Preface

In this book we journey into the realm of cellular automata (CA). CA states are displayed in various ways, which will allow readers of limited mathematical training to appreciate the beauty of CA-evolved patterns. Most chapters have an associated Windows 95 program that generated the chapter's examples. Readers are encouraged to create their own examples, perform their own experiments, but most importantly have fun with this software.

Chapter 1 discusses why CA are of interest and to what they have been applied. Basic terminology and definitions are also introduced.

In Chapter 2 we make an excursion into the land of 0-dimensional CA. This gives a simple introduction to some fundamental concepts used in subsequent chapters, and introduces vocabulary used in the fields of both continuous and discrete dynamical systems. All of the examples were generated with the program FiniteDynamicalSystems.

In Chapter 3 we explore examples of 1-dimensional CA, and show some connections with the field of 1-dimensional digital signal processing. The examples of this chapter were all created with the program CA1dim.

Chapter 4 looks at 2-dimensional CA. The states of these CA are very easy to display on a computer screen or on paper, by translating a cell's state into a pixel color. In this chapter many visually interesting examples are shown, and connections with central topics from the field of image processing are discussed. These examples were all created with the program CA2dim, with which the reader can create virtually unlimited numbers of colorful and geometrically rich patterns. Artists have found this program a useful pattern-generating tool.

A particular class of ecosystem simulations, using 2-dimensional CA, is the subject of Chapter 5. Various experiments are performed, and the resulting population dynamics discussed. These experiments were done with the help of the program CAecosystem, which allows the user to set various ecosystem parameters and then observe the subsequent dynamics.

Chapter 6 introduces Stephen Wolfram's classification of CA into four qualitative classes, and how these classes are related to fundamental issues in the theory of computation. Here are discussed various philosophical issues that arise in the context of CA.

An extensive bibliography and related web links will allow readers to continue in their studies of CA in many different directions.

I would like to thank Bakersfield College administrators and the Kern Community College District Board of Trustees for granting me the sabbatical leave which made this work possible.
Contents

Title Page

Preface

Chapter 1  What Are Cellular Automata?

1.0  Introduction
1.1  Basic Definitions

Chapter 2  Zero-Dimensional Cellular Automata

2.0  Preliminaries
2.1  Example:  \( x^2 + 1 \pmod{17} \)
2.2  Shannon's Entropy
2.3  Example:  \( x^2 + 1 \pmod{37} \)
2.4  Example:  \( x^2 + 1 \pmod{221} \)
2.5  Example:  \( x^3 + 2x^2 + 3x + 2 \pmod{221} \)
2.6  Ulam's Function

Chapter 3  1-Dimensional Cellular Automata

3.0  Preliminaries
3.1  Transition Rules as Convolution Kernels
3.2  Example:  \( k = 5, \, n = 21, \, \text{kernel} = [1 \, 1 \, 1] \)
3.3  Example:  \( k = 2, \, n = 512, \, \text{kernel} = [1 \, 0 \, 1] \)
3.4  Examples:  \( n = 512, \, \text{kernel} = [1 \, 0 \, 1], \)
  \( k = 3, \, 4, \, 5, \, 15, \, 17, \, 29, \, 30 \)
3.5  Polynomial Representation of Additive CA
3.6  Sum of Squares:  \( n = 200 \)
3.7  A Strange Non-Additive Example,  \( k = 256, \, n = 512 \)

Chapter 4  2-Dimensional Cellular Automata

4.0  Preliminaries
4.1  2-Dimensional Convolution Kernels
4.2  Example:  \( k = 256, \, n = m = 500, \)
  \[
  \begin{array}{ccc}
  1 & 0 & 0 \\
  1 & 1 & 1 \\
  1 & 0 & 0 \\
  \end{array}
  \]
  \( \text{kernel} = \)


4.3 Example: \( k = 256, \quad n = m = 500, \)

\[
\text{kernel} = \begin{pmatrix}
-2 & 1 \\
4 & -2 \\
-2 & 1
\end{pmatrix}
\]

4.4 Polynomial Representation of 2-Dimensional Additive CA
4.5 A Famous Non-Additive CA: *The Game of Life*
4.6 A Non-Additive CA Involving Euler's Totient Function

**Chapter 5**  
*Ecosystem Modeling Using CA*

5.0 *CAecosystem*

**Chapter 6**  
*Qualitative Classification of CA*

**Appendices**

A  Modular Arithmetic  
B  Pascal's Triangle  
C  Geometric Construction of Sierpinski's Gasket  
D  The Euler Totient Function

**Bibliography**

**Related Links**

Copyright © 1997 by Rafael Espericueta. All rights reserved.
Chapter 1

What Are Cellular Automata?

1.0 Introduction

Back in 1971 an article appeared in the Mathematical Games section of Scientific American describing The Game of Life, created by Cambridge mathematician John Conway. Conway had been looking for a simple discrete dynamical system capable of self-replication. Soon after the article appeared, computers the world over began devoting unprecedented amounts of time to exploring Life’s ramifications, often to the chagrin of computing facility administrators.

Conway’s Game of Life is one of an infinite class of mathematical systems known as Cellular Automata (CA). Creating a CA is creating a simple universe, with its own space-time structure and laws of physics. One sets up the initial state (perhaps using randomization techniques), defines the laws of physics, starts time’s progression, and beholds the subsequent evolution of the universe. When playing God in this way, one is often surprised and delighted by what ensues.

CA were first introduced by applied mathematicians John von Neumann and Ulam in the 1940s, who called them cellular spaces. They had been looking for simple mathematical models of biological systems. CA have since been used as mathematical models of phenomena from diverse disciplines. Any system analyzable in terms of large numbers of discrete elements with local interactions is amenable to being modeled as a CA. With such systems, simple interaction rules between the system’s components tend to give rise to very complex emergent behavior.

CA have been shown capable of yielding discrete approximations to the solutions of systems of differential equations, in terms of which much of the macroscopic physics of our world can be expressed. CA models have been applied to fluid dynamics, plasma physics, chemical systems, growth of dendritic crystals, economics, two-directional traffic flow, image processing and pattern recognition, parallel processing, random number generation, and have even been used as a model for the evolution of spiral galaxies. Uses of CA as models in the biological sciences include the functioning of groups of cells, heart fibrillation, neural networks, and ecosystems.

My program CAecosystem uses a CA to simulate a three-species ecosystem, and captures the dynamics typical of such systems. Mathematicians have traditionally used generalizations of the Volterra-Lotka system of differential equations to model population dynamics of these types of predator/prey systems. CA provide a more intuitive model of ecosystems than do differential equations. Furthermore CA can actually model the ecosystem itself, not just the population numbers of the various species.
CA provide a useful mathematical model of massively parallel multi-processor systems. Each cell can be considered a processor, with the cell states corresponding to the finite possible states of the processor. The processors in the neighborhood of a given processor, P, are the processors directly connected to P. The above could also be describing a neural net, with ‘neuron’ in place of ‘processor’. How to get such a system to perform useful computational tasks, making optimal use of all that parallel computing power, is a central problem in computer science. CA experiments have provided much needed insight into how simple local interactive dynamics can give rise to complex emergent global behavior.

Similar problems involving simple local rules yielding complex emergent global behavior arise and are central problems in many fields. Some examples of important unanswered questions involving how emergent global dynamics arise are:

- How did life emerge from the interactions of various chemicals in the pre-biotic conditions on primordial earth?
- How does morphogenesis proceed in a developing embryo; how do the relatively simple chemically modulated interactions of undifferentiated cells in an embryo result in such a globally orchestrated complex of specialized organs and tissues?
- How does consciousness emerge from the relatively simple interactions of hundreds of millions of neurons?
- How does cultural evolution unfold from the interaction of its underlying milieu of concepts and beliefs?
- How do relatively simple (economic) behavior patterns on the parts of individuals give rise to overall economic system dynamics?

CA computer experiments and simulations provide insight into problems such as these. CA also provide a conceptual framework in which theoretical progress may be made on the phenomena of emergence in general. The Santa Fe Institute was formed to address just these types of problems, from a variety of fields. It isn’t surprising that CA continue to be a major focus of the Institute’s research activities.

As single processor computer systems yield to massively parallel systems, the simulation of CA will become easier, and increasingly practical applications of CA will abound. The inherently parallel structure of CA make them particularly suited to implementation on multi-processor systems. The MIT Physics of Computation research group has developed a CA-machine (CAM) specially designed to run CA programs. This CAM is being successfully applied to problems from various areas of physics, as well as to medical imaging applications.
1.1 Basic Definitions

Cellular Automata (CA) are a class of discrete dynamical systems, consisting of an array of nodes (cells) of some dimension, n. Each cell can be in one of \( k \) different states at a given tick of the clock. At each discrete tick of the clock, each cell may change its state, in a way determined by the transition rules of the particular CA. The transition rules describe precisely how a given cell should change states, depending on its current state and the states of its neighbors. Which cells are in the neighborhood of a given cell must be specified explicitly.

More precisely, a CA consists of…

1. A lattice of cells each of which can be in one of a finite number of distinct states at each discrete point in time. This lattice may be of any dimension, and is of infinite extent. One-dimensional CA can be visualized as having a cell at each integral point on the real number line. Two-dimensional CA have cells at all points in the plane that have only integral coordinates (the Gaussian Integers). Higher dimensional CA similarly have cells at each point in \( n \)-dimensional Euclidean space with all integral coordinates. In practice this infinite spatial extent is made effectively finite in various ways (for computer implementation), usually via a “wrap around” of the lattice. For example, in one dimension (\( n = 1 \)) the straight line becomes a circle, in two dimensions (\( n = 2 \)) the plane becomes a torus. This is equivalent to imposing a periodic condition on the original infinite space of cells.

