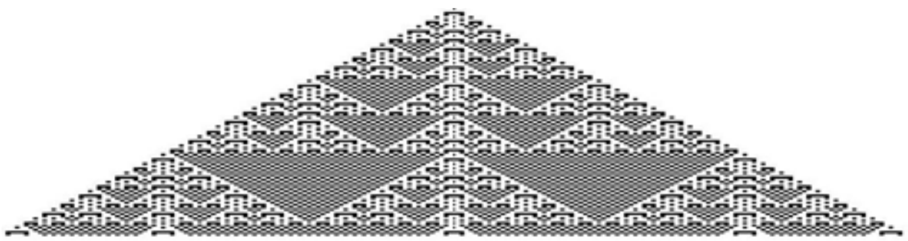
Figure 14. Rule 25: Nested Structure

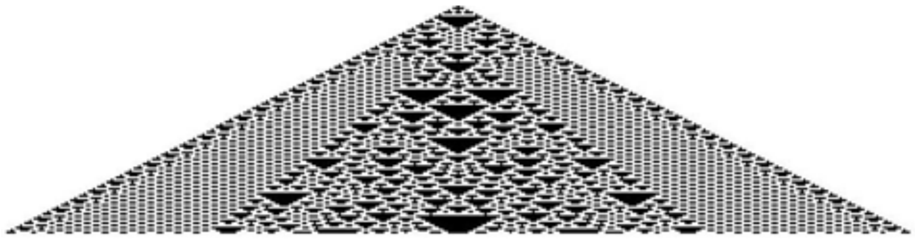
Figure 15. Rule 35: A generally random pattern with a border which gets wider as the iterations increase.

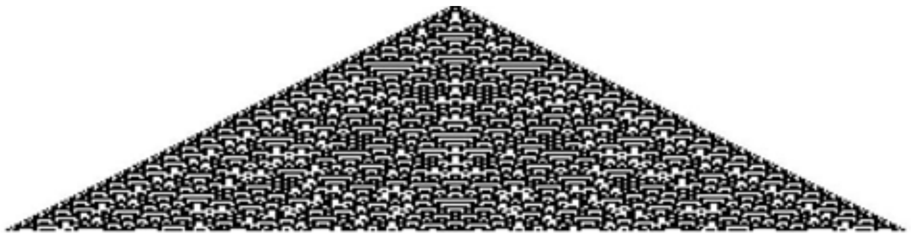
Figure 16. Rule 27: A generally random pattern with a uniform repetitive border.

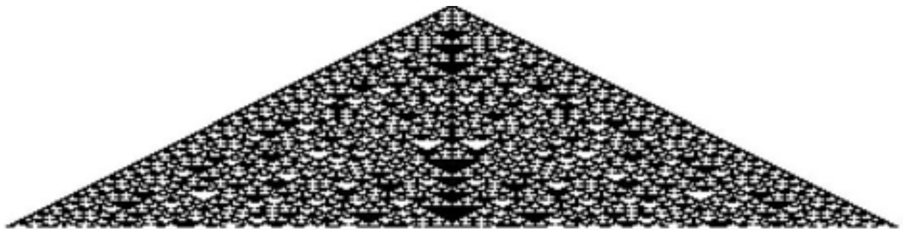
Figure 17. Rule 46: A seemingly chaotic pattern

## Concluding discussion

In this article we have described the second phase of our exploratory study on one - dimensional Cellular Automata. In order to define our own cellular automata we adopted the totalistic approach where the state of a given cell is dependent on the sum of values of its five neighbouring cells (the cell just above it, two to the left and two to the right) in the previous iteration. These led to neighbourhood values ranging from 0 to 5, each of which can give rise to a cell with value 0 or 1. This defining rule led to 64 cellular automata with some interesting patterns. We developed a code in Java to study their evolutionary patterns. The results were interesting as we were able to classify the 64 automata into four major categories – Uniform, Repetitive, Nested and Random. However an additional category with a repetitive fern-like pattern in the border was also identified which did not fit into the other four categories. The results of our investigations have been compiled in Table 2.

In this project we have explored two colour (two – state) ECAs and TCAs. It would be interesting to explore other kinds of automata which emerge when there are more than two states. A treatment may be found in http://mathworld.wolfram.com/TotalisticCellularAutomaton.html.

