Chapter 3

One-Dimensional Cellular Automata

Cellular automata may be viewed as computers, in which data represented by initial configurations is processed by time evolution.

Stephen Wolfram

3.1 The Cellular Automaton

We will consider a lattice network of cells that are most commonly square in shape, but the cells can be hexagonal and other shapes as well. Each cell can exist in k different states, where k is a finite number equal to or greater than 2. One of these states has a special status and will be known as the 'quiescent state'. The simplest case where each cell can exist in two possible states (not simultaneously), can be denoted by the symbols 0 and 1 and graphically by white and black, respectively. In more anthropomorphic terms, we can think of cells in the 0 (white/quiescent) state as 'dead' and those in the 1 (black) state as 'alive'.

The lattice of cells can be $n(\geq 1)$ dimensional, but most of the work on cellular automata has been for one and two dimensions, particularly the former. In the sequel, we shall generally consider square cells as the basic unit of our automata. In the one-dimensional case, these form a row of adjacent boxes. In principle, the number of boxes in any array is infinite, but for practical purposes it will simply be taken sufficiently large to illustrate the behavior in question, often with certain conditions imposed on the cells along the boundaries. In some cases, intentional restrictions will be made on the size of the lattice as well.

In order for the cells of our lattice to evolve we need time to change, and this is done in an unusual manner by considering changes to the states of cells in the lattice only at discrete moments in time, that is, at time steps $t = 0, 1, 2, 3....$ as in the ticking of a clock. The time $t = 0$ usually denotes the initial time period before any change of the cells' states has taken place. One further ingredient that is needed for our cellular lattice to evolve with discrete time steps is a local rule or *local transition function* governing how each cell alters its state from the present instant of time to the next based on a set of rules that take into account the cell's current state and the current state of its neighbors. Of course, which cells are considered to be neighbors, needs to be precisely defined. This alteration of cell states takes place *synchronously* for all cells in the lattice. The transition function can be deterministic or probabilistic, but in most cases (some exceptions are models discussed in the chapter on Applications) we will consider only the former. The lattice of cells, the set of allowable states, together with the transition function is called a *cellular automaton.* Any assignment of state values to the entire lattice of cells by the transition function results in a *configuration* at a any particular time step.

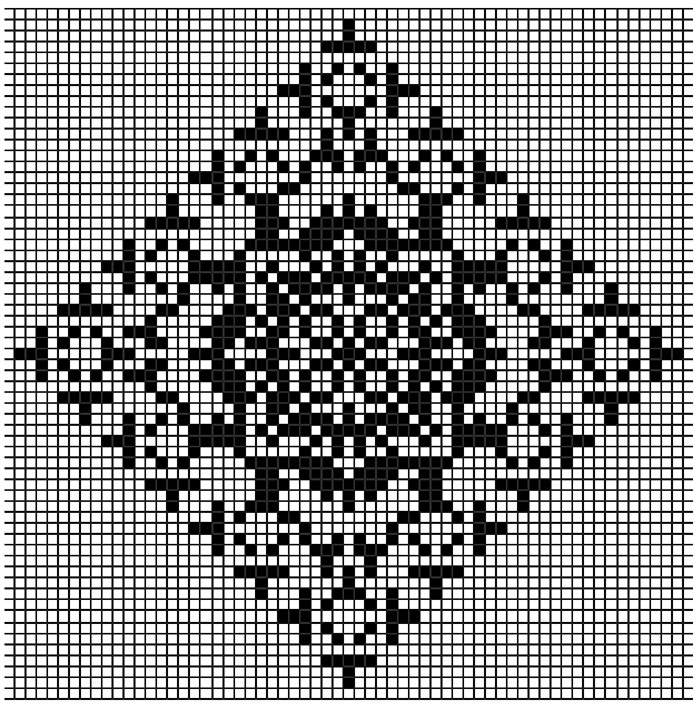


Figure 3.1: A two-dimensional configuration with each cell taking one of two state values, black or white according to a particular local transition function. The first printed reference to a system such as the above, even using the word 'automata', is to be found in the paper by Ulam [1950].

Therefore the three fundamental features of a cellular automaton are:

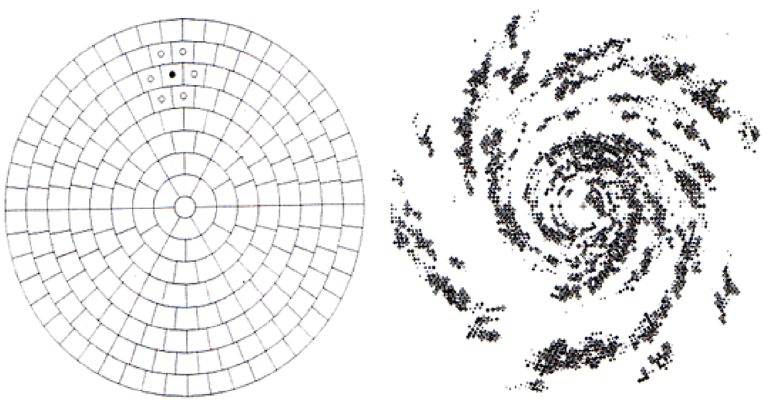


Figure 3.2: In this setting the neighbors of each cell change due to the differential rotation of the rings of the polar grid that is used to emulate galaxy formation. The black circle is an active region of star formation which induces star formation in its neighbors with a certain probability at the next time step. At right is a typical galaxy simulation.

- *homogeneity*: all cell states are updated by the same set of rules;
- *parallelism*: all cell states are updated simultaneously;
- *locality*: the rules are local in nature.

There has been some recent work on the *asynchronous* updating of cell states (cf. eg. Bersini and Detour [1994], Ingerson and Buvel [1984], Schönfisch & de Roos [1999]). This can be achieved in a number of different ways and is discussed in Section 4.4.

Other liberties with the rules of a cellular automaton can be taken. In one interesting example, galaxy formation has been simulated by Shulman & Seiden [1986] using a CA approach on a polar grid whose rings rotate at different rates (see Figure 3.2). Theirs was a simple percolation model and achieves the basic structure of a spiral galaxy without a detailed analysis of the astrophysical dynamics involved.

In another digression from the conventional CA definition, Moshe Sipper [1994] considered different rules for different cells as well as the evolution of rules over time. There will be other instances in the sequel when we will stray slightly from strict adherence to the classical cellular automaton definition.

A fundamental precept of cellular automata is that the local transition function determining the state of each individual cell at a particular time step should be based upon the state of those cells in its immediate neighborhood at the previous time step or even previous time steps. Thus the rules are strictly local in nature and each cell becomes an information processing unit integrating the state of the cells around it and altering its own state in unison with all the others at the next time step in accordance with the stipulated rule. Thus, any global patterns of the system are an *emergent* feature of the effect of the locally defined transition function. That is, global features emerge from the strictly local interaction of individual cells each of which is only aware of its immediate environment. This emergent behavior will be discussed more comprehensively in Chapter 5, but it is a salient feature that one should always be aware of in our development of cellular automata theory.

3.2 Transition functions

Most of the dynamical features of cellular automata can be found in the study of the one-dimensional case. Here we define a neighborhood of a cell c having radius (range) r as the r cells to the left of c and the same number of cells to the right of c. Counting c itself, this neighborhood contains $2r+1$ cells. In the simplest case $r = 1$ and $k = 2$ for the allowable states. In this instance, a three-cell neighborhood with two different states 0 and 1 for each cell can be expressed in $2^3 = 8$ different ways. All eight neighborhood-states with $r = 1$ and $k = 2$ are illustrated below (Figure 3.3), and in general there are k^{2r+1} one-dimensional neighborhood-states..

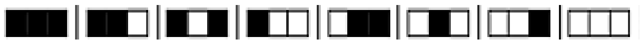


Figure 3.3: The eight possible neighborhood-states with $r = 1$ and $k = 2$.

Let us adopt this notation: $c_i(t)$ denotes the state of the ith cell at time t (Figure 3.4).

At the next time step, $t+1$, the cell state will be $c_i(t+1)$. Mathematically we can express the dependence of a cell's state at time step $t+1$ on the state of its left-hand and right-hand nearest neighbors $c_{i-1}(t)$ and $c_{i+1}(t)$ at time step t, by the relation:

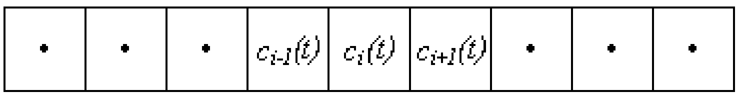


Figure 3.4: The cell states of the central cell $c_i$and its two nearest neighbors c_{i-1} and c_{i+1} at the time step t.

$$c_i(t+1) = \varphi\left[c_{i-1}(t), c_i(t), c_{i+1}(t)\right],$$

where φ is the local transition function. For example, consider the simple transition rule

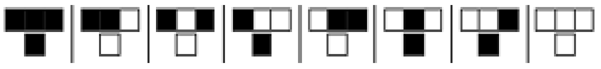
$$c_i(t+1) = c_{i-1}(t) + c_i(t) + c_{i+1}(t) \mod 2, \tag{3.1}$$

where mod 2 means taking the remainder after division of the indicated sum by 2, resulting in either 0 or 1. We can put this rule into a (transition) table format by adding $c_{i-1}(t) + c_i(t) + c_{i+1}(t) \mod 2$ for the eight possible different input values:

$c_{i-1}(t)$	$c_i(t)$	$c_{i+1}(t)$	$c_i(t+1)$
1	1	1	1
1	1	0	0
1	0	1	0
1	0	0	1
0	1	1	0
0	1	0	1
0	0	1	1
0	0	0	0

(3.2a)

A very convenient way to illustrate the allowable rules for one-dimensional cellular automata with $r = 1$ and $k = 2$ is to indicate the state (color) of the middle cell at the next time step, given the state (color) of itself and its two nearest neighbors:



(3.3)

Here the middle cell at time t (in the top row) has its state altered according to the state of its two nearest neighbors and its own state to yield the cell's new state at time $t + 1$ (bottom row). This also represents the preceding rule 3.1 above. Observe that in four of the eight cases a black cell appears. One consequence of this is that if we took a disordered array of an infinite number of cell sites, then the average fraction (or *density*) of black cell sites that evolve after one iteration will be 0.5.

The cellular automaton defined by this rule has the graphical representation depicted in Figure 3.5 starting with a single black cell with each subsequent generation of the automaton appearing on the next line down.

So how many possible rules are there for $r = 1$ and $k = 2$? Since there are 8 possible neighborhood-states of 3 cells and each of these results in

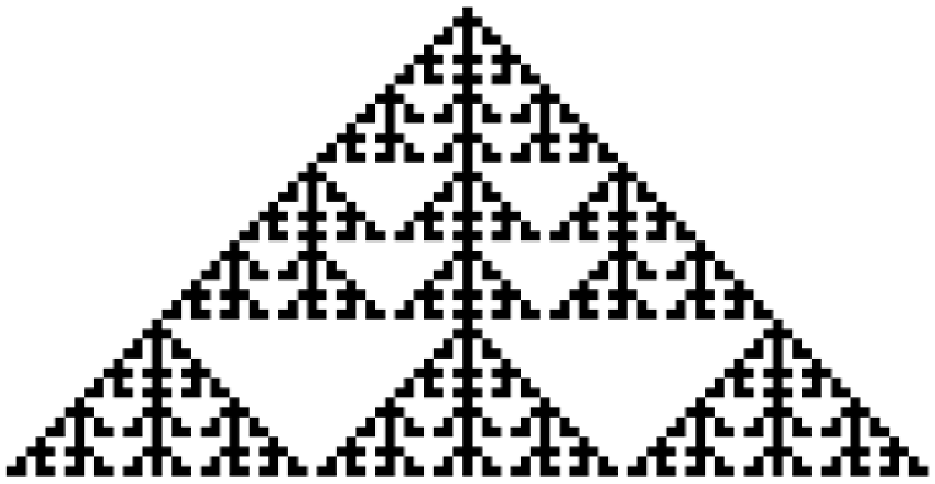
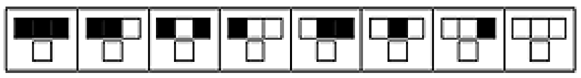
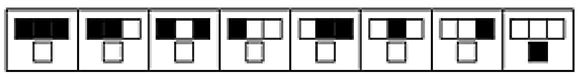


Figure 3.5: The evolution of the rule in the text starting with a single black cell. Each new line downward represents the evolution of the automaton at the next time step.

two possible state outcomes for the middle cell, there are $2^8 = 256$ possible transition rules by which this can be achieved. Wolfram described these as *elementary* cellular automata. By considering a white cell as being in state 0 and a black cell in state 1 we can identify the rule given in the form of 3.2a or 3.3 by its 8 output states (or *rulestring*): 10010110, which in base 2 happens to be the number 150. So this rule is referred to as Rule 150 and is the rule above depicted in four different formats.