2. A neighborhood defined for each cell. For homogeneous CA the same relative neighborhood is defined for all cells.

3. Transition rules, which specify how the dynamics unfold for the CA. Each cell transforms from its current state to a new state (at the next tick of the clock) based upon its current state and the states of its neighbors, according to the transition rules. All cells transform synchronously at discrete points in time.

Let \( n \) be the dimension of the lattice, \( k \) the number of states, \( T \) the transition rule function, \( C_t(i_1, \ldots, i_n) \) the state of the cell at position \( (i_1, \ldots, i_n) \) at time \( t \), \( N_t(i_1, \ldots, i_n) \) the values (given in a specific order) of the neighboring cells to this location at time \( t \). Then the dynamics of the CA are completely specified by the initial states of all the cells, \( C_0 \), along with the recursion rule

\[
C_{t+1}(i_1, \ldots, i_n) = T(N_t(i_1, \ldots, i_n)).
\]

As is usual with formal mathematical definitions, this is best understood by examining some simple examples. The remainder of this chapter looks at examples in which the underlying lattice of cells has dimension zero, so there is only one cell! We shall see that even this extremely simple case holds some surprises.
Chapter 2

0-Dimensional Cellular Automata

2.0 Preliminaries

A two-dimensional lattice can be thought of as the points in the plane with integer coefficients, and a one-dimensional lattice can be thought of as the integer points on the real number line. A zero-dimensional lattice then must consist of a single point, not really much of a lattice at all. Thus the parallelism inherent in higher dimensional CA is totally lacking in a 0-dimensional CA, so one might well think that this zero-dimensional case is a trivial one, totally lacking in interest. However, no one would consider the concept of 'function' trivial, and functions are 0-dimensional CA! More precisely, the discrete functions of one variable, when thought of as dynamical systems are 0-dimensional CA.

We will now take a little detour through the land of finite dynamical systems. Through this we shall illustrate important concepts these relatively simple systems share with far more complex dynamical systems, from higher-dimensional CA to systems of differential equations. We start by seeing how finite dynamical systems fit into the CA framework. As with other CA, to be well defined we need to define our lattice, allowable cell states, neighborhoods, transition rules, and an initial state. The lattice consists of simply one cell, and the allowable cell states we will take to be the integers modulo some fixed integer n. The neighborhood of our single cell must be the cell itself, since there are no others in this lonely universe. Our transition rule will be a function that has integers for its inputs and outputs (an arithmetic function); we then take the function's output modulo n. Just as with higher dimensional CA, to get things going we need an initial state for our one and only cell, in this case a single integer \( X_0 \). Notice that there are only n possible initial states for our cell: 0, 1, 2, . . . , n-1. Letting the dynamics unfold, we obtain a sequence of numbers by iterating the recursion rule

\[
X_{k+1} = f(X_k) \pmod{n}.
\]

Thus we obtain the sequence of values

\[
X_0, X_1, X_2, \ldots, X_M, \ldots, X_N, \ldots.
\]

This sequence is termed a trajectory, the same term used to describe the path traversed by an initial point under the influence of a differential equation. Here instead of a continuous path the trajectory is a discrete sequence, and instead of a differential equation (describing an infinitesimal recursion rule), we have a recursive arithmetic function. The language used to describe the continuous situation and the discrete one are almost identical, reflecting the analogous relationship of these two realms.
At some point in the above sequence, say $X_M$, we must encounter a number for the second time, since each $X_i$ can only have one of the $n$ allowable values. When this occurs, we are destined to repeat whatever sequence followed that first occurrence of $X_M$. We will then have entered a cycle. The part of a sequence preceding its entry into a cycle is called a transient, and if a cycle has transients it is then called an attractive cycle. In differential equations the analogous concept is that of limit-cycles.

The concept of trajectories gives us a local way of understanding the dynamics, both of differential equations and of recursive functions. To obtain a more global, qualitative way to understand the dynamics of differential equations, Poincare' introduced the "state-space" (also known as "phase space"). The analogous concept in the discrete realm is the basin of attraction field. These are easy to construct in the 0-dimensional case and illustrate properties of basin of attraction fields in general.
2.1 Example: $x^2 + 1 \pmod{17}$

As our first particular example, we let $n = 17$ and define our recursion function by

$$f(x) = x^2 + 1 \pmod{17}$$

Starting with $x = 0$ we get the trajectory $0, 1, 2, 5, 9, 14, 10, 16, 2, \ldots$ In this sequence notice that 2 has appeared a second time, and so begins the attractive cycle $2, 5, 9, 14, 10, 16$. My program FiniteDynamicalSystems computed the following basin of attraction field for this recursive function (Figure 1).

In Figure 1 notice the attractive cycle (traversed in a clockwise direction), and the incoming transient branches (traversed in an inward direction, towards the attractive cycle). For this function there was only one attractive cycle; ordinarily there are more than one, and the basin of attraction field consists of the entire collection of cycles with their associated transients. Each node has exactly one output, implying that the system is deterministic. Some nodes possess more than one input, implying that the system is irreversible and that the entropy of the system tends to decrease as the system evolves. CA of all dimensions share these qualities. The tendency of entropy to decrease in CA is no doubt partly responsible for the emergent behavior often evident in such systems.

Notice in Figure 1 that the sum of the two predecessors of a given node always adds up to 17. This is a consequence of the algebraic property that

$$x^2 = (-x)^2$$

along with the modular arithmetic property,
It follows that
\[ x^2 = (n - x)^2 \pmod{n}, \]
hence,
\[ x^2 + 1 = (n - x)^2 + 1 \pmod{n}, \]
and so
\[ f(x) = f(n - x) \pmod{n}, \quad \text{for} \quad 0 \leq x \leq n - 1. \]

And indeed, \( x + (n - x) = n \). In this example \( n = 17 \), but the same result holds in the examples to follow that use this same function. Verify that this is true for the examples of sections 2.3 and 2.4.

The outermost states on the basin of attraction field of Figure 1, with values 4, 6, 7, 8, 11, 12, 13, and 15, all have no predecessor states. After one generation our CA is guaranteed not to be in one of these states. Outlying states such as these are known as "Garden of Eden" states.

Before going on to other examples, we digress to define the important concept of entropy. The concept of entropy is not separate from that of information, in precisely the same way that the concept of a half-empty glass is not separable from the concept of a half-full one. Out of this concept grows the entire field of information theory.
2.2 Shannon's Entropy

Entropy for 0-dimensional CA with \( n \) states may be defined via Shannon's Entropy,

\[
H_t = - \sum_{k=0}^{n-1} p_t(k) \log_2 p_t(k)
\]

where \( p(k) \) is the probability of the system having value \( k \) after \( t \) time steps from an initial randomly selected value. In this notation, we agree that "0 log 0" equals 0. Using log base 2 results in entropy measured in units of bits (base 2 digits). \( H \) is a measure of the system's basic information potential.

To understand this entropy formula, we will examine two extreme cases. First, consider a CA where all initial states evolve to 0 (e.g., take \( f(x) = 0 \) for all \( x \)). Choosing \( t \) large enough that all initial states must have evolved to 0, we have \( p(0) = 1 \), and \( p(k) = 0 \) for all other choices of \( k \). Thus the above summation reduces to the single term, \( \log(1) \) which equals 0, so for this CA we get zero entropy measure. The term entropy is associated with randomness and uncertainty, so it makes sense that this system evolves to a state of zero entropy, since a priori we were totally certain about the outcome of this system's evolution.

For our second example, consider a CA in which all states are equally likely for all time (e.g., take \( f(x) = x + 1 \)). Then \( p(k) = 1/n \), for all \( n \), so the above formula reduces to

\[
H = - \log_2 \frac{1}{n} = \log_2 n,
\]

the maximum entropy possible for a 0-dimensional CA with \( n \) states (see Table 1). Notice that \( H \), rounded up, is the number of digits necessary to represent \( n \) as a binary number.

<table>
<thead>
<tr>
<th>( n )</th>
<th>1</th>
<th>2</th>
<th>5</th>
<th>10</th>
<th>15</th>
<th>20</th>
<th>30</th>
<th>50</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>( S )</td>
<td>0</td>
<td>1.0</td>
<td>2.32</td>
<td>3.32</td>
<td>3.91</td>
<td>4.43</td>
<td>4.91</td>
<td>5.64</td>
<td>6.64</td>
</tr>
</tbody>
</table>

Table 1: Maximum entropy, \( S \), as a function of the number of cell states, \( n \).

My program \textit{FiniteDynamicalSystems} computes and graphs \( H_t \) from \( t = 0 \) to a \( t \) value entered by the user. To compute \( p_t(k) \) the program counts the number of values which in \( t \) time steps (i.e., in \( t \) iterations) end up with value \( k \); call this number \( \text{Num}(k,t) \). Since there are \( n \) possible initial states, the probability of randomly picking one that leads to \( k \) after \( t \) iterations is precisely
\[ p_i(k) = \frac{\text{Num}(k, t)}{n}. \]

\( H_0 \) is thus always the maximal entropy value possible for the system. When enough time steps have elapsed to assure all values have settled into a cycle, the entropy becomes thenceforth constantly equal to the minimum entropy of the system.

In general the less the entropy, the greater one's a priori information about the state of the system; entropy is a measure of the degree of one's uncertainty about the state of a system.
2.3 Example: $x^2 + 1 \pmod{37}$

Figure 2 depicts the basin of attraction field for the function

$$f(x) = x^2 + 1 \pmod{37}$$

In this case there are three separate attractive cycles, two of length 1 ( '27' in (A) and '11' in (B) ), and one of length 2 ( 8, 28 in (B) ). Attractive 1-cycles are usually just called attractors or fixed points.

