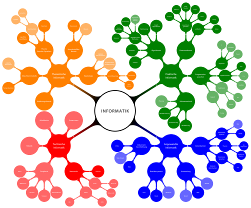
UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR
THEORETISCHE INFORMATIK

Kapitel 34

Berechenbarkeit

Möglichkeiten und Grenzen von Computern

Vorlesung Einführung in die Informatik 2 vom 6. Mai 2014 von Till Tantau



Lernziele von Kapitel 34

1. Ein mathematisches Modell für Computer kennen
2. Begriff der Berechenbarkeit verstehen
3. An Beispiele verstehen, dass Computer nicht alles berechnen können
4. Aussage der Church-Turing-These kennen

Gliederung von Kapitel 34

Auf dem zum Begriff des »Berechenbaren«.

Public domain

- Um 1900 erschien dem großen Mathematiker *David Hilbert* die Zeit reif, eine Vision zu verwirklichen.
- Sein Ziel war es, nicht mehr für jeden mathematischen Satz *mühselig einen Beweis zu ergrübeln,* er wollte *den Beweis einfach »berechnen«*.
- Gesucht war also eine Art »Verfahren«, bei dem man mit dem Satz begann und dann den Beweis als Resultat erhielt (oder einen Beweis für das Gegenteil).

Auf dem zum Begriff des »Berechenbaren«.

Public domain

- Man begann also, den Begriff der »Berechenbarkeit« zu formalisieren.
- Wie bei Mathematikern üblich, waren diese Modell *nur für Mathematiker verständlich*. Er war völlig unklar, ob eines der vorgeschlagenen Modelle *tatsächlich* den Begriff der Berechenbarkeit adäquat fasste.
- Die Arbeit 1936 von Alan Turing war anders: Hier wurde erstmalig ein *maschinenbasiertes Modell* vorgeschlagen.

Turing machte alles anders in seiner Definition

State of the Art vor Turing

- Komplexe, undurchsichtige mathematische Kalküle
- Begründet wurden die Kalküle dadurch, dass sie »bei vielen Beispielen funktionierten«
- Rechnen mit Zahlen
- Grenzen der Mächtigkeit der Kalküle unklar

Turings Ideen

- Minimalistisches *mathematisches Modell* einer *Maschine*
- Eine Argumentation, dass *alle überhaupt denkbaren Berechnungen* durch das Modell abgedeckt werden
- Rechnen mit Zeichenketten
- Grenzen der Berechenbarkeit (recht) leicht beweisbar

Turings Ziel

Turing wollte formal fassen, »wie ein Mathematiker eine sehr lange Berechnung durchführen würde«:

Ausgangspunkt

Eine »Berechnung« bedeutet für Turing, einen formalen Beweis zu überprüfen.
Der Mathematiker hat dazu

- einen *beliebig großen Stapel an Karopapier*,
- einen *unerschöpflichen Bleistift*,
- einen *unerschöpflichen Radiergummi* und
- eine *eiserne Disziplin*.

Ein Turings Worten:

> *»A man provided with paper, pencil, and rubber, and subject to strict discipline, is in effect a universal machine.«*

Turings erste Idee: Endlich vielen Symbole pro Zelle

Turings Ideen

- Der Mathematiker mag gute Augen haben,. . .
- . . . jedoch passen in ein Kästchen nur endlich viele unterschiedliche Symbole.
- Es nützt dem Mathematiker auch nichts, Symbole »über mehrere Kästchen zu strecken«,. . .
- . . . denn das entspricht einfach mehreren Symbolen in mehreren Kästchen.

Formalisierung

- Es gibt ein *Bandalphabet*. Wie jedes Alphabet ist dieses *endlich*.
- Die Symbole stehen in Kästchen, formal (Speicher-)*Zellen* genannt.

Turings zweite Idee: Zellen auf einem Band

Turings Idee

- Die Zellen mögen auf einem zweidimensionalen »Blatt« verteilt sein,. . .
- . . . bei einer sehr langen Rechnung muss man aber so oder so »blättern«.
- Da kann man die zweidimensionalen Blätter gleich weglassen und nur ein *Band* benutzen.

Formalisierung

- Die Zellen werden in *eindimensionalen Bändern* angeordnet.
- Prinzipiell sind Bänder »unendlich lang«, sie sind aber bis auf einen endlichen Anteil leer.
- Dieser endliche Anteil ist damit einfach ein *Wort über dem Bandalphabet*.

Turings dritte Idee: Diskrete Zeitschritte

Turings Idee

Der Mathematiker kann nicht »unendlich schnell« arbeiten.

Formalisierung

Die Berechnung verläuft in *diskreten Schritten*.

Turings vierte Idee: Der Schreib-Lese-Kopf

Turings Idee

- Während der Mathematiker arbeitet, hat er immer nur ein Blatt »aktuell vor sich«.
- Will er ein anderes Blatt anschauen, so muss er sich zunächst dorthin »durchblättern«.

