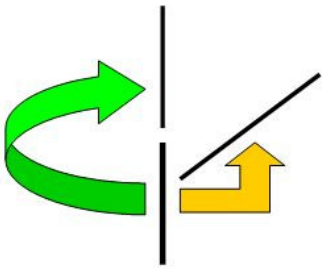
Relationale Wachstumsgrammatiken und andere L-System-Verallgemeinerungen

Der Schritt zu relationalen Wachstumsgrammatiken

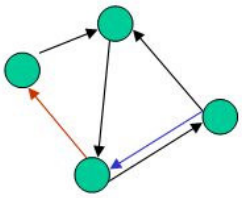
Nachteil von L-Systemen:

- 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)

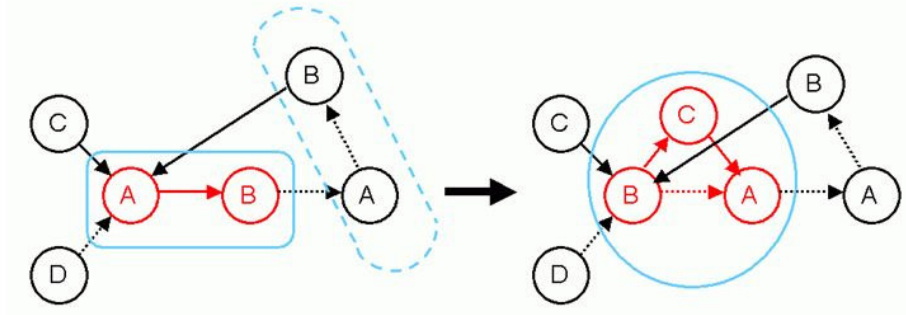


ebenfalls regelbasierter Mechanismus:

Graph-Grammatiken

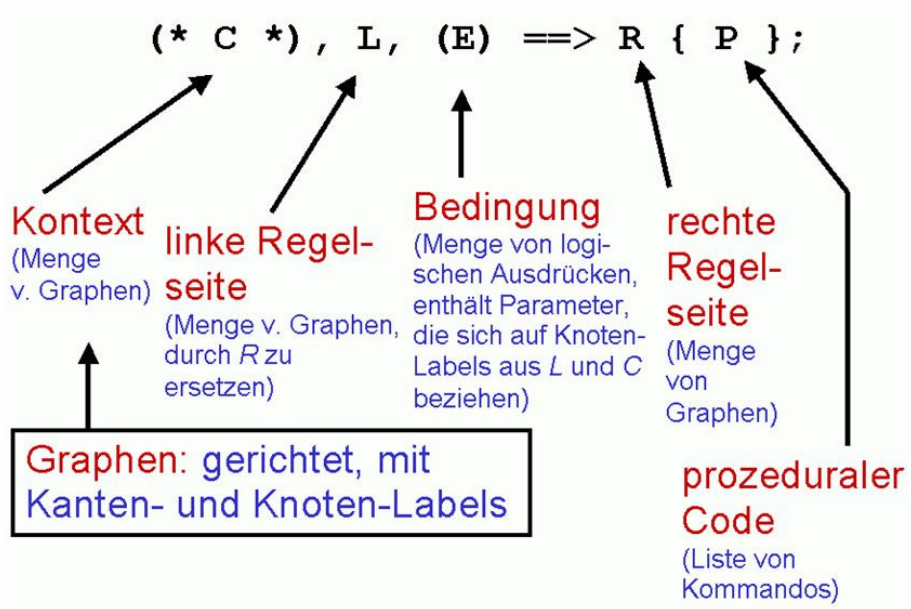
Regel:

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;`

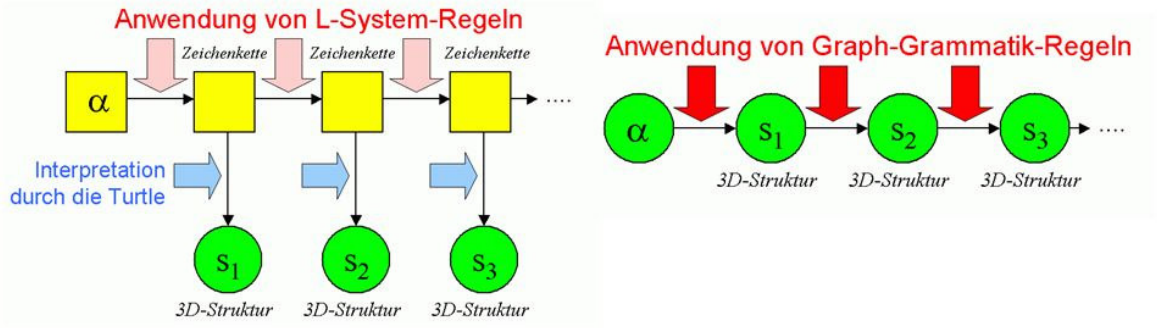
Instanzierungsregeln: einzelne Zeichen werden in Substrukturen aufgelöst, ohne Einfluss auf den nächsten Entwicklungsschritt
(Regel muss dann direkt in der Moduldeklaration stehen)

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

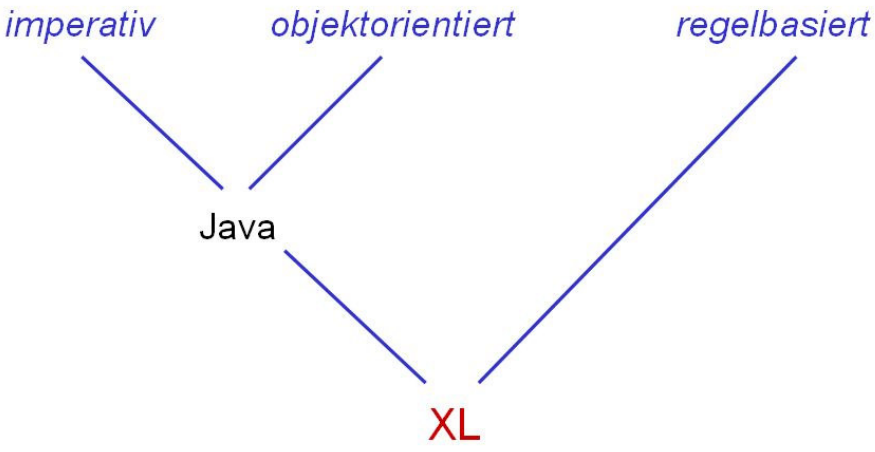
→ Erweiterungen:

- 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): [...]