Since eventually this system evolves to one of four values, the entropy should be relatively small after six iterations. In Figure 3 is graphed the entropy, $H$, of the system as a function of the number of iterations.

Figure 2: Basin of attraction field for $f(x) = x^2 + 1 \pmod{37}$.

(A) Attractor 27 with its basin of attraction containing 7 values.
(B) 2-cycle with it's 4 element basin of attraction.
(C) Attractor 11 with its 22 element basin of attraction.
This system ends up in one of its attractors (27 or 11) or its 2-cycle (8, 28) after 6 time steps, and thus we know that the system will have one of these 4 values after that time. A uniformly random system with 4 states has an entropy equal to

\[ \log_2 4 = 2, \]

rather larger than this system's minimum entropy, 1.478. Our system's smaller final entropy is due mostly to the greater number of transient values associated with attractor 11. This attractor has 23 values in its basin of attraction, so the probability of this system ending up with value equal to 11 is \( \frac{23}{37} = 0.621\ldots \), and most of the time the system ends up with value 11. This additional information about the system results in less uncertainty about the state of the system than were it a uniformly random one, where the probability of ending up with value 11 would be only \( \frac{1}{4} = .25 \); thus our system has less entropy than does a uniformly random one. The greater one's a priori information about a system, the less one's uncertainty about the system's states, and the lower the system's entropy.
2.4 Example: $x^2 + 1 \pmod{221}$

Figures 4 through 7 together depict the basin of attraction field for the function

$$f(x) = x^2 + 1 \pmod{221}$$

In this case there are four separate attractive cycles, two of length 1 (‘27’ in (A) and ‘11’ in (B), and one of length 2 (8, 28 in (B)).

**Figure 4:** One of the basins of attraction, that of an attractive 12-cycle, of $f(x) = x^2 + 1 \pmod{221}$

**Figure 5:** One of the basins of attraction, that of an attractive 6-cycle, of $f(x) = x^2 + 1 \pmod{221}$
Figure 6: One of the basins of attraction, that of an attractive 6-cycle, of $f(x) = x^2 + 1 \pmod{221}$. This basin has the same topology as the basin depicted in Figure 4.

Figure 7: One of the basins of attraction, that of an attractive 12-cycle, of $f(x) = x^2 + 1 \pmod{221}$.

The above-observed topological repetition among basins of attraction is very common for large moduli, and this redundant tendency becomes even more pronounced for CA with higher dimensional lattices. In Figure 8 we see the entropy graph for the above system.
The result of many iterations of this system is far less certain than the system we examined in Example 0.1.3. This system thus has a higher minimum entropy ($H_{\text{min}} = 4.983$) than the system of that previous example ($H_{\text{min}} = 1.478$).

![Figure 8: Entropy as a function of time for $f(x) = x^2 + 1 \pmod{221}$](image)

This system ends up in one of its 4 cycles after 3 time steps, and thus we know the system will have a value from one of these cycles after that time. Thus the system will then be limited to the 36 values in its cycles. A uniformly random system with 36 states has an entropy equal to

$$\log_2 36 \equiv 5.170$$

which is very close to this system's minimum entropy. Our system's smaller final entropy is due to the greater number of transient values associated with the 12-cycles. This results in a greater probability that our system will end up in one of the 12-cycles than in one of the 6-cycles. This additional information about the system results in less uncertainty about the state of the system than were it a uniformly random one, thus our system has slightly less entropy than does a uniformly random one.
2.5 Example: $x^3 + 2x^2 + 3x + 2 \pmod{28}$

Figure 9 shows the basin of attraction field for the function

$$f(x) = x^3 + 2x^2 + 3x + 2 \pmod{28}$$

In this case there are two separate attractive cycles making up the basin of attraction field.

Figure 9: Basin of attraction field consisting of 2 basins of attraction.
(A) Attractive 6-cycle with associated transients.
(B) Attractive 2-cycle with associated transients.

In Figure 10 we see the entropy graph for the above system

Figure 10: Entropy as a function of time for $f(x) = x^3 + 2x^2 + 3x + 2 \pmod{28}$.
2.6 Ulam's Function

In the late '40s Stanislaw Ulam introduced the \( 3N + 1 \) problem to the Los Alamos scientific community (though he didn't invent it), which was beginning to play around with the very first digital computers. The problem involves a simple recursively defined arithmetic function that to this day holds unfathomed mysteries. The function in question is defined by

\[
f(n) = \begin{cases} 
\frac{n}{2}, & \text{if } n \text{ even;} \\
2n + 1, & \text{if } n \text{ odd.}
\end{cases}
\]

The problem that Ulam introduced to the folks at Los Alamos had to do with the observation that no matter what number one starts with, iterating the function results in arriving at the attractive cycle \( 4 - 2 - 1 \). Nobody knows if all positive integers belong to this cycle's basin of attraction, but it has been verified for all numbers less than \( 10^{12} \).

If one allows zero as an argument for this function, zero becomes another attractor. The program \textit{FiniteDynamicalSystems} has this function built in, modulo various values. When one computes this function using modular arithmetic, zero has its own basin of attraction. Figure 11 shows the results of one such computation, modulo 20, and Figure 12 shows the result of choosing 40 for the modulus. In Figure 13 the modulus is 47. Notice the surprising occurrence of yet another attractor, 23, in Figure 13.

\[\text{Figure 11: Ulam's function modulo 20. (A) and (B) are the Basin of attraction field, and (C) shows the entropy as a function of number of iterations.}\]
Figure 12: Ulam’s function modulo 40. (A) and (B) are the Basin of attraction field, and (C) shows the entropy as a function of number of iterations.

Figure 13: Ulam’s function modulo 47. (A), (B), and (C) are the Basin of attraction field, and (D) shows the entropy as a function of number of iterations.
Chapter 3
1-Dimensional Cellular Automata

3.0 Preliminaries

As a paradigm for parallel processing, 0-dimensional dimensional CA aren't really CA at all - parallel processing is meaningless when you only have one processor. So we need to have at least one dimension to have a real CA; this is where most treatises on CA begin, so now we're off to confront the real beast.

A one-dimensional lattice is an array of cells one may imagine distributed along a line. Each cell takes on a value from a finite set at each tick of the clock. We denote the state of the CA at time $t$ by

$$C^{(t)} = \cdots C^{(t)}_{-2} C^{(t)}_{-1} C^{(t)}_{0} C^{(t)}_{1} C^{(t)}_{2} \cdots .$$

We will only consider CA possessing periodic boundary conditions, a technique for making infinite arrays effectively finite. To make the above infinite array finite implies we have some specific number of cells in mind, and we call this number $n$. Then the above infinite array becomes a finite array of $n$ cells,

$$C^{(t)} = C^{(t)}_{0} C^{(t)}_{1} C^{(t)}_{2} \cdots C^{(t)}_{n-2} C^{(t)}_{n-1}.$$

Mentally connect the two ends of this queue to form a circular one, so that $C^{(t)}_{0}$ and $C^{(t)}_{n-1}$ are considered to be adjacent. For $n = 5$ this would appear as in Figure 1.

![Figure 1: periodic boundary conditions](image)

Equivalently, one could define initial conditions for only cell number 0 through cell number $n-1$, and then repeat this pattern of $n$ values over and over endlessly to the infinite numbers of cells to the right and to the left of our initial $n$ cells. This results in any cell having an index congruent to 0 modulo $n$ actually having the same value as cell 0, any cell having an index congruent to 1 modulo $n$ having the same value as cell 1, etc.

Thus whenever $b$ and $d$ are congruent modulo $n$, we have

$$C^{(t)}_{b} = C^{(t)}_{d} .$$
In short, all arithmetic on cell indices is performed modulo $n$, where $n$ is the total number of cells in our cellular automaton.

Although CA neighborhoods in general may be defined arbitrarily, for the CA we will examine the same relative neighborhood is defined for each cell. For example if the neighborhood of one cell consists of the two immediately adjacent cells, then this is also true of all other cells. Such CA are called *homogeneous*.

For the examples that follow, we let $k$ denote the number of states possible for each cell, and further identify the states with the integers $0, 1, 2, \ldots, k-1$. Computations on cell values are performed modulo $k$.

Before moving on to examples we first discuss a useful way to represent some classes CA transition rules borrowed from the field of 1-dimensional digital signal processing: *convolution kernels*. 
3.1 Transition Rules as Convolution Kernels

In digital signal processing one of the most basic tools, borrowed from differential equations, is convolution. Students in an introductory course on differential equations are first introduced to a continuous version of this concept when they study the Laplace transform. For discrete signals, such as digital ones, the convolution operation is equivalent to applying a CA to the digital signal. One simply takes the original signal as the initial condition of the CA, and defines the CA transition rules by the convolution kernel. The actual computation as implemented on a computer performing a convolution operation is identical to applying the corresponding CA to the signal. The difference is that a convolution is usually only applied once or at most a small number of times to a signal, whereas a CA usually iterates its recursive procedure for many generations. In digital signal processing, this process is used to produce a filtered output signal from an input signal, and that's all. With CA this process is iterated repeatedly, even endlessly.

Not all CA can be represented by convolution kernels; those CA that can be expressed in terms of a convolution kernel are called additive. CA examples 1, 2, 3, 4, 6 are all additive, while 5 (Life) and 7 are not additive.

To see how a 1-dimensional convolution kernel is applied to a signal (or a CA), we start with a specific example. We apply the convolution kernel \([1 2 1]\) to the signal

\[
\begin{array}{cccccccc}
0 & 0 & 3 & 4 & 5 & 0 & 0 \\
\end{array}
\]

First position the kernel over the beginning of the signal:

\[
\begin{array}{cccccccc}
[1 & 2 & 1] \\
0 & 0 & 3 & 4 & 5 & 0 & 0
\end{array}
\]

and compute \(1*0 + 2*0 + 1*3 = 3\)

the first value of the output “filtered” signal. Next we shift the kernel to the right by one position and repeat this process:

\[
\begin{array}{cccccccc}
[1 & 2 & 1] \\
0 & 0 & 3 & 4 & 5 & 0 & 0
\end{array}
\]

obtaining \(1*0 + 2*3 + 1*4 = 10\)

the second value of the filtered signal. Keep repeating this process, until we get to the end of the original signal…

\[
\begin{array}{cccccccc}
[1 & 2 & 1] \\
0 & 0 & 3 & 4 & 5 & 0 & 0
\end{array} \Rightarrow 1*3 + 2*4 + 1*5 = 16
\]

\[
\begin{array}{cccccccc}
[1 & 2 & 1] \\
0 & 0 & 3 & 4 & 5 & 0 & 0
\end{array} \Rightarrow 1*4 + 2*5 + 1*0 = 14
\]

\[
\begin{array}{cccccccc}
[1 & 2 & 1] \\
0 & 0 & 3 & 4 & 5 & 0 & 0
\end{array} \Rightarrow 1*5 + 2*0 + 1*0 = 5
\]
So the output filtered signal is $3 \ 10 \ 16 \ 14 \ 5$. The output signal is two numbers shorter than the input signal. Often one wants the output to be the same length as the input, and this can be accomplished in various ways. One way is to “pad” the input signal with zeros on each end; for example our example input signal might have actually been $0 \ 3 \ 4 \ 5 \ 0$. Adding an extra 0 on each end and then performing the above convolution operation would yield an output signal of the same length as the input. Another way to assure that the output has the same length as the input is to use wrap-around, where the value to the left of the leftmost value is considered to be the rightmost value, and conversely the value to the right of the rightmost value is considered to be the leftmost value. This is analogous to joining the two ends of a piece of string to form a circular loop. Sometimes this technique is described as imposing periodic boundary conditions on the original signal.

In mathematical notation, convolution of a 1-dimensional signal, $x_0 \ x_2 \ldots x_{n-1}$, with kernel $[k_{-m}, \ldots, k_{-1}, k_0, k_1, \ldots, k_m]$ of length $2m+1$, assuming periodic boundary conditions, may be expressed as

$$y_h = \sum_{l=-m}^{m} k_l \cdot x_{(h+l) \mod n}$$

where $y_0 \ y_1 \ldots y_{n-1}$ is the resulting output signal. We will be taking this sum modulo $k$.

Here we have followed the digital signal processing terminology, referring to the input and output arrays as signals. The same algorithmic process can just as easily be applied to arrays considered as CA states. The input state is one generation of the CA, and the output state is the next generation.

With 1-dimensional CA, we can see at a glance the whole evolutionary process, by displaying successive generations as successive rows of pixels on a computer display or a piece of paper. For higher dimensional CA we must resort to other techniques.
3.2 Example: $k = 5$, $n = 21$, kernel $= [1 1 1]$.  

For this example $k = 5$, and the total number of cells, $n$, is 21. Thus each generation of our cellular automata will be a 21 component vector, which we write as

$$C_0 C_1 C_2 \cdots C_{20},$$

where each $C_j$ takes values from the numbers 0, 1, 2, 3, 4. Thus each state of this CA can be represented by a 21-digit base 5 number. The cell “to the left” of $C_0$ is considered to be $C_{20}$, and the cell “to the right” of $C_{20}$ is $C_0$. This defines the lattice of the CA, so we now move on to the definition of its neighborhoods. For this CA, the neighborhood of a cell is the cell itself along with the cell to its left and the cell to its right. The transition rule of our CA is to take the sum of all neighboring cells, modulo 5.

For 1-dimensional CA, the notation $C_t(i_1)$ is instead written $C_j^{(t)}$.

$$C_j^{(t+1)} = T(N_t(j)) = T(C_{j-1}^{(t)}, C_j^{(t)}, C_{j+1}^{(t)}) = C_{j-1}^{(t)} + C_j^{(t)} + C_{j+1}^{(t)} \pmod{5}$$

Note that our wrap-around boundary condition amounts to modulo 21 arithmetic on the (right) subscripts in the above expression. Now to get our dynamics going all we need is an initial state for each cell... We start with a very simple initial state, with all cells zero except for one cell with value one.

**Initial state:**

```
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
```

This initial state (generation 0) along with our transition rule generates successive generations, where each cell in the next generation is simply the sum of its neighbor cell values in the previous generation. In other words, the value of a cell is the sum of the three (closest) cells above it.

**Generation 0:**

```
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
```

**Generation 1:**

```
0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0
```

**Generation 2:**

```
0 0 0 0 0 0 0 0 0 1 2 3 2 1 0 0 0 0 0 0
```

**Generation 3:**

```
0 0 0 0 0 0 0 0 0 1 3 1 2 1 3 1 0 0 0 0 0
```

**Generation 4:**

```
0 0 0 0 0 0 0 0 0 1 4 0 1 4 1 0 4 1 0 0 0 0
```

**Generation 5:**

```
0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0
```

**Generation 6:**

```
0 0 0 0 1 1 1 0 0 1 1 1 0 0 1 1 1 0 0 0 0
```

**Generation 7:**

```
0 0 0 1 2 3 2 1 1 2 3 2 1 2 3 2 1 0 0 0
```

**Generation 8:**

```
0 0 1 3 1 2 1 4 4 1 2 1 4 4 1 2 1 3 1 0 0
```

**Generation 9:**

```
0 1 4 0 1 4 2 4 2 4 2 4 2 4 2 4 2 4 1 0 4 1 0
```

A computer run of 136 generations generated Figure 2, where the first generation is the leftmost column. Each column of this image, black = 0, blue = 1, green = 3, orange = 3, and red = 4. Notice that the pattern starts to repeat after 125 generations. This type of periodic behavior is inevitable for all CA with periodic boundary conditions.
The transition rules for this example CA are identical to the convolution with kernel

$$[1 \ 1 \ 1].$$

The inner product of this kernel with consecutive cells yields the new value of the center cell (just another way of expressing the above transition rule).

With 5 possible states for each cell and 21 cells, the number of possible states for this CA is equal to

$$5^{21} = 476,837,158,203,125.$$  

This makes it difficult to display its basin of attraction field. Computer programs have been written to compute basin of attraction fields for a large number of 1-dimensional CA with two possible states, and an atlas of these has been published [1]. These published basin of attraction fields are highly reminiscent of those of 0-dimensional CA, though more complex. Most CA have decreasing Shannon entropy as time proceeds, as with the 0-dimensional case. The fact that CA are deterministic precludes the possibility that entropy might increase, since each node in their basin of attraction fields is connected to one and only one next-node.

The periodicity observed in Figure 2 is a cycle of length 124 beginning with

$$\ldots 0 0 1 1 0 0 \ldots .$$

Our initial state $$\ldots 0 0 1 0 0 \ldots$$ is part of a transient in the basin of attraction of this observed attractive cycle.
3.3 Example: \( k = 2, \ n = 512, \ \text{kernel} = [1 \ 0 \ 1]. \)

For this example \( k = 2 \), and the total number of cells, \( n \), is 512. Thus each generation of our cellular automata will be a 512 component vector, which we denote

\[
C_0 \ C_1 \ C_2 \ C_3 \ C_4 \ C_5 \cdots \ C_{510} \ C_{511}.
\]

Each \( C_j \) has value 0 or 1. Thus each state of this CA can be thought of as a 512-digit binary number. The cell “to the left” of \( C_0 \) is considered to be \( C_{512} \), and the cell “to the right” of \( C_{512} \) is \( C_0 \). This defines the lattice of the CA. Now we define its neighborhoods. For this CA, the neighborhood of a cell is the cell to its left and the cell to its right. Notice that in this case the cell itself is not included in its own neighborhood. The transition rule of our CA is to take the sum the neighboring cells, modulo 2. This rule is equivalent to convolution with the convolution kernel \([1 \ 0 \ 1]\).

\[
C^{(t+1)}_j = T(N_j(j)) = T(C^{(t)}_{j-1}, C^{(t)}_{j+1}) = C^{(t)}_{j-1} + C^{(t)}_{j+1} \pmod{2}
\]

Note that the wrap-around boundary condition amounts to modulo 512 arithmetic on the (right) subscripts in the above expression. Now to get our dynamics going all we need is an initial state for each cell… We start with a very simple initial state, with all cells zero except for one cell with value one.

Initial state : \( \ldots 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ldots \)

This initial state (generation 0) along with our transition rule generates successive generations, where each cell in the next generation is simply the sum of its neighbor cell values in the previous generation. In other words, the value of a cell is the sum of the two cells diagonally to the right and left above it, modulo 2.

Generation 0: \( \ldots 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ldots \)
Generation 1: \( \ldots 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ldots \)
Generation 2: \( \ldots 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ldots \)
Generation 3: \( \ldots 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ldots \)
Generation 4: \( \ldots 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ldots \)
Generation 5: \( \ldots 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ldots \)
Generation 6: \( \ldots 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ldots \)
Generation 7: \( \ldots 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ldots \)
Generation 8: \( \ldots 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ldots \)
Generation 9: \( \ldots 0 \ 1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ldots \)
Generation 10: \( \ldots 0 \ 1 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ldots \)
Generation 11: \( \ldots 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ldots \)
A computer run of 256 generations generated the pattern seen in Figure 3. In this image, black represents a value of 1 and white represents 0. This pattern of ones and zeros forms successive approximations to a fractal known as *Sierpinski’s gasket* (see Appendix 2 for a discussion of Sierpinski’s geometric construction of this fractal).