We hope we have succeeded in taking the reader on an exciting journey into the computational world of the one - dimensional cellular automata!

# APPENDIX I

JAVA Code for the Totalistic Cellular Automaton

```
import java.awt.*;
import java.awt.Color;
import java.awt.Graphics;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.imageio.ImageIO;
import java.awt.image.BufferedImage;
import java.io.*;
public class Processor extends JPanel
{
  static int line=0;
  static int max_lines=120;
  static BufferedImage[]images=new BufferedImage[max_lines];
  public Processor()
  {
    setSize(1000,1000);
  }

  @Override
  public void paintComponent(Graphics g)
  {
    System.out.println("paintComponent called from "
+ Thread.currentThread().getName());
    //new Exception().printStackTrace(System.out);
    super.paintComponent(g);
    if(line==0)
    {
      g.setColor(Color.WHITE);
      g.fillRect(0,0,1000,1000);
      g.setColor(Color.BLACK);
      g.fillRect(500,0,2,2);
    }
    else if(line!=0)
    {
      System.out.println("Line-1="+(line-1));
      g.drawImage(images[line-1],0,0,null);
      int x_cor=500-line*4;
      int y_cor=0+line*2;//The x and y coordinates of the points where the new squares are to be added from
      for(int lv=1;lv<=4*(line+1)-3;lv++)
      {
        BufferedImage img=images[line-1];
        if(lv>1)
          x_cor=x_cor+2;//Value adjusted for each square to be drawn in the same line
        int test_x=x_cor-3;
        int test_y=y_cor-1;
        int sum=0;
        for(int lv2=1;lv2<=5;lv2++)
        {
          System.out.println("(x,y):"+test_x+","+test_y);
          Color c=new Color(img.getRGB(test_x,test_y));
          if(c.getRed()==0&&c.getGreen()==0&&c.getBlue()==0)
          {
            sum+=1;
          }
          test_x=test_x+2;
        }
```

```
                if(sum==5||sum==4||sum==2||sum==0)//Set rule number here
                {
                    g.fillRect(x_cor,y_cor,2,2);
                }
            }
        }
    }

    public static BufferedImage toBufferedImage(Component component)
    {
        BufferedImage image = new BufferedImage(component.getWidth(), component.getHeight(), BufferedImage.
TYPE_INT_RGB);
        Graphics g = image.getGraphics();
        component.paint(g);
        return image;
    }

    public static void main(String[]args)throws IOException
    {
        JFrame frame=new JFrame("Automaton");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(1000,1000);
        BufferedImage image;
        Processor t=new Processor();
        frame.add(t);
        image=new BufferedImage(1,1,BufferedImage.TYPE_INT_RGB);
        image=toBufferedImage(t);
        images[0]=image;
        line++;
        while(line<max_lines)
        {
            image=toBufferedImage(t);
            images[line]=image;

            line++;
        }
        //Fie I/O operation
        File f=new File("Rule 100.png");
        ImageIO.write(images[line-1],"PNG",f);
        frame.setVisible(true);
    }

}
```

**JONAKI GHOSH** is an Assistant Professor in the Dept. of Elementary Education, Lady Sri Ram College, University of Delhi where she teaches courses related to math education. She obtained her Ph.D. in Applied Mathematics from Jamia Milia Islamia University, New Delhi, and her M.Sc. from IIT Kanpur. She has taught mathematics at the Delhi Public School, R K Puram, where she set up the Math Laboratory & Technology Centre. She has started a Foundation through which she conducts professional development programmes for math teachers. Her primary area of research interest is in the use of technology in mathematics instruction. She is a member of the Indo Swedish Working Group on Mathematics Education. She regularly participates in national and international conferences. She has published articles in journals and authored books for school students. She may be contacted at jonakibghosh@gmail.com.

**ROHIT ADSULE** is a 12th grade student from The Shri Ram School Aravali, Gurugram, Haryana. His hobbies include chess, piano and football. He loves integrating his skills in mathematics with concepts related to computer science, and it was thus he came across the topic of cellular automata. He wishes to explore and understand the connection of mathematics to seemingly unrelated fields such as music theory and behavioural mechanics.