In a similar fashion, all 256 elementary cellular automata can be numbered starting from:

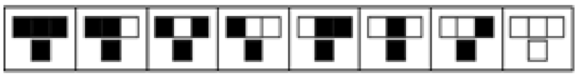

= Rule 0

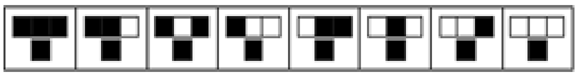

= Rule 1

.

.

.


= Rule 254


= Rule 255

A complete illustration of all 256 elementary cellular automata starting with a standard initial condition of one black cell is given in the Appendix. Many of the salient features found in cellular automata theory can be observed in these elementary ones and we shall refer to them often. These elementary cellular automata are examples of *first order automata* in the sense that the state of a cell at time step $t+1$ only depends on the state of its neighbors at the previous time step t. Whereas in a *second order automaton* a cell's state at time step $t+1$ is rather more demanding and depends on the state of its neighbors at time steps t as well as $t-1$. Unless otherwise specified, we will mainly be considering first order automata although second order automata will come into play in our discussion of reversibility.

Perhaps you are thinking that this is not so much to work with, so if we merely increase the radius to $r = 2$ then there are $2^5 = 32$ possible neighborhood-states and $2^{32} = 4,294,967,296$ possible rules. On the other hand, if we keep $r = 1$ and merely increase the allowable states per cell to 3, then there are $3^3 = 27$ possible neighborhood-states and an astronomical $3^{27} = 7,625,597,484,987$ possible rules! That should be plenty.

3.3 Totalistic rules

This class of rules is defined by taking the local transition function as some function of the *sum* of the values of the neighborhood cell sites. So for example, if $r = 1$, $k = 2$, we have

$$c_i(t+1) = \varphi\left[c_{i-1}(t) + c_i(t) + c_{i+1}(t)\right].$$

An example is the Rule 150 above given by

$$c_i(t+1) = c_{i-1}(t) + c_i(t) + c_{i+1}(t) \mod 2.$$

So how many such totalistic rules are there with $r = 1$, $k = 2$? With the two states 0 and 1 for each of the three cells, there are just the four possible totals of the cell states: $0, 1, 2$, and 3. For example, our Rule 150 can also be depicted by the following transition table:

sum (t)	3	2	1	0
$c_i(t+1)$	1	0	1	0

.

So we can see that in general each of the four sums accruing at time step t can yield at the next time step either a 0 or 1, thus giving $2^4 = 16$ possible totalistic rules, of which the preceding Rule 150 is one.

Breaking away from the elementary cellular automata and allowing three states per cell ($k = 3$) but maintaining nearest neighbors ($r = 1$), let us take cell states: 0 = white, 1 = gray, and 2 = black. Thus there are seven possible sums: $0, 1, 2, 3, 4, 5$, and 6, each of which can yield a $0, 1$ or 2 at the next time step. An example is:

sum (t)	6	5	4	3	2	1	0
$c_i(t+1)$	1	1	0	1	2	1	0

and in fact, there are $3^7 = 2187$ such totalistic rules. In order to systematically enumerate them, the cellular automaton number is the base 3 value of $c_i(t + 1)$ in the second row as in the preceding table. The number given above, 1101210, is the value 1020 in base 3 and so this automaton is labeled Code 1020 and has the appearance in Figure 3.6 starting with a single gray cell.

3.4 Boundary conditions

There is the small matter of what should be the neighborhood of the cells at the extreme left and right of the lattice. In many cases we will take the lattice to be sufficiently large in extent so that these cells do not come into our considerations and the lattice can be considered to be effectively infinite. However, in some cases the lattice will be finite in extent and there are three main types of boundary conditions:

(i) Periodic (also known as ‘wrap’): Here we consider the two cells at the extreme left and right ends to be neighbors of each other, thus making the lattice into a continuous loop in the one-dimensional case. As well, it is also possible to prescribe the array in the vertical direction (in the two-dimensional case) so that the top and bottom cells are also neighbors, thus making the cellular array into the shape of a torus, or donut. Periodic

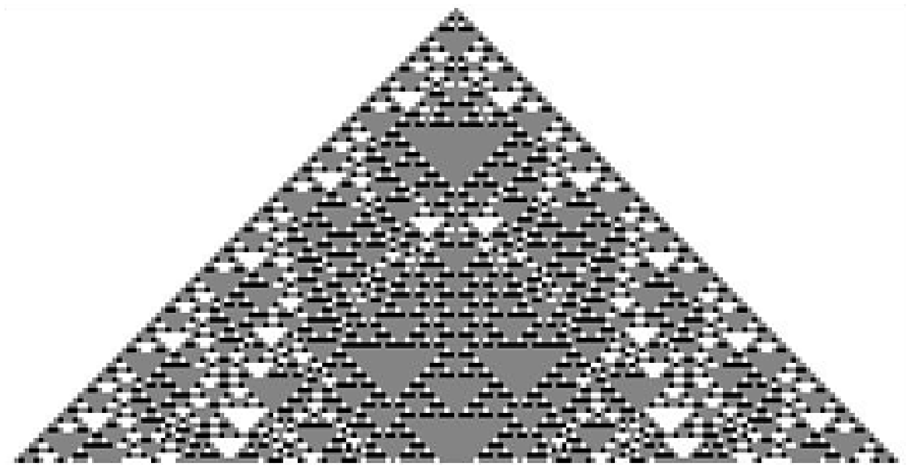


Figure 3.6: Totalistic Code 1020 = 1101210 in base 3, generated from a single gray cell (cell state = 1) with 3 states per cell and nearest neighbors.

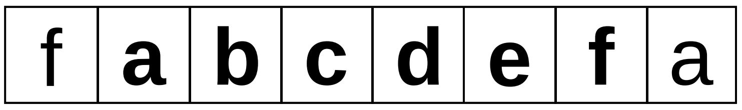


Figure 3.7: For the finite 1-dimensional array of cells labeled: a, b, c, d, e, f, periodic boundary conditions ensure that cell a has the left-hand neighbor f and cell f has the right-hand neighbor a as if the array were joined into a loop.

boundary conditions are the best at simulating an infinite array since no boundary is apparent to any cell.

(ii) Reflective: In this case the cells states at the extremities are repeated in order to provide neighboring cell values for them.

(iii) Fixed: In some instances, we want the boundary values to take on some fixed value such as when modelling a problem in heat conduction (see Section 5.5.5). Thus the values along the boundary remain constant throughout the simulation.

3.5 Some Elementary Cellular Automata

In the sequel we will consider various elementary cellular automata that have an initial condition of a single black (i.e. state 1) cell, all the others being white (state 0). Of course we will not discuss all 256 cases and nor is this necessary, as for example, Rule 0 only produces cells that are in state 0

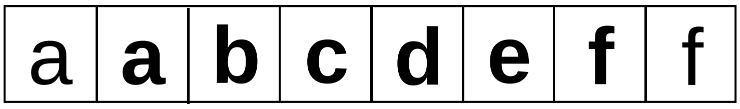


Figure 3.8: With reflective boundary conditions, the same array of cells will have the first and last cells repeated in order to provide neighbors for them.

and is therefore of zero interest.

Rule 1 has the distinguishing feature (noted above) that a neighborhood of cells entirely in state 0 (quiescent) spawns a live cell at the next time step. In some theoretical considerations (see Section 4.1.6) this condition will not be permitted. This yields the following automaton starting from a single black cell:

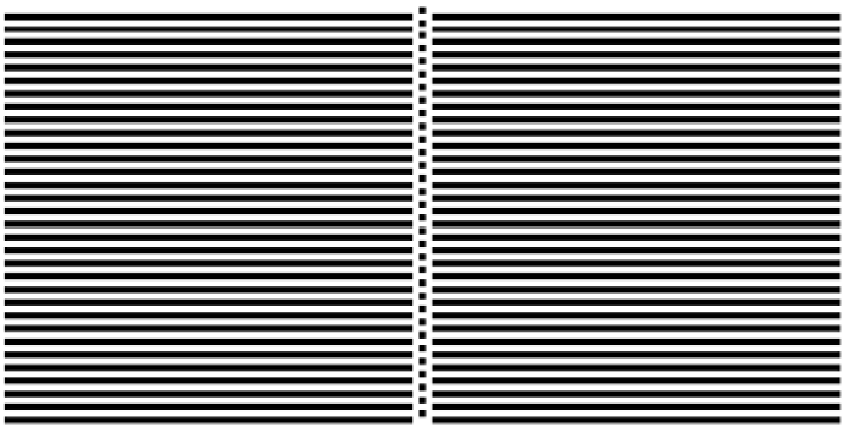


Figure 3.9: Rule 1 starting with a single black cell illustrating its repetitive nature.

In principle, this array must be infinite in extent to the left and right of the starting cell.

Let us skip now to look at Rule 30:

$$c_i(t+1) = [c_{i-1}(t) + c_i(t) + c_{i+1}(t) + c_i(t)c_{i+1}(t)] \mod 2$$

where we are immediately struck by its quite random appearance (Figure 3.10).

Considering the central column of cells, which have states: 1101110011..., Wolfram [1986] applied various statistical tests and found the sequence of 1's and 0's to be completely random. Of course this is the intrinsic randomness as discussed earlier and every time Rule 30 is executed it will produce the

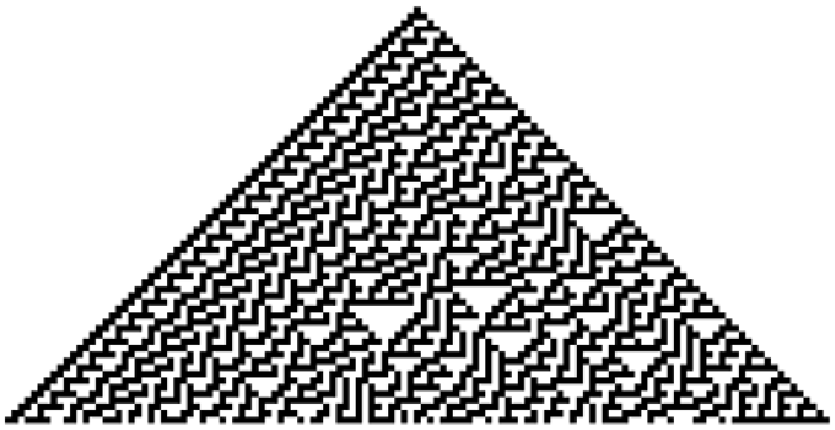


Figure 3.10: Rule 30 depicting a rather random appearance containing triangles of various sizes.

same sequence of digits along the central column. Nevertheless, no pattern whatsoever can be discerned in these digits in the sense mentioned, and so they are considered random.

To consider the randomness of the central column of (in this example taking a sample of 960) cells a bit further, if it is truly random, then one would expect that the number of 0's and 1's in any long sequence to be roughly equal, so that each digit occurs roughly half the time. Likewise, we should find approximately equal numbers of the four pairs of digits: 00, 01, 10, 11, (among the 480 possible pairs of digits) so that the frequency of occurance (called the *block frequency*) of each pair is 1/4, and for the eight triples of digits: 000, 001, ... 111, (among the 320 triples) each should have a block frequency approximately of 1/8, and similarly for the sixteen quadruples: 0000, 0001, ... 1111, (among the 240 quadruples) each should have a block frequency of 1/16. To test this, a sequence of 960 cells in length from the central column was sampled and the block frequency of each block length up to length four was calculated. Here is a table of the counts for each block size:

Block size	**Observed count**							
1	498	462						
2	134	116	114	116				
3	55	38	34	41	41	38	28	45
4	22	16	16	19	9	16	16	15
	19	11	11	13	17	19		

The expected counts E_i for each block size respectively are based on their probability of occurance p_i, with $E_i = p_i N$, and $N = 960$:

Block size	**Expected count**
1	$E_1 = E_2 = 1/2 \times 960 = 480$;
2	$E_1 = E_2 = E_3 = E_4 = 1/4 \times 480 = 120$;
3	$E_1 = E_2 = ... = E_8 = 1/8 \times 320 = 40$;
4	$E_1 = E_2 = ... = E_{16} = 1/16 \times 240 = 15$.

In order to compare the observed counts for each particular block size with the expected counts in each case, we use a statistical tool known as the *chi-squared test.* This is just a general measure of how much the observed values deviate from the expected ones determined by their probabilities, and is given by:

$$\chi^2 = \sum_{i=1}^{k} \frac{(O_i - E_i)^2}{E_i},$$

where O_i represents the *observed* counts for a particular block size, E_i are the *expected* counts and k is the number of different blocks considered in each case. The formula gives the value denoted by χ^2, called *chi-squared.* So for example, if we consider blocks of size 3, then $k = 8$ and each E_i equals 40. Based on the data above we find:

Block size 1: $\chi^2 = 1.35$
Block size 2: $\chi^2 = 2.2$
Block size 3: $\chi^2 = 11$
Block size 4: $\chi^2 = 16$

Next we need to interpret these numbers to see if they mean that our data is actually random. In order to do this, there are standard tables for comparing your chi-squared values with truly random ones. So for example, if we look up the value of $\chi^2 = 1.35$ for our block size = 1, we find that between 20 - 30% of the time a truly random sequence will have an even larger chi-squared value. Since the χ^2 value is a measure of the deviation from the theoretical expected values, this is indeed a good indication of randomness for Rule 30. Taking say, block size 4 and our value of $\chi^2 = 16$, we find from the tables that between 30 - 50% of the time a random sequence will have a larger chi-squared value, again a strong indication of randomness for Rule 30. The other χ^2 values give similar results and we may confidently conclude that the pattern of black and white cells along the central column of Rule 30 is indeed random.

Various other tests for randomness are discussed in the book by Donald Knuth [1981], and again Rule 30 passes all of these tests as well (Wolfram [1986]).

Rule 30 has also been proposed by Wolfram in 1986 as the basis for an encryption system. To implement such a system the first step is to transcribe all of the plaintext message into 8 bit ASCII characters. So for example, the letter 'a' is denoted by 01100001 (=97), 'b' is 01100010 (=98) and 'space' is 00100000 (=32) and so forth. For ease of handling purposes we put this into an $8 \times n$ array where n is the length of the message (including spaces and punctuation). Thus the plaintext message:

Once upon a time...

is put into an array where each row represents a letter with white = 0, black = 1; see Figure 3.11.

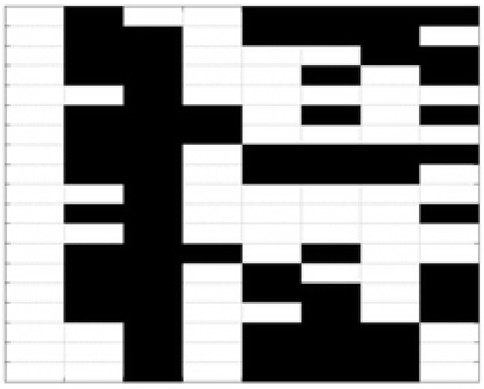


Figure 3.11: The plaintext message, "Once upon a time..." transcribed into 8-bit ASCII with 0 = white and 1 = black.

So the first letter 'O' is 79 = 01001111 appears on the first row and on the fifth row we have a 'space' which is 32 = 00100000 and so forth. Let us denote an abitrary member of this array by a_i. To encrypt this message we take some specific set of random initial values and evolve Rule 30 from them. Now take the cell values of 1's and 0's of the central (or any other) column and denote these values by b_i. (Actually taking every other value from a particular column provides a higher level of security). In Figure 3.12 are the values b_i of Rule 30 derived from the initial condition 1772712395 expressed as a 31-digit binary number taking every second cell value of the 16th column that we will use to encrypt our message.

Then we simply apply the 'EXCLUSIVE OR' (XOR) operation

$$a_i \vee^* b_i = c_i$$

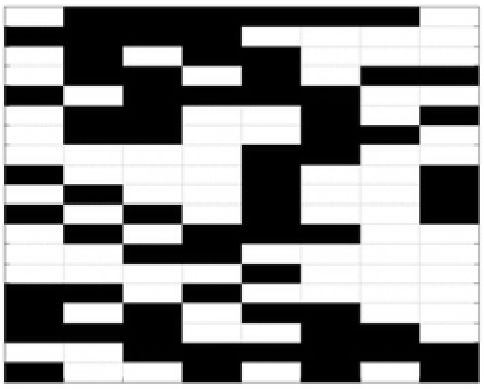


Figure 3.12: The values given by Rule 30 derived from the initial condition 1772712395 expressed as a binary digit number and taking every second number of the 16 column.

taking elements from the plaintext array and Rule 30 array consecutively, with 1 = True, 0 = False. This results in the ciphertext c_i, which we also put into an array format (Figure 3.13).

For example, starting with the top row and taking consecutively the elements of the message and Rule 30, forming $a_i \vee^* b_i = c_i$ we have, $0 \vee^* 0 = 0$, $1 \vee^* 1 = 0$, $0 \vee^* 1 = 1$, etc. The 'exclusive or' is slightly more restrictive than the usual $\vee$ operation and differs from it only in so far as if P and Q both true, then $P \vee^* Q$ is now false. If either just one of P or Q is true, then as for the usual disjunction, $P \vee^* Q$ is likewise true.

The ciphertext message can now be transmitted. In order to decode the message, one simply has to run Rule 30 taking the *same set* of random initial conditions and applying the 'exclusive or' operation consecutively to the ciphertext and the same elements b_i of Rule 30 that were used for the encryption:

$$c_i \vee^* b_i = a_i.$$

The result of this second application of the 'exclusive or' operation returns the original values a_i which is this case translates into: *Once upon a time...*

The reason this works is that two consecutive applications of the 'exclusive or' operation returns the original value. For example, for a value of 1 in the text and a value of 1 from Rule 30, we have $1 \vee^* 1 = 0$ and taking this result again with 1 from Rule 30, we have: $0 \vee^* 1 = 1$, which is the original text value. The other three possibilities are verified similarly.

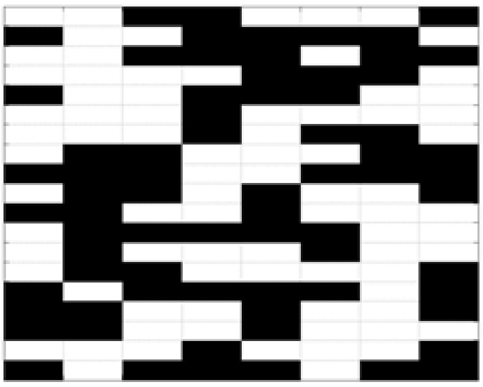


Figure 3.13: The ciphertext of the original message encoded using Rule 30 according to the algorithm explained in the text.

One disadvantage to this method of encipherment is that the random initial conditions for Rule 30 must be communicated to the message recipient in advance. This has been a weakness of secret codes over the centuries that have relied on the 'key' having to be known to both parties to encipher and decipher a message. There is always the danger that the key will be discovered by those for whom the message is not intended. Sometimes this key would change daily and so-called *one time pads* would be used for this purpose. New developments in quantum cryptography are making the distribution of the key absolutely secure (see Bennett, Brassard and Ekert [1992]). Individual photons are sent by fiber optic cable between the two parties and any attempt to eavesdrop on these photons will perturb them in such a way that can be detected by the intended recipients.

Another form of encryption, *public key encryption*, based on the extreme difficulty of factoring the product of two large prime numbers into its constituent parts, is considered uncrackable at the present time. This method was discovered by Clifford Cocks of the British Government Communication Headquarters in 1973 and independently in 1977 by Ron **R**ivest, Adi **S**hamir, and Leonard **A**dleman of MIT and has become known as RSA. In the quaint jargon of cryptography, Alice creates a public encryption key which is just the product of two prime numbers, say $N = p \times q = 10151 \times 14533 = 147524483$, but keeping the values of p and q secret. This allows anyone who knows the key to send Alice a message. For example, if Bob wants to send Alice a message he simply uses a particular mathematical algorithm (called a 'one-way function' – also publically known) that invokes Alice's key $N = 147524483$ to encrypt the plaintext and send it to her. To decipher the message requires knowing the values of p and q so that

Alice is the only one who can do this. The ciphertext message is secure because in practice, the prime numbers that Alice has chosen, p and q, are very large (say 100 digits long!) and once multiplied together, the number N cannot be factored into its constituent parts by even the most powerful computers. It is conceivable that someday there will be found some mathematical algorithm for achieving the factorization of extremely large numbers in a reasonable period of time, but until then the RSA encryption method is totally secure although encrypting large amounts of data with it is cumbersome. It is the de-facto standard for secure transactions on the Internet. A CA scheme for the authentication of digital messages and images has been developed by Mukherjee *et al.* [2002]. A fascinating account of the history of cryptography is Simon Singh's, *The Code Book* [1999].

Several elementary cellular automata (Rules 18, 22, 26,82,146,154, 210, and 218 among others – see Appendix) have the nested triangle appearance of Rule 90 which is given by the formula: $c_i(t+1) = c_{i-1}(t) + c_{i+1}(t) \mod 2$.

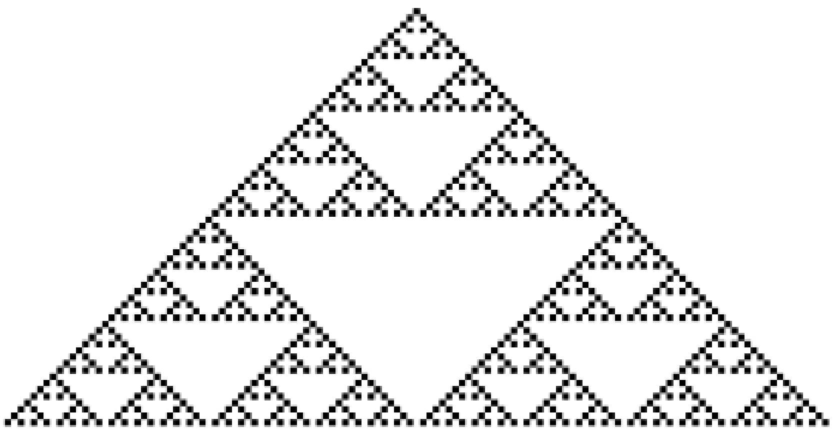


Figure 3.14: Rule 90 after 66 steps of its evolution.

In the limiting case, this automaton depicts the Sierpiński triangle and has fractal dimension 1.585. It has been shown by D.A. Lind [1984] that starting from any initial configuration, the limiting frequency of 1's and 0's is always 1/2 for each. This is an exceptional result because in general it is impossible to say much of anything about the long term evolution of a particular cellular automaton.

Rule 90 also has a connection with Pascal's triangle, a familiar object in elementary mathematics. A consequence of the well-known Binomial Theorem is the expansion:

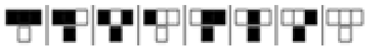
$$(1+x)^n = 1 + nx + \frac{n(n-1)}{1\cdot 2}x^2 + \frac{n(n-1)(n-2)}{1\cdot 2\cdot 3}x^3 + ... + nx^{n-1} + x^n,$$

for positive integer values of n. So for example:

$$(1+x)^5 = 1 + 5x + 10x^2 + 10x^3 + 5x^4 + x^5.$$

Pascal's triangle derives the coefficients of the expansion by starting with a 1 at the apex of the triangle, then the first binomial expansion of $(1+x)^1$ has coefficients $(1,1)$ which form the next row of the triangle, and the coefficients of $(1+x)^2$ which are $(1,2,1)$ are on next row, where the 2 is the sum of the previous two 1's. The next row will have a 1 at either end and between them we add the first 1 and 2 from the previous row to get 3 as the next digit, then add the 2 and 1 from the previous row to get another 3 so that the 3rd row is: $(1,3,3,1)$. In general, we always start and finish a row with a 1 and the digits in between are obtained by adding the consecutive digits of the previous row. This generates a triangle of digits known as Pascal's triangle that gives the binomial coefficients of $(1+x)^n$. If we overlay this triangle of digits with a grid of square cells and color the cells black that have odd numbers in them, then we obtain the pattern of Rule 90.

Rule 110 is quite unremarkable in appearance starting from a single black cell (Figure 3.15).

Its rule structure:  gives nothing away regarding its sophisticated potential, which Wolfram suspected would be nothing less than universal computation. And indeed, during the 1990s his research assistant, Matthew Cook, demonstrated that Rule 110 was capable of universal computation! This was demonstrated by showing that Rule 110 could in fact emulate any of what are called *cyclic tag systems.* Since some of these can emulate any Turing machine and since some Turing machines are capable of universal computation, it logically follows that Rule 110 is likewise capable of universal computation.

Rule 150 has been introduced in Section 3.2 and starting from a single black cell has the appearance depicted in Figure 3.5. Remarkably, in the same way that Rule 90 is related to the coefficients of the binomial expansion, with odd numbered cell entries being black, Rule 150 is related to the coefficients of the trinomial expansion

$$(1+x+x^2)^n.$$

So for example, taking $n = 8$ gives the coefficients: **1**, 8, 36, 112, 266, 504, 784, 1016, **1107**, 1016, 784, 504, 266, 112, 36, 8, **1** with the first, middle

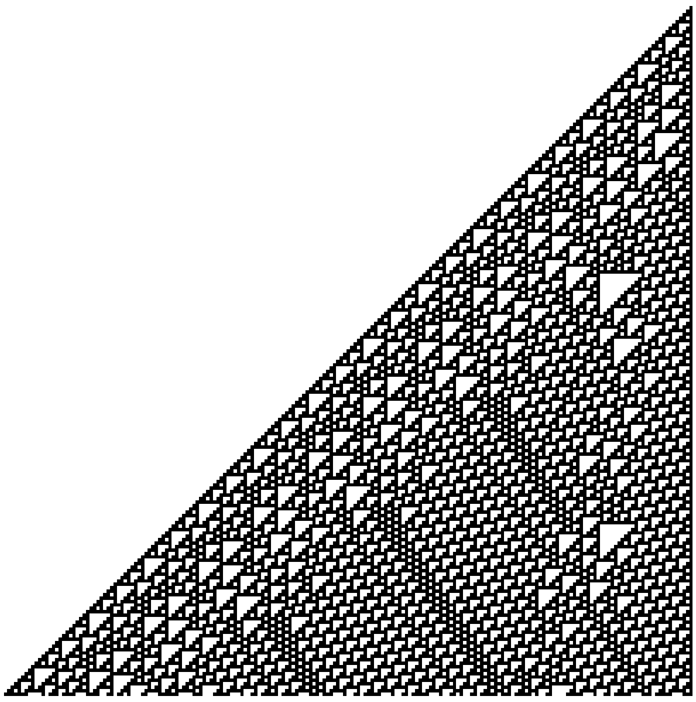


Figure 3.15: The evolution of Rule 110 starting with one black cell.

and last coefficients being the only odd ones. These determine which cells are black in the *9th* row of Rule 150 and all other black cells are obtained in a similar fashion.

3.6 Additivity

Some rules like Rule 90 or Rule 150 have a special property of being what is called 'additive'. Taking another look at Rule 90: $c_i(t+1) = c_{i-1}(t)+c_{i+1}(t) \mod 2$, we find that the value of the central cell at the next time step depends on a sum of the neighboring cells' states. We can extend the sum notion slightly to what mathematicians call a 'linear combination' which for $k = 2$ and $r = 1$ is a sum of integer multiples of the neighborhood states:

$$c_i(t+1) = \alpha c_{i-1}(t) + \beta c_i(t) + \gamma c_{i+1}(t) \mod 2,$$

and the α, β, γ are integer constants 0 or 1. In general, for k states per cell there are 0 to $k-1$ values for the constants and all the arithmetic is taken $\bmod k$. For k states and range r there are k^{2r+1} additive rules or $2^{2\cdot 1+1} = 8$ additive rules among the 256 elementary cellular automata.

These are found by taking all possible choices of 0 and 1 for α, β, γ giving: Rule 0 ($\alpha = \beta = \gamma = 0$), Rule 60 ($c_{i-1} + c_i$), Rule 90 ($c_{i-1} + c_{i+1}$), Rule 102 ($c_i + c_{i+1}$), Rule 150 ($c_{i-1} + c_i + c_{i+1}$), Rule 170 ($c_{i+1}$), Rule 204 ($c_i$), Rule 240 ($c_{i-1}$).

The above notion of additivity can easily be seen by taking Rule 150 for example and looking at its transition function (Figure 3.16) :

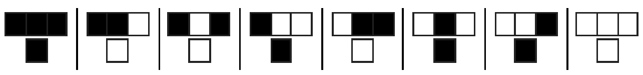


Figure 3.16:

It is clear from the rule that the right-hand neighbor, central cell, and left-hand neighbor state values can be added (mod 2) to give the state value of the central cell at the next time step.

The additive CA also have the advantage that an algebraic analysis can be used to determine certain of their global properties (Martin, Odlyzko & Wolfram [1984]).

3.7 Reversibility

It is thought that the laws of physics are time-reversible, in other words, if the state of a system is known at a given instant of time, it is possible to completely describe not only the future evolution of the system, but its past as well. This is due to the fact that in the equations describing these systems, the time variable t can be replaced by $-t$ and the equations still remain valid. According to Einstein, "... the distinction between past, present, and future is only an illusion, however persistent" (excerpt from a letter to the widow of his friend Michele Besso). This is seemingly at odds with the Second Law of Thermodynamics which describes irreversible processes that proceed with increasing entropy along an arrow of time. This dichotomy – the microscopic laws of physics are reversible, yet the macroscopic world is filled with irreversible phenomena, has exercised the minds of physicists and philosophers at least since the 1860s amid much controversy and confusion. To demonstrate the flavor of the controversy, let me just quote Jean Bricmont [1997], professor of theoretical physics at the University of Louvain, Belgium: "It is perfectly possible to give a natural account of irreversible phenomena on the basis of reversible fundamental laws, and of suitable assumptions about initial conditions. This was essentially done a century ago by Boltzmann, and despite numerous misunderstandings and misguided objections (some of them coming from famous scientists, such as Zermelo or Poincaré), his explanation still holds today. Yet [Ilya] Prigogine writes...

'He (Boltzmann) was forced to conclude that the irreversibility postulated by thermodynamics was incompatible with the reversible laws of dynamics'. This is in rather sharp contrast with Boltzmann's own words: 'From the fact that the differential equations of mechanics are left unchanged by reversing the sign of time without anything else, Herr Ostwald concludes that the mechanical view of the world cannot explain why natural processes run preferentially in a definite direction. But such a view appears to me to overlook that mechanical events are determined not only by differential equations, but also by initial conditions...'

As Bricmont further elaborates, "...irreversibility does not lead to a *contradiction* with the basic physical laws. Indeed, the laws of physics are always of the form given some initial conditions, here is the result after some time. But they never tell us how the world *is* or *evolves*. In order to account for that, one always needs to assume something about the initial conditions." (Incidently, Jean Bricmont is also co-author of *Intellectual Impostures* (published in the USA as *Fashionable Nonsense*), an exposé of pseudo-intellectual, pseudo-scientific discourse that passes for profound philosophical thought).

In fact, there are some reversible cellular automata that do proceed from a disordered state to an ordered one, i.e. from a state of high entropy to a state of low entropy. And of course the key here is to precisely know the correct initial conditions. In Figure 3.17, the automaton proceeds in the first column from a state of randomness to a highly ordered state. How was this highly unusual state of affairs achieved? Namely, by running the inverse of the automaton (starting at the top of column 5) until a final state is reached and capturing the values of this final state for the initial conditions we started with in column 1.

Thus in order to attain reversibility, it all depends on knowledge of the initial conditions of the system. But therein lies the rub when it comes to actual physical systems. As Stephen Wolfram states [2002], "... no reasonable experiment can ever involve setting up the kind of initial conditions that will lead to decreases in randomness, and that therefore all practical experiments will tend to show only increases in randomness.... It is this basic argument that I believe explains the observed validity of the Second Law of Thermodynamics."

By way of contrast, *irreversible* cellular automata may exhibit a decrease in the number of possible configurations over time ("compression of states") and hence a decrease in entropy. This means that the number of reachable configurations diminishes over time and the system tends towards a more ordered state.

For example, let us consider a finite cellular automaton with N cells and two possible states per cell, so that there are 2^N possible configurations. If we consider our automaton to be initially completely disordered then any configuration is equally likely, having a probability of occurence of $1/2^N$.

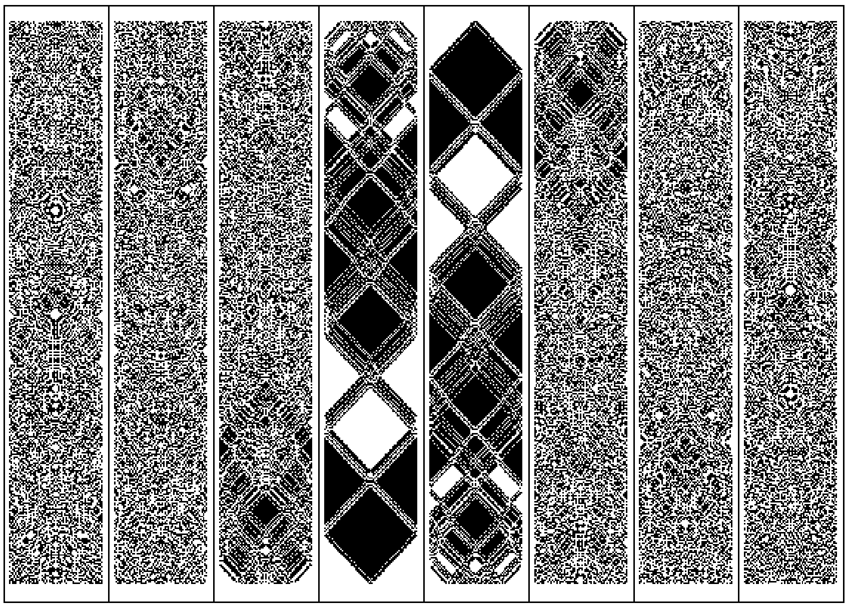


Figure 3.17: This reversible automaton proceeds from a completely disordered state to a highly ordered one, contrary to the Second Law of Thermodynamics. This was achievable by knowing precisely the initial conditions to start with. So the trick is to run the inverse of the rule until the automaton becomes completely disordered and the final state of the system is used for the initial state of the original rule.

Thus the automation is in a state of maximum entropy. Now at the first time step let each one of the 2^N possible initial configurations evolve according to the transition rule and let us record each of the configurations thusly reached. Some of these configurations may now be the same, say one configuration appears m times among these 2^N configurations. Then its probability of occurence at this time step becomes $m/2^N$. At each subsequent time step, we will compute the probability of occurrence in this fashion for each of the descendent configurations generated by the time evolution of the initial ensemble of configurations. Whenever some configurations appear multiple times, others will not appear at all and the probability of occurence of these latter will be zero. We have illustrated a simplified version of this situation in the table below with the label of *all* the possible initial configurations of an automaton along the top row and time being vertical. So 'a' is just the name for a configuration of black and white cells, likewise 'b' etc. If our automaton was just 10 cells wide, there would indeed be $2^{10} = 1024$ such configurations, but the eight listed here will suffice to illustrate our point. Now the local transition function will give rise to the transformation of one configuration into another, say: $a \longrightarrow b$, $b \longrightarrow c$, $c \longrightarrow d$, $d \longrightarrow e$, $e \longrightarrow f$, $f \longrightarrow g$, $g \longrightarrow h$, $h \longrightarrow g$. This automaton is irreversible as configuration g has two predecessors, namely f and h.

	a	b	c	d	e	f	g	h
1	b	c	d	e	f	g	h	g
2	c	d	e	f	g	h	g	h
3	d	e	f	g	h	g	h	g
4	e	f	g	h	g	h	g	h
5	f	g	h	g	h	g	h	g
6	g	h	g	h	g	h	g	h

All the possible initial configurations of a cellular automaton are on the top row and their subsequent evolution after 6 time steps runs vertically. Note that the number of configurations diminishes so that only g and h are final states. At the 1st time step, the probability of occurence of configuration g is $1/4$ and that of h is $1/8$, whereas at the 6th time step, the probability of occurence of configurations g and h is $1/2$.

In this simplified example, we find that as the system evolves a, b, c, d, e, and f, are unreachable configurations. A configuration that is unreachable by any initial configuration of a particular cellular automaton is called a *Garden of Eden* configuration and this topic will be pursued more rigorously in Section 4.1.6

Taking a real example that nicely illustrates this compression of states as the automaton evolves is the graph of the evolution of Rule 126 (Figure 3.18). Considering all the $2^{10} = 1024$ initial configurations that are 10 cells

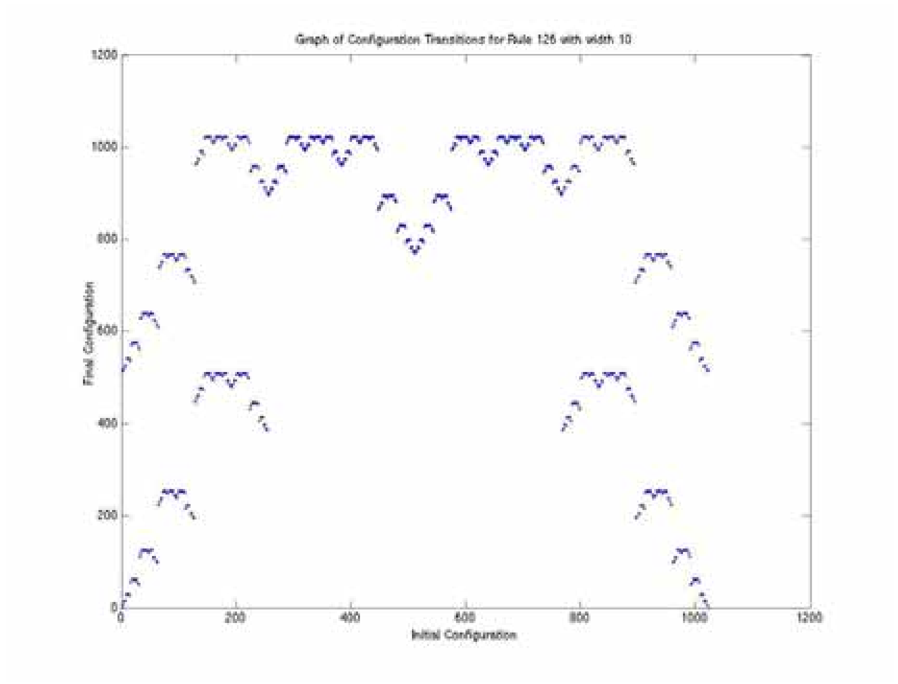


Figure 3.18: The time evolution of Rule 126 for each of the 1024 initial configurations of 10 cells wide. A dot indicates the final configuration and the various plateaus show the favored final states. The graph is also symmetric about the centerline owing to the fact that it has a symmetric rulestring: 126 = 01111110.

wide and letting each evolve until it reaches a final configuration, we find various final states (indicated by a dot) are reached multiple times.

In order to compute the entropy of a finite automaton as above, we use the formulation: $S = \sum_{i=1}^{2^N} P_i \log_2 (1/P_i)$, where P_i is the probability of occurence of the ith configuration at a given time step as described above. If we compute S at each of various time steps, we typically obtain the result illustrated in Figure 3.19) showing a decrease over time from maximum entropy that is indicative of the increasing self-organization of the automaton.

If we apply the same procedure in computing the entropy for Rule 30, which in Section 3.5 indicated complete randomness along its central column, we find something rather interesting. Starting from a random initial configuration, the entropy over time still decreases somewhat, although to a lesser extent than for Rule 126, indicating some degree of overall self-organization (Figure 3.20). This is in spite of the fact that the black and white pattern in any individual column is completely random!

On the other hand, reversibility in cellular automata is very uncommon but is prevalent in a few elementary cellular automata, namely Rules 15, 51,

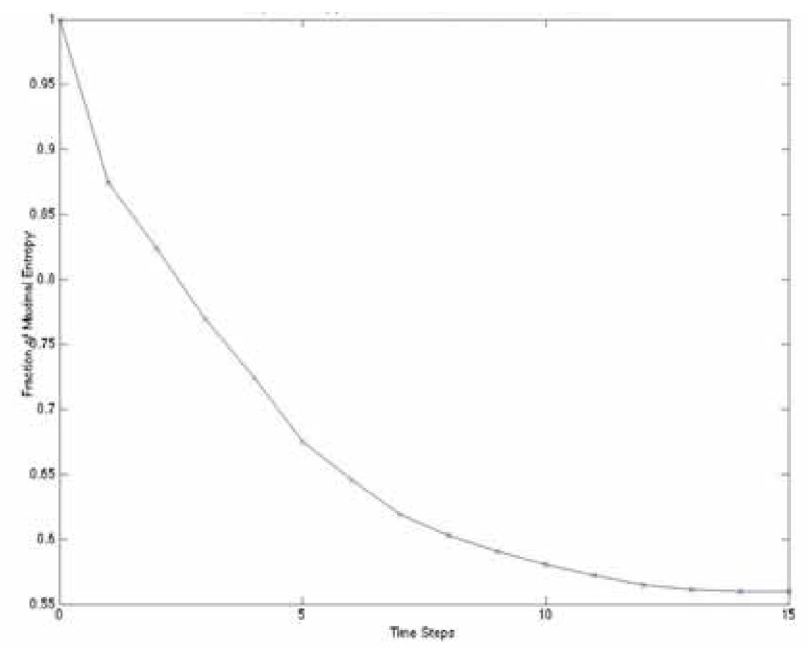
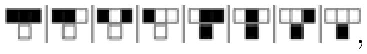
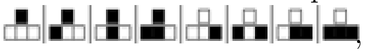
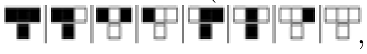
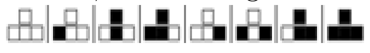


Figure 3.19: Graph showing the entropy as a function of time for the ensemble of 2^{15} initial configurations (cell width = 15) evolving according to (irreversible) Rule 94. The entropy decreases from a maximum at its initial state to a minimum after 14 time steps indicative of irreversibility.

85, 170, 204 and 240. As in the case with Rule 15 with random initial conditions (Figure 3.21)whose transition rule is given by: , one can start on any line and go backwards in time steps using the rule rotated by 180 degrees, that is: , which turns out to be Rule 85 (and hence also reversible). Next, considering Rule 204: , its rotated rule:  is actually itself and the same is true for Rule 51. That just leaves Rule 170 whose inverse is Rule 240.

A mathematical framework for the notion of reversibility is considered later in the sequel concerning the Garden of Eden.

We found that in irreversible automata, the number of configurations could diminish with time ("compression of states"), however, it is a theorem of the great French mathematician, Joseph Liouville, that the total number of configurations in a reversible system remains constant.

There is an interesting way in which to create *reversible* rules due to Edward Fredkin.What one does is to consider the transition function for the particular rule, say it is given by $\varphi\left[c_{i-1}(t), c_i(t), c_{i+1}(t)\right]$, and then add the state of the central cell at the previous time step: $c_i\,(t-1)$. This gives the

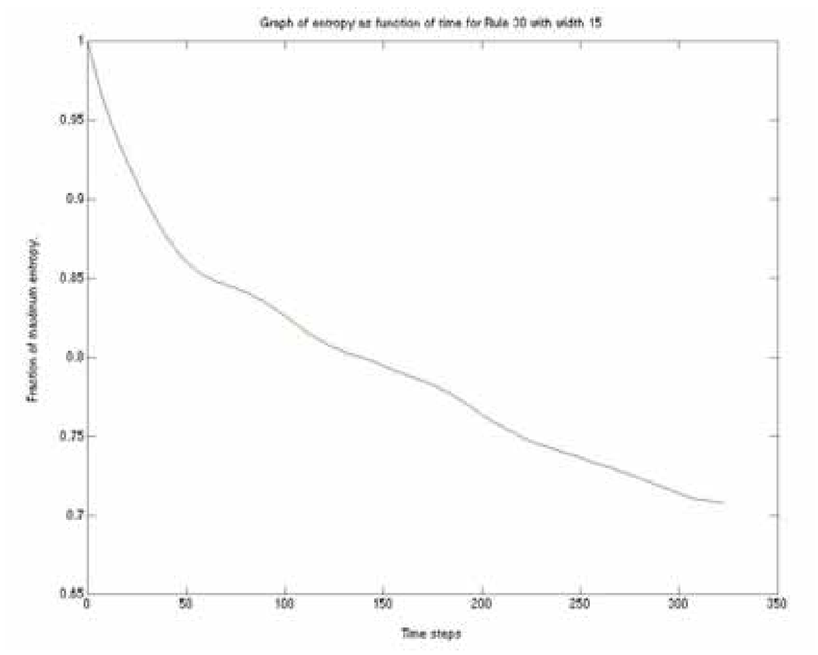


Figure 3.20: Computing the entropy of Rule 30 as it evolves with time we find that it decreases somewhat, indicating a certain degree of overall self-organization even though any individual column is random.

new rule

$$c_i(t+1) = \varphi\left[c_{i-1}(t), c_i(t), c_{i+1}(t)\right] + c_i\left(t-1\right) \mod 2,$$

which has a unique inverse given by

$$c_i(t-1) = \varphi\left[c_{i-1}(t), c_i(t), c_{i+1}(t)\right] + c_i\left(t+1\right) \mod 2.$$

For example, Rule 90 can be described by

$$c_i(t+1) = c_{i-1}(t) + c_{i+1}(t) \mod 2,$$

which normally is not reversible, but it does have a reversible counterpart, Rule 90R, which can be expressed as:

$$c_i(t+1) = c_{i-1}(t) + c_{i+1}(t) + c_i\left(t-1\right) \mod 2.$$

In other words, to determine the value of a cell at the next time step, look to the immediate left of the cell, look to the right, look behind the cell

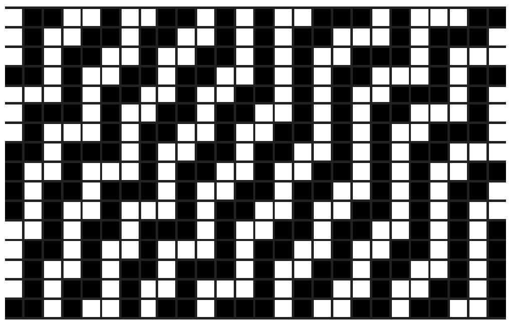


Figure 3.21: The reversible CA evolved using Rule 15 and random initial conditions.

and take the sum of these three cell states and divide by 2. The remainder, either 0 or 1, is the cell's state at the next time step.

The inverse then will have the form:

$$c_i(t-1) = c_{i-1}(t) + c_{i+1}(t) + c_i\,(t+1) \mod 2.$$

The evolution of Rule 90R depicted in Figure 3.22 with periodic boundary conditions can be visually checked and seen to be fully reversible according to the above rules.

The above reversible rules are examples of *second order* cellular automata because a cell's state not only depends on the value of the neighboring cell states at the previous time step, but also at the time step before that. The reversible rule in Figure 3.17 is Rule 94R.

Cellular automata with more states per cell also have some that are reversible, but as in the case of the elementary ones, it is only a small percentage of the total that actually are.

3.8 Classification of Cellular Automata

In the early 80s, Wolfram began to classify cellular automata by the complexity of their evolutionary behavior. Four basic classes were distinguished, although some automata have a mixture of different classes and some fall between two classes. Nevertheless, the vast majority of automata can be classified in this fashion, from elementary automata, to more complex systems. In some respects, this classification parallels the types of behavior exhibited by dynamical systems discussed in the Preliminaries. The same four fundamental classes of behavior are to be found over a broad range of cell state values (k), neighborhood radius (r) and dimension of the cellular automaton.

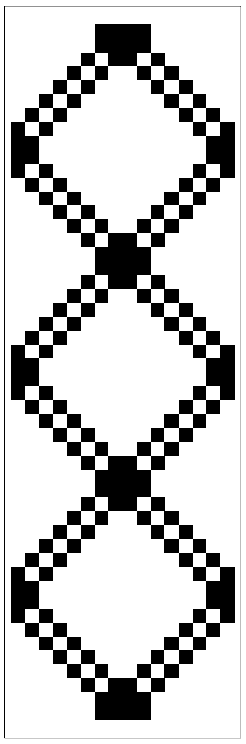


Figure 3.22: This is the reversible Rule 90R with periodic boundary conditions.

Class I: This behavior is the simplest and evolves to a uniformly constant state for all new cells (Figure 3.23).

This class of automaton resembles dynamical systems that tend to a fixed point attractor. In fact, the configuration of all black cells is a fixed point (stable, stationary) attractor in the space of all space of all possible configurations, in that for all initial configurations of the cells, the automaton evolves to this one fixed state. Automata in this class cannot be reversible because the initial information is completely lost.

Class II: In this case, the evolution is towards continually repeating structures or periodic structures (Figure 3.24). Class II automata resemble the periodic (cyclic) behavior exhibited by dynamical systems. They can also act as a kind of filter in that certain sequences of data can be preserved while others are made void. For example, the cellular automaton in Figure 3.25 preferentially selects the sequence 010 (white-black-white) while all other sequences of 1's and 0's are lost, becoming 0. This sort of selection

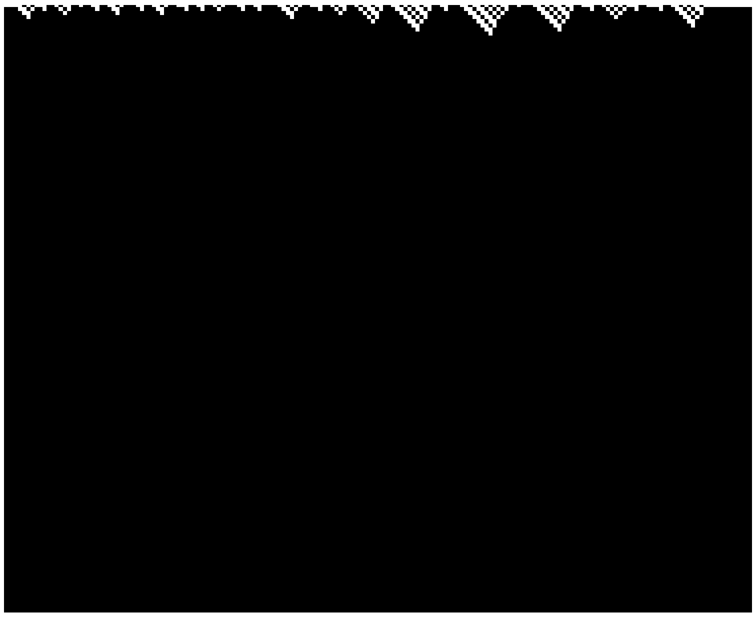


Figure 3.23: This is Rule 249 depicting the evolution to a uniformly constant state from random initial conditions. Any other set of initial conditions would evolve to the same fixed state.

process is useful in digital image processing.

