2 Pattern formation

God used beautiful mathematics in creating the world.
Paul Dirac

In chapter two we will look at some examples illustrating basic principles of pattern formation in natural and artificial systems such as cellular automata, Lindenmayer systems (L-systems), and fractals. We will see that complex patterns can emerge from simple rules applicable to individual cells and through local interactions between these cells. We will see that two important requisites for pattern formation are: (a) the availability of many cells, and (b) the parallel and concurrent processing of many "simple" rules. One important consequence will be that there is no need for central control.

2.1 Cellular automata

Cellular automata are examples of the large class of so-called complex systems. Complex Systems are dynamical systems that exhibit overall behavior that cannot directly be traced back to the underlying rules, that is, emergent or self-organized behavior. Complex systems typically consist of many similar, interacting, simple parts. 'Simple' means that the behavior of parts is easily understood, while the overall behavior of the system as a whole has no simple explanation. But often this emergent behavior has much simpler features than the detailed behavior of individual parts.

Introduction to Cellular Automata

Cellular automata (CA) are mathematical models in which space and time are discrete. Time proceeds in steps and space is represented as a lattice or array of cells (see Figures 2.1 and 2.2). The size of this lattice is called the dimension of the CA. The cells have a set of properties (variables) that may change over time. The values of the variables of a specific cell at a given time are called the state of the cell and the state of all cells together form (as a vector or matrix for example) the global state or global configuration of the CA.

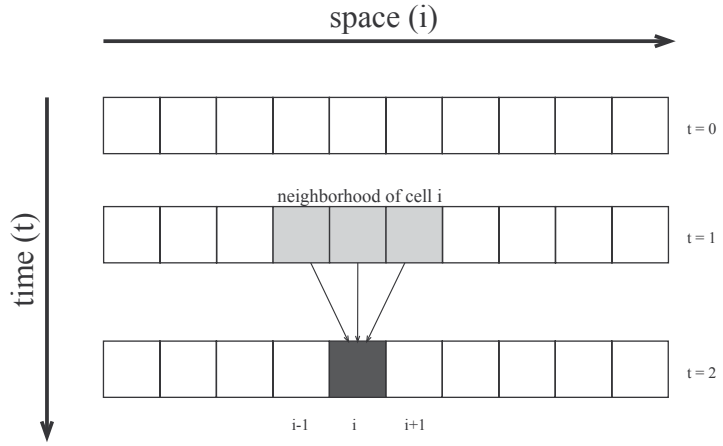


Figure 2.1: *Space and time in a 1-dimensional CA.*

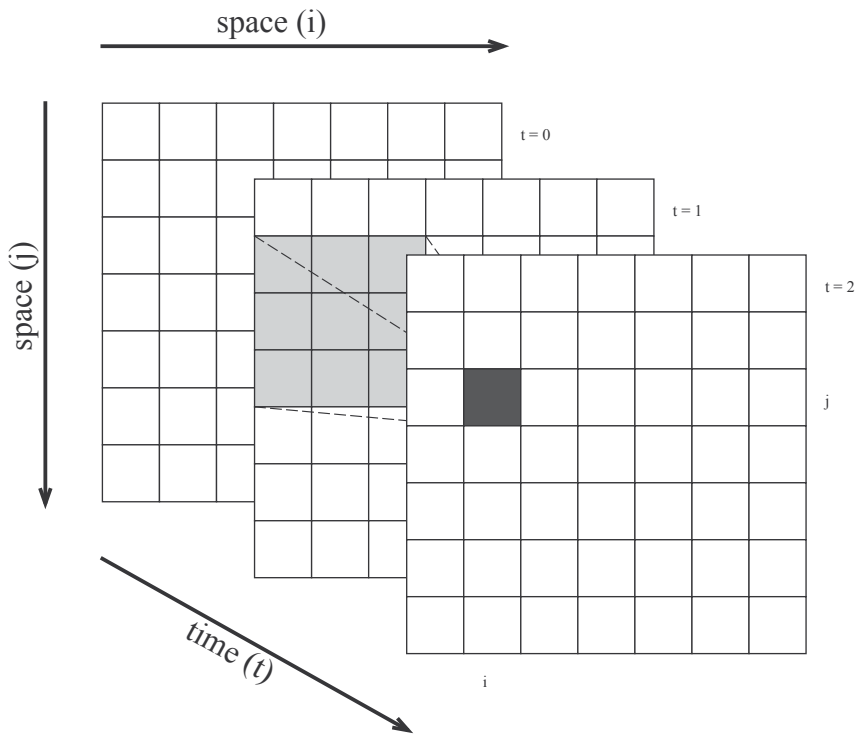


Figure 2.2: *Space and time in a 2-dimensional CA.*

We will consider only 1 and 2-dimensional CA. But the concept can be extended easily to any higher dimensional spaces. In Table 2.1 the mathematical notation for 1 and 2-dimensional CA is summarized. Typically the state variables have discrete values. Also, a CA is discrete in time, discrete in space and therefore perfectly suited for simulation on a computer.

Table 2.1: *Mathematical notation for 1- and 2-dimensional CA.*

symbol	*Meaning*
t	Time
Δt	time step, typically 1
$a_i(t)$	state of cell at position i at time t (1 dim.)
$a_{ij}(t)$	state of cell at position (i,j) at time t (2 dim.)
$A(t)$	global state of the CA at time t

Local Rules

Each cell has a set of local rules. Given the state of the cell and the states of the cells in its neighborhood these rules determine the state of the cell at the next time step. These rules are local in two senses: First each cell has its own set of local rules and second the future state of the cell only depends on the neighbors of this cell. It is important to note that the states of all cells are updated simultaneously (synchronously)

based on the (actual or current) values of the variables in their neighborhood according to the local rules. If all cells have the same set of rules the CA is called homogeneous. We will consider only homogeneous CA.

The lattice of a CA can either be finite or infinite. Typically (especially if the CA is simulated on a computer) it is finite. Infinite lattices are mainly of mathematical interest. Infinite lattices have no borders, whereas on finite lattices one has to define what happens at the borders of the lattice, that is one has to define boundary conditions. The problem is that cells at the borders have only incomplete neighborhoods. There are three straightforward possibilities to solve this problem, either to assume that there are "invisible" cells next to the border-cells, which are in a given predefined state, (FIXED boundary), that the cells on the edge do not diffuse out of the system and only diffuse inwards (REFLECTIVE boundary), or to assume that the cells on the edge are neighbors of the cells on the opposite edge, (PERIODIC boundary), as depicted in Figure 2.3.

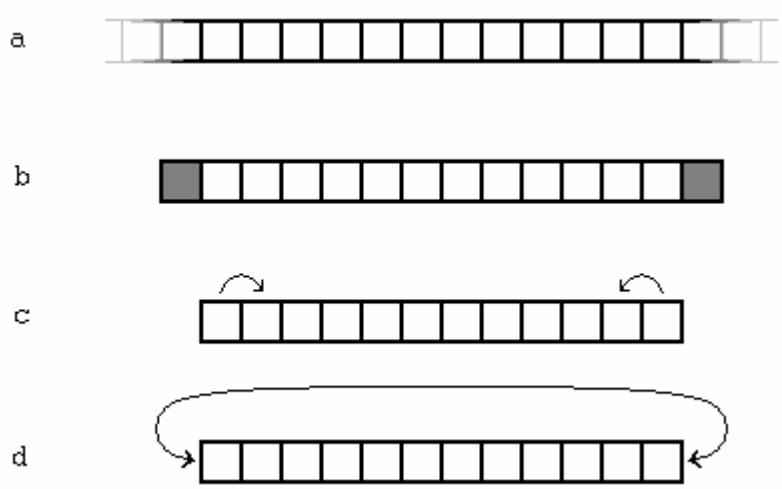


Figure 2.3: *Four possibilities for boundary conditions in a 1-dimensional CA. a) infinite (unbounded) array of cells. b) finite array of cells with fixed boundaries. The end points have cells in their neighborhood with a fixed value. c) finite array of cells with reflective boundary. The leftmost cell can only diffuse to the right. d): finite array of cells closed to a circle, periodic boundary. The leftmost cell becomes a neighbor of the rightmost cell.*

The initial values of all the state variables are referred to as the initial conditions. Starting from these initial conditions the CA evolves in time, changing the states of the cells according to the local rules. The evolution of the CA from its initial conditions is uniquely defined by the local rules, as long as they are deterministic (we will only consider deterministic rules). Thus, CA are deterministic systems whose behavior results from local rules. Cells that are not neighbors do not directly affect each other. CA have no memory in the sense that the actual state alone (and no other previous state) determines the next state. Because the rules and the states of the cells are local, any global pattern that might evolve is thus emergent.

