Computer Science

Kerstin Andersson

Cellular Automata

Bachelor's Project

2003:26

Cellular Automata

Kerstin Andersson

This report is submitted in partial fulfillment of the requirements for the Bachelor's degree in Computer Science. All material in this report which is not my own work has been identified and no material is included for which a degree has previously been conferred.

Kerstin Andersson

Approved, 2003-10-17

Advisor: Reine Lundin

Examiner: Stefan Lindskog

Abstract

Properties and behavior of cellular automata are considered. Cellular automata can simply be described as lattices of cells, where the cells can be in a finite number of states. By using simple rules the states of the cells are updated at discrete time steps. The evolution of cellular automata can be used for computations. Some cellular automata display universality meaning that there is no limit to the sophistication of the computations they can perform.

Contents

List of Figures

List of Tables

1 Introduction

The purpose of this treatise is to give a short account of cellular automata. Cellular automata are very simple constructions with their lattices of cells which can be in a finite number of states. By using simple rules the states of the cells are updated in parallel at discrete time steps. Despite their simplicity they do not seem to have appeared before about the 1950s. In Section 2 a brief historical background and a formal definition of cellular automata are given.

The writing of this treatise has been greatly inspired by Stephen Wolfram's book "A New Kind of Science" [1], in which cellular automata play a vital role. Cellular automata posess a large number of various kinds of properties. In Sections 2, 4, and 5 some of the properties are discussed. The properties discussed in Section 4 are such that also appear in the physical world, such as randomness, entropy, and conservation. Cellular automata provide good examples which is illustrated in that section.

Cellular automata are sometimes referred to as simple programs and in Section 3 other simple programs like Turing machines [2] are discussed. Cellular automata can be considered as constructions in the field of computer science and in Section 5 some properties related to this field, like universality, are discussed.

Finally, in Section 6 a short summary of the present treatise will be given together with implications for further studies in this field.

2 Cellular Automata

In this section the definition of cellular automata will be given together with a discussion of some of their properties and a brief historical background.

2.1 Historical background

Despite their simplicity, cellular automata are not old constructions. It seems that general cellular automata have not appeared before about the 1950s. Of course, a variety of precursers can be identified. Operations on sequences of digits in doing arithmetic has been used since antiquity. Finite difference approximations to differential equations emerged in the early 1900s. And Turing machines [2] invented in 1936 were based on operations on sequences of discrete elements.

How cellular automata were introduced and how they got their name was through the work by John von Neumann in trying to develop an abstract model of self-reproduction in biology. He constructed a cellular automaton with a two-dimensional lattice, 29 states for each cell, and complicated rules. To give a mathematical proof of the possibility of self-reproduction, von Neumann outlined the construction of a 200 000 cell configuration which would reproduce itself. Seeing the complexity of actual biological organisms, von Neumann appears to have believed that a high level of complexity would be necessary for a system to exhibit sophisticated capabilities such as self-reproduction. In the 1960s constructions were found for simpler cellular automata capable of self-reproduction.

By the end of the 1950s it was realized that cellular automata could be viewed as parallel computers [2] and in the 1960s a sequence of theorems were proved about their formal computational capabilities (see further Section 5).

Without going into any details, it can be mentioned that cellular automata have entered a vast number of areas dealing with, for example, electronic devices, special-purpose computers, image processing, electronic miniaturization, cryptography (see Section 5.1), and neural networks. Despite all these different directions, research on systems equivalent to cellular automata had largely come to an end by the late 1970s. Ironically this happened around the time when computers were first becoming widely available for exploratory work. But the field of cellular automata got a revival in 1981 when Stephen Wolfram started his work in it, by defining the field in a new way.

2.2 Definition

Several informal definitions of cellular automata can be found and the following is one such example [3] which gives a fair description of what a cellular automaton is. As the name suggests a cellular automaton is a discrete model with interactions that are uniform in structure. Cellular automata are characterized by the following fundamental properties:

- They consist of a regular discrete lattice of cells.
- The evolution takes place in discrete time steps.
- Each cell is characterized by a state taken from a finite set of states.
- Each cell evolves according to the same rule which depends only on the state of the cell and a finite number of neighboring cells.
- The neighborhood of nearby cells is defined in the same way for each cell, i.e., the neighborhood relation is local and uniform.

Along with this informal definition a formal definition [3] will be given:

Definition 2.1 *Let*

- *L be a regular lattice (the elements of L are called cells),*
- *S be a finite set of states,*
- *N be a finite set (of size $n = |N|$) of neighborhood indices such that $\forall c \in N$, $\forall r \in L$: $r + c \in L$,*
- *$f : S^n \to S$ be a transition function.*

Then the 4-tuple (L,S,N,f) is called a cellular automaton.

In practical applications the lattice is usually restricted to a finite size and then one has to consider what the neighborhood indices should be close to the boundary of the lattice. This will be exemplified in the next subsection. In the meantime the following sets will be introduced:

- $\forall r \in L$, *N(r) is a finite set (of size* $n = |N|$*) of neighborhood indices of* r *such that* $\forall c \in N(r)$*:* $c \in L$.

A configuration $C_t : L \to S$ is a function that associates a state with each cell of the lattice. The effect of the transition function f is to change the configuration C_t into a new configuration C_{t+1} according to

$$C_{t+1}(r) = f(C_t(i_1) \times \cdots \times C_t(i_n)), \tag{2.1}$$

where $i_1, \ldots, i_n \in N(r)$. Observe that the order of i_j in the expression above is of vital importance for the outcome.

2.3 Elementary cellular automata

In this subsection a set of simple cellular automata, in the following called elementary cellular automata, will be defined. The lattice L is a one-dimensional finite lattice illustrated in the following way:

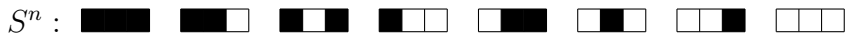
L : 

The finite set of states $S = \{0, 1\} = \{white, black\}$. That is each cell in L can be in either of two states, namely 0 or 1. In the pictures below white and black will be used for denoting the states.

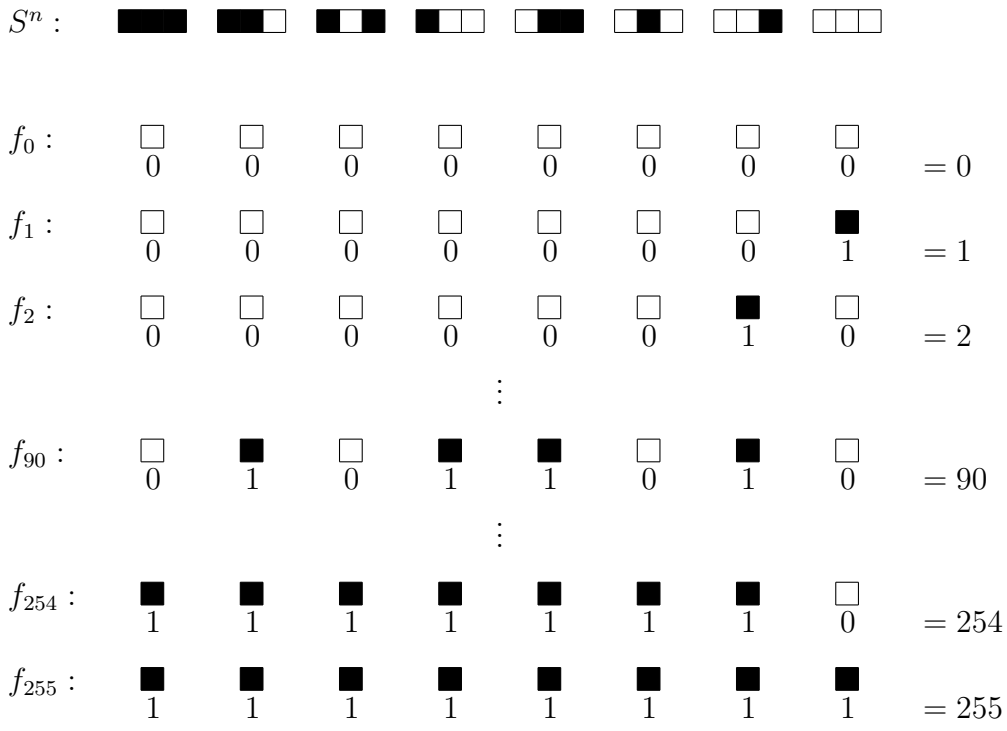
The finite set of neighborhood indices will be of size $n = 3$ and for most cells $N = \{-1, 0, 1\}$. The boundary cells are special cases and here the last cell, the first cell, and the second cell are considered neighborhood indices of the first cell. Similarly, the next

last cell, the last cell, and the first cell are considered neighborhood indices of the last cell. What the procedure above does is to connect the first and last cells of the lattice, so that instead of a finite string of cells a circular lattice is obtained.

The final item to discuss is the transition function f. Since the size of the set of neighborhood indices is $n = 3$ and the size of the set of states S is 2 then the size of the definition set S^n is $2^3 = 8$. The definition set of the transition function can be illustrated in the following way:



Since the value set of the transition function f is S, the size of the value set is 2 and therefore the number of transition functions that can be defined for the given definition and value sets is $2^8 = 256$. So for the given L, S and N 256 elementary cellular automata can be defined and the transition functions for each of these are denoted f_i, where $i = 0, \ldots, 255$. The transition functions can be illustrated in the following way:



From the picture above it is hopefully clear how the 256 elementary cellular automata are defined. The cellular automaton with transition function f_i will in the following be denoted i, i.e., the elementary cellular automaton with rule number i.

The workings of a cellular automaton is best illustrated by an example. Let L be a lattice of size $2T+1$, where T is an integer larger or equal to zero. For starting the evolution of the cellular automaton an initial configuration $C_0 : L \rightarrow S$ has to be chosen. Let us make the middle cell black and all other cells white. Other choices will be demonstrated in Section 2.4. Given the initial configuration the configurations C_t, $t > 0$, can be easily calculated using Equation 2.1. Let us demonstrate the evolution for the elementary cellular automaton with rule number 90. The transition function for this cellular automaton is given in the picture above. In the pictures below the value $T = 10$ has been selected. The initial configuration can be illustrated in the following way:

C_0 : 

For the next configuration C_1 the transition function f_{90} has to be considered. Since most cells are white, most cells will remain white according to this transition function. The middle black cell has two white neighbors and according to the transition function the middle cell will turn white. The cell to the left of the middle cell is white with a white neighbor to the left and a black neighbor to the right. This cell will turn black according to the transition function. Similarly, the cell to the right of the middle cell is white with a black neighbor to the left and a white neighbor to the right. This cell will also turn black. The result is demonstrated in the following picture:

C_1 : 

This process can be continued forever and below a few more configurations are given:

C_2 : 

C_3 : 

The evolution (C_0, C_1, ...) of an elementary cellular automaton can be summarized as in Figure 2.1. Here the top row is the initial configuration.

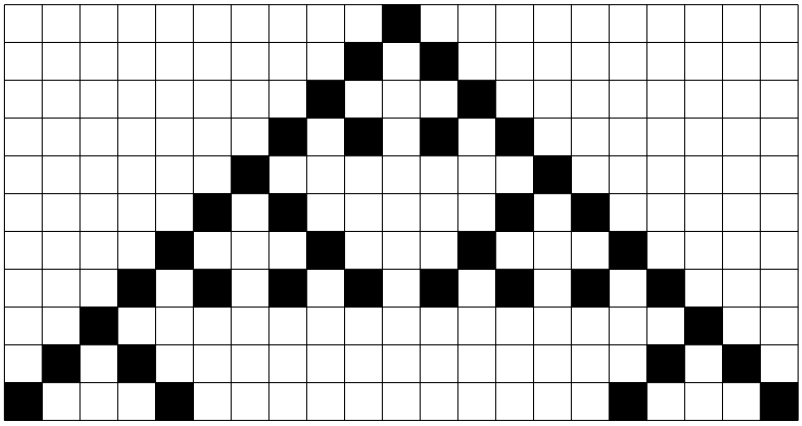


Figure 2.1: *Elementary cellular automaton with rule number 90.*

The LaTeX-code for the picture in Figure 2.1 was generated using the C++-code in Appendix A. Here $T + 1$, i.e., the number of rows (configurations) that should be printed, is given as input. The size of the lattice L is then calculated by the program as $2T + 1$. Further, the rule number of the cellular automaton is also given as input, as well as the size (in tenths of millimeters) of the cells. In addition, one can choose whether a mesh should be printed or not. In Appendix B all elementary cellular automata with $T = 25$ and initial configuration consisting of a single black cell in the middle are collected. In Section 2.4 other initial configurations will be discussed.

The elementary cellular automata in Appendix B will now be discussed. After a brief look in Appendix B it will be realized that the cellular automata can be divided into three groups. These can be denoted uniform/repetitive, nested, and random. Most of the elementary cellular automata fall into the first category, namely 219 out of 256. In the simplest cases in this category all cells end up having the same color after just one step. Thus, for example, in cellular automata 0, 8, and 72 all cells become white, while in 159, 183, and 255 all cells become black. In other cases, as in 7, 23, and 127, all cells alternate between black and white on successive steps. In this first category there are also some

more advanced behavior when a pattern consisting of one or several cells persists, either stationary as in 4, 29, and 91, or non-stationary as in 3, 6 and 103 where the pattern is moving either left or right. In most cases the patterns remain of a fixed size, while in other cases, like in 50 and 109, the patterns grow forever. Owing to the way the elementary cellular automata are defined the maximum speed of moving patterns is one cell per step. Likewise, the maximum growth of a pattern is one cell per step to the left and one cell per step to the right. This is the reason why configurations, C_t, with $t > T$ are not included in Appendix B. The patterns will not pass the borders of the lattice or grow beyond the borders.

In the second category, consisting of nested (fractal) structures, there are in all 24 elementary cellular automata. Since they are relatively few in number they will be listed here: 18, 22, 26, 60, 82, 90, 102, 105, 126, 129, 146, 150, 153, 154, 161, 165, 167, 169, 181, 182, 195, 210, 218, and 225. Most of them are easily identified as nested, but 105, 169, and 225 are somewhat more difficult to classify. By increasing the number of steps and the size of the lattice, as in Figure 2.2, the nested structure will appear more clearly. Since 169 and 225 are mirror images of each other only 169 is shown in Figure 2.2.

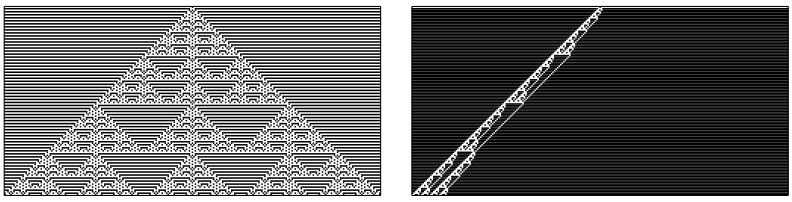


Figure 2.2: *Elementary cellular automata with rule numbers 105 and 169.*

In the third and last category, consisting of random structures, there are in total 13 elementary cellular automata. Wolfram claims there should be 14, but after inspecting Appendix B several times only 13 could be identified. Also in this category there are few members and they will be listed here: 30, 45, 73, 75, 86, 89, 101, 110, 124, 135, 137, 149, and 193. There are four basic forms and they are illustrated in Figures 2.3 and 2.4.

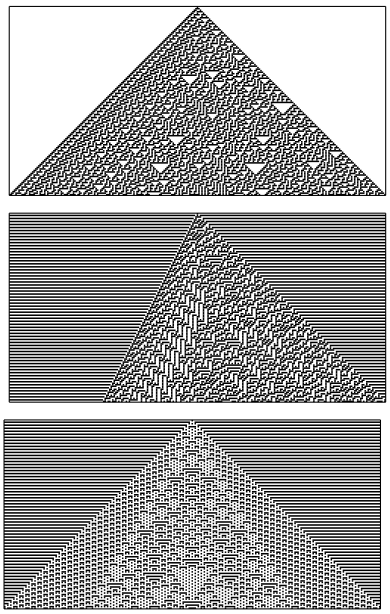


Figure 2.3: *Elementary cellular automata with rule numbers 30, 45, and 73.*

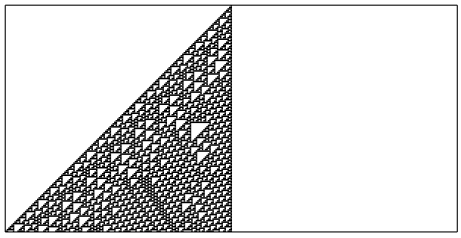


Figure 2.4: *Elementary cellular automaton with rule number 110.*

The four basic forms in the third category appear because of two reasons. Firstly, the initial configuration is symmetric, with respect to left and right, and therefore the non-symmetric elementary cellular automata will appear in pairs, being the mirror images of each other. By studying the definition of the transition functions for the elementary cellular automata it is realized that by interchanging the 2nd and 5th and the 4th and 7th components, respectively, in the binary representation of the rule numbers one will obtain a mirror image of a cellular automaton. This applies to all 256 elementary cellular automata.

Table 2.1: *The elementary cellular automata obtained by interchange of left and right (l/r) and/or interchange of black and white (b/w) for the cellular automata in the random and nested categories.*

Category	Rule number	l/r	b/w	l/r, b/w
Random	30	86	135	149
	45	101	75	89
	73	73	109	109
	110	124	137	193
Nested	18	18	183	183
	22	22	151	151
	26	82	167	181
	60	102	195	153
	90	90	165	165
	105	105	105	105
	126	126	129	129
	146	146	182	182
	150	150	150	150
	154	210	166	180
	161	161	122	122
	169	225	106	120
	218	218	164	164

For example, the binary form of 30 is 00011110. By interchanging the components as described above one obtains 01010110, which in decimal notation is 86. As can be seen in Appendix B the elementary cellular automata with rule numbers 30 and 86 indeed are the mirror images of each other. Likewise, 45 and 101, and 110 and 124, respectively, are the mirror images of each other. The cellular automaton with rule number 73 is already symmetric. The mirror-image pairs of the random and nested elementary cellular automata are collected in Table 2.1.

Secondly, by interchanging black and white it will also be possible, in certain cases, to group the cellular automata in pairs. The reason it is not always possible is that the initial configuration will remain the same, i.e., no interchange of black and white in this configuration. By reading the binary representation of the rule number of a transition function from right to left and by interchanging 0 and 1 will produce the desired result. For example, the binary form of 30 is 00011110. The procedure described above will give the result 10000111, which in decimal notation is 135. By inspecting Appendix B the similarity between the elementary cellular automata with rule numbers 30 and 135

can be confirmed. Likewise, 86 and 149, 45 and 75, 101 and 89, 110 and 137, and 124 and 193, respectively, are similar. 73 and 109 should also be similar, however, the given initial configuration makes 109 repetitive instead. The black-white-interchange pairs of the random and nested elementary cellular automata are collected in Table 2.1.

Another issue that has to be discussed is the division of the third, random, category into two subcategories. Three of the four basic forms form one subcategory (see Figure 2.3) and the remaining basic form makes up the other subcategory (see Figure 2.4). The reason for this division can be seen by studying Figures 2.3 and 2.4. The elementary cellular automata in Figure 2.3 contain parts that are regular and distinctive parts that are random. The cellular automaton in Figure 2.4, on the other hand, contains localized random structures on a regular background. The difference between the two subcategories would have been more clear if more configurations had been printed in Figure 2.4. The LaTeX-code for producing this manuscript does not allow that, however, in Ref. [1] (pages 33–38) the elementary cellular automaton in Figure 2.4 with 3200 configurations is printed. Figure 2.6, in the next subsection, also gives a clearer illustration of the behavior of this cellular automaton.

2.4 Random initial conditions

The investigation of the elementary cellular automata will now continue by studying other initial configurations than the simple one, consisting of a single black cell, that was discussed in the previous subsection. Here random initial configurations, in which every cell is chosen to be black or white at random, will be investigated. The C++-code in Appendix A is easily modified for dealing with random initial configurations. By changing the function `init_array` in Appendix A to

```
// Initializes an array A of size s with random values.
// Precondition: s > 0.
void init_array(bool * A, int s)
{
```

```
    int r;

    srand(1);

    for (int i = 0; i < s; i++) {
        r = (int)((2.0 * rand()) / (RAND_MAX + 1.0));
        A[i] = (bool)(r);
    }
}
```

the modification is achieved. The library `stdlib.h`, which contains `rand()` and `RAND_MAX`, also has to be included. With random initial configurations the number of cells in the lattice and the number of configurations printed are no longer related and this modification can also easily be introduced into the program. In Appendix C all 256 elementary cellular automata with the same random initial configuration are displayed.

With random initial configurations one might suspect that no order will occur. But in fact, as is illustrated in Appendix C, many of the elementary cellular automata end up producing behavior that is not at all random. In the most simple cases, like 0, 32, 168, and 251, the cellular automata quickly organize themselves to become either uniformly white or uniformly black. 24 out of 256 elementary cellular automata display such a behavior. These cellular automata will make up the first class (class 1) in Wolfram's classification of cellular automata.

A second class of structures consists of those cellular automata that end up in stable states which involve a collection of definite structures that either remain fixed on successive steps, as in 4, 72, 92, amd 219, or repeat periodically, as in 2, 23, 91, and 125. These cellular automata belong to class 2 according to Wolfram's terminology. Actually, almost all elementary cellular automata not in class 1 and not included in Table 2.1 belong to class 2. By inspecting Appendix C this can be confirmed easily, except in a couple of cases. By using the fact that cellular automata with rules that are symmetric, with respect to either left and right or black and white or both, belong to the same class, simplified the confirmation in certain cases. For other cases, like 41 and 62, the periodicity appeared first

after printing out some more configurations. Finally, only rule 54 and the related rule 147 (through interchange of black and white) remain as cases that neither seem to belong to class 1 nor to class 2. In Figure 2.5 the elementary cellular automaton with rule number 54 is given with 150 cells, 300 configurations, and a random initial configuration.

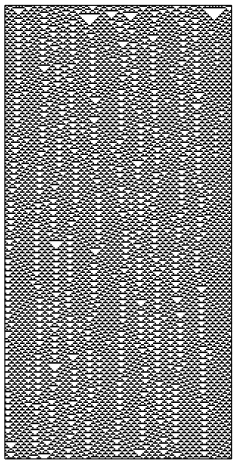


Figure 2.5: *Elementary cellular automaton with rule number 54.*

The remaining elementary cellular automata to be discussed are those collected in Table 2.1. Those on the same row are related and should show the same behavior. This is indeed confirmed in Appendix C. First it is noticed that the cellular automata with rule numbers 26, 154, and 218 and those related to them belong to class 2. The characteristics of the remaining cellular automata in Table 2.1 are that they never seem to settle down. This is quite clear in cases like 18, 30, 60, and 169. For them Wolfram introduces a new class of uniformly random structures which he denotes class 3. The members of this class display a behavior that appears in many respects completely random. But a certain degree of organization is indicated by the triangles and other small structures that are dotted around the pictures. Almost all cellular automata in Table 2.1, except those belonging to class 2, belongs to class 3. However, there are some cellular automata that differ from the rest. These are cases 73 and 110, and the cellular automata related to them. First the cellular automaton with rule number 110 will be discussed. This cellular automaton

is given in Figure 2.6 with a lattice of 150 cells, with 300 configurations, and with a random initial configuration. This cellular automaton does not seem to belong to either class 1, class 2, or class 3. The cellular automaton quickly organizes itself into a set of definite localized structures on a uniform background. These structures do not remain fixed but move around and interact with each other in complicated ways. For this cellular automaton Wolfram introduces a new class, class 4, which contains structures that mix order and randomness. In a sense the members of class 4 display the greatest complexity since they neither stabilize completely nor exhibit uniform random behavior.

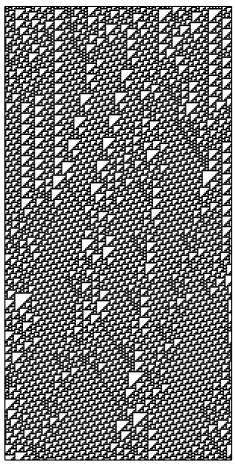


Figure 2.6: *Elementary cellular automaton with rule number 110.*

Wolfram originally discovered the four classes discussed previously in the early 1980's by looking at thousands of pictures similar to those on the last few pages and in Appendix C. At first Wolfram based his classification purely on the general visual appearance of the patterns he saw. But when studying more detailed properties of cellular automata he found that most of these properties were closely correlated with the four classes.

But with almost any general classification scheme there are borderline cases which can be assigned to different classes depending on the definition. And so it is with cellular automata. In Figure 2.7 the elementary cellular automaton with rule number 73 is given with a lattice of 150 cells, with 300 configurations, and with a random initial configuration.

This cellular automaton displays a behavior that can either be classified as belonging to class 2 (the stripes) or to class 3 (the random pattern between the stripes). Wolfram claims that such rules are quite unusual, and that in most cases the behavior falls squarely into one of the four classes previously described.

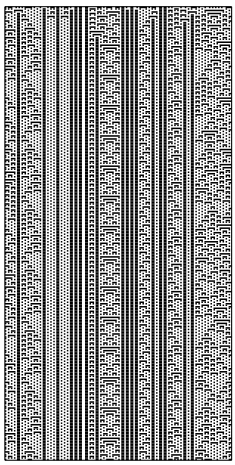


Figure 2.7: *Elementary cellular automaton with rule number 73.*

The analysis in this subsection is based on a visual investigation of the configurations produced by the 256 elementary cellular automata using the same random initial condition. Of course, no guarantees can be made that the analysis will remain the same using other random initial conditions. In fact special initial conditions can make a cellular automaton belonging to, for example, class 3 yield a typical class 2 behavior.

The discussion will now return to the elementary cellular automaton with rule number 54 (see Figure 2.5). This has not yet been classified and just by visually inspecting it it appears to belong to class 4. In Section 2.3 a special initial condition (a single black cell) was investigated and then the cellular automaton with rule number 54 displays a repetitive behavior (see Appendix B) while the cellular automaton with rule number 110 displays a class 4 behavior. See a further discussion of rule number 54 in the next subsection.

2.5 Sensitivity to initial conditions

In the previous subsection four basic classes of elementary cellular automata were identified by simply looking at the overall appearance of the patterns they produce. These four classes have also other significant distinguishing features. Here the sensitivity to small changes in the initial condition will be investigated.

In Figure 2.8 the effect of changing the state of a single cell in an initial random configuration for typical cellular automata from each of the four classes is illustrated. The LaTeX-code for the pictures in Figure 2.8 are produced by a slight modification of the C++-code in Appendix A. Two identical elementary cellular automata with identical random initial configurations (except for the middle cell) are run in parallel. For each step the two configurations generated are compared cell by cell. Cells with differing states in the two configurations are made black and otherwise white.

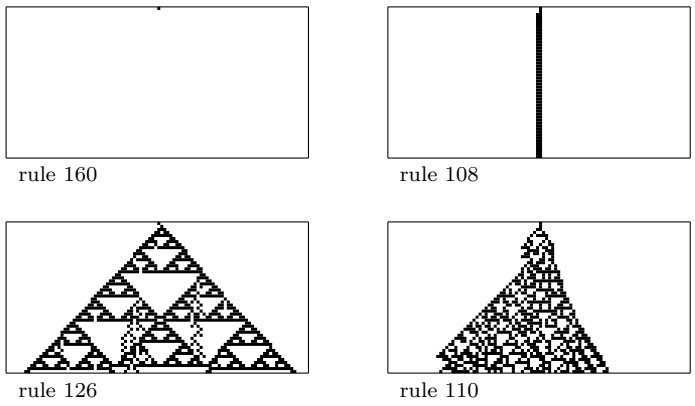


Figure 2.8: *The effect of changing the state of a single cell in the random initial conditions for typical elementary cellular automata from each of the four classes. Black indicates all the cells that change.*

The results are rather different for each of the four classes. In class 1 changes die out. The same final configuration is reached independent of the initial configuration. In class 2 changes may persist, but they remain localized in a small region of the lattice. In class 3, on the other hand, a change is typically spread at a uniform rate, and eventually it is

affecting every part of the lattice. Finally, in class 4 changes also spread, but only in a sporadic way. Figure 2.9 is a further illustration of this (with more cells and configurations than in Figure 2.8).

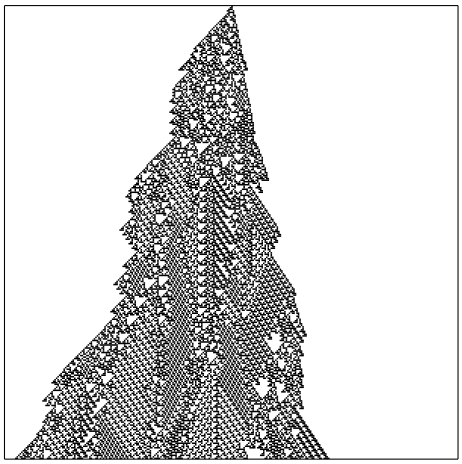


Figure 2.9: *Sensitivity of initial conditions for elementary cellular automaton with rule number 110. The lattice consists of 300 cells and 300 configurations are printed.*

What the pictures in Figures 2.8 and 2.9 reveal are basic differences in the way that each class handles information. In class 1 information about initial conditions is rapidly forgotten, while in class 2 some information about initial conditions is remained, but only in localized parts of the lattice. In class 3 a long-range communication of information is shown where a change made anywhere in the lattice usually will be communicated to all parts of the lattice. In class 4, on the other hand, a long-range communication of information is in principle possible but it will not always occur. Only if localized structures that move across the lattice are affected by changes, the changes will be communicated to other parts of the lattice.

The handling of information is something that seems to be a fundamental characteristic of the elementary cellular automata in the various classes. In the previous subsection the cellular automaton with rule number 73 was classified as being an intermediate of class 2 and 3. In Figure 2.10 its sensitivity to initial conditions is shown. The behavior is a typical

class 2 behavior.

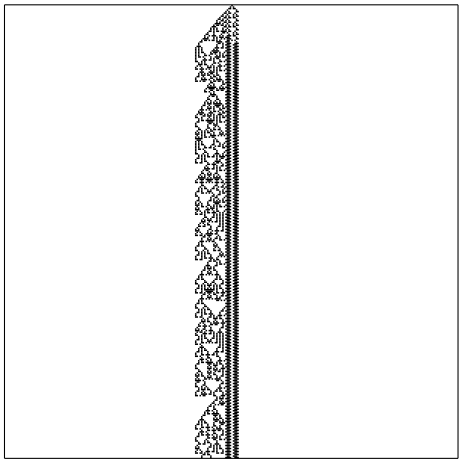


Figure 2.10: *Sensitivity of initial conditions for the rule 73 elementary cellular automaton. The lattice consists of 300 cells and 300 configurations are printed.*

Another cellular automaton which was not properly classified in the previous subsection was that with rule number 54. In Figure 2.11 its sensitivity to initial conditions is shown. By comparing with Figure 2.9 it is clear that rule number 54 shows a typical class 4 behavior, at least concerning the sensitivity of initial conditions.

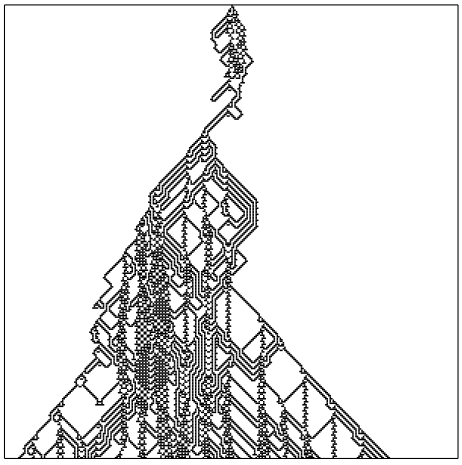


Figure 2.11: *Sensitivity of initial conditions for the rule 54 elementary cellular automaton. The lattice consists of 300 cells and 300 configurations are printed.*

2.6 Systems of limited size

So far the lattices considered have been of limited size. With a size of n cells there can be only 2^n different configurations if each cell can be in two possible states. This indicates that all elementary cellular automata with all possible initial configurations eventually (at most after 2^n steps) will yield a configuration that has already appeared. After that the steps will repeat. This makes all cellular automata with finite lattices repetitive.

The repetiveness is necessary to mention, but the practical consequences of it are negligible. This is due to the fact that, while the repetition period of cellular automata in class 2 is roughly of $O(n)$, the repetition period of cellular automata in class 3 is roughly of $O(2^n)$. For $n = 30$ there are already around a billion of configurations. In the figures in the previous subsections there have been even more cells and in those cases the number of configurations has greatly exceeded billions of configurations. So even though the number of configurations is limited for a finite lattice, this number is so large that class 3 systems still can be considered random. The repetition period is so large that it is no longer significant to talk about repetition.

2.7 Other cellular automata

In the previous subsections elementary cellular automata have been investigated in quite some detail. These cellular automata are by most measures the simplest possible. But it is possible also to look at rules that are more complicated, for example rules that involve three states rather than two. The total number of possible rules of this kind turns out to be immense—$3^{3^3} = 7.625.597.484.987$ in all. Compare this number to $2^{2^3} = 256$ which is the number of elementary cellular automata.

It is an impossible task to investigate all these cellular automata where each cell can be in three possible states. Instead Wolfram has investigated the "totalistic" cellular automata where the number of states is the same but where the rule is set up so that the

new state of every cell is determined by the average of the previous states of the cell and its immediate neighbors. There are in total $3^7 = 2187$ possible totalistic cellular automata and Wolfram has investigated them all. Although the rules of the totalistic cellular automata are more complicated than the rules of the elementary cellular automata, the totalistic cellular automata do not seem to have a fundamentally more complicated behavior. The same basic themes, as repetition, nesting, uniform randomness, and localized structures, occur. The conclusion seems to be that adding complexity to the underlying rules does not ultimately lead to more complex overall behavior. So it seems, for all cellular automata, that all the essential ingredients needed to produce even the most complex behavior already exist in the rules of the elementary cellular automata.

Apart from increasing the number of states of each cell for producing more complicated cellular automata one can increase the dimension of the lattice. Wolfram's conclusion also here is that the basic phenomenon of complexity does not seem to depend in any crucial way on the dimensionality of the cellular automaton.

3 Simple Programs

Cellular automata represent a class of programs that are extremely simple (see Appendix A). Yet they are able to produce a behavior that is diverse and often complex. In this section other simple programs will be investigated and the question is, of course, if they also will demonstrate a diverse and complex behavior. Or is this behavior something unique for cellular automata.

3.1 Mobile automata

One basic feature of a cellular automaton is that the states of all cells in the lattice are updated in parallel at every step in its evolution. To investigate the importance of this feature in the overall behavior of a simple program another class of systems, which Wolfram

denotes mobile automata, will be considered. Mobile automata are exactly like cellular automata except that instead of updating all cells in each step, only one cell, called the active cell, gets updated. In addition to the rules that apply to cellular automata, mobile automata have rules for how the active cell should move from one step to the next.

Wolfram has investigated mobile automata with rules like the elementary cellular automata and where the active cell moves either left or right one cell at each step. The behavior of these mobile automata is very simple and repetitive. By a slight modification of the rules, allowing not only the state of the active cell to be updated at each step but also the states of its immediate neighbors, also nested and random structures can be seen.

After studying generalized mobile automata, which allow more than one active cell at a time, Wolfram came to the conclusion that complex behavior almost never occurs except when a large number of cells is active at the same time. Thus, there seems to be a direct correlation between overall activity and the likelihood of complex behavior. This explains why complex behavior is so much more common in cellular automata than in mobile automata.

3.2 Turing machines

Turing machines [2] are similar to mobile automata in that they also consist of a line of cells, known as the "tape", and that they also have a single active cell, indicated by the "head". But unlike mobile automata the head of a Turing machine can be in several possible states. Another difference is that the rule for a Turing machine can not depend on the states of any neighboring cells, but only on the state of the head and the state of the active cell.

Turing machines are widely used in theoretical computer science, where examples with a large number of possible states for both the head and the cells are constructed. But in fact also "simple" Turing machines, with just two possible states for both the head and the cells, can be non-trivial. Both repetitive and nested behavior is seen to occur, but nothing

more complicated. Increasing the number of states of the head to three does not change the situation—at least not if all cells are in the same state initially. With four states of the head, however, more complicated behavior is seen as patterns with seemingly random features. By increasing the number of states of the head further does not introduce new changes in the behavior, although apparent randomness becomes slightly more common.

Just as for mobile automata, it seems that there is a threshold for complex behavior for a Turing machine (reached with four states of the head). And just as in cellular automata, a further addition of comlexity to the rules does not yield more complex behavior.

3.3 Substitution systems

One common feature of cellular automata, mobile automata, and Turing machines is that they, at the lowest level, consist of a fixed array of cells. Substitution systems, on the other hand, are set up so that the number of cells can change. In Figure 3.1 an example of a substitution system is given, illustrated in to ways. In the picture to the left of Figure 3.1 all cells have the same size while in the picture to the right, the total length of all cells at each step has the same size.

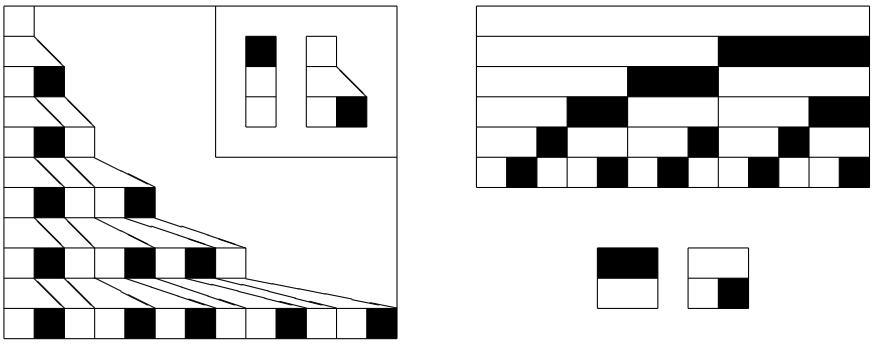


Figure 3.1: *Example of a substitution system, with two possible states of the cells (black and white), illustrated in two ways.*

In the picture to the right of Figure 3.1 the regularity of the pattern produced is obvious. The pattern just consists of a collection of identical nested pieces. To get a more complicated behavior the rule must depend not only of the state of a single cell, but also

on the state of at least one of its neighbors.

Another feature of the substitution system in Figure 3.1 is that the total number of cells never decreases from one step to the next. It is, however, also possible to construct substitution systems where cells can disappear. If the rate of such disappearance is too large, then almost any pattern will quickly die out. If there are too few disappearances, on the other hand, then most patterns will grow very rapidly. In a small fraction of rules the creation and destruction of elements are almost perfectly balanced. It turns out that substitution systems with the same type of rules as discussed above, all those which yield slow growth also seem to produce simple repetitive patterns. But as with mobile automata and with Turing machines, allowing three or four states rather than two makes more complicated behavior possible.

A substitution system works in parallel on all cells by replacing each cell by a new sequence of cells. It is also possible to consider sequential substitution systems where the array of cells is scanned from left to right, looking for a sequence of cells in a certain arrangement of states and then to perform a replacement for the first such sequence that is found. This setup is directly analogous to the search-and-replace function of a typical text editor. Also sequential substitution systems produce behavior of great complexity.

3.4 Register machines

The various kinds of systems discussed so far can be implemented on practical computers. However, none of them act like typical computers. Register machines are specifically designed to be simple idealizations of present-day computers. Most practical computers have a number of registers and support a number of instructions. As a simple idealization of a computer one can consider register machines with just two registers—each storing a number of any size—and just two instructions—increments and decrement-jumps. The increment instruction increases the number in a particular register by one. The decrement-jump instruction does two things. First, it decreases the number in a particular register

by one and then jumps to some specified other instruction in the sequence of instructions making up the rule or program for the given register machine.

Since it is assumed that the numbers in the registers can not be negative, a register that is already zero can not be decremented. A decrement-jump instruction performed on such a register will do nothing. The register will be left unchanged and the next instruction in the program will be executed. This feature of the decrement-jump instruction makes it possible for the register machine to take different paths depending on the values in the registers.

The overall behavior of register machines with four or fewer instructions in the program is essentially repetitive. With five instructions slightly more complicated behavior, like regular nested structures, becomes possible. With up to seven instructions no more complicated behavior is seen. But with eight instructions more complicated behavior shows up that is complex and seemingly quite random.

To make the register machines more complicated, by increasing the number of registers and/or by increasing the number of underlying instructions, for example by introducing instructions that refer to two registers at a time, does not seem to have much effect on either the form of complex behavior that can occur or how common it is.

Register machines can provide fairly accurate idealizations of the low-level operations of real computers. And as a result, programs for register machines are often very much like programs written in actual low-level computer languages. With this correspondence, the general results on register machines can also be expected to apply to programs written in low-level computer languages.

4 Fundamental Issues

In the past sections the main purpose has been to address the question of how simple programs behave. In this section the purpose is to discuss some fundamental issues that

are relevant in the study of actual phenomena in nature.

4.1 Randomness

After the investigation of many simple programs, Wolfram has come to the conclusion that three basic mechanisms for randomness can be identified. These are illustrated in Figure 4.1.

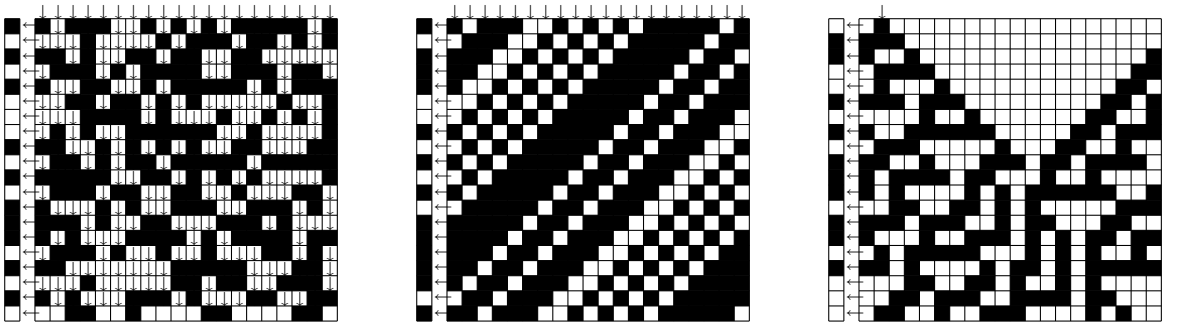


Figure 4.1: *Three possible mechanisms responsible for randomness. The vertical arrows represent external input. In the first case, random input from the environment is present at every step. In the second case, there is random input only in the initial conditions. In the third case there is hardly any random input at all. Despite their differences, all three mechanisms lead to randomness in the left-most column.*

In the first mechanism (see Figure 4.1) randomness is explicitely introduced into the underlying rules for the system, so that in effect a random state is chosen for every cell at each step. This mechanism for randomness is the one most commonly considered in the traditional sciences. Essentially it corresponds to letting a random external environment continually affecting the system under consideration and continually injecting randomness into it. The real origin of such randomness is that there are innumerable details about the environment that it is very difficult to know about.

In the second mechanism (see Figure 4.1) there is no such interaction with the environment. Randomness is introduced into the system by the randomly chosen initial conditions. The subsequent evolution of the system, however, is supposed to follow definite rules that

involve no randomness. In the example shown in Figure 4.1 the rules are simply set up to shift the state of a cell one position to the left at each step. This makes the sequence of states taken on by a particular cell depend on the states of cells progressively further and further to the right in the initial conditions. Since the initial conditions are chosen randomly, the sequence of states of any particular cell will be correspondingly random. The basic idea in the second mechanism is therefore that the randomness seen is some kind of transcription of the randomness present in the initial conditions.

The two mechanisms for randomness discussed so far, both assume that randomness in any particular system comes from outside of that system. What the previous sections have shown is that simple programs, like cellular automata, can produce apparently random behavior even when they are given no random input whatsoever. This is the third mechanism in Figure 4.1. Practically every kind of simple program that can be constructed is capable of generating such randomness. Therefore, as Wolfram claims, it is quite reasonable to expect that this same mechanism should also occur in many systems in nature. Wolfram goes even further and says that he believes that this third mechanism is in fact ultimately responsible for a large fraction, if not essentially all, of the randomness seen in the natural world.

The rule 30 elementary cellular automaton in Figure 4.2 is the cellular automaton in which Wolfram first identified the third mechanism for randomness. Despite simple rules and simple initial conditions this cellular automaton yields a behavior that in many respects seems random.

Although there are some regularities, like the pattern to the left, the overall pattern seems random. By specifically picking out the state of the center cell on successive steps, one gets a seemingly random sequence. What one usually means with random in practice, is that one can not see any regularities in it. For detecting regularities methods in, for example, mathematics and statistics, can be used. And none of these methods seem to reveal any real regularities in the rule 30 cellular automaton center sequence.

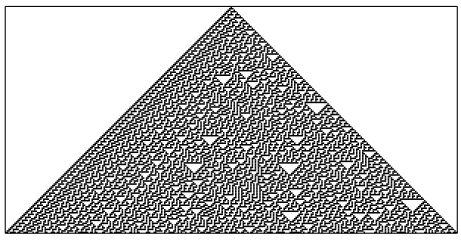


Figure 4.2: *The rule 30 elementary cellular automaton. Despite no random input to the system, its behavior seems in many respects random.*

Most computer systems and languages have facilities for generating random numbers. In *Mathematica* (Wolfram's creation) this has been done using the rule 30 cellular automaton. Every time the random number generator function is called, another step in the cellular automaton evolution is performed, and the state of the center cell is returned. For practical reasons the pattern is not allowed to grow wider and wider forever. Instead, it is wrapped around in a lattice that is a few hundred cells wide. As was discussed in Section 2.6, one consequence of this is that the sequence will finally repeat. But this has no practical relevance since the actual period of repetition will typically be more than a billion billion times the age of the universe.

In *Mathematica*, the initial conditions of the rule 30 cellular automaton are not explicitely specified. Instead various features of the computer system at the time of the first call of the random number generator function are used as an initial condition.

4.2 Discrete versus continuous

The different kinds of programs discussed in the previous sections have at least one thing in common. They all have cells that can take on only a discrete set of possible states, typically black and white. This fact, of course, arises the question whether the complexity seen in these systems depends on the discreteness of the systems.

First the problem of discrete sets of possible states will be addressed. By investigating a generalization of cellular automata, in which each cell is not just black or white, but

instead can have any of a contiuous range of possible levels of gray, Wolfram came to the conclusion that in these continuous cellular automata it takes only extremely simple rules to generate behavior of great complexity. And in fact, it is also possible to find cases that exhibit localized structures.

Although the discreteness in the states was removed in the continuous cellular automata described above there is still much discreteness that remains. A continuous cellular automaton is still made up of discrete cells whose states are updated at discrete time steps. By introducing partial differential equations this last source of discreteness can be removed. After, what it seems, some trial end error, Wolfram has found examples of partial differential equations with highly complex behavior. And indeed, even though the underlying equations are continuous, the patterns they produce seem to involve patches with a discrete structure.

The conclusion from the discussion above is that complex behavior is in no way restricted to systems based on discrete elements. The same kind of behavior can also occur in completely continuous systems such as partial differential equations, although discrete systems are far easier to study.

In nature there is a plentitude of discrete systems that at a macroscopic level seem smooth and continuous. For example, air and water seem like continuous fluids, even though they in fact are made up of discrete molecules. Another example concerns sand which flows like a continuous fluid, even though it is actually made up of discrete grains. So what is the basic mechanism that allows systems with discrete components to behave smooth and continuous? The answer seems to be randomization. When randomness is present microscopic details are averaged out, leaving no trace of discreteness. Instead the results appear smooth and continuous. A classic example of this phenomenon is the so called random walk. A discrete particle is moving randomly and at each step it is either moving one step to the left or one step to the right. At any particular time the particle will be at a definite discrete position. In Figure 4.3 random walks for one, two, and ten

particles are shown. Instead of looking at the position of each individual particle, one can look at the overall distribution of positions of all particles. As is illustrated in Figure 4.4, a large number of particles makes the distribution take on a smooth and continuous form. The randomness has in a sense washed out the traces of the underlying discreteness of the system.

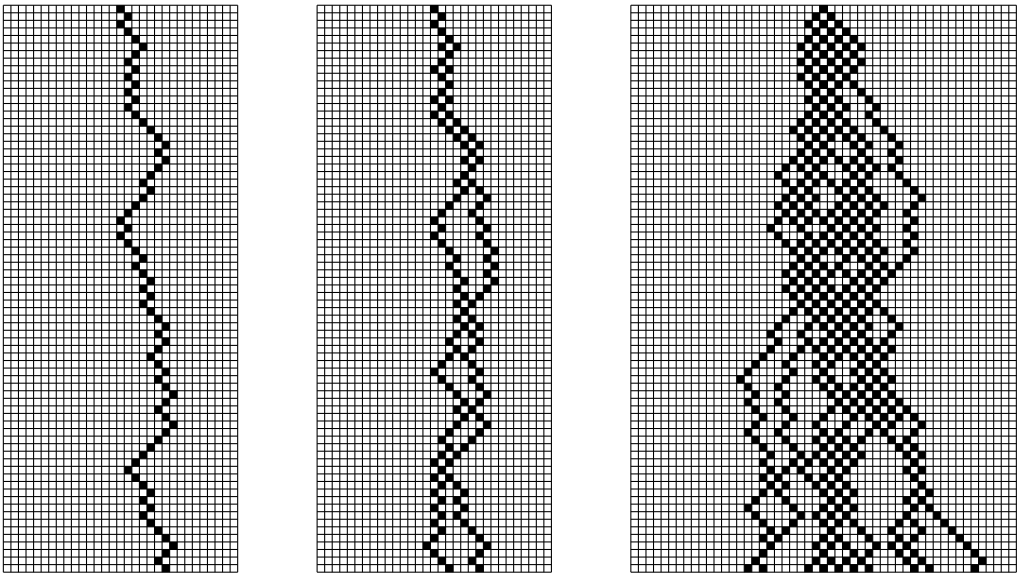


Figure 4.3: *Random walks for one, two, and ten particles, respectively. At each step a particle can go either one step to the left or one step to the right with equal probability.*

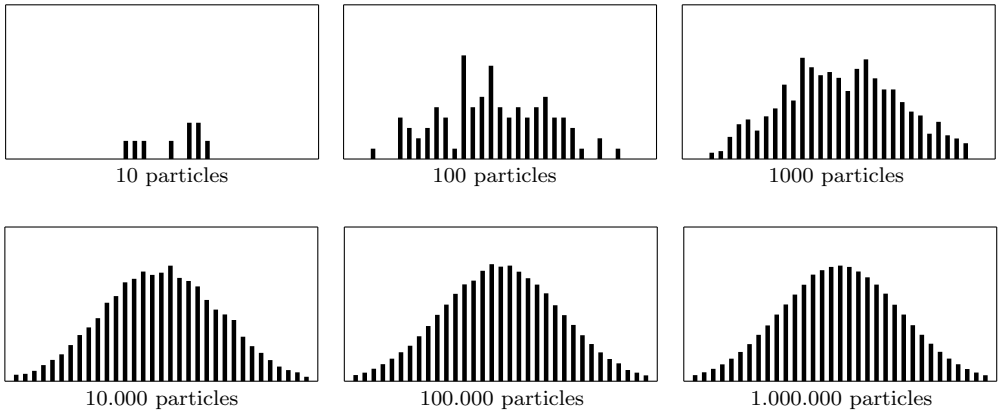


Figure 4.4: *Distribution of positions of 10, 100, 1000, 10000, 100000, and 1000000 particles performing random walks (175 steps).*

In the random-walk example, randomness is inserted from outside at each step in the evolution of the system. In Figure 4.5 a system where randomness comes from the mechanism of intrinsic randomness is given. The detailed pattern of black and white cells changes at every step. But the large domains of black and white that form have boundaries which move only rather slowly. At an overall level these boundaries behave in a smooth and continuous way.