Related to Class II systems are systems of *finite size*. In these systems, the domain of cells is finite in extent which is what we have in all practical cases, and this inevitably leads to periodic behavior. In the example below of Figure 3.26, Rule 150 shows differing periodicities depending on the size of the domain. Periodic boundary conditions are to be used. Because of the finiteness of the domain, the system eventually finds itself in a previous state and thus is forced to repeat itself. The maximum possible period is of course affected by the maximum possible number of states, which in the case of two states per cell, is 2^n for a domain of n cells. Thus the period can in principle be extremely large even for a modest sized domain. Often however, the period is much less than the maximum and its actual value is greatly effected by the domain size.

In a Class II system, as in the preceding example, various parts evolve independently of one another without any long-range communication, thus producing the periodic behavior typical of a system of finite size.

Class III: These automata exhibit random behavior typically with triangular features present (Figure 3.27).

Class III automata are analogous to chaotic dynamical systems. Like the

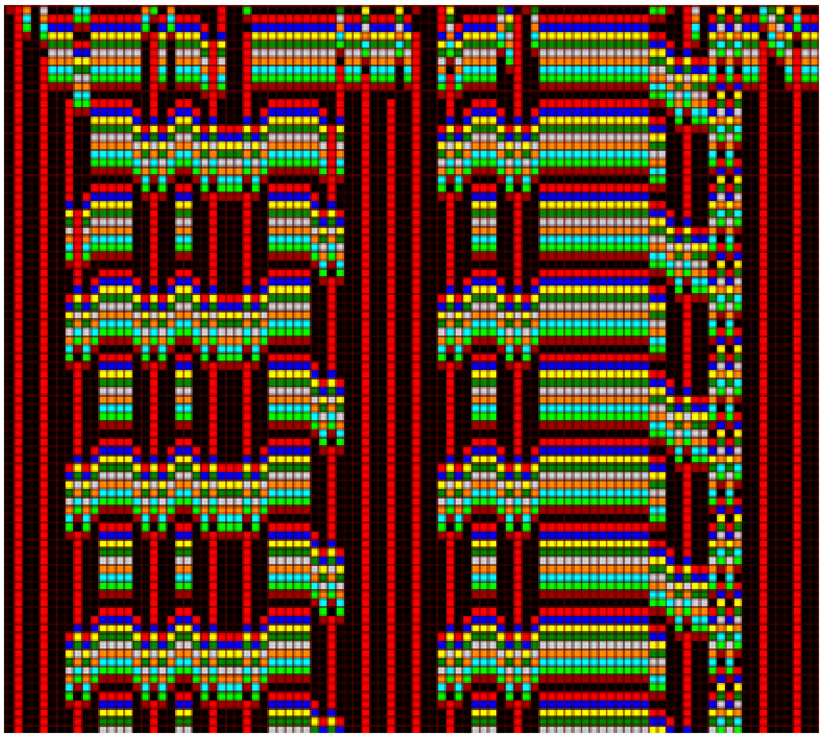


Figure 3.24: This automaton exemplifies the typical behavior of repeating structures found in Class II systems.

latter, they are very sensitive to initial conditions and play an important role in the study of randomness. For example, Figure 3.28 shows the Class III Rule 126 with the value of just one cell changed in the initial conditions.

Class IV: In this, the most interesting of all the Wolfram classes, but not as rigorously defined as the others, localized structures are produced that move about and interact with each other as the automaton evolves (See Figure 3.29).

Although the classes I, II, and III are similar in nature to corresponding types of behavior found in dynamical systems, Class IV behavior has no dynamical system equivalent and in fact fits between Class II and Class III in the sense that it is poised between periodicity and randomness. According to Wolfram [1984], this is where universal computation could be possible – in the netherworld between order and chaos. This region is the so-called "edge of chaos", a term coined by Norman Packard although the idea probably originates with Christopher Langton [1986]. The salient feature of Class IV behavior is that information can be transmitted through time and space opening up the possibility for complex computations to be made.

Of the elementary one-dimensional cellular automata, we have already

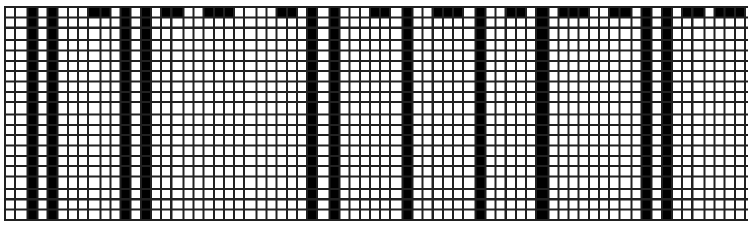


Figure 3.25: This is an example of a the preservation of the data string 010 by elementary Rule 4.

mentioned that Rule 110 is capable of universal computation and is of Class IV. By trivially reversing left and right we obtain Rule 124 which of course is also of Class IV as are the black and white reversals in Rules 137 and 193. But these all effectively have the same dynamics. Among all the other elementary cellular automata, it is possible that Rule 54 is also of Class IV but according to Wolfram this is not entirely conclusive.

3.8.1 Langton's Parameter

Because there is at least a formal similarity between the types of behavior exhibited by dynamical systems and cellular automata, Christopher Langton sought a 'tunable' parameter as we have seen in the analysis of the one-dimensional dynamical system that could be used in some sense to classify automata. However, it must of necessity only be an approximate classification scheme for most questions about the long term evolution of cellular automata are undecidable. For example, the automaton (Code 357, $r = 1$, $k = 3$) of Figure 3.30 evolves to the quiescent state only after 88 steps so that this is a Class I automaton.

However, another automaton that starts out looking similar to this one, goes on indefinitely (Figure 3.31).

If the behavior dies out quickly, then the automaton is clearly seen to be of Class I, but if it does not, it could still be of Class I, simply reaching a constant state after a very long number of time steps. Or it could be of Class II with a repeating pattern of possibly a very high period, or even some other class. There is simply no way to determine *a priori* whether the automaton dies out or not other than letting the system evolve for as long as practicable. This question is essentially the same as the 'halting problem' for Turing machines.

Figure 3.32 is another example where first appearances are deceiving. There are 10 states per cell and there appears to be the kind of complex interactions that one would expect from a Class IV automaton. However, letting the automaton evolve for some considerable period of time, we find that it settles into typical Class II behavior (Figure 3.33).

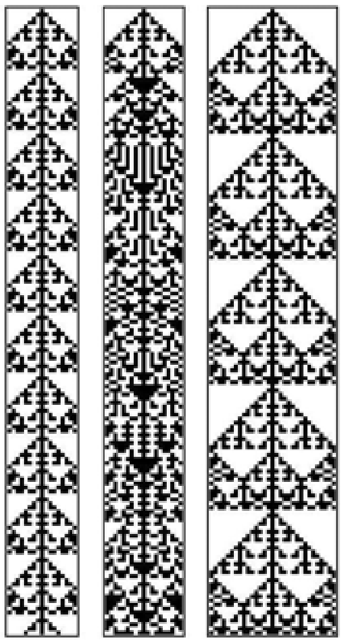
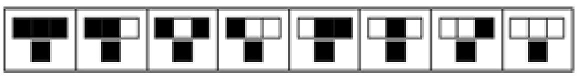


Figure 3.26: Rule 150 in finite domains (with periodic boundaries) of widths 17, 19, and 31 with periods 15, 510, and 31 respectively.

Related to this are cellular automata that Wolfram refers to as 'computationally irreducible'. There are some physical systems for which it is possible to find a formula or algorithm that will give the state of the system at some time t so that it is not actually necessary to let the system evolve until time t in order to see this. Certainly Rule 255:



is an obvious case in point. If there is to be a formula or algorithm defining the state of a system at a given time t, then it will have to involve less computation than the explicit time evolution of the system itself, and hence must be 'more sophisticated'. But as we will see below, there are some cellular automata (even one-dimensional) that are capable of universal computation and thus able to perform the most sophisticated computations that can be done. The upshot is that there will be some automata for which a description of their long term behavior is not possible and the only way to find out is to let them evolve. This can even be the case for Class III automata that are not capable of universal computation such as Rule 30 but whose output is nevertheless random (in the sense we discussed). Thus questions about such automata as to whether or not they evolve to a quiescent state

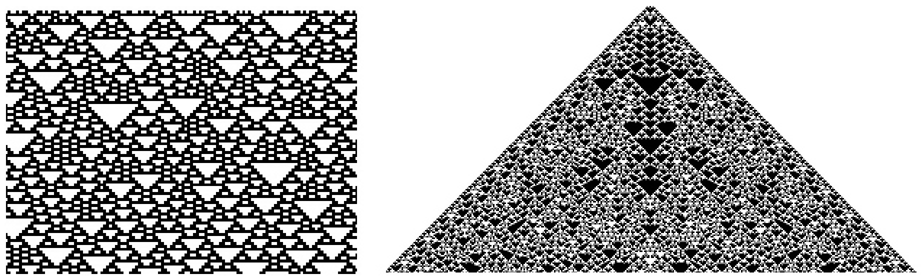


Figure 3.27: This is Rule 126 (left) and Code 2013 showing the typical random behavior found in Class III cellular automata.

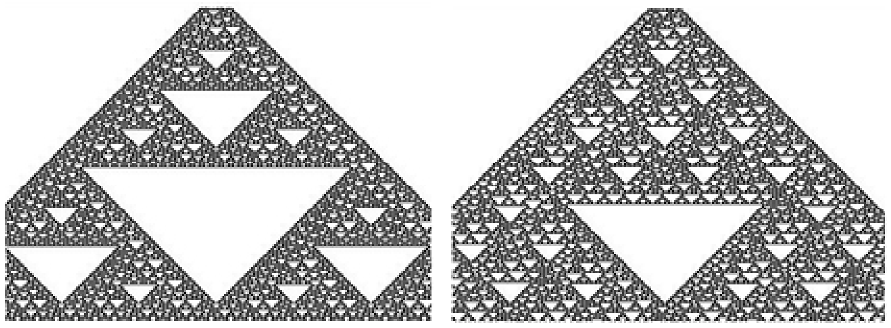


Figure 3.28: The evolution of Rule 126 from initial conditions differing by only one cell's state. This is typical of Class III behavior and chaotic dynamical systems.

are formally undecidable.

In spite of the foregoing difficulties with classification of cellular automata there is still some quantitative measure that we can associate with the four different Wolfram classes. Note that for one particular cellular automaton with k states per cell and a neighborhood radius of r, we found (Section 3.2) there are $n = k^{2r+1}$ different neighborhood-states. For instance, when $k = 2$ and $r = 1$, we have $n = 8$ neighborhood-states making up the transition function. In general, the number of neighborhood-states is given by $n = k^{\rho}$, where ρ is the number of neighbors. If we denote the number of quiescent states of the central cell resulting from these neighborhood-states by n_q, then the Langton parameter is given by:

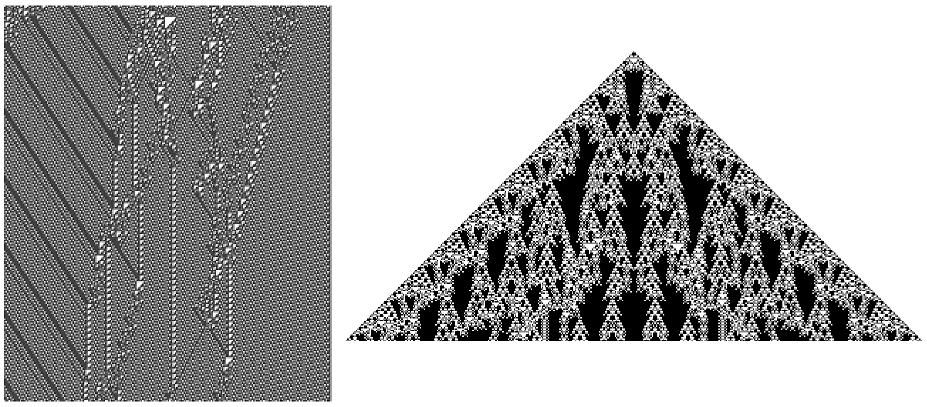
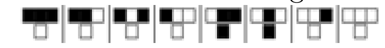


Figure 3.29: Complex interactions between localized structures in Rule 110 and Code 1635, typical of Class IV behavior.

$$\lambda = \frac{k^\rho - n_q}{k^\rho},$$

which simply represents the fraction of output states that are active (non-quiescent) in the particular transition function defining the automaton. When $n_q = k^\rho$, then all the output states are quiescent and $\lambda = 0$. When $n_q = 0$, there are no quiescent output states and $\lambda = 1$. When all the states are equally represented in the output, we have $\lambda = 1 - \frac{1}{k}$ so that the domain of interest is $0 \le \lambda \le 1 - \frac{1}{k}$. When $k = 2$, the roles of the quiescent and non-quiescent cells tend to reverse themselves for high values of λ.

As an example, Rule 12:  has $\lambda = \frac{8-6}{8} = 0.25$, which is just the fraction of 1's in the output, and indeed it exhibits the simple periodic behavior of Class II (Figure 3.34).

In Langton's words, the λ value of a transition function "... is an aggregrate statistic that is *correlated* with, but not a certain predictor of a certain level of behavioral complexity." It reflects average behavior of a class of rules. Bearing this in mind, we find a correlation with the four classes of cellular automata as indicated in Figure 3.35.

Here again we observe that Class IV is associated with Class II behavior and precedes Class III. In Class IV, there are periodic structures as in Class II, but unlike Class II where these structures do not interact, in Class IV automata some of these structures interact in complex ways. Because of these complex interactions, Class IV behavior is also on the verge of the random behavior exhibited in Class III automata. This suggests (on average) the following transition of class behavorial regimes with increasing λ from 0 to $1 - 1/k$:

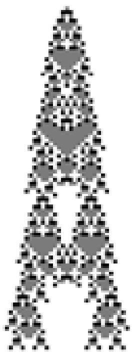
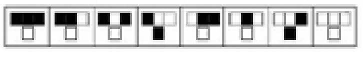
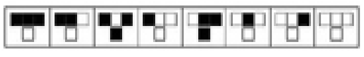


Figure 3.30: Code 357, $r = 1$, $k = 3$, evolves to a quiescent state after 88 time steps and is thus a Class I automaton.

$$\text{Fixed Point} \Longrightarrow \text{Periodic} \Longrightarrow \text{Complex} \Longrightarrow \text{Chaotic}$$

This dynamical discrimination works reasonably well for 'large' values of k and ρ, say $k \geq 4$, $\rho \geq 5$ but less so for the simplest 1-dimensional case of $k = 2$. For example, taking $\rho = 3$, the dynamics are "only roughly correlated" with the Langton parameter. Rule 18:  as well as Rule 40:  both have $\lambda = 0.25$, yet Rule 18 is Class III (Figure 3.36 left) and Rule 40 (Figure 3.36 right) is Class I and we have already seen that Rule 12 ($\lambda = 0.25$) exhibits Class II behavior. Another approach to distinguish the four Wolfram classes, based on what are called *mean field theory curves*, has been taken by H.V. McIntosh [1990].

3.9 Universal Computation

It is a remarkable fact that one-dimensional cellular automata are capable of universal computation. As mentioned in the Preliminaries, this is essentially a consequence of the fact that the logic gates, NOT, AND, and OR can be emulated coupled with memory storage and retrieval. These computer functions were demonstrated by Wolfram [*2002,* p.662] with the composition of each logic gate given in Figures 3.37, 3.38, and 3.39. There are five cell states (denoted by five colors – white, light gray, medium gray, dark gray = 0, black = 1) and the transition rule, without going into the precise details is the same for each of the logic gates.

In addition to the logic gates themselves, Wolfram demonstrated how information could be stored and retrieved in random-access memory (Figure 3.40).

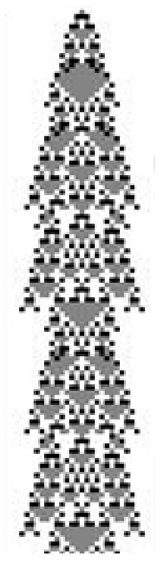


Figure 3.31: Code 600 ($r = 1$, $k = 3$) that does continue indefinitely as it has a repeating pattern.

It is Chris Langton's contention ([1990]), based on empirical experiments, that rules capable of complex computations and indeed universal computation are likely to be found for λ values near the transition between order and chaos. This is analogous to Wolfram's contention that Class IV automata are capable of universal computation. The idea being that complex computations require information transmission and storage coupled with interaction between them. The likely place to find such capabilities, according to Langton, is not in the ordered or chaotic regimes but rather in the "phase transition" between them where there are long transients and complex, unpredictable behavior.

3.10 Density Problem

A prototype of many problems in the CA field is that of the so-called density classification problem. In the one-dimensional two-state version it is desired that the evolved cellular automaton becomes either all 1's or all 0's depending on whether the initial configuration was more than half 1's or more than half 0's respectively. The evolution of such a CA algorithm is thus a means to 'classify' the binary string of 0's and 1's according to their frequency (density). The problem is actually difficult for a cellular automaton because the global property of discerning the higher frequency of either the 0's or 1's must be carried out by local interactions only. In fact, it was shown by Land & Belew [1995] that the density problem cannot be solved

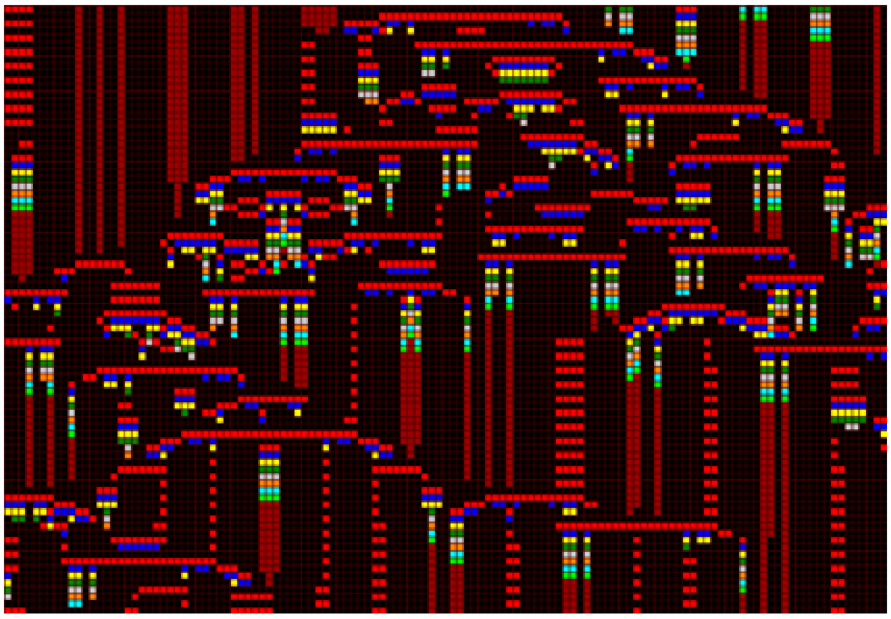


Figure 3.32: This cellular automaton apparently looks like Class IV but subsequently evolves into Class II.

perfectly by any two-state CA having radius $r \geq 1$.

However, all is not lost. If the output conditions of the problem are relaxed slightly then in fact there is a perfect solution proved by Capcarrere, Sipper, & Tomassini [1996]. The humble elementary Rule 184 given by:

$$c_i(t+1) = \begin{cases} c_{i-1}(t) \text{ if } c_i(t) = 0 \\ c_{i+1}(t) \text{ if } c_i(t) = 1, \end{cases}$$

is able to classify any binary string of length N in the following sense. If the density of 1's (resp. 0's) is > 0.5, then the automaton evolves to a fixed state with a block (or blocks) of at least two cells in state 1 (resp. state 0) within $[N/2]$ time steps with the rest of the output being alternating 0's and 1's. (Here the brackets mean to round down to the nearest integer). Even in the case when the density of 1's and 0's is exactly 0.5, the automaton evolves towards a state of solely alternating 1's and 0's, thus perfectly classifiying the density of the initial binary string. The same holds true for Rule 226 which is just the reflected version of Rule 184.

Since the density problem is not perfectly solvable in its initial formulation, attempts have been made to classify the density of a binary string 'most' of the time. One of the best algorithms is the Gaks-Kurdyumov-Levin (GKL) rule which is nearly always successful with binary strings having high

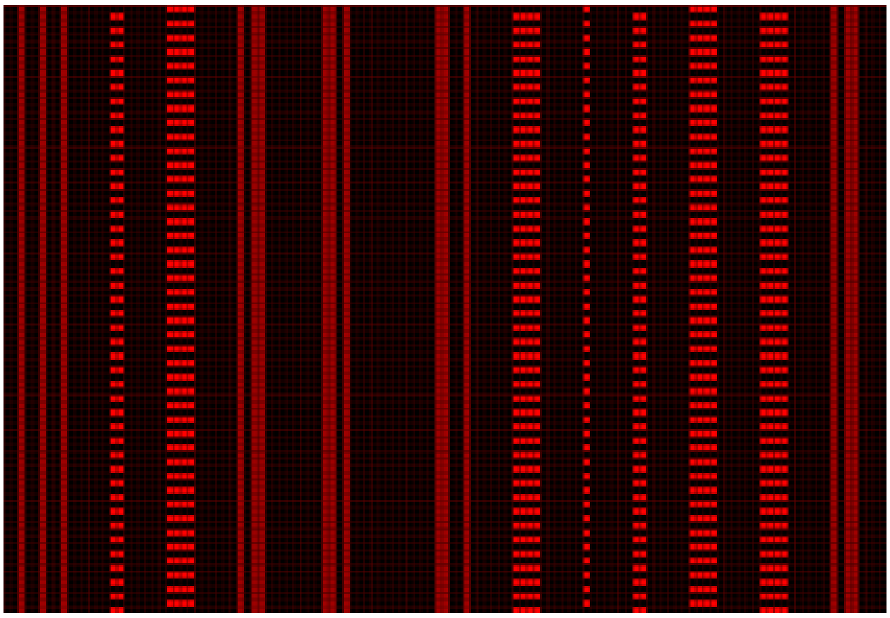


Figure 3.33: The evolution of the cellular automaton in the preceding figure.

or low initial densities. The GKL rule is a wonderful piece of ingenuity that is a two-state, $r = 3$ rule defined by:

$$\begin{aligned} c_i(t+1) &= majority\{c_i(t), c_{i-1}(t), c_{i-3}(t)\} && \text{if } c_i(t) = 0 \\ c_i(t+1) &= majority\{c_i(t), c_{i+1}(t), c_{i+3}(t)\} && \text{if } c_i(t) = 1, \end{aligned}$$

where the term 'majority' means that if two out of the three cells are in state 0 (resp. 1), then that value is taken for the central cell update at the next time step.

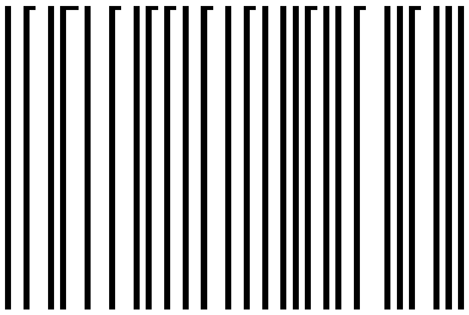


Figure 3.34: Rule 12 displaying Class II behavior with $\lambda = 0.25$.

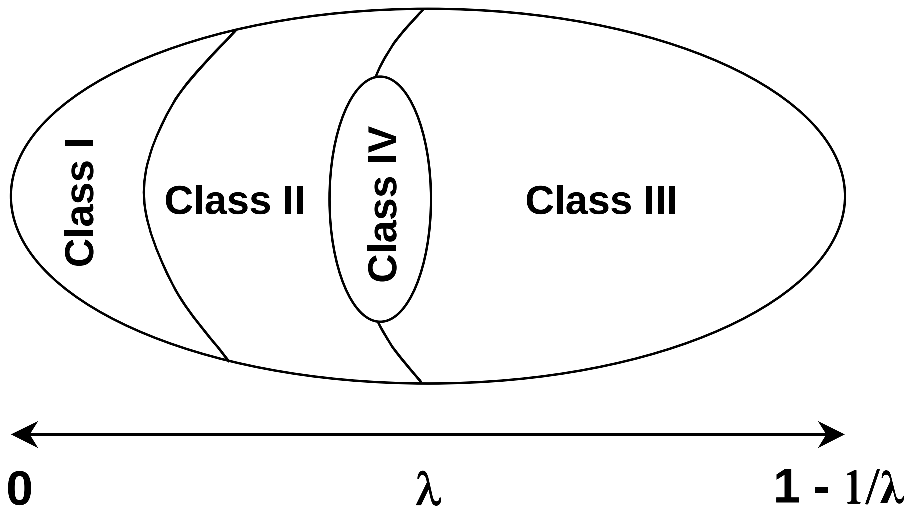


Figure 3.35: The association of the Langton parameter λ with the different Wolfram classes. Note that a specific value of λ can be associated with more than one class. Adapted from W. Flake, *The Computational Beauty of Nature*.

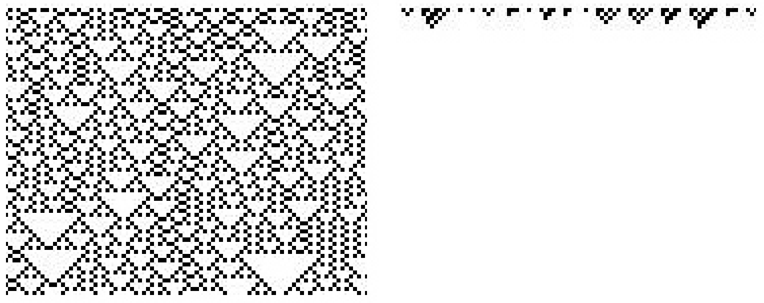


Figure 3.36: The evolution of Class III Rule 18 (left) and Class I Rule 40 (right). Both rules have Langton parameter $\lambda = 0.25$.

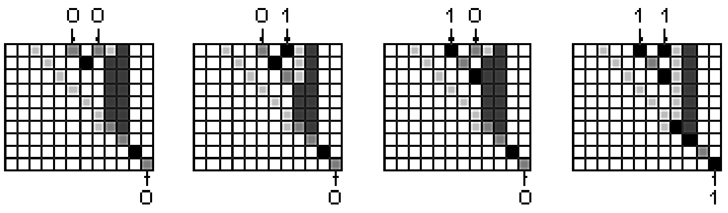


Figure 3.37: The logic AND gate where the input is fed into the first row of the automaton with specific initial conditions and the desired output of 1 at the bottom right only in the case when both inputs P and Q are 1.

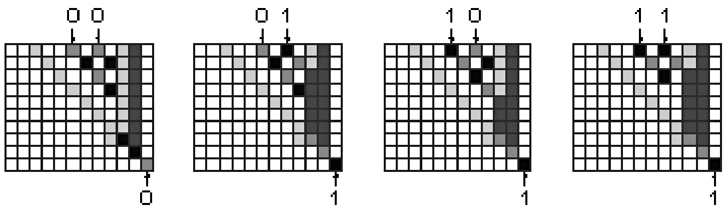


Figure 3.38: The logic OR gate which is a slight variation of the AND gate above giving the output 1 in all but the first case when inputs P and Q are both 0.

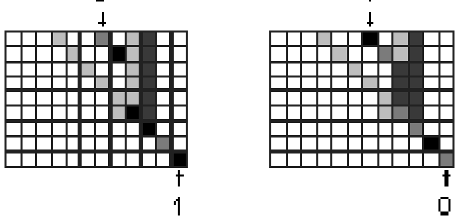


Figure 3.39: The NOT gate turning an input of 0 into 1 and 1 into 0.

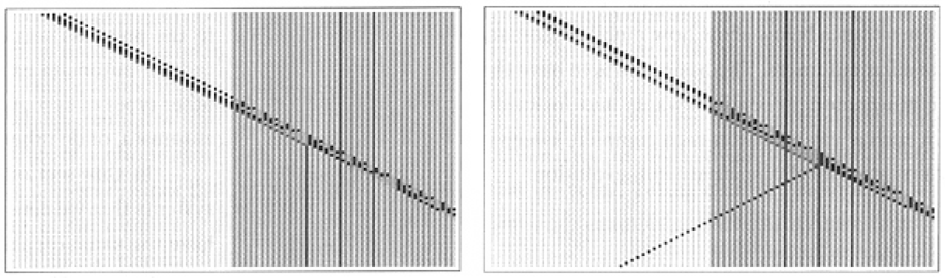


Figure 3.40: A one-dimensional cellular automaton model to illustrate the storage and retrieval of information. The memory is on the right in each figure and in the first figure a 1 is stored in the memory position 13. In the second figure a 1 is retrieved from the memory position 19. From Wolfram [2002], p. 663.

Roughly speaking, densities of local regions are classified and these regions expand with time. Where there are regions of equal density of 0's and 1's, a checkerboard or white/black boundary signal is propagated indicating that the classification be carried out on a wider scale. As effective as the GKL rule is with initial conditions away from the critical density of 0.5, it was found that for densities very near the critical value, the GKL rule failed up to 30% of the time (Mitchell, Hraber, and Crutchfield [1993]).

Norman Packard [1988] experimented with genetic algorithms (see Section 5.4.1) to solve the density classification problem and came to the seemingly reasonable conclusion that when CA are evolved (as in GAs) to perform complex computations, the evolution is driven to rules near the transition to chaos. Similar experiments conducted by Mitchell *et al.* also applied genetic algorithms to the density problem to produce very successful evolved CAs, although as it happens, none quite as successful as GKL. The authors results however were “strikingly different” from those reported by Packard. They found that the most successful evolved rules for solving the density problem were found close to $\lambda = 0.5$ which is the theoretical value they also deduced, and not near the transition to chaos. Packard's results were seen as possible “artifacts of mechanisms in the particular GA that was used rather than a result of any computational advantage conferred” by the transitional regions. Interestingly, Rule 184 also has $\lambda = 0.5$.

Again using GAs, an asynchronous approach to the problem was studied by Tomassini & Venzi [2002], although their results showed that the performance of asynchronous algorithms were inferior to the best evolved synchronous CAs or the GKL rule although they were more robust in the presence of noise.

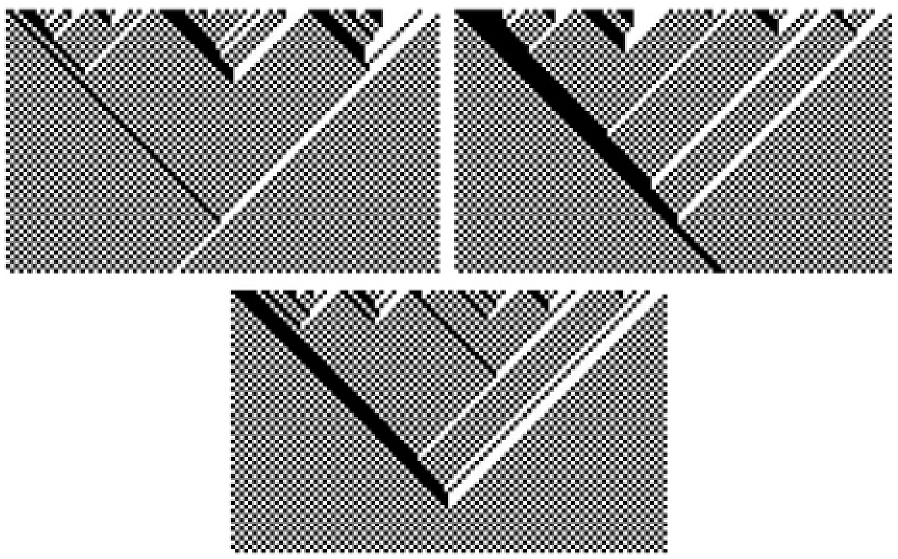


Figure 3.41: Rule 184 classifying densities in the sense mentioned in the text of 51 white/49 black (left), 51 black/49 white (right) and 50 white/50 black (bottom).

3.11 Synchronization

An interesting problem, of which there are now many variations, and first posed by John Myhill in 1957, has to do with the synchronization of the output of an arbitrarily long but finite row of CA. At some time step, all the cells should, for the first time, attain the same state value. Like the density problem, this is difficult for CA since a particular global state must be reached solely by local interactions. This has become known as the 'firing squad problem' or 'firing squad synchronization problem' (FSSP). A cell at one end of the row (a 'general') is distinguished from the others (the 'soldiers'). Taking a 3-cell neighborhood, the problem is to choose the states for each cell and suitable transition functions such that the soldier cells will all be in the same (firing) state for the first time at the same time step. McCarthy and Minsky (see Minsky [1967]) showed that there exist solutions for n cells in $3n-1$, $\frac{5}{2}n-1$,... time steps. A 'minimal time solution' is one achieved in $2n-1$ time steps, which is the number of time steps it takes for a general to send a signal to the far-end member and receive a reply. Waksman [1966] gave a minimal time solution involving 16 states per cell, while Balzer [1967] gave one with 8 states per cell and showed that there was no minimal time solution with only 4 states per cell.

The best minimal time solution to date has been demonstrated by Jacques

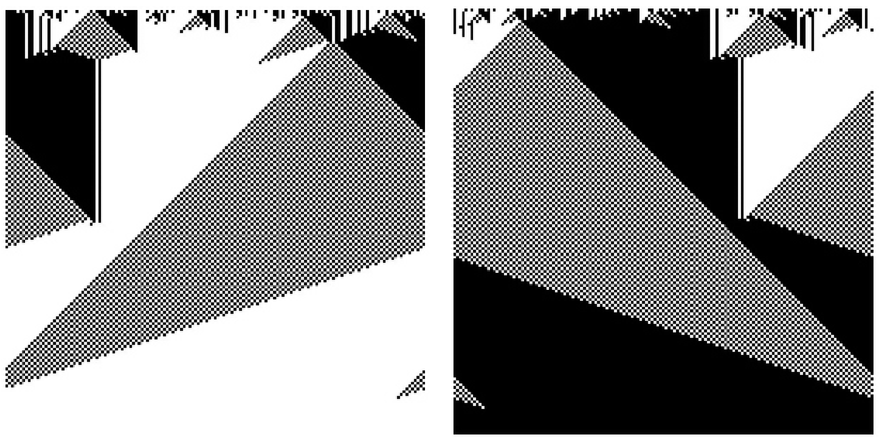


Figure 3.42: The GKL rule classifying initial densities of 77 white/72 black cells (left) and 72 black/77 white (right).

Mazoyer [1987] in a *tour de force* of technical analysis using 6 states. The question is still open whether 6 or 5 states per cell is the least number required. Two generals can also be taken, one at each end who do not reach the firing state (see examples in Wolfram [2002], p.1035).

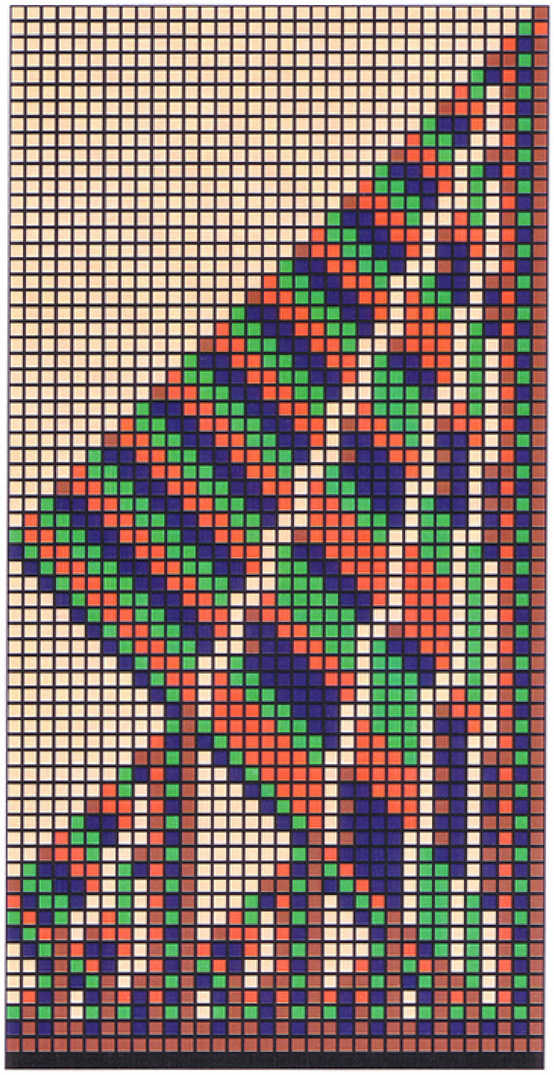


Figure 3.43: The minimal time solution of Jacques Mazoyer of the firing squad synchronization problem using 6 states. Here $n = 34$ and time increases from top to bottom. The general is on the right-hand side and also reaches the firing state (black) together with all the soldiers in 67 time steps. Courtesy Jacques Mazoyer.