![Figure 3: Sierpinski triangle generated by the CA with transition rules represented by convolution kernel [1 0 1].](image)

The number of possible states for this CA is $2^{512}$, approximately equal to $10^{154}$. This number is considerably greater than the number of fundamental particles in the universe. We will forego printing out the complete basin of attraction field for this CA!

For our next examples, we use the same transition rule, but allow more possible states. Fractal triangle patterns similar to Sierpinski’s gasket result.
### 3.4 Examples: $n = 512$, kernel $= [1 \ 0 \ 1]$, $k = 3, 4, 5, 15, 17, 29, 30$

Interesting patterns are obtained using the same CA as in Example 3.3 but with more states, by taking larger moduli. Next come a series of figures showing the patterns that emerge by allowing 3, 4, 5, 15, 17, 29, and 30 states. Notice that for prime moduli the patterns are simpler and more regular, and that the complexity increases with the number of factors of the modulus.

For each of the following images, the initial state is

$$\ldots 0 0 0 0 1 0 0 0 0 \ldots$$

and the transition rules are defined by the convolution kernel

$$\begin{bmatrix} 1 & 0 & 1 \end{bmatrix}.$$

The only difference is the number of allowable states for each cell, $k$. The computation of value of cell $j$ in generation $t + 1$ is computed via the recursion formula

$$C_j^{(t+1)} = T(N_r(j)) = T(C_{j-1}^{(t)}, C_{j+1}^{(t)}) = C_{j-1}^{(t)} + C_{j+1}^{(t)} \pmod{k}.$$

White is used to represent the cell value 0, and colors are used to represent the other $k-1$ cell states.

![Figure 4: $k = 3$](image)
Figure 5: $k = 4$

Figure 6: $k = 5$
Figure 7: $k = 15$

Figure 8: $k = 17$
Figure 9: $k = 29$

Figure 10: $k = 30$
Figure 5a: \( k = 4 \) (detail of Figure 5, magnified 3x)

Figure 6a: \( k = 5 \) (detail of Figure 6, magnified 3x)
**Figure 7a:** $k = 15$ (detail of Figure 7, magnified 4.7x)

**Figure 8a:** $k = 17$ (detail of Figure 8, magnified 4.2x)
Figure 9a:  $k = 29$ (detail of Figure 9, magnified 4.1x)

Figure 10a:  $k = 30$ (detail of Figure 10, magnified 2.9x)
3.5 Polynomial Representation of Additive CA

Additive CA (those CA with transition rules representable in terms of convolution kernels) also have an interesting polynomial representation that facilitates an algebraic analysis of the CA dynamics. For all of the following we assume the usual wrap-around periodic boundary condition.

For 1-dimensional CA with \( n \) cells and \( k \) possible cell values, the state of the CA at time \( t \) can be represented as an \( n \)-digit integer, base \( k \):

\[
A^{(t)} = a_0^{(t)} a_1^{(t)} a_2^{(t)} ... a_{n-1}^{(t)}
\]

The characteristic polynomial of the CA is given by

\[
A^{(t)}(x) = a_0^{(t)} + a_1^{(t)} x + a_2^{(t)} x^2 + ... + a_{n-1}^{(t)} x^{n-1}.
\]

The convolution kernel that represents the transition rules also can be represented as a polynomial, \( T(x) \), and we will see that

\[
A^{(t+1)}(x) = T(x) A^{(t)}(x).
\]

From this we conclude (in slightly simplified notation) that

\[
A^{(t)} = T \cdot A^{(0)}.
\]

In all of the above algebraic calculations the coefficient arithmetic is all done modulo \( k \), and the exponent arithmetic is done modulo \( n \). In abstract algebra, this polynomial ring is denoted by

\[
\mathbb{Z}_k[x]/\langle x^n - 1 \rangle
\]

"the ring of polynomials in variable \( x \) with modulo \( k \) integer coefficients, modulo the ideal generated by the polynomial \( x^n - 1 \)."

This means in particular that

\[
x^n \equiv 1, \quad \text{and so} \quad x^{n-1} = x^n x^{-1} \equiv 1 \cdot x \equiv x, \quad \text{etc.}
\]

Furthermore in this polynomial ring we have that
\[ x^{n-1}x = x^n = 1, \quad \text{and so} \quad x^{-1} = x^{n-1}, \quad x^{-2} = x^{n-2}, \quad \text{etc.}, \]

so that negative powers of \( x \) are meaningful for this version of polynomial algebra.

We now illustrate the process of finding the polynomial representative of the transition rules, by means of a simple example. Starting with the convolution kernel \([1 \ 2 \ 3]\) the characteristic polynomial is then calculated as

\[
T(x) = x^{-1} + 2 + 3x^1 = x^{n-1} + 2 + 3x = 2 + 3x + x^{n-1}.
\]

For another example, consider the convolution kernel \([-1 \ 2 \ -3 \ 4 \ -5]\). Converting this to its polynomial representation we get

\[
T(x) = -x^{-2} + 2x^{-1} - 3 + 4x^1 - 5x^2 = -x^{n-2} + 2x^{n-1} - 3 + 4x - 5x^2
\]

\[= -3 + 4x - 5x^2 + x^{n-2} - 2x^{n-1}.\]

More generally, from the convolution kernel

\[
[ k_{-r} k_{-r+1} \cdots k_{-1} k_0 k_1 \cdots k_{r-1} k_r ]
\]

we construct the characteristic polynomial

\[
T(x) = \sum_{i=-r}^{r} k_i x^i = k_0 + \sum_{i=1}^{r} d x^i + k_{-i} x^{n-i} 1 i.
\]

Starting with an initial state

\[
A(x) = \sum_{j=0}^{n-1} a_i x^j,
\]

the next state is explicitly given by

\[
T(x)A(x) = \sum_{i=0}^{n-1} \left[ \sum_{j=0}^{r} k_j a_{i+j \text{mod } n} \right] (\text{mod } k) x^i.
\]

A property that greatly simplifies the analysis of additive CA is that of superposition. One can add two CA states by adding their characteristic polynomials:

\[
A^{0}(x) + B^{0}(x) = (A + B)^{(0)}(x)
\]
The superposition principle states that

\[ T^{(t)}(x)(A^{(0)}(x) + B^{(0)}(x)) = T^{(t)}(x)A^{(0)}(x) + T^{(t)}(x)B^{(0)}(x) \]

Where \( T(x) \) is the polynomial representing the transition rules of the CA. Clearly this is true for such CA; in the above form it follows immediately from the distributive property for these polynomials. The superposition principle means that for additive CA we only need to examine the time evolution of a single cell, since any initial state is the sum of such single cells. This greatly simplifies the analysis.

For an example consider the transition rule defined by the kernel \([1 \ 0 \ 1]\) (see also Example 3.2). This is representable as the polynomial

\[ T(x) = x^{-1} + x = x^{n-1} + x = x(1 + x^{n-2}) \cdot \]

We start with an initial state consisting of one non-zero cell, so

\[ A^{(0)}(x) = a_c x^c, \quad \text{for some} \quad 0 < c < n - 1. \]

Then

\[ A^{(t)}(x) = T(x)^t a_c x^c = x^t (1 + x^{n-2})^t a_c x^c = a_c x^{c + t} \sum_{i=0}^{t} {t \choose i} (n-2)^i, \]

so

\[ A^{(t)}(x) = a_c \sum_{i=0}^{t} {t \choose i} (n-2)^i x^{c+t}. \]

Notice the binomial coefficients appearing as coefficients here, modulo \( k \). This explains the Sierpinski Gasket pattern previously observed for this transition rule.

Since any initial CA state may be written as a sum of such terms, the polynomial representation in principle gives us a closed form analytic expression describing the time evolution of additive CA from any initial state.
3.6 Sum of Squares: n = 200

Now for our first example of a non-additive CA, one with transition rules not representable by convolution kernels. The neighborhood of a cell in this CA consists of the cell itself along with its two immediately adjacent cells. The transition rule is given by

\[
C_{i}^{(t+1)} = C_{i-1}^{(t)} + 2C_{i}^{(t)} + C_{i+1}^{(t)} \mod k.
\]

The following figures are the result of computing 100 generations with various values for modulus \( k \). Notice the appearance of the Sierpinski gasket pattern when \( k \) is a power of 2.

Figure 11: Modulus \( k = 7 \)

Figure 12: Modulus \( k = 8 = 2^3 \)
Figure 13: Modulus $k = 9$

Figure 14: Modulus $k = 56$

Figure 15: Modulus $k = 128 = 2^7$
3.7 A Strange Non-Additive Example, \( k = 256, \ n = 512 \)

Now we examine a rather bizarre example of a non-additive CA. Although this example was constructed with 256 states, the essential behavior of this system only requires 2 states. The other 254 states were provided primarily to add color, so just consider any color to represent "alive" and white to represent "dead". To define the transition rule for this CA, let \( L \) be the number of live cells in the set

\[
\{ C_{i-3}^{(t)}, C_{i-1}^{(t)}, C_{i+1}^{(t)}, C_{i+3}^{(t)}, C_{i+4}^{(t)} \}.
\]

Then

\[
C_i^{(t+1)} = \begin{cases} 
\text{alive, if } L = 3; \\
\text{alive, if } L = 2 \text{ and } C_i^{(t)} \text{ is alive}; \\
\text{dead otherwise}.
\end{cases}
\]

This strange transition rule is a 1-dimensional version of the 2-dimensional CA discussed in section 4.5. For the following run, the initial configuration was a random assignment of life and death values to cells, with the probability of life being \( 1/2 \). Figure 16 shows a section of this run that looks amazingly like elementary particle tracks in a vacuum chamber. Figure 17 shows magnified details of Figure 16.