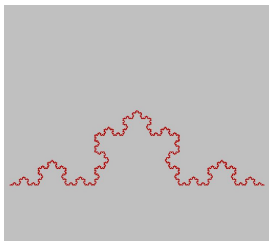
- Knoten der Graphen sind Java-Objekte, auch Geometrie-Objekte

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);
  ]
```

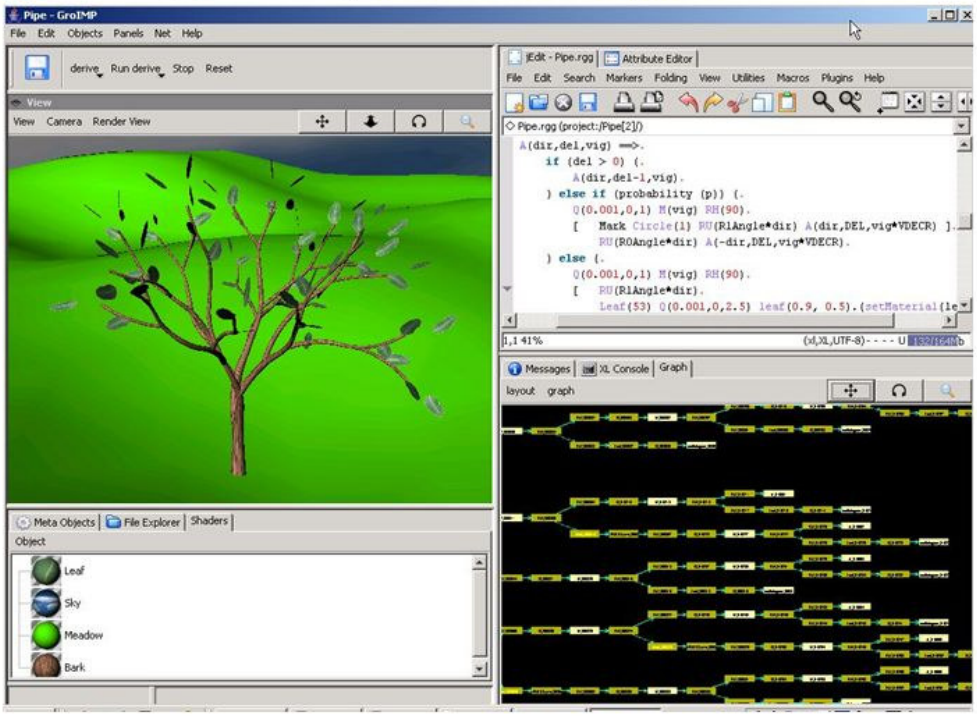
Knoten des Graphen

Kanten (Typ „Nachfolger“)



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

Anwendungsbeispiel: Modellierung von Parklandschaften

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

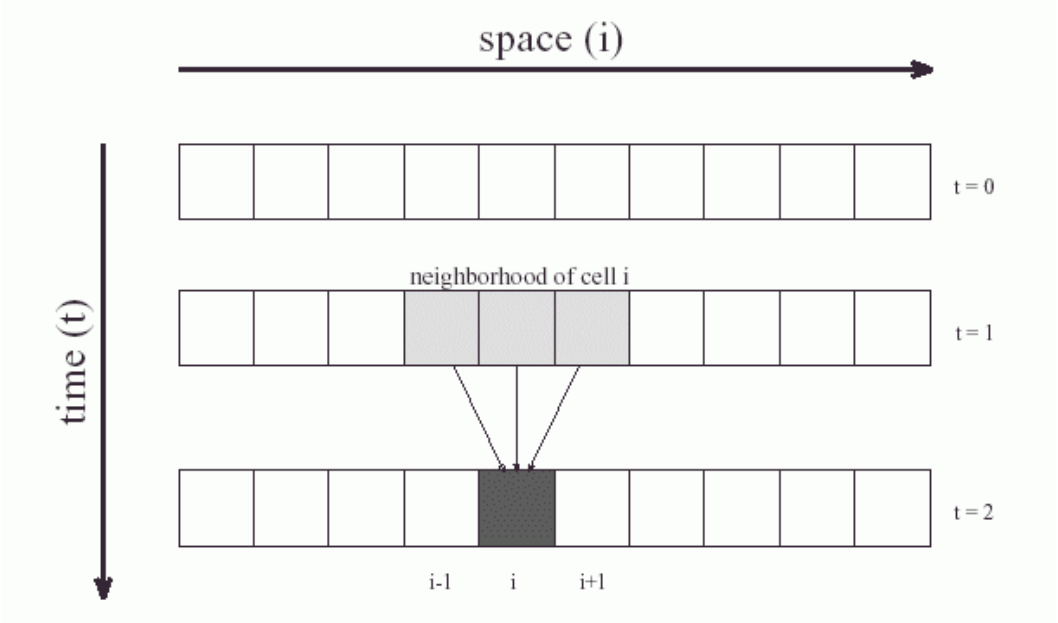
mit GroIMP generierte Erle in VRML-Welt

Zelluläre Automaten

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

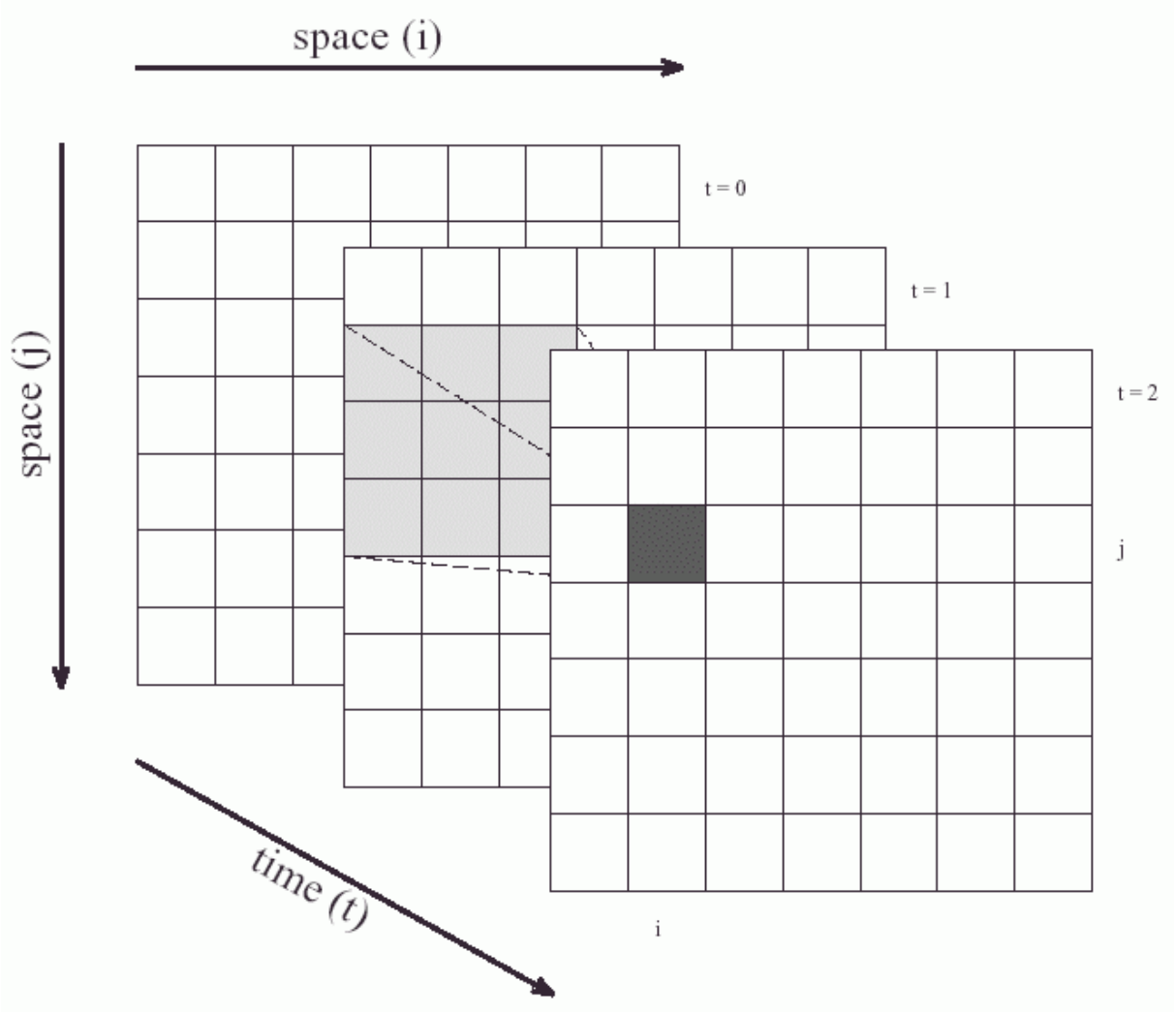
- mathematische Modelle mit diskretem Raum und diskreter Zeit
- die Zeit verläuft in Schritten
- Raum wird als *Gitter* oder *Array von Zellen* repräsentiert
- in den klassischen CA-Modellen kann jede Zelle nur endlich viele Zustände annehmen
- für jede Zelle gilt eine Menge lokaler Regeln, die festlegen, 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.

Das Problem der Gitter-Ränder:

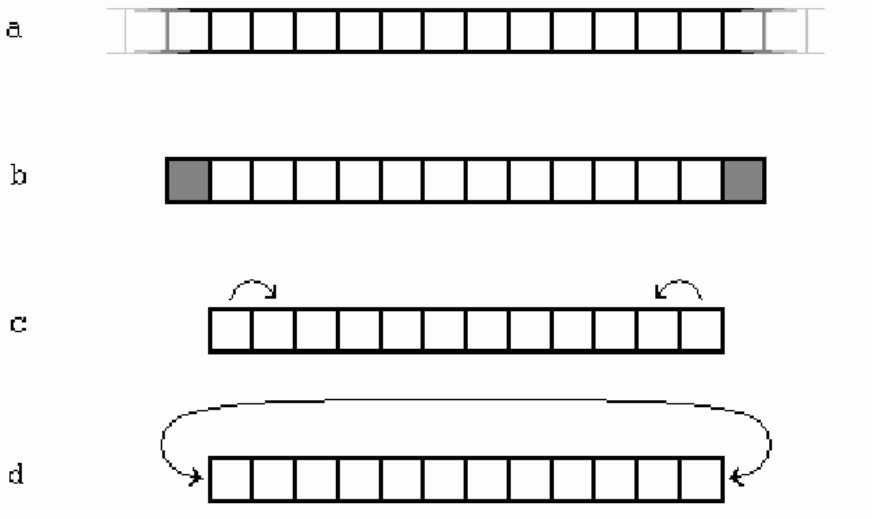
- Simulation eines wirklich unendlichen Gitters unmöglich (simulationstechnisch und möglicherweise durch reales System bedingt - natürliche Grenzen)
- Periodische Grenzen

Bei zwei-dimensionalen Automaten - „Torus“

- Reflektive Grenzen

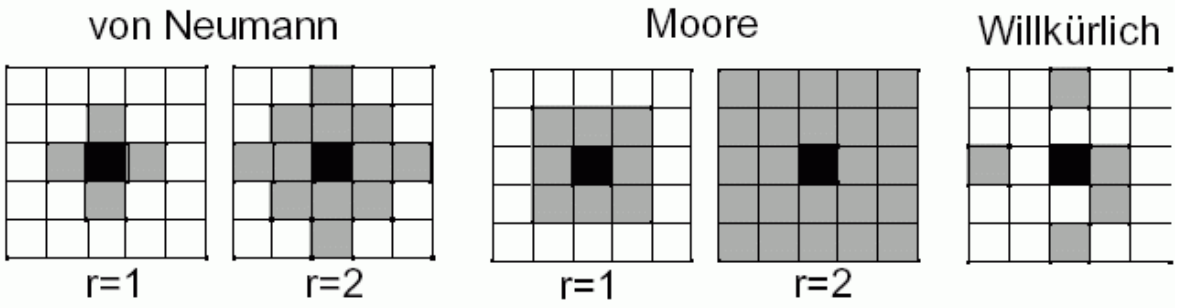
Wenn das reale System Grenzen hat, aber der Wert an der Grenze nicht vorgegeben ist.

- 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*"

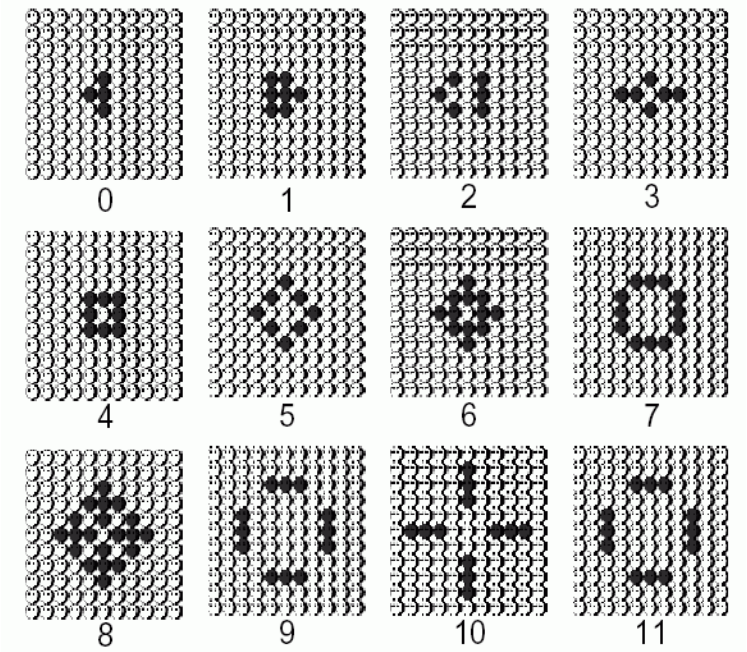
1970 von Martin Gardner in seiner mathematischen Kolumne in "Scientific American" vorgestellt, löste Welle weltweiter Forschung an diesem CA aus.

- 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.

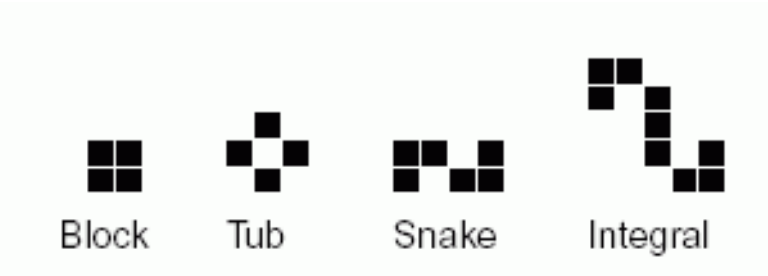
Beispiel-Entwicklung:



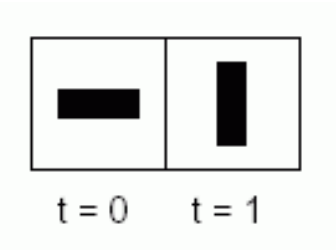
Muster lassen sich durch ihr Verhalten (bei der weiteren Entwicklung) klassifizieren.

Beispiele:

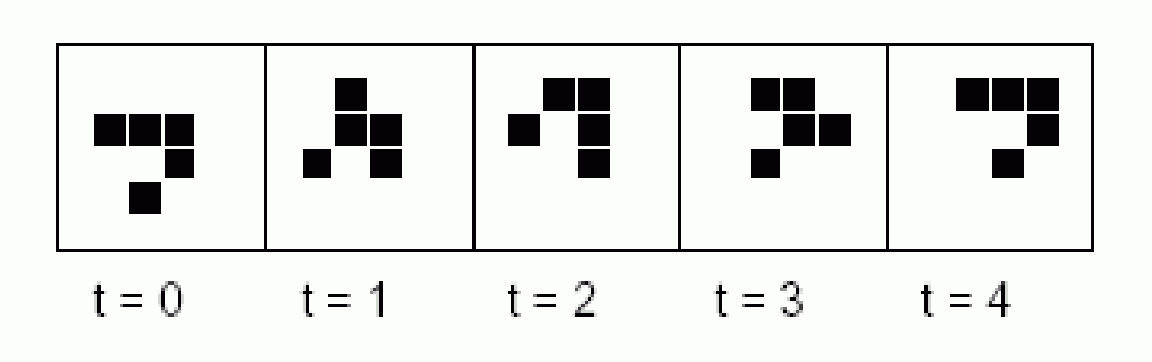
Typ I: "Stillleben", Muster, die sich nicht ändern (Fixpunkte der Übergangsfunktion):



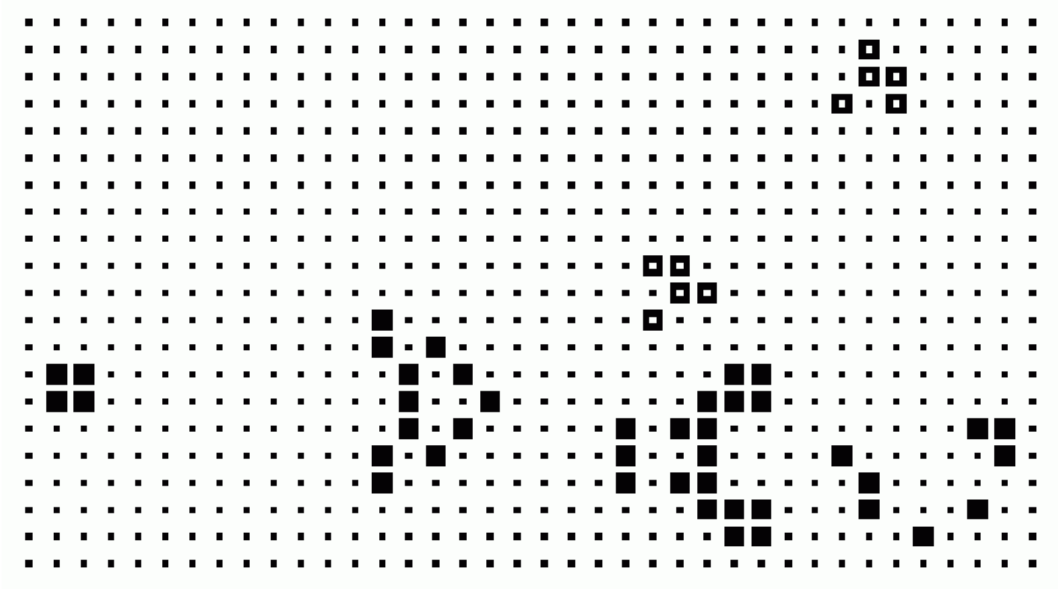
Typ II: Oszillatoren (periodische Muster): z.B. der "Blinker"



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).

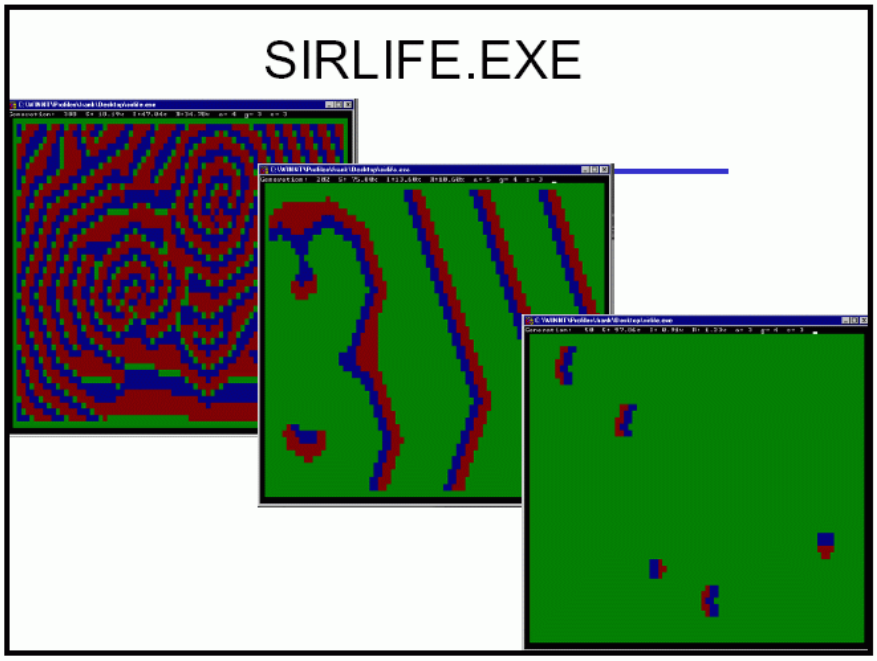
Typ V: unstabile Muster. Muster, die sich durch eine Folge von Zuständen weiterentwickeln und nie zum Ausgangszustand zurückkehren. Kleine Muster mit "langer" Entwicklungsphase vor der Stabilisierung heißen "Methusalems".

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:



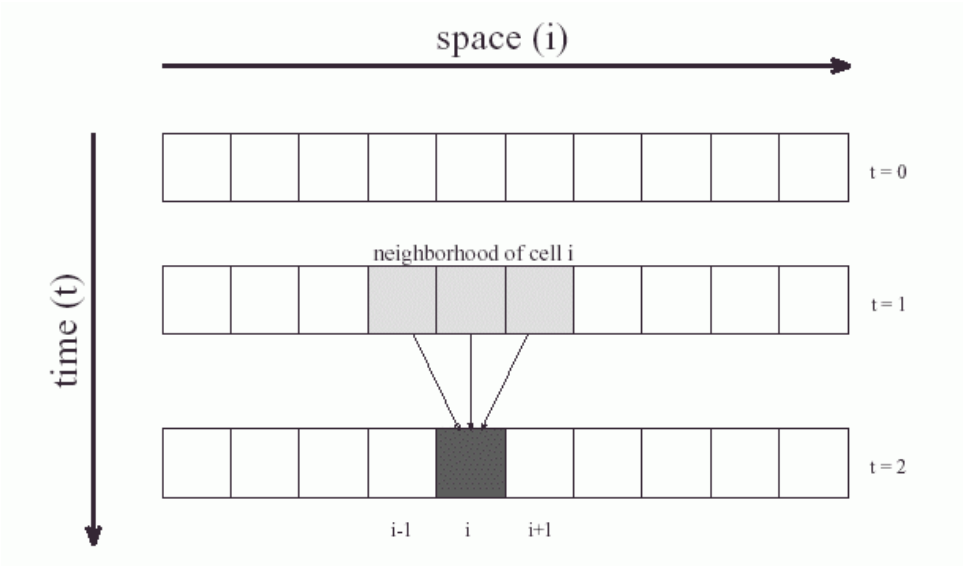
(Universität Leipzig)

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)



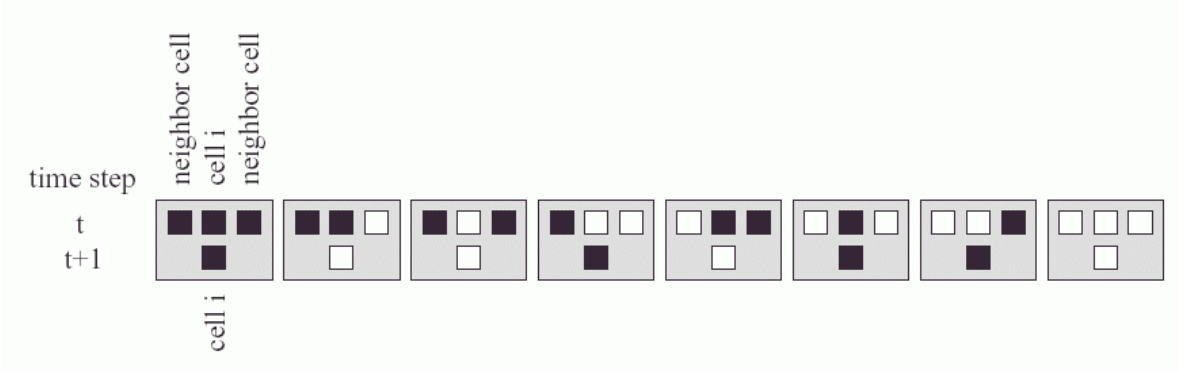
Beschreibung der Regel durch eine Funktionstabelle (look up table, *rule table*, manchmal auch der *Genotyp* genannt):

z.B.

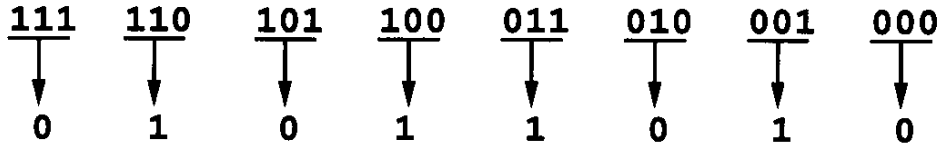
$a_{i-1}(t)$	$a_i(t)$	$a_{i+1}(t)$	$a_i(t+1)$
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

(*)

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):

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

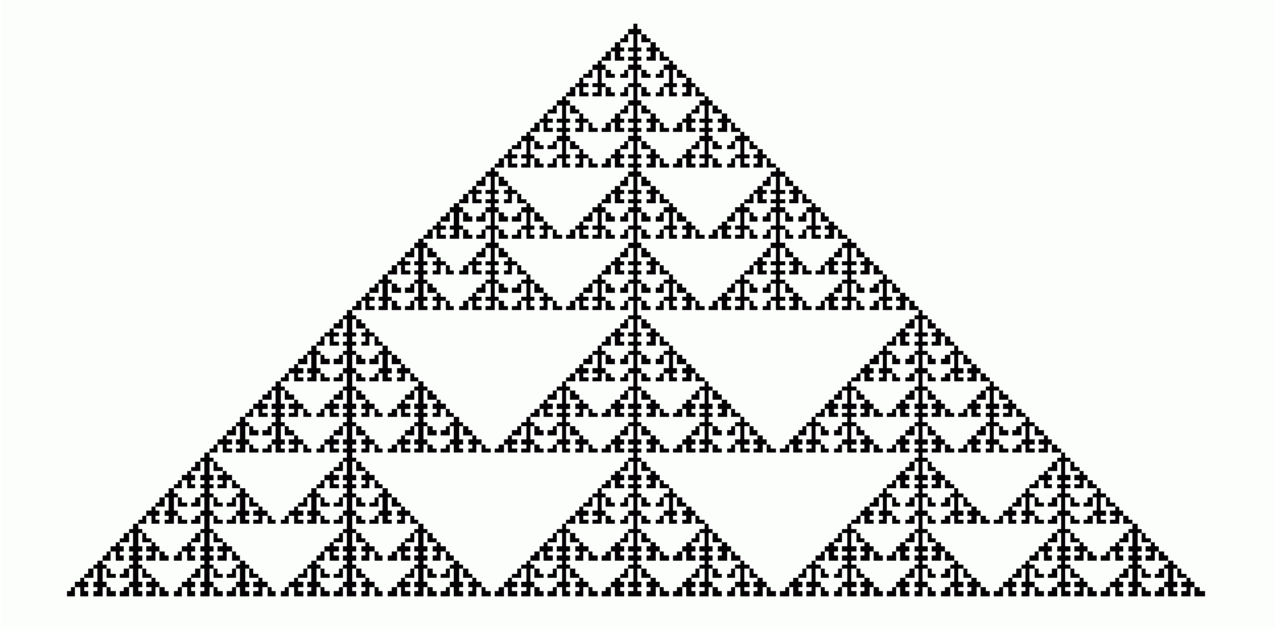
$$\begin{aligned} \Sigma=1 &: 01\,\mathbf{0}\,00 \longrightarrow \mathbf{0} \\ \Sigma=2 &: 10\,\mathbf{0}\,01 \longrightarrow \mathbf{1} \\ \Sigma=3 &: 01\,\mathbf{1}\,01 \longrightarrow \mathbf{0} \\ \Sigma=4 &: 11\,\mathbf{1}\,01 \longrightarrow \mathbf{1} \\ \Sigma=5 &: 11\,\mathbf{1}\,11 \longrightarrow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

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

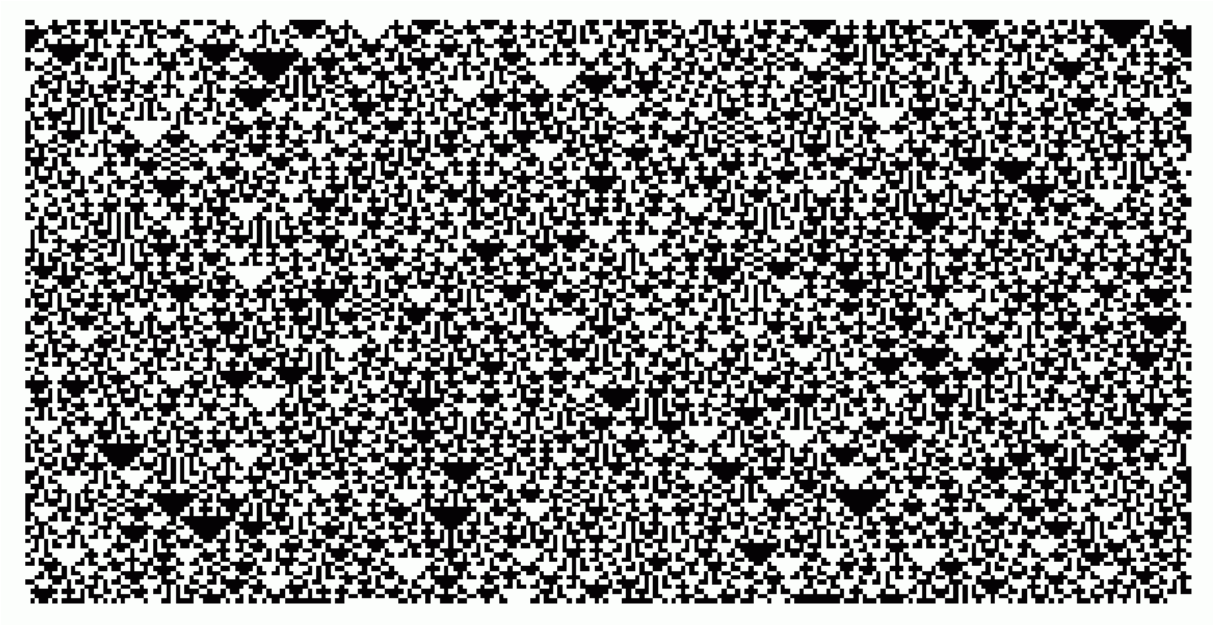
- Symmetrie der Regeln (z.B. 100 und 001 müssen denselben Zustand liefern)

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)

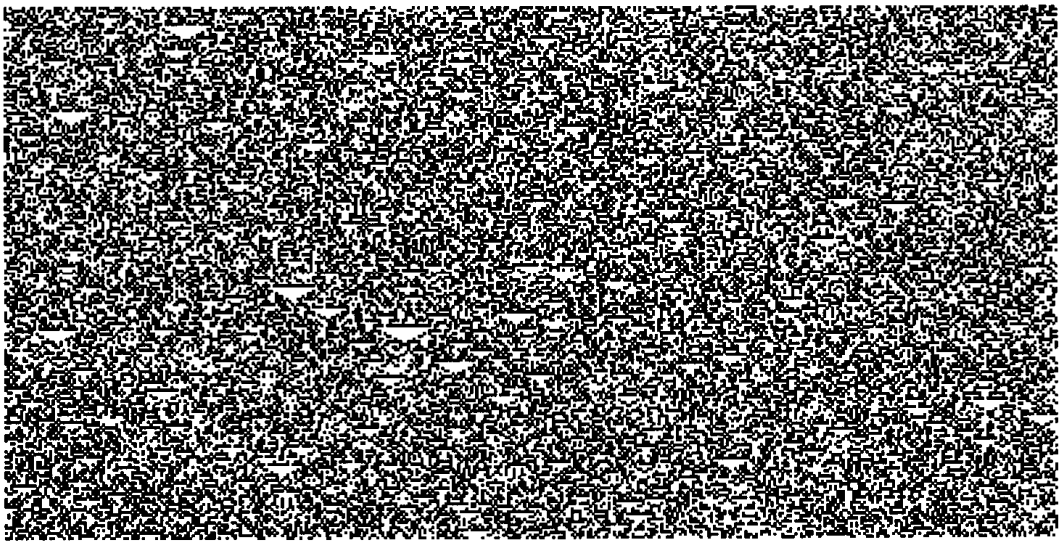
Klasse II:
liefert Folgen von einfachen stabilen oder periodischen Strukturen (endlose Zyklen derselben Zustände).

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.

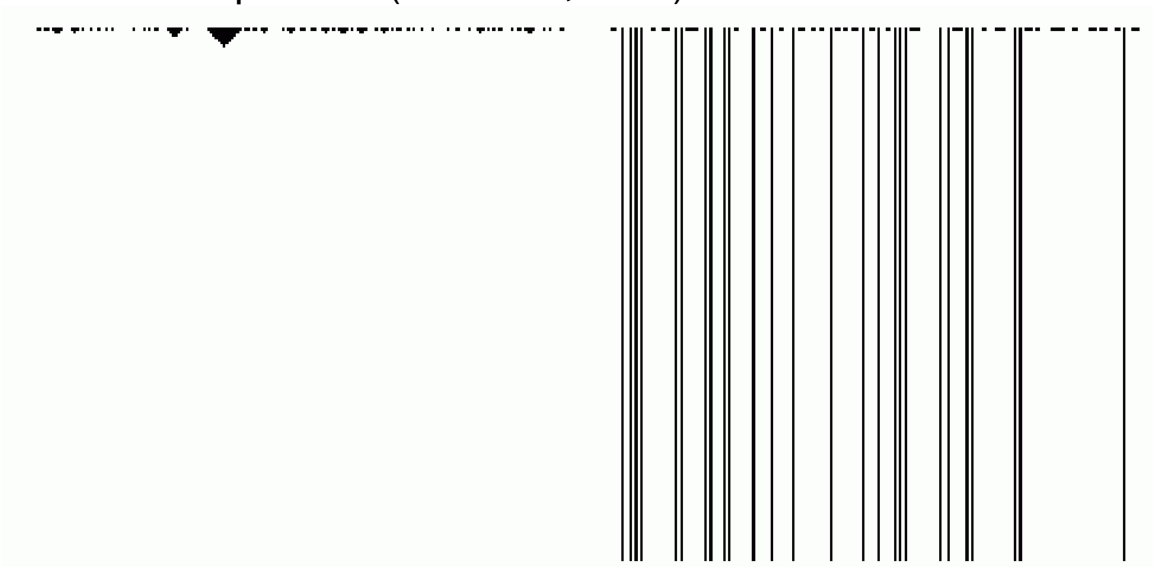
Beispiel:

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

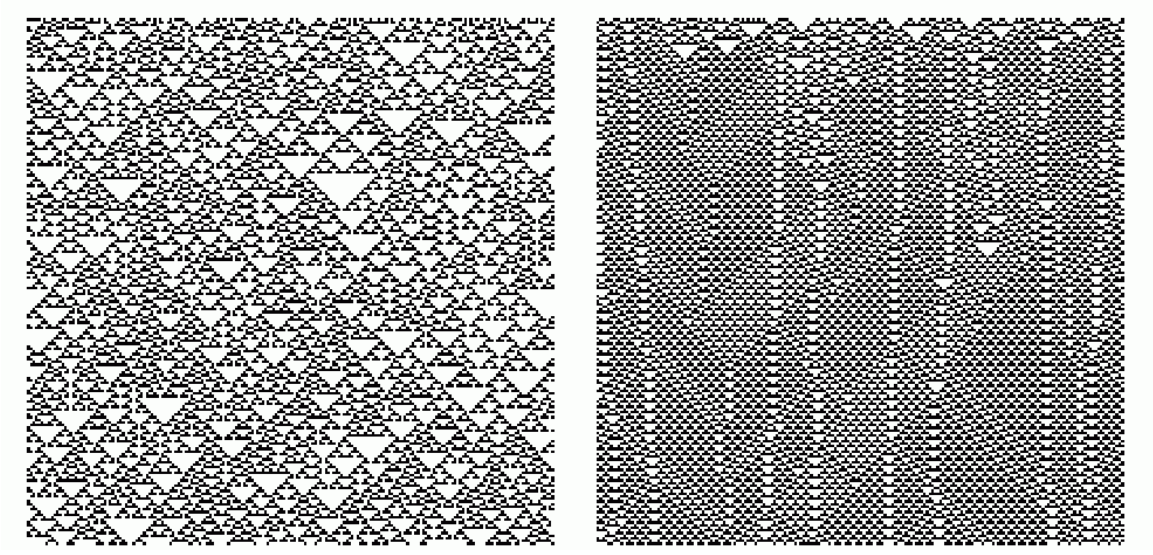
Die 4 Klassen unterscheiden sich auch durch die Wirkung, die kleine Änderungen in den Anfangsbedingungen haben (Wolfram 1984):

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 $k = 2$, $r = 1$):

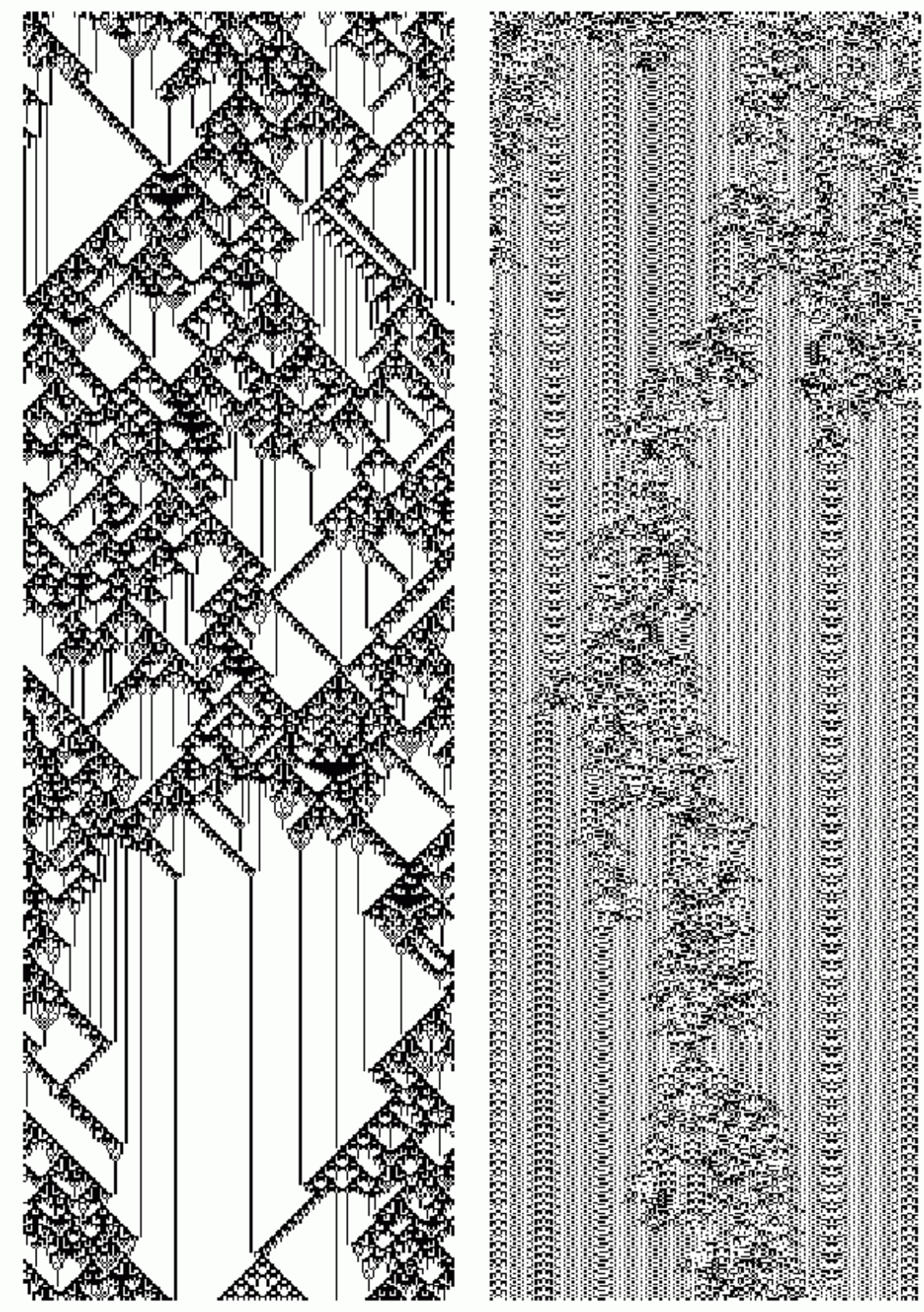


Klasse I (allg. Regel mit Code 128) Klasse II (Code 4)



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

weitere Beispiele für Klasse-IV-CA:

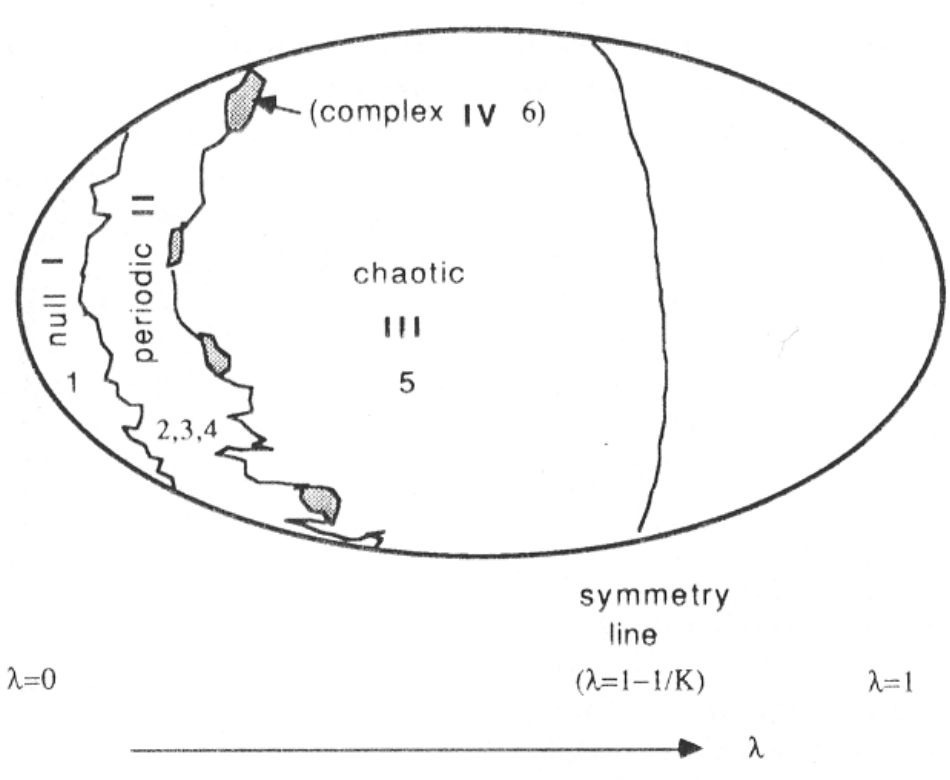


