Artificial Life
Summer 2015

Cellular Automata (CA)

Master Computer Science [MA-INF 4201]
Mon 8:30 – 10:00, LBH, Lecture Hall III.03a

Dr. Nils Goerke, Autonomous Intelligent Systems,
Department of Computer Science, University of Bonn

Modalities MA-INF 4201, Artificial Life SS15

Lecture:
Mondays, 8:30 – 10:00, Lecture Hall III.03a, LBH
some slides will be on the respective exercises page.
They will not cover the complete lecture !!!

Assignments: on a **voluntary** basis
11 times weekly assignments, pencil & paper and programming,
work in 2 person groups, >50% points are favorable.

Exercise groups:
weekly, 2hrs, participation is **voluntary**

Exam: written examination, 100 minutes
probably between Mon 27.7.15 – Fri 31.7.2015

Access to Slides & Exercise Sheets

The exercise sheets, and some of the slides from the lecture will be available on the web page for the module:
www.ais.uni-bonn.de/SS15/4201_L_AL.html

There is a link to the **Exercises** page:
www.ais.uni-bonn.de/SS15/4201/4201_E_AL.html

Benutzername: **AL-SS15**
Passwort: *will be given in the lecture*

Voluntary exercise groups:

The exercises will probably start on **Tuesday 21.4.2015**

The exercises will be in the LBH building,
Friedrich-Ebert-Allee 144
Room **E.23** (ground floor),
turn right after entering the building,
almost last door on left side.

The detailed times for the exercises will be negotiated during the lecture on Mon 20.4.2015:

- **A: Tue 14-16** (confirmed)
- **B: Tue 16-18** (confirmed)
- **C: Thu 14-16** (almost confirmed)

Artificial Life
Summer 2015

Cellular Automata (CA)

Master Computer Science [MA-INF 4201]
Mon 8:15 – 10:00, LBH, Lecture Hall III.03a

Dr. Nils Goerke, Autonomous Intelligent Systems,
Department of Computer Science, University of Bonn

Cellular Automata

Cellular Automata are an approach to perform information processing.
Their structure is not based on the von-Neumann computing architecture that most of our classical computers have.
Cellular Automata are often referred to as,
NON-von-Neumann computers.

This classification sounds a bit strange,
because one of the first scientific work on the structure that we call Cellular Automata has been done and published by John von Neumann:
„*Theory and Organisation of Complicated Automata (1949)*“

Cellular Automata

John von Neumann, Stanislav Ulam and Arthur Burks, are pioneers in the field of computing. A lot of our computing structures have been proposed and developed by them.
In addition, they have worked in the 40ies and 50ies of the last century on Cellular Automata for a possibility to do information processing and computing.

In 1982 a young scientist (Stephen Wolfram), was so fascinated from the possibilities of Cellular Automata, that he started to investigate the properties and capabilities of Cellular Automata in a structured way.
He started to focus his work on investigating 1-dimensional Cellular Automata.

Cellular Automata

Cellular Automata (CA) are a discrete model of information processing.

They consist of cells, that are organized as a grid or lattice with a specific topology.

Each cell has a state (from an alphabet).
An initial state for the cells.

A (transition) rule is determining the new state of a cell with respect to the state of that cell and the states of some other cells (neighborhood).

Cellular Automata

Cellular Automata (CA) are a discrete model of information processing. They are discrete (space, time, value), and deterministic.

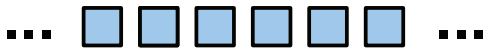
A Cellular Automaton (CA) consists of:

- a lattice of cells,
- a neighborhood,
- a finite set of states (an alphabet),
- an initial state,
- a rule, determining the next state of a cell.

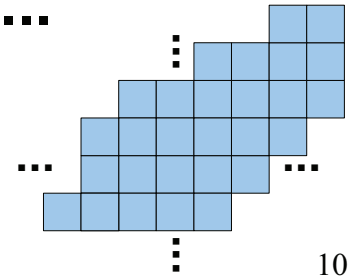
CA: lattice

The **lattice or grid** of a Cellular Automaton consists of cells. The grid is typically organized as a regular, rectangular grid in one or two dimensions.
This is denoted as dim=1 or dim=2, or d=1, d=2.

Thus, a **d=1** CA is just a chain of cells adjacent to each other.

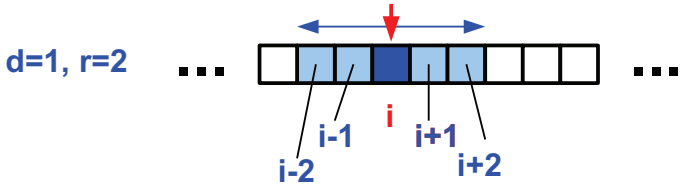


In **d=2** dimensions, a normal CA has a rectangular grid of cells, that cover the 2-dim plane in a regular way.
A dim>2 is possible but very unusual.



CA: neighborhood

The **neighborhood** of a specific cell is a set of cells from the CA that is (typically) in the direct vicinity of that very cell.
In one dimensional CAs the neighborhood is defined as those cells that have a distance closer or equal to a given neighborhood-radius **r** to the cell.



The (d=1,r=2) neighborhood for cell i: **{** i-r, ..., i-1, **i**, i+1, ..., i+r **}**

CA: neighborhood

The size of the **neighborhood,** the number n of cells for the d=1, r case is:

n = 2*r + 1

The cell **i**, itself is part of the neighborhood.

The cells of the neighborhood without the cell are called: periphery.

Typical values for d=1, are **r=1** and **r=2**.

Remark: **r=0** is rather unusual, but is explicitly allowed.

CA: states

Each of the cells of the CA can have a state from the finite **set of states, (alphabet).**

The number of allowed states (the size of the alphabet) is typically denoted with: **k**

In a lot of cases, the CA has only 2, binary states, with **k=2** , and with the binary set = **{ 0 , I }**

The states don't have to be numerical values, they can be e.g. letters, items, or colors.

Sometimes the two states {0,I} are called {dead, alive}.

CA: initial state

The states of all cells from the CA, at the beginning is called **initial state**.

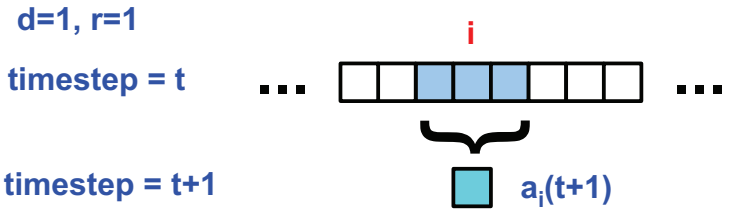
Since the computational complexity of a CA is tremendous, the effect of the initial state is only investigated in parts.

There are three usual ways to initialize a CA:

- by **random**, the state of each cell is set randomly,
- as a **seed**, only one cell is „set“, all other cells are „0“,
- an initial **pattern**, to be investigated further.

CA: rule

The **rule,** (or **transition rule**) is determining the next state of a cell within the CA.
The rule takes the state of all cells from the neighborhood of cell **i**, at time **t**,
to yield the next state $a_i(t+1)$ of cell **i** for timestep **t+1**.



The rule is applied for all cells, to get the next state of the CA.

CA: rule

The rule is applied in a synchronous way, which means, that all cells are updated for time-step t+1 at the same time.
The rule is using only states from time-step t, to determine the states for time-step t+1.

If all cells of the CA obey the same rule,
have the same neighborhood, and the same set of states,
the CA is called to be homogeneous.

When working with a finite lattice, the cells at the *border* may have some extra rule defined.

CA: rule example

An example rule for a simple Cellular Automaton:
with d=1, r=1, k=2; states {□, ■}

Thus, we have a one dimensional row of cells,
a neighborhood with n = 2*r +1 = 2*1 +1 = 3 cells,

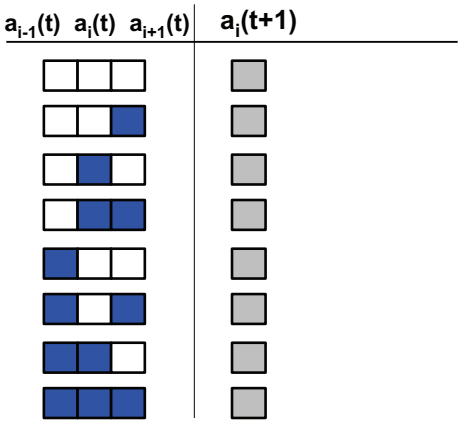
Each of these neighborhood cells can have
one out of 2 states (k=2),

Thus, we have a total of $k^n = k^{(2*r+1)} = 2^{(2*1+1)} = 2^3 = 8$
possible states for the neighborhood.

The rule has to implement a mapping for all possible states
of the neighborhood to one of the allowed states for cell i,

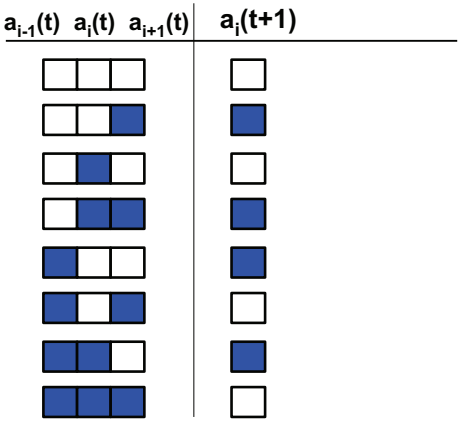
CA: rule example

The rule can be implemented and visualized as a **table,**
mapping each of the possible states of the neighborhood
onto one of the states from the set, here d=1, r=1, k=2.



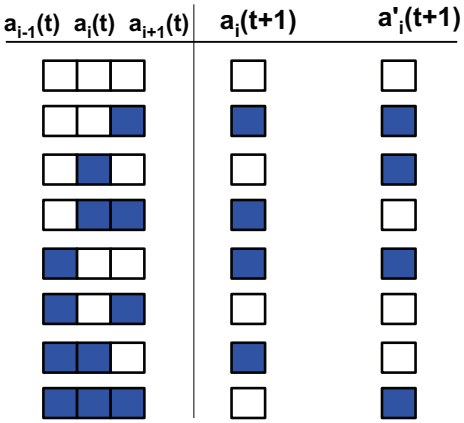
CA: rule example

The rule can be implemented and visualized as a **table,**
mapping each of the possible states of the neighborhood
onto one of the states from the set, here d=1, r=1, k=2.



CA: rule example

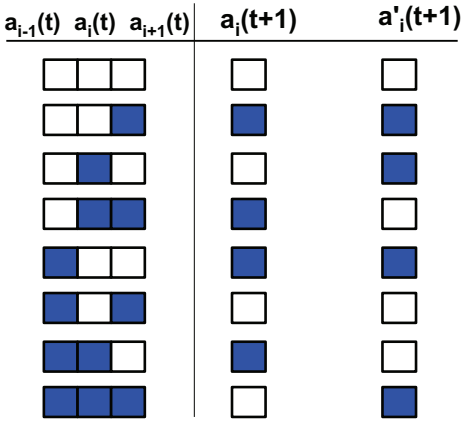
The rule can be implemented and visualized as a **table,**
mapping each of the possible states of the neighborhood
onto one of the states from the set, here d=1, r=1, k=2.



Two examples for rules
with d=1, r=1, k=2

CA: rule example

The rule can be implemented and visualized as a **table,** mapping each of the possible states of the neighborhood onto one of the states from the set, here d=1, r=1, k=2.



How many (different) rules can you define for a d=1, r=1, k=2, CA ?

Two examples for rules with d=1, r=1, k=2

CA: many rules

How many (different) rules can you define for a CA with d=1, r=1, k=2 ?

The table of such a rule has $L = k^{(2*r+1)} = 2^3 = 8$ Lines, Therefore we have to compute all vectors with the length of 8 entries, and k=2 possible states per entry:

$Z = k^L = k^8 = 256$ possible rules.

As a complete formula:

$$Z = k^L = k^{k^{(2*r+1)}}$$

CA: many rules

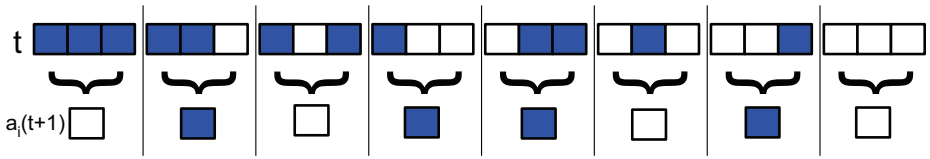
The amount Z of possible rules for even small numbers of k and r is tremendous (already in the one dimensional case). It is by no means thinkable of investigating or testing all rules in a complete, systematic way.

e.g. d=1, r=2, k=2, => a neighborhood of n=2*r+1 = 5 cells and a rule table with $L= 2^5 = 32$ lines, and a total of possible rules: $Z = k^L = 2^{32} = 4$ Giga.

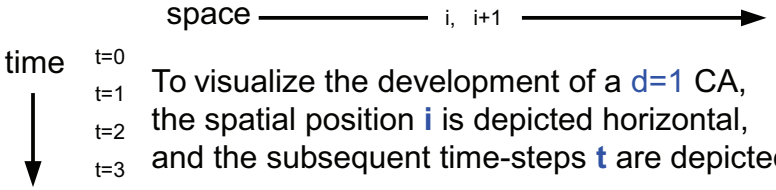
e.g. d=1, r=2, k=8, => the neighborhood can have 32768 different states ($L= k^{(2*r+1)} = 8^5= 32768$), and a total of rules $Z = k^L = 8^{32768}$

CA: rule example

Sometimes the rule can be visualized easily by a picture:



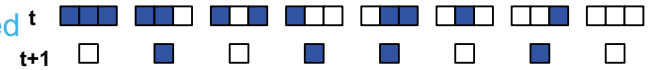
To calculate the CA, we have to apply the rule for every cell, and for every time-step, starting with t=0, the initial state.

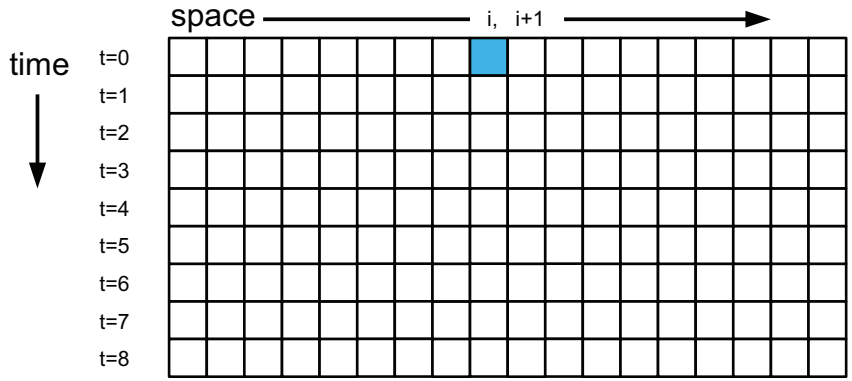


To visualize the development of a d=1 CA, the spatial position **i** is depicted horizontal, and the subsequent time-steps **t** are depicted vertical.

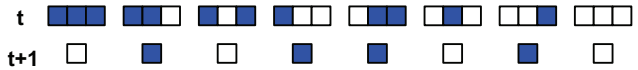
CA: rule example

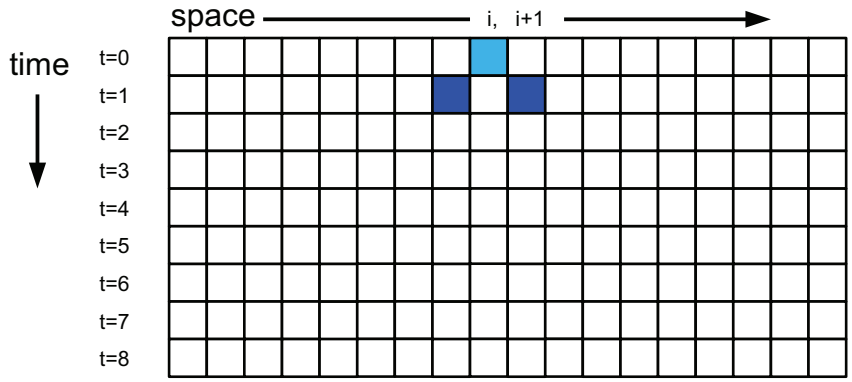
Initial condition: seed
just one cell is set.



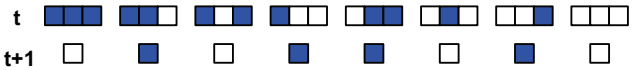


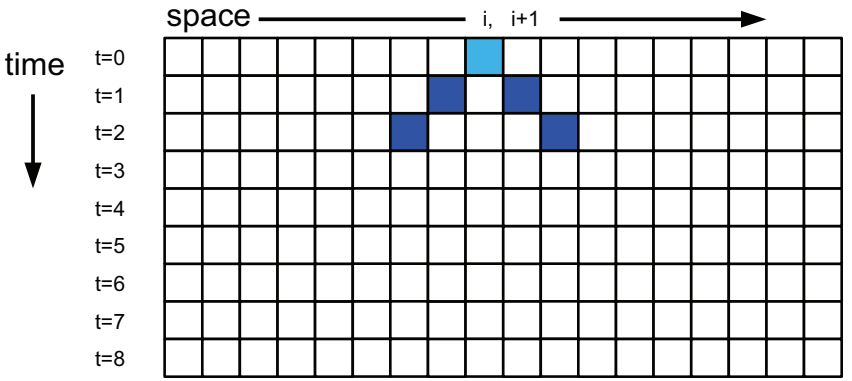
CA: rule example



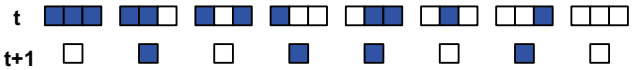


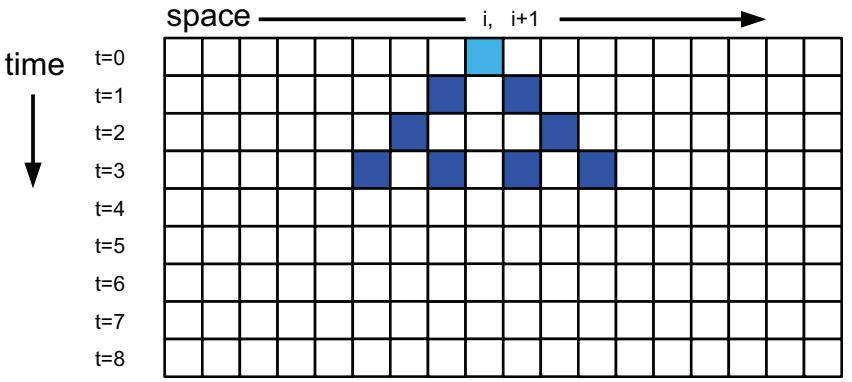
CA: rule example



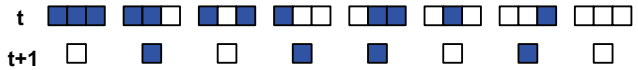


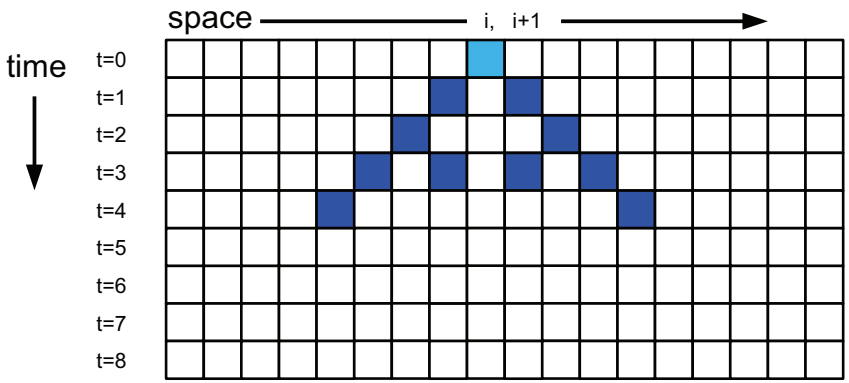
CA: rule example



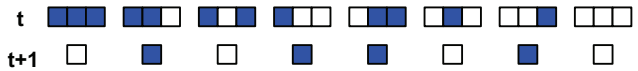


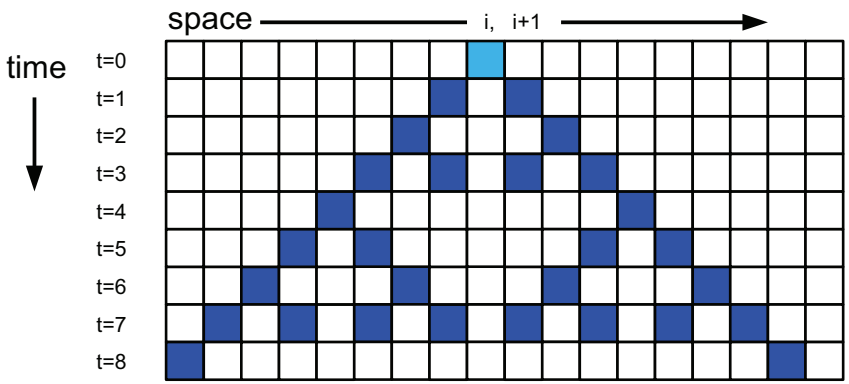
CA: rule example



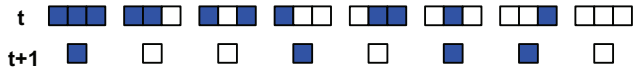


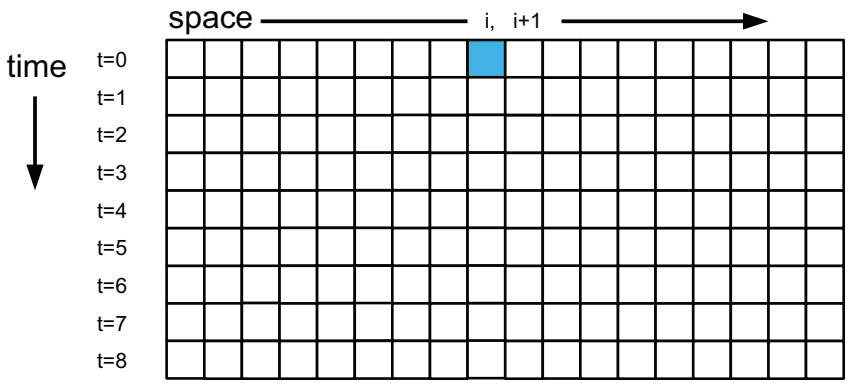
CA: rule example



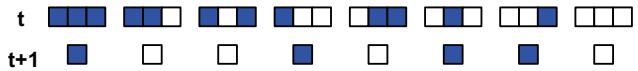


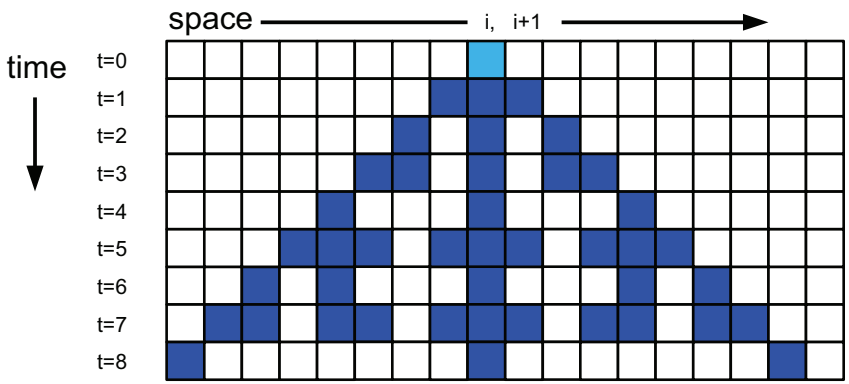
CA: rule example





CA: rule example





CA: rule properties

For easier handling, and grouping the rules, S.Wolfram proposed to define some terms, aligned with the properties of the rules:

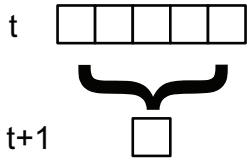
- silent state,
- symmetric rules,
- legal rules,
- peripheral rules
- totalistic rules.

CA: rule properties

Silent state:

A rule is denoted to have a silent state, if the state of the neighborhood with all cells „unset“ or set to state „0“ is mapped onto the state „0“

e.g. d=1, r=2, k=2, states { 0= , I= }



Especially, when all cells of the CA are „unset“ the CA remains calm; the „silent state“ persists.

CA: rule properties

Symmetric rules:

A rule is denoted to be symmetric if states and mirrored states yield the same result for the next state of the cell.

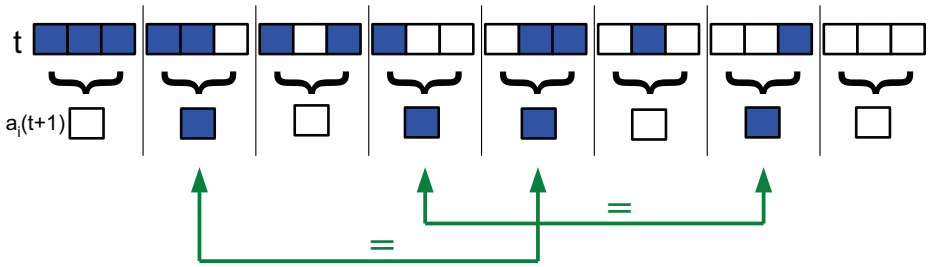
e.g. d=1, r=1, k=2, states { 0= , I= }

These pairwise states must yield the same result:

The other states don't care because they are identical to their mirrored state:

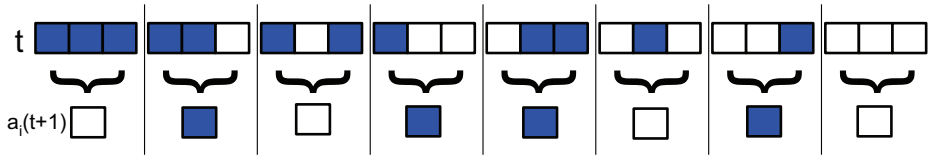
CA: rule properties

Is this rule symmetric? **Yes, this rule is symmetric.**



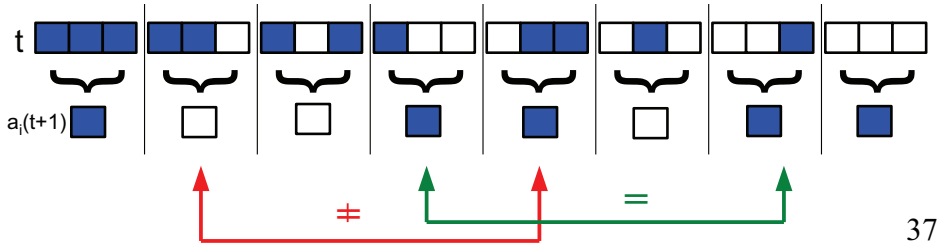
CA: rule properties

This rule is symmetric.



Is this rule symmetric?

No, this rule is not symmetric.



CA: rule properties

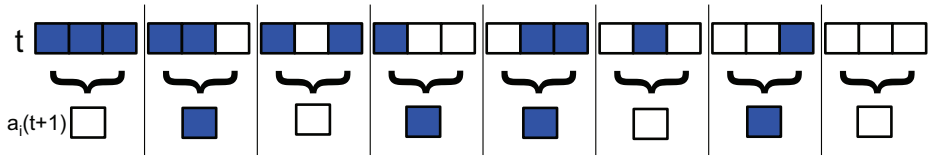
Legal rules:

A rule is denoted **legal**, or a „**legal rule**“, if it has the following two properties:

- the rule must be **symmetric**
- and must have a **silent state**.

Is this rule **legal?**

Yes, this rule is legal: it is symmetric, and has a silent state.



CA: rule properties

Peripheral rule:

A rule is denoted peripheral, if the state of the cell itself is not influencing the result.
The next state of the cell depends only of the periphery, the state of the cell itself is regarded as „don't care“.

e.g. d=1, r=1, k=2, states { 0=□, I=■ }

These pairwise states must yield the same result:

CA: rule properties

Totalistic rule:

A rule is denoted totalistic, if only the sum of the cells that are set, within the neighborhood determine the next state.
The table for the rule thus, depends only on the
$SUM(t) = a_{i-r}(t) + \ldots + a_{i-1}(t) + a_i(t) + a_{i+1}(t) + \ldots + a_{i+r}(t)$
(if k=2, and the set is { 0 , I })

SUM(t)	3	2	1	0
$a_i(t+1)$	**I**	**0**	**0**	**I**

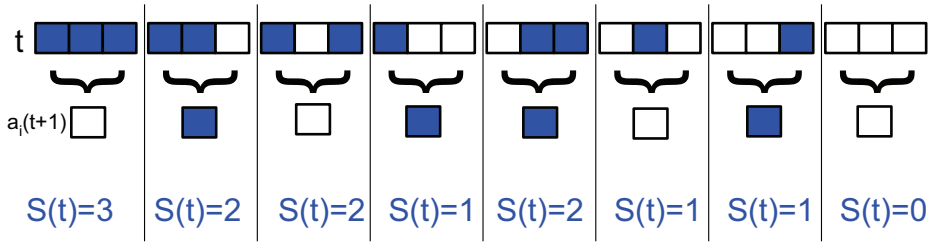
d=1, r=1, k=2

SUM(t)	5	4	3	2	1	0
$a_i(t+1)$	**I**	**0**	**I**	**0**	**0**	**I**

d=1, r=2, k=2

CA: rule properties

Is this rule totalistic?

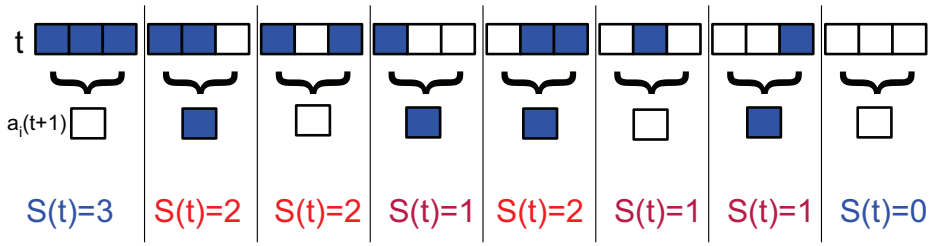


SUM(t)	3	2	1	0
$a_i(t+1)$	?	?	?	?

d=1, r=1, k=2

CA: rule properties

Is this rule totalistic? **No, this rule is not totalistic.**



SUM(t)	3	2	1	0
$a_i(t+1)$	0	?	?	0

d=1, r=1, k=2

CA: rule properties

Totalistic rule:

For CAs with different states than {0, I}, or more than k=2 states, the way how to calculate the SUM is not obvious.

If the SUM is defined properly, it is still possible to talk about totalistic rules.
Typically, SUM denotes the number of cells that are „set“, that are not in silent state.

Other, exotic, definitions might apply as well, if the SUM is defined reasonable.

CA: rule properties

Totalistic rule:

For CAs with different states than {0, I}, or more than k=2 states, the way how to calculate the SUM is not obvious.

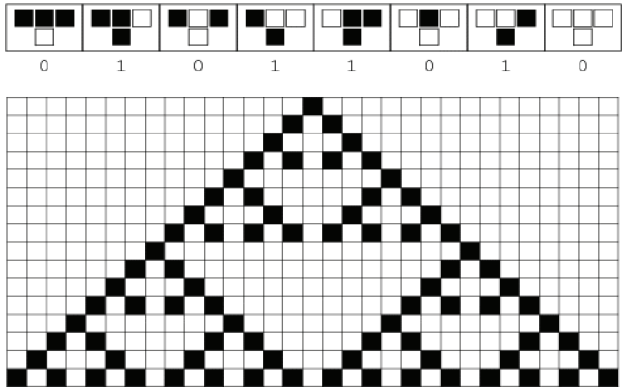
If the SUM is defined properly, it is still possible to talk about totalistic rules.
Typically, SUM denotes the number of cells that are „set“, that are not in silent state.

Other, exotic, definitions might apply as well, if the SUM is defined reasonable.

Cellular Automata: examples

Example for a 1-dim Cellular Automaton

rule 90

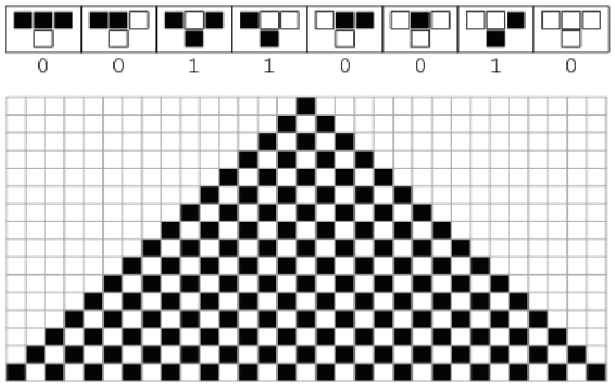


from: http://mathworld.wolfram.com/CellularAutomaton.html

Cellular Automata: examples

Example for a 1-dim Cellular Automaton

rule 50

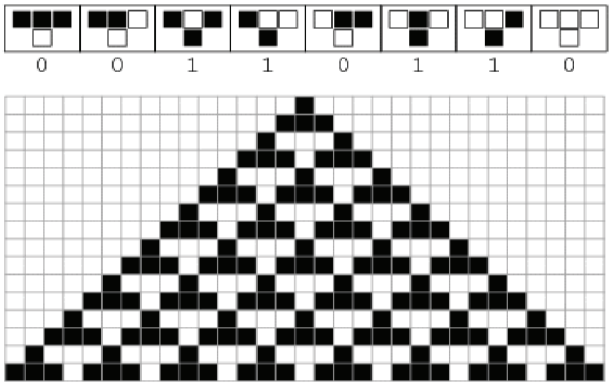


from:http://mathworld.wolfram.com/CellularAutomaton.html

Cellular Automata: examples

Example for a 1-dim Cellular Automaton

rule 54

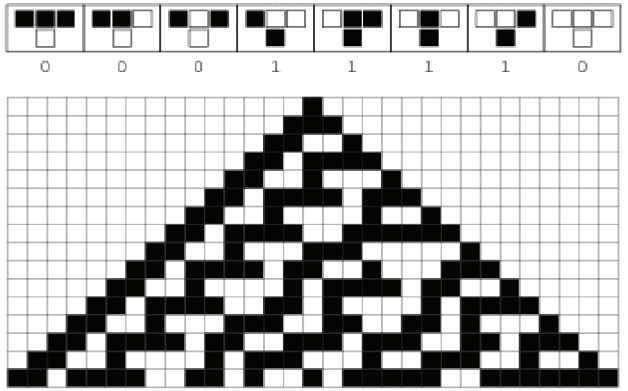


from: http://mathworld.wolfram.com/CellularAutomaton.html

Cellular Automata: examples

Example for a 1-dim Cellular Automaton

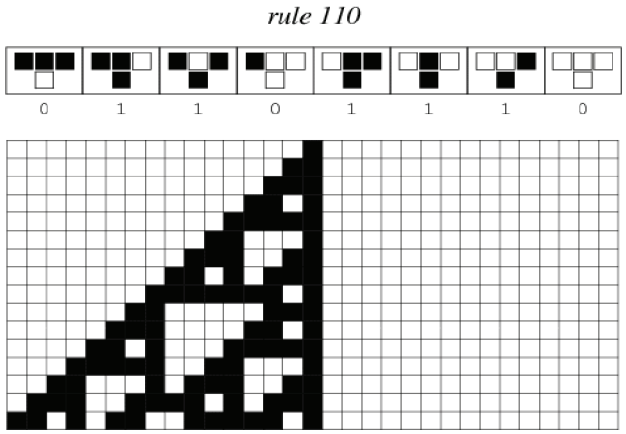
rule 30



from: http://mathworld.wolfram.com/CellularAutomaton.html

Cellular Automata: examples

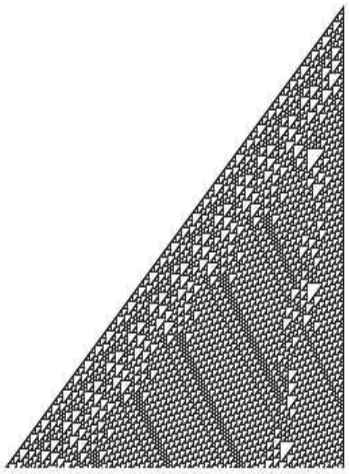
Example for a 1-dim Cellular Automaton

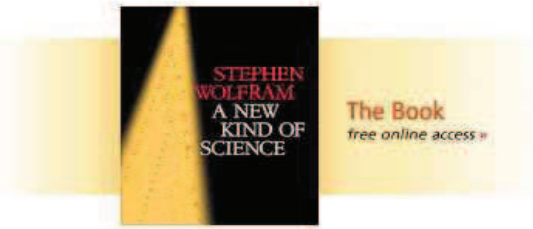


from: http://mathworld.wolfram.com/CellularAutomaton.html

Cellular Automata: examples

Example for a 1-dim Cellular Automaton

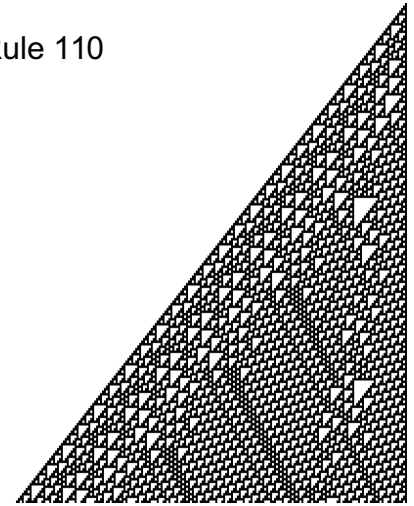




from: http://mathworld.wolfram.com/CellularAutomaton.html

Cellular Automata: examples

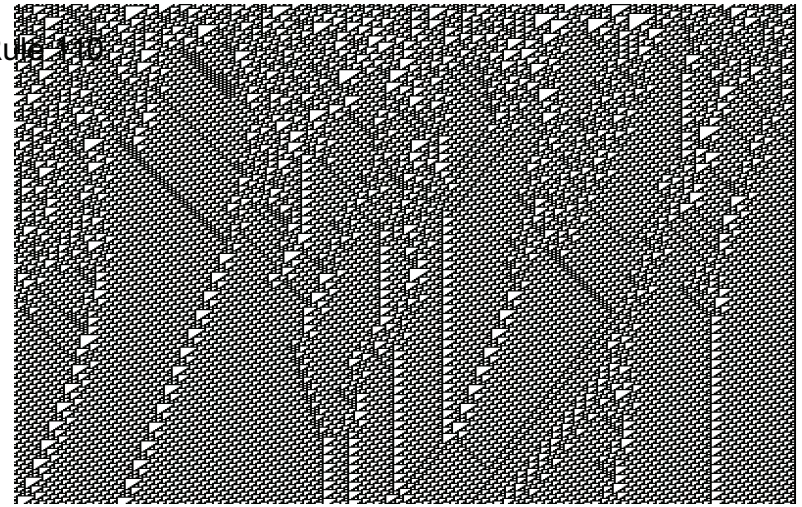
Rule 110



from: http://liinwww.ira.uka.de/~rahn/ca.pics/

Cellular Automata: examples

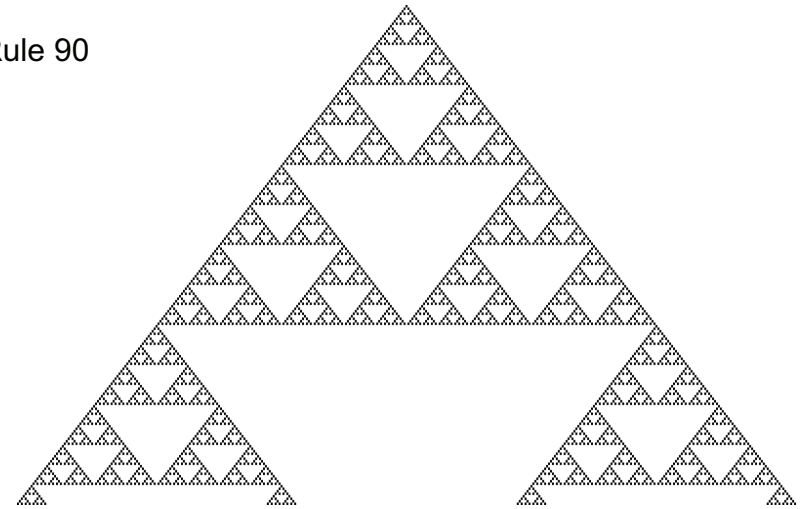
Rule 110



from: http://liinwww.ira.uka.de/~rahn/ca.pics/

Cellular Automata: examples

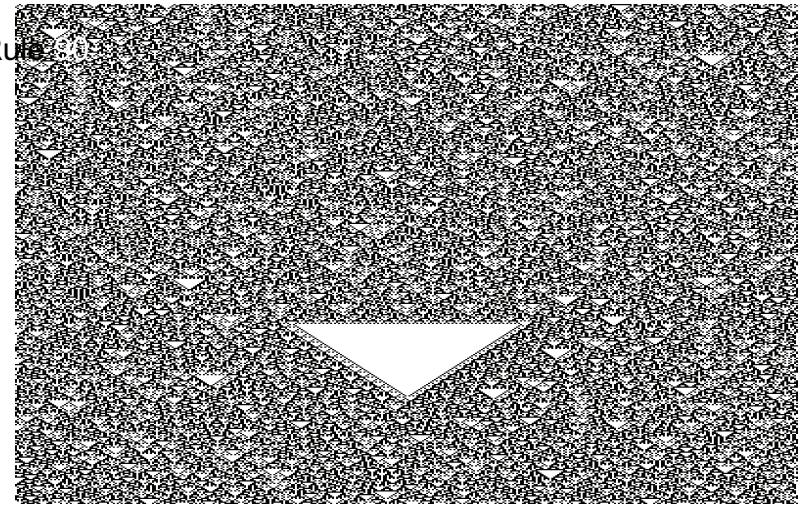
Rule 90



from: http://liinwww.ira.uka.de/~rahn/ca.pics/

Cellular Automata: examples

Rule 30



from:: http://liinwww.ira.uka.de/~rahn/ca.pics/

CA: Wolfram number

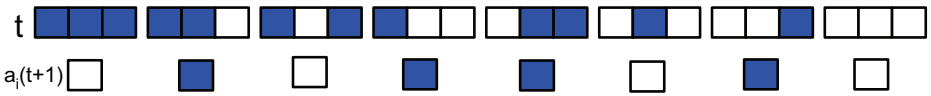
For 1-dim CA, with a neighborhood radius of r=1, and the binary states k=2, {0,I}, Stephen Wolfram has proposed a numbering system for easier access to the rules.

Since in this case, d=1,r=1,k=2 the right hand side of the rule table is a binary vector with L=8 lines, yielding a total $Z=2^8=256$ rules a binary numbering system comes to mind.

Identifying each of the output states with a power of 2, the rule can be converted into a decimal number.

CA: Wolfram number

For the CA, *d=1,r=1,k=2*, each of the output states is identified with a power of 2.
Thus the rule can be converted into a decimal number



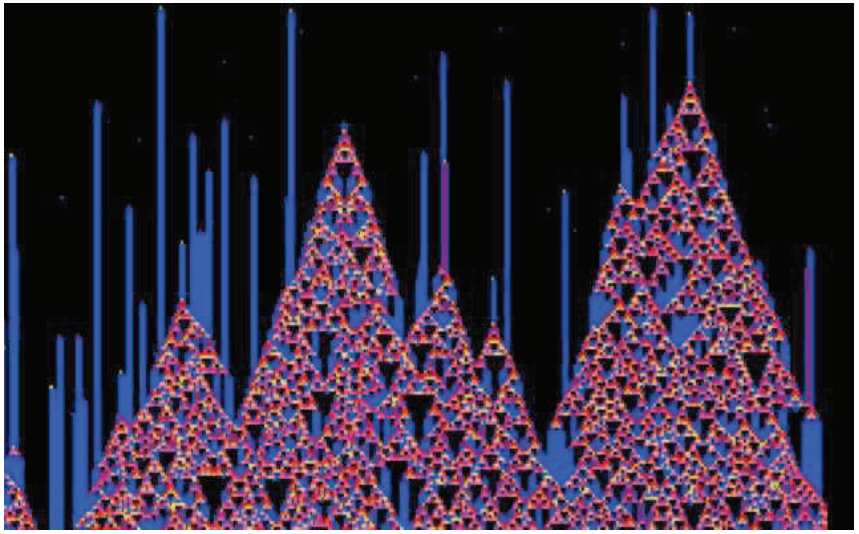
0 I 0 I I 0 I 0

$$0*2^7 \quad 1*2^6 \quad 0*2^5 \quad 1*2^4 \quad 1*2^3 \quad 0*2^2 \quad 1*2^1 \quad 0*2^0$$

0*128 + 1*64 + 0*32 + 1*16 + 1*8 + 0*4 + 1*2 + 0*1

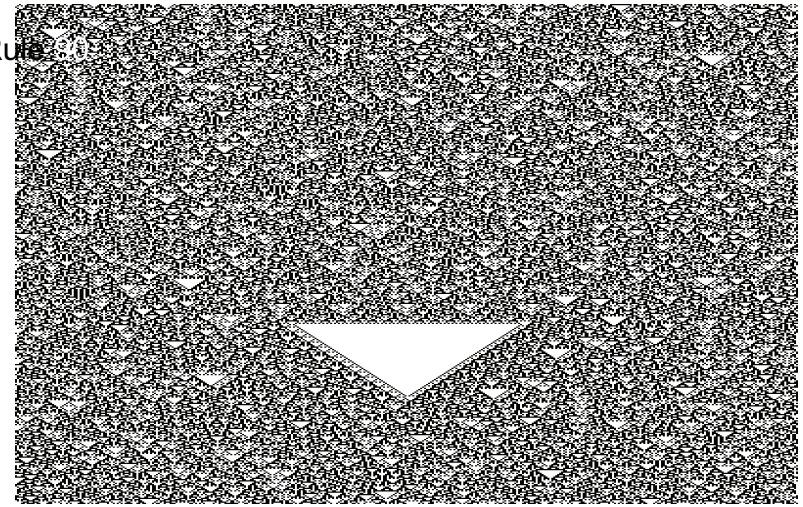
= 90_D = $0I0II0I0_B$ = $5A_{HEX}$

Cellular Automata: examples



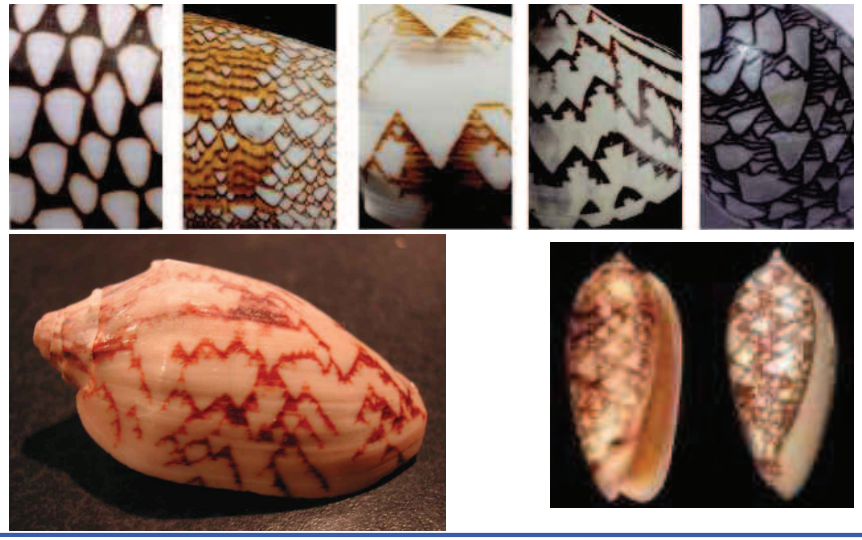
Cellular Automata: examples

Rule 90:



from:: http://liinwww.ira.uka.de/~rahn/ca.pics/

Cellular Automata: examples



Boundary in CA Grids

In real Cellular Automata the underlying grid is typically not infinite in size. Therefore several approaches have been proposed to cope with this problem.

- Virtually infinite (no boundary)
- Periodic, cyclic topology of the grid
- Fixed boundary (assigned boundary)
- Random boundary (assigned boundary)
- Adiabatic boundary (copy, mirror)
- Open boundary (absorbing boundary)
- Closed boundary (reflecting boundary)

The first four approaches for implementing a boundary condition are the typical ones for CAs.

Virtually Infinite Grid

To virtually implement an infinite grid for the CA, the size of the grid can be:

- larger than the expected space needed,
- or enhanced, enlarged whenever necessary.

A virtually infinite grid is reasonable when the starting condition is a seed, or a (small) fixed pattern.

Implementing a virtually infinite grid requires a smart and fast memory management for the CA simulation tool.

Periodic, Cyclic Grid Toplogy_

To circumvent any problems of boundary conditions, the grid can be made periodic, or cyclic, by wrapping around the grid, gluing one end of the grid directly with the grid on the opposite side.

The implementation of a cyclic grid is easy,
by just doing all index calculations using the *modulo* function,
Thus leaving the grid on one side, is entering the grid on the other side.

In one dimension (d=1) a line becomes a *ring* or *circle.*
In two dimensions (d=2) a rectangle becomes a *torus.*
In three dimensions (d=3) a block becomes a *hyper-torus*.
In 4 dimensions, …

Fixed Boundary

The cells at the boundaries of the CA are set to a predefined, fixed value.

Implementation can be done, by:
- not applying the update rule for these cells,
- re-setting the state of these cells to the predefined values.

The size of this fixed boundary should be set to the neighborhood radius r, for not having any undefined values within the neighborhood.

It is often a good choice, to use the silent state for these cells.

A fixed, random boundary is a special case.

Boundary in CA Grids

Implementing an Adiabatic Boundary (copy, mirror) the grid is extended by some cells (r) that are „behind the boundary“. The values of these cells mirror the state of those grid cells that are „before the boundary“.

The other variants of boundary conditions Open Boundary (absorbing boundary), or Closed Boundary (reflecting boundary), require deeper knowledge of the dynamical properties that are to be modeled with the specific CA, and can be complex to realize.
In general, they are only used in specialized applications (e.g. simulation of diffusion processes, or behavior of quasi-particles like Solitons).

4 Classes of Behavior

Cellular Automata develop a tremendous variety of patterns, with respect to their rule, and their initial state.
S. Wolfram has investigated the CA in an systematic way, and found some typical behaviors that occurred several times. Thus, he tried to sort the resulting long term development of the CA into **4 Classes of Behavior.**

To determine the class of behavior, the CA is initialized with a random init pattern, and iterated for a long time.
This is repeated for several random initializations, then the typical, occurring dynamics of the CA is classified.

The Wolfram classes of behavior are to some extent aligned with observations from nonlinear dynamical systems theory.

4 Classes of Behavior

Class I: Homogeneous

Homogeneous state for all cells (mostly silent state)

Class II: Periodic

Periodic, oscillatory patterns (incl. stable patterns)

Class III: Chaotic

Deterministic chaos, no periodicity is observable.

Class IV: Complex, Patterns, „*Self Organization*“

Interesting structures evolve, persist, seem to interact, and generate new structures.

4 Classes of Behavior

Class I: Homogeneous

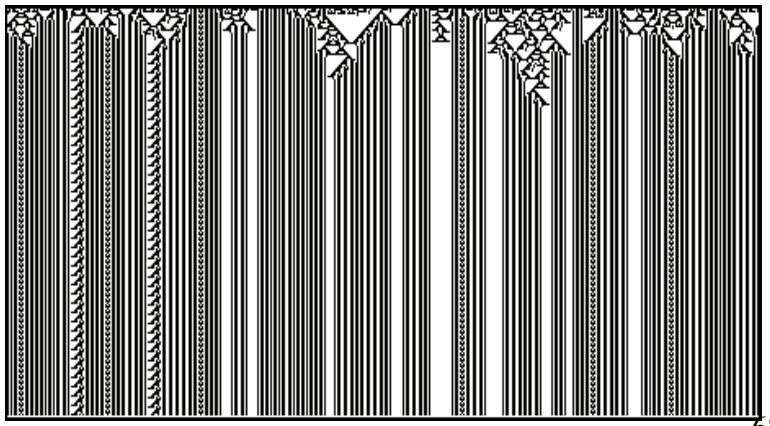
Homogeneous state for all cells (mostly silent state)

http://classes.yale.edu/fractals/CA/CAPatterns/Wolfram/Wolfram1.html

4 Classes of Behavior

Class II: Periodic

Periodic, oscillatory patterns (incl. stable patterns)



http://classes.yale.edu/fractals/CA/CAPatterns/Wolfram/Wolfram2.html

4 Classes of Behavior

Class III: Chaotic

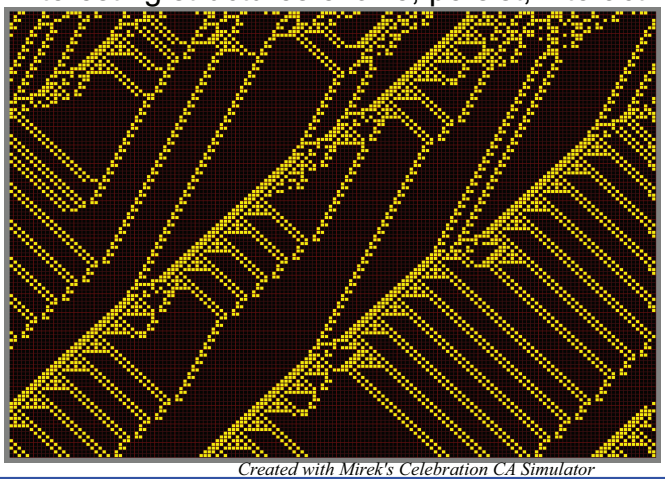
Deterministic chaos, no periodicity is observable.

Created with Mirek's Celebration CA Simulator

4 Classes of Behavior

Class IV: Complex, Patterns, „*Self Organization*“

Interesting structures evolve, persist, interact.

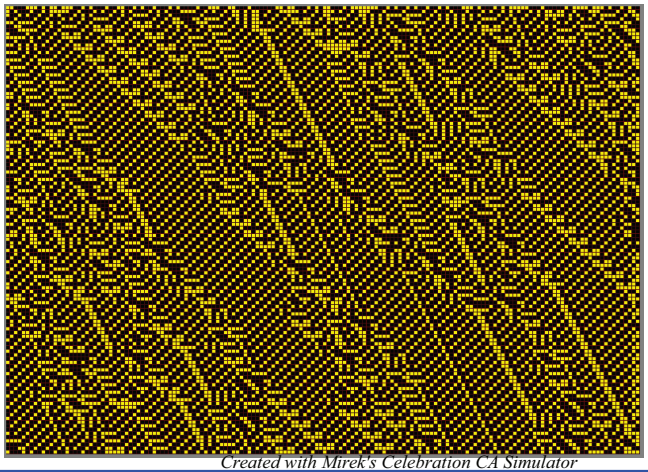


Created with Mirek's Celebration CA Simulator

4 Classes of Behavior

Class IV: Complex, Patterns, „*Self Organization*“

Interesting structures evolve, persist, interact.



Created with Mirek's Celebration CA Simulator

Artificial Life
Summer 2015

Cellular Automata (CA)

Thank you for listening

Master Computer Science [MA-INF 4201]
Mon 8:30 – 10:00, LBH, Lecture Hall III.03a

Dr. Nils Goerke, Autonomous Intelligent Systems,
Department of Computer Science, University of Bonn