![Figure 16](image1.png)

**Figure 16**: 1-Dimensional version of the 2-dimensional *Game of Life* CA

![Figure 17](image2.png)

**Figure 17**: Details of Figure 16. (B) shows a cycle.
Chapter 4

2-Dimensional Cellular Automata

4.0 Preliminaries

A 2-dimensional lattice is an array of cells one may imagine tiling an infinite planar surface. Each cell takes on a value from a finite set at each tick of the clock. We denote the state of the CA at time $t$ by

$$C^{(t)} = \begin{bmatrix}
  \cdots & \cdots & \cdots & \cdots \\
  C^{(t)}_{-1,-1} & C^{(t)}_{-1,0} & C^{(t)}_{-1,1} & \cdots \\
  \vdots & C^{(t)}_{0,-1} & C^{(t)}_{0,0} & C^{(t)}_{0,1} & \cdots \\
  C^{(t)}_{1,-1} & C^{(t)}_{1,0} & C^{(t)}_{1,1} & \cdots \\
  \vdots & \vdots & \vdots & \ddots
\end{bmatrix}.$$ 

We will only consider CA possessing periodic boundary conditions, previously encountered in the previous chapter in the 1-dimensional case. The 2-dimensional case is similar, but the wrap-around takes place in two directions. Not only does the right wrap-around to the left and the left wrap around to the right, but now the bottom wraps-around to the top and the top wraps-around to the bottom. As before, to make the above infinite array finite implies we have some specific number of cells in mind; in this 2-dimensional case we must have a specific number of rows and number of columns for our array in mind, and we call these dimensions $n$ and $m$ respectively. Then the above infinite array becomes a finite $n \times m$ array cells,

$$C^{(t)} = \begin{bmatrix}
  \cdots & \cdots & \cdots & \cdots \\
  C^{(t)}_{0,0} & C^{(t)}_{0,1} & \cdots & C^{(t)}_{0,m-1} \\
  C^{(t)}_{1,0} & C^{(t)}_{1,1} & \cdots & C^{(t)}_{1,m-1} \\
  \vdots & \vdots & \ddots & \vdots \\
  C^{(t)}_{n-1,0} & C^{(t)}_{n-1,1} & \cdots & C^{(t)}_{n-1,m-1}
\end{bmatrix}.$$ 

One way to visualize this periodic boundary condition is to imagine joining the top and bottom of this array to form a tube, and then joining the ends of the tube to form a torus. Thus our CA lattice has the topology of a torus, depicted in Figure 1 with $n = m = 21$. 
Equivalently, one could define initial conditions for an $n \times m$ array of cells, and then tile the entire infinite plane with copies of this rectangular array of values. This can be accomplished by requiring that whenever $a$ is congruent to $b$ modulo $n$, and $c$ is congruent to $d$ modulo $m$, we then must have that

$$C_{a,c}^{(t)} = C_{b,d}^{(t)}.$$ 

In short, all arithmetic on cell indices is performed modulo $n$ for the first index and $m$ for the second index.

Although CA neighborhoods in general may be defined arbitrarily, for the CA we will examine the same relative neighborhood is defined for each cell. For example if the neighborhood of one cell consists of the four immediately adjacent cells, then this is also true of all other cells. Such CA are called *homogeneous*.

For the examples that follow, we let $k$ denote the number of states possible for each cell, and further identify the states with the integers $0, 1, 2, \ldots, k - 1$. Computations on cell values are performed modulo $k$.

The number of possible states for this CA is $k^{NM}$. In example 4.2, with $k = 256$ and $N = M = 500$, the number of states is thus $256^{500 \times 500}$, more than $10^{602059}$.

Before moving on to examples we first discuss a generalization of the convolution kernels we found so useful in representing 1-dimensional CA transition rules in the previous chapter. These 2-dimensional convolution kernels are used extensively in the field of image processing.
4.1 2-Dimensional Convolution Kernels

One of the most oft used procedures in the field of image processing is that of filtering an image by convolving it with a 2-dimensional kernel. Convolving an image with a 2-dimensional kernel is very similar to the 1-dimensional case, and as before we demonstrate the procedure with a specific example. The convolution kernel

\[
\begin{bmatrix}
0 & -1 & 0 \\
-1 & 4 & -1 \\
0 & -1 & 0 \\
\end{bmatrix}
\]

applied to a digital image could serve as an edge extractor, since homogenous regions in an image are mapped to zero by this operation. As in the 1-dimensional case, we center this kernel over each pixel of the image and do our multiply and add procedure to obtain the output pixel value. We will only show how this procedure is used to compute the output pixel for one input pixel; in actual convolution this must be repeated for each pixel in the image. The yellow central cell represents the input pixel, with a value of 6. Think of the following little table as being a piece of a very large table of numbers, the image.

\[
\begin{bmatrix}
1 & 2 & 3 \\
5 & 6 & 7 \\
8 & 9 & 1 \\
\end{bmatrix}
\]

Imagine the previous kernel table being “over” this little piece of the overall image. The new output value for the input pixel is computed as follows:

\[
0*1 + (-1)*2 + 0*3 + (-1)*5 + 4*6 + (-1)*7 + 0*8 + (-1)*9 + 0*1 = 1
\]

Thus the output value for the pixel at the yellow position would be 1.

In mathematical notation, given an input \( N \times M \) image \( x_{ij} \) (where \( i \) ranges from 0 to \( N \) and \( j \) ranges from 0 to \( M \)), and a \((2Q-1) \times (2Q-1)\) convolution kernel, \( k_{f,h} \) (where \( f \) and \( h \) both range from 1-2Q to 2Q-1), we have

\[
y_{ij} = \sum_{n=-Q}^{Q} \sum_{m=-Q}^{Q} k_{n,m} x_{i+n \mod N, j+m \mod M} \mod k_g
\]

where \( y_{ij} \) is the output value of the pixel at position \((i, j)\) in the image.

Additive CA are amenable to detailed mathematical analysis, and admit a polynomial representation similar to the 1-dimensional case (but with two variables).
4.2 Example: k = 256, n = m = 500,

For this example the state of our cellular automata at generation \( t \) is a 500 x 500 array, which we denote

\[
C^{(t)} = \begin{bmatrix} C_{ij}^{(t)} \end{bmatrix}.
\]

Where each \( C_{ij}^{(t)} \) takes values from the numbers 0, 1, 2, …, 255. The usual wrap-around is in effect, with the row above the topmost row being the lowest row, and the column to the right of the rightmost column is the leftmost column. This wrap-around is accomplished by imposing modulo 500 arithmetic on the \( ij \)-indices. For this CA, the neighborhood of a cell is the cell itself along with the cell to its left, the cell to its right, the cell above, and the cell below. The transition rule of our CA is to take the sum of all neighboring cells, modulo 256.

\[
C_{ij}^{(t+1)} = T \left( N_t(i, j) \right) = C_{i,j,1}^{(t)} + C_{i,1,j}^{(t)} + C_{i,j+1}^{(t)} + C_{i+1,j}^{(t)} \pmod{256}.
\]

This transition rule is more compactly expressed via the 2-dimensional convolution kernel

\[
\begin{bmatrix} 1 & 0 \end{bmatrix}.
\]

Now to get our dynamics going all we need is an initial state for each cell. We start with a very simple initial state, with all cells zero except for one cell with value one. For the following images, black represents a zero value and colors represent the other 254 states of the CA.

Initial state, zeroing in on the sole nonzero cell (with value 1), at 5x magnification.

From this simple seed an ever growing (up to the limiting 500x500 dimensions of our CA space) mandala of intricate complexity and symmetry unfolds. Figure 2 shows how our
simple seed has grown after 100 generations; Figure 3 shows the results of 250 generations of growth.

Figure 4 shows a new initial state of 255 seeds (with values 1…255), and Figure 5 depicts how these 255 seeds have grown after 13 generations.

Figure 2: Generation 100 at magnification 3.5x
Figure 3: Generation 250 at magnification 1.4x
Figure 3a: Detail of Figure 3 at magnification 5.4x
Figure 4: Initial state of 256 seeds, values from 0 through 255.

Figure 5: Generation 13, from initial
Figure 5a: Magnification of portion of Figure 5
4.3 Example: $k = 256$, $n = m = 500$,

$$\begin{bmatrix}
-2 & 1 \\
4 & -2 \\
-2 & 1 \\
\end{bmatrix}$$

For this example the state of our cellular automata at generation $t$ is a $500 \times 500$ array, as in the previous example, which we again denote

$$C^{(t)} = \begin{bmatrix}
C_{00}^{(t)} & C_{01}^{(t)} & \cdots & C_{0,499}^{(t)} \\
C_{10}^{(t)} & C_{11}^{(t)} & \cdots & C_{1,499}^{(t)} \\
\vdots & \ddots & \ddots & \vdots \\
C_{499,0}^{(t)} & \cdots & C_{499,499}^{(t)} \\
\end{bmatrix}$$

Each $C_{ij}^{(t)}$ takes values from the numbers $0, 1, 2, \ldots, 255$. The usual wrap-around is in effect, just as described in Example 4.2, by imposing modulo 500 arithmetic on the $ij$-indices. For this CA, the neighborhood of a cell consists of the cell itself along with the cell’s eight nearest adjacent cells. The transition rule of our CA can be represented by the convolution kernel

$$\begin{bmatrix}
-2 & 1 \\
4 & -2 \\
-2 & 1 \\
\end{bmatrix}$$

or given explicitly by

$$C_{ij}^{(t+1)} = T(N_t(i,j)) = C_{i+1,j-1}^{(t)} - 2C_{i+1,j}^{(t)} + C_{i+1,j+1}^{(t)} - 2C_{i,j-1}^{(t)} + 4C_{i,j}^{(t)} - 2C_{i,j+1}^{(t)} + C_{i-1,j-1}^{(t)} - 2C_{i-1,j}^{(t)} + C_{i-1,j+1}^{(t)} \pmod{256}.$$  

We start with a very simple initial state (as in the first example of 4.2), with all cells zero except for one cell with value equal to one. For the following images, white represents a value of zero and colors represent nonzero values.