Frage nach der Struktur des "*rule space*", d.h.: welche Klassen sind "benachbart"? Wie ändert sich das Verhalten, wenn man nur 1 Bit der Regel-Tabelle ändert?

- 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 $\neq 0$:

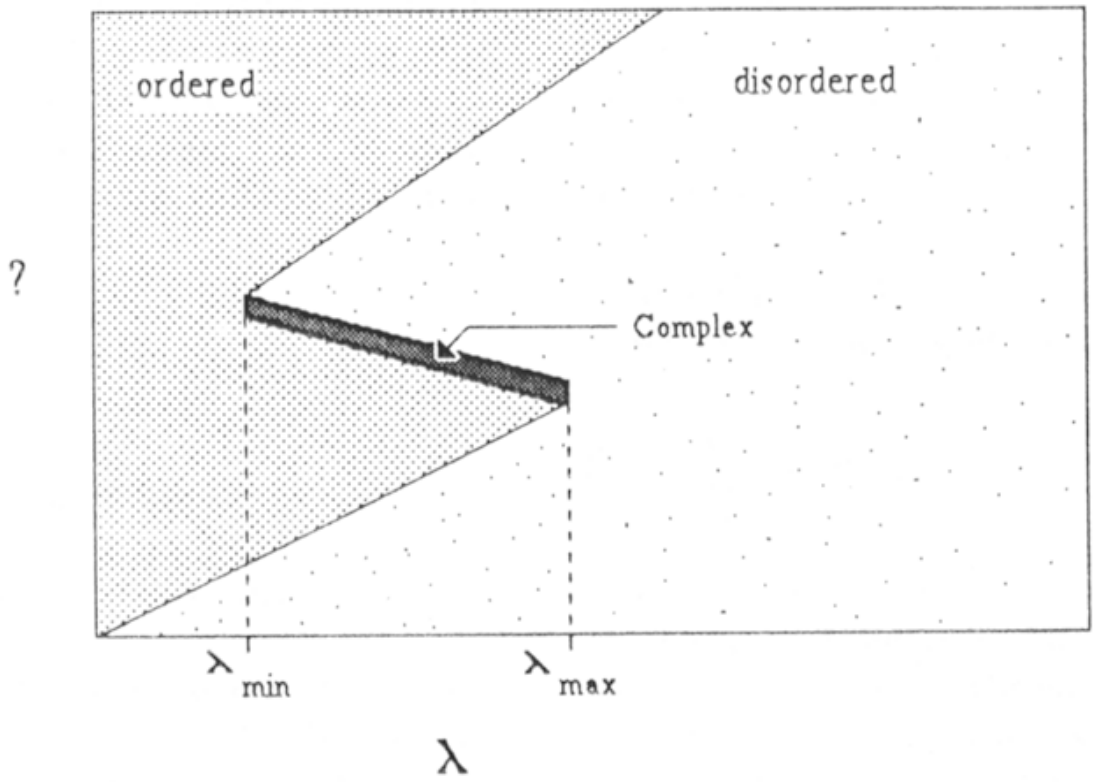
Mit wachsendem λ gelangt man von Klasse I in der Regel zu Klasse II und III, und schließlich aus Symmetriegründen in umgekehrter Reihenfolge wieder zurück. Klasse IV ist eingestreut "zwischen" II und III.

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:



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