Formalisierung

- Für jedes Band gibt es einen »Schreib-Lese-Kopf«.
- Dies ist einfach ein Index einer Bandposition.
- In jedem Berechnungsschritt kann sich der Kopf bewegen, aber nur eine Zelle vor oder zurück.

Turings fünfte Idee: Die endlich vielen Geisteszustände

Turings Idee

- Während der Mathematiker arbeitet, ist sein Gehirn immer in einem »Geisteszustand«.
  Beispiel: Ich muss jetzt den aktuellen Beweis überprüfen.
  Beispiel: Ich suche nach der schließenden Klammer.
- Turing argumentiert, es gäbe zwar viele, aber eben nur endlich viele Geisteszustände.

Formalisierung

Es gibt eine endliche Menge Q von Zuständen.

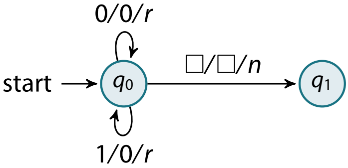
Turing-Berechenbarkeit

Definition (Turing-Berechenbar)

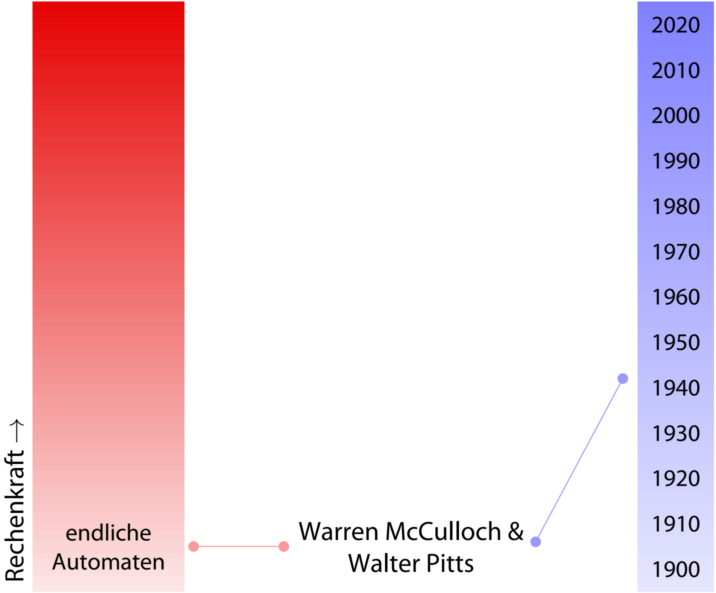
Eine Funktion f, die Wörter auf Wörter abbildet, heißt *Turing-berechenbar,* wenn eine Turing-Maschine bei Eingabe eines beliebigen Wortes w nach endlich vielen Schritten das Wort $f(w)$ auf ihrem Band stehen hat und anhält.

Beispiel (Eine Turing-Maschine für $f: w \mapsto 0^{|w|}$)

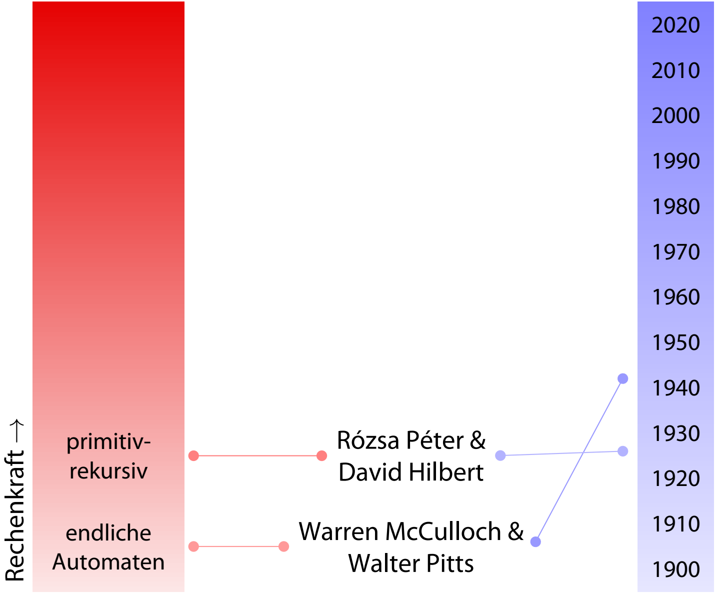
- Die Zustandsmenge ist $Q = \{q_0, q_1\}$.
- Der Anfangszustand ist q_0.
- Das Bandalphabet ist $\Gamma = \{0, 1, \square\}$.
- Das Eingabealphabet ist $\Sigma = \{0, 1\}$.
- Die Bandanzahl ist 1.
- Das Programm ist