From this simple seed we once again see unfold a mandala of intricate complexity and symmetry. Figure 6 shows how this simple seed has grown after 250 generations. We then start with a new initial state of 255 seeds (depicted in Figure 4). Figure 7 shows how these seeds have progressed after 10 generations.
Figure 6a: Detail of Figure 6 (5x magnification)
Figure 7a: Detail of Figure 7 (5x magnification)
4.4 Polynomial Representation of 2-Dimensional Additive CA

2-dimensional additive CA have polynomial representations similar to those of the 1-dimensional case we encountered in section 3.5. For all of the following we assume the usual wrap-around periodic boundary conditions previously described.

Each state of a 2-dimensional CA can be considered a matrix of integers, modulo $k$. We here assume the usual wrap-around boundary conditions, and assume the matrix formed by the array of cell states has $m$ rows and $n$ columns.

Each such state of a CA can be represented by the corresponding characteristic polynomial in two variables,

$$A(x) = a_{00} + a_{01}y + \cdots + a_{0n}y^n + a_{10}x + a_{11}xy + \cdots + a_{1n}xy^n + \cdots + a_{mn}x^my^m + \cdots = \sum_{i=0}^{m} \sum_{j=0}^{n} a_{ij}x^i y^j .$$

Transition rules representable by convolution kernels can also be represented by polynomials, in a manner analogous to that described in the 1-dimensional case. We will illustrate this by means of an example, the transition rule described by the convolution kernel of Example 4.3:

This kernel translates into the polynomial

$$T(x, y) = x^{-1}y^{-1} - 2x^{-1} + x^{-1}y - 2y^{-1} + 4 - 2y + xy^{-1} - 2x + xy$$

$$= x^{n-1}y^{m-1} - 2x^{n-1} + x^{n-1}y - 2y^{m-1} + 4 - 2y + xy^{m-1} - 2x + xy .$$
As in the 1-dimensional case, multiplying A by T will yield the next generation of the CA. Here we do the $x$ exponent arithmetic modulo $n$, the $y$ exponent arithmetic modulo $m$, and the coefficient arithmetic modulo $k$.

Although the algebra gets a bit involved for higher dimensional cases, the polynomial representation provides an interesting conceptual tool for understanding CA dynamics. As we have seen, the superposition principle is simply a manifestation of the distributive property of multiplication over addition for polynomials. This superposition principle says basically if you've seen one cell evolve, you've seen all possible configurations of cells evolve.

Unfortunately, the most interesting of CA are not additive and don't have this simple superposition property.
4.5 A Famous Non-Additive CA: *The Game of Life*

Conway’s Game of Life is a simple two-dimensional CA, where each node can be in only one of two states, alive or dead. If one imagines the nodes as squares on a checkerboard, the neighborhood of a node consists of the eight neighboring squares - left, right, up, down, as well as the four diagonal squares. The transition rules are quite simple. If a given square is dead, then it will become alive in the next “generation” provided it has exactly three living neighbors. If a given square is alive, it remains alive for the next generation only if it has two or three living neighbors. In the following illustrations, consider the blue cells alive, all other cells currently dead.

![The yellow cells form the neighborhood of the blue cell.](image)

There will be a birth at cell 1, since it is adjacent to three living cells. However, all the blue cells will die as they are adjacent to no live cells.

![All blue cells shall continue to live, & births will occur at the “!” cells.](image)

In the *Game of Life* figures, white represents dead cells; all other cells are alive. The live cells are color coded to represent their age. Cerise cells are newborn, red are very young, orange are older, yellow older yet, green still older, blue very old and violet are ancient, more than 250 generations old. Figure 8b shows evolution from the initially random state of Figure 8a, and Figure 9 shows evolution from a row of initially living cells.
Figure 8a: Initially random state, with probability of life = .3

Figure 8b: After 500 generations, various typical life structures have emerged.
Figure 9: This figure shows evolution from an initial row of 71 live cells. The number under each image is the subsequent generation number. In generations 200 on, gliders can be seen at each corner moving outward. In generation 400 the gliders have moved out of the frame.
4.6 Example: A Non-Additive CA Involving Euler's Totient Function

This example is our second example of a non-additive 2-dimensional CA. For this example $k = 256$, and the total number of cells is 500 x 500. Thus each generation of our cellular automata will be a 500 x 500 matrix, which we denote

$$C^{(t)} = \begin{bmatrix} C_{0,0}^{(t)} & \cdots & C_{0,499}^{(t)} \\ \vdots & \ddots & \vdots \\ C_{499,0}^{(t)} & \cdots & C_{499,499}^{(t)} \end{bmatrix}$$

Each $C_{ij}$ takes values from the numbers 0, 1, 2, ... , 255. The usual wrap-around is in effect, with the row above the topmost row being the lowest row, and the column to the right of the rightmost column is the leftmost column. This wrap-around is accomplished by imposing modulo 500 arithmetic on the ij-indices. For this CA, the neighborhood of a cell consists of the cell itself along with the cell’s eight nearest adjacent cells. The transition rule of our CA uses a function borrowed from number theory, the Euler totient function, $\phi$ (see Appendix 4). This transition rule is given by

$$C_{ij}^{(t+1)} = T(N_t(i, j)) = \sum_{a=-1}^{1} \sum_{b=-1}^{1} \phi(C_{i+a, j+b}^{(t)} \mod 256).$$

For the following figures, white represents a zero value and colors represent the other 254 states of the CA. We start with a very simple initial state, with all cells zero except for one cell with value one.

```
Initial state (detail), one nonzero cell
(with value 1), at 5x magnification.
```

From this simple seed we once again see a mandala of intricate complexity and symmetry unfold. Figure 10 shows how this simple seed has grown after 250 generations.

We then start with a new initial state of 255 seeds (Figure 4). Figure 11 shows how these seeds have progressed after 12 generations.
Figure 10: Generation 250 of the Euler totient transition rule
Figure 10a: Detail from Figure 10 (5x magnification)
Figure 11: Generation 12, starting from 255 seeds

Figure 11a: Detail of Figure 11 (magnified 5.2x)
Chapter 5

Ecosystem Modeling Using Cellular Automata

5.0 CAecosystem

The program CAecosystem illustrates how CA can be used to create simple and intuitive models of complex real world phenomena that can help illuminate the underlying principles. This ecosystem simulator allows the user to set various parameters of an artificial ecosystem with three interdependent species – plants, prey, and predators. The user can then set in motion this artificial universe, observe the ensuing dynamics, and display the population data in various ways. These created worlds have a toroidal topology, which simply means that a critter creeping off the bottom of the display screen instantly appears crawling onto the top of the screen, and a one crawling off the right of the screen enters on the left. So don’t worry, these critters can’t escape and infest your home!

For my first run I used a 500x500 cell space, with the initial probability of a cell being a prey set to 0.212 and the initial probability of a cell being a predator set to 0.029. The other parameters were left at their default settings. The initial state of the ecosystem is depicted in Figure 1.

![Initial state of the ecosystem (detail), where green = plants, yellow = prey, red = predators.](image)
Figure 2 depicts the result of evolving this CA for 2000 generations.

![Figure 2](image1.png)

**Figure 2:** 2000 generations later (detail); here the purple cells are devoid of life.

Figure 3 shows the graphs of the populations as a function of time, up to generation 2000, sampled every fourth generation. Without the regulating influence of the predator population, the prey would eat all the plants and then would all starve to death.

![Figure 3](image2.png)

**Figure 3:** Normalized population graphs; the horizontal axis represents number of generations.
The three populations are each graphed on a normalized scale, so as to be shown on the same graph with good resolution. The predator population curve (red) and the prey curve (yellow) show the out of phase oscillatory behavior typical of predator/prey systems. In such systems large numbers of prey result in easy hunting for the predators, which results in large numbers of predator births. This then results in a decline in the prey population, as they are being eaten in large numbers. But then the predator population goes into a decline, as food is scarce. When the predator population becomes relatively scarce, the prey’s population growth is unchecked, and rapidly increases, thus starting the cycle once again.

This cycle is less obvious between the prey and the plants (relative to whom they are the predators), due to the relatively rapid growth of plants in this simulation. The plant population graph is almost the negative of the prey graph, i.e. the graphs are 180 degrees out of phase with each other. Notice that the initially large oscillations (called transients) settle down to relatively stable amplitudes, for all the species. It is interesting to note that the predators and plants have a symbiotic relationship with each other in this ecosystem.

Another way to graph this same information is via the 3-dimensional phase portrait, where at each sample point in time we plot a point with coordinates equal to the populations of the three species at that time. The mathematician Poincare in his attempt to understand the global dynamics of differential equations first introduced this type of graph. Below are two such graphs, looked at from different angles.

FIGURE 4: 3-dimensional phase portraits, representing the same data as in Figure 3, shown from two different angles.
The program CAecosystem allows one to perform real-time rotations of these phase portraits, which helps one to see the 3-dimensional shape of the curve. Oscillation in the previous graphs of Figure 3 manifest in the phase portrait of Figure 4(a) as an almost circular orbiting of the curve about the equilibrium point. The previous observation concerning the dampening of the oscillations in the previous graphs here can be seen in terms of the tightening of the spiral about the equilibrium point.