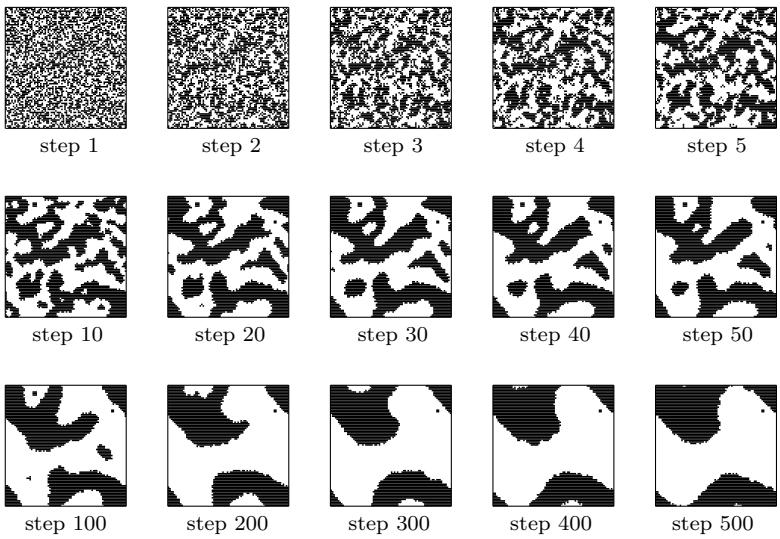


Figure 4.5: *2-dimensional cellular automaton with a random initial configuration. The rule is as follows: If the number of black cells in the 9-cell neighborhood of a cell (which includes the cell itself and the 8 cells adjacent to it) is less than 4 or equal to 5 then the cell becomes white at the next step, otherwise it becomes black. Each picture is 80 cells across and is wrapped around to avoid boundaries.*

Just as discrete models can yield continuous behavior, continuous models can yield behavior that appears discrete. The boiling of water provides a classic example. When heating water, nothing much happens at first. But when the temperature reaches 100 °C, a discrete transition occurs, and all the water evaporates into steam. Another example is the motion of a ball in a double well. The motion is governed by mathematical equations of a continuous form, but still a discrete transition occurs depending on whether the ball starts to the left or right of the center line.

In nature there are in fact many kinds of systems where continuous changes lead to discrete transitions. Also in elementary cellular automata it is possible to find such phenomena. In Figure 4.6 the rule 184 cellular automaton is shown where the density of black cells in the initial configuration is continuously changed (which actually only can be achieved with infinite lattices). Maybe one would expect that continuous changes would result in effects that are correspondingly continuous. However, what the pictures in Figure 4.6 show is that if the initial density of black cells has any value less than 50%, only white stripes survive. But when the initial density increases above 50%, a discrete transition occurs and only black stripes survive. One can also argue whether this is actually a true discrete transition since with exactly 50% black cells in the initial configuration no stripes survive, neither black nor white. Looking at Figure 4.6 it seems that the density of white stripes is smoothly approaching zero when the initial density of black cells is approaching 50%. The same is true for the density of black stripes which is smoothly approaching zero when the initial density of black cells is decreasing towards 50%.

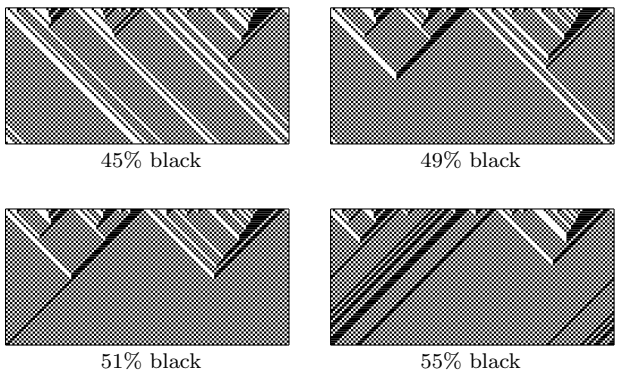


Figure 4.6: *The rule 184 elementary cellular automaton with different densities of black cells in the initial configuration. When the initial density is less than 50% only white stripes survive, and when the density is more than 50% only black stripes survive.*

It turns out that discrete transitions are fairly rare among one-dimensional cellular automata. In two or more dimensions they become increasingly common. In Figure 4.7 the same two-dimensional cellular automaton as in Figure 4.5 is shown, illustrating a discrete transition. In regions dominated by black cells, an increasingly large fraction of

cells become black, and vice versa. As long as the boundaries of the regions do not get stuck—as happens in many one-dimensional cellular automata—the result is that the color that initially was more common will eventually take over the whole system.

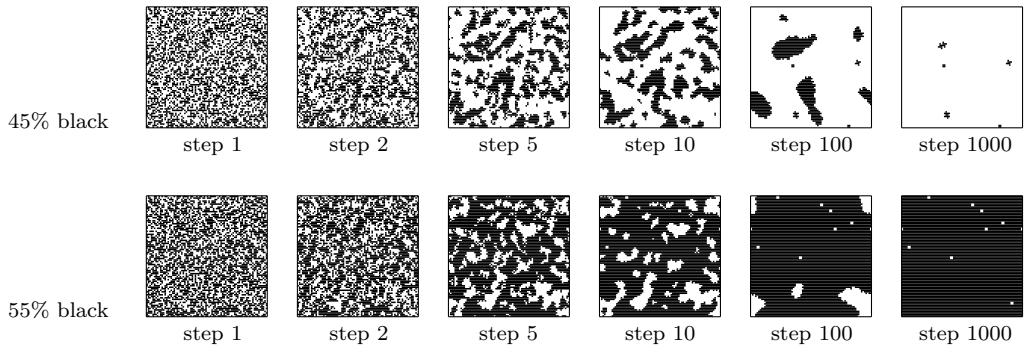


Figure 4.7: *The same two-dimensional cellular automaton as in Figure 4.5. A discrete transition occurs when the density of black cells is continuously varied.*

4.3 Reversability

The rules of a cellular automaton tell how to proceed from one step to the next. But what about going backwards? Is it possible to deduce from the arrangement of states of the cells at a particular step what the arrangement was at previous steps? In Figure 4.8 two elementary cellular automata are given. In one of them it is straightforward to do this. The state of a cell is changing in every step independent of going forwards or backwards. This cellular automaton is reversible. The other cellular automaton works differently. Here it is not possible to go backwards since all cells end up being in the same state after a couple of steps. This cellular automaton preserves no information about the arrangement of states of the cells on earlier steps, it is therefore irreversible.

There are more elementary cellular than the one shown in Figure 4.8 that are reversible. In Figure 4.9 all six of the 256 elementary cellular automata that are reversible are displayed. They all exhibit fairly trivial behavior.

It is not always the case that reversible cellular automata exhibit simple behavior.

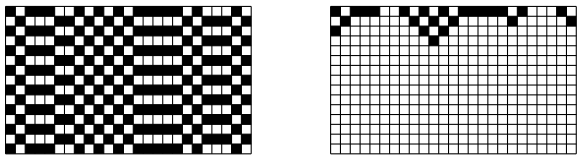


Figure 4.8: *Examples of elementary cellular automata that are reversible (rule 51) and irreversible (rule 32).*

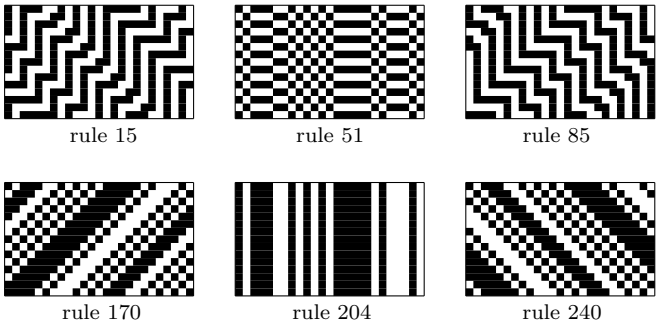


Figure 4.9: *The six reversible elementary cellular automata.*

Out of the 7.625.597.484.987 cellular automata with three states and nearest-neighbor rules 1800 are reversible, and of these most exhibit simple behavior, but some do exhibit complex behavior.

In order to understand the range of behavior that can occur in reversible systems it is convenient to consider classes of cellular automata with rules that are constructed to be reversible. One such class is given in Figure 4.10.

The class of cellular automata in Figure 4.10 is constructed by taking the elementary cellular automata and then add dependence on states two steps back. Therefore two preceding steps are needed for the construction of the next step of evolution of such cellular automata. The resulting rules can run both forwards and backwards, and can be used to determine the configuration of cells on either future or past steps.

The class of reversible cellular automata in Figure 4.10 will be denoted elementary reversible cellular automata, and similar to the ordinary elementary cellular automata they will be numbered from 0R to 255R, where R stands for reversible. In Figures 4.11

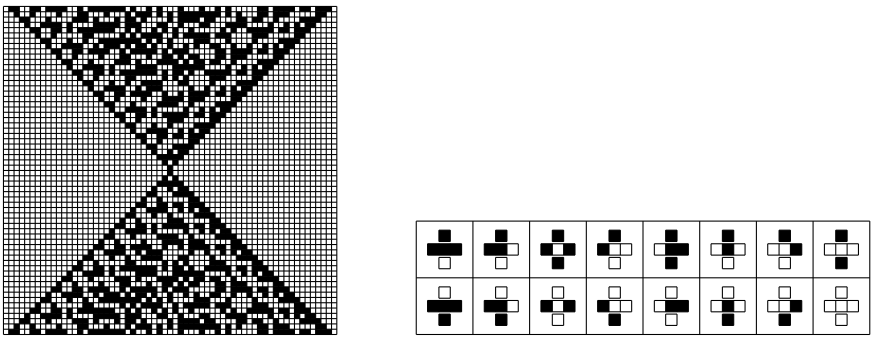


Figure 4.10: *A reversible cellular automaton determined from the rule 214 elementary cellular automaton with the addition of the specification that the new color of a cell should be inverted whenever the cell was black two steps back.*

and 4.12 examples of the behavior of such cellular automata with random and simple initial configurations, respectively, are given.

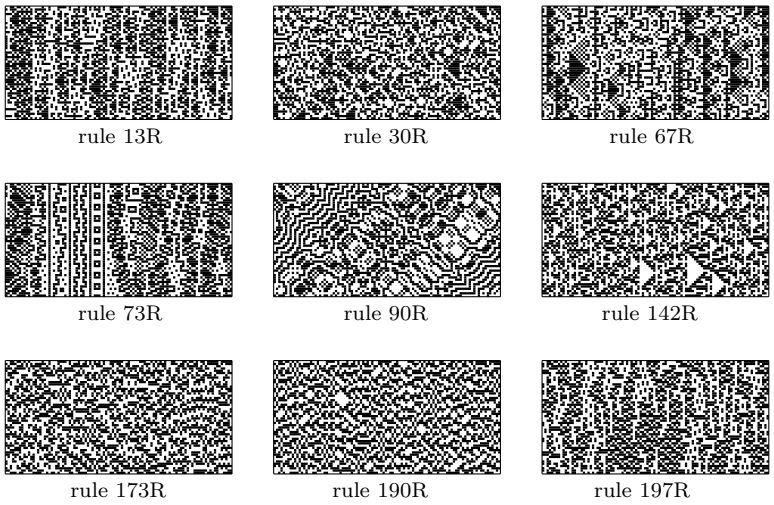


Figure 4.11: *Examples of elementary reversible cellular automata starting from random initial configurations. Every cell is chosen to be black or white with equal probability on the two successive first steps.*

As is illustrated in Figure 4.12 the behavior of some elementary reversible cellular automata is fairly simple, and the patterns produced have simple repetitive or nested structures. But in many cases, especially with random initial configurations, but even with simple initial configurations, the patterns produced are highly complex, and seem in

many respects random. In Figure 4.13 it is demonstrated that localized structures also can occur in reversible systems. Thus the conclusion of this subsection is that, even though reversible systems are quite rare among possible systems they can still exhibit behavior just as complex as other systems.

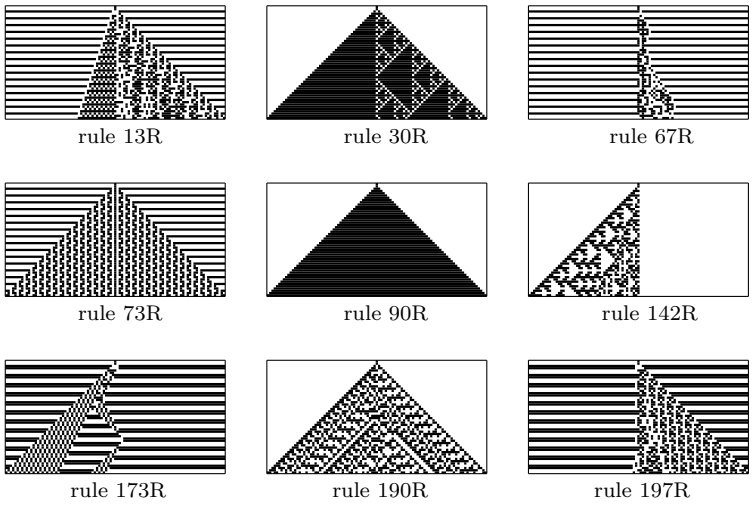


Figure 4.12: *Examples of elementary reversible cellular automata starting from simple initial configurations. Only the center cell is black on the two successive first steps.*

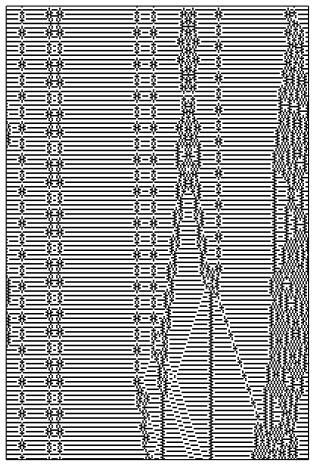


Figure 4.13: *An example of an elementary reversible cellular automaton where localized structures occur (rule 37R). The initial configurations (the first two steps) were prepared to contain just about 5% black cells, randomly chosen.*

4.4 Entropy

Although the underlying laws of physics are reversible, at least suggested by current research, there are many examples of systems of nature which seem irreversible. A system might start in a fairly regular or organized state and then progressively become more and more random and disorganized. This phenomenon can also be seen in many simple programs. Figure 4.14 shows an example based on an elementary reversible cellular automaton. At the beginning the black cells are placed at the center of the lattice, with identical configurations in the first two steps. But over the course of time the distribution of black and white cells becomes progressively more random.

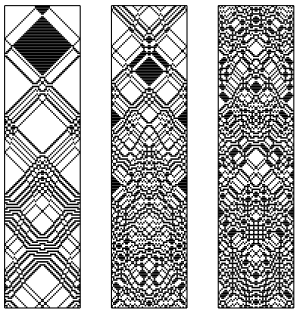


Figure 4.14: *A reversible cellular automaton (rule 122R) that exhibits seemingly random behavior. Initially all black cells lie at the center of the lattice, in the first two configurations, but progressively this distribution becomes more and more random.*

Although it is, by now, not surprising that a system with simple initial conditions can generate randomness, it may seem odd that a reversible system can generate, what seems like, an irreversible increase in randomness. This apparent conflict can be resolved quite straightforward. As is demonstrated in Figure 4.15 a carefully constructed initial configuration can through the evolution of the system produce a simple distribution of black cells.

The systematic decrease in randomness seen in Figure 4.15 yet seems contradictory to everyday experiences. The resolution of this paradox is that it is not possible to set up such precise initial conditions in practice. Practical experiments involve only initial conditions

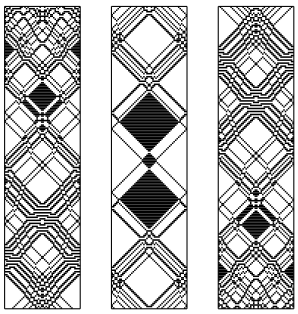


Figure 4.15: *An extended version of Figure 4.14 in which the initial configurations are carefully constructed so that a simple arrangement of black cells will be produced.*

that are fairly simple to describe and to construct. And in these cases systems, like the one in Figure 4.14, always tend to exhibit increasing randomness. Wolfram argues that no reasonable experiment can ever involve setting up the kind of initial conditions that will lead to a decrease in randomness. According to Wolfram this is the basic argument that explains the observed validity of the Second Law of Thermodynamics, which says that entropy increases with time.

The randomness produced by various cellular automata seems in many respects to be independent of the details of the initial conditions. This is demonstrated by the rule 122R elementary reversible cellular automaton in Figure 4.16. If a system generates sufficient randomness, it is possible to think of it as evolving towards a unique equilibrium whose properties are independent of its initial conditions.