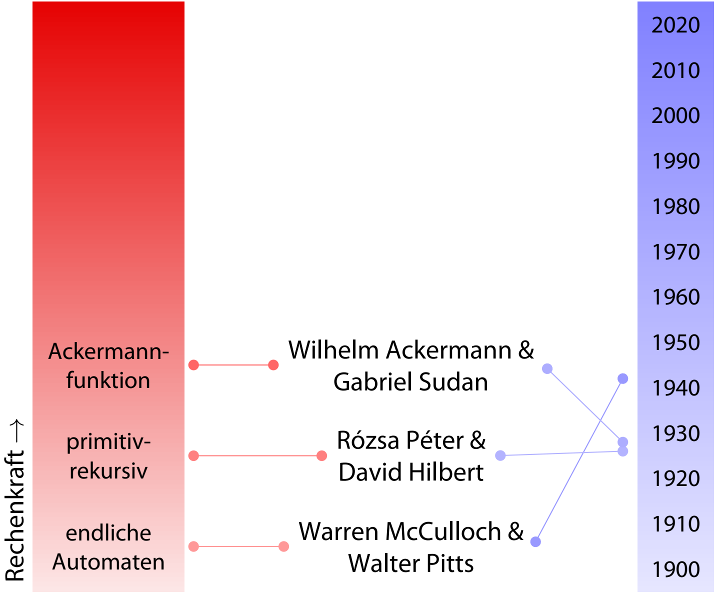
Geschichte der Berechnungsmodelle



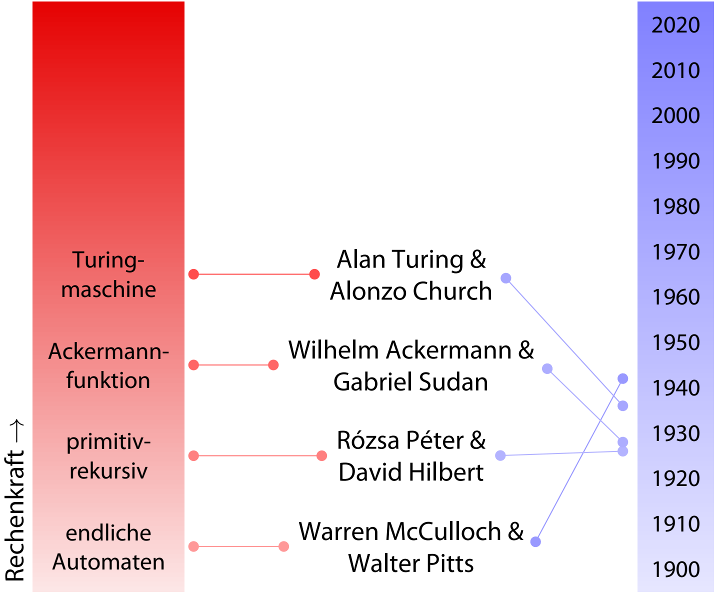
Geschichte der Berechnungsmodelle



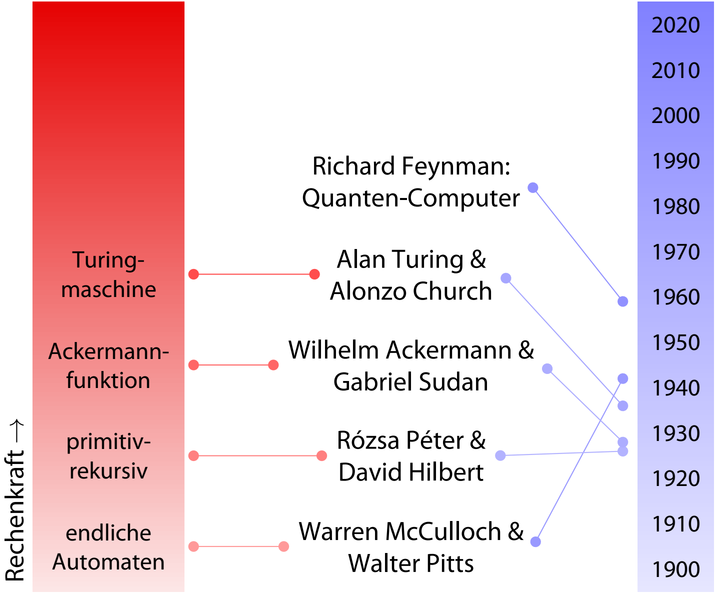
Geschichte der Berechnungsmodelle



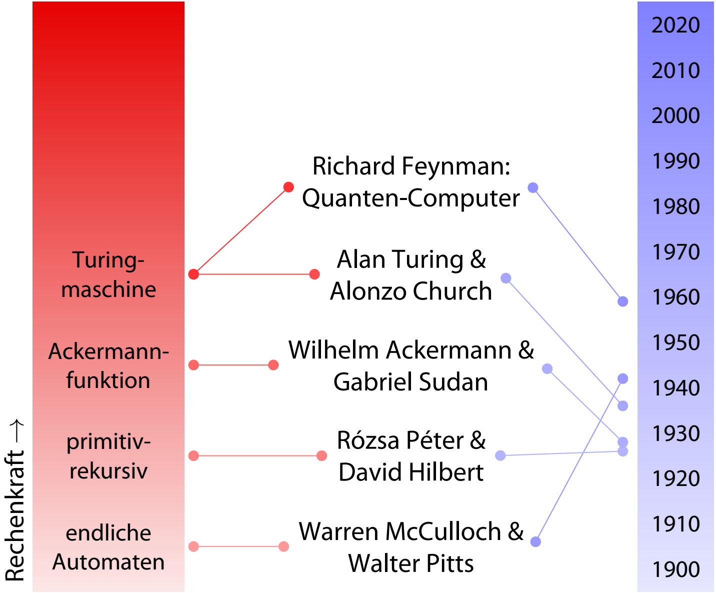
Geschichte der Berechnungsmodelle



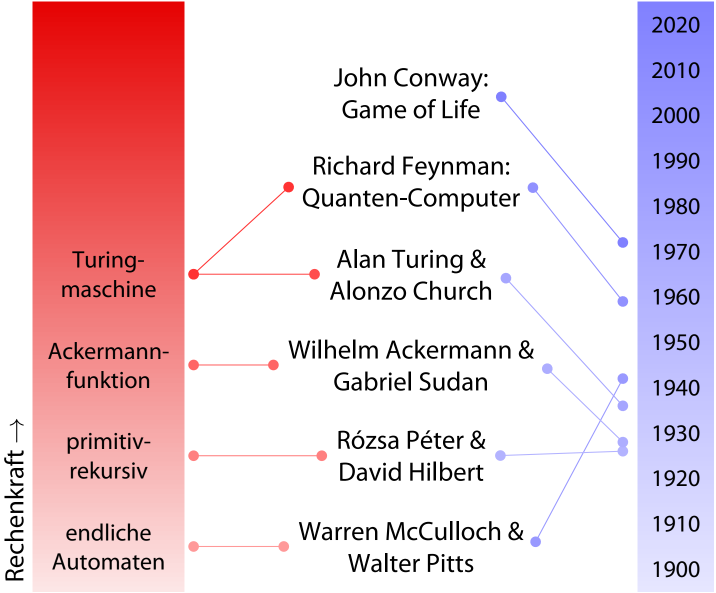
Geschichte der Berechnungsmodelle



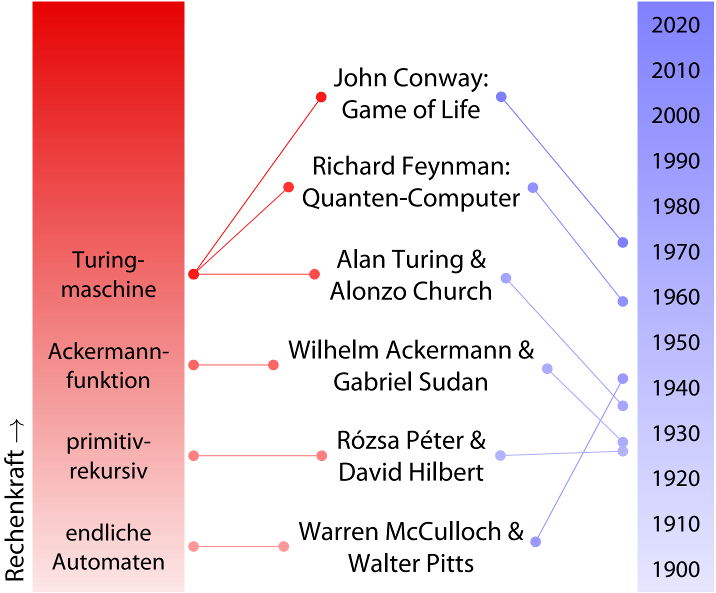
Geschichte der Berechnungsmodelle



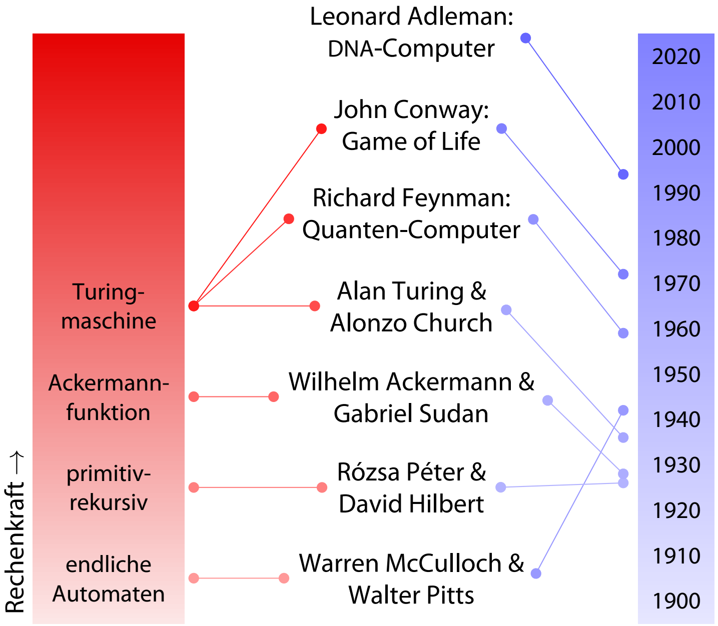
Geschichte der Berechnungsmodelle



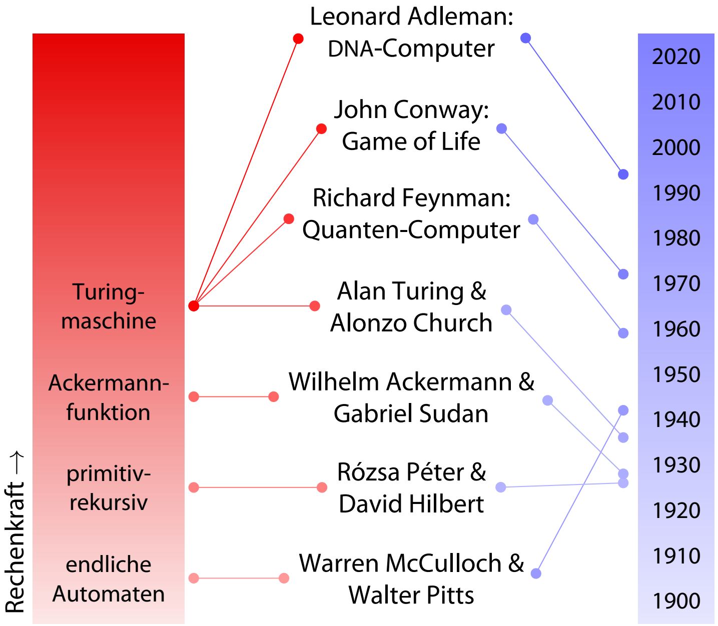
Geschichte der Berechnungsmodelle



Geschichte der Berechnungsmodelle



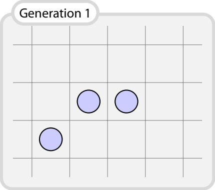
Geschichte der Berechnungsmodelle



Conways Game of Life.

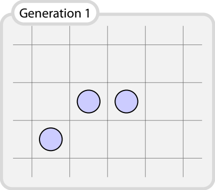
Das Spielbrett

- Das *Spielbrett* ist ein unendliches Gitter.
- In jedem Gitterquadrat kann sich eine *Zelle* befinden oder auch nicht.
- Am Anfang sind einige Gitterquadrate mit Zellen belegt. Sie bilden die *Anfangspopulation*.



Die Spielrunden

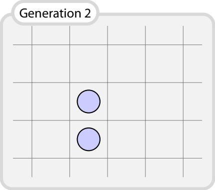
- Das Spiel verläuft in *Generationen* (Runden).
- In jeder Generation werden eventuell neue Zellen *geboren*, alte können *sterben* und Zellen können auch einfach *überleben*.
- Dafür sind die acht umliegenden Zellen wichtig, genannt ihre *Umgebung*.



Conways Game of Life.

Die Spielrunden

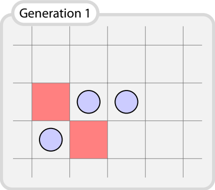
- Das Spiel verläuft in *Generationen* (Runden).
- In jeder Generation werden eventuell neue Zellen *geboren*, alte können *sterben* und Zellen können auch einfach *überleben*.
- Dafür sind die acht umliegenden Zellen wichtig, genannt ihre *Umgebung*.



Conways Game of Life.

Die Regeln

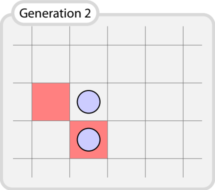
- Eine Zelle wird *geboren*, wenn es in ihrer Umgebung genau *3* Zellen gibt.
- *Eine Zelle überlebt, wenn es in ihrer Umgebung genau 2 oder 3 Zellen gibt.*
  Sonst stirbt sie an Vereinsamung oder Überbevölkerung.



Conways Game of Life.

Die Regeln

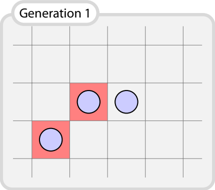
- Eine Zelle wird *geboren*, wenn es in ihrer Umgebung genau *3* Zellen gibt.
- Eine Zelle *überlebt*, wenn es in ihrer Umgebung genau *2* oder *3* Zellen gibt.
  Sonst *stirbt* sie an Vereinsamung oder Überbevölkerung.



Conways Game of Life.

Die Regeln

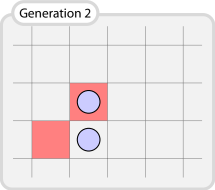
- Eine Zelle wird *geboren*, wenn es in ihrer Umgebung genau *3* Zellen gibt.
- Eine Zelle *überlebt*, wenn es in ihrer Umgebung genau *2* oder *3* Zellen gibt.
  Sonst *stirbt* sie an Vereinsamung oder Überbevölkerung.



Conways Game of Life.

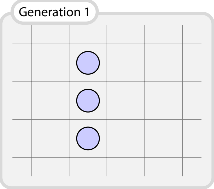
Die Regeln

- Eine Zelle wird *geboren*, wenn es in ihrer Umgebung genau *3* Zellen gibt.
- Eine Zelle *überlebt*, wenn es in ihrer Umgebung genau *2* oder *3* Zellen gibt.
  Sonst *stirbt* sie an Vereinsamung oder Überbevölkerung.

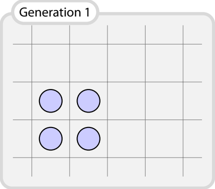


Ein paar interessante Anfangspopulationen

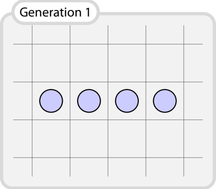
Blinker



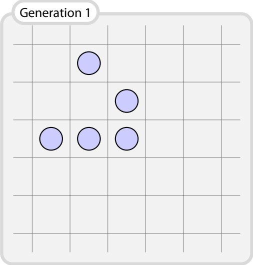
Block



Bienenwabe

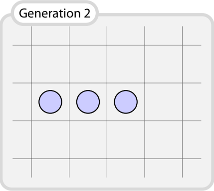


Gleiter

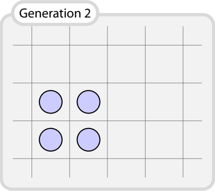


Ein paar interessante Anfangspopulationen

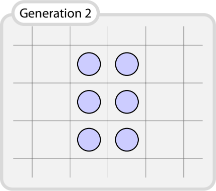
Blinker



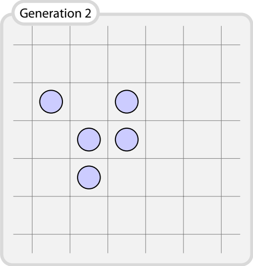
Block



Bienenwabe

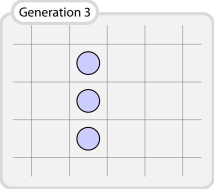


Gleiter

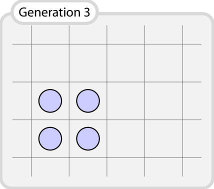


Ein paar interessante Anfangspopulationen

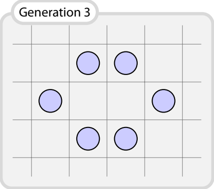
Blinker



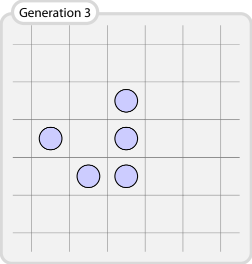
Block



Bienenwabe

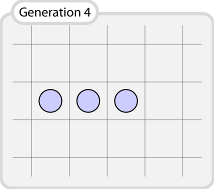


Gleiter

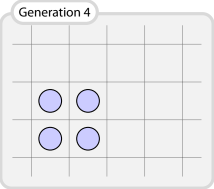


Ein paar interessante Anfangspopulationen

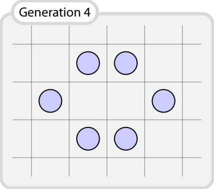
Blinker



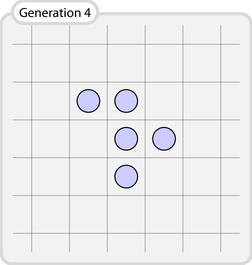
Block



Bienenwabe

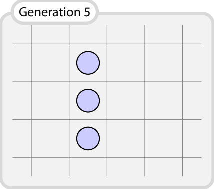


Gleiter

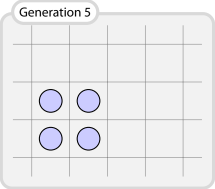


Ein paar interessante Anfangspopulationen

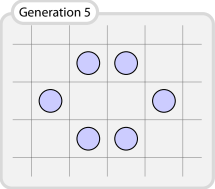
Blinker



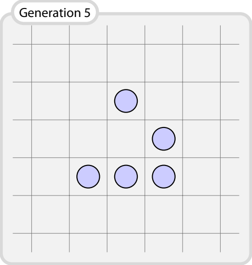
Block



Bienenwabe



Gleiter



Einfache Regeln können komplexe Effekte haben.

Behauptung

Das Game of Live kann genauso viel wie Turing-Maschinen.
(Man sagt, es sei *Turing-mächtig.*)

Etwas genauer:

Satz

Es gibt eine (einfache) Kodierung von Wörtern $w \in \{0,1\}^*$ als Life-Spielbretter, so dass für jede Turing-berechenbare Funktion f gilt: Wenn man das Spiel mit w startet, so wird nach endlich vielen Schritten eine Population erreicht, die gerade $f(w)$ kodiert.

Dies beweist man, indem man mit Gleitern, Gleiterkanonen, Reflektoren, Verknüpfern und allerlei weiteren Konstruktionen die Berechnung einer Turing-Maschine auf dem Spielbrett nachvollzieht.

Die Church-Turing-These

These

Die »intuitiv berechenbaren« Funktionen sind gerade Turing-berechenbaren.

- Diese These kann man *prinzipiell nicht beweisen*.
- Man kann aber versuchen, sie zu *widerlegen*.
- Dazu müsse man eine Funktion finden, »die man berechnen kann« (wie auch immer), die aber nicht von Turing-Maschinen berechnet werden kann.

Über das Postsche Korrespondenzproblem.

- Das *Postsche Korrespondenzproblem* (PKP) ist eine *Sprache*.
- Sie geht auf Emil Post zurück.
- Die Sprache ist recht einfach aufgebaut und *erscheint zunächst einfach zu berechnen*.
- Erstaunlicherweise ist sie aber *nicht berechenbar.*

Die Idee

- Die *Eingabe* für das PKP ist ein *Wörterbuch*:

Deutsch	Vogonisch
apfel	apf
birne	rnx
mus	elmus
nixe	qwertz

(Vogonisch, insbesondere in Gedichtform, ist nicht unbedingt für seine melodische Struktur bekannt.)

- Die Frage ist nun, ob man einen Satz in der einen Sprache finden kann, so dass die *wortwörtliche Übersetzung* in die andere Sprache *genau denselben Satz* wie in der ersten Sprache liefert.
- Beispielsweise wird aus »apfel mus« die Übersetzung »apf elmus«, also – wenn man die Leerzeichen ignoriert – in beiden Fällen »Apfelmus«.
- Man nennt dies eine *Korrespondenz*.

Ein erstaunliches Resultat.

Definition (Die Sprache PKP)

Das *Postsche Korrespondenzproblem (PKP)* enthält (die Codes von) allen Wörterbüchern, für die es eine Korrespondenz gibt.

Beispiel

`{(1,100000),(00,1),(0110,0)}` $\in$ *PKP*

Satz

PKP *ist nicht berechenbar.*

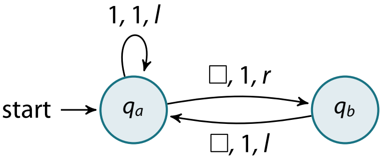
Über fleißige Biber.

Unknown author, Creative Commons Attribution Sharealike

- Ein *fleißiger Biber* ist ein Turing-Maschine.
- Er hat *möglichst wenige Zustände*,. . .
- . . . kennt nur die Symbole 1 und $\square$,. . .
- . . . startet auf einem leeren Band,. . .
- . . . schreibt möglichst viele 1en. . .
- . . . und hält dann aber irgendwann an.

Definition von fleißigen Bibern.

Beispiel (Fleißiger Biber mit zwei Zuständen)

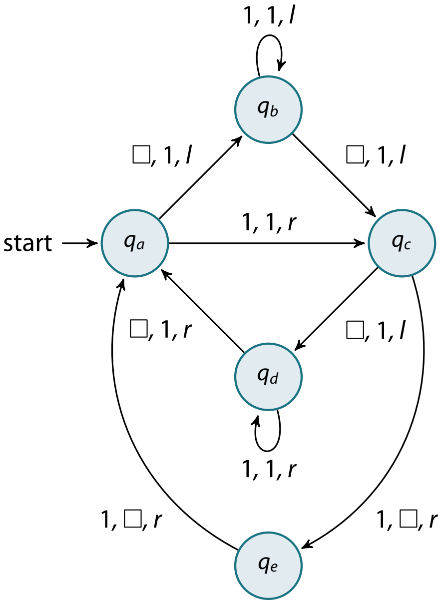


Definition (Fleißige-Biber-Funktion)

Die *Fleißige-Biber-Funktion* $busybeaver\colon \mathbb{N} \to \mathbb{N}$ ordnet jeder Zahl $n \geq 0$ die Anzahl an 1en zu, die ein fleißiger Biber mit n Zuständen schreibt.

Anzahl n an Zustände	$busybeaver(n)$
1	0
2	3
3	5
4	12
5	vermutlich 4097
6	$\geq 4{,}64 \cdot 10^{1439}$

Kandidat für einen fleißigen Biber mit fünf Zuständen.



- Dieser Biber macht 47.176.870 Schritte, bevor er anhält.
- Er hinterlässt 4097 viele 1en.
- Es ist nicht bewiesen, dass dies tatsächlich ein *fleißiger* Biber ist.

Die Fleißige-Biber-Funktion ist nicht berechenbar.

Satz

Die Fleißige-Biber-Funktion ist nicht berechenbar.

Über Komprimierer.

- Ein *Komprimierer* ist ein Programm.
- Es bekommt als *Eingabe einen String*.
- Es liefert als *Ausgabe einen String,*
  - der möglichst kurz ist und
  - aus dem sich die Eingabe rekonstruieren lässt.

Beispiel (Klassische Komprimierer)

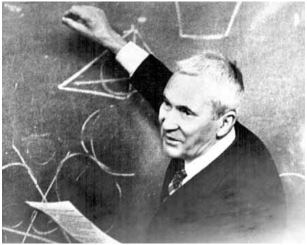
- Lempel-Ziv-Komprimierer (`gzip`)
- Huffmann-Komprimierer (`gzip`, `mpeg`)

Beispiel (Moderne Komprimierer)

Burrows-Wheeler-Transformation (`bzip2`)

Die Idee hinter dem Kolmogorov-Komprimierer

- Bei guten Komprimierern gibt es *generell viele Möglichkeiten, einen String zu kodieren*.
- Dadurch ist dann der Komprimierer besonders flexibel: Er kann die *»passendste« Art* suchen, einen String zu kodieren.
- Man kann sich nun fragen: *Was ist die ultimativ flexibelste Art, einen String zu kodieren?*
- Die Antwort lautet: *Als Programmtext, der diesen String als Ausgabe liefert.*



Public domain

Der Kolmogorov-Komprimierer

Definition (Kolmogorov-Komprimierer)

Der *Kolmogorov-Komprimierer* ist eine Funktion $K\colon \{0,1\}^* \to \mathrm{ASCII}^*$, so dass für alle Worte $w \in \{0,1\}^*$ gilt:

1. $K(w)$ ist ein Java-Programm,
2. das, wenn man es startet, w ausgibt, und
3. minimale Länge hat.

Der Kolmogorov-Komprimierer kann sehr stark komprimieren. . .

Manche Worte lassen sich sehr gut Kolmogorov-komprimieren:

Beispiel

Das Wort $1^{1000000}$ (also eine Million 1en) lässt sich mittels eines sehr kurzen Programms beschreiben:

```
for (int i=0; i<1000000; i++) System.out.print("1");
```

Beispiel

Das Wort $1^{10^{100}}$ (also ein Googol viele 1en) lässt sich auch kurz beschreiben:

```
int p = 1;
for (int i=0; i<100; i++) p = p * 10;
for (int i=0; i<p;   i++) System.out.print("1");
```

. . . aber nicht alles lässt sich komprimieren.

- Andererseits lassen sich *viele Worte gar nicht komprimieren*.
- Dies liegt daran, dass man ja sonst *rekursiv immer wieder komprimieren könnte* – irgendwann muss Schluss sein.
- Grob gesprochen lassen sich »zufällige Worte« nicht komprimieren:

Gut komprimierbar	schlecht komprimierbar
0000000000	0101101101
0000011111	1011111010
0101010101	0010110111

Ein trauriges Resultat.

Satz

Der Kolmogorov-Komprimierer ist nicht berechenbar.

Beweis.

Nehmen wir an, wir könnten K berechnen.

- Sei w das kürzeste und lexikographisch erste Wort, so dass die Länge des Programms $K(w)$ gerade eine Million ist.
- Dann kann man ein (kurzes) Programm schreiben, das in einer Schleife alle Wörter in der Reihenfolge λ, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, ... durchläuft.
- Für jedes Wort x berechnet es $K(x)$ und gibt das erste aus, für das $|K(x)| = 1000000$.
- Dieses Programm ist dann ein *sehr kurzes Programm, das w beschreibt* – aber das kürzeste solche Programm sollte ja Länge 1000000 haben. □

Zusammenfassung

Turing-Maschine

Die Turing-Maschine ist ein *extrem einfaches Modell von Rechnern,* das Alan Turing 1936 vorgeschlagen hat.

Turing-Berechenbarkeit

Eine Funktion heißt *Turing-berechenbar,* wenn eine Turing-Maschine zu jedem Eingabewort den Funktionswert nach endlich vielen Schritten auf ihrem Band zu stehen hat.

Church-Turing-These

Alles, was sich berechnen lässt, lässt sich von Turing-Maschinen berechnen.

Nichtberechenbare Probleme

Es gibt Funktionen, die nicht berechenbar sind. Beispiele sind das Postsche Korrespondenzproblem, die Fleißige-Biber-Funktion und der Kolmogorov-Kompressor.

Zusammenfassung

Paul Chapman.
Life Universal Computer, 2002.
http://www.igblan.free-online.co.uk/igblan/ca/, Zugriff April 2014

The LifeWiki.
http://www.conwaylife.com/wiki, Zugriff April 2014