11. Automata and languages, cellular automata, grammars, L-systems

11.1 Automata and languages

Automaton (pl. *automata*): in computer science, a simple *model* of a machine or of other systems. ($\rightarrow$ a simplification of real machines or systems)

Examples:
Some systems to be modelled as automata with finite description

Example 1: Traffic lights, with a timer for stepping from state to state; outputs switch on or off the different lamps.

Inputs: Timer (t)

Outputs: switch to red (red), switch to red and yellow (red-and-yellow), switch to green (green), switch to yellow (yellow).

States: red, red-and-yellow, green, yellow

Example 2: A cookie vending machine.

Inputs: a coin (c); buttons for choosing some sort of cookies (b_1), (b_2) or for wanting the money back (back).

Outputs: a coin back (c); and for giving out one of several kinds of cookies (c_1), (c_2); an error signal (bell).

States: waiting (w); coin-accepted (c)

Example 3: A sugar-digesting bacterium

Inputs: glucose (*g*), lactose (*l*), nothing (*n*)

Outputs: lactase gene activated (*a*), lactase gene inactive (*i*) – can be measured in the laboratory by methods from molecular genetics

States: lactose-digesting (*1*), glucose-digesting (*2*), dormant (*3*)

Abstract notion: the *Mealy automaton*

What do the examples have in common?

Approach: A **finite set** of inputs, a **finite set** of outputs, and a **finite set** of states. At any moment, the system is in one state.

Formally: **Mealy automaton**

A **Mealy automaton** is a tuple

$$(Q, \Sigma, \Delta, \delta, \lambda, q_0)$$

where Q is a finite set (the set of **states** of the automaton), Σ is the **input alphabet**, Δ is the **output alphabet**, $\delta : Q \times \Sigma \to Q$ is a *transition function*, $\lambda : Q \times \Sigma \to \Delta$ is an **output function**, and $q_0 \in Q$ is the **initial state**.

(remember: $\delta : Q \times \Sigma \to Q$ means that the function δ associates to each *pair* (q, s) with q from Q, s from Σ an element $\delta(q, s) \in Q$. For λ analogously.)

Interpretation: Q models the **different states** the system can be in, and q_0 is the state it starts in. Σ models the **different inputs** on which the system can react, and Δ models the **different ways in which the system can react** on an input. δ models the **change of the state** which can happen when an input event takes place: if the system is in state q and input σ is entered in the system, the system enters state $\delta(q, \sigma)$. λ models the reaction of the system: If the system is in state q when an input σ is entered, the system outputs $\lambda(q, \sigma)$.

The set of outputs Δ might include a do-nothing action, which can be used for combinations of states and inputs in which no output is generated.

A Mealy automaton can be understood as a **translator**: each word of letters from Σ is translated to a word of letters from Δ of the same length.

Example for a Mealy automaton: The cookie vending machine from above.

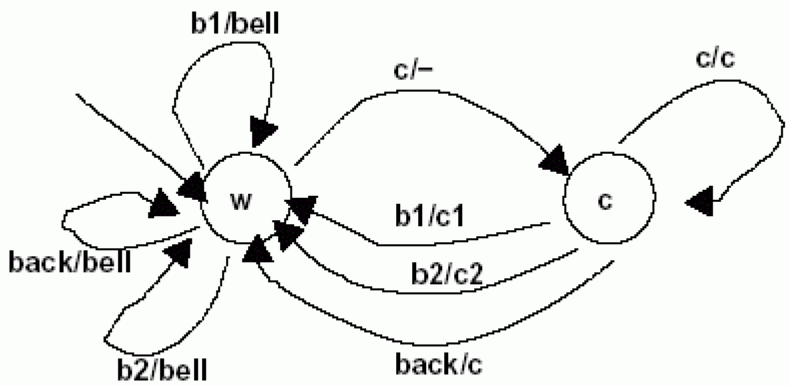
Model: $(\{w, c\}, \{c, b_1, b_2, \text{back}\}, \{c, c_1, c_2, \text{bell}, \text{nothing}\}, \delta, \lambda, w)$

with δ and λ defined by the following table:

q	σ	$\delta(q, \sigma)$	$\lambda(q, \sigma)$
w	c	c	nothing
w	b_1	w	bell
w	b_2	w	bell
w	back	w	bell
c	c	c	c
c	b_1	w	c_1
c	b_2	w	c_2
c	back	w	c

There must be a follower state and an output **for each state/input combination**.

Graphical representation: Initial state marked an incoming arrow not starting at a state. Each Q represented as a circle. δ and λ represented as arrows between circles, which are labeled with a pair of form Σ/Δ. Each line in the tabular representation corresponds to an arrow:



Formal languages:

Formal language: (short: language) Set of **words** (=character sequences) over an **alphabet** Σ (=a finite set, interpreted as set of characters)

Examples:

$\Sigma_1 = \{a, b, ..., z\}$, with language $L_1 = \{\text{one}, \text{two}, \text{three}\}$

$\Sigma_2 = \{0, 1\}$ with language $L_2 = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, ...\}$ (L_2 contains all sequences of 0's and 1's of finite length, even the empty sequence, denoted ϵ, of length zero.

$\Sigma_3 = \{0, 1\}$ with language

$$L_3 = \{w \in \Sigma_3^* | \text{The number of 1's in } w \text{ is positive and even}\}$$

Definitions concerning words and formal languages:

The *length* $|w|$ of a word w is the number of its letters.
The length of the empty word ε is 0.

The language of *all* finite words over an alphabet Σ, including the empty word, is denoted by Σ^*.
(In the example above: $L_2 = \Sigma_2^*$.)

The language of all *non-empty* words over Σ is denoted by Σ^+. $\Sigma^+ = \Sigma^* - \{\varepsilon\}$.

The language of all words of length n over Σ is denoted by Σ^n. $\Sigma^0 = \{\varepsilon\}$.

Operations on formal languages

Let L, L_1 and L_2 be languages over a common alphabet Σ. Then we can define the following other languages over Σ:

- L^* is the set of words w over Σ which can be split into a finite number of sub-words $w_1 w_2 \ldots w_n$ such that each $w_i \in L$. The empty sequence of words is allowed, i.e. $\epsilon \in L^*$.

- L^+ is defined similar to L^*, only the empty sequence of subwords is not allowed. ε is in L^+ only if it is already in L.

- $L_1 L_2$ is defined as the set of words w which can be split into two sub-words $w = w_1 w_2$ with $w_1 \in L_1 \wedge w_2 \in L_2$.

The connection between automata and formal languages:

How can we define a language using an automaton?

First idea: collect all input sequences for which the corresponding output of the automaton ends with a special symbol "accept".

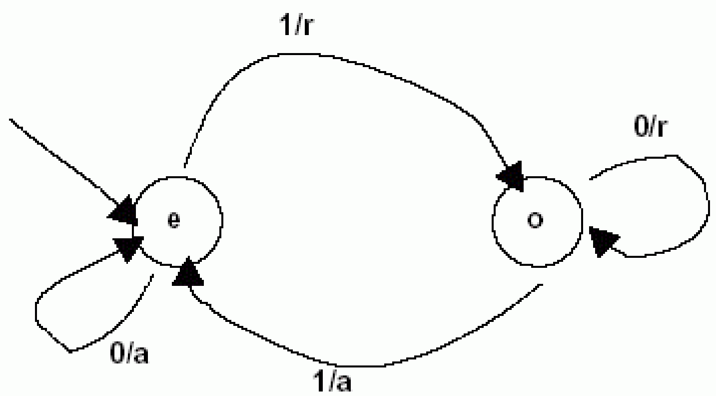
$\rightarrow$ Formal language definition by Mealy automata:

Consider a Mealy automaton with $\Delta = \{a, r\}$ (for 'accepted' and 'rejected').

The language $L(M)$ over the input alphabet Σ defined by a Mealy automaton M consists of all words $w \in \Sigma^*$ such that when w is processed, the last output symbol is a.

Example:

A Mealy automaton which accepts L_3 (words from $\{0, 1\}$ with an even and positive number of ones):



Problem: Languages which contain ϵ can not be defined in this way.

We simplify the notion of automaton:
The output is no longer necessary, only the states are used.

$\rightarrow$ notion of "*finite automaton*" (FA); also called "deterministic finite automaton" (DFA):

A finite automaton is a quintuple

$$(Q, \Sigma, \delta, q_0, F)$$

with Q a finite set (the **states** of the DFA), Σ the *input alphabet*, $\delta : Q \times \Sigma \rightarrow Q$ the *transition function*, q_0 the **initial state** and $F \subseteq Q$ the set of **final states**.

Difference to Mealy automata: There is no output alphabet and no output function; but there is a set of final states.

A FA is an “acceptor”: Input words (i.e. elements of Σ^*) which lead the automaton from q_0 to an element of F are *accepted* by the automaton, other input words are **rejected**.

Extension of δ to words: $\hat{\delta}$

$$\hat{\delta} : Q \times \Sigma^* \rightarrow Q$$

with

- $\hat{\delta}(q, \epsilon) = q$
- for each word $aw \in \Sigma^+$ starting with some letter $a \in \Sigma$ and continuing with word w:

$$\hat{\delta}(q, aw) = \hat{\delta}(\delta(q, a), w)$$

Language accepted of a FA M:

$$L(M) = \{w \in \Sigma^* | \hat{\delta}(q_0, w) \in F\}$$

The set of languages which are accepted by some FA are denoted as $\mathcal{L}_{fa}$.

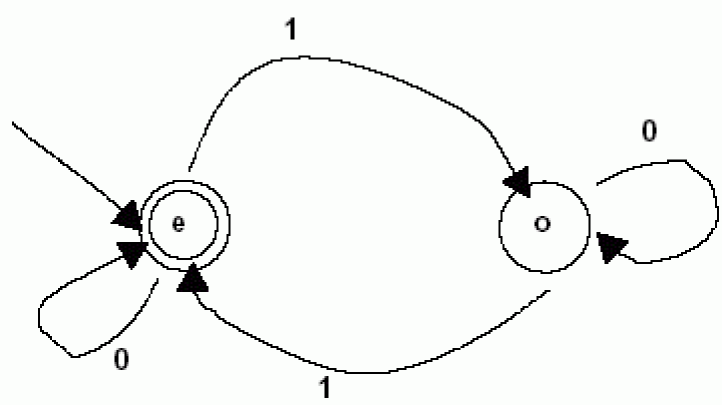
Example of a FA which recognizes all words over $\{0, 1\}$ with an even number of 1's:

$$(\{e, o\}, \{0, 1\}, \delta, e, \{e\})$$

with δ defined by

q	σ	$\delta(q, \sigma)$
e	0	e
e	1	o
o	0	o
o	1	e

Graphical representation: No outputs; final states as double circles:



FA can be used *to check the correctness* of statements in simple programming languages ("accept" = input was syntactically correct; "reject" = an error was detected). However, more complicated languages need other, more refined forms of automata.

The languages for which a finite automaton exists form a special class of languages:

Regular languages

A language L is called **regular** if and only of there is a FA M with $L = L(M)$

Regular expressions over an alphabet Σ

= a way to define "allowed" character sequences (or sequences of commands...), also used e.g. in search queries in databases or information systems
– *stand in connection with regular languages*

We give a recursive definition:

- If $a \in \Sigma$, then a is a r.e. (one-word/one-letter language)
- $\{\}$ is a r.e. (empty language)
- If r is a r.e., then $(r*)$ is a r.e. (repetition)
- If r and s are r.e., then (rs) is a r.e. (concatenation)
- If r and s are r.e., then $(r|s)$ is a r.e. (alternative)
- No other expressions are r.e.

If parentheses are dropped: $*$ binds tighter than concatenation, which binds tighter than $|$.

The language $L(t)$ of a regular expression t

$L(t)$ is defined in the following way:

- If $t = a$ for $a \in \Sigma$, then $L(t) = \{a\}$.
- If $t = \{\}$, then $L(t) = \{\}$.
- If $t = (r*)$ for some r.e. r, then $L(t) = (L(r))^*$
- If $t = (rs)$ for some r.e.s r and s, then $L(t) = L(r)L(s)$.
- If $t = (r|s)$, for some r.e.s r and s, then $L(t) = L(r) \cup L(s)$.

The languages which are definable by regular expressions are denoted by $\mathcal{L}_{re}$.

Examples for regular expressions

All sequences of 0's and 1's: $(0|1)*$

All nonempty sequences of 0's and 1's: $(0|1)*(0|1)$

$\{\epsilon\}$: $\{\}*$

Sequences of 0's and 1's with at least one pair of contiguous 1's: $(0|1)*11(0|1)*$

The language with the two words 000 and 111: $000|111$

Sequences of 0's and 1' with an even number of 1's: $(0*10*1)*0*$

Floating point numbers, using abbreviation Z for $(0|1|2|3|4|5|6|7|8|9)$:

$$(\{\}*|+|-)(ZZ*)(\{\}*|(.ZZ*))(\{\}*|E(\{\}*|+|-)ZZ*)$$

The connection to regular languages (i.e., to finite automata):

Theorem:
The languages which can be defined by regular expressions are exactly the regular languages.

$$\mathcal{L}_{fa} = \mathcal{L}_{re}$$

This justifies the word "regular" for languages definable by FA.

Proof: by construction of FAs for each of the alternatives for the construction of regular expressions, and vice versa.

Summary of this section:

Mealy automata: Some systems are adequately modeled by a *finite number of states*, reacting in each state on one of a *finite number of inputs*, by changing the state and emitting one of a **finite number of output symbols**.

Formal languages: sets of finite words over a fixed alphabet.

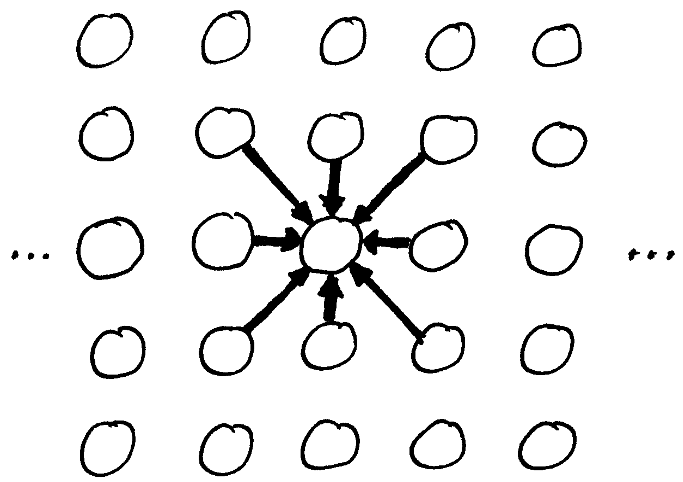
Finite automata: Some formal languages are definable by automata. Special kind of automata, specifically for language definition: FA.

Regular expressions: An often more convenient way to define the same languages.

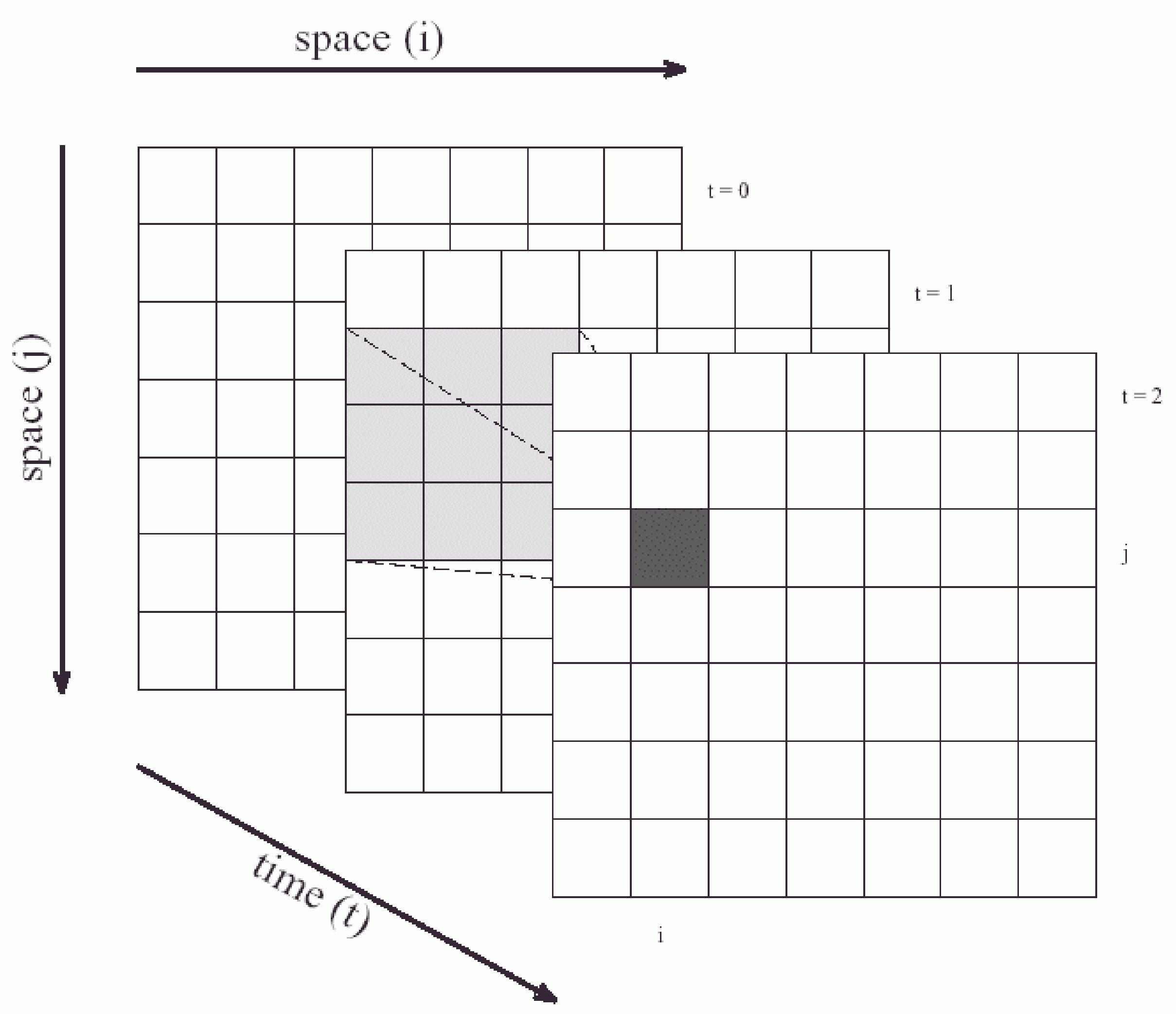
11.2 Cellular Automata

What will happen if we connect several finite automata with each other?

- A 2-dimensional *Cellular Automaton* (CA) is a rectangular (potentially infinite) array of cells. In each of the cells there is a finite automaton ($\rightarrow$ each cell can adopt a finite number of states).
- All the automata in the cells have the same functioning.
- In each time step, each cell takes as input the states of all its neighbour cells and uses them to calculate its own new state (using a "transition function").
- The output of the CA is in each time step the pattern of the states of all the cells.



Formally: (G, Q, δ, c)
$G_{(i,j)}$: grid of cells with indices i, j
Q : set of states
$\delta : Q^{n+1} \rightarrow Q$ transition function (works in each cell)
(n = size of the neighbourhood)
$c : G \rightarrow Q$ initial configuration



1- dimensional, 3-dimensional ... CA can be defined analogously. Sometimes also triangular or hexagonal grids are used.

An example of a 2-dimensional CA:
Conway's "Game of Life"

Only 2 states: 1 = "living", 0 = "dead".
Neighbourhood: 8 cells (as above).
Simple transition rule, counting only the number of living cells in the neighbourhood as input.

Living cell, surrounded by 2 or 3 living cells → living
Dead cell, surrounded by exactly 3 living cells → living
in all other cases → dead

Expressed in other words:

- The **world** is structured like an infinite chess board; on each cell, there either **is** an individual, or there is **no individual**.

- The **time** proceeds in cycles (generations). In each generation, some individuals **die**, others **stay alive**, and in some empty cells, new individuals **are born**.

- The **conditions** for death and birth are: only the **current** state of a cell and that of the **neighbors** are relevant for the state of the cell in the next cycle.

The neighborhood consists of the **eight surrounding cells**.

– An individual **stays alive** if it has **two or three** living neighbors; otherwise, the individual **dies**.

– In an empty cell, an individual is born if there were **three** living neighbors during a cycle.

Example developments (environment is assumed to be empty)

Flip forth and back:

```
 | |      |X|      | |
-+-+-    -+-+-    -+-+-
X|X|X ->  |X|  -> X|X|X -> ...
-+-+-    -+-+-    -+-+-
 | |      |X|      | |
```

Steady state:

```
 | | |        | | |        | | |
-+-+-+-      -+-+-+-      -+-+-+-
 |x|x|        |x|x|        |x|x|
-+-+-+-  ->  -+-+-+-  ->  -+-+-+-  ->  ...
 |x|x|        |x|x|        |x|x|
-+-+-+-      -+-+-+-      -+-+-+-
 | | |        | | |        | | |
```

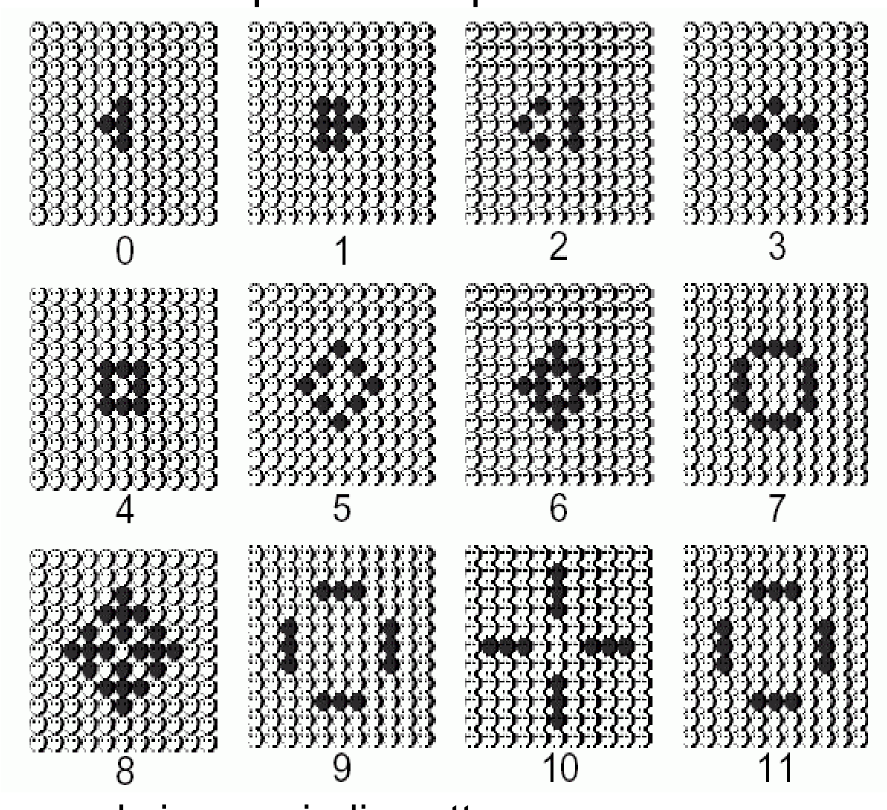
Move to south-east in four steps:

```
 | |x| | | |   |x| | | | |   | |x| | | |   | | | | | |   | | | | | |
-+-+-+-+-+-+- -+-+-+-+-+-+- -+-+-+-+-+-+- -+-+-+-+-+-+- -+-+-+-+-+-+-
x| |x| | | |   | |x|x| | |   | | |x| | |   |x| |x| | |   | | |x| | |
-+-+-+-+-+-+- -+-+-+-+-+-+- -+-+-+-+-+-+- -+-+-+-+-+-+- -+-+-+-+-+-+-
 |x|x| | | |   |x|x| | | |   |x|x|x| | |   | |x|x| | |   |x| |x| | |
-+-+-+-+-+-+- -+-+-+-+-+-+- -+-+-+-+-+-+- -+-+-+-+-+-+- -+-+-+-+-+-+-
 | | | | | |   | | | | | |   | | | | | |   | |x| | | |   | |x|x| | |
```

this configuration is called a "glider".

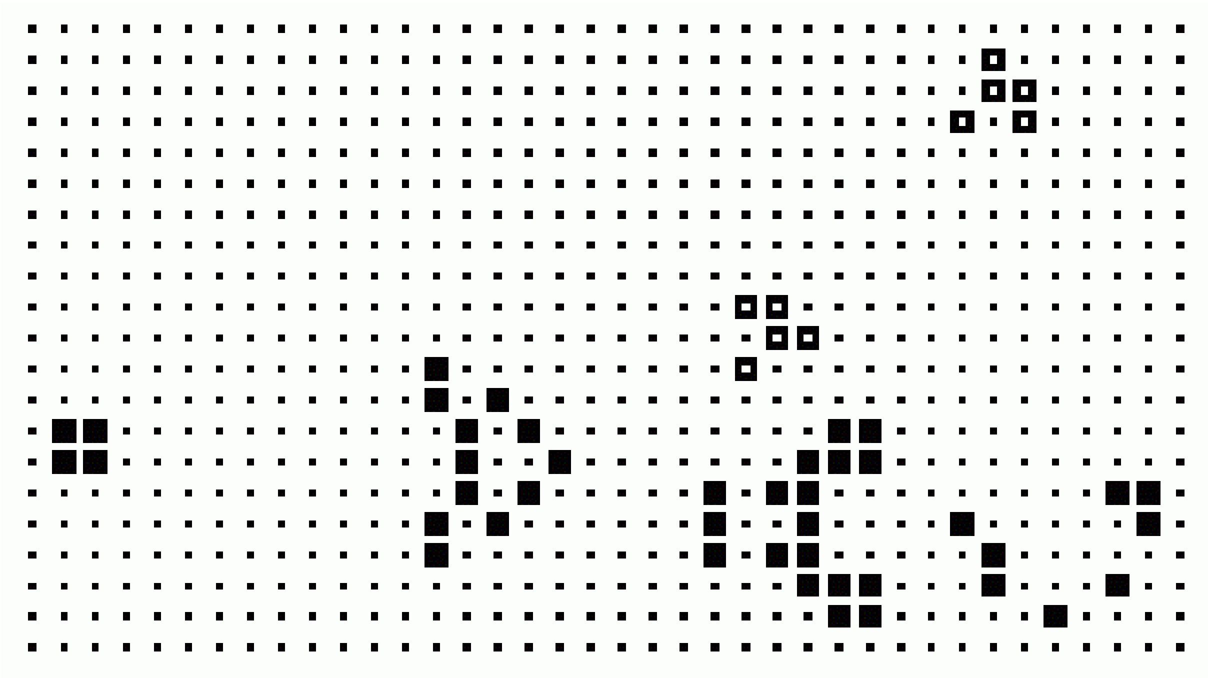
What else can happen in this simple "world"?

Other example development:



$\rightarrow$ ends in a periodic pattern.

"Glider gun" (dark) emitting "gliders" (brighter):



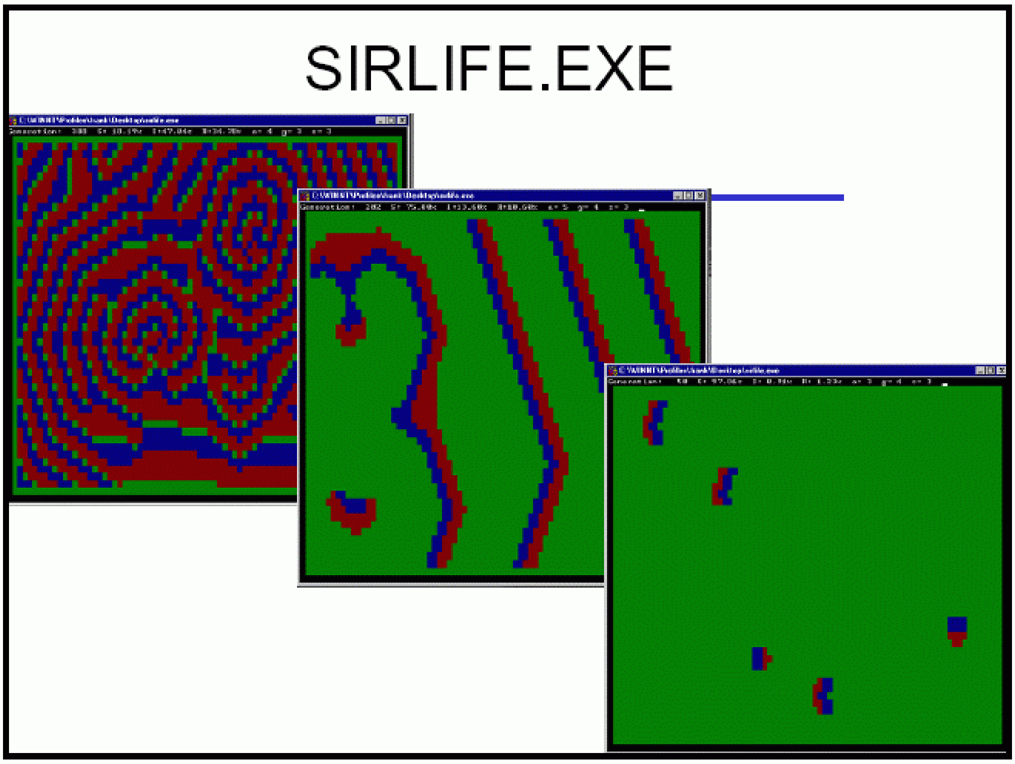
Computer scientists have shown that the Game of Life can even simulate a computer with logical circuits (AND, OR, NOT - switches) – hence all what can be calculated can be calculated using the Game of Life!

But: Game of Life rules do not reflect real-world properties
$\rightarrow$ no significance for real-world ecological modelling

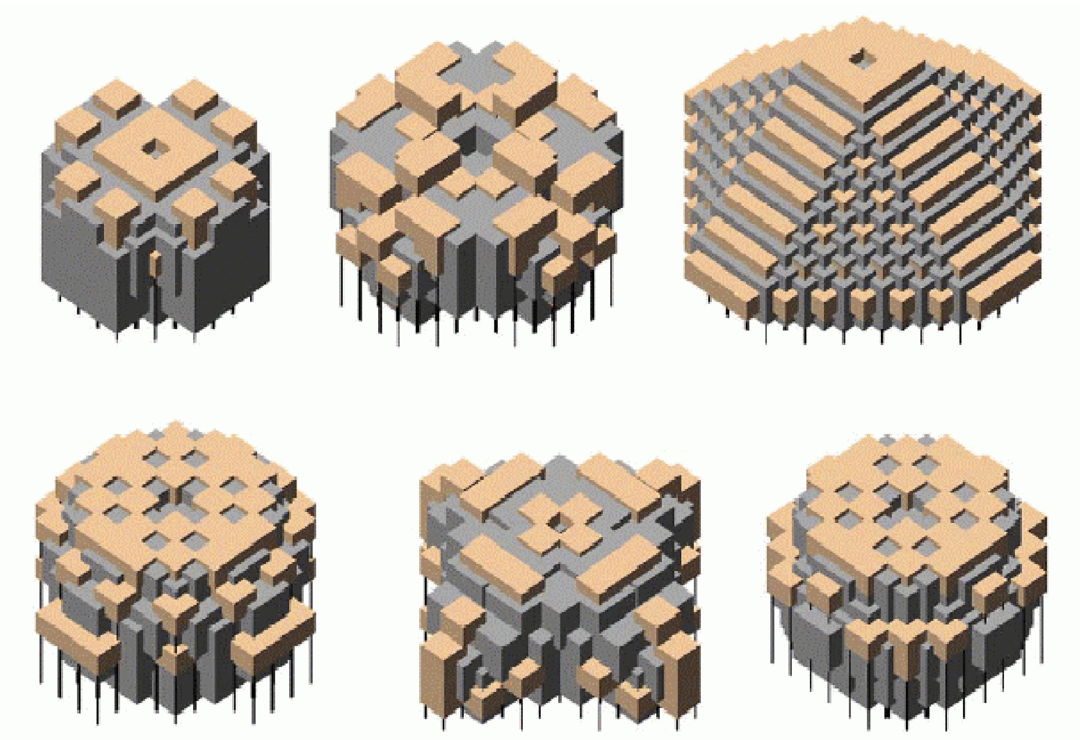
Using more states and other sets of transition rules, cellular automata can be designed *to simulate real-world spatial patterns and dynamics*.

Examples:
- spreading of forest fires
- spreading of rabies disease in fox populations
- colonisation of a habitat by a new species
- formation of colour patterns on seashells
- chemical reaction patterns
- architecture...

Example: CA simulating the spreading of an epidemic
(University of Leipzig)



Example: Architectural interpretation of a 3-dimensional CA (Robert J. Krawczyk, http://www.iit.edu/~krawczyk/rjkga03.pdf)



Disadvantages of CA in simulation applications:

- only discrete time steps
- some directions in space are preferred (due to the underlying grid)
- limited speed of interaction – no far-reaching, immediate effects possible

11.3 Grammars

Idea: Complex structures, e.g. the sentences of our natural language, can be described by simple rules.

For example, many sentences have the form:
Subject - predicate - object.

Definition:
A *Chomsky grammar* is a quadruplet

$$(\Sigma_N, \Sigma_T, X, R),$$

where Σ_N and Σ_T are disjunct sets of symbols, $X \in \Sigma_N$ and R is a finite set of rules of the form $A \rightarrow B$
where A and B are words over the alphabet $\Sigma_N \cup \Sigma_T$ and A contains at least one symbol from Σ_N.
(after Noam Chomsky, American linguist and philosopher)

Symbols from Σ_N are called *nonterminal symbols*, those from Σ_T *terminal symbols*.
X is the *start symbol*.

A *derivation* of a grammar is a sequence of words, beginning with X, where each word is obtained from its predecessor by replacing a subword according to one of the rules.
The *language defined by the grammar* consists of all words which can be derived from X and contain only terminal symbols.

Example:

$$\Sigma_N = \{ S, P, O \}$$

$$\Sigma_T = \{ \text{Mary}, \text{John}, \text{takes}, \text{reads}, \text{the book}, \text{the newspaper}, \text{the apple} \}$$

$$X = SPO$$

$$\begin{aligned} R = \{ & S \rightarrow \text{Mary}, \quad S \rightarrow \text{John}, \\ & P \rightarrow \text{takes}, \quad P \rightarrow \text{reads}, \\ & O \rightarrow \text{the book}, \quad O \rightarrow \text{the newspaper}, \quad O \rightarrow \text{the apple} \} \end{aligned}$$

A derivation:

$$\begin{array}{l} S\,P\,O \\ \downarrow \\ \text{John } P\,O \\ \downarrow \\ \text{John takes } O \\ \downarrow \\ \text{John takes the book} \quad \in \Sigma_T^* \\ \qquad\qquad\qquad\qquad\qquad \text{(terminals only)} \end{array}$$

Applications:

- description of the syntax of natural languages
- precise definition of the syntax of programming languages
- use of grammar-based derivations as an alternative approach to problem-solving, particularly in Artificial Intelligence applications (automated theorem proving, decision-making, speech recognition, games...): rule-based programming paradigm

11.4 L-Systems

L-systems = Lindenmayer systems
(after Aristid Lindenmayer, botanist)

special sort of grammars
designed for modelling the shape of organisms, particularly plants

Differences to Chomsky grammars:

- no distinction between terminal and nonterminal symbols
- only 1 symbol on the left-hand side of each rule
- all symbols for which a rule is applicable are replaced in parallel
- additional component: an interpretation which assigns a geometrical meaning to each generated word.

Formally:

$$(\Sigma, X, R, I),$$

where Σ is a set of symbols, $X \in \Sigma$ and R is a finite set of rules of the form $a \to \mathrm{B}$
where a is a symbol from Σ and B a word over the alphabet Σ
I is an interpretation mapping $I : \Sigma^* \to \mathbf{R}^3$ (from the set of words into 3-dimensional space).

normally used for the interpretation:
"turtle geometry"
"Turtle": device for drawing or constructing lines or cylindrical elements (virtual)

- stores (graphical and other) information
- has an internal "stack memory" (last in - first out)

- current state of the turtle contains information about current line thickness, step length, colour to be used, further properties of the object which will be constructed next

Turtle *commands* (selection):

`F` "Forward", including construction of an element (line segment, internode of a plant...) uses the current step length as the length of the new segment

`f` forward without construction ("move" command)

`L(x)` change the current step length to *x*

`L+(x)` increment the current step length by *x*

`L*(x)` multiply the current step length by *x*

`D(x), D+(x), D*(x)` analogous for thickness (diameter of the next segment)

`RU(45)` Rotation of the turtle around the "up" axis by 45°

`RL(...), RH(...)` analogously around the "left" and "head" axis

up-, *left*- and *head* axis form an orthonormal system with positive orientation which is carried by the turtle

`+, –` abbreviations for **RU**(φ) and **RU**(–φ) with fixed angle φ

Branching: Realized with "stack commands"

`[` put current state on the stack memory

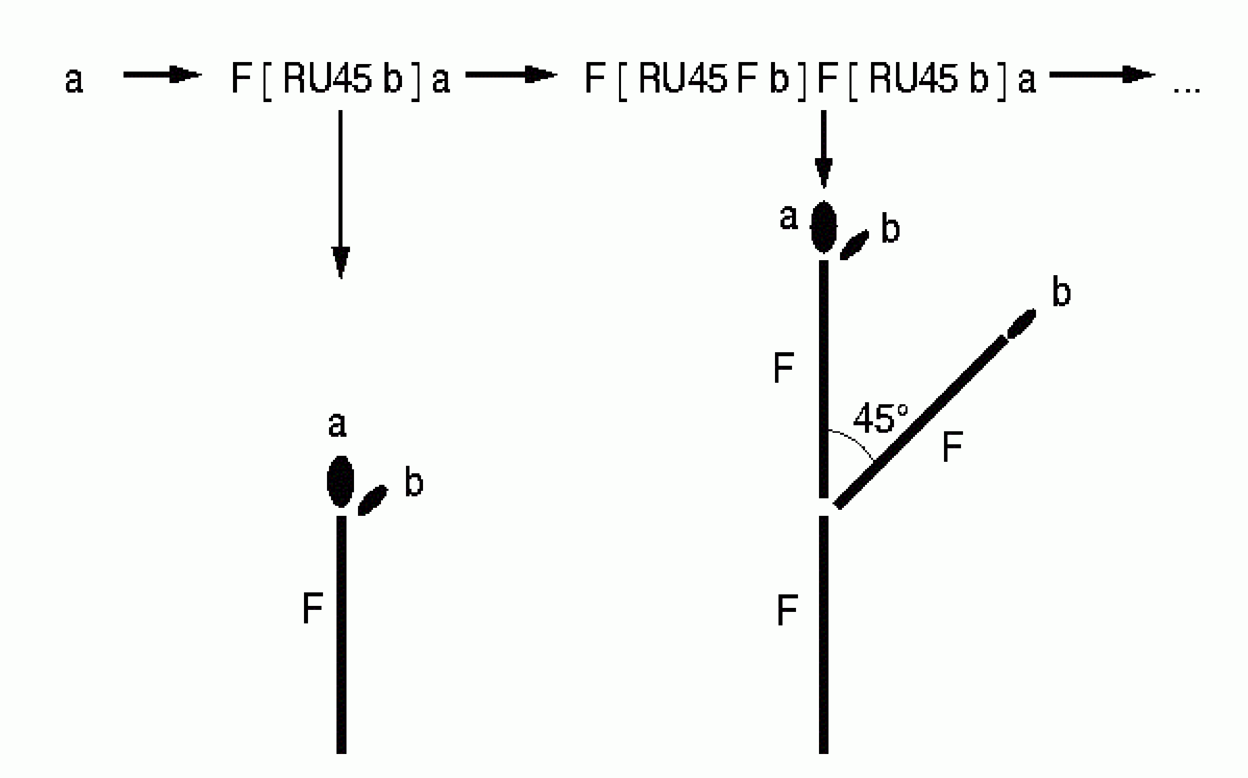
`]` take the state from the memory which was just put there and make it the current state of the turtle (finishes a branch)

Example:

Rules

a → F [RU45 b] a,
b → F b

Start word **a**



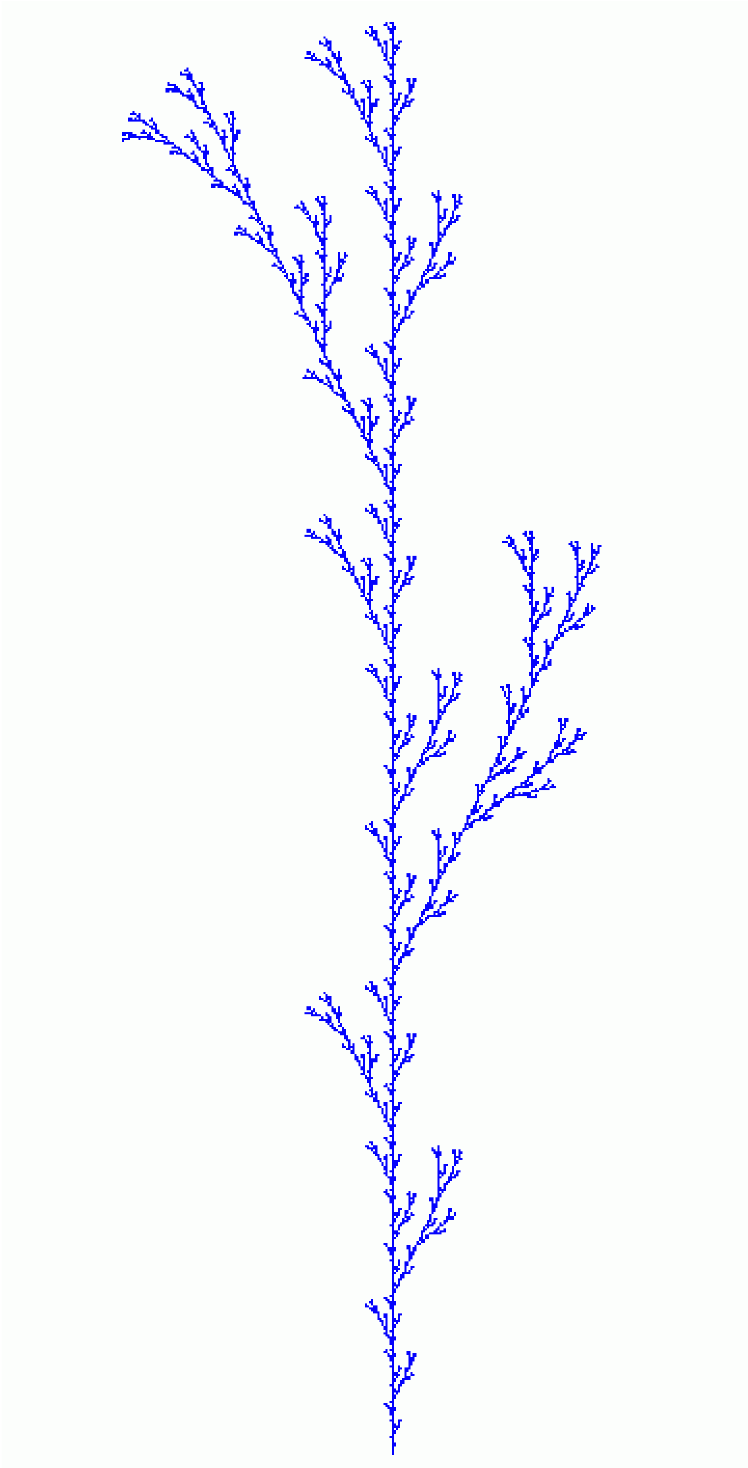
(**a** and **b** are normally not interpreted geometrically.)

Further examples:

\angle 25.7,

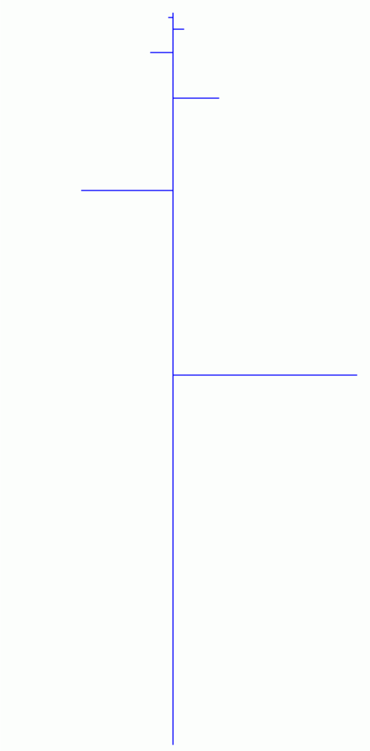
F $\rightarrow$ F [+ F] F [– F] F

Result after 7 steps:



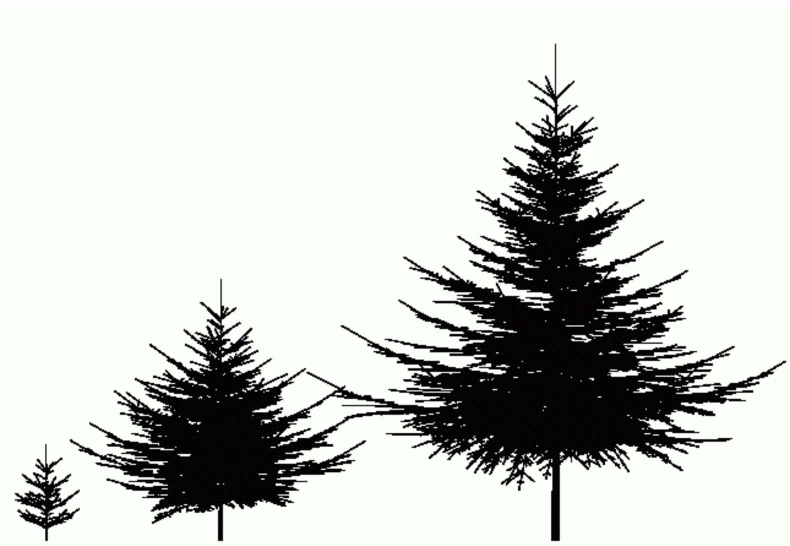
Branching, alternating orientation of branches and progressive shortening (like in real plants):

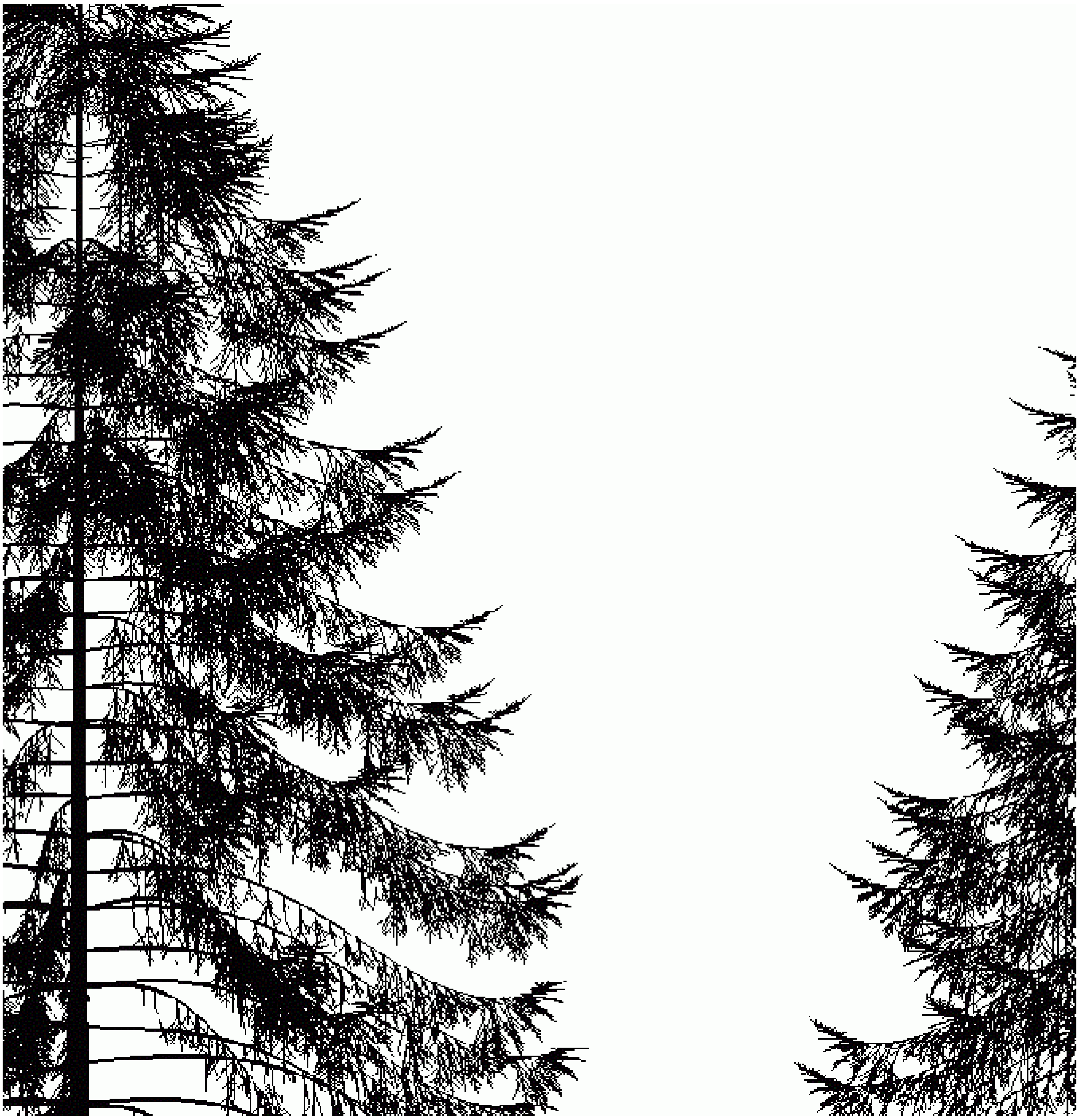
*** → F a,**
a → L*0.5 [RU90 F] F RH180 a



Examples of real-world vegetation modelled with L-systems:

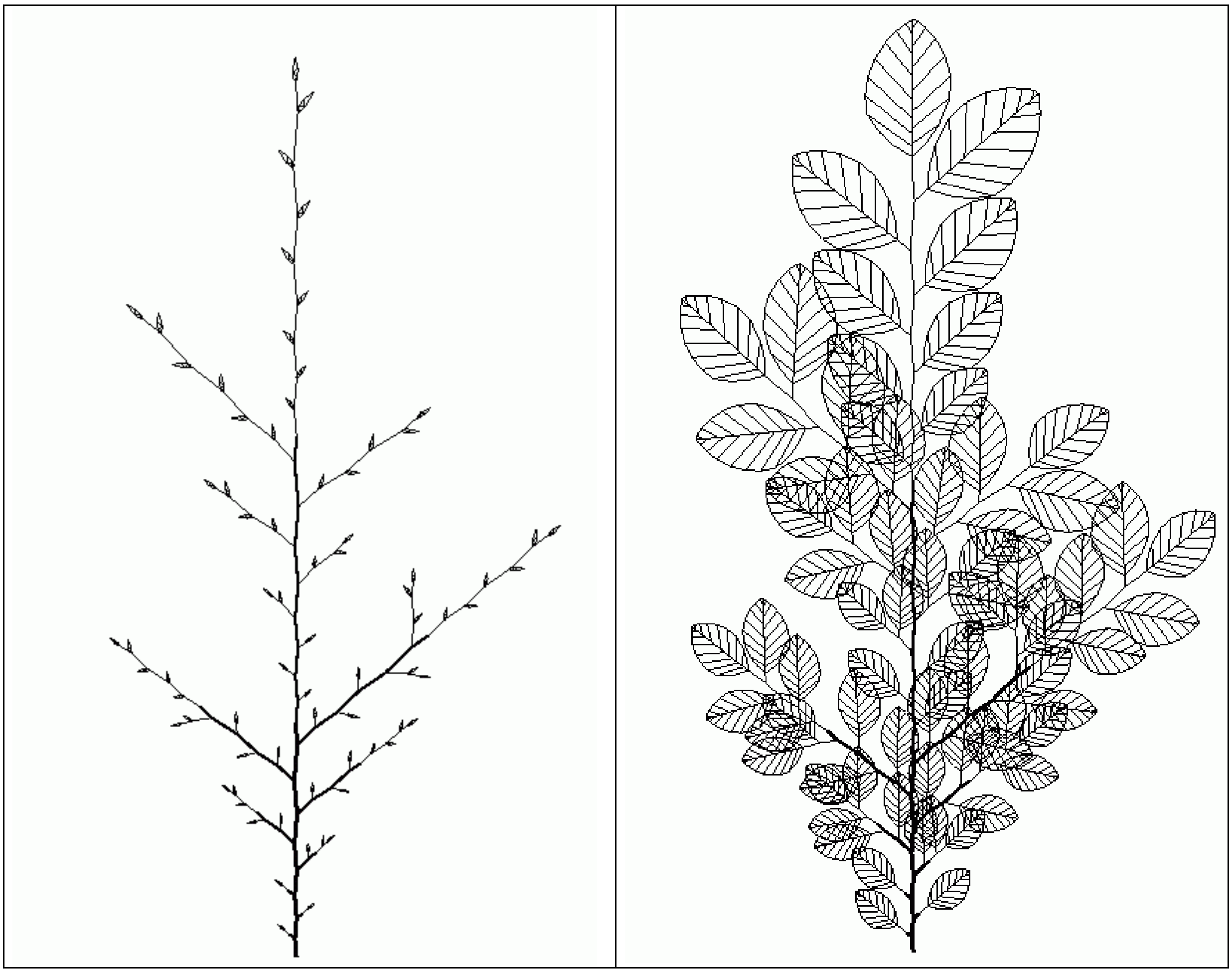
spruce trees





Example mint (by Prusinkiewicz & Lindenmayer):

Beech twigs:



Development of flowering plants: