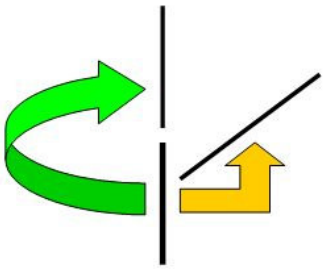
Relationale Wachstumsgrammatiken und andere L-System-Verallgemeinerungen

Der Schritt zu relationalen Wachstumsgrammatiken

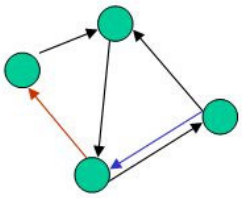
Nachteil von L-Systemen:

- in L-Systemen mit Verzweigungen (über Turtle-Kommandos) nur 2 mögliche Relationen zwischen Objekten: "direkter Nachfolger" und "Verzweigung"



Erweiterungen:

- Zulassen weiterer Relationstypen (beliebig wählbar)
- Zulassen von Zyklen (→ Graph-Grammatik)

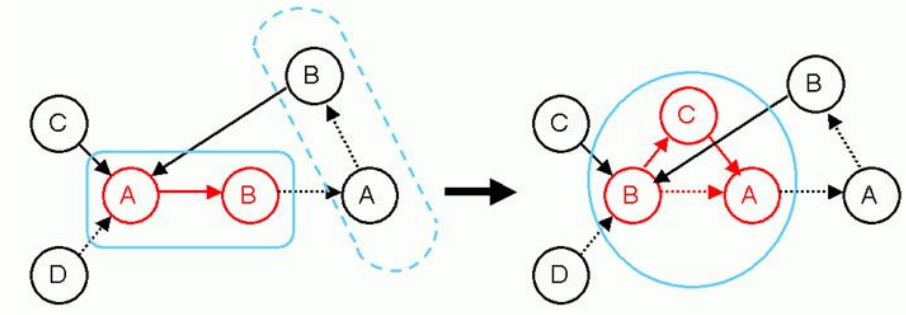


ebenfalls regelbasierter Mechanismus:

Graph-Grammatiken

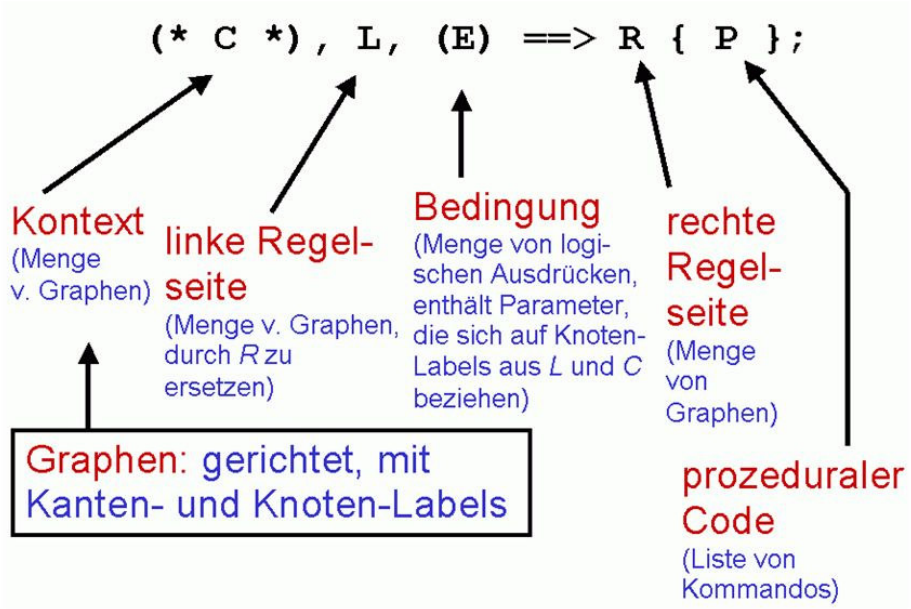
Regel:

Anwendung:



RELATIONALE WACHSTUMSGRAMMATIKEN
(RGG: Relational Growth Grammars, parallele Graph-Gramm.)

Aufbau einer Regel einer RGG:



Kanten-Markierungen repräsentieren verschiedene Arten von Relationen:

- ist Nachbar von
- enthält
- trägt
- codiert (genetisch)
- ist gepaart mit
- (...)

→auch möglich: Darstellung von multiskalierten Strukturen

Standard-Kantentypen:

successor (> oder *blank*), branch (+> oder erste Kante bei Klammern [...]), refinement (/>)

RGG als Verallgemeinerungen von L-Systemen:

Zeichenketten entsprechen speziellen Graphen

In Textform schreiben wir allgemeine (selbstdefinierte) Kanten als `-kantensorte->`

Kanten des speziellen Typs "Nachfolger" werden meist als Leerzeichen geschrieben (statt `-successor->`)

Sonderformen von RGG-Regeln:

Aktualisierungsregeln (Regelpfeil `::>`): es werden nur Parameter verändert

Beispiel: `s:Sphere ::> s[radius] += increment;`

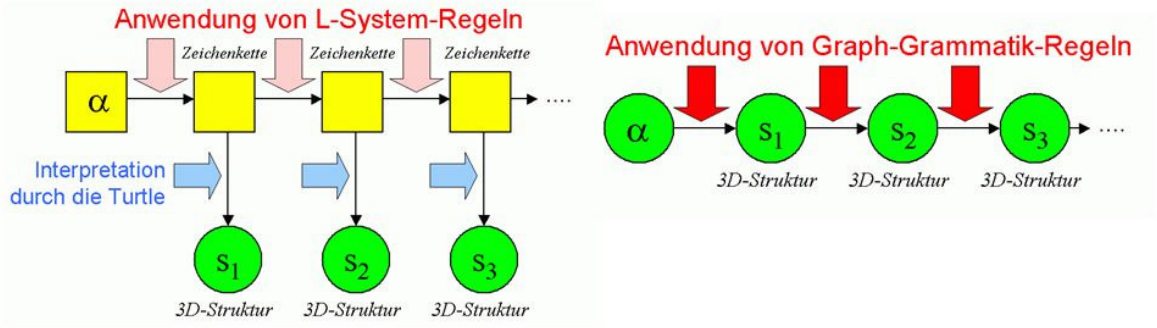
Instanzierungsregeln: einzelne Zeichen werden in Substrukturen aufgelöst, ohne Einfluss auf den nächsten Entwicklungsschritt
(Regel muss dann direkt in der Moduldeklaration stehen)

Beispiel:

```
module S(float value) extends Null ==>
   { float x = value;}
   Sphere(0.1).(setShader(new RGBAShader(1,1-x,1-x)));
```

Charakteristika von RGGs:

- Grammatik modifiziert direkt den Graphen, Umweg über String-Codierung entfällt (bzw. wird nur noch für Regel-Input gebraucht)



außerdem Nachteil der Turtle-Interpretation von L-Systemen: Segmente sind nur Zylinder, keine Objekte im Sinne der OOP

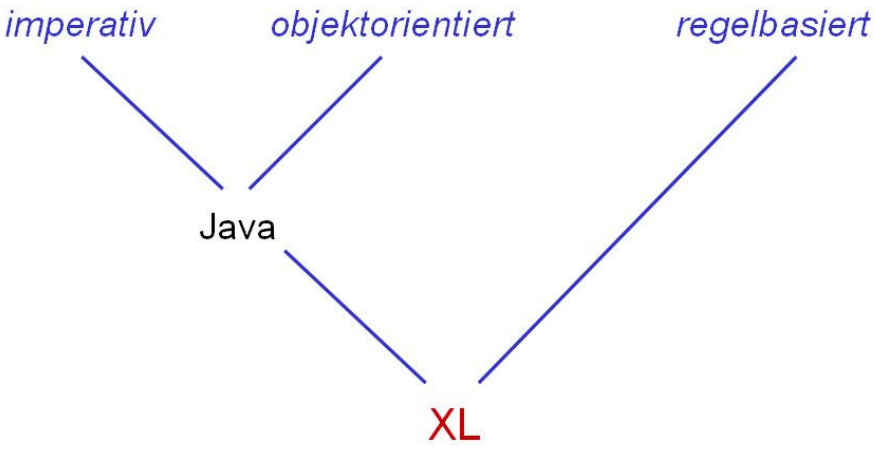
→ Erweiterungen:

- Knoten des Graphen können beliebige Objekte sein (auch Grafikobjekte)
- Einbettung von Code einer höheren, imperativen oder objektorientierten Programmiersprache in die Regeln (für uns: Java)

Programmiersprache zur Implementation von RGG: XL

„eXtended L-system language“

Programmiersprache, die parallele Graph-Grammatiken (RGG) einfach verfügbar macht



Die Sprache XL

Sprachspezifikation: Kniemeyer (2007/08)

Ole Kniemeyer: Design and Implementation of a Graph Grammar Based Language for Functional-Structural Plant Modelling. Dissertation, BTU Cottbus 2008.
http://nbn-resolving.de/urn/resolver.pl?urn=urn:nbn:de:kobv:co1-opus-5937

- Erweiterung von Java
- erlaubt zugleich Spezifikation von L-Systemen und RGG in intuitiv verständlicher Regelschreibweise

prozedurale Blöcke, ähnlich Java: { ... }

regelorientierte Blöcke (RGG-Teil): [...]

Eigenschaften der Sprache XL:

- Knoten der Graphen sind Java-Objekte, auch Geometrie-Objekte

Beispiel: XL-Programm für die Koch'sche Kurve

```
public void derivation()
  [
  Axiom ==> RU(90) F(10);
  F(x) ==> F(x/3) RU(-60) F(x/3) RU(120) F(x/3) RU(-60) F(x/3);
  ]
```

Knoten des Graphen

Kanten (Typ „Nachfolger“)

3D-Objekte in XL

Spezielle Knoten:

Geometrieobjekte

Box, Sphere, Cylinder, Cone, Frustum, Parallelogram...

Zugriff auf Properties über Parameterliste (Konstruktor):

```
Box(x, y, z)
```

oder mit setter-Methoden:

```
Box(...).(setColor(0x007700))
```

Transformationsknoten

`Translate(x, y, z)`, `Scale(cx, cy, cz)`, `Scale(c)`,
`Rotate(a, b, c)`, `RU(a)`, `RL(a)`, `RH(a)`, `RV(c)`, `RG`, ...

Lichtquellen

PointLight, DirectionalLight, SpotLight, AmbientLight

Eigenschaften der Sprache XL:

- Regeln in Blöcken [...] organisierbar, Steuerung der Anwendung durch Kontrollstrukturen

Beispiel: Regeln für den stochastischen Baum

```
Start ==> L(100) D(5) A;

A ==> F0 LMul(0.7) DMul(0.7)
  if (probability(0.5))
   ( [ RU(50) A ] [ RU(-10) A ] )
  else
   ( [ RU(-50) A ] [ RU(10) A ] );
```

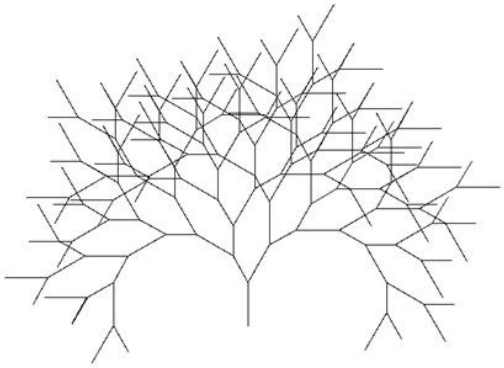
- parallele Regelanwendung
- parallele Ausführung von Zuweisungen möglich
- Operatorüberladung (z.B. „+“ für Zahlen wie für Vektoren)
- mengenwertige Ausdrücke (genauer: Producer statt Mengen)
- Graph-Abfragen (queries) zur Analyse der aktuellen Struktur

Beispiel für Graph-query:

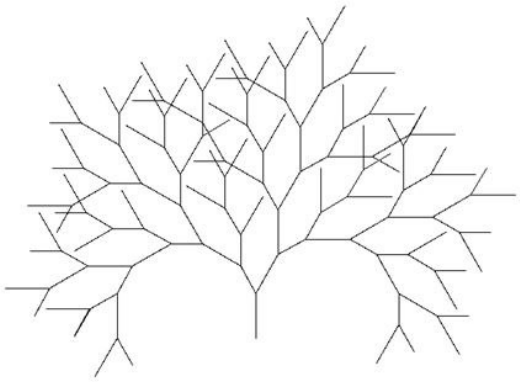
Binärer Baum, Wachstum soll nur erfolgen, wenn genügender Abstand zu anderen F-Objekten

```
Axiom ==> F(100) [ RU(-30) A(70) ] RU(30) A(100);
a:A(s) ==> if ( forall(distance(a, (* F *)) > 60) )
           ( RH(180) F(s) [ RU(-30) A(70) ] RU(30) A(100) )
```

ohne die if-Bedingung



mit der if-Bedingung



Queries bieten Möglichkeit der Verbindung von Struktur und Funktion in Organismen-Modellen

- aggregierende Operatoren (z.B. "sum", "mean", "forall", "selectWhereMin") dienen der Verarbeitung der Ergebnisse von Queries

Beispiel: suche alle Blätter, die Nachfolger des Knotens c sind, und summiere deren Fläche

```
sum((* c (>)+ Leaf *).area)
```

transitive Hüllenbildung

Ergebnis kann übergeben werden an prozedurale Berechnung

Aggregationsoperator

Query in einem Pflanzen- / Tier-Modell:

```
p:Plant,
(* a:Animal, (distance(a,p) < p[radius]) *)
```

sucht alle Tiere innerhalb des Radius von p

was ist von der in XL erzeugten Graph-Struktur sichtbar?

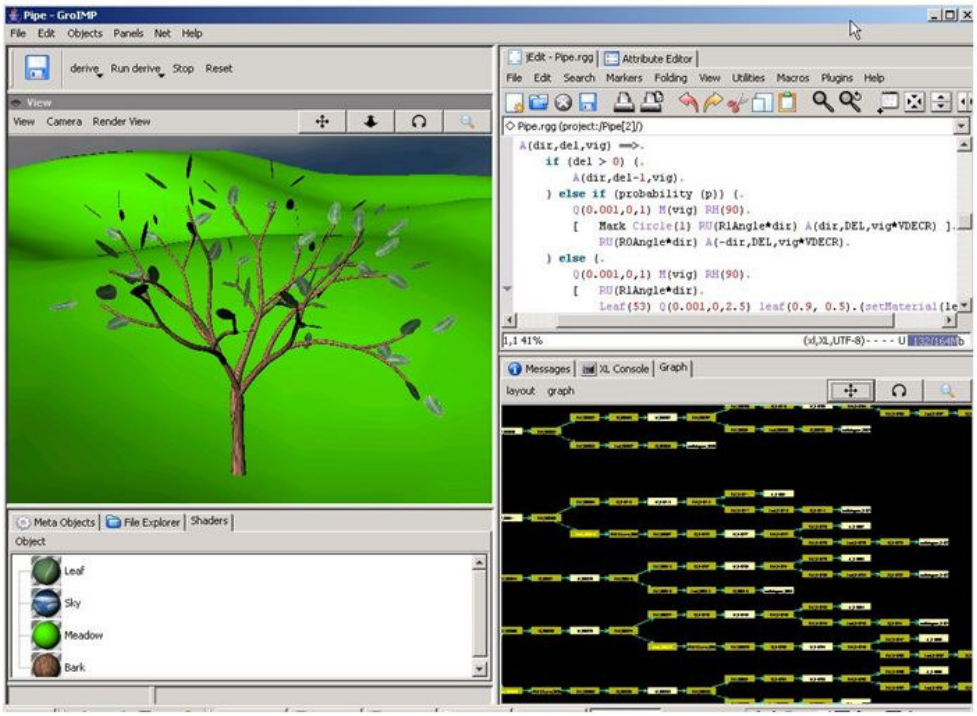
alle Geometrieknoten, die von der Wurzel (Zeichen: ^) des Graphen über genau einen Pfad, der nur aus "successor"- und "branch"-Kanten besteht, erreichbar sind.

Erzwingen, dass ein Objekt auf jeden Fall sichtbar ist:

```
==>> ^ Objekt
```

zum Arbeiten mit XL:

Interaktive 3D-Plattform ***GroIMP*** (Growth-grammar related Interactive Modelling Platform) mit XL-Compiler



- GroIMP ist ein Open Source-Projekt

http://www.grogra.de

http://sourceforge.net/projects/groimp/

siehe auch das Wiki:

http://sourceforge.net/apps/mediawiki/groimp/index.php?title=Main_Page

GroIMP ist eine Kombination von:

- Graph-Grammatiken- (XL-) Interpreter
- Entwicklungsumgebung für XL
- 3D-Modeller
- 3D-Renderer (mehrere Varianten)
- 2D-Graphen-Visualisierer
- Editor für 3D-Objekte und Attribute
- Texturerzeugungswerkzeug
- X3D- (VRML-) Viewer *(in Arbeit)*

Anwendungsbeispiel: Modellierung von Parklandschaften

(Rogge & Moschner 2007, für Stiftung Branitzer Park, Cottbus)

mit GroIMP generierte Erle in VRML-Welt

Erweiterungen des L-System-Formalismus, die bisher nicht unmittelbar in die Sprache XL integriert sind:

Timed L-systems (zeitverstetigte L-Systeme)

Überwindung der Beschränkung auf diskrete Zeit:

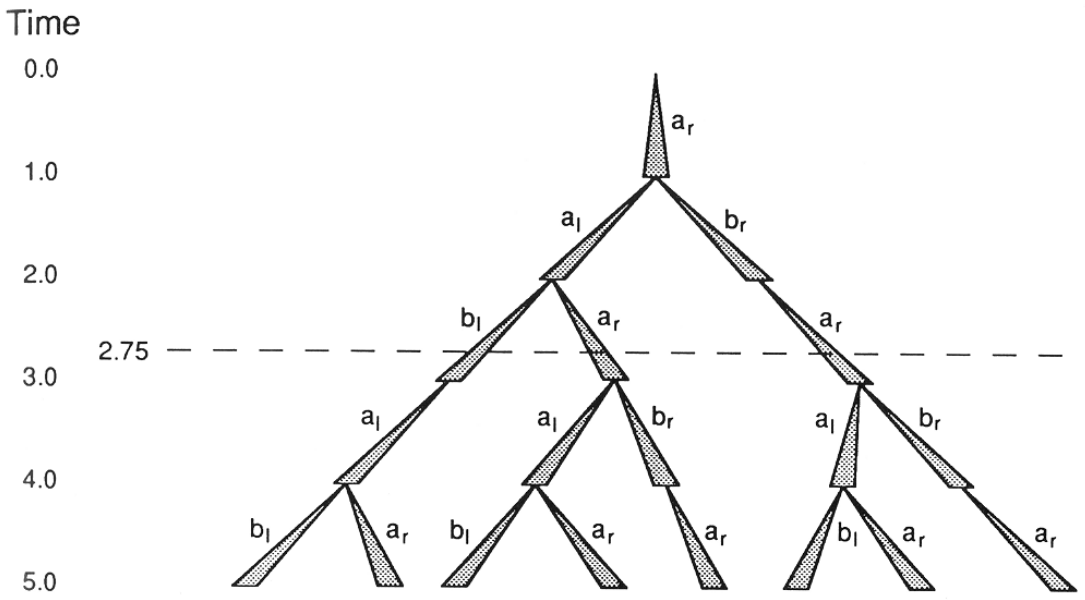
- jedes Symbol erhält als (zusätzlichen) Parameter einen Zeit-Wert
- dieser wird im Verlauf der Simulation stetig "hochgezählt"
- Produktionsregeln werden ausgelöst, wenn das "terminale Alter" eines Symbols erreicht ist

formale Def. siehe Prusinkiewicz & Lindenmayer (1990)

Beispiel:

$$
\begin{array}{lll}
\omega: & (a_r, 0) & \\
p_1: & (a_r, 1) & \rightarrow (a_l, 0)(b_r, 0) \\
p_2: & (a_l, 1) & \rightarrow (b_l, 0)(a_r, 0) \\
p_3: & (b_r, 1) & \rightarrow (a_r, 0) \\
p_4: & (b_l, 1) & \rightarrow (a_l, 0)
\end{array}
$$

(ω = Startsymbol, p_1, ..., p_4 = Produktionsregeln)



(Realisierung in XL möglich mit Rate Assignment Operator und Monitorfunktionen; siehe http://sourceforge.net/apps/mediawiki/groimp/index.php?title=How_to_use_the_ODE_framework)

Überwindung der Beschränkung auf lokal 1-dimensionale Topologien (Verzweigungssysteme):

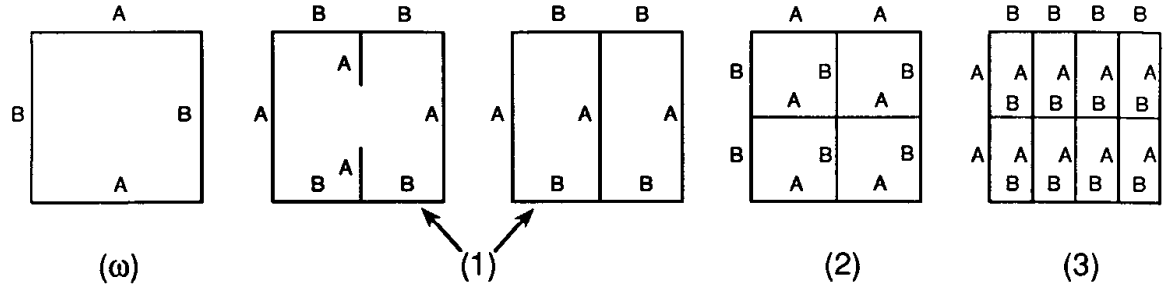
map L-systems, cellwork L-systems

verschiedene Varianten möglich
Beispiel:
Binary propagating map 0L-systems with markers (mBPM0L-systems; Prusinkiewicz & Lindenmayer 1990)

binary: eine Region kann sich in höchstens 2 Tochter-Regionen teilen
propagating: es können keine Kanten gelöscht werden
Marker: legen die Positionen neuer Kanten fest, die eine Region teilen

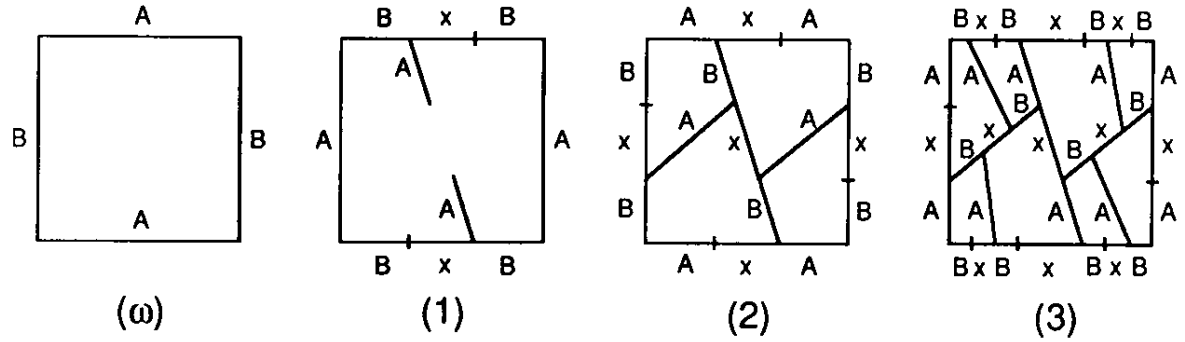
Symbole stehen für markierte, gerichtete Kanten (*edge labels*)
ein Ableitungsschritt besteht aus 2 Phasen: Ersetzung aller Kanten durch Nachfolgekanten und ggf. Marker entsprechend den Produktionsregeln, und Verbinden von Markern, die "matchen", zu neuen Kanten.

Beispiel:
Startwort **ABAB**,
Regeln **A → B [− A] [+ A] B** und **B → A**



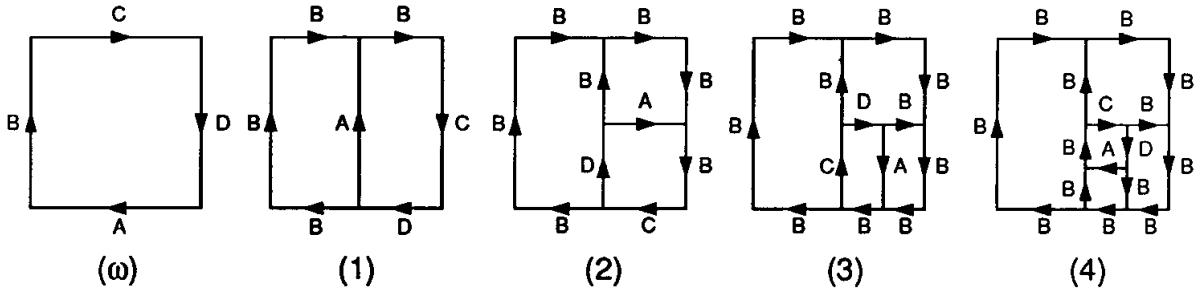
ω : ABAB
p_1 : A → B[-A][+A]B
p_2 : B → A

2. Beispiel:



$$\begin{array}{lcll} \omega & : & \mathtt{ABAB} & \\ p_1 & : & \mathtt{A} & \rightarrow \quad \mathtt{B[-A]x[+A]B} \\ p_2 & : & \mathtt{B} & \rightarrow \quad \mathtt{A} \end{array}$$

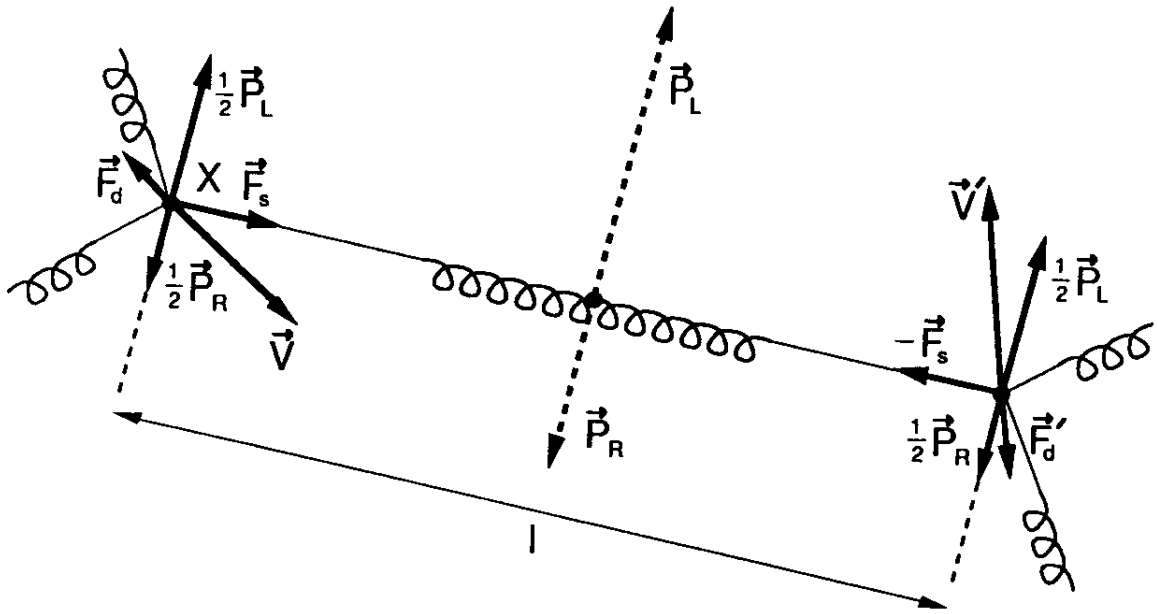
3. Beispiel (mit gerichteten Kanten):



$$\begin{array}{lcll} \omega & : & \vec{\mathtt{A}}\vec{\mathtt{B}}\vec{\mathtt{C}}\vec{\mathtt{D}} & \\ p_1 & : & \vec{\mathtt{A}} & \rightarrow \quad \vec{\mathtt{D}}[-\vec{\mathtt{A}}]\vec{\mathtt{B}} \\ p_2 & : & \vec{\mathtt{B}} & \rightarrow \quad \vec{\mathtt{B}} \\ p_3 & : & \vec{\mathtt{C}} & \rightarrow \quad \vec{\mathtt{B}}[-\overleftarrow{\mathtt{A}}]\vec{\mathtt{B}} \\ p_4 & : & \vec{\mathtt{D}} & \rightarrow \quad \vec{\mathtt{C}} \end{array}$$

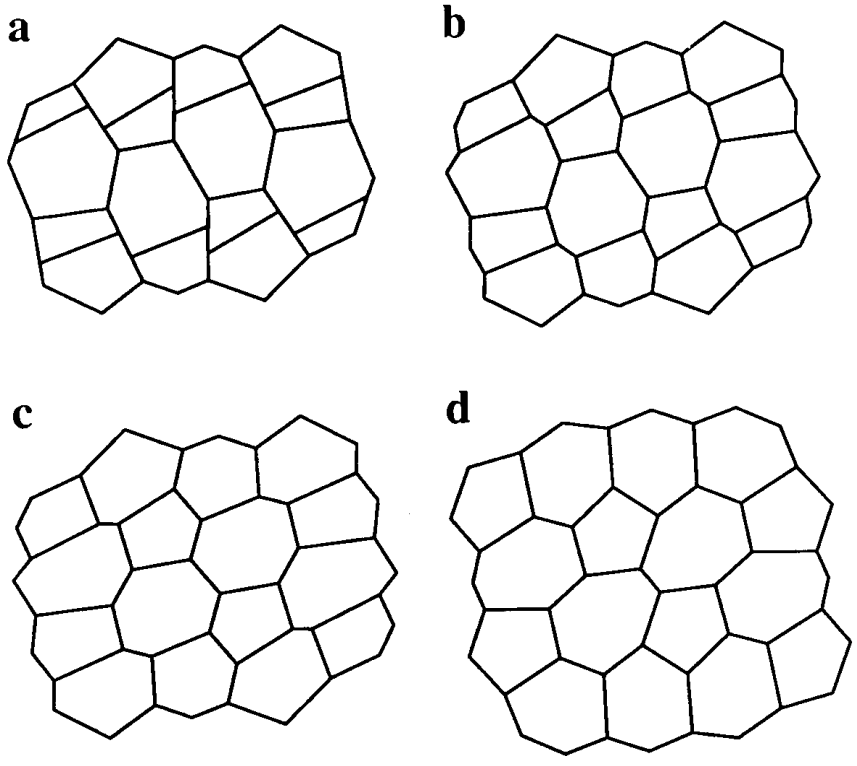
Problem: Geometrie der erzeugten Strukturen ist durch die Regeln nicht festgelegt
(anders als bei gewöhnlichen L-Systemen mit Turtle-Interpretation)

ein möglicher Ansatz: dynamisches Modell, das Kräfte annimmt, die jede Zelle auf ihre Wände (Kanten) ausübt



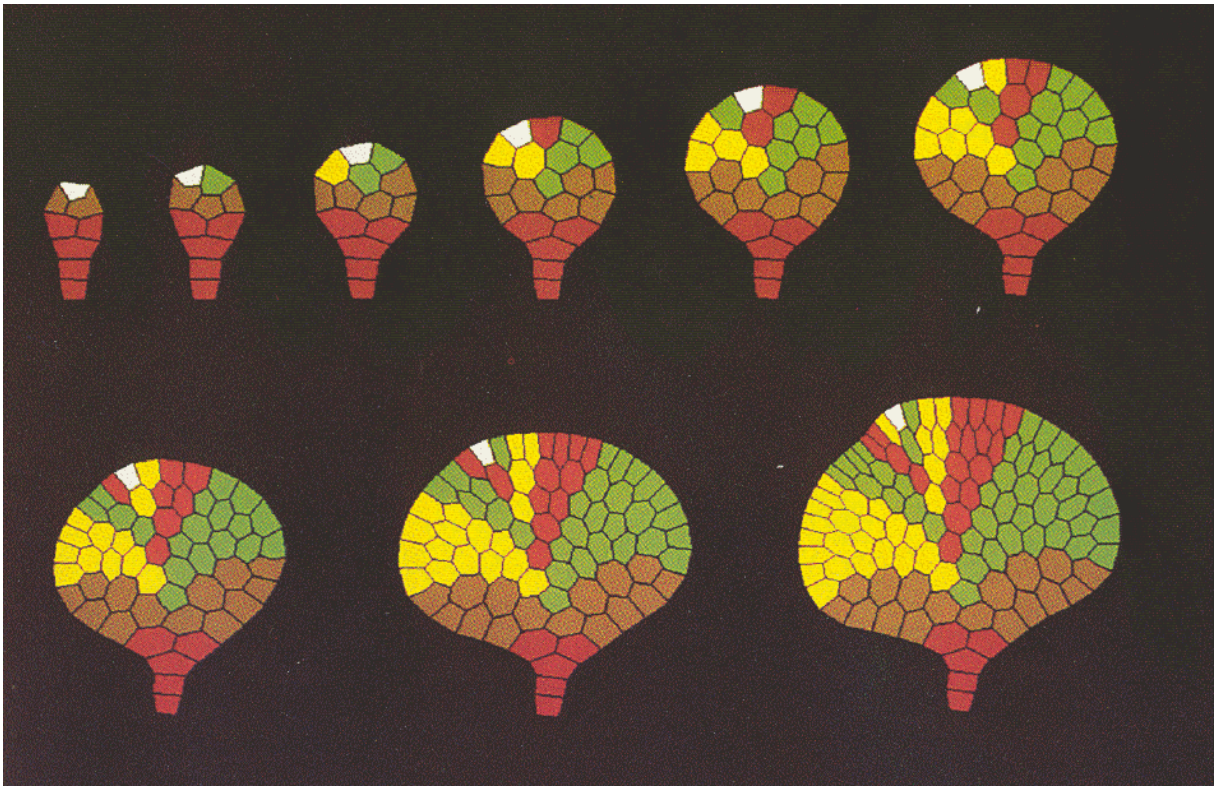
$\Rightarrow$ führt auf System von Differentialgleichungen,
Lösung führt auf Gleichgewichtszustand

Beispiel (Iterationsschritte bei Lösung):

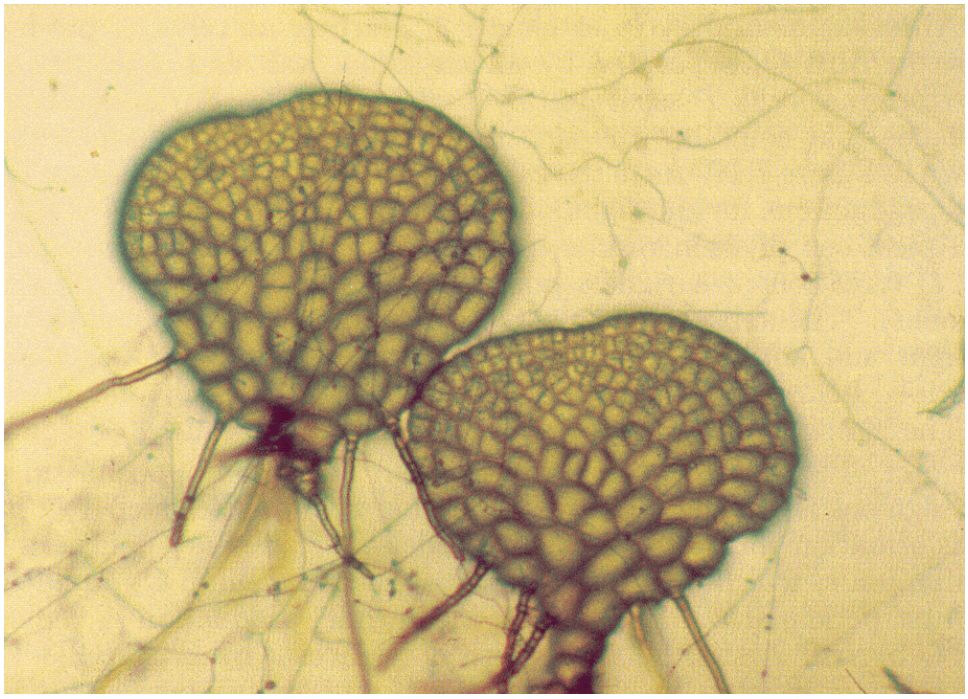


Anwendung: Simulation von Mikroorganismen
hier: *Microsorium linguaeforme* (Farn-Gametophyt; Modell beruht auf botanischen Daten von de Boer; s. Prusinkiewicz & Lindenmayer 1990)

simulierte Entwicklung:

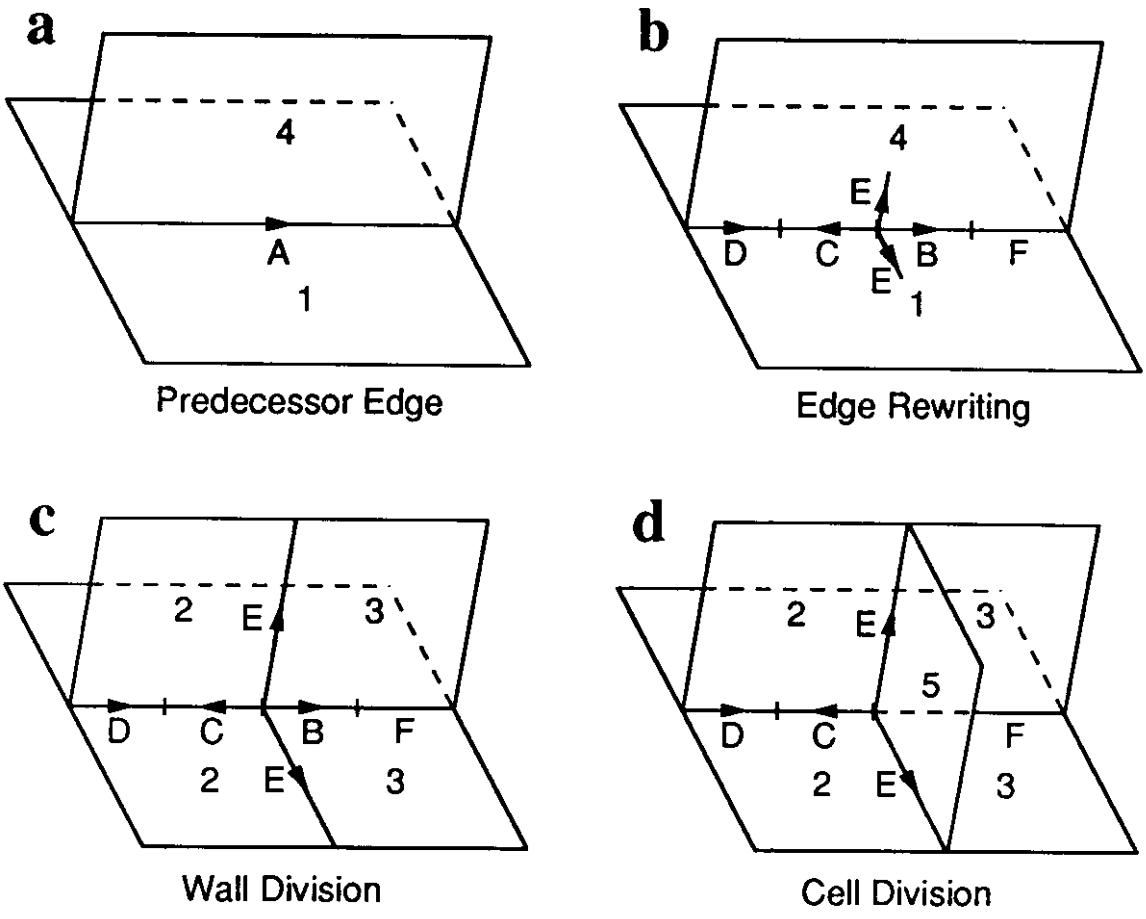


Mikrofotografie der realen Organismen:

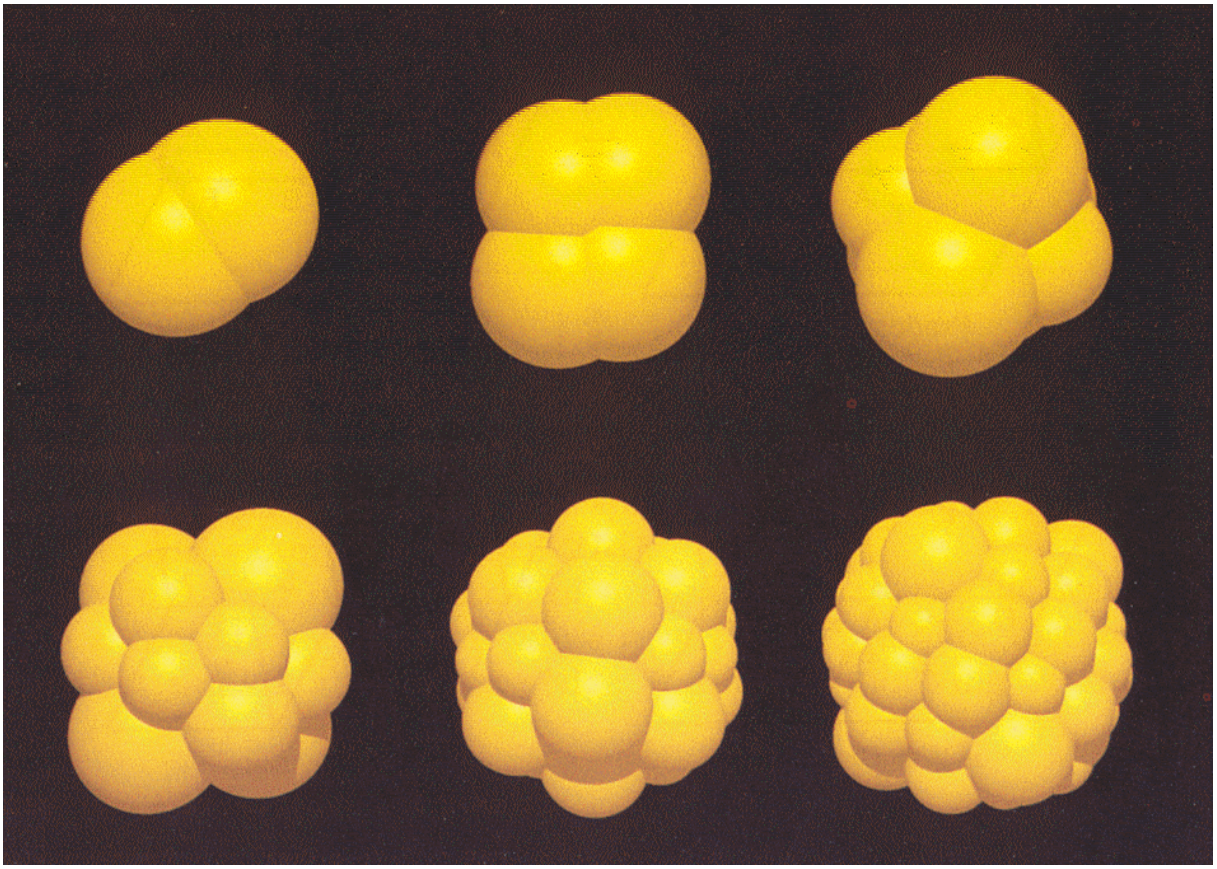


3D-Variante: *Cellwork L-systems*

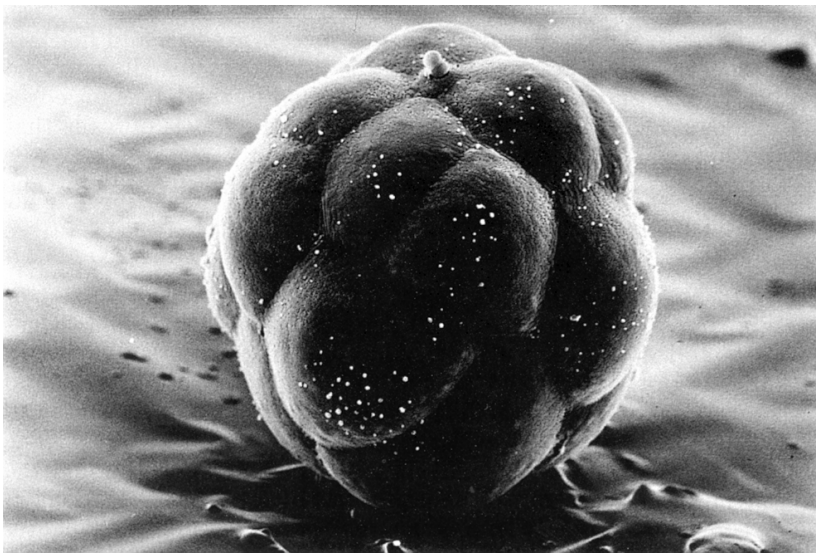
- Produktionsregeln wirken ebenfalls auf die Kanten
- 3 Phasen: Rewriting, Teilung der Zellwände, Teilung der Zellen (jeweils muss *matching* geprüft werden)



Beispiel: Modell des Schnecken-Embryos *Patella vulgata*



zum Vergleich: elektronenmikroskopische Aufnahme des realen Embryos



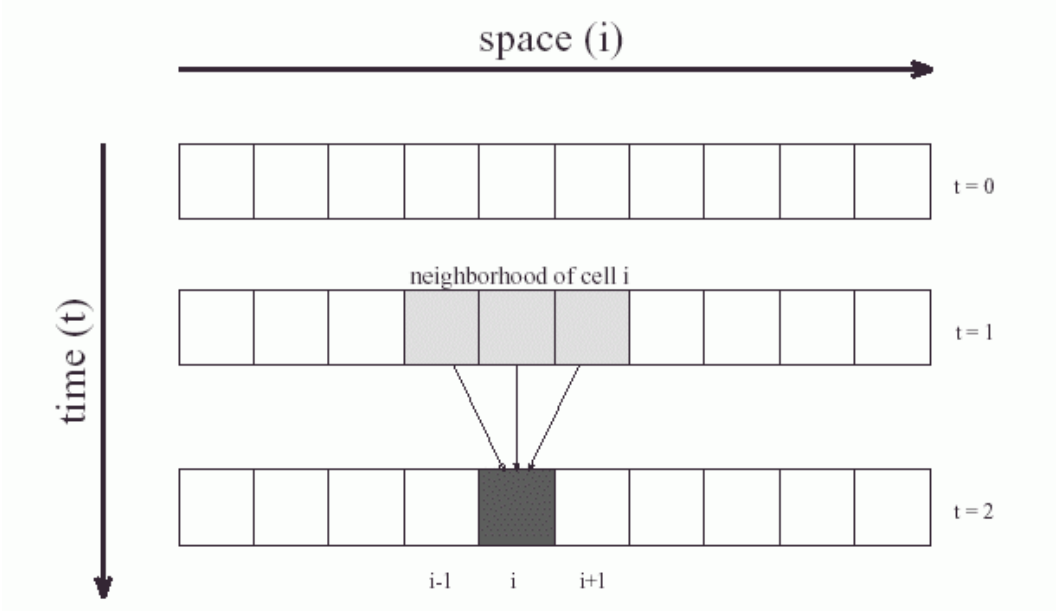
Ausschnitte entnommen aus
Prusinkiewicz & Lindenmayer (1990)
(genaue Quellenangaben siehe http://www.uni-forst.gwdg.de/~wkurth/fs10_lit.htm)

Zelluläre Automaten

(auch: Zellularautomaten, Polyautomaten,
cellular automata, CA)

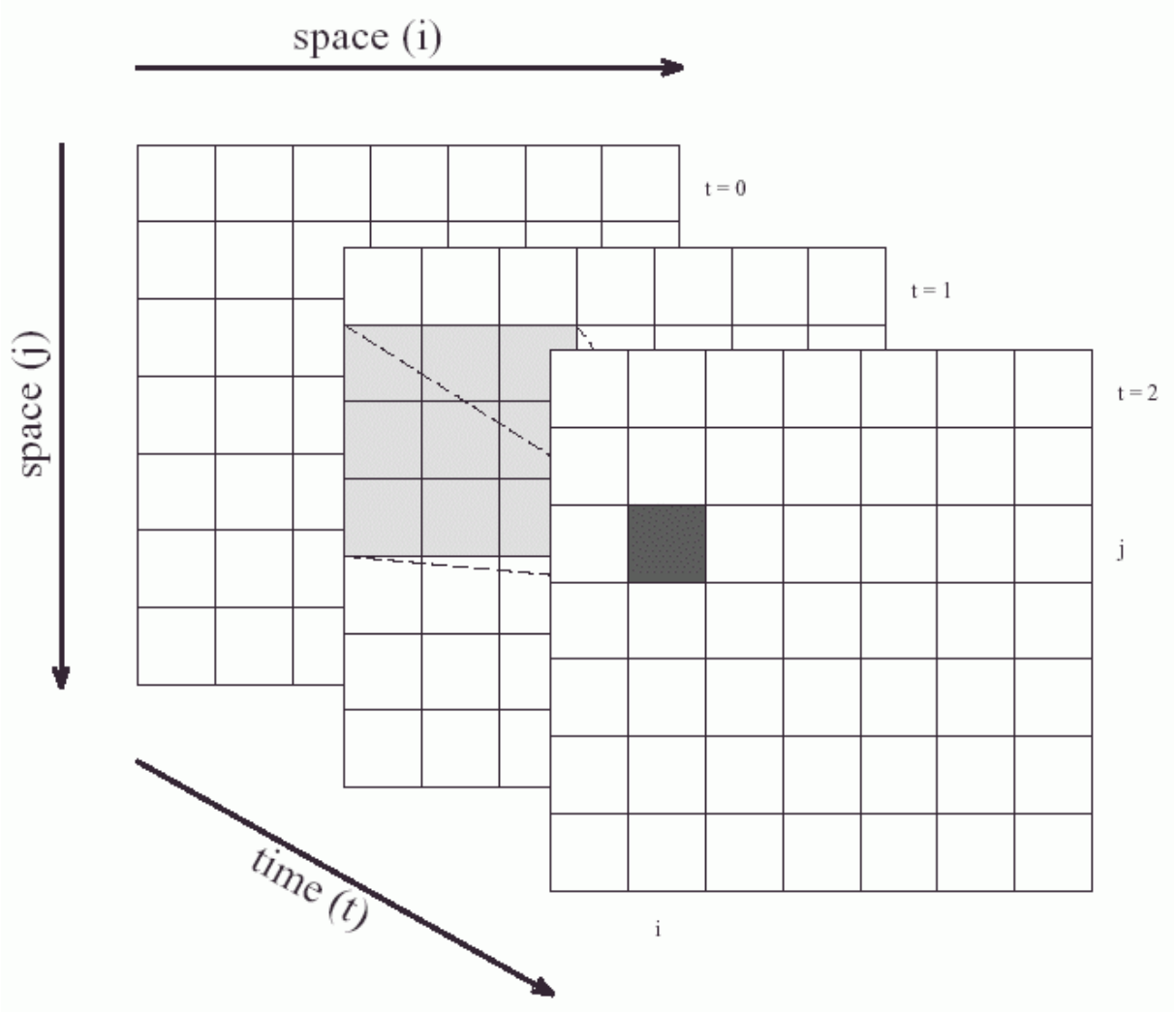
- mathematische Modelle mit diskretem Raum und diskreter Zeit
- die Zeit verläuft in Schritten
- Raum wird als *Gitter* oder *Array von Zellen* repräsentiert
- in den klassischen CA-Modellen kann jede Zelle nur endlich viele Zustände annehmen
- für jede Zelle gilt eine Menge lokaler Regeln, die festlegen, wie sich der neue Zustand dieser Zelle aus ihrem Zustand und dem der Nachbarzellen (im vorherigen Zeitschritt) ergibt.

- Ein zellulärer Automat heißt *homogen*, wenn alle Zellen dieselbe Regelmenge haben. Wir betrachten im Folgenden nur homogene CA.



Je nach Dimensionszahl des zugrundeliegenden Gitters unterscheidet man 1-, 2- und höherdimensionale CA.

Raum und Zeit in einem 2-dimensionalen CA:



Formale Definition von CA:

4-Tupel (L,S,N,f) mit

L regelmäßiges Gitter (Elemente von L: „Zellen“)

S endliche Menge von Zuständen

N endliche Menge von Nachbarschaftsindizes mit

$\forall c \in N, r \in L: r+c \in L$

$f: S^n \rightarrow S$ Übergangsfunktion

Konfiguration $C_t: L \rightarrow S$ weißt jeder Zelle einen Zustand zu,

f ändert C_t zu C_{t+1}: $C_{t+1}(r) = f(\{C_t(i): r \text{ in } N(r)\})$ mit N(r) Nachbarschaft von r.

- Wichtige Eigenschaften:
  - Homogenität
    Jede Zelle hat die selbe Zustandsmenge und Übergangsfunktion.
  - Lokalität
    Zustandsübergänge sind in Zeit und Raum lokal.

Wegen ihrer diskreten Struktur ist die Realisierung von CA auf Rechnern besonders einfach.

Simulation von Zellulären Automaten

- Für jede Generation
  - F = Feld der aktuellen Generation
  - Für jedes Gitterelement F(i,j)
    Wende alle Regeln auf F(i,j) an und speichere Ergebnis in G(i,j)
  - Kopiere G nach F

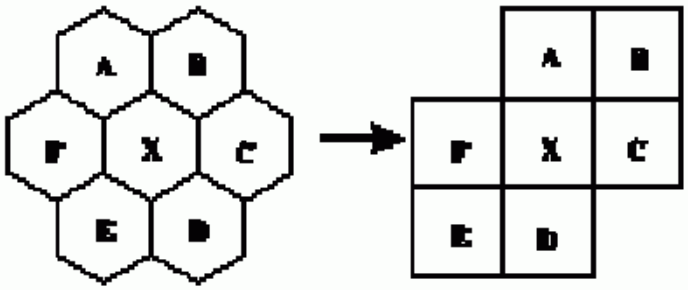
- Verbesserungen:
  - Statt Kopierschritt abwechselnd aktueller ZA in einem der beiden Felder
  - Kompilieren der Regeln in eine „look-up-table“

- CA können regelbasiert in der Sprache XL implementiert werden

- mit Parallelrechnern hochgradige Parallelisierung der Berechnung möglich.

Gitter-Geometrie

- Reguläres Gitter
- Ein-dimensionaler Automat: Lineares Feld
  Beispiele: Verkehrssimulation, „Color-Eater“
- Zwei-dimensionale Automaten:
  - Dreieckige Zellen: wenige direkte Nachbarn,
  - Quadratische Zellen: einfache Repräsentation, einfache Visualisierung, manchmal nicht ausreichend isotrop.
  - Hexagonale Zellen: geringste Anisotropie
- Dreieckige und Hexagonale Automaten müssen auf ein quadratisches Gitter abgebildet werden:
  - Scheren-Abbildung: $shear(i,j) = (i + floor(j/2), j)$

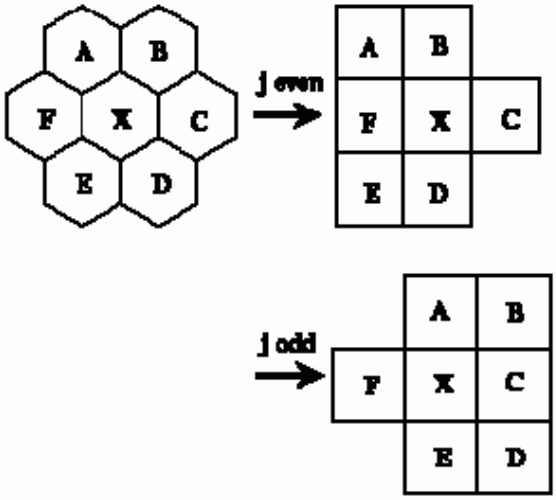


Nachbarschaft wird einheitlicher erhalten, aber Bedingungen für Grenzen schwieriger berechenbar.
Für Visualisierung Rücktransformation notwendig

Literatur:

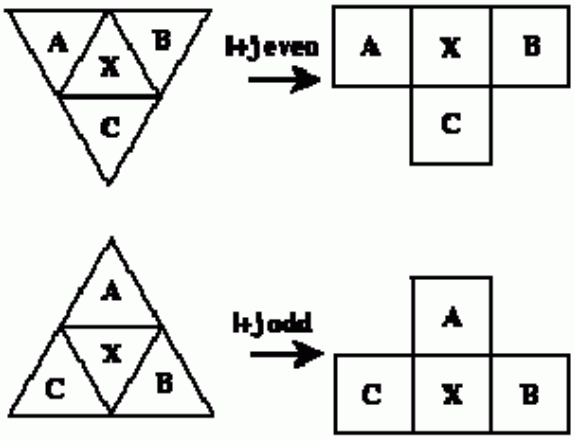
J. Weimar (1998): Simulation with Cellular Automata,
http://www.tu-bs.de/institute/WiR/weimar/Zaskript/

– Shift-Abbildung $shift(i,j) = (i,j)$

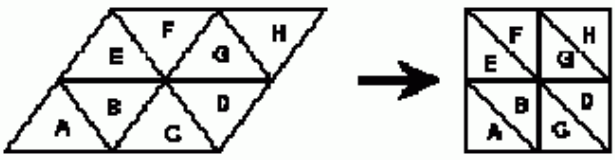


Grenzbedingungen einfach zu implementieren, Visualisierung einfach, Nachbarschaft abhängig von Parität des Index ➜ inhomogen
Fehlende Indizes hinzunehmen und auf der Basis eines invarianten Zustandsteils die sechs relevanten Nachbarn ausfiltern.

– Abbildungen für Dreieckszellen:



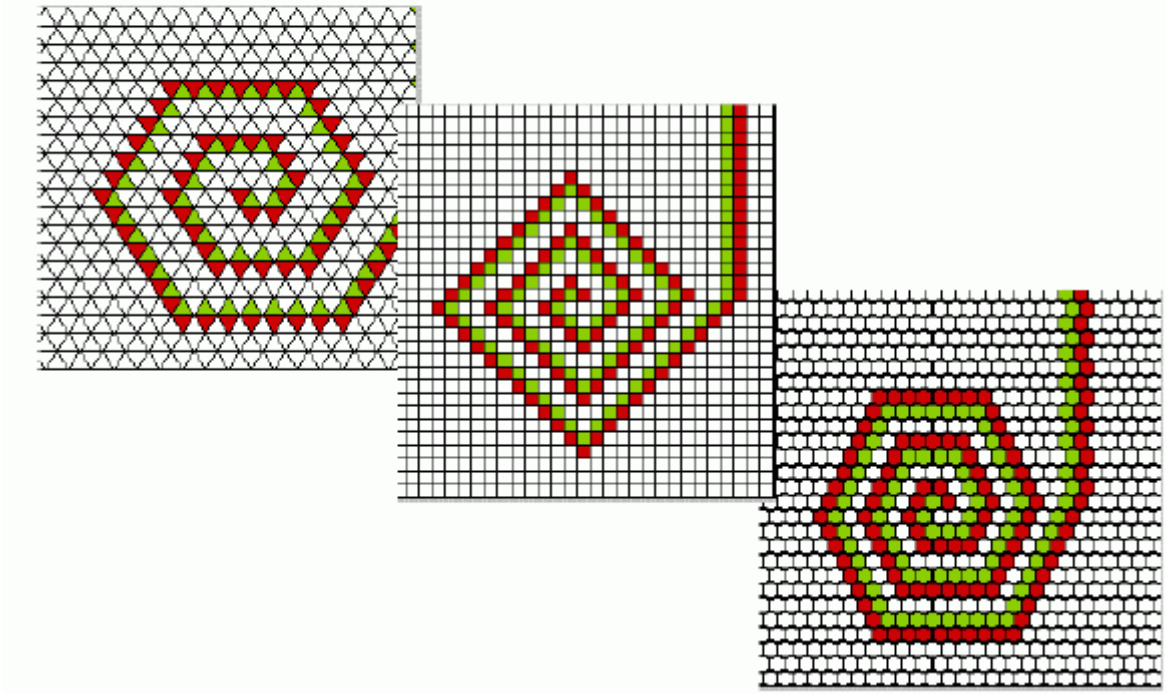
Jede Zeile von Dreiecken wird auf eine Zeile von Quadrate abgebildet ➜ Nicht einheitliche Nachbarschaften



Erweiterung des Zustands einer Zelle notwendig (SxS), ➜ größere Zustandsmenge einheitliche Transformation der Nachbarschaften,

Gitter

Topologie 2D



Das Problem der Gitter-Ränder:

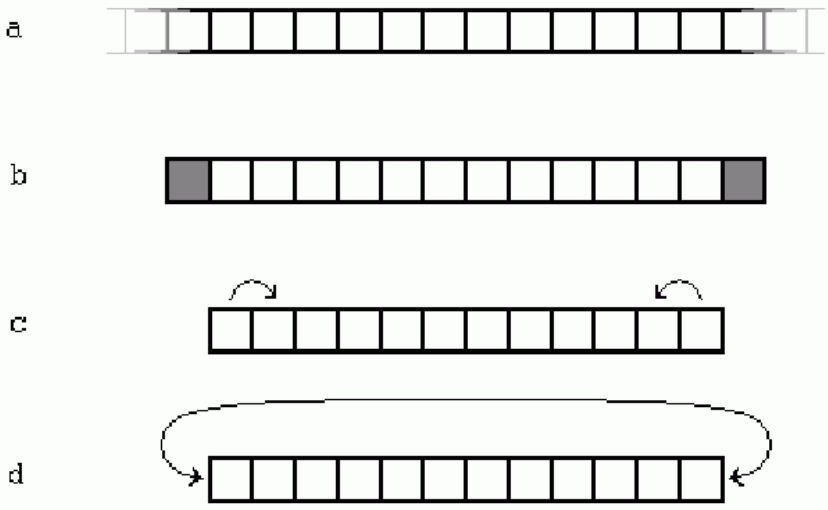
- Simulation eines wirklich unendlichen Gitters unmöglich (simulationstechnisch und möglicherweise durch reales System bedingt - natürliche Grenzen)
- Periodische Grenzen

Bei zwei-dimensionalen Automaten - „Torus“

- Reflektive Grenzen

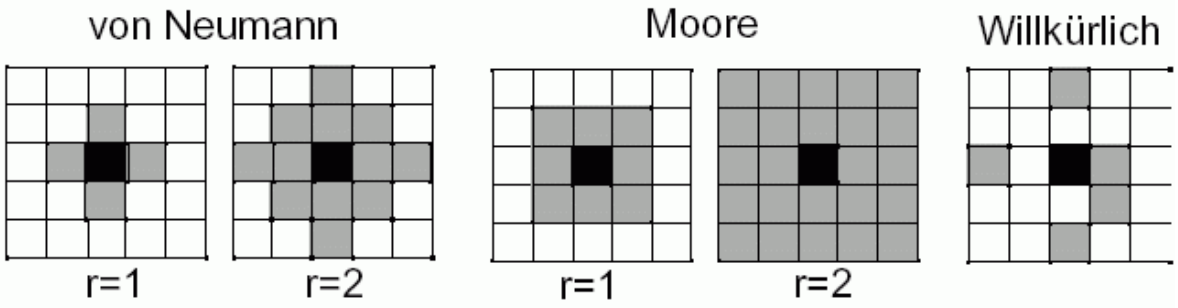
Wenn das reale System Grenzen hat, aber der Wert an der Grenze nicht vorgegeben ist.

- Kombinationen möglich, z.B. um einen langen Tunnel zu simulieren sind periodische Grenzen in der Horizontalen, reflektive Grenzen in der Vertikalen sinnvoll.
- Fixe Grenzen:
  Zellen an der Grenze besitzen fixen Wert.



a: unendliches Gitter, b: fixe Grenzen, c: reflektive Grenzen,
d: periodische Grenzen

Nachbarschaften



Gegeben: Zelle i,j

- Von Neumann-Nachbarschaft
  $N_{i,j} = \{(k,l) \in L : |k-i| + |l-j| \leq r\}$
- Moore-Nachbarschaft
  $N_{i,j} = \{(k,l) \in L : |k-i| \leq r \wedge |l-j| \leq r\}$
- Grundsätzlich kann jede Art von Nachbarschaftsmenge gebildet werden, solange sie für alle Zellen gleich und endlich ist.
- Große Nachbarschaften ➔ bessere Isotropie, beim Simulieren ineffizient.

Zustände

- jede Zelle in G befindet sich in einem definierten Zustand, der Element der endlichen Menge von Elementarzuständen ist, z.B. E={ein,aus}

Lokale Funktion / Regeln

- definiert für jede Zelle und alle möglichen Umgebungen den nächsten Zustand, z.B. als Regel
- Regeln heißen **totalistisch**, falls sie nur summarisch von den Nachbarzuständen abhängen
- Regeln sind i.Allg. nicht umkehrbar - eine Konfiguration kann verschiedene Vorgänger haben
- **probabilistische** Regeln erlauben, im Gegensatz zu **deterministischen** Regeln eine stochastische Auswahl mehrerer Folgezustände für die gleiche Umgebung

Regelanwendung

- **synchrone** Automaten wenden f zu jedem Zeitschritt auf alle Zellen gleichzeitig an
- **asynchrone** Automaten wenden f sequentiell auf alle Zellen an:
  - in fester, deterministischer Folge
  - in stochastischer Folge
  - in dynamisch gesteuerter, deterministischer Folge
  - ...

im Folgenden werden nur synchrone CA betrachtet.

Beispiel eines zweidimensionalen zellulären Automaten:

John H. Conway (späte 60er Jahre): "*Game of Life*"

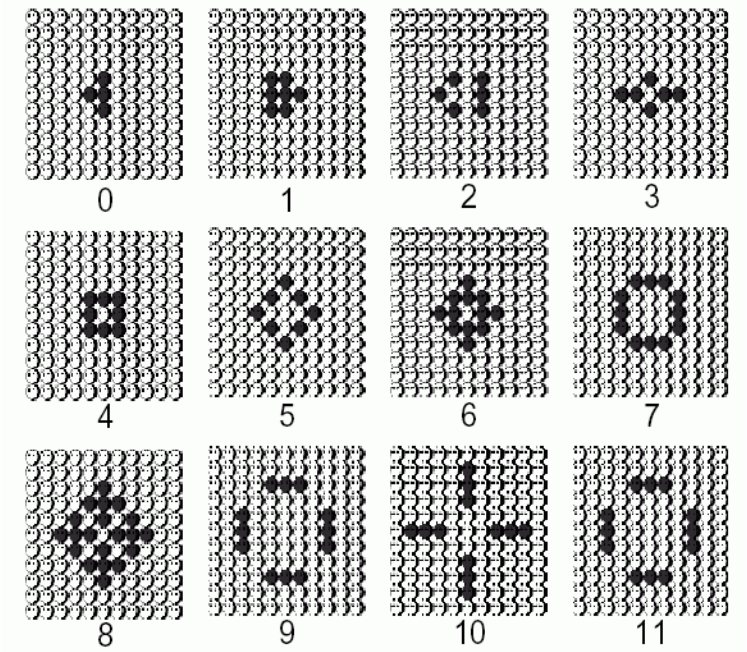
1970 von Martin Gardner in seiner mathematischen Kolumne in "Scientific American" vorgestellt, löste Welle weltweiter Forschung an diesem CA aus.

- regelmäßiges Quadratgitter
- nur 2 Zustände ("tot" und "lebendig", bzw. "0" und "1")
- Moore-Nachbarschaft mit Radius 1, d.h. es werden 8 Nachbarzellen betrachtet
- Regeln:

Life-Regeln

- Eine Zelle ist im nächsten Takt „lebendig“, wenn genau drei ihrer Nachbarn lebendig sind
- Eine lebendige Zelle ist im nächsten Takt „tot“, wenn weniger als 2 oder mehr als 4 ihrer Nachbarn lebendig sind.

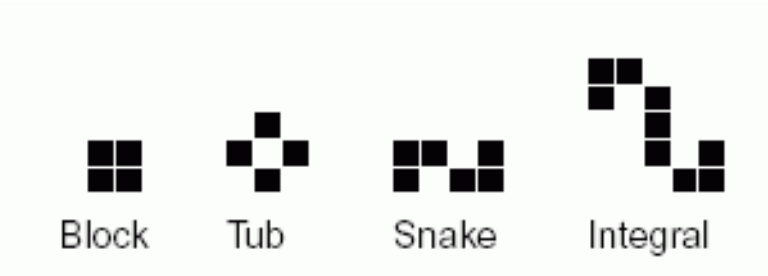
Beispiel-Entwicklung:



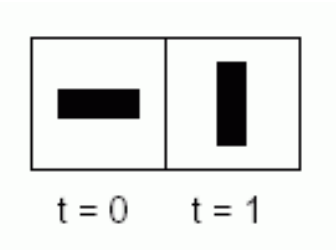
Muster lassen sich durch ihr Verhalten (bei der weiteren Entwicklung) klassifizieren.

Beispiele:

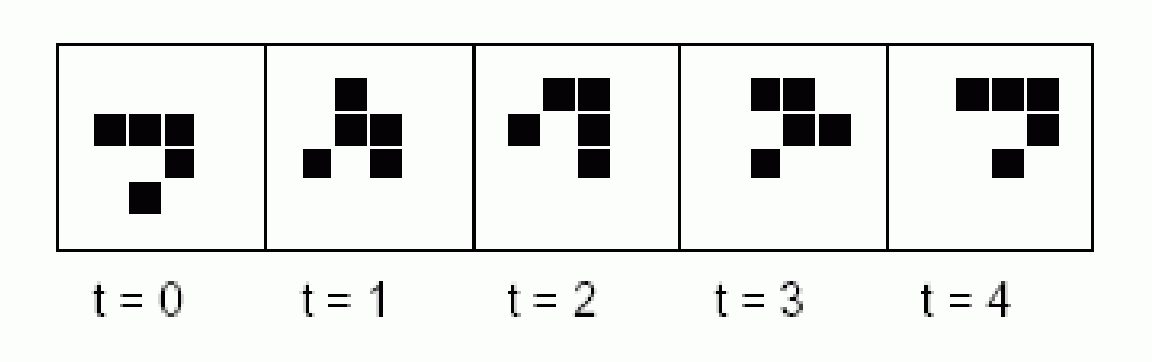
Typ I: "Stillleben", Muster, die sich nicht ändern (Fixpunkte der Übergangsfunktion):



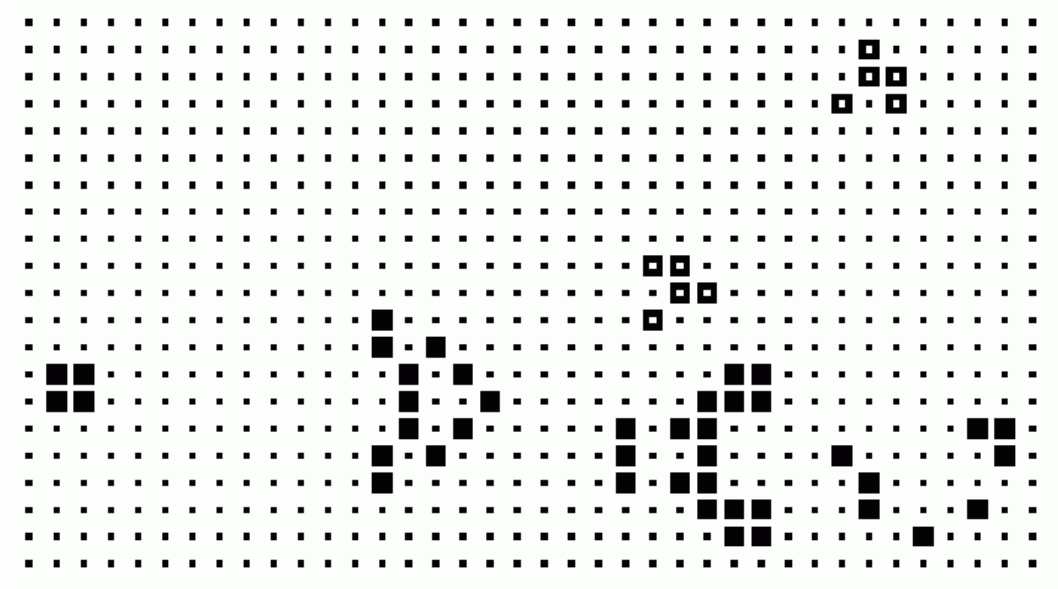
Typ II: Oszillatoren (periodische Muster): z.B. der "Blinker"



Typ III: "Raumschiffe" = Muster, die nach einer endlichen Zahl von Schritten wiederkehren, aber räumlich verschoben. Muster, die sich mit konstanter Geschwindigkeit über das Gitter bewegen. Beispiel: der "Gleiter"



Typ IVa: "Kanonen" = Oszillatoren, die in jedem Zyklus Raumschiffe emittieren.
Gleiterkanone:



Typ IVb: "Dampfer" = Raumschiffe, die Stillleben, Oszillatoren und / oder Raumschiffe hinter sich lassen.

Typ IVc: "Brüter" = Muster, die ihre Populationsgröße quadratisch (oder noch schneller) vergrößern (z.B. ein Raumschiff, das Gleiterkanonen produziert).

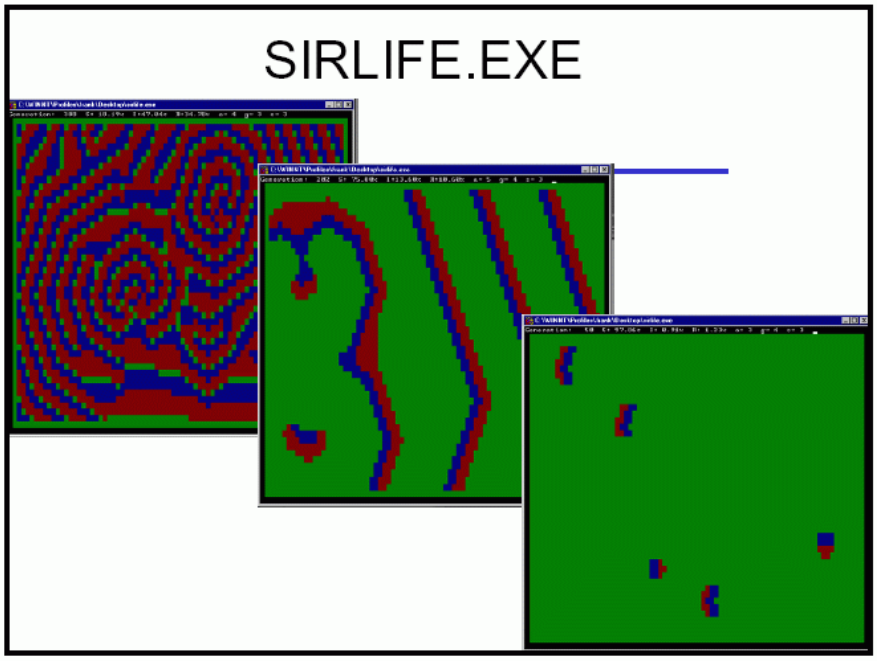
Typ V: unstabile Muster. Muster, die sich durch eine Folge von Zuständen weiterentwickeln und nie zum Ausgangszustand zurückkehren. Kleine Muster mit "langer" Entwicklungsphase vor der Stabilisierung heißen "Methusalems".

The Game of Life
Beobachtungen

- für eine gegebene Anfangskonfiguration kann i.Allg. das zukünftige Verhalten nicht ohne Simulation vorhergesagt werden.
- einige Startzustände können nicht im Laufe der Simulation entstehen. Diese Zustände heißen *Garten-Eden* Konfigurationen.
- das Spiel ist (berechnungs-)universell. Der Beweis wird über die Konstruktion und Interaktion logischer Gatter geführt.

Praktischer Einsatz zweidimensionaler CA:
in der Simulation natürlicher Phänomene, z.B. der Ausbreitung von Epidemien

Beispiel:



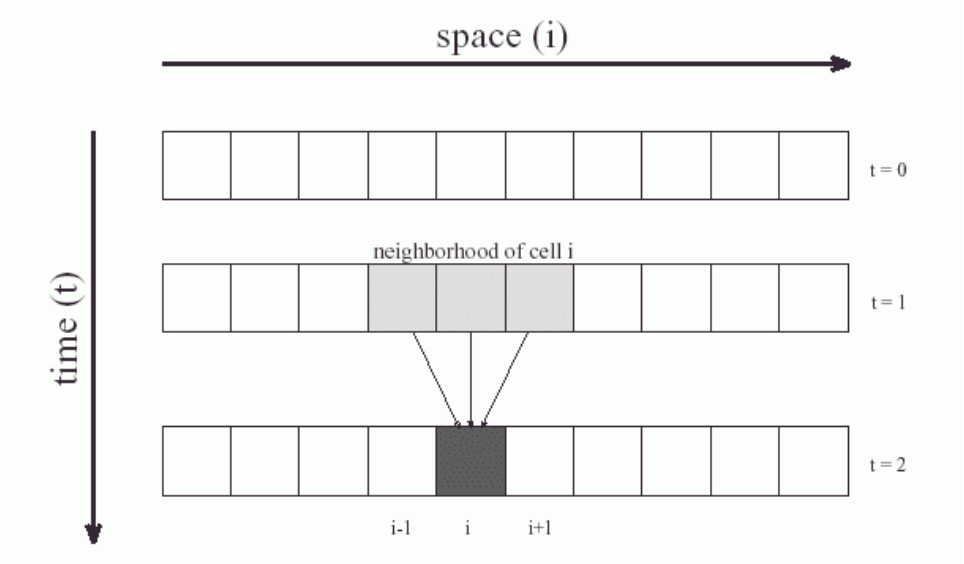
(Universität Leipzig)

1-dimensionale zelluläre Automaten

zugrunde liegt ein (potenziell endloses) Band mit Gitterzellen

jede Zelle nimmt einen von *k* möglichen Zuständen an (einfachster Fall: $k = 2$)

Nachbarschaft mit Radius *r*: betrachte *r* nächste Nachbarn auf jeder Seite (einfachster Fall: $r = 1$, d.h. nur 2 Nachbarn und die Zelle selbst werden betrachtet)



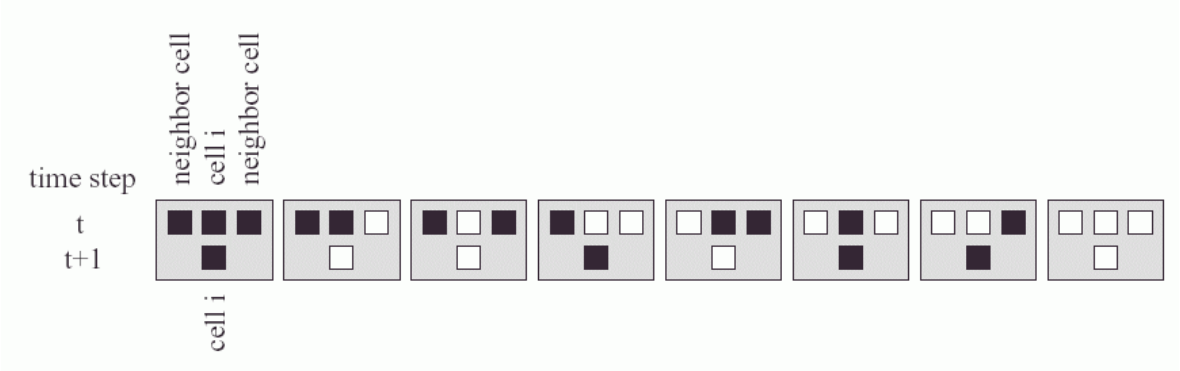
Beschreibung der Regel durch eine Funktionstabelle (look up table, *rule table*, manchmal auch der *Genotyp* genannt):

z.B.

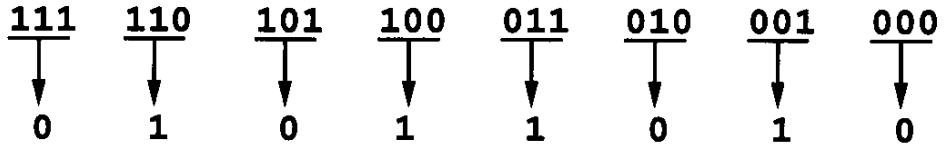
$a_{i-1}(t)$	$a_i(t)$	$a_{i+1}(t)$	$a_i(t+1)$
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

(*)

grafische Darstellung dieser Regel:



Codierung einer Regel als Dezimalzahl (gewonnen aus dem 01-String der Funktionswerte, der die Wirkung eindeutig festlegt):



01011010 = 0x128 + 1x64 + 0x32 + 1x16 + 1x8 +0x4 + 1x2 + 0x1 = 90

Bei *totalistischen* Regeln hängt das Ergebnis der Funktion nur von der Summe der Werte in der Nachbarschaft ab.
Hier wird eine andere Form der Codierung gewählt:

z.B. für einen totalistischen CA mit $r = 2$ und $k = 2$ (unten sind Beispiel-Zustandsübergänge aufgezählt, die von dieser Regel induziert werden):

$$\begin{array}{cccccccc} \Sigma=5 & \Sigma=4 & \Sigma=3 & \Sigma=2 & \Sigma=1 & \Sigma=0 & & \\ 0 & 1 & 0 & 1 & 0 & 0 & = & 20 \end{array}$$

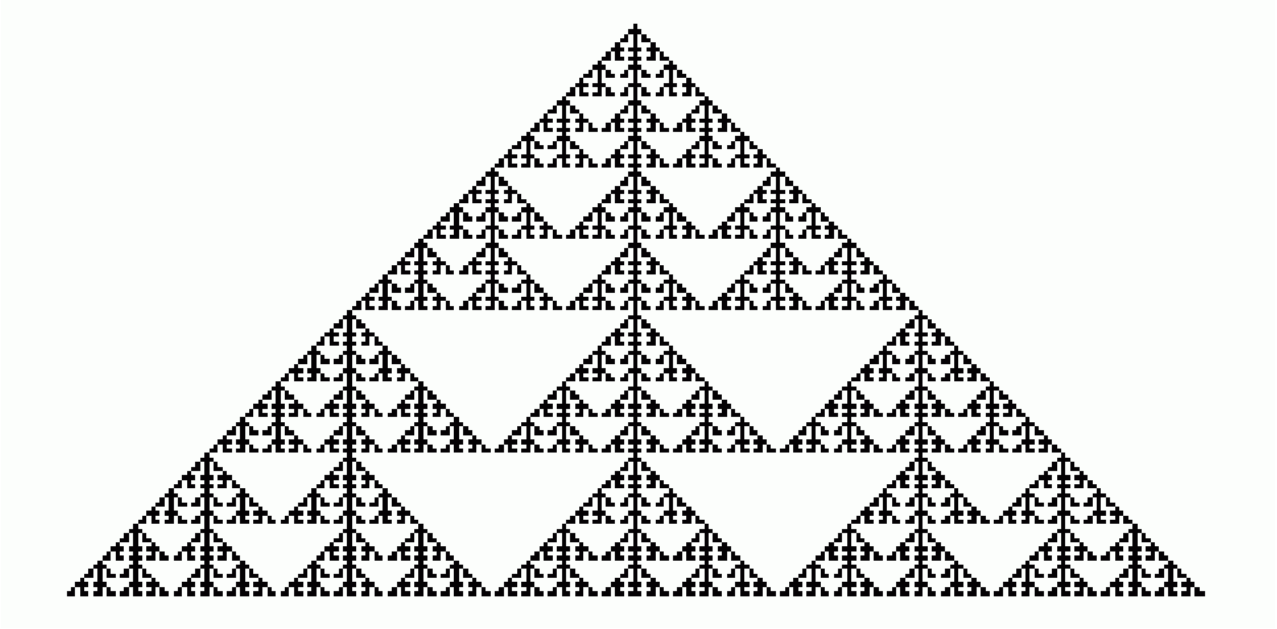
$$\begin{array}{lcll} \Sigma=1 & : & 01\,\mathbf{0}\,00 & \longrightarrow\ \mathbf{0} \\ \Sigma=2 & : & 10\,\mathbf{0}\,01 & \longrightarrow\ \mathbf{1} \\ \Sigma=3 & : & 01\,\mathbf{1}\,01 & \longrightarrow\ \mathbf{0} \\ \Sigma=4 & : & 11\,\mathbf{1}\,01 & \longrightarrow\ \mathbf{1} \\ \Sigma=5 & : & 11\,\mathbf{1}\,11 & \longrightarrow\ \mathbf{0} \end{array}$$

weitere, häufig verwendete Restriktionen für die Regeln:

- es gibt einen speziellen Zustand 0 ("quiescent state"), und wenn alle Zellen der Nachbarschaft (und die aktuelle Zelle) den Zustand 0 haben, ist das Ergebnis wieder 0

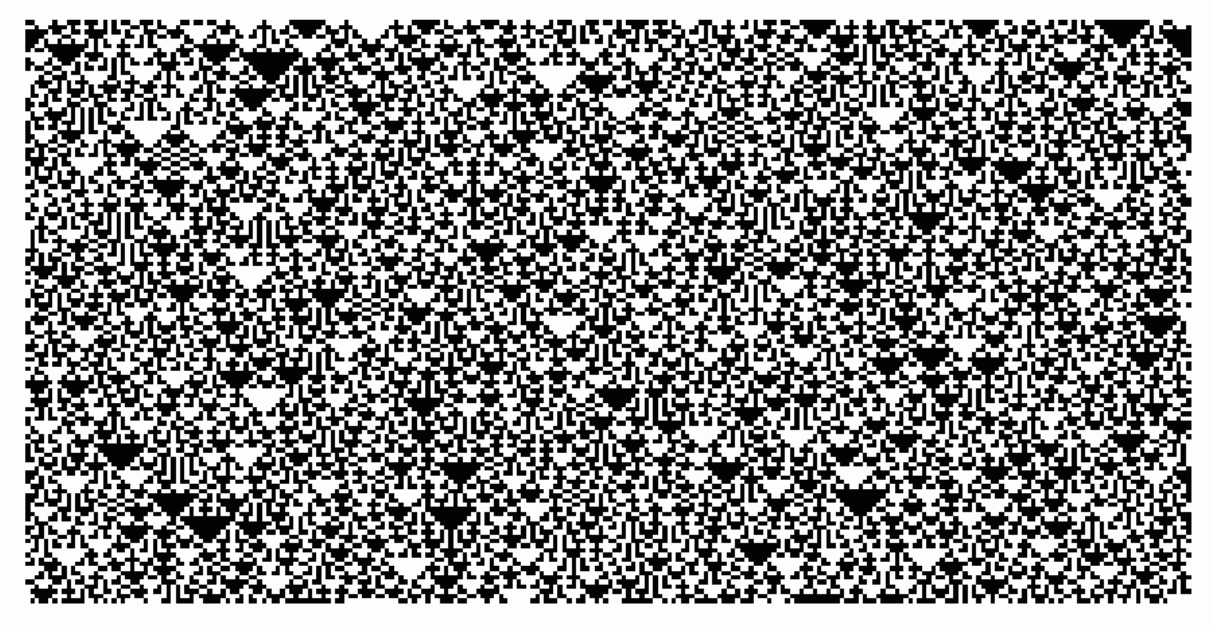
- Symmetrie der Regeln (z.B. 100 und 001 müssen denselben Zustand liefern)

Anwendung des CA mit der obigen Regelmenge (*) auf eine Zeile mit einer einzigen 1, sonst lauter Nullen:



(senkrechte Achse = Zeitachse!)

ein anderes Muster entsteht, wenn man (bei gleicher Regelmenge!) von einer Anfangszeile mit *zufälliger* Verteilung von Nullen und Einsen ausgeht:



typisch: "Dreiecke" als kleine, geordnete Strukturen im Chaos.

Resultierendes Muster als Ergebnis der Anwendung einer CA-Regelmenge: "*Phänotyp*".

Wolfram's Klassifikation

Stephen Wolfram ordnete die 1-dim. CA-Regeln in 4 Klassen ein, je nach der Dynamik, die sich entwickelt bei "typischen" (zufälligen) Ausgangszeilen (d.h. nach dem typischen Phänotyp):

Klasse I:
Es entsteht ein räumlich homogener Zustand. Muster verschwinden. "Fixpunkt"-Zustand.

Beispiel:

(totalistische Regel mit *k*=2, *r*=2 und Code 60)

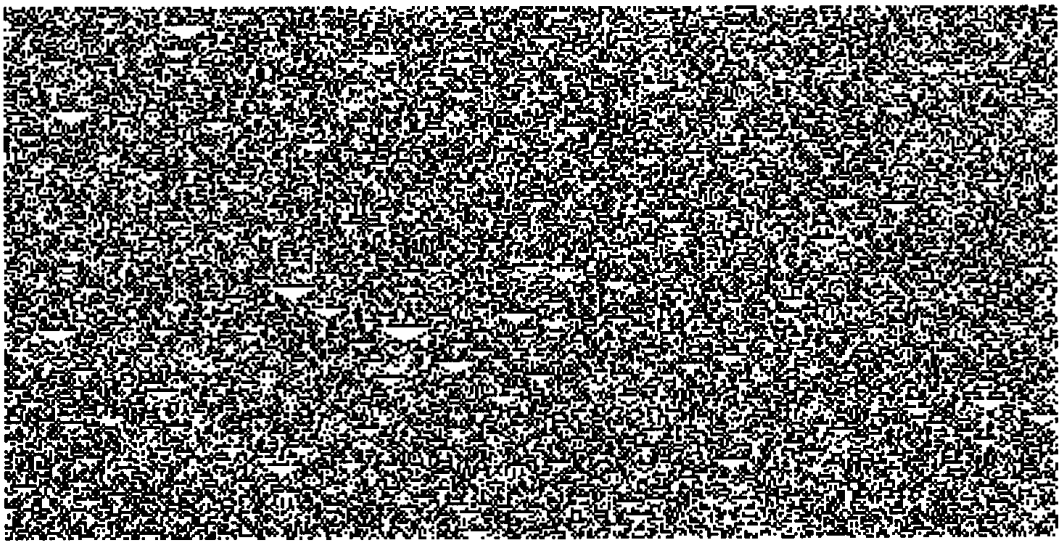
Klasse II:
liefert Folgen von einfachen stabilen oder periodischen Strukturen (endlose Zyklen derselben Zustände).

Beispiel:

(totalistische Regel mit *k*=2, *r*=2 und Code 56)

Klasse III:
zeigt chaotisches, aperiodisches Verhalten. Muster wachsen unbegrenzt mit fester Wachstumsrate.

Beispiel:



(totalistische Regel mit *k*=2, *r*=2 und Code 10)

Klasse IV:
liefert komplizierte lokale Strukturen, von denen sich einige fortbewegen können. Muster wachsen und schrumpfen mit der Zeit. Die "interessanteste" Klasse, möglicherweise mit universeller Berechnungskapazität, wie beim Game of Life.

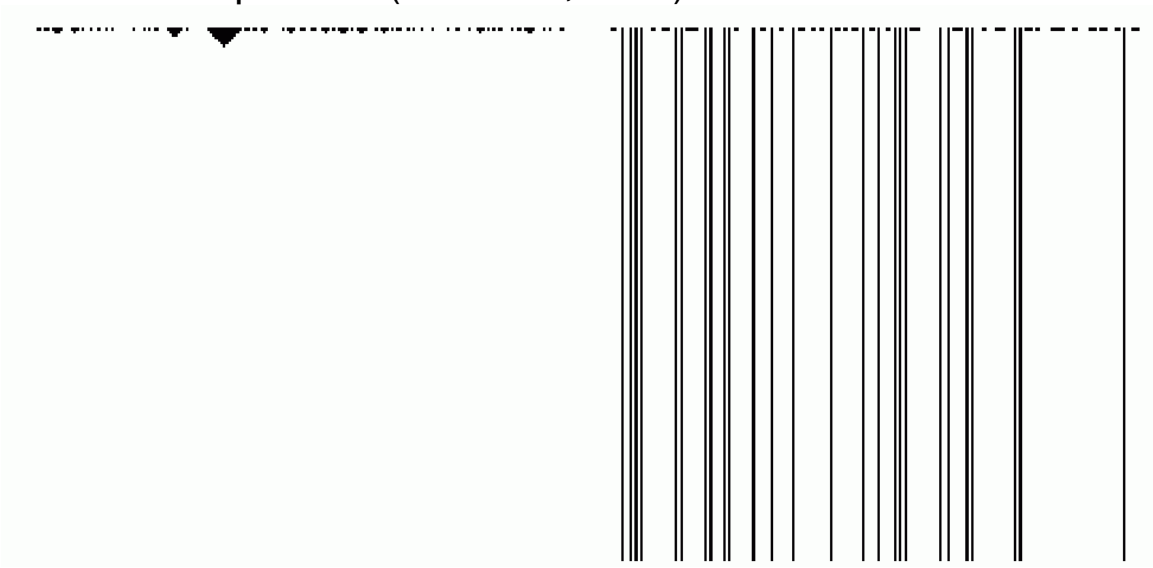
Beispiel:

(totalistische Regel mit *k*=2, *r*=2 und Code 20)

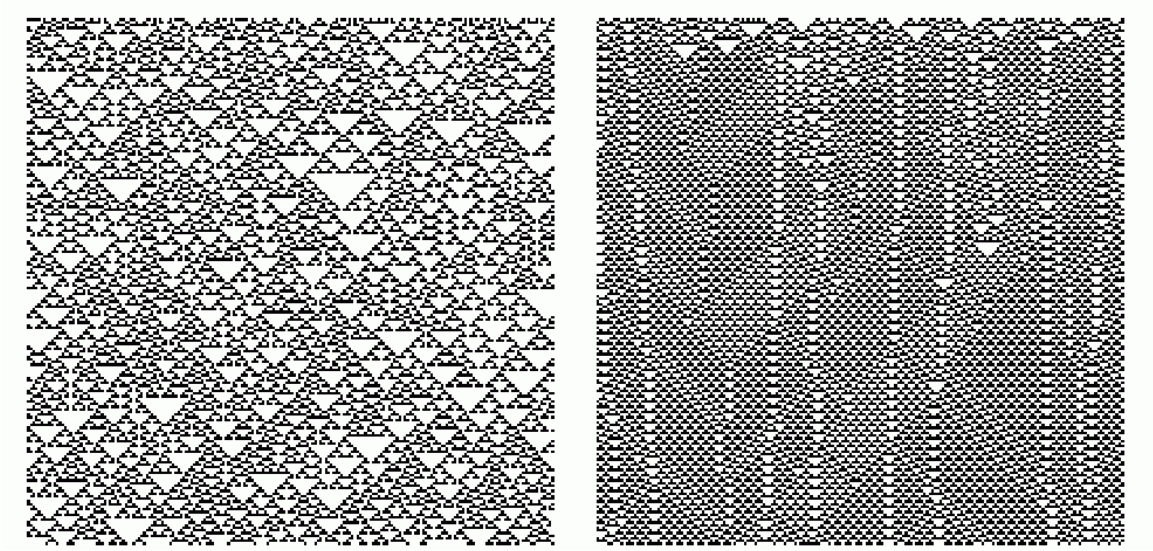
Die 4 Klassen unterscheiden sich auch durch die Wirkung, die kleine Änderungen in den Anfangsbedingungen haben (Wolfram 1984):

Klasse I: keine Änderung im Endzustand.
Klasse II: Änderungen nur in einer Region endlicher Größe.
Klasse III: Änderungen in einer Region mit ständig wachsender Größe.
Klasse IV: Irreguläre Änderungen.

weitere Beispiel-CA (mit $k = 2$, $r = 1$):

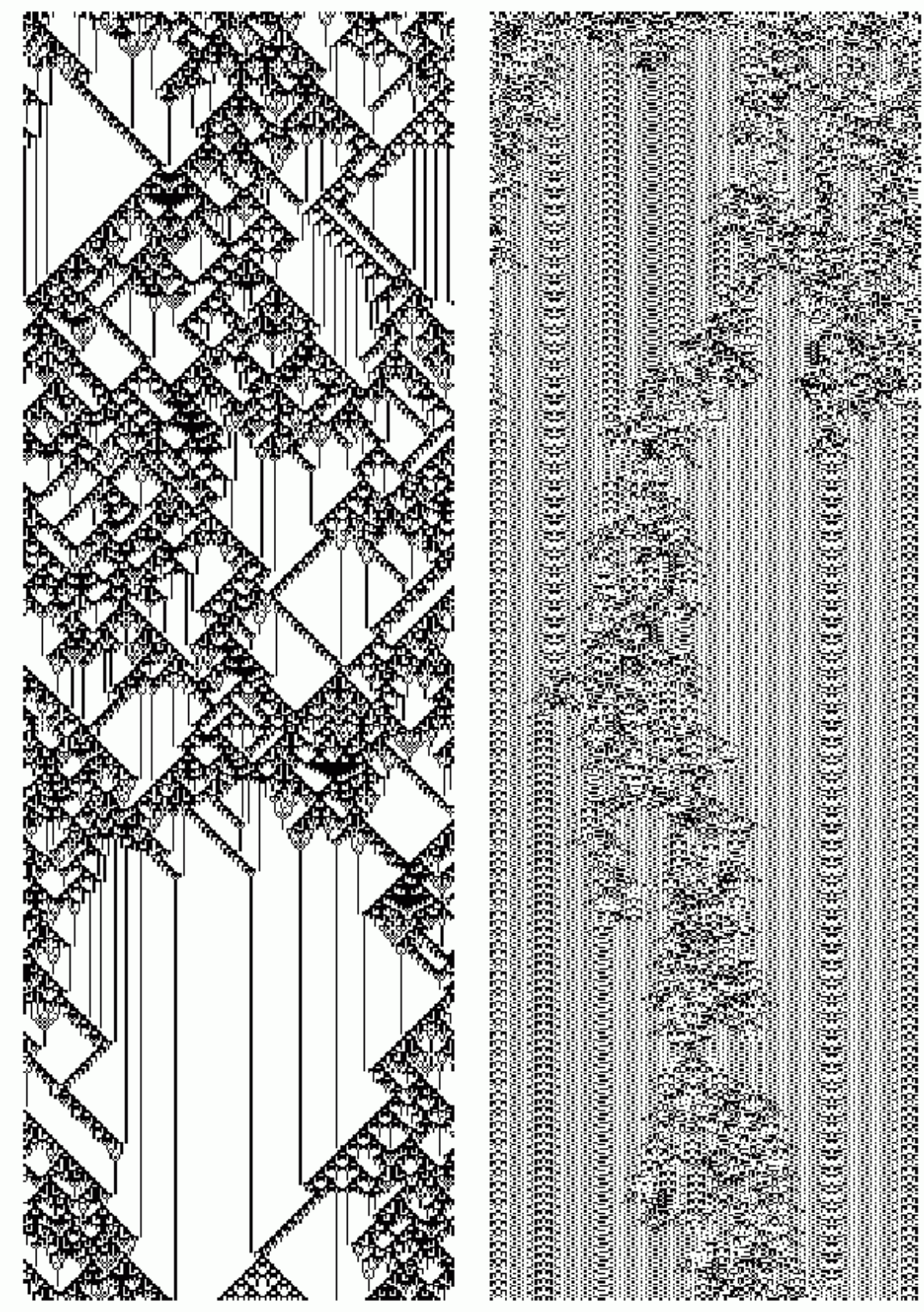


Klasse I (allg. Regel mit Code 128) Klasse II (Code 4)



Klasse III (Code 22) Klasse IV (Code 54)

weitere Beispiele für Klasse-IV-CA:



Klassifikation aller symmetrischen Regeln mit "quiescent state" und mit k=2, r=1 (P = peripheral, d.h. der Zustand der aktuellen Zelle selbst hat keinen Einfluss;
T = totalistisch):

Rule	Classif.	Rule	Classif.	Rule	Classif.	Rule	Classif.
0	T,P,I	72	I	128	T,I	200	II
4	II	76	II	132	I	204	II
18	III	90	P,III	146	III	218	II
22	T,III	94	II	150	T,III	222	II
32	I	104	T, I	160	P, I	232	T, II
36	II	108	II	164	II	236	II
50	I/II	122	III	178	II	250	P, I
54	III	126	T,III	182	III	254	T, I

(aus Adami1998)

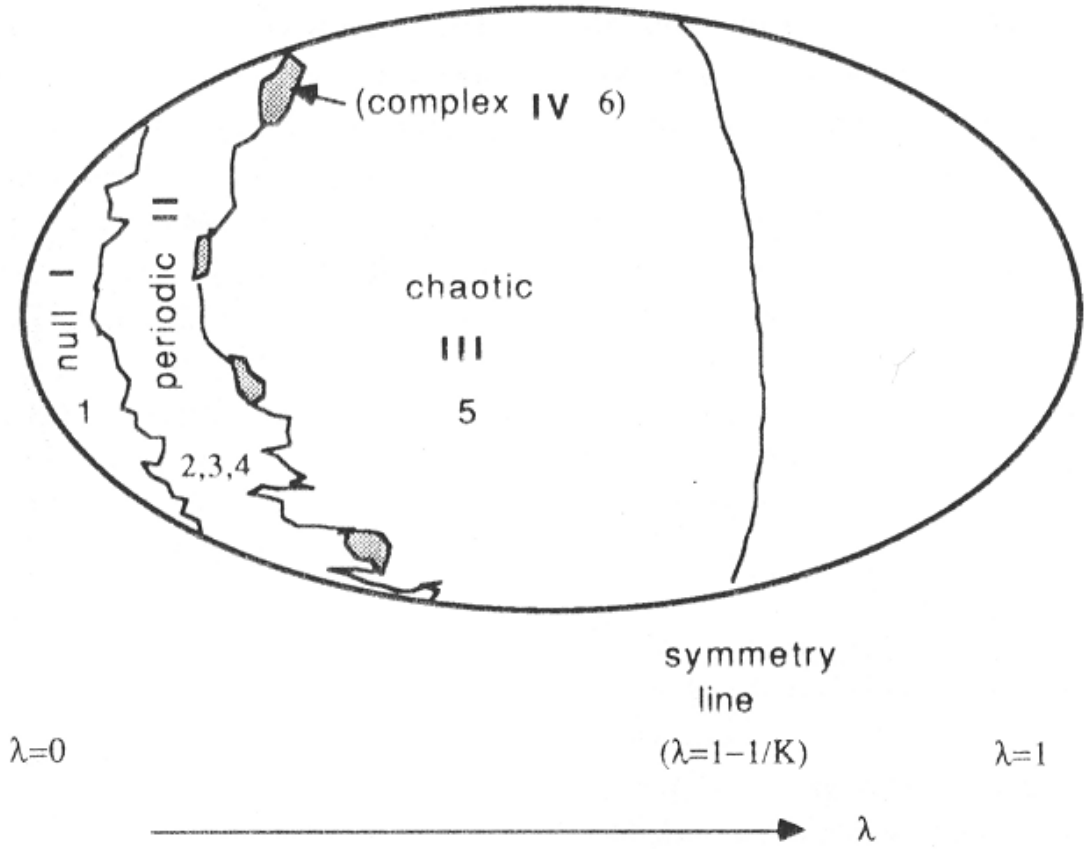
Interessanter als die bloße Auflistung ist die Struktur des "*rule space*", d.h.: welche Klassen sind "benachbart"? Wie ändert sich das Verhalten, wenn man nur 1 Bit der Regel-Tabelle ändert?

- zunächst einmal lässt sich aus der Kenntnis der Regeln wenig über die Dynamik des CA erschließen (außer in einfachen Fällen, wo das Vorliegen von Klasse I sofort evident ist).

- Eine interessante Charakterisierung der Funktionstabelle liefert der Anteil λ der Einträge $\neq 0$:

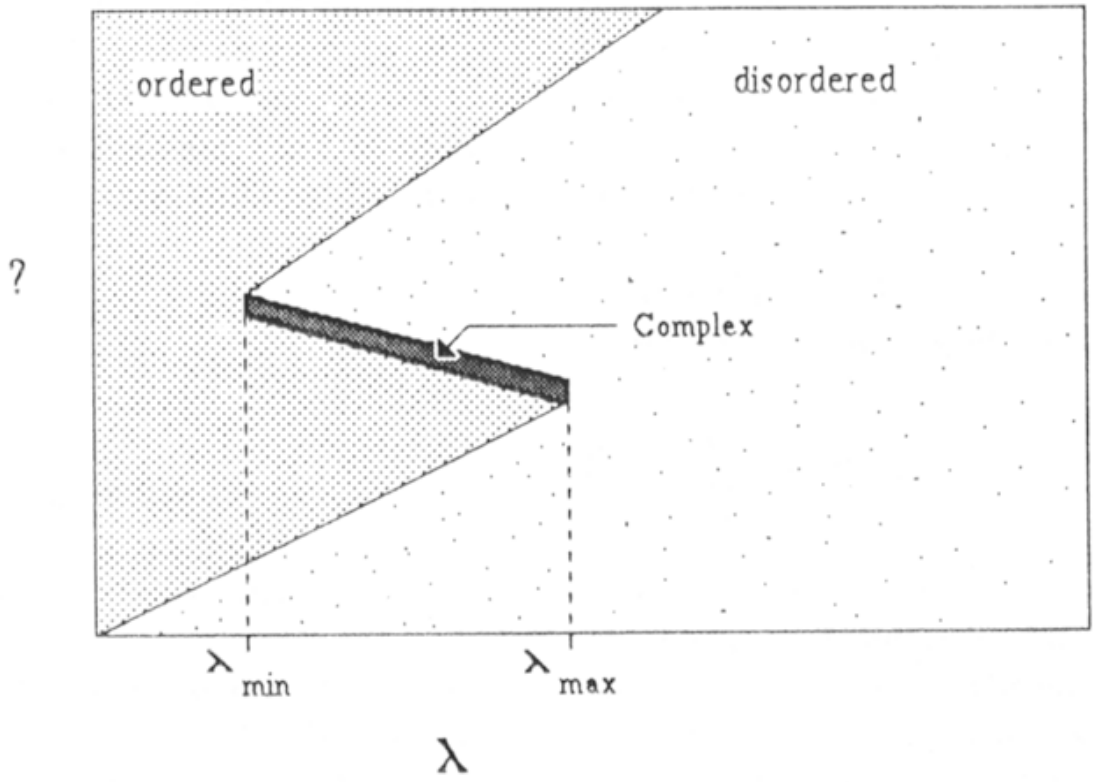
Mit wachsendem λ gelangt man von Klasse I in der Regel zu Klasse II und III, und schließlich aus Symmetriegründen in umgekehrter Reihenfolge wieder zurück. Klasse IV ist eingestreut "zwischen" II und III.

Li, Packard & Langton (1990) skizzieren den "rule space" wie folgt:



"Edge-of-Chaos-Prinzip":
die interessantesten Strukturen (mit größtem morphogenetischem und Rechen-Potenzial, d.h. Klasse IV) liegen "am Rand des Chaos", d.h. zwischen den Klassen II und III.

Die Struktur der "Chaos-Grenze" ist jedoch bei den CA nicht ganz klar, da ein zweiter, ordnender Parameter (neben λ) nicht definiert werden konnte. Dieser bleibt hypothetisch:



Reaktions-Diffusions-Systeme

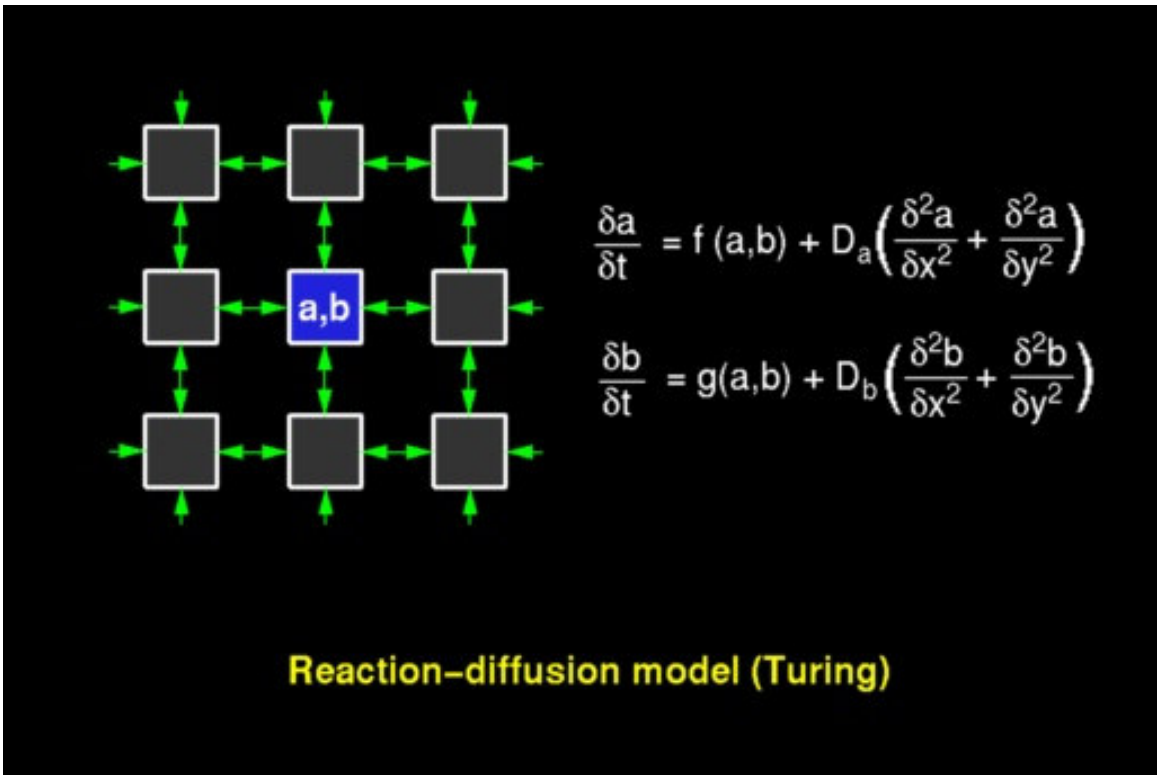
Gegenstück zu zellulären Automaten:
ebenfalls raumorientiert, mit fester Nachbarschafts-relation und kontextsensitiven Regeln
aber: **kontinuierlich** in Raum, Zeit und Strukturen

Formalismus: partielle Differentialgleichungen

(diese müssen aber für die Realisierung der Simulation im Rechner wieder diskretisiert werden → enger Bezug zu zellulären Automaten!)

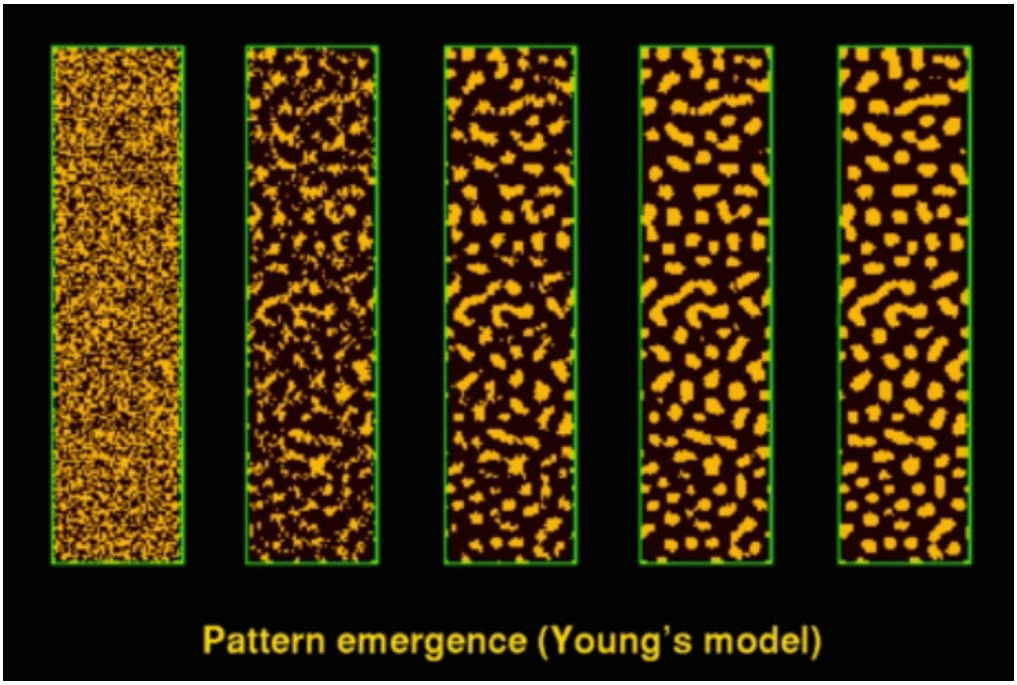
Grundideen:

- Substrate diffundieren im Raum, wechselwirken miteinander und mit Objekten
- Systeme partieller Differentialgleichungen legen die Dynamik der Substrate fest
- charakteristisch: Modellierung von (Konzentrations-) *Feldern*, -gradienten.



bei bestimmten Funktionen und Parameterkombinationen kann es zur räumlichen und zeitlichen (makroskopischen) Musterbildung kommen – auch bei homogenem Ausgangszustand! ("Symmetriebrechung")

Beispiel:



- räumliche Muster: Patches, Streifen, Zickzackmuster, Wellen...
- zeitliche Muster: Oszillationen, Bewegung von Wellenfronten, Interferenzmuster...

Erklärungsleistung von R.-D.-Modellen

- Wahl von f und g bestimmt Mustertyp und Stabilität
- Anfangsbedingungen („Keim") bestimmen Realisierung
- Substratverbrauch durch zeitlich veränderliche f und g darstellbar

Biologische Anwendungen:

- Schneckenhäuser, Muschelschalen
- Tier"fell"zeichnungen (Leopard, Zebra, Vögel, Fische, ...)

(Lange 2001)

Aktivator-Inhibitor-Modelle

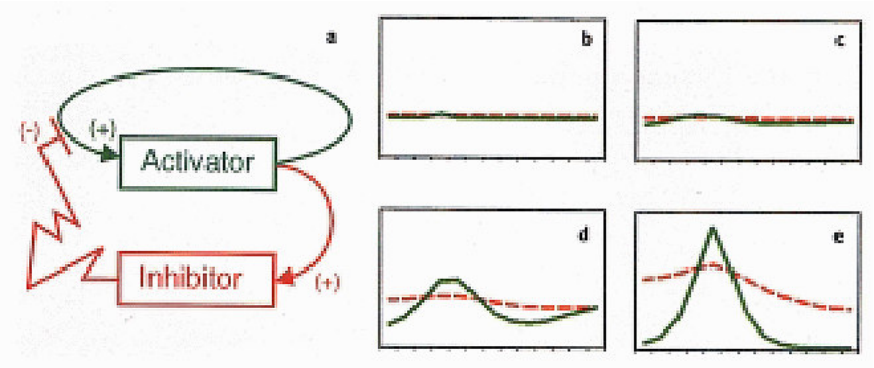
Spezialfall des Reaktions-Diffusions-Ansatzes

beispielhaft lässt sich hieran die Musterbildung in Reaktions-Diffusions-Modellen studieren

2 Substanzen:

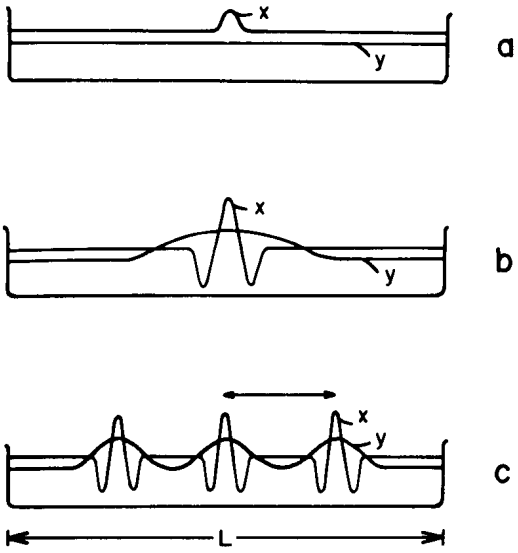
- Aktivator: fördert die eigene Entstehung ("autokatalytisch") *und* die des Inhibitors, fördert in visuellen Modellen zugleich die Pigmentbildung
- Inhibitor: hemmt die Bildung des Aktivators

Inhibitor diffundiert schneller als der Aktivator (größere Diffusionskonstante)



Ergebnis:

Die homogene Verteilung beider Substanzen (b) ist instabil; ein winziges lokales Anwachsen der Aktivator-Konzentration wächst weiter an (c, d), bis ein *steady state* erreicht wird, in dem sich Selbstaktivierung und Inhibition gerade ausgleichen. In größerer Entfernung vom Peak der Aktivator-Konzentration können weitere Peaks entstehen (x Aktivator, y Inhibitor):



math. Modell:

$a(x, t)$ Konzentration des Aktivators an Position x zur Zeit t

$b(x, t)$ entsprechend für den Inhibitor

r_a, r_b Abbauraten (decay rates)

D_a, D_b Diffusionskonstanten

weitere Größen: Konstanten

$$\frac{\partial a}{\partial t} = s\left(\frac{a^2}{b} + b_a\right) - r_a a + D_a \frac{\partial^2 a}{\partial x^2} \qquad (1)$$

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

Wenn man in (1) den Einfluss des Inhibitors weglässt und alle Konstanten zu 1 bzw. 0 vereinfacht, so ergibt sich mit

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

eine Differentialgleichung mit einem labilen *steady state* bei a=1 (jede kleinste Abweichung wird sofort verstärkt).

Für den Inhibitor hat man bei entspr. Vereinfachung:

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

$\Rightarrow$ *steady state* bei $b = a^2$

Wenn man die Inhibitorkonzentration im *steady state* annimmt und die Wirkung des Inhibitors auf a jetzt einbezieht, bekommt man:

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

und dies charakterisiert einen *steady state* bei a=1 mit *negativem* feedback (Ableitung hängt von $-a$ ab), d.h. durch die Berücksichtigung des Inhibitors ist der *steady state* nun *stabil*!

bei Einbeziehung der Raumdimension und der Diffusion: stabile räumliche Muster

Zur Simulation werden die Ableitungen durch Differenzen approximiert:

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

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

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

Einsetzen in (1) und (2) ergibt:

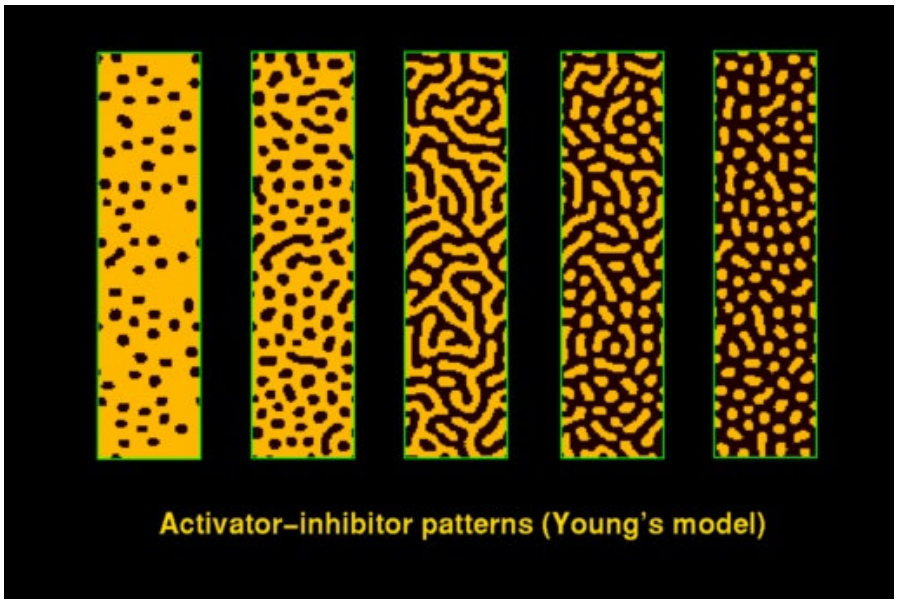
$$a_i(t+1) = a_i(t) + s\left(\frac{a_i^2(t)}{b_i(t)} + b_a\right) - r_a a_i(t) + D_a(a_{i-1}(t) + a_{i+1}(t) - 2a_i(t))$$

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

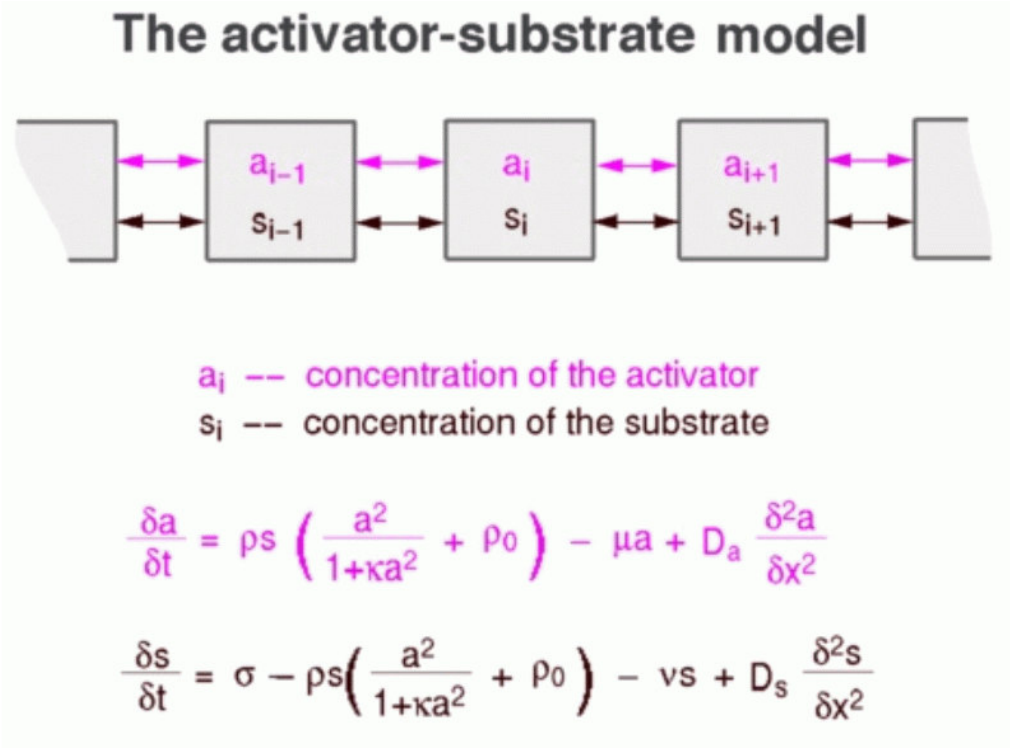
Diese Gleichungen kann man als Regeln eines 1D-Zellular-automaten mit 2 Variablen für jede Zelle ansehen

- allerdings sind die Variablen reellwertig – Abweichung von der Voraussetzung diskreter Zustände

Simulationsergebnisse im 2-dim. Fall:



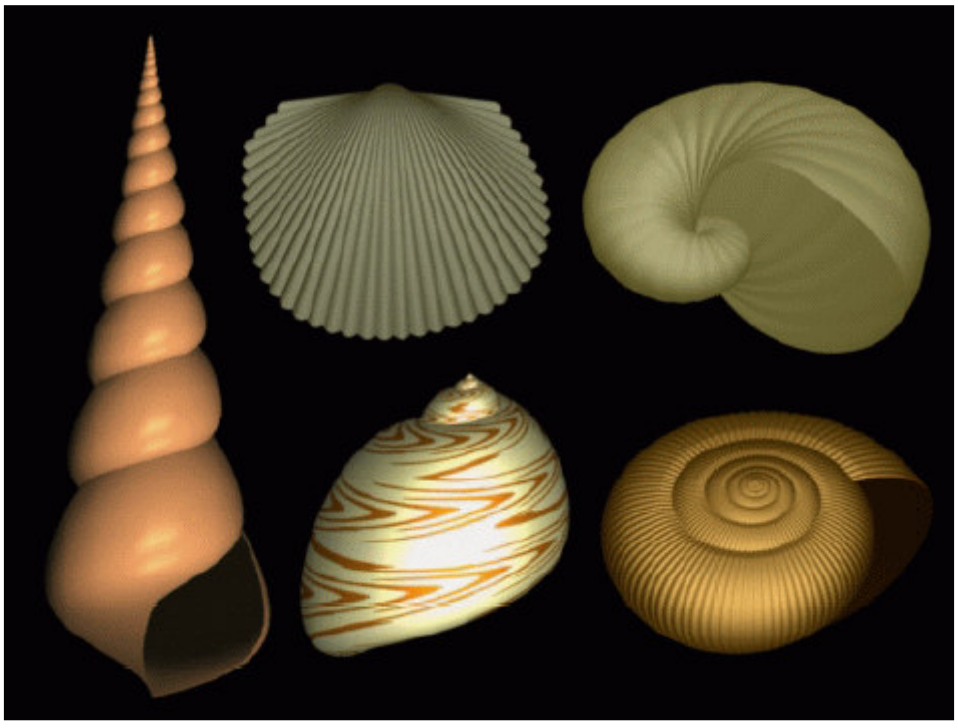
andere Variante: Aktivator-Substrat-Modell



Anwendung:

im linken Bild läuft die Zeit von oben nach unten (wie bei 1D-Zellularautomaten) – im rechten Bild ist das Streifenmuster lediglich geometrisch transformiert

Anwendung: Modellierung von Schneckenschalen (Fowler et al. 1992)



Simulationsergebnisse, aus Fowler et al. 1992

links: Foto, *rechts:* Simulation

Ausschnitte entnommen aus
Adami (1998), Lange (2001), Li et al. (1990)
(genaue Quellenangaben siehe http://www.uni-forst.gwdg.de/~wkurth/fs10_lit.htm)