Applications

CA have been used for a wide variety of purposes. For example: for modeling nonlinear chemical systems (Greenberg et al., 1978) and the evolution of spiral galaxies (Gerola and Seiden, 1978; Schewe, 1981). In

these two cases the lattice of cells in the CA corresponds directly to the physical space of the modeled system.

Any physical system satisfying (partial) differential equations may be approximated by a CA by discretisation of space, time, and state variables. Physical systems consisting of many discrete elements with local interaction are especially well suited to being modeled as CA. Also biological and social systems can often conveniently be modeled as CA.

CA and AL

CA are good examples of the paradigms of AL: complex systems made of similar (or identical) entities and local rules, parallel computation and thus local determination of behavior.

In the next few sections we will encounter some simple examples of 1-dimensional CA and explore the terms and concepts introduced. In section 2.2 we will see an example of a 2-dimensional CA.

1-dimensional Cellular Automata

Let's start with some very simple CA: a 1-dimensional CA with one variable at each cell taking only k possible values, say 0, 1, …, k-1. The value of the cell at position i at time t is denoted as $a_i(t)$. We will assume that the neighborhood consists always of the r nearest neighbors on each side[1] and the cell itself, thus the neighborhood consists of $2r+1$ cells. Each cell updates its state at every time step according to some set of rules Φ, and thus

$$a_i(t+1) = \Phi[a_{i-r}(t), a_{i-r+1}(t), \mathrm{K}\ , a_{i+r-1}(t), a_{i+r}(t)]$$

Let's look at a simple example. Assume that we have 256 cells in a row, and that each cell can take the values 0 or 1. Each cell updates its state depending on its own state and the state of its two immediate neighbors according to the following rule table:

Table 2.2: *Example of a simple local rule (rule table).*

$a_{i-1}(t)$	$a_i(t)$	$a_{i+1}(t)$	$a_i(t+1)$
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

This rule can be re written in a much more compact form using predicate calculus

$$a_i(t+1) = a_{i-1}(t) \oplus a_i(t) \oplus a_{i+1}(t)$$

where $\oplus$ denotes addition modulo 2 (XOR). The graphical representation shown in Figure 2.4 is much more intuitive. We will assume that the row of cells of the CA is closed to a circle (see Figure 2.3), thus the leftmost cell is the neighbor of the rightmost cell.

[1] r: radius of neighbors

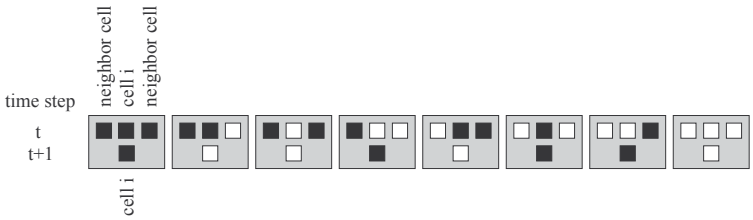


Figure 2.4: *Graphical representation of a CA rule. The top row (in the gray boxes) corresponds to the configuration of a cell and its immediate neighbors. In our example there are eight possible configurations of cells (3 cells, 2 states (on, off) each). For each of these eight configurations the bottom row specifies the state of the cell in the next time step.*

Figure 2.5 shows the pattern that is generated by the above rule when we start with one black cell in the middle of the CA array. (Remember that the rows correspond to the CA cells at subsequent time steps). Note the self-similarity of the patterns[2]. Although this figure is not a fractal[3] in the strict sense (because it has no infinitely fine structures) it is indeed very fractal-like. You can imagine that in an infinite CA array this pattern would grow forever, thereby generating bigger and bigger triangles, and repeat the patterns it has generated before. A very different picture is observed when we start the same CA (with the same rules) from a random initial configuration (Figure 2.6). Note that the regular pattern observed before has gone. Still, the pattern is not a random one. Triangles and other structures appear over and over again, although at irregular times and at unforeseen places.

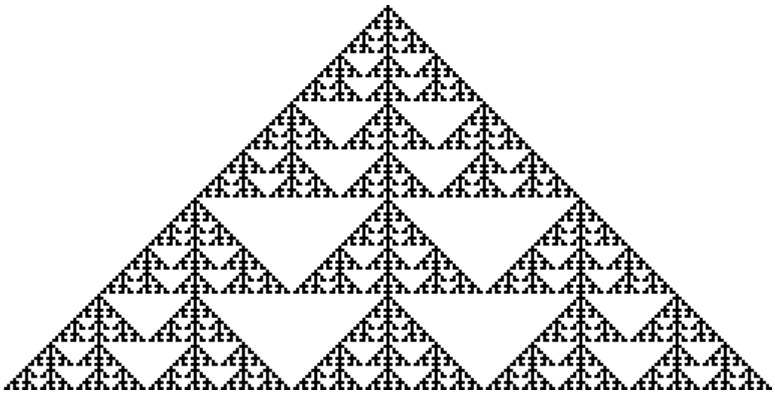


Figure 2.5: *Pattern generated by 1-dimensional CA. The pattern is generated by the 1-dimensional CA rule introduced in the text. The top row corresponds to the initial configuration (one black cell in the middle) and the bottom one to the state of the CA after 128 time steps. Note the self-similarity of the patterns.*

[2] Self-similarity at multiple levels is a key feature of fractality (see section 2.4).
[3] see section 2.4

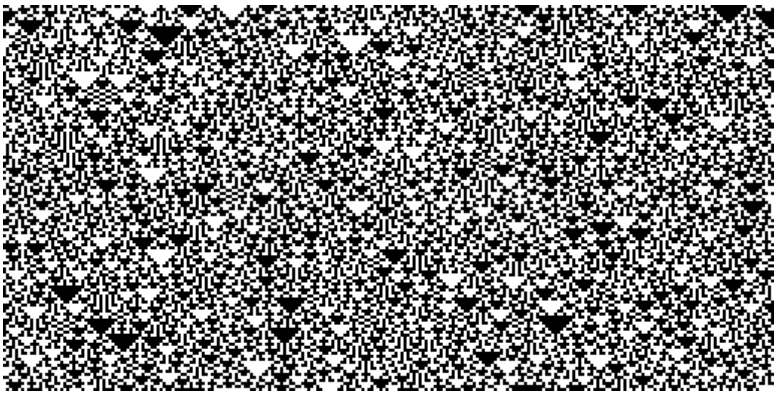


Figure 2.6: *Pattern generated by a 1-dimensional CA. The same CA as in Figure 2.5 is used, but in contrast to Figure 2.5, the initial configuration (top row) is a random one. At first sight the patterns seems random too but at closer inspection many small structures that appear over and over again can be discerned.*

We conclude from these two examples that even such simple deterministic systems as 1-dimensional CA can produce astonishingly complex patterns. These patterns are very regular if the initial configuration is regular too (Figure 2.5). If the initial configuration is random the generated pattern is much less regular (Figure 2.6). In both cases the patterns are complex but they reveal simple higher-level structures, the triangles. Note that by simply looking at the rules (Figure 2.4) it is not at all obvious that such triangles will emerge. It is very common to find emergent structures in CA.

Number of Possible CA Rules

There are many different possibilities for CA rules. In the previous we had two states per cell, and three neighbors. Therefore there are $2^3 = 8$ entries in the rule table of a CA (the top row of Figure 2.4) and thus $2^8 = 256$ possible rule tables. So there are 256 different possible cellular automata of this type. This number grows exponentially if we increase the number of states k per cell and the range r of the neighborhood (or the number of neighbors $2r + 1$). For k states per cell and $2r + 1$ neighbors we have

$$k^{2r+1}$$

entries in the rule table and

$$k^{k^{2r+1}}$$

possibilities for rule tables or CA. For $k = 10$ and $r = 5$ we have 10^{11} entries in the rule table and $10^{100'000'000'000}$ different possible CA. To put this number into perspective, there are only about 10^{80} molecules in the universe. Thus we will never be able to examine all or even a significant fraction of all possible CA.

The Four Classes of Cellular Automata

In this section we will consider again 1-dimensional CA with 256 cells. Each cell can take the values 0 or 1 ($k = 2$). But this time we will use neighborhoods of a varying number of cells ($r = 1$; $r = 2$, that is the cell itself and the two nearest neighbors on each side; $r = 3$). Although these types of CA are, again, very simple they exhibit a wide variety of qualitatively different phenomena. In Figure 2.7 typical examples of the

evolution of such cellular automata from random initial conditions are depicted. Structures of different quality and complexity are formed. Wolfram (1984a) divided the CA rules into four classes; according to what quality of structures they give rise. These are:

Class I: Tends to a spatially homogeneous state (all cells are in the same state). Patterns disappear with time.

Class II: Yields a sequence of simple stable or periodic structures (endless cycle of same states). Pattern evolves to a fixed finite size.

Class III: Exhibits chaotic aperiodic behavior. Pattern grows indefinitely at a fixed rate.

Class IV: Yields complicated localized structures, some propagating. Pattern grows and contracts with time.

The classes II and III correspond to the different types of attractors:

Class II: Point attractor or periodic attractor.

Class III: Strange or chaotic attractor.

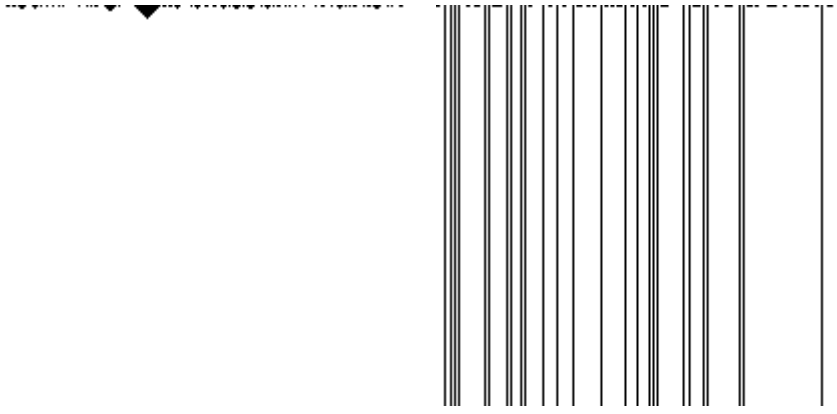
The four classes can also be distinguished by the effects of small changes in the initial conditions (Wolfram, 1984a):

Class I: No change in final state.

Class II: Changes only in a region of finite size.

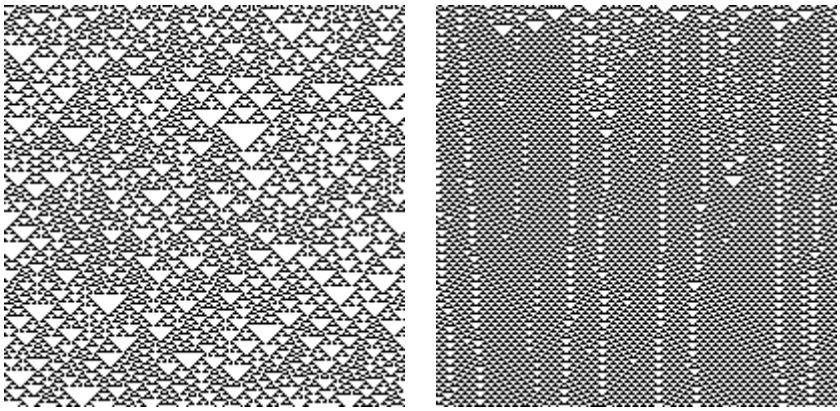
Class III: Changes over a region of ever-increasing size.

Class IV: Irregular changes.



Class I: empty (rule 128[4]) Class II: stable or periodic (rule 4[5])

[4] Rule 128: '111' goes to '1', else '0'.
[5] Rule 4: '010' goes to '1', else '0'.



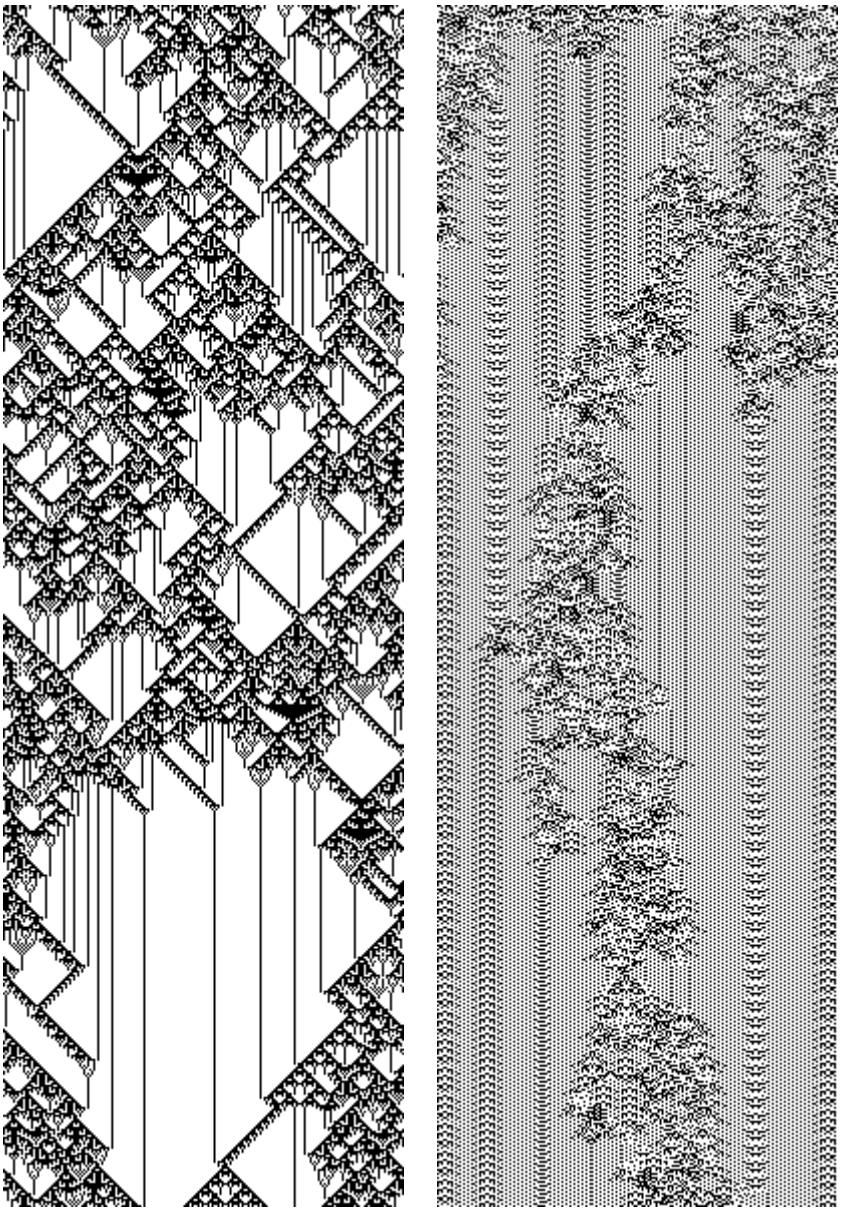
Class III: chaotic (rule 22[6]) Class IV: complex (rule 54[7])

Figure 2.7: *Typical examples of CA (k = 2, r = 1) starting from a random initial configuration. Depicted here are examples of the four classes of CA as introduced by Wolfram (1984a).*

Figures 2.8 and 2.9 show the behavior of class IV CA. Their behavior is difficult to describe. It is not regular, not periodic, but also not random. It contains a bit of each. Class IV CA remain at the boundary between periodicity and chaos. Moreover, the behavior is not predictable without explicit calculation. That is very little information on the behavior of a class IV CA can be deduced directly from properties of its rules.

> *It seems likely, in fact, that the consequences of infinite evolution in many dynamical systems may not be described in finite mathematical terms, so that many questions concerning their limiting behavior cannot be formally decided. Many features of the behavior of such systems may be determined effectively only by explicit simulation: no general predictions are possible. (Wolfram, 1984a, p. 23)*

[6] Rule 22: ‘001’, ‘100’, and ‘010’ go to ‘1’, else ‘0’.
[7] Rule 54: ‘001’, ‘100’, ‘010’, and ‘101’ go to ‘1’, else ‘0’.



$k = 2, r = 2$ $k = 2, r = 3$

Figure 2.8: *Two examples of class IV CA.*

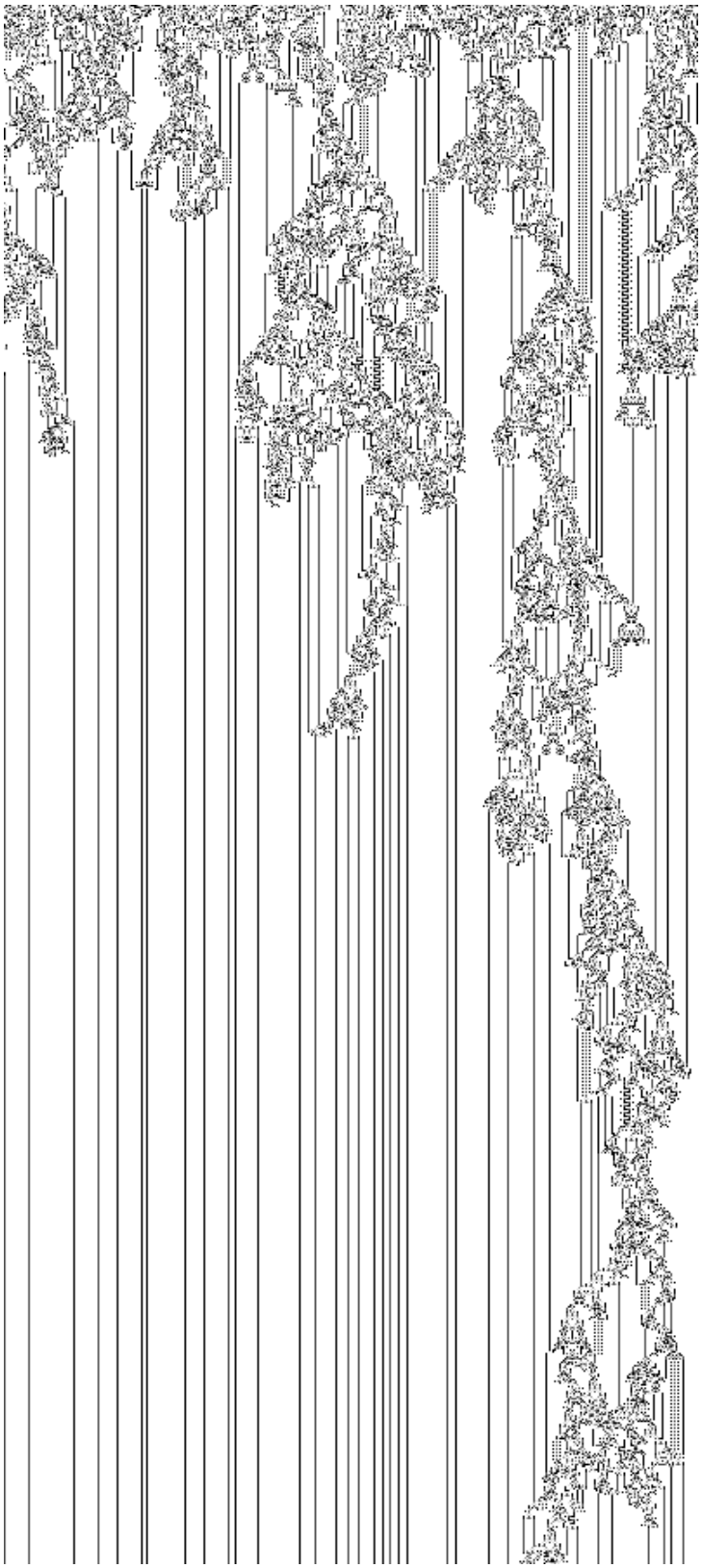


Figure 2.9: *Another example of a class IV CA (k=5, r=2).*

2.2 Game of Life

In the late 1960s John Conway, motivated by the work of von Neumann, used simple 2-dimensional CA which he called the "game of life." Each cell has two possible states 0 or 1 (or "dead" and "alive", thus the name of the CA), and a very simple set of rules (Flake, 1998):

Loneliness: If a live cell has less than two neighbors, then it dies.

Overcrowding: If a live cell has more that three neighbors, then it dies.

Reproduction: If a dead cell has three live neighbors, then it comes to life.

Stasis: Otherwise, a cell stays as it is.

In 1970 Martin Gardner described the Game of Life and Conway's work in Scientific American (Gardner, 1970). This article inspired many people around the world to experiment with Conway's CA. Many interesting configurations were found. We will encounter some of them in the following discussion.

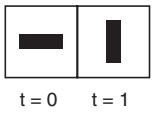
Patterns in the Game of Life are usually characterized by their behavior. There are several categories (of increasing complexity)[8]:

Type I (still-lives): Patterns that do not change; that are static.

Examples:

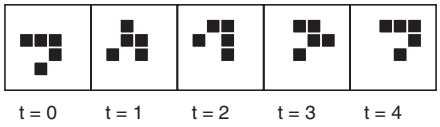
Type II (oscillators): Patterns that repeat themselves after a fixed sequence of states and return to their original state; periodic patterns.

The 'blinker' is an example of a period-2 oscillator:



Type III (spaceships): Patterns that repeat themselves after a fixed sequence of states and return to their original state, but translated in space; patterns that move at a constant velocity.

The 'glider' is one simple example:

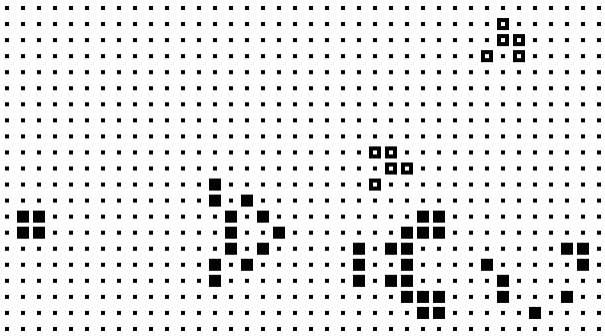


Type IV: Patterns that constantly increase in population size (living cells).

[8] For exhaustive collection of life patterns and animations see for example home.interserv.com/~mniemiec/lifeterm.htm.

Type IVa (guns): Oscillators that emit spaceships in each cycle.

Example: A glider gun (black squares) that emits gliders (empty squares):



Type IVb (puffers): Spaceships that leave behind still-life, oscillators, and/or spaceships.

Type IVc (breeders): Patterns that increase their population size quadratically (or even faster). For example, a breeder may be a spaceship that emits glider guns.

Type V (unstable): Patterns that evolve through a sequence of states, which never return to the original state. Small patterns that last a long time before stabilizing are called "Methuselahs".

Again, the message is that despite the simplicity of the rules, amazingly complex and sophisticated structures can emerge in the Game of Life.

Universal Computation

Universal Computation means that there is the capability of computing anything that can be computed. A universal computer, i.e. a computer capable of universal computation, can do so. The best-known example of such a universal computer is a Turing Machine, an imaginary machine proposed in 1936 by Alan Turing, an English mathematician (Turing, 1936). A Turing Machine has a read/write head mounted to a tape of infinite length, i.e. consisting of an infinite number of cells. The action performed by the head (read, write, move forward, move backward or no action/movement) depends on the current state of the head and of the cell underneath. Due to the infinite length of the tape and the lack of any limitations regarding the number of possible states, the Turing machine can solve every computable problem and is capable of universal computation.

Looking again at the Game of Life from a computational point of view we can say that type I objects, i.e. static objects, can be seen as a kind of memory needed in every computer; type II objects, i.e. periodic patterns, can fulfill the task of counting or synchronizing parallel processes and type III objects which repeat themselves regularly but move in space are required for information flow in a computer, thus the Game of Life includes the basic elements necessary for a computer. Through repeated collisions of moving objects with static objects the latter get altered and increase in size, i.e. new objects are created. Such process of recursively assembling pieces to make larger and more complicated objects can be carried to the extreme of building a self-reproducing machine (Flake, 1999). Based on the above and since operations in computers are usually implemented in terms of logical primitives (AND, OR, NOT) we can say that it is possible to build a general-purpose computer in the Game of Life and that it can emulate any Turing

machine. As a consequence the Game of Life is unpredictable (in the same sense as the Class IV CA of Wolfram, see above):

> *"There are important limitations on predictions, which may be made for the behavior of systems capable of universal computation. The behavior of such systems may in general be determined in detail essentially only by explicit simulation. […] No finite algorithm or procedure may be devised capable of predicting detailed behavior in a computationally universal system. Hence, for example, no general finite algorithm can predict whether a particular initial configuration in a computationally universal cellular automaton will evolve to the null configuration after a finite time, or will generate persistent structures, so that sites with nonzero values will exist at arbitrarily large times. (This is analogous to the insolubility of the halting problem for universal Turing machines [see for example Beckmann, 1980].) (Wolfram, 1984b, p. 31)"*

Another way in which sophisticated structures can emerge from sets of simple rules are the so-called Lindenmeyer Systems.

2.3 Lindenmeyer systems

In 1968 the biologist Aristid Lindenmeyer invented a mathematical formalism for modeling the growth of plants. This formalism, known as Lindenmeyer system or L-system, is essentially a traditional production system. Productions, or rewriting rules, are rules, which state how new symbols or cells grow from old symbols, or cells. A production system as a whole states how at each time step its production rules are applied to symbols in such a way that as many old symbols as possible are simultaneously substituted by new symbols.

Consider the following L-system as a simple example:

Axiom: *B* *(starting cell or starting seed of the L-system)*

Rule 1: $B \rightarrow F[-B] + B$

Rule 2: $F \rightarrow FF$

If we like, we can interpret the effect of the individual rules in this rule system as follows.

Axiom: Initially, we start with a lone *B*-cell (see Figure 2.10).

Rule 1: Each *B* cell divides, producing an *F* cell and two *B* cells arranged as depicted in figure 2.10. The brackets and the "+" and "-" signs indicate the arrangement of the cells ("+" rotate right, "-" rotate left).

Rule 2: Each *F* cell divides, producing two *F* cells arranged as shown in Figure 2.10.

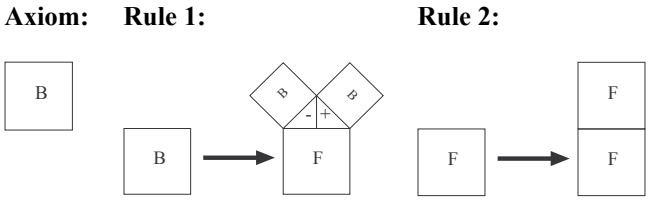


Figure 2.10: *Effect of individual rules.*

Note that the rules are applied in parallel, that is, all possible deductions (i.e. all possible applications of the rules) are performed simultaneously. Now let's look at the strings produced by the production system described above. The axiom tells us to start with a single *B* cell. Therefore the initial string is simply

$$B$$

Now we apply the rules to this string and obtain (only Rule 1 matches)

$$F[-B] + B$$

Note that the rules of the L-system are used as substitution rules. In the next step both rules match and are applied resulting in the following string:

$$FF[-F[-B] + B] + F[-B] + B$$

As can be seen, the length of the string grows dramatically and gets increasingly confusing:

$$FFFF[-FF[-F[-B] + B] + F[-B] + B] + FF[-F[-B] + B] + F[-B] + B$$

Let us perform one more step:

$$FFFFFFFF[-FFFF[-FF[-F[-B] + B] + F[-B] + B] + FF[-F[-B] + B] + F[-B] + B] + FFFF[-FF[-F[-B] + B] + F[-B] + B] + FF[-F[-B] + B] + F[-B] + B$$

Much more intuitive is the graphical representation of the same process as depicted in Figure 2.11.

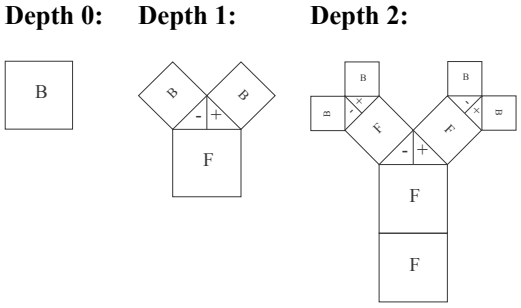


Figure 2.11: *Effect of joint action of rule system. What emerges is a kind of tree structure.*

Turtle Graphics

L-systems by themselves do not generate any images; they merely produce large sequences of symbols. In order to obtain a picture, these strings have to be interpreted in some way. In Figures 2.10 and 2.11 we have already seen a possibility. More generally, these L-systems can be interpreted by turtle graphics.

The concept of *turtle graphics* originated with Seymour Papert. Intended originally as a simple computer language for children to draw pictures (LOGO), a modified turtle graphics language is well suited for drawing L-systems. Plotting is performed by a (virtual) turtle. The turtle sits at some position looking in some direction of the computer screen and can move forward, either with or without drawing a line, and can turn left or right. A brief summary of commands used for drawing L-systems is given in table 2.3. Note that without the brackets the drawing of branching structures is impossible. Figure 2.12 shows the first five stages of the drawing process of the following L-system:

Axiom: *F*

Rule: *F=|[-F][+F]*

Angle: *20*.

Table 2.3: *Turtle graphics commands*

command	turtle action
F	draw forward (for a fixed length)
|	draw forward (for a length computed from the execution depth)
G	go forward (for a fixed length)
+	turn right (by a fixed angle)
-	turn left (by a fixed angle)
[	save the turtle's current position and angle for later use
]	restore the turtle's position and orientation saved by the most recently applied *[* command

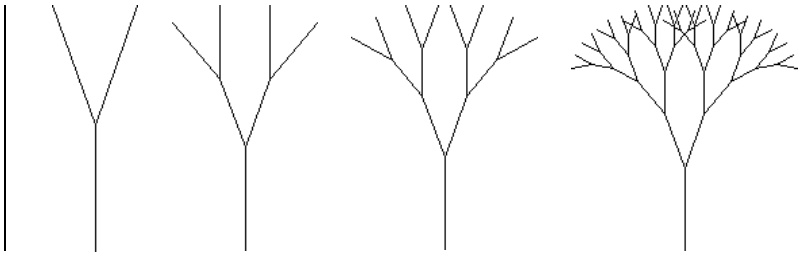


Figure 2.12: *The first five L-system stages*

The first drawing has an execution depth 0 and the drawing corresponds to the string

F

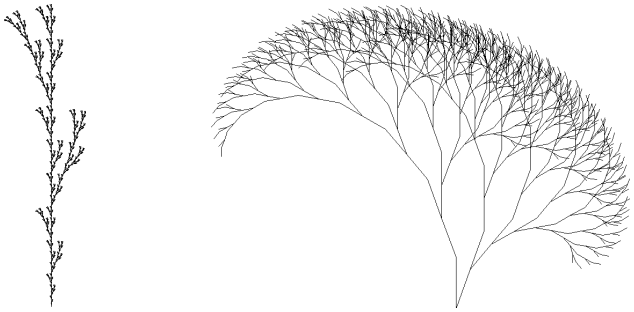
The execution depth denotes the number of times the rule is applied. The above string corresponding to execution depth 0 is therefore the axiom. In the second drawing, the rule is applied once, thus leads to the string

| [-F] [+F]

and the third drawing has execution depth 2 resulting in the string

| [-| [-F] [+F]] [+| [-F] [+F]]

In Figure 2.13 two examples of L-systems are given. A sample program to generate these images can be downloaded from mitpress.mit.edu/books/FLAOH/cbnhtml/index.html.



Axiom: *F*
Rule: *F = F [-F] F [+F] F*
Angle: 20
Depth: 7

Axiom: *F*
Rule: *F = | [+F] | [-F] +F*
Angle: 20
Depth: 9

Figure 2.13: *A few examples of L-systems (from Flake, 1998).*

Developmental Models

The interpretation of an L-system can be extended to three dimensions by adding a third dimension to the orientation of the turtle. In order to simulate the development of plants additional information has to be included into the production rules. Also, an additional assumption is made that plants control the important aspects of their own growth. Such information can include a delay mechanism or the influence of environmental factors but also a stochastic element, so that not all the plants look the same. Some examples of such more complex models of development are depicted in Figures 2.14 and 2.15 (from Prusinkiewicz, 1990). Additional information on these can be found on the really beautiful website www.cpsc.ucalgary.ca/projects/bmv/vmm-deluxe/TitlePage.html. We will discuss additional models of how shapes can grow when we discuss artificial evolution in Chapter 6.

Figure 2.14: *A sophisticated plan (a mint) grown with L-systems.*

Figure 2.15: *Simulated development of Capsella bursapastoris.*

2.4 Fractals

The simple L-systems we met in the previous section are instances of the more general structures known as *fractals*. The term "fractal"[9] was first used by Benoit Mandelbrot (e.g., Mandelbrot, 1983). Fractals are geometric figures that have one striking quality: They are self-similar on multiple scales, which means that a small portion of a fractal often looks similar to the whole object (in theory a fractal is perfectly regular and has infinitely fine structures). A description of a fractal-like object could be something like this: "It has a miniature version of itself embedded inside it, but the smaller version is slightly rotated." For example, one branch of a particular L-system plant looks exactly like the whole plant, only smaller (e.g. Figure 2.18). To be precise, this perfect self-similarity of L-systems holds only if the L-system is calculated to infinite depth, or to infinitely fine details. In this case the somewhat paradoxical statement holds that an arbitrary branch of the L-system plant is exactly the same as the whole plant, only rotated and scaled. In other words, a fractal contains itself. In addition, a fractal consists of infinitely many copies of itself.

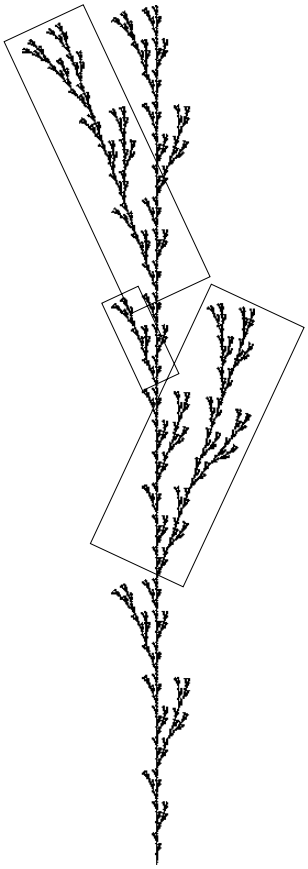


Figure 2.18: *The fractal structure of L-system turtle graphics. Each branch in the boxes contains a rotated and re-scaled copy of the whole figure.*

[9] The name fractal has been given based on the fractal dimension of these structures. A fractal can have a non-integer dimension meaning for example that it is "more than a line but less than a plane".

Fractals in Nature

Fractals often appear in nature. Not only plants like trees or ferns (see Figure 2.19) have a fractal structure, but also snowflakes, and blood vessels (see Figure 2.20).

Figure 2.19: *A fractal fern (from www.mhri.edu.au/~pdb/fractals/fern/)*

Fractals are nature's answer to hard "optimization" problems, i.e. how to find the optimal solution if there are conflicting goals. In case of the blood vessel system the hard task is to supply every part of the body with blood using as few resources as possible and at the same time minimizing the amount of time used for a single round trip. (Without this last condition one thin long blood vessel visiting every part of the body would do the job.) Because blood vessel systems are the result of millions of years of evolution one is allowed to expect that they are not just any solution to the problem but a good one, one that is close to an optimal one.

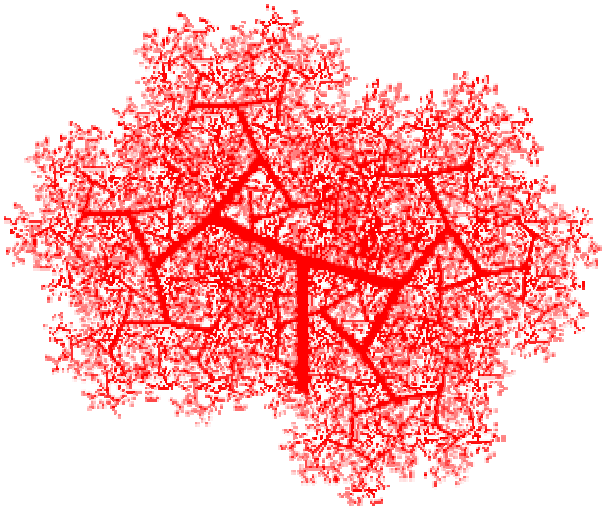


Figure 2.20: *A fractal model of the blood vessel system (from www.cs.ioc.ee/ioc/res98/fractal.html).*

Of course, fractals in nature are not perfect mathematical fractals; they have no infinitely fine structures and are not perfectly regular. Blood vessels, for example, do not become indefinitely small; there is some minimal diameter. Interestingly, the smallest blood vessels, the capillaries, are always of about the same size. For example the capillaries of an elephant have the same diameter as those of a mouse. The difference is that the elephant's blood vessel system has a few more branching levels than that of the mouse.

Generating fractals

Above we introduced the concept of self-similarity, i.e. the fact that fractals contain miniature versions of themselves. The trick in generating fractals is to come up with a way to describe where and how the miniature version of the whole should be placed. In the previous section we have used L-systems and turtle graphics. In general, there are four types of transformations that one could imagine as being useful: translation (move to different place), scaling (alter size), reflection, and rotation. Algorithms for generating fractals are always recursive and based on self-similarity, using combinations of these basic four transformations. An example is the Multiple Reduction Copy Machine Algorithm (MRCM). Figure 2.21 shows a schematic representation of the MRCM algorithm[10].

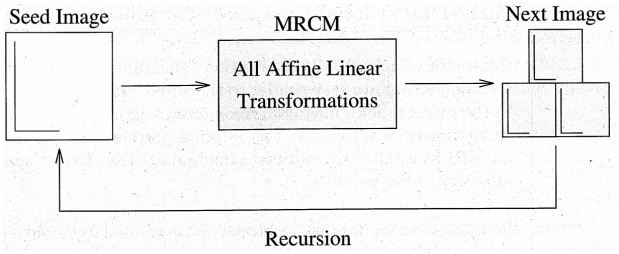


Figure 2.21: *A schematic of the MRCM algorithm. The input image is simultaneously transformed by translation and scaling.*

There is a vast literature on fractals. It would be beyond the scope of this chapter to provide extensive coverage. The interested reader is referred to Flake (1998; Chapter 7: Affine Transformation Fractals), Barnsley (1988), Mandelbrot (1983), and Peitgen et al. (1992).

2.5 Sea shells

Another fascinating case of pattern formation that can be conveniently described by cellular automata is the evolution of the colorful patterns of seashells. We all know the pigment patterns of tropical seashells and are impressed by their beauty and diversity. The fascination comes from their mixture of regularity and irregularity (see Figure 2.23). No two shells are identical but we can immediately recognize similarities. The patterns on the shell resemble the patterns we met in the sections on 1-dimensional CA. And this coincidence has a deeper reason.

[10] The MRCM algorithm's name is based on the fact that it could at least partly be simulated with a real copy machine (make simultaneously several copies of an object and alter place and size, such process to be repeated several times).

The patterns on seashells consist of calcified material. A mollusk can enlarge its shell only at the shell margin. Therefore, in most cases the calcified material, and thus the pigmentation patterns, is added at this margin. In this way the shell preserves a time record of the pigmentation process that took place at its margin. This process is much like the 2-dimensional pictures of 1-dimensional CA that are a time record of the CA dynamics. In this sense, it is straightforward to simulate pattern growth on a seashell with a 1-dimensional CA. Let us look at this idea in some detail.

As Meinhardt argues in his book "The Algorithmic Beauty of Sea Shells" (Meinhardt, 1995) the process of pattern formation in seashells can be conceived in terms of an activator-inhibitor dynamics (Figure 2.22) whereby the activator causes and the inhibitor suppresses pigmentation. These dynamics are often called reaction-diffusion dynamics. Pattern formation is the result of local self-enhancement (also called autocatalysis) and long-range inhibition.

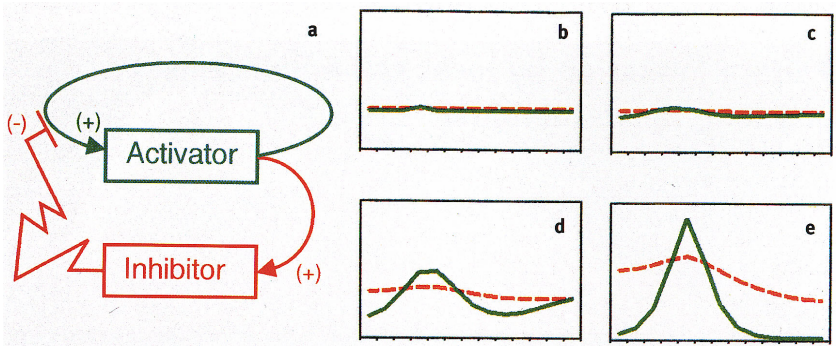


Figure 2.22: *Reaction scheme for pattern formation by autocatalysis and long-range inhibition. An activator catalyzes its own production and that of its antagonist (the inhibitor). The diffusion constant of the inhibitor must be much higher than that of the activator. A homogenous distribution of both substances is unstable (b) (the x-axis represents position and the y-axis the concentration). A minute local increase of the activator (——) grows further (c, d) until a steady state is reached in which self-activation and inhibition (- - - -) are balanced (from Meinhardt, 1995).*

Activator-inhibitor dynamics can be described either by a set of partial differential equations, or by cellular automata.

Meinhardt (1995) introduces the following differential equations to describe the dynamics of the activator-inhibitor system that relate the concentration change per time unit of both substances a and b as a function of the present concentration.

$$\frac{\partial a}{\partial t} = s\left(\frac{a^2}{b} + b_a\right) - r_a a + D_a \frac{\partial^2 a}{\partial x^2}$$

$$\frac{\partial b}{\partial t} = sa^2 - r_b + D_b \frac{\partial^2 b}{\partial x^2} + b_b$$

where $a(x,t)$ is the concentration of an auto-catalytic activator at position x at time t, $b(x,t)$ is the concentration of its antagonist, D_a and D_b are the diffusion coefficients and r_a and r_b are the decay rates of a and b.

Let us briefly outline the main intuitions why the interaction as stated in the above equations can lead to stable patterns. Let's assume that all constants, as well as the inhibitor concentration, are equal to 1, and let's disregard diffusion. This leads to the following simplified equations:

$$\frac{\partial a}{\partial t} = a^2 - a$$

Here the activator has a steady state but only at $a = 1$; otherwise the state is unstable. Simplifying the equation for the inhibitor b leads to

$$\frac{\partial b}{\partial t} = a^2 - b$$

Now the steady state is at $b = a^2$.

Let us include now the action of the inhibitor in the equation for the activator. Under the assumption that the inhibitor reaches the equilibrium rapidly after a change in activator concentration, this can be expressed as function of the activator concentration alone

$$\frac{\partial a}{\partial t} = \frac{a^2}{b} - a \approx \frac{a^2}{a^2} - a = 1 - a$$

The inclusion of the inhibitor leads to a steady state at $a = 1$ which remains stable because for $a > 1$ $(1\text{-}a)$ is negative and the concentration returns to $a = 1$ (Meinhardt, 1995).

As seen above the action of the inhibitor leads to stabilization of the autocatalysis and to stable patterns. On shells, stable patterns lead to permanent pigment production in some positions caused by a higher concentration of activator a and its suppression in between (higher concentration of inhibitor b). This leads, for example, to an elementary pattern of stripes parallel to the direction of growth.

The above partial differential equations, which represent the continuous change over time, can be approximated by a system of difference equations representing change in discrete time steps. Accordingly the discrete i will take the role of x (the position). The differentials $\frac{\partial a}{\partial t}, \frac{\partial a}{\partial x}, \frac{\partial^2 a}{\partial x^2}$ can be approximated by differences:

$$\frac{\partial a}{\partial t}(x,t) \approx a_i(t+1) - a_i(t)$$

$$\frac{\partial a}{\partial x}(x,t) \approx a_{i+1}(t) - a_i(t)$$

$$\frac{\partial^2 a}{\partial x^2}(x,t) \approx \left[a_{i+1}(t) - a_i(t)\right] - \left[a_i(t) - a_{i-1}(t)\right].$$

Inserted into the system of differential equations given above the concentration of the activator is

$$a_i(t+1) = a_i(t) + s\left(\frac{a_i^2(t)}{b_i(t)} + b_a\right) - r_a a_i(t) + D_a(a_{i-1}(t) + a_{i+1}(t) - 2a_i(t)) .$$

Time is now discrete with a time step of Δt=1. If we interpret *i* as the number of a cell (in a row) the above equation is in fact a local rule for a 1-dimensional CA. Analogously, the equation for the inhibitor *b(x,t)* can be reformulated and we obtain a local rule for $b_i(t)$:

$$b_i(t+1) = b_i(t) + sa_i^2(t) - r_b b_i(t) + D_b(b_{i-1}(t) + b_{i+1}(t) - 2a_i(t)) + b_b .$$

Therefore our resulting CA has *two* variables $a_i(t)$ and $b_i(t)$ for each cell *i*. The difference to the CA discussed earlier is that the state variables here can take arbitrary values and not just discrete ones.

In Figure 2.23 two examples of seashells and their simulated counterparts are shown (from Meinhardt, 1995). The patterns were calculated as discussed above and the mapped onto a 3-dimensional model of a seashell. The results are striking.

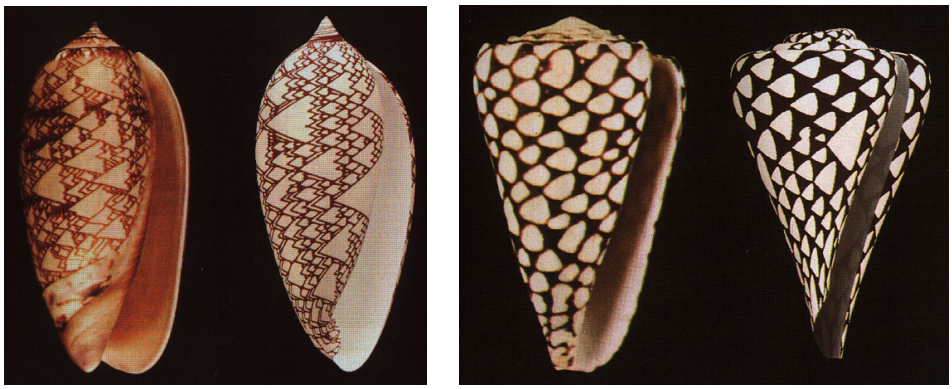


Figure 2.23: *Two examples of seashells and the simulated patterns using — in essence — the dynamics described in this section (taken from Meinhardt, 1995, p. 179, 180).*

2.6 Sandpiles

While studying the fundamental question why nature is so complex and not as simple as the laws of physics would imply, Bak, Tang and Wiesenfeld (1988) identified the concept of self-organized criticality (SOC) – a mathematical theory of how dissipative dynamical systems naturally evolve into a self-organized critical state[11] with no characteristic time or length scales (see also Bak, 1997). In other words, according to SOC Nature is perpetually out of balance but organized in a poised state where anything can happen within well-defined statistical laws. SOC attempts to explain some complex phenomena that we find in nature: distribution of solar flares and intensity of sunspots, distribution of earthquake sizes (Gutenberg-Richter Law), the size distribution of initial masses of stars (Salperter Law), and the distribution of sizes of extinction events (see also theory of punctuated equilibria or “evolution by jerks”).

A good and easily understandable example of SOC is the sandpile model. One can imagine a flat table onto which grains of sand are added randomly one at a time. In the beginning the grains will mostly stay where they land. With more sand added grains start to pile up and sand slides and avalanches occur. First such

avalanches only have a local effect in one particular region of the table but with more sand added the piles cannot get any higher since the slope is too steep for additional grains of sand. Consequently the avalanches become stronger and also affect the piles in the other regions of the table or may even cause sand to leave the table (see figure below).

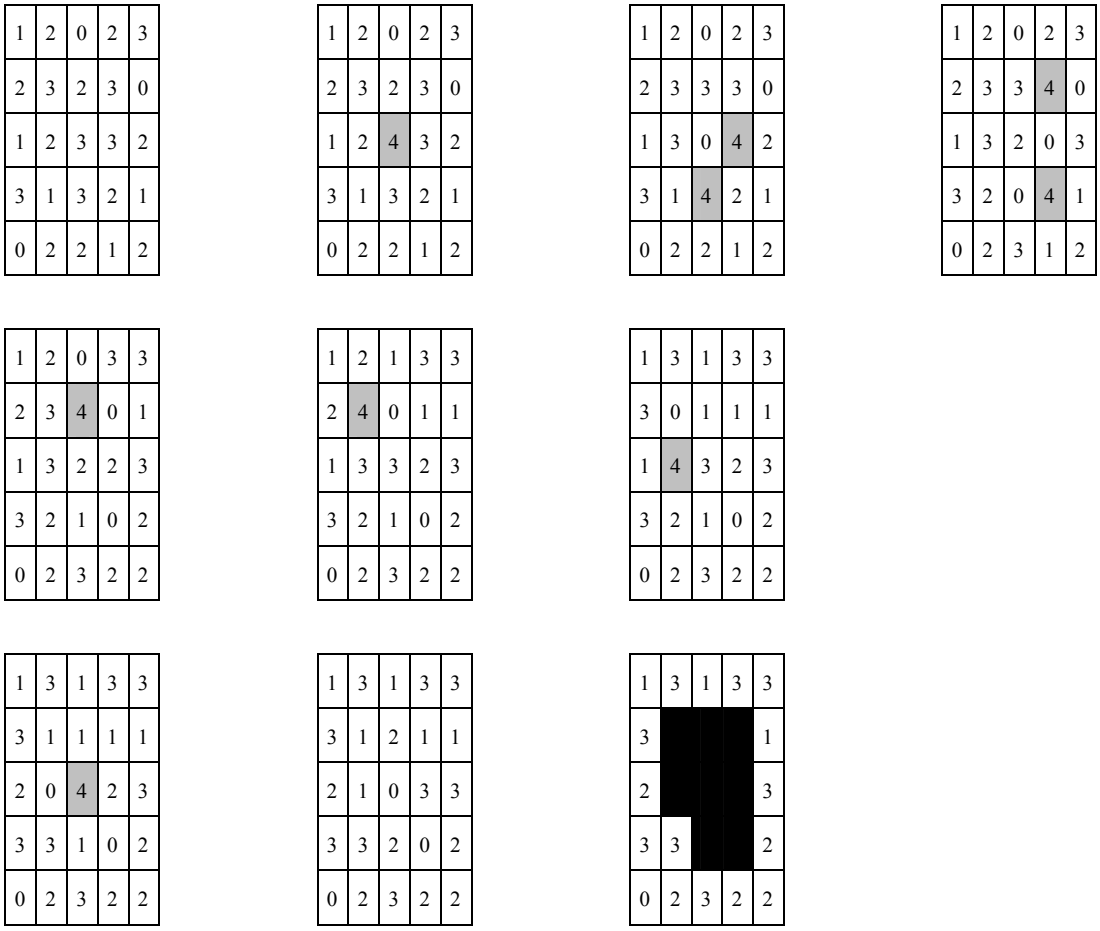


Figure2.24: *Illustrating of toppling avalanche in a small sandpile. A grain falling at the site with height 3 at the center of the grid leads to an avalanche composed of nine toppling events, with a duration of seven update steps. The avalanche has a size s=9. The black squares indicate the eight sites that toppled. One site toppled twice. (Bak, 1997, p.53)*

In the end new grains added to the pile will result in average in the same number of grains rolling down the pile and falling off the table. In order to achieve such balance between sand added to and sand leaving the table, communication within the system is required. Such state is the self-organized critical (SOC) state.

The number of avalanches of size *s* (distribution of frequency) can be expressed by a power law

$$N(s) = s^{-\tau}$$

where the exponent τ defines the slope of the curve which plotted on a double-logarithmic paper results in a quasi straight line. Stated in simple terms the power law states that "small avalanches appear more often than big ones."

[11] Critical in the sense that it is neither stable nor unstable, but near phase transition.

The addition of grains of sand has transformed the system from a state in which the individual grains follow their own local dynamics to a critical state where the emergent dynamics is *global* (Bak, 1997). The individual elements obeying their own simple rules have – through interaction – lead to a unique, delicately balanced, poised, global situation in which the motion of any element might affect any other element in the system.

Accordingly the sandpile model shows how an open system has naturally organized itself into a critical scale-free state without any external organizing force, thus a simple model for complexity in nature has been developed.

2.7 Conclusion

In this chapter we have looked at a number of examples illustrating basic principles of pattern formation in natural and artificial systems. The essence is that sophisticated patterns can emerge on the basis of *simple rules* that are based on local interactions. There is no need for a global blueprint. Cellular automata, Lindenmeyer systems (L-systems), fractals and SOC are convenient formalisms to model pattern formation processes.

Another central factor in pattern formation is — almost trivially — the *availability of many cells*, and that all the cells are processed in parallel: there must be *no central control*.

2.8 Bibliography

Adami, C. (1998). *Artificial Life: An Introduction*. Springer-Verlag: New York, Berlin, Heidelberg.

Bak, P., Tang., C. and Wiesenfeld, K. (1988). Self-organized criticality. *Phys. Rev. A*, 38(1):364-374.

Bak, P. (1997). *How Nature Works*. Oxford University Press.

Ball, P. (1999). *The Self-Made Tapestry*. Oxford University Press.

Barnsley, M. (1988). *Fractals Everywhere.* Academic Press

Beckmann, F. S. (1980). *Mathematical Foundations of Programming.* Addison-Wesley.

Berlekamp, E. R., J. H. Conway, R. K. Guy (1982). *What is Life?* Chapter 25 in Winning Ways (Volume 2), Academic Press.

Flake, G. W. (1998). *The Computational Beauty of Nature.* MIT Press: Cambridge, MA. A Bradford Book

Gardner, M. (1970). Mathematical Games. The fantastic combinations of John Conway's new solitaire game "life". *Scientific American*, 223(4):120-123

Gerola, H. and P. Seiden (1978). Stochastic star formation and spiral structure of galaxies. *Astrophys. J.*, 223:129.

Greenberg, J. M., Hassard, B.D. and S. P. Hastings (1978). Pattern formation and periodic structures in systems modeled by reaction-diffusion equations. *Bull. Am. Math. Soc.*, 84:1296

Mandelbrot, B. (1983). *The Fractal Geometry of Nature.* Freeman

Meinhardt, H. (1995). *The Algorithmic Beauty of Sea Shells*. Springer Verlag.

Peitgen, H.-O., Jürgens, H. and Saupe, D. (1992). *Chaos and Fractals*, Springer Verlag.

Prusinkiewicz, P. (1990). *The Algorithmic Beauty of Plants*. Springer Verlag.

Schewe, P. F (ed.) (1981). Galaxies, the Game of Life, and Percolation. *Physics News, Amer. Inst. Phys. Pub.* R-302:61.

Wolfram, S. (1984a). Computation Theory of Cellular Automata. *Commun. Math. Phys.*, 96:15-57.

Wolfram, S. (1984b). Universality and Complexity in Cellular Automata. *Physica D*, 10:1-35.

Wolfram, S. (2002). *A New Kind of Science*. Wolfram Media.