This same qualitative behavior is seen in solutions to the Voltera-Lotka differential equations traditionally used to model ecosystem population dynamics. A little experimentation will reveal that an imprudent choice of ecosystem parameters results in a situation where the initial transients are too wild, resulting in extinction of all animal life (often all plant life as well).

Even in the differential equation models, the equilibrium point is virtually impossible to attain, since it would almost always entail fractions of an animal being alive (one of the absurdities of using continuous models to model discrete phenomena). This difficulty is even more acute in discrete models, so the oscillatory behavior is unavoidable. The initial conditions in this experiment were chosen very close to this equilibrium point. This can be seen in Figure 4(a); the trajectory begins near the center of the loops.

In spite of this one still sees the initial transients occur prior to the system settling down into a more regular oscillatory behavior. One reason for the initial transients in our ecosystem is that the initial spatial arrangement of cell states is random, whereas the distribution of cell states once steady-state has been achieved is far from random. The transient period involves the spatial drifting of cell states from the initial random configuration to a more optimal one (compare the distribution of cell states in Figures 1 and 2).

Another reason for the initial transients becomes evident upon examination of the phase portrait of Figure 4(b). Here the initial point of the trajectory is seen up and to the right of the main coil. Notice that this point has a plant (green) coordinate greater than points in the main coil of the trajectory. Thus our ecosystem was initially a bit too lush with plant life.

Next we take a look at how things have developed for our little ecosystem after 11,200 generations. The actual state of our ecosystem looks very much like it did in Figure 2. Figure 5 shows the population curves up to generation 11,200.
Figure 5: Normalized population curves
Appendix A: Modular Arithmetic

Modular arithmetic (sometimes described as “clock arithmetic) is just ordinary arithmetic with the integers involving addition and multiplication, but with one additional frequently performed procedure: Whenever a result is obtained, it is divided by the modulus, and only the remainder is reported as the final result. For example, let the modulus be 5, and consider a simple arithmetic computation.

\[ 3 + 4 = 7 \]

Now 5 goes into 7 once with a remainder of 2, so ‘2’ would be given as the result of the above computation modulo 5. This is usually written

\[ 3 + 4 \equiv 2 \pmod{5} \]

This reads “three plus four is congruent to two modulo 5”. Any integer, \( n \), can be reduced modulo 5 to one of the digits 0, 1, 2, 3, 4. If \( n \) is positive it reduces to the remainder when divided by 5, as described above. If \( n \) is negative just keep adding 5 to it until it becomes positive, and it will be one of these digits. Notice that since

\[ 5 \equiv 0 \pmod{5} \]

Adding 5 won’t affect the result, modulo 5. The additive inverse of 1 is \(-1\) in the integers, and since \((-1) + 5 = 4\), it must be that

\[ -1 \equiv 4 \pmod{5} \]

In general,

\[ n \equiv 0 \pmod{n} \]

And so

\[ -1 \equiv n - 1 \pmod{n} \]

Modulo 2 arithmetic has only two digits, 0 and 1, and can be summarized by three very simple rules (here we simplify the above notation; ‘\(=\)’ here means congruent modulo 2):

\[ 0 + 0 = 0 \]
\[ 0 + 1 = 1 \]
\[ 1 + 1 = 0 \]

Verify that these rules yield the same results as the previously described procedure. These rules are also those of the ‘exclusive or’ operation in Boolean algebra, which has central importance in the design of digital circuits.
Appendix B: Pascal's Triangle

The simplest way to construct Pascal’s famous triangle is:

start with a one

\[
\begin{array}{c}
1 \\
\end{array}
\]

add the values diagonally above

\[
\begin{array}{c}
1 \ 1 \\
\end{array}
\]

add once again

\[
\begin{array}{c}
1 \ 2 \ 1 \\
\end{array}
\]

and again

\[
\begin{array}{c}
1 \ 3 \ 3 \ 1 \\
\end{array}
\]

and again

\[
\begin{array}{c}
1 \ 4 \ 6 \ 4 \ 1 \\
\end{array}
\]

. . . and so on, till you get tired.

The numbers occurring in Pascal’s triangle are known as the binomial coefficients, since they arise as coefficients in the binomial expansion

\[
b_n^k = \binom{n}{k} = \frac{n!}{k!(n-k)!}.
\]

The binomial coefficient

\[
\binom{n}{k}
\]

is the number at the \(k\)th position of the \(n\)th row of Pascal’s triangle, if one starts numbering the rows with zero and also starts counting position within a row with zero.

Pascal’s triangle taken modulo 2 is transformed into an interesting pattern of ones and zeros…

\[
\begin{array}{cccccccccccc}
1 & 1 \\
1 & 2 & 1 \\
1 & 3 & 3 & 1 \\
1 & 4 & 6 & 4 & 1 \\
1 & 5 & 10 & 10 & 5 & 1 \\
1 & 6 & 15 & 20 & 15 & 6 & 1 \\
1 & 7 & 21 & 35 & 35 & 21 & 7 & 1 \\
\end{array}
\]

…recognizable as none other than Sierpinski’s gasket. This pattern of ones and zeros forms successive approximations to the fractal. In particular, computing down to row \(2^{n+1}-1\) yields an approximation to \(S_n\) (see the discussion in Appendix C).
Appendix C: Geometric Construction of Sierpinski's Gasket

In 1916 Sierpinski described the following geometrical construction of the fractal triangle now bearing his name. We start with an equilateral triangle and its interior, denoted $S_0$.

Call this triangle $S_0$. To construct $S_1$ remove an inverted equilateral triangle from the center of $S_0$, with sides $\frac{1}{2}$ the length of the original triangle’s.

To construct $S_2$ we now apply the previous operation to each of the three triangles which make up $S_1$

$S_3$ is similarly constructed, by removing a triangle from each of the 9 triangles that make up $S_2$. 
Continuing in this way we obtain a sequence of geometric figures $S_0, S_1, S_2, S_3, S_4, \ldots$, which may be considered to be a list of better and better approximations to the Sierpinski gasket, which itself is the limit of this sequence.

Notice the similarity to the following pattern evolved by the CA with rule $[1 \ 0 \ 1]$ modulo 2, for 256 generations. This part of the CA evolved pattern looks like $S_7$. By evolving the CA for $2^{n+1}$ generations one obtains a figure resembling the approximation $S_n$.

Interestingly, Sierpinski’s gasket also arises by writing Pascal’s triangle modulo 2:

\[
\begin{array}{cccccccc}
1 & & & & & & & \\
1 & 1 & & & & & & \\
1 & 2 & 1 & & & & & \\
1 & 3 & 3 & 1 & & & & \\
1 & 4 & 6 & 4 & 1 & & & \\
1 & 5 & 10 & 10 & 5 & 1 & & \\
1 & 6 & 15 & 20 & 15 & 6 & 1 & \\
1 & 7 & 21 & 35 & 35 & 21 & 7 & 1 \\
\end{array}
\]

This pattern of ones and zeros in fact forms the successive approximations to the fractal, the more rows of Pascal’s triangle one computes. By computing $2^n$ rows of Pascal’s triangle one obtains a figure resembling approximation $S_n$ to Sierpinski’s gasket.
Appendix D: *The Euler Totient Function, φ*

This function features prominently in number theory. It is defined for the positive integers \( n \) by

\[
\phi(n) = \text{“The number of numbers relatively prime to and less than n.”}
\]

Two numbers are relatively prime if they have no common divisors but 1. Thus 6 and 10 are not relatively prime since they are both divisible by 2, while 9 and 10 are relatively prime since only 1 divides both of them.

The following table lists the numbers relatively prime to the first few integers and the resulting values of Euler’s totient function.

<table>
<thead>
<tr>
<th>( n )</th>
<th>Numbers relatively prime to and less than n</th>
<th>( \phi(n) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1, 2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>1, 3</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>1, 2, 3, 4</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>1, 5</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>1, 2, 3, 4, 5, 6</td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>1, 3, 5, 7</td>
<td>4</td>
</tr>
<tr>
<td>9</td>
<td>1, 2, 4, 5, 7, 8</td>
<td>6</td>
</tr>
<tr>
<td>10</td>
<td>1, 3, 7, 9</td>
<td>4</td>
</tr>
<tr>
<td>11</td>
<td>1, 2, 3, 4, 5, 6, 7, 8, 9, 10</td>
<td>10</td>
</tr>
<tr>
<td>12</td>
<td>1, 5, 7, 11</td>
<td>4</td>
</tr>
<tr>
<td>13</td>
<td>1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12</td>
<td>12</td>
</tr>
<tr>
<td>14</td>
<td>1, 3, 5, 9, 11, 13</td>
<td>6</td>
</tr>
<tr>
<td>15</td>
<td>1, 2, 4, 7, 8, 11, 13, 14</td>
<td>8</td>
</tr>
<tr>
<td>16</td>
<td>1, 3, 5, 7, 9, 11, 13, 15</td>
<td>8</td>
</tr>
<tr>
<td>17</td>
<td>1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16</td>
<td>16</td>
</tr>
<tr>
<td>18</td>
<td>1, 5, 7, 11, 13, 17</td>
<td>6</td>
</tr>
</tbody>
</table>

\( \phi(1) \) is defined to be 1. As this table suggests (and a little thought proves), for prime numbers \( p \) we have

\[
\phi(p) = p - 1.
\]

Thus \( \phi \) has local maxima at prime numbers, and local minima at numbers with large numbers of factors.