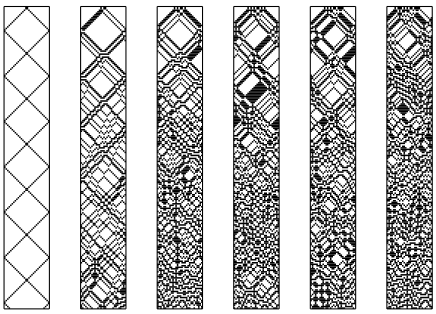


Figure 4.16: *The rule 122R reversible cellular automaton approaching equilibrium using various initial conditions. Apart from exceptional cases the behavior is quite indistinguishable in its overall properties.*

The fact that systems evolve towards a unique equilibrium is implicit in many everyday applications of physics. It is this fact that makes it possible to characterize a physical system just by specifying a few parameters such as temperature and chemical composition. The existense of a unique equilibrium that a system evolves to is a common statement of the second law of thermodynamics. But how general is the second law? In Figure 4.17 examples of various reversible cellular automata are shown. Some systems clearly exhibit the behavior implied by the second law, and others do not. Examples from the last category are rules 0R and 90R, with their repetitive forms. Existing mathematical studies have identified these simple exceptions to the second law. But what about rule 37R? This system does not seem to settle down. Sometimes it becomes less orderly and sometimes more so. For practical purposes it has to be concluded that rule 37R does not obey the second law.

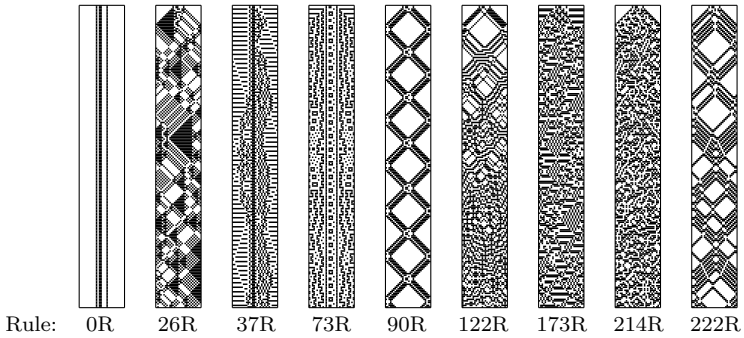


Figure 4.17: *Examples of elementary reversible cellular automata. Some quickly randomize, as suggested by the second law of thermodynamics, while others do not.*

Just as in the world of simple programs there might be many systems in nature that do not obey the second law of thermodynamics. The second law is an important and quite general principle, but it is not universally valid. To conclude this subsection it can be stated that simple programs are useful not only in understanding why the second law is often true, but also in seeing some of its limitations.

4.5 Conservation

Another general feature in the basic laws of physics is the conservation of various quantities. Cellular automata do not usually exhibit such conservation laws. But just as with reversability there are some elementary cellular automata with conservation properties. In Figure 4.18 the cellular automata that conserve the number of black cells that appears on each step are given.

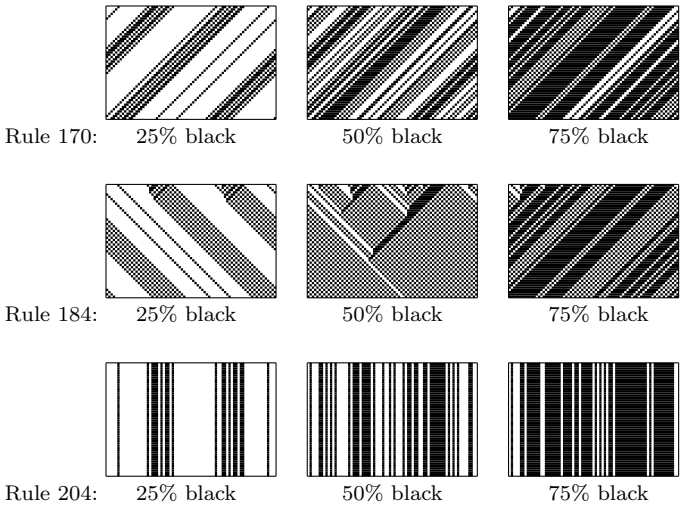


Figure 4.18: *Elementary cellular automata whose evolution conserves the total number of black cells.*

5 Computer Science

In this section some issues related to computer science are discussed.

5.1 Cryptography

Cryptography has been used since ancient times to hide the contents of messages by encrypting them so as to make them unrecognizable. An example of a scheme for encryption of a message, represented by a sequence of black and white cells, is shown in Figure 5.1.

The basic idea is to have an encrypting sequence, and from the original message the encrypted version of the message is obtain by reversing the color of every cell for which the corresponding cell in the encrypting sequence is black.

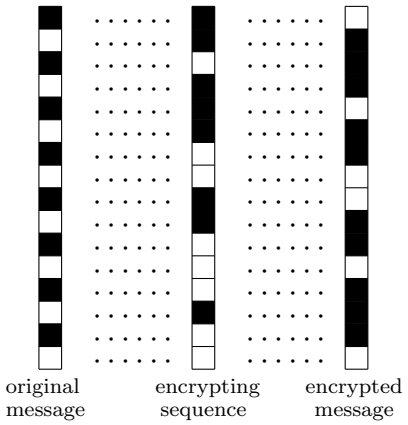


Figure 5.1: *Example of a scheme for encryption. An encrypted message is generated be reversing the color of each cell for which the corresponding cell in the encrypting sequence is black.*

So if one receives an encrypted message, then the original message can be easily recovered if the encrypting sequence is known. But in most situations it is not feasible to transmit the whole encrypting sequence in a secure way. Instead only a short key, from which the whole encrypting sequence is generated, can be transmitted in a secure way.

There are several approaches of how to get an encrypting sequence from a key. A simple approach that was widely used until less than a century ago is shown in Figure 5.2. The encrypting sequence is formed by repeatedly cycling through the elements of the key.

It is fairly easy to decrypt messages encoded with such a simple encryption system as in Figure 5.2. If a sufficiently long segment in the encrypting sequence can be found out then this immediately gives the key. So what about more complicated rules for generating an encrypting sequence from a key? In Figure 5.3 an additive elementary cellular automaton in a lattice of limited size is used to generate an encrypting sequence. The key is the initial configuration and the evolution of a particular cell gives the encrypting sequence. This procedure was widely used in the early years of electronic cryptography, and it is still

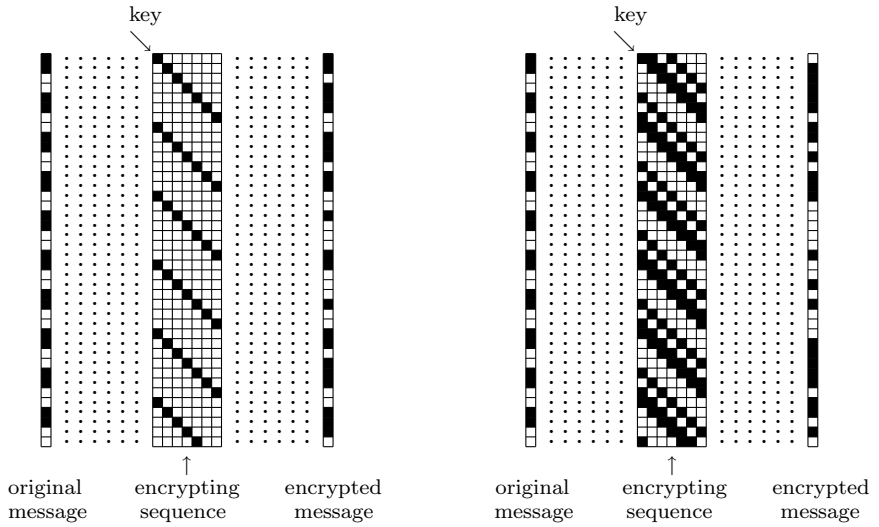


Figure 5.2: *A simple example of an encryption system in which the encrypting sequence is formed by repeatedly cycling through the elements of the key. Encryption with two different keys is shown.*

sometimes used today.

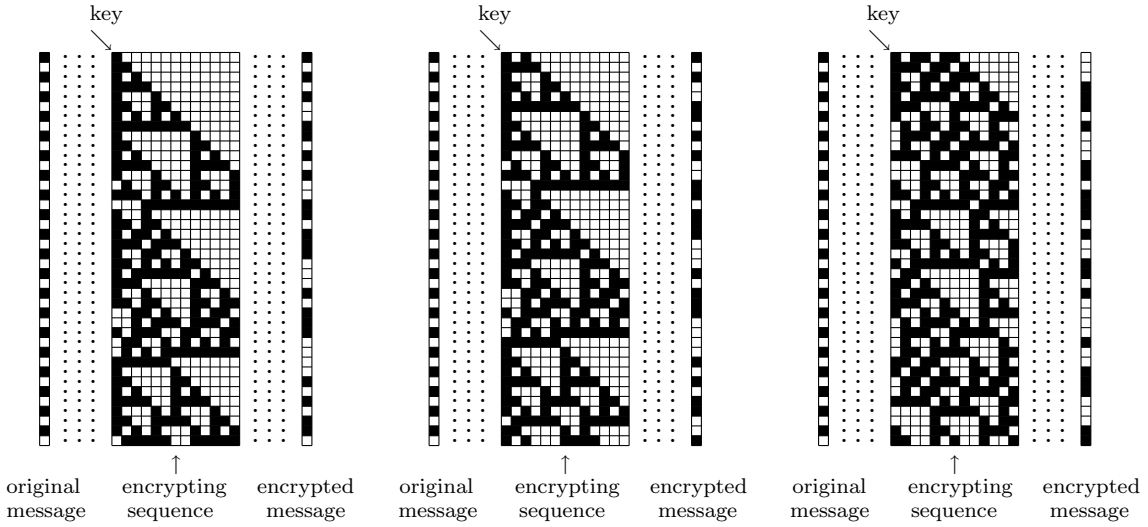


Figure 5.3: *Encryption using the rule 60 elementary cellular automaton. This rule is additive, meaning that the new state of a cell is obtained by adding the states of the cells in the neighborhood modulo 2 with weights between 0 and 1.*

Messages encoded with the encryption system in Figure 5.3 are, however, fairly easy to

decrypt. Even though the encrypting sequence does not include every single cell in a particular column, the additive nature of the underlying rule still makes cryptanalysis possible. The question now is whether there are elementary cellular automata that can be used for cryptography. The rule 30 cellular automaton, see Figure 5.4, was suggested by Wolfram in 1985. Unlike the additive cellular automaton in Figure 5.3 with its nested structure, most sequences that are generated with rule 30—even with simple initial configurations—appear completely random. By trying every possible initial configuration it will in principle be possible to decrypt a message, but as the width of the cellular automaton increases, the total number of possible keys will rapidly become astronomical. To test all these keys would be completely infeasible. However, by studying the rule of the cellular automaton it is possible to do some cryptanalysis and to find candidate keys. But it is rather easy to destroy this possibility to cryptanalysis on the rule 30 cellular automaton by not including every single cell in a given column when forming the encrypting sequence. Direct attempts to find easy ways to deduce the key in rule 30 have failed and it appears that there are no easy ways to deduce the key for rule 30 from any suitable chosen encrypting sequence.

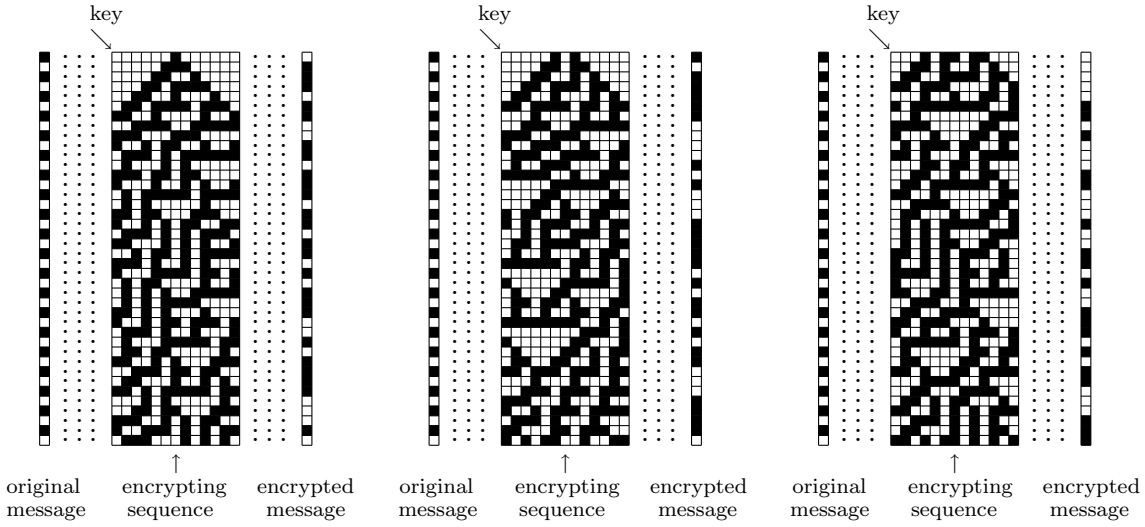


Figure 5.4: *Encryption using the rule 30 elementary cellular automaton.*

5.2 Computation

Systems such as cellular automata have previously been referred to as simple computer programs. One can think of these systems in terms of the computations they can perform. The initial configuration can be viewed as the input to a computation, while the configuration after some number of steps can be viewed as the output.

What kinds of computations are cellular automata able to do? In Figure 5.5 an elementary cellular automaton performing a simple computation is shown. Starting with an even number of consecutive black cells, no black cells survive after a few steps of evolution. Starting with an odd number of consecutive black cells, on the other hand, a single black cell survives forever. So this cellular automaton can be viewed as computing whether a given number is even or odd.

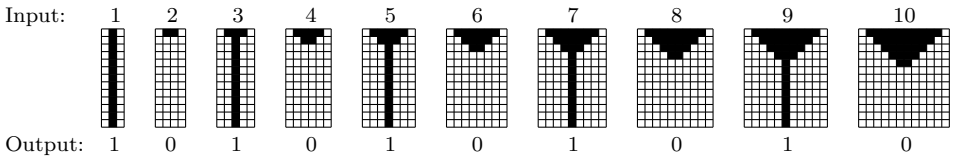


Figure 5.5: *The rule 132 elementary cellular automaton, which effectively computes the remainder after division of a number, represented by the number of consecutive black cells in the initial configuration, by 2.*

It is also possible to construct cellular automata that can do more complicated things. For example, a cellular automaton with eight possible states for each cell can be constructed to compute the square of any number. A cellular automaton that computes the successive prime numbers: 2, 3, 5, 7, 11, 13, 17, etc. can also be constructed. This, however, requires complicated rules and a total of 16 states for each cell.

To return to the elementary cellular automata, what kinds of computations can they actually perform? The computations of some of the elementary cellular automata can easily be described in terms of traditional mathematical notions, but others can not. Thus, for example, as shown in Figure 5.6, rules 94, 62 (to the left), and 190 can be described as enumerating numbers that are multiples of 2, 3, and 4, respectively. And if looking down

the center column of the pattern produced by rule 129, it can be thought of as enumerating numbers that are powers of 2. For the cellular automata in Figure 2.3, on the other hand, there is no simple description. Traditional mathematics is of no much help.

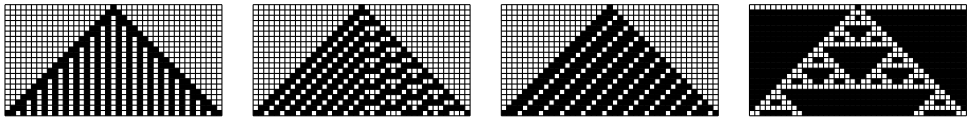


Figure 5.6: *Examples of elementary cellular automata whose computations can easily be described in traditional mathematical terms. The white cells at the bottom row of rules 94, 62 (to the left), and 190 correspond to numbers that are multiples of 2, 3, and 4, respectively, and in the center column of rule 129 to powers of 2.*

5.3 Universality

In the previous subsection the fact that cellular automata can be constructed to perform quite sophisticated computations was discussed. However, for each computation a new cellular automaton with specific rules had to be set up. Is this really necessary or is it possible to construct a universal cellular automaton in analogy with the universal Turing machine [4] in theoretical computer science on which present-day computers are based on? A single computer is not constructed to perform just one task instead it is made universal, programmable to perform various kinds of tasks determined by the program, which can be considered as the input to the computer.

The rules for a possible universal cellular automaton should always be the same. But the fact that it is universal means that, by given appropriate initial conditions, it should be able to emulate any possible cellular automaton, with any set of rules. In Ref. [1] the design and workings of such a universal cellular automaton is discussed. This universal cellular automaton is far from trivial, involving 19 possible states for each cell and rules depending on next-nearest neighbors as well as on nearest neighbors. This cellular automaton was set up to make its operation as easy to follow as possible. But even so and despite a

large number of illustrations it is not easy to understand the workings of it. However, the basic idea, for emulating the elementary cellular automata, for example, is that a block of 20 cells in the universal cellular automaton is used to represent each single cell in the elementary cellular automaton that is being emulated. Within this block of 20 cells both a specification of the current state of the cell that is being represented and the rule by which the state of that cell is to be updated are encoded.

The universal cellular automaton discussed above is in no way restricted to emulate cellular automata with rules involving only nearest neighbors. In Ref. [1] the emulation of a cellular automaton involving also next-nearest neighbors is shown. Instead of blocks of 20 cells, that were required in emulating the elementary cellular automata, blocks of 70 cells are now needed. Similarly, increasing the size of a block of cells even further, will make it possible to emulate cellular automata with rules with even more neighbors.

The question whether the universal cellular automaton given above will be able to emulate a cellular automaton with rules involving more than two states will now be addressed. It turns out that it is possible to emulate such rules by using rules with just two states of the cells but with a larger number of neighbors. The conclusion then is that also cellular automata with any number of states of the cells can be emulated by the universal cellular automaton. Therefore any cellular automaton can be emulated regardless of how many states and how many neighbors it involves. This means that the universal cellular automaton is able to emulate cellular automata with more complicated rules than its own. So as a matter of fact nothing fundamental can be gained by setting up a cellular automaton whose rules are more complicated than those for the universal cellular automaton since more complicated rules can always be emulated by the universal cellular automaton just by setting up appropriate initial conditions.

After demonstrating the existence of a universal cellular automaton Wolfram [1] proceeds by showing that systems like mobile automata, Turing machines, substitution systems, and register machines can be emulated by cellular automata. and then in particular

by a universal cellular automaton. Since cellular automata are also able to emulate other important aspects of practical computers, like logic expressions and data retrievel, Wolfram draws the conclusion that a universal celllular automaton is able to emulate a practical computer in its entirety.

Just as cellular automata are able to emulate other systems, other systems are able to emulate cellular automata. For example, by emulating the universal cellular automaton, it is possible to construct a universal Turing machine. This was in fact done already in 1936 using different methods. According to Wolfram this was historically the very first clear example of universality shown in any system.

An implication of the discussion above is that from a computational point of view various kinds of systems are fundamentally equivalent. However, this does not mean that systems of a particular type—say cellular automata—are able to support the same kinds of computations. This was clearly demonstrated in the previous subsection. Some cellular automata can perform only very simple computations. But when studying cellular automata with progressively greater computational capabilities, the threshold of universality will eventually be passed. And once passed, the set of computations that can be performed will always be exactly the same.

How complicated do the underlaying rules for a system need to be in order to achieve universality? As a matter of fact they do not have to be as complicated as the ones of the universal cellular automaton discussed in the beginning of this subsection. In the early 1970s it was known that cellular automata with 18 states and nearest-neighbor rules could be universal. And in the late 1980s examples of universal cellular automata with 7 states and nearest-neighbor rules were constructed. In the mid-1980s Wolfram began to suspect that universality actually also could be found among the elementary cellular automata. The leading candidate would be the rule 110 elementary cellular automaton displayed once again in Figure 5.7.

Despite its simple underlying rules, rule 110 supports a whole variety of localized struc-

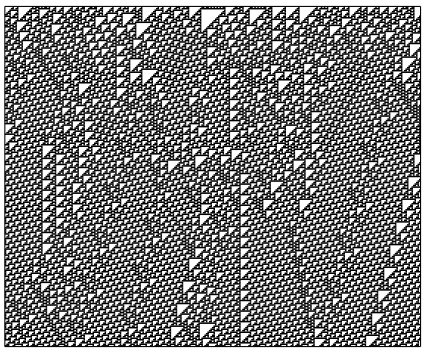


Figure 5.7: *The rule 110 elementary cellular automaton, an example of a universal cellular automaton.*

tures that move around and interact in many complicated ways. Perhaps these localized structures can be arranged to perform meaningful computations. And indeed by building up components from combinations of localized structures a large number of components can be obtained without having to increase the complexity of the underlying rules. It took several years of research to develop the necessary ideas and tools for showing that rule 110 in fact is universal. In Ref. [1] some of these are discussed in quite some detail. The significance of universality in rule 110 is that it sugggests that universality is a far more common phenomenon than one might have expected.

As was discussed in Section 2.4 rule 110 is in class 4. Wolfram strongly suspects that all cellular automata which show overall class 4 behavior will turn out to be universal. So expect from rule 110—and the related rules 124, 137, and 193—among the elementary cellular automata rule 54 might be in class 4, and if so it should be possible to show that it too exhibits the phenomenon of universality.

Previoulsy the threshold of universality was mentioned, and after making the connection between class 4 behavior and universality one might wonder what is ultimately needed for an elementary cellular automaton to be able to achieve universality. Systems displaying typical class 1 or class 2 behavior can be excluded to exhibit universality. Among class 3 systems quite a few systems might be presumptive candidates for universality, meaning that the threshold for universality ought to be in this class.

6 Summary

The topic of this treatise is cellular automata. These simple constructions, with their lattices of cells being in a finite number of states governed by simple rules, display a richness in properties and behavior. Even the property of universality, well known from the universal Turing machine [4], has been found in certain of these constructions and actually even among the simpler ones. This means that there are cellular automata that can perform computations that are as sophisticated as those that we can ever imagine and certainly as sophisticated as those that can be performed by computers.

What is the reason for studying cellular automata, apart from getting insights in aspects of computations? A hint is given in Section 4 where certain fundamental issues concerning the physical world are discussed. Wolfram has formulated a principle which he calls the Principle of Computational Equivalence, which states that "all processes, whether they are produced by human effort or occur spontaneously in nature, can be viewed as computations". So what this principle says is that, for instance, universal cellular automata can be used in describing all natural processes, even such things that normally are considered as fundamentally continuous, like positions in space and values of quantum mechanical probability amplitudes. To investigate the role of cellular automata in the modelling process in science will be the subject of a forthcoming treatise.

7 References

1. Stephen Wolfram, A New Kind of Science, Wolfram Media, 2002.

2. J. Glenn Brookshear, Computer Science An Overview, Addison-Wesley, 1997.

3. http://www.tu-bs.de/institute/WiR/weimar/ZAscriptnew/intro.html

4. John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman, Introduction to Automata Theory, Languages, and Computation, Addison-Wesley, 2001.

A C++-code for generating LaTeX-code for elementary cellular automata

```
#include <stdio.h>

#define MAX_FILE_NAME    20

// size_i: the integer part of the size of a square (in mm).
// size_d: the decimal part of the size of a square (in mm).
int size_i, size_d;

// Calculates i to the power of j.
// Precondition: j >= 0.
int exp(int i, int j)
{
    if (j == 0)
        return 1;
    else
        return i * exp(i, j - 1);
}

// Transforms an integer (I) to a binary number (B)
// Precondition: 0 <= I <= 255.
// Postcondition: B[7] B[6] B[5] B[4] B[3] B[2] B[1] B[0]:  I
//                 F    F    F    F    F    F    F    F :  0
//                 F    F    F    F    F    F    F    T :  1
//                 F    F    F    F    F    F    T    F :  2
//                 ......
//
//                 T    T    T    T    T    T    T    T :  255
void int_binary(int I, bool * B)
{
    int rest = I;

    for (int i = 7; i >= 0; i--)
        if (rest >= exp(2,i)) {
            B[i] = true;
            rest = rest - exp(2,i);
        }
        else
```

```
            B[i] = false;
}

// Initializes an array A of size s.
// Precondition: s > 0.
void init_array(bool * A, int s)
{
    for (int i = 0; i < s; i++)
        A[i] = false;

    A[s/2] = true;
}

// Returns update of array element at position p of array A of size s.
// Update is determined by array B (rule number in binary of elementary
// cellular automaton).
// Precondition: 0 <= p < s, s > 0.
bool update_el(int p, bool * A, int s, bool * B)
{
    int l = p - 1;
    int r = p + 1;

    if (l < 0) l = s - 1;
    if (r > s - 1) r = 0;

    if (A[l] && A[p] && A[r]) return B[7];
    else if (A[l] && A[p] && !A[r]) return B[6];
    else if (A[l] && !A[p] && A[r]) return B[5];
    else if (A[l] && !A[p] && !A[r]) return B[4];
    else if (!A[l] && A[p] && A[r]) return B[3];
    else if (!A[l] && A[p] && !A[r]) return B[2];
    else if (!A[l] && !A[p] && A[r]) return B[1];
    else if (!A[l] && !A[p] && !A[r]) return B[0];
}

// Updates an array A of size s according to array B (which stores an
// elementary cellular automaton rule in binary).
// Precondition: s > 0.
void update_arr(bool * A, int s, bool * B)
{
    bool * Ac = new bool[s];
```

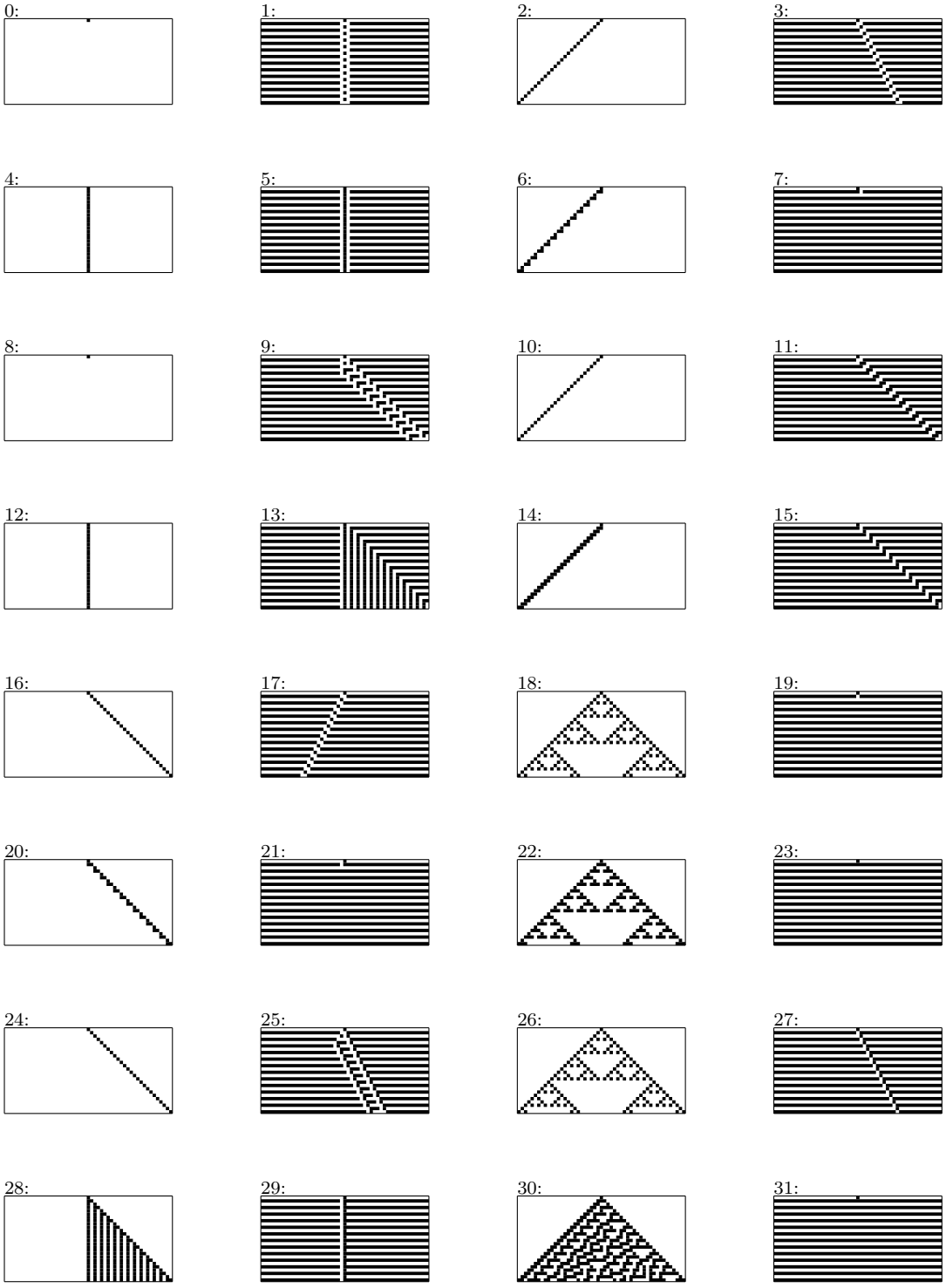
```
    for (int i = 0; i < s; i++)
        Ac[i] = A[i];

    for (int i = 0; i < s; i++)
        A[i] = update_el(i, Ac, s, B);

    delete [] Ac;
}

// Prints instructions in LATEX for a frame of height h and width w to a file.
// Precondition: h, w > 0.
void print_frame(FILE * fd, int h, int w)
{
    fprintf(fd, "%s%i%s%i%s\n", "\\put(0,", 0, "){\\line(1,0){", w, "}}");
    fprintf(fd, "%s%i%s%i%s\n", "\\put(0,", h, "){\\line(1,0){", w, "}}");
    fprintf(fd, "%s%i%s%i%s\n", "\\put(", 0, ",0){\\line(0,1){", h, "}}");
    fprintf(fd, "%s%i%s%i%s\n", "\\put(", w, ",0){\\line(0,1){", h, "}}");
}

// Prints instructions in LATEX for a mesh to a file.
// The mesh should fit in a frame with height h and width w and the
// size of a square is 1 x 1.
// Precondition: h, w > 0.
void print_mesh(FILE * fd, int h, int w)
{
    for (int i = 1; i < h; i++)
        fprintf(fd, "%s%i%s%i%s\n", "\\put(0,", i, "){\\line(1,0){", w, "}}");
    for (int i = 1; i < w; i++)
        fprintf(fd, "%s%i%s%i%s\n", "\\put(", i, ",0){\\line(0,1){", h, "}}");
}

// Prints instructions in LATEX for filling a row of l squares starting with
// square with coordinates x and y of the lower left corner to a file.
void fill_squares(FILE * fd, int x, int y, int l)
{
    fprintf(fd, "%s%i.%i%s\n", "{\\linethickness{", size_i, size_d, "mm}");
    fprintf(fd, "%s%i,%i%s%i%s\n", "\\put(", x, y, ".5){\\line(1,0){",l,"}}}");
}

// Prints instructions in LATEX for an elementary cellular automaton to a file.
```

```
// The rule number (in binary) is stored in B, the number of steps is given by step,
// the result of a step is stored in array A with size s.
// Precondition: Arrays A and B initialized, s > 0.
void print_cell_aut(FILE * fd, bool * A, int s, int step, bool * B)
{
    int l = 0;
    int x0;
    bool pblack = false;

    for (int y = step - 1; y >= 0; y--) {
        for (int x = 0; x < s; x++) {
            if (A[x]) {
                if (l == 0) {
                    x0 = x;
                    pblack = true;
                }
                l++;
            }
            else {
                if (pblack) {
                    fill_squares(fd, x0, y, l);
                    l = 0;
                    pblack = false;
                }
            }
        }
        if (pblack) {
            fill_squares(fd, x0, y, l);
            l = 0;
            pblack = false;
        }
        update_arr(A, s, B);
    }
}

// Prints instructions in LATEX for an elementary cellular automaton to a file.
// The name of the file is given by fn, the rule number of the elementary
// cellular automaton is given by rule, the number of steps is given by height,
// the size of a square is given by size (in tenths of mm), mesh indicates whether
// a mesh should be printed or not.
// Precondition: 0 <= rule <= 255, height > 0, 0 < size < (about) 100.
```

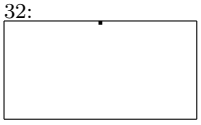
```
void cell_aut(char * fn, int rule, int height, int size, bool mesh)
{
    FILE * fd = fopen(fn, "w+");
    int width = 2 * height - 1;
    bool B[8];
    bool  * A = new bool[width];

    size_i = size / 10;
    size_d = size % 10;

    // convert the rule number to a binary number
    int_binary(rule, B);
    // initialize the cellular automaton
    init_array(A, width);

    if (fd == NULL)
        printf("error in opening file %s\n", fn);
    else
    {
        fprintf(fd, "%s%i.%i%s\n", "\\setlength{\\unitlength}{", size_i, size_d, "mm}");
        fprintf(fd, "%s%i,%i%s\n", "\\begin{picture}(", width, height, ")(0,0)");
        // print frame
        print_frame(fd, height, width);
        // print mesh
        if (mesh) print_mesh(fd, height, width);
        // print cellular automaton
        print_cell_aut(fd, A, width, height, B);
        fprintf(fd, "%s\n", "\\end{picture}");
    }

    fclose(fd);

    delete [] A;
}

void main()
{
    char fn[MAX_FILE_NAME];
    int rule, height, size;
    char cmesh;
    bool mesh = false;
```

```
    printf("write a file name: ");
    scanf("%s", fn);
    printf("write a rule number (0, 1, ..., 255): ");
    scanf("%d", &rule);
    printf("write a step number (> 0): ");
    scanf("%d", &height);
    printf("write a size number (1, 2, ..., 100): ");
    scanf("%d", &size);
    printf("mesh? (y/n): ");
    scanf("\n%c", &cmesh);
    if (cmesh == 'y') mesh = true;

    if (rule >= 0 && rule <= 255 && height > 0 && size >= 1 && size <= 100)
        cell_aut(fn, rule, height, size, mesh);
    else
        printf("something wrong with input data\n");
}
```

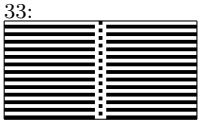
B Elementary cellular automata



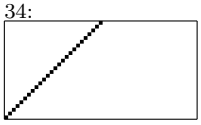
32:



33:

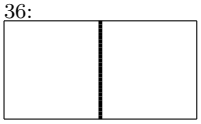


34:

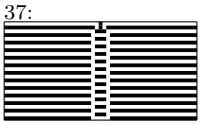


35:

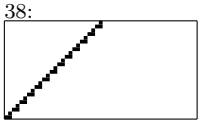
36:



37:

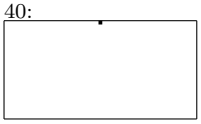


38:



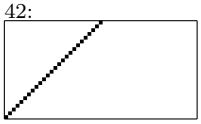
39:

40:

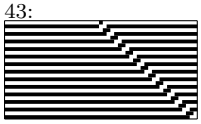


41:

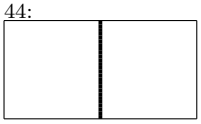
42:



43:

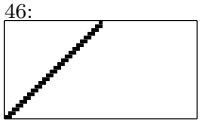


44:

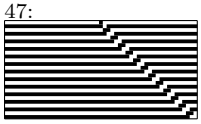


45:

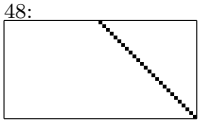
46:



47:

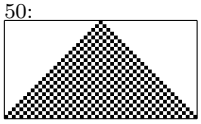


48:

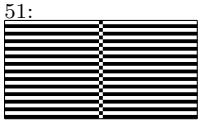


49:

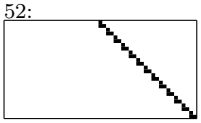
50:



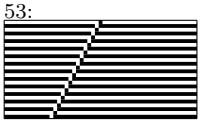
51:



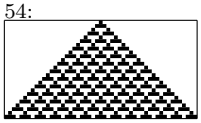
52:



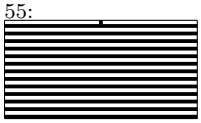
53:



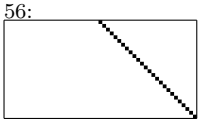
54:



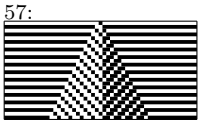
55:



56:



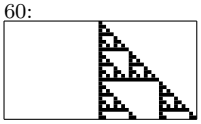
57:



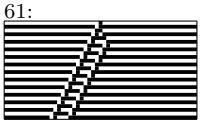
58:

59:

60:



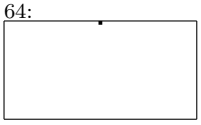
61:



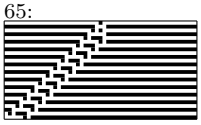
62:

63:

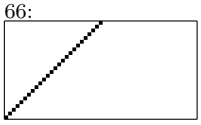
64:



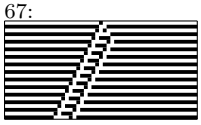
65:



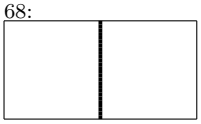
66:



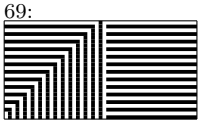
67:



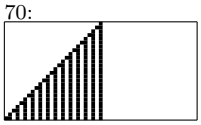
68:



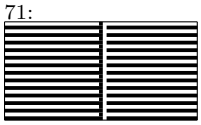
69:



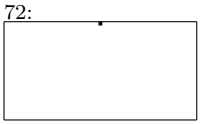
70:



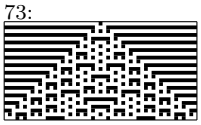
71:



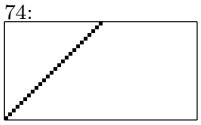
72:



73:

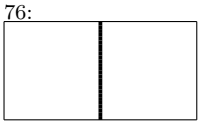


74:

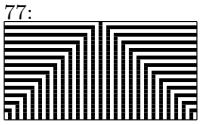


75:

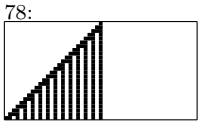
76:



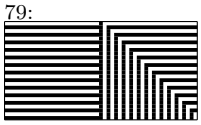
77:



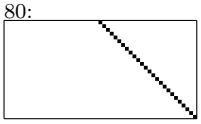
78:



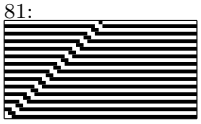
79:



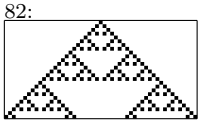
80:



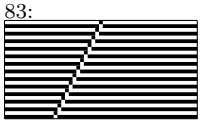
81:



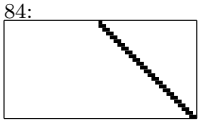
82:



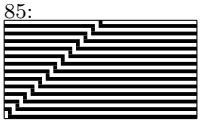
83:



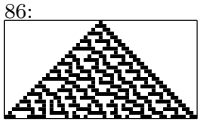
84:



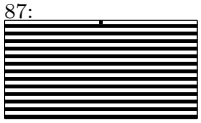
85:



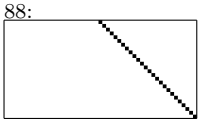
86:



87:

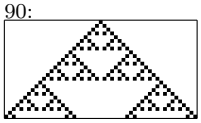


88:

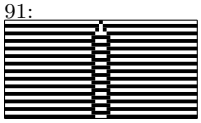


89:

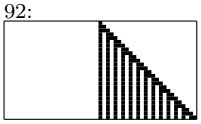
90:



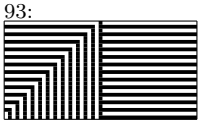
91:



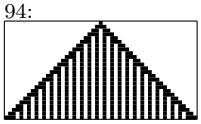
92:



93:

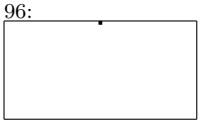


94:



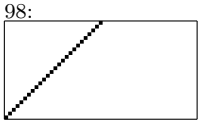
95:

96:

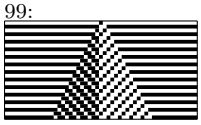


97:

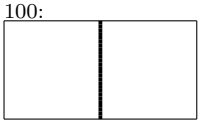
98:



99:

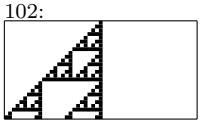


100:



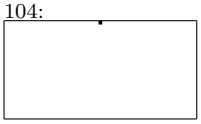
101:

102:

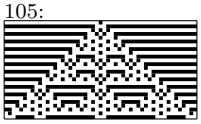


103:

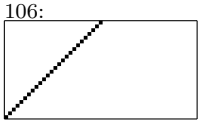
104:



105:



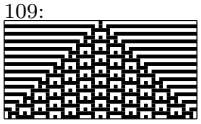
106:



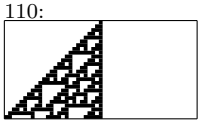
107:

108:

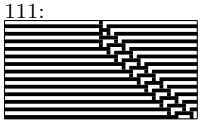
109:



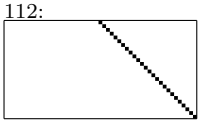
110:



111:

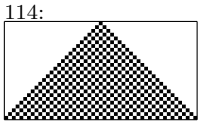


112:

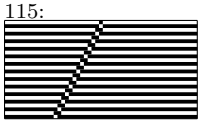


113:

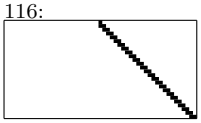
114:



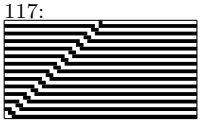
115:



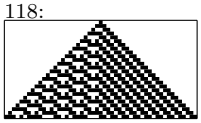
116:



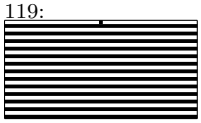
117:



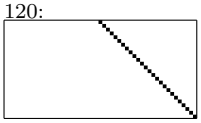
118:



119:

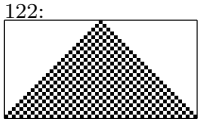


120:

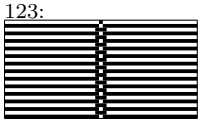


121:

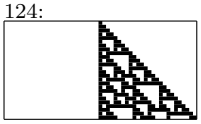
122:



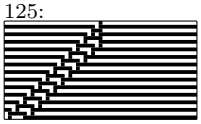
123:



124:

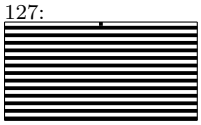


125:

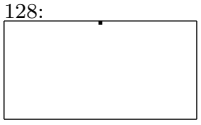


126:

127:

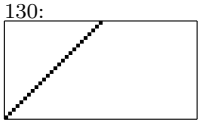


128:

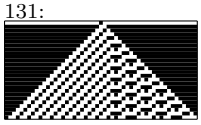


129:

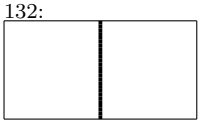
130:



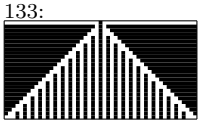
131:



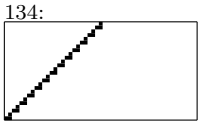
132:



133:

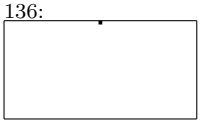


134:



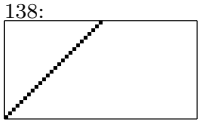
135:

136:

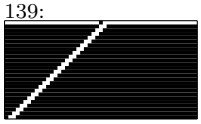


137:

138:

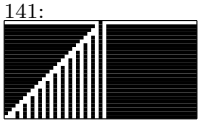


139:

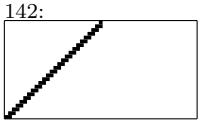


140:

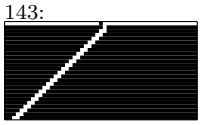
141:



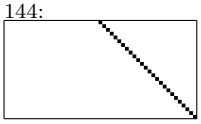
142:



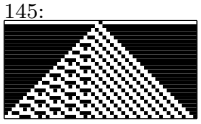
143:



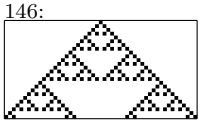
144:



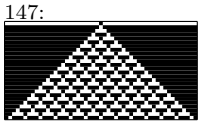
145:



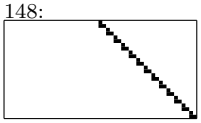
146:



147:

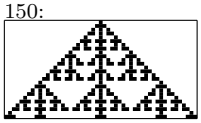


148:

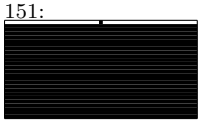


149:

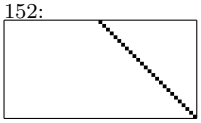
150:



151:

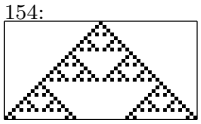


152:

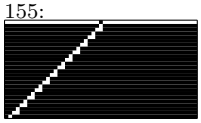


153:

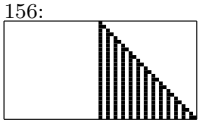
154:



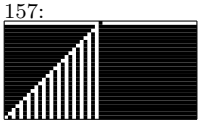
155:



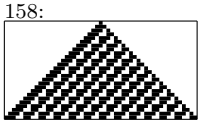
156:



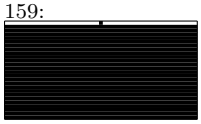
157:



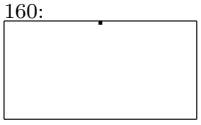
158:



159:

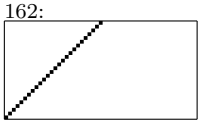


160:

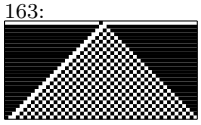


161:

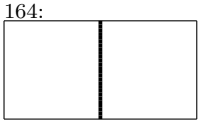
162:



163:

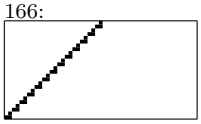


164:



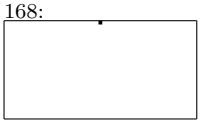
165:

166:



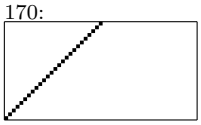
167:

168:

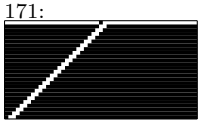


169:

170:

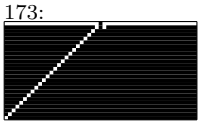


171:

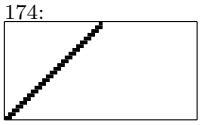


172:

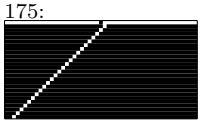
173:



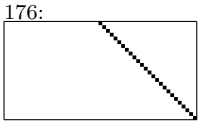
174:



175:

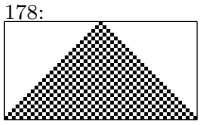


176:

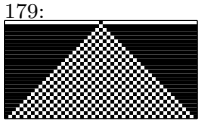


177:

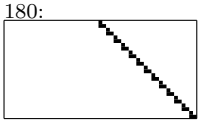
178:



179:

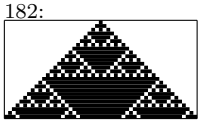


180:

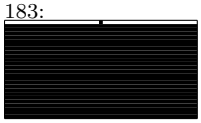


181:

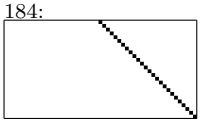
182:



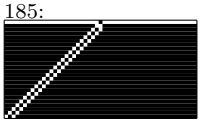
183:



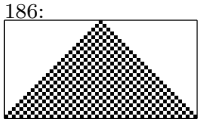
184:



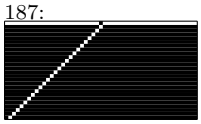
185:



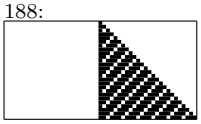
186:



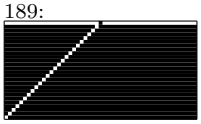
187:



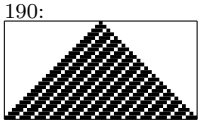
188:



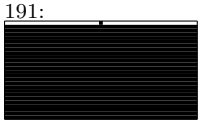
189:



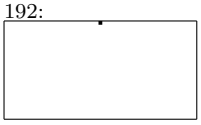
190:



191:

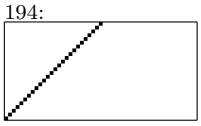


192:



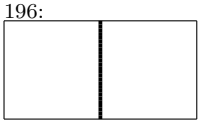
193:

194:

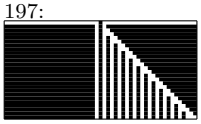


195:

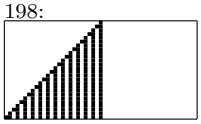
196:



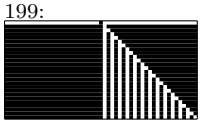
197:



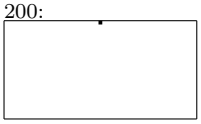
198:



199:

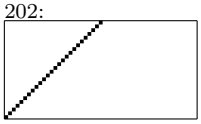


200:



201:

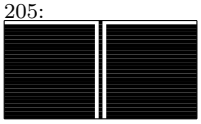
202:



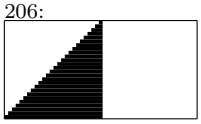
203:

204:

205:

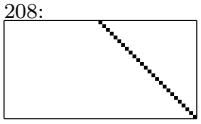


206:

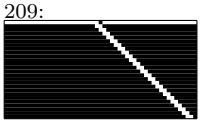


207:

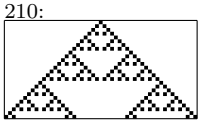
208:



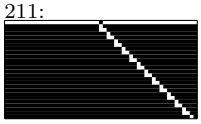
209:



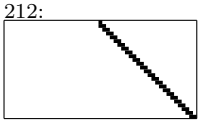
210:



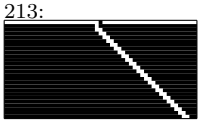
211:



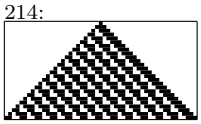
212:



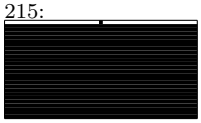
213:



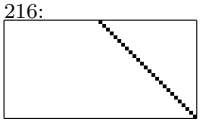
214:



215:

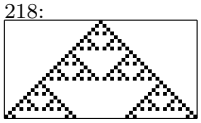


216:

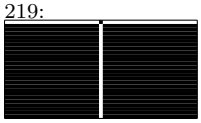


217:

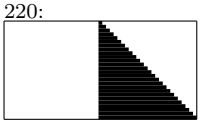
218:



219:

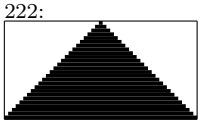


220:

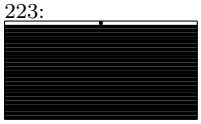


221:

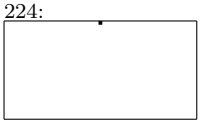
222:



223:

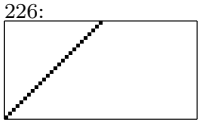


224:



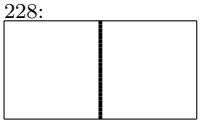
225:

226:

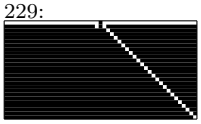


227:

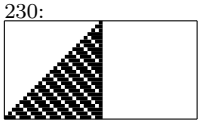
228:



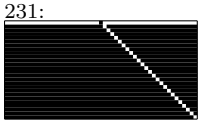
229:



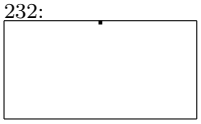
230:



231:

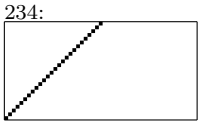


232:



233:

234:

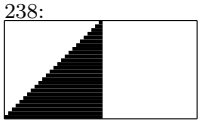


235:

236:

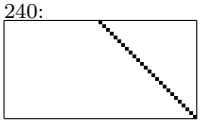
237:

238:

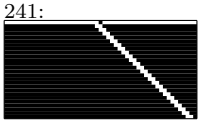


239:

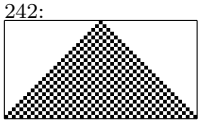
240:



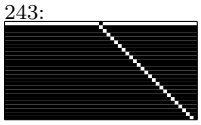
241:



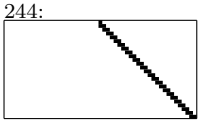
242:



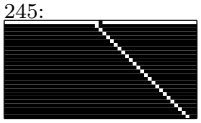
243:



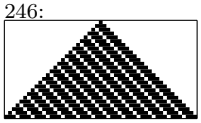
244:



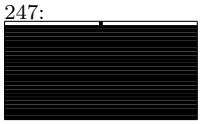
245:



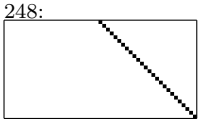
246:



247:

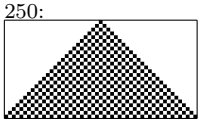


248:

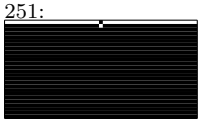


249:

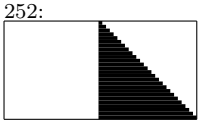
250:



251:

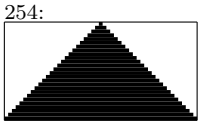


252:

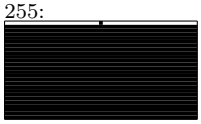


253:

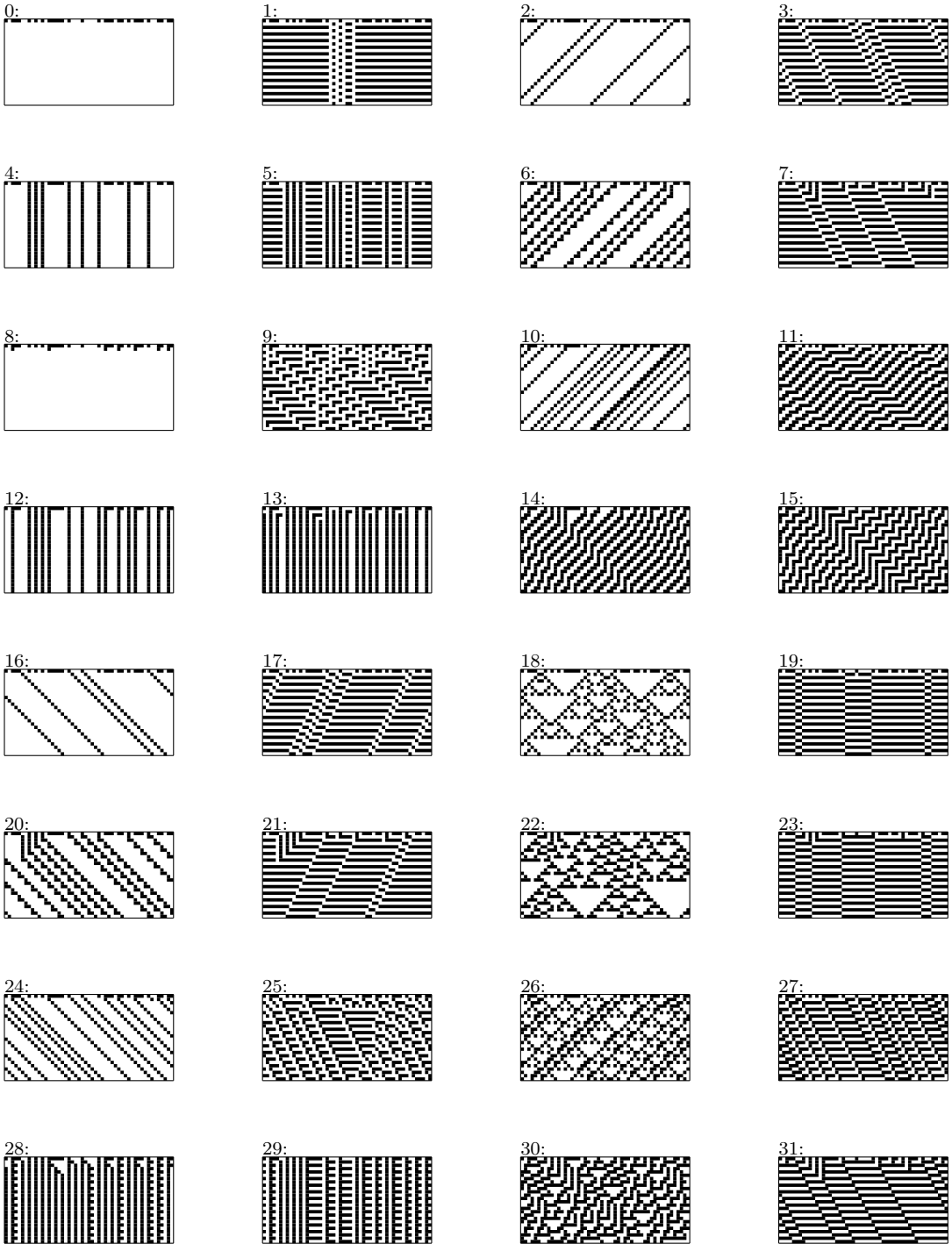
254:



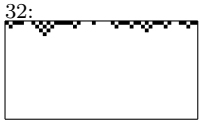
255:



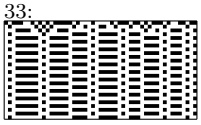
C Elementary cellular automata with random initial configurations



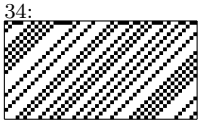
32:



33:



34:

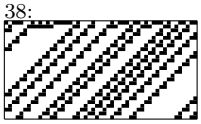


35:

36:

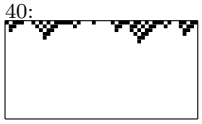
37:

38:



39:

40:



41:

42:

43:

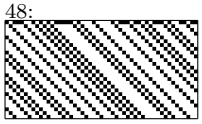
44:

45:

46:

47:

48:

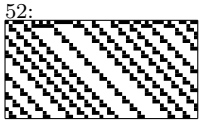


49:

50:

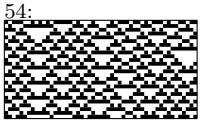
51:

52:



53:

54:



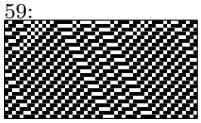
55:

56:

57:

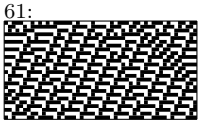
58:

59:



60:

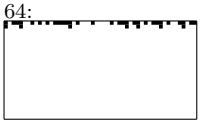
61:



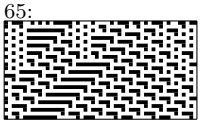
62:

63:

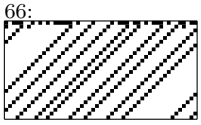
64:



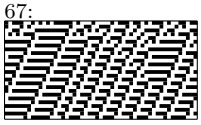
65:



66:

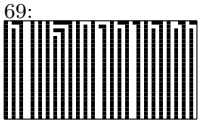


67:

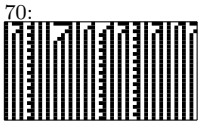


68:

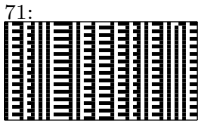
69:



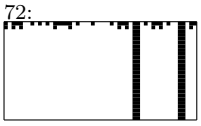
70:



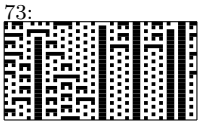
71:



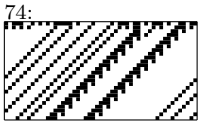
72:



73:



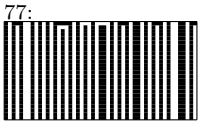
74:



75:

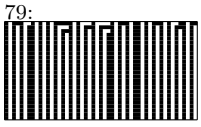
76:

77:



78:

79:



80:

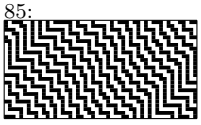
81:

82:

83:

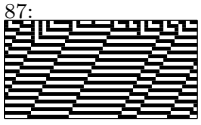
84:

85:

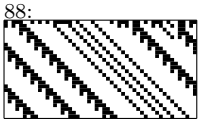


86:

87:



88:



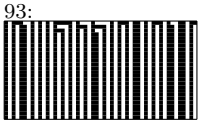
89:

90:

91:

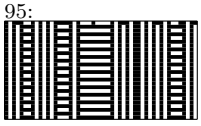
92:

93:

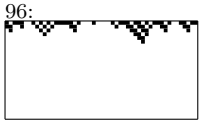


94:

95:



96:

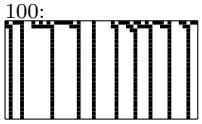


97:

98:

99:

100:

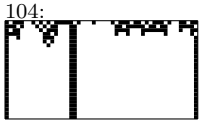


101:

102:

103:

104:

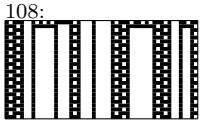


105:

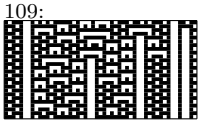
106:

107:

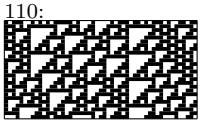
108:



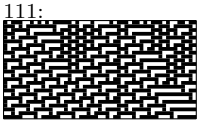
109:



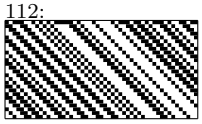
110:



111:



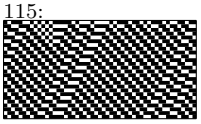
112:



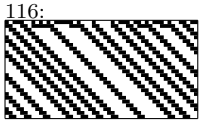
113:

114:

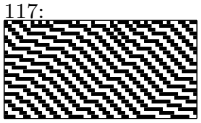
115:



116:

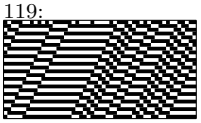


117:



118:

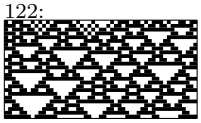
119:



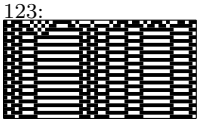
120:

121:

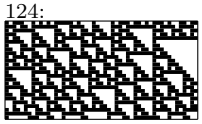
122:



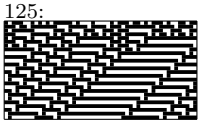
123:



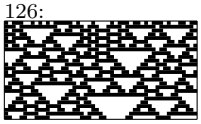
124:



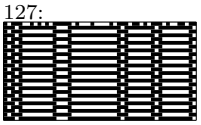
125:



126:



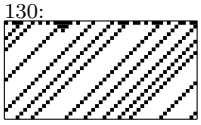
127:



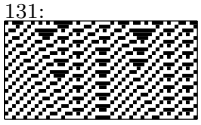
128:

129:

130:

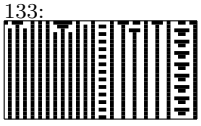


131:

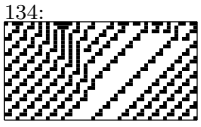


132:

133:



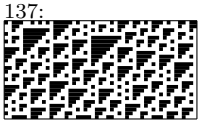
134:



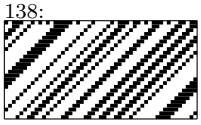
135:

136:

137:

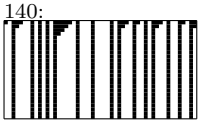


138:



139:

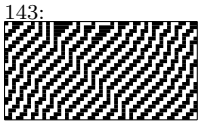
140:



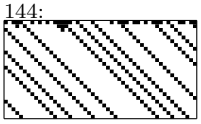
141:

142:

143:



144:

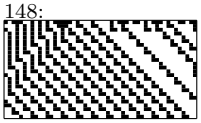


145:

146:

147:

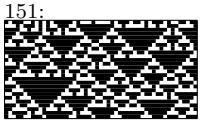
148:



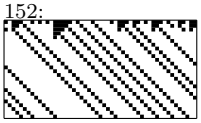
149:

150:

151:



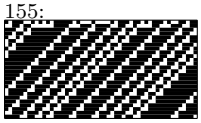
152:



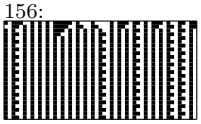
153:

154:

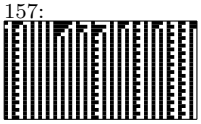
155:



156:



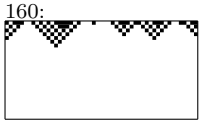
157:



158:

159:

160:

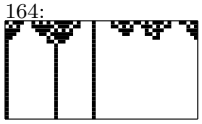


161:

162:

163:

164:



165:

166:

167:

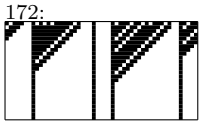
168:

169:

170:

171:

172:



173:

174:

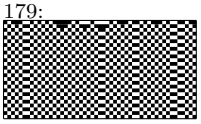
175:

176:

177:

178:

179:

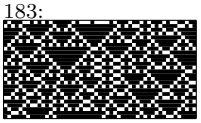


180:

181:

182:

183:



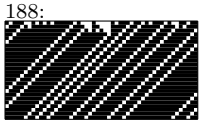
184:

185:

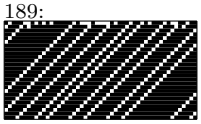
186:

187:

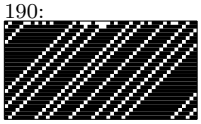
188:



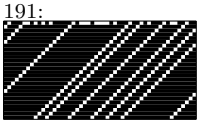
189:



190:



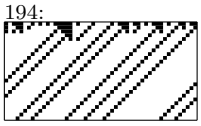
191:



192:

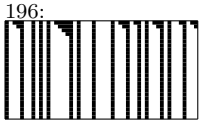
193:

194:



195:

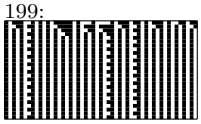
196:



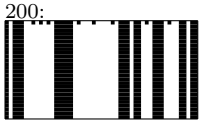
197:

198:

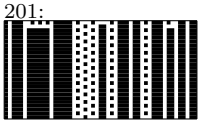
199:



200:

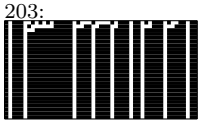


201:

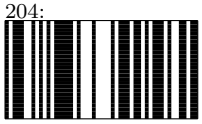


202:

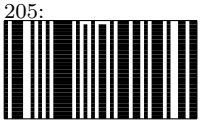
203:



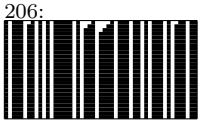
204:



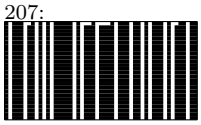
205:



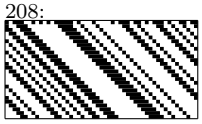
206:



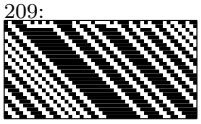
207:



208:

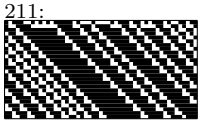


209:



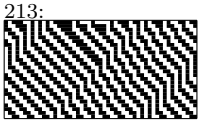
210:

211:

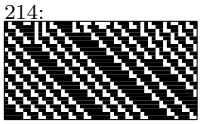


212:

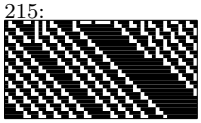
213:



214:

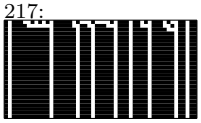


215:



216:

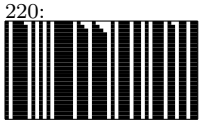
217:



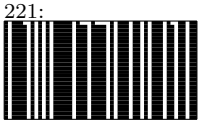
218:

219:

220:

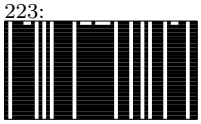


221:



222:

223:



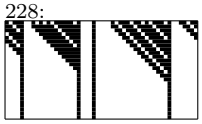
224:

225:

226:

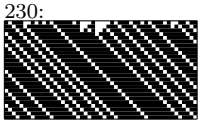
227:

228:

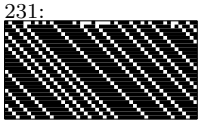


229:

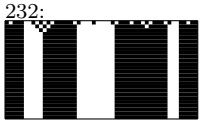
230:



231:



232:

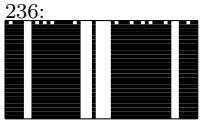


233:

234:

235:

236:



237:

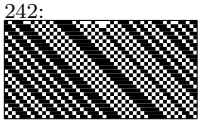
238:

239:

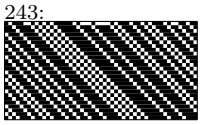
240:

241:

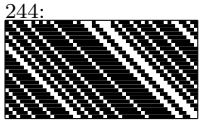
242:



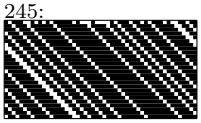
243:



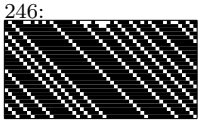
244:



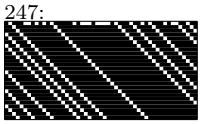
245:



246:



247:



248:

249:

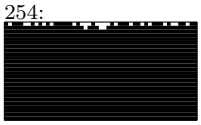
250:

251:

252:

253:

254:



255: