Berechenbarkeit

Sind endliche Automaten das letzte Wort?

+: Ein Digitalrechner mit endlich viel Speicher ist ein endlicher Automat!

−: Viel Speicher ⇝ astronomisch viele Zustände, sehr komplexe Automaten

−: Wir wollen einfaches Maschinenmodell für Automaten die z.B. $a^n b^n c^n$ akzeptieren.

Welche Probleme lassen sich überhaupt algorithmisch lösen?

Berechenbarkeit Hauptergebnis

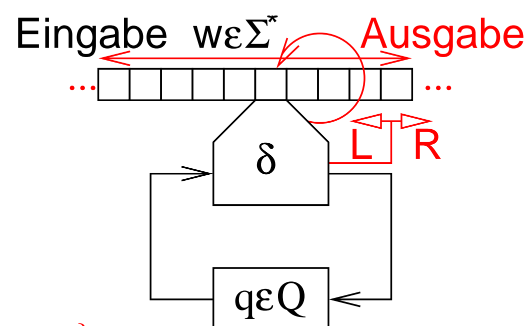
Sicherstes Rezept Nichtinformatiker zu vergraulen:
Eine hitzige Debatte über DIE richtige Programmiersprache

- Alle Programmiersprachen und Maschinenmodelle sind „gleich mächtig“
- In jedem Modell gibt es die gleichen nichtberechenbaren Probleme

Deterministische Einband-Turingmaschinen (DTM)

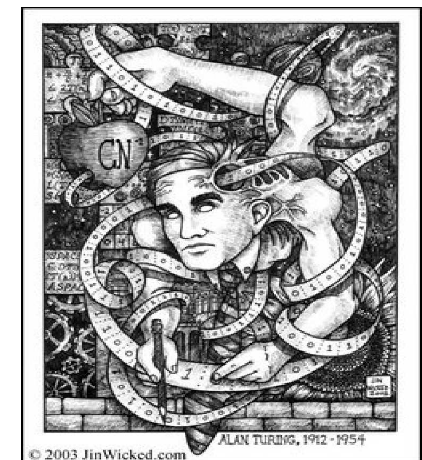
$T = (Q, \Sigma, \Gamma, \delta, s, F)$:

- Q, Zustände
- Σ, Eingabealphabet
- Γ Bandalphabet, $\sqcup \notin \Sigma$: Leersymbol, $\Sigma \cup \{\sqcup\} \subseteq \Gamma$
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, N\}$, Übergangsfunktion;
- $s \in Q$, Startzustand
- $F \subseteq Q$, Endzustände



Warum Turingmaschinen?

- Historisch der erste Ansatz [Turing 1936]
- Ursprüngliche Motivation: Reduktion der Arbeitsweise eines menschlichen Mathematikers beim Rechnen auf das Wesentliche
- Minimalistische Erweiterung eines endlichen Automaten
- Curchsche These: Alle „hinreichend mächtigen“ Maschinenmodelle sind äquivalent



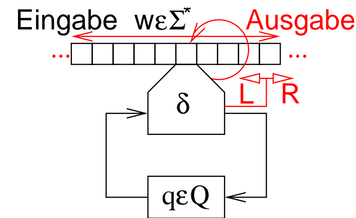
Ursprüngliche Motivation

Analog ⇝ diskrete Positionen, endliches Alphabet: begrenzte Genauigkeit

Stift auf Papier ⇝ Schreib-Lesekopf

Blatt Papier (2D) ⇝ Band (1D):

- vertikale Bewegungen ⇝
- größere horizontale Bewegungen,
- ggf. Zeilenendemarkierungen



Gehirn befolgt Rechenvorschriften ⇝ Endlicher Automat

Potentiell unendlicher Speicher?

+: Einzige Alternative zu monströsen endlichen Automaten

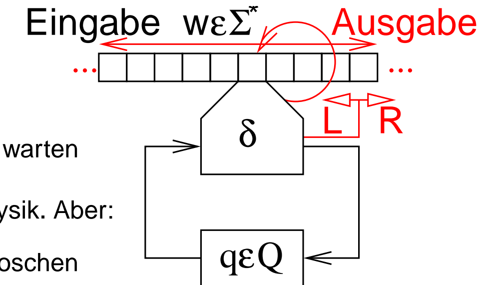
+: Blanks an Anfang und Ende nicht abspeichern

+: Wenn Platz ausgeht
mehr „Band" kaufen

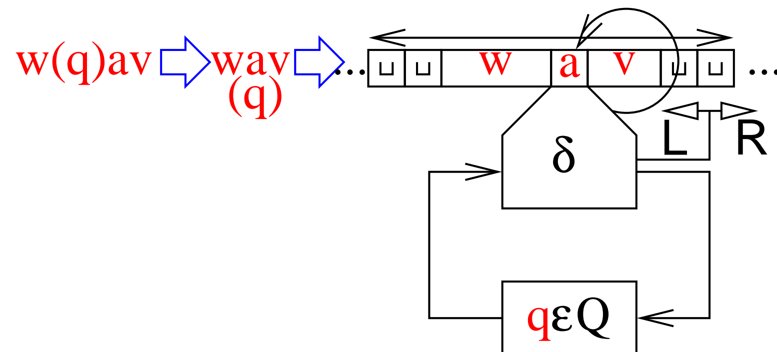
+: Wenn Band ausverkauft
auf nächste Technologiestufe warten

−: Irgendwann kriegt uns die Physik. Aber:

- +: Dann ist unsere Sonne erloschen
- +: Bis dahin ist das eine nützliche Abstraktion



Konfiguration einer TM



$w, v \in \Gamma^*, a \in \Gamma, q \in Q$

Funktionsweise

$$wa(q)bcv \xrightarrow{\delta(q,b)=(q',b',N)} wa(q')b'cv$$

$$wa(q)bcv \xrightarrow{\delta(q,b)=(q',b',L)} w(q')ab'cv$$

$$wa(q)bcv \xrightarrow{\delta(q,b)=(q',b',R)} wab'(q')cv$$

Übergangsfunktion δ hat drei Ausgaben:

- ☐ Neuer Zustand — wie bei EA
- ☐ Neues Bandsymbol — überschreibt altes Symbol an Kopfposition
- ☐ Bewegungsrichtung des Kopfes

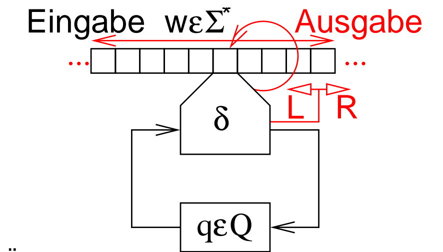
Wann hält ein TM?

T hält in Konfiguration $w(q)av$ gdw
$\delta(q,a) = (q,a,N)$.

Konvention:
$\forall q \in F : \forall a \in \Gamma : \delta(q,a) = (q,a,N)$

Turingmaschinen als Akzeptor

$T = (Q,\Sigma,\Gamma,\delta,s,F)$.
$L(T)$?



Definition:

T akzeptiert $w \in \Sigma^*$ gdw
T angesetzt auf $(s)w$
hält nach endlich vielen Zustandsübergängen
in einer Konfiguration $x(f)y$ mit $f \in F$.
(kurz: sie hält in f.)

$L(T) := \{w \in \Sigma^* : T \text{ akzeptiert } w\}$.

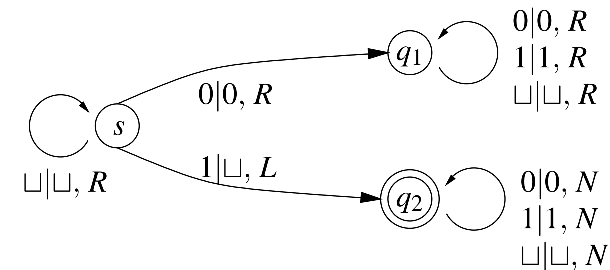
Formale Sprachen $L \subseteq \Sigma^*$ und Turingmaschinen $T = (Q,\Sigma,\Gamma,\delta,s,F)$

T akzeptiert L $\Leftrightarrow$
T akzeptiert alle $w \in L$ und kein $w \notin L$.

L rekursiv aufzählbar (oder semientscheidbar) $\Leftrightarrow$
$\exists T : T$ akzeptiert L.

L rekursiv (oder entscheidbar) $\Leftrightarrow$
$\exists T : T$ akzeptiert $L \wedge \forall w \in \Sigma^* : T$ hält

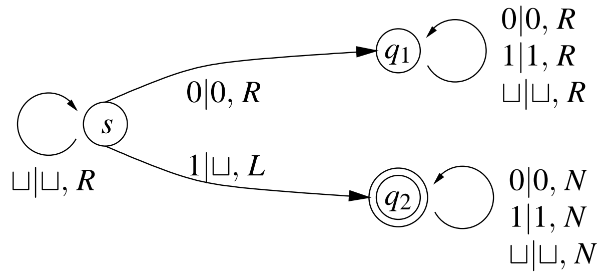
Beispiel: Akzeptor für $L = 1(0,1)^*$



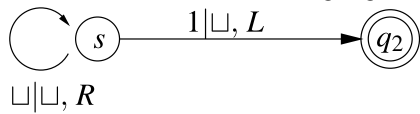
Grafische Notation:

Erweiterung von EA. Erweiterte Kantenbeschriftung:
Eingabezeichen|Ausgabezeichen,Bewegungsrichtung

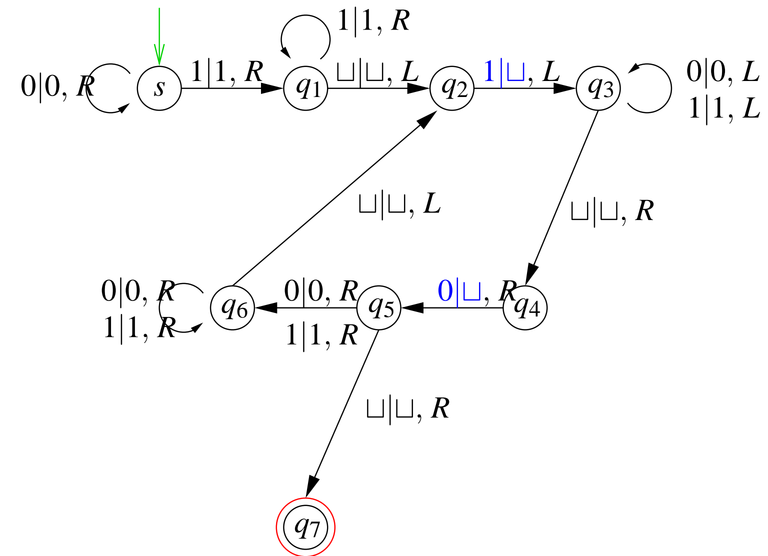
Vervollständigung



Konvention: Fehlender Übergang $\rightsquigarrow$ die TM hält.



Beispiel: Akzeptor für $\{0^n1^n : n \geq 1\}$

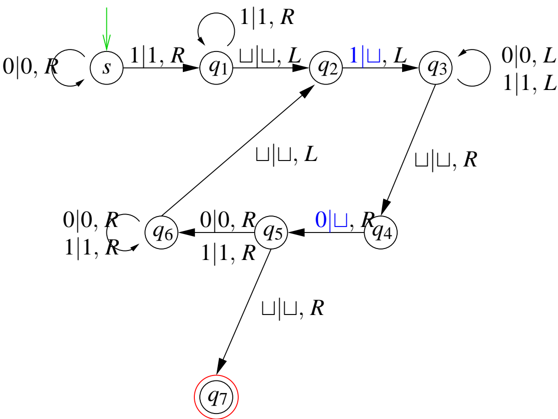


$(s)000111$
$0(s)00111$
$00(s)0111$
$000(s)111$
$0001(q_1)11$
$00011(q_1)1$
$000111(q_1)$
$00011(q_2)1$
$0001(q_3)1\sqcup$
$0001(q_3)1$
$000(q_3)11$
$00(q_3)011$
$0(q_3)0011$
$(q_3)00011$
$(q_3)\sqcup 00011$

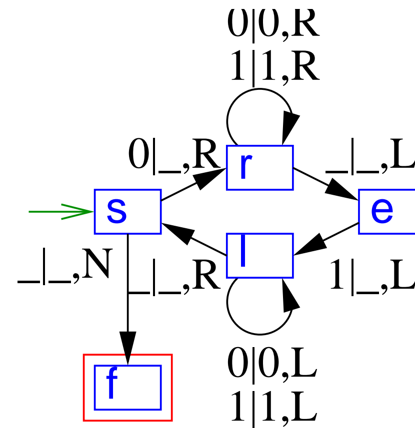
$(q_4)00011$
$\sqcup(q_5)0011$
$0(q_6)011$
$00(q_6)11$
$001(q_6)1$
$0011(q_6)$
$001(q_2)1$
$00(q_3)1\sqcup$
$0(q_3)01$
$(q_3)001$
$(q_3)\sqcup 001$
$(q_4)001$
$\sqcup(q_5)01$
$0(q_6)1$
$01(q_6)$

$0(q_2)1$
$(q_3)0\sqcup$
$(q_3)\sqcup 0$
$(q_4)0$
$\sqcup(q_5)$

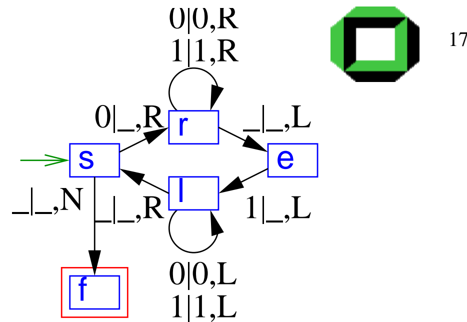
(q_7)



Beispiel: $\{0^n1^n : n \geq 0\}$.



Beispiel: $\{0^n1^n : n \geq 0\}$.



Sei $k \geq 1$, $w \in \{0,1\}^*$

ε: $(s) \to (f)$.

0: $(s)0 \to (r) \to (e)$ hält.

$1w$: $(s)1w$ hält.

$0w0$: $(s)0w0 \to (r)w0 \overset{|w|+1}{\to} w0(r) \to w(e)0$ hält.

$0w1$: $(s)0w1 \to (r)w1 \overset{|w|+1}{\to} w1(r) \to w(e)1 \overset{|w|+1}{\to} (\ell)\sqcup w \to (s)w$

0^n1^n: $(s)0^n1^n \Rightarrow (s)0^{n-1}1^{n-1} \Rightarrow \cdots \Rightarrow (s) \to (f)$

$0u1,\ u \notin \{0^n1^n : n \geq 0\}$: $(s)0u1 \Rightarrow (s)u$. Hält nicht in f. (Induktion)

Turingmaschinen berechnen Funktionen

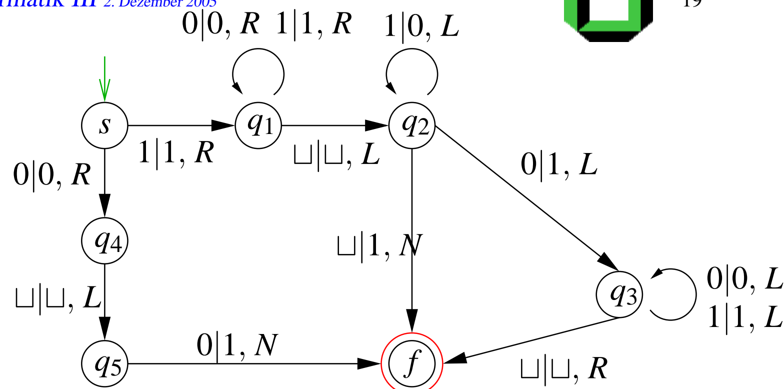
$T = (Q, \Sigma, \Gamma, \delta, s, F)$ realisiert die partielle Funktion $f_T : \Sigma^* \to \Gamma^*$:

$$f_T(w) := \begin{cases} v & \text{falls } T \text{ hält nach Eingabe von } w \text{ mit Ausgabe } v \\ & ((s)w \Rightarrow u(q)v) \\ \perp = \text{(undefiniert)} & \text{sonst} \end{cases}$$

g berechenbar (total rekursiv)$\Leftrightarrow \exists T : f_T = g$

Bemerkung: wenn $g(x) = \perp$ hält T nicht.

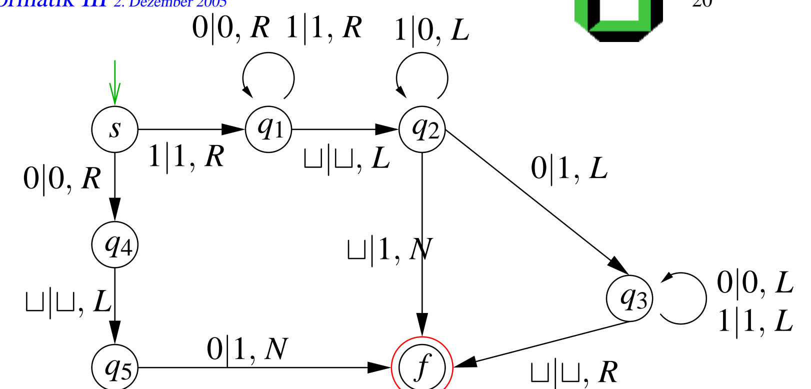
Beispiel



$$f(w) = \begin{cases} w+1 & \text{falls } w \in 0 \cup 1(0 \cup 1)^*, \\ & w \text{ interpretiert als Binärzahl} \\ \text{undefiniert} & \text{sonst} \end{cases}$$

Bemerkung: Nichteingezeichnete Übergänge gelten hier als Endlosschleife.

Beispiel

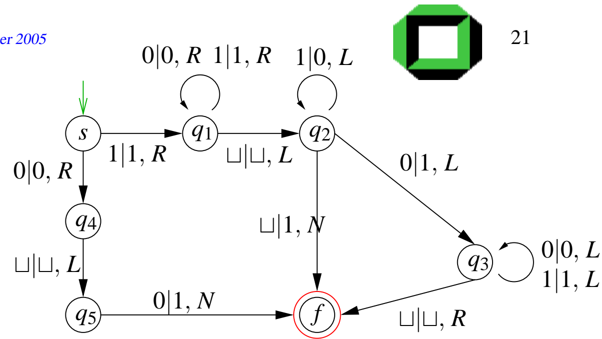


$(s)11 \to 1(q_1)1 \to 11(q_1) \to 1(q_2)1 \to (q_2)10 \to (q_2)\sqcup 00 \to (f)100$

Funktionsweise

Fallunterscheidung nach Aufbau der Eingabe.

Sei $w \in \{0,1\}^*$ beliebig, $a \in \{0,1\}, n \geq 1$.



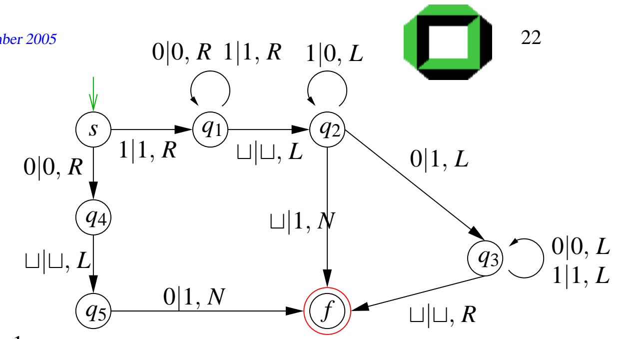
0: $(s)0 \rightarrow 0(q_4) \rightarrow (q_5)0 \rightarrow (f)1$

$0aw$: $(s)0aw \rightarrow 0(q_4)aw$ nichtdefiniert

Funktionsweise

Fallunterscheidung nach Aufbau der Eingabe.

Sei $w \in \{0,1\}^*$ beliebig, $a \in \{0,1\}, n \geq 1$.



1^n: $(s)1^n \rightarrow 1(q_1)1^{n-1} \xrightarrow{n-1} 1^n(q_1) \rightarrow 1^{n-1}(q_2)1 \xrightarrow{n} (q_2)\sqcup 0^n \rightarrow (f)10^n$

$1w0$: $(s)1w0 \rightarrow 1(q_1)w0 \xrightarrow{|w|+1} 1w0(q_1) \rightarrow 1w(q_2)0 \rightarrow 1w(q_3)1 \xrightarrow{|w|+1} (q_3)\sqcup 1w1 \rightarrow (f)1w1$

$1w01^n$: $(s)1w01^n \rightarrow 1(q_1)w01^n \xrightarrow{|w|+1+n} 1w01^n(q_1) \rightarrow 1w01^{n-1}(q_2)1 \xrightarrow{n} 1w(q_2)00^n \rightarrow 1w(q_3)10^n \xrightarrow{|w|+1} (q_3)\sqcup 1w10^n \rightarrow (f)1w10^n$

Programmiertechniken für Turingmaschinen

- Lokale Variablen
- Hintereinanderschalten
- Spuren
- While-Schleifen

Lokale Variablen

Lokale Variable $x \in A$ speichern, ($|A| < \infty$!):

$Q \rightsquigarrow Q \times A$

Hintereinanderschalten

Gegeben: $T = (Q, \Sigma, \Gamma, \delta, s, F)$

OBdA: $(s)w \Rightarrow (r) f_T(w)$ für ein $r \in F$ falls $f_T(w) \neq \perp$.,

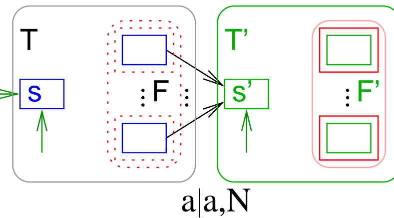
$T' = (Q', \Sigma, \Gamma', \delta', s', F')$.

Ausgabe: Turingmaschine $T^\circ = (Q^\circ, \Sigma, \Gamma^\circ, \delta^\circ, s, F')$ für $f_{T'}(f_T(x))$:

$Q^\circ = Q \dot\cup Q'$

$\Gamma^\circ = \Gamma \cup \Gamma'$

$$\delta^\circ(q,a) = \begin{cases} \delta(q,a) & \text{falls } q \in Q \setminus F \\ (s', a, N) & \text{falls } q \in F \\ \delta'(q,a) & \text{falls } q \in Q' \end{cases}$$



Spuren

$\Gamma = \Gamma_1 \times \cdots \times \Gamma_k$.

Beispiele: Arithmetische Operationen auf 2 Binärzahlen,

Markierungen...

Kleine Komplikation: Eingabealphabet ändert sich.

Auswege:

- $\Gamma = \Sigma \dot\cup \Gamma_1 \times \cdots \times \Gamma_k$
- Ersetze $a \in \Sigma$ durch $(a, 0, \ldots, 0)$ in der Eingabe.

While-Schleifen: While $i \neq 0$ Do tape:= f_T(tape)

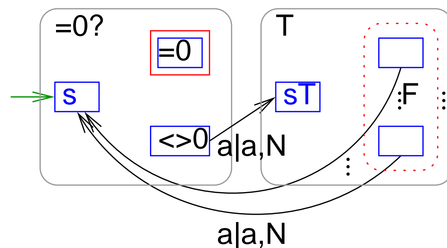
Spure i definiere einen Zähler (unär oder binär)

Unterprogramm: teste ob Spur $i = 0$.

Wenn ja: halt

Lasse T laufen

Zurück zum Startzustand. (Übergang $\delta(f, a) = (s, a, N)$)



Varianten von Turingmaschinen

k Köpfe: $\delta : Q \times \Gamma^k \to Q \times \Gamma^k \times \{L, R, N\}^k$

k Bänder: i.allg. ein Kopf pro Band

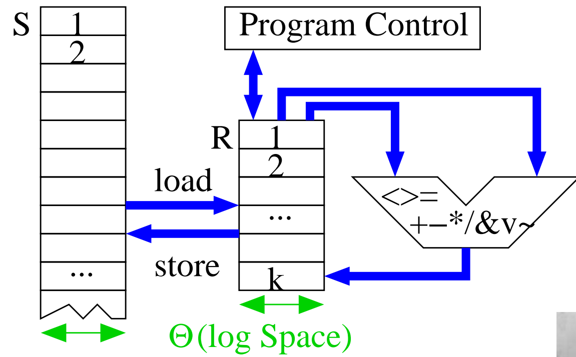
d-dimensionale Bänder: z.B. $d = 2$,

$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, U, D, N\}$$

probabilistisch: zusätzliches undirektionales Leseband mit Zufallsbits

nichtdeterministisch: $\delta : Q \times \Gamma \to 2^{Q \times \Gamma \times \{L,R,N\}}$ analog NEA, stay tuned ...

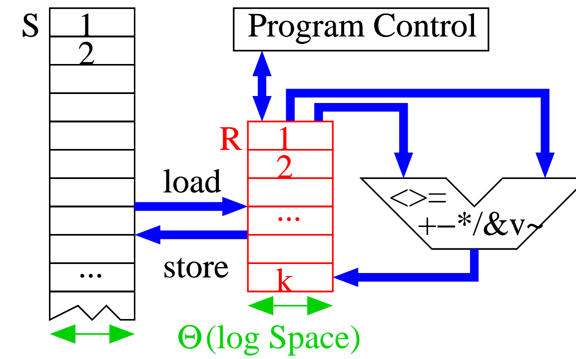
RAM: Random Access Machine



Moderne (RISC) Adapation des

von Neumann-Modells [von Neumann 1945]

Register

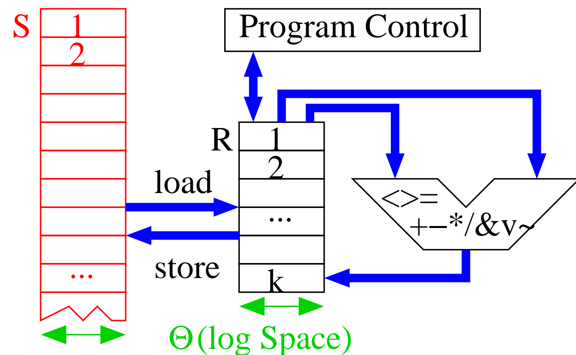


k (irgendeine Konstante) Speicher

$R_1, \ldots, R_k$ für

(kleine) ganze Zahlen

Hauptspeicher

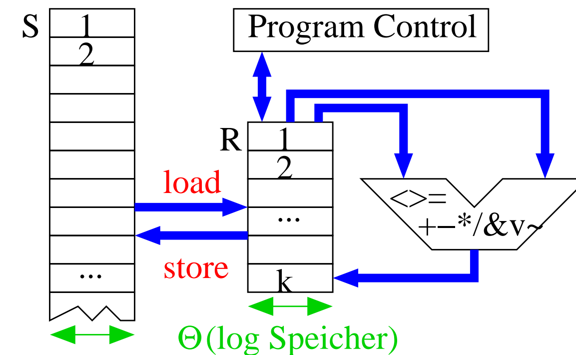


Unbegrenzter Vorrat an Speicherzellen

$S[1]$, $S[2] \ldots$ für

(kleine) ganze Zahlen

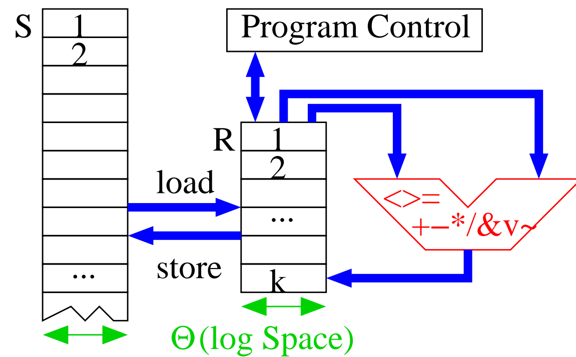
Speicherzugriff



$R_i := S[R_j]$ lädt Inhalt von Speicherzelle $S[R_j]$ in Register R_i.

$S[R_j] := R_i$ speichert Register R_i in Speicherzelle $S[R_j]$.

Rechnen

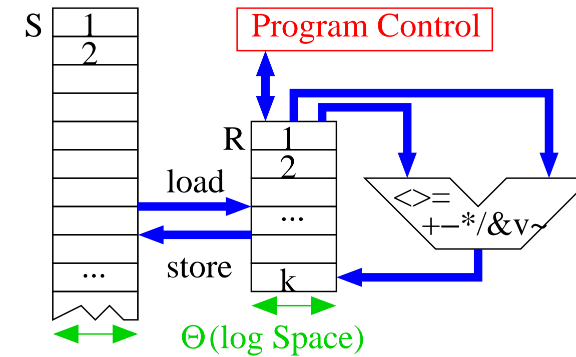


$R_i := R_j \odot R_\ell$ Registerarithmetik.

'$\odot$' ist Platzhalter für eine Vielzahl von Operationen

Arithmetik, Vergleich, Logik

Bedingte Sprünge



JZ j, R_i Setze Programmausführung an Stelle j fort falls $R_i = 0$

„Kleine" ganze Zahlen?

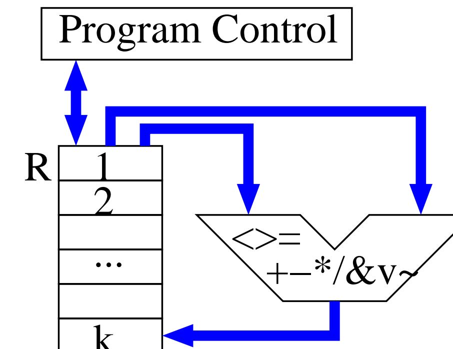
Alternativen:

Konstant viele Bits (64?): theoretisch unbefriedigend weil nur endlich viel Speicher adressierbar $\rightsquigarrow$ endlicher Automat

Beliebige Genauigkeit: viel zu optimistisch für vernünftige Komplexitätstheorie. Beispiel: n maliges quadrieren führt zu einer Zahl mit $\approx 2^n$ bits.
OK für Berechenbarkeit

Genug um alle benutzten Speicherstellen zu adressieren: Bester Kompromiss.

Registermaschine



ggf. eingeschränkt auf Inkrementieren, Dekrementieren

Beliebt in der Berechenbarkeit.

Führt zu merkwürdigen Algorithmen.

Höhere Programmiersprachen

Java, C/C++, Pascal,...

ML, Lisp,...

Prolog, Oz,...

...

sind das uns am meisten geläufige Programmiermodell.

Compiler übersetzen das routinemäßig in RAM Code.

While-Programm

Minimalistische Programmiersprache für Berechenbarkeitstheorie:

$\mathbb{N}$ main$(\mathbb{N}x_1,\ldots,\mathbb{N}x_k)\{$

$\quad\mathbb{N}\ x_0 = 0; \mathbb{N}\ v_1 = 0; \ldots; \mathbb{N}\ v_n = 0;$

$\quad$ body;

$\quad$ return x_0;

$\}$

body darf benutzen

Zuweisung: $x := y$

Increment, Decrement: $x{+}{+}$ oder $x{-}{-}$ (mit $0{-}{-} := 0$)

';': Sequenz von Anweisungen

while$(x \neq 0)$: Schleife

Church'sche These

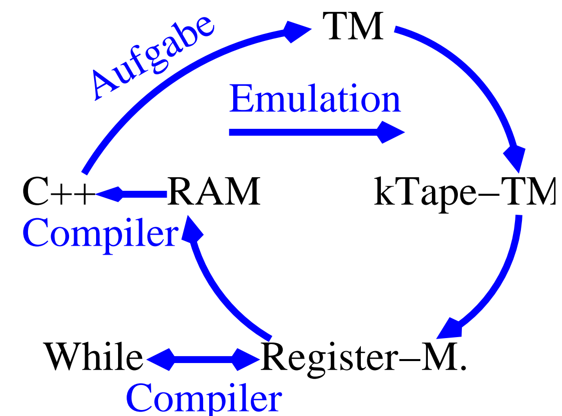
Von Turingmaschinen berechenbare Funktionen
sind genau die im intuitiven Sinne berechenbare Funktionen.

Kein Satz aber allgemein akzeptiert.

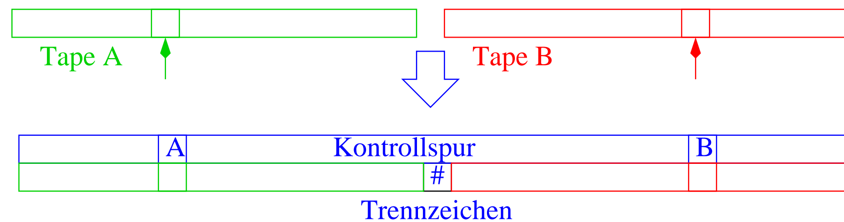
Begründung

- Alle bekannten Berechnungsmodelle selbst sind schwächer oder äquivalent.
  das kann man beweisen
- Keine „intuitiv" berechenbare Funktion bekannt, die nicht Turing-berechenbar ist.

Äquivalenz von Maschinenmodellen



Turingmaschine emuliert k-Band-TM



- Nichleere Bandteile aneinanderhängen (Trennzeichen benutzen)
- Kopfpositionen markieren
- Zustand speichert $k-1$ Bandsymbole

Satz: Zeit T mit k-Band-TM $\longrightarrow$ Zeit $O\left(T^2\right)$ auf Einband-TM.

k-Band-TM emuliert Registermaschine

- Ein Band pro Register (Binärformat oder Unärformat)
- Eigene Zustände für jede Programmzeile
- Unterprogramme für Arithmetik
- Zuweisung $\longrightarrow$ Band kopieren

Registermaschine emuliert RAM

Idee: zusätzliches Register R_S repräsentiert Speicher:

$$R_S = \sum_i S[i] \cdot 2^{bi}$$

mit b =Anzahl Bits der RAM

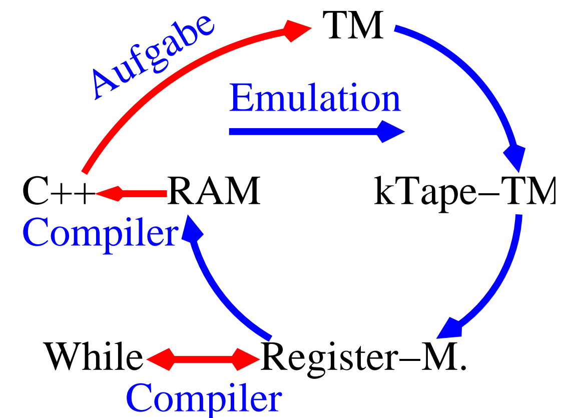
$S[i]$ in R_j laden:

$$R_j := \frac{R_S}{2^{bi}} \bmod 2^b \quad .$$

R_j in $S[i]$ speichern:

$$R_S := R_S + R_j \cdot 2^{bi} - \left(\frac{R_S}{2^{bi}} \bmod 2^b\right) 2^{bi}$$

Äquivalenz von Maschinenmodellen



Markov-Algorithmen

deterministisches regelbasiertes String-Rewriting.

Geg: Eingabe $w \in \Sigma^*$

Menge von Regeln $\Delta \in (\Gamma^* \times \Gamma^*)^*$

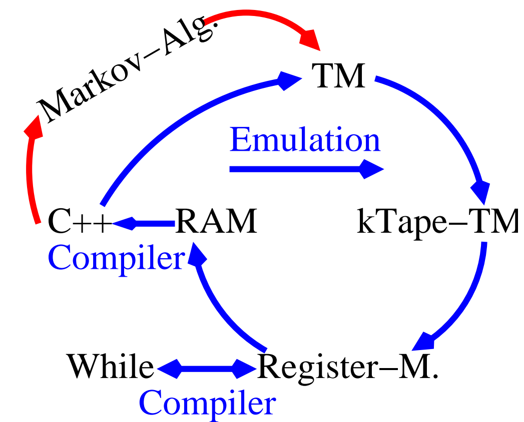
while $\exists(\ell, r) \in \Delta, u, v \in \Gamma^* : w = u\ell v$ **do**

find the first rule $(\ell, r) \in R$

and the shortest $u \in \Gamma^*$ such that $w = u\ell v$ for some $v \in \Gamma^*$

$w := urv$

Markov-Algorithmen: Turingmächtigkeit



Markov-Algorithmen: Turingmächtigkeit

Gegeben: TM $M = (Q, \Sigma, \Gamma, \delta, s, F)$ mit Eingabe w.

OBdA: max 1 $\sqcup$ links und rechts der Einbabe wird angeschaut.

Betrachte Markovalgorithmus für Alphabet $Q \cup \Gamma \cup \{(,)\}$.

$\Delta = \cdots$ Sonderregeln für den Rand

$\circ \langle ((q)a, (q')a') : \delta(q, a) = (q', a', N) \rangle$

$\circ \langle (c(q)a, (q')ca') : \delta(q, a) = (q', a', L) \rangle$

$\circ \langle ((q)ac, a'(q')c) : \delta(q, a) = (q', a', R) \rangle$

Eingabe des Markovalgorithmus: $\sqcup(s)w\sqcup$

Die Folge der produzierten Strings ist gerade die Folge der Konfigurationen der Turingmaschine!

Semi-Thue-Systeme

Sowas wie nichtdeterministische Markov-Algorithmen.

Ebenfalls turingmächtig.

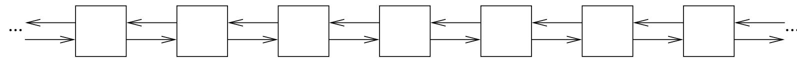
Unsere TM-Simulation hat immer genau eine anwendbare Regel.

Zellularautomaten

Betrachte den endlichen Automaten $(\{0,1\},\{0,1\}^2,\delta,\emptyset)$ mit

Q	0	1	0	1	0	1	0	1
$L\times R$	$(0,0)$	$(0,0)$	$(0,1)$	$(0,1)$	$(1,0)$	$(1,0)$	$(1,1)$	$(1,1)$
δ	0	1	1	1	0	1	1	0

Verbinde unendlich viele solcher Automaten zu einer Kette



[M. Cook 2002]: Diese Maschine ist Turing-mächtig.
siehe auch Wikipedia „rule 110 cellular automaton“

Quantencomputer

- Ein Qubit speichert Superpositionen von 0 und 1.
- Berechnungen mit n Qubits berechnen ein Superposition von 2^n klassischen Berechnungen
- Messungen erhalten bestimmte Informationen über all diese Berecnungen bestimmen
- Quantencomputer können in polynomieller Zeit faktorisieren und diskrete Logarithmen bestimmen
- Dies würde viele kryptografische Algorithmen kompromittieren

Quantencomputer: Berechenbarkeit und Komplexitätstheorie

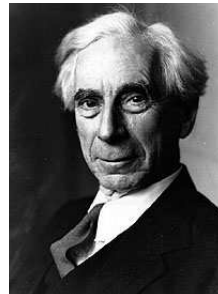
- Vermutung der Komplexitätstheorie:
  - Faktorisieren, DLog sind nicht in P (selbst mit Randomisierung)
  - Faktorisieren, DLog sind nicht NP hart.
- Turingmaschinen können Quantencomputer simulieren

Fazit: Quantencomputer wären schneller aber nicht mächtiger als klassische Computer

Unentscheidbare Probleme

- Gödelnummerierung: TMs könen sich selbst als Eingabe verarbeiten
- Wichtiges Beispiel: Universelle TM
- Diagonalisierungsargument: definiere uentscheidbare Sprache
- Reduktionen: Zeige, dass auch nützliche Probleme unentscheidbar sind

Paradoxien und Selbstbezüglichkeit



Der Barbier von Hintertupfingen
rasiert genau die Männer im Dorf,
die sich nicht selbst rasieren.

Wer rasiert den Barbier?

Paradoxien und Selbstbezüglichkeit

Daniel Düsentrieb behauptet, eine allwissende Maschine erfunden zu haben.

Ja Nein

Man stellt eine Ja/Nein-Frage und die Antwort leuchtet auf.

Dagobert Duck kauft die Maschine.
Will aber nur bei korrekter Funktion zahlen.
Er stellt der Maschine die Frage:
Wirst Du mit Nein antworten?

Was passiert?

Normierung von Turing-Maschinen

Betrachte $T = (Q, \Sigma, \Gamma, \delta, s, F)$. OBdA:

- $Q = \{1, \ldots, n\}$
- $\Sigma = \{0.1\}$
- $\Gamma = \{0, 1, \sqcup\}$, $\sqcup = 2$
- $s = 1$
- $F = \{2\}$

für geeignete Konstante n

Gödelnummer $\langle M \rangle$ einer Turingmaschine M

Definiere folgende Zeichenketten über $\{0, 1\}$:
Kodiere $\delta(q, a) = (r, b, d)$ durch $0^q\,1\,0^{a+1}\,1\,0^r\,1\,0^{b+1}\,1\,0^d$
wobei $N = 1, L = 2, R = 3$ für die Richtungscodierung d gewählt wird.

Die Turing-Maschine wird dann kodiert durch die Binärzahl:

$$111\text{code}_1 11\text{code}_2 11 \ldots 11\text{code}_z 111,$$

wobei code_i für $i = 1, \ldots, z$ alle Funktionswerte von δ in beliebiger Reihenfolge beschreibt.

Konvention:
n ist nicht Gödelnummer einer TM,
$\rightarrow n$ beschreibt eine TM, die $\emptyset$ akzeptiert

Gödelnummer

Beobachtung

Die Gödelnummerrierung beschreibt eine

injektive Abbildung von normierten TMs auf natürliche Zahlen

Beispiel

Sei $M = (\{1,2,3\},\{0,1\},\{0,1,\sqcup\},\delta,1,\{2\})$, mit

$\delta(1,1) = (3,0,R)$

$\delta(3,0) = (1,1,R)$

$\delta(3,1) = (2,0,R)$

$\delta(3,\sqcup) = (3,1,L)$

$\langle M \rangle$ ist dann:

111010010001010011000101010010011000100100101001
1000100010001001010111

Diagonalsprache L_d

Sei M_i die TM mit $\langle M_i \rangle = i$.

Sei w_i die Binärrepräsentation von i.

$L_d := \{w_i : M_i \text{ akzeptiert } w_i \text{ nicht}\}$

Satz: L_d ist unentscheidbar

Beweis:

Annahme:

$L_d = \{w_i : M_i \text{ akzeptiert } w_i \text{ nicht}\}$ ist entscheidbar.

$\stackrel{\text{Def. „entscheidbar“}}{\longrightarrow} \exists M_i : M_i$ akzeptiert L_d und hält stets.

Was macht M_i mit w_i?

$w_i \in L_d \stackrel{\text{Def. } M_i}{\longrightarrow} w_i$ wird akzeptiert. $\stackrel{\text{Def. } L_d}{\longrightarrow} w_i \notin L_d$

$w_i \notin L_d \stackrel{\text{Def. } M_i}{\longrightarrow} w_i$ wird nicht akzeptiert. $\stackrel{\text{Def. } L_d}{\longrightarrow} w_i \in L_d$

Beides führt zu einem Widerspruch.

Korollar:

$\bar{L}_d = \{w_i : M_i$ akzeptiert $w_i\}$ ist unentscheidbar

Annahme: $\bar{L}_d$ ist entscheidbar.

$\longrightarrow \exists M : M$ akzeptiert $\bar{L}_d$

modifiziere $M \rightsquigarrow M'$ so, dass M' L_d akzeptiert

(Austausch akzeptierende/nichtakzeptierende Haltezustände).

Widerspruch.

Universelle Turingmaschine

$U = (Q_u, \{0,1\}, \{0,1,\sqcup\}, \delta_u, s_u, F_u)$

Eingabe: $\langle M \rangle w$

M ist die zu simulierende TM, w ist die binär codierte Eingabe.

U simuliert M auf w.

D.h. U akzeptiert $\langle M \rangle w$ gdw M akzeptiert w

Universelle Turingmaschine

3 Bänder:

1. $\langle M \rangle$
2. Zustand q_M von M unär codiert
3. Bandinhalt w von M

Universelle Turingmaschine

```
if Präfix v von w repräsentiert eine TM then            // 111Tupel111
    verschiebe v auf Band ⟨M⟩
    q_M := 1                                            // Startzustand von M
    while q_M ≠ 2 do                                    // Endzustand von M
        laufe zum Anfang von ⟨M⟩
        foreach (q,a,r,b,d) ∈ ⟨M⟩ do                    // Feld für Feld
            if q = q_M then                             // Vergleich mit Band q_M
                if Eingabezeichen von Band 3= a then
                    q_M := r                            // auf Zustandsband kopieren
                    b auf Band 3 ausgeben
                    Bewegung von Band 3 entsprechend d ausführen
```

Universelle Turingmaschine: 3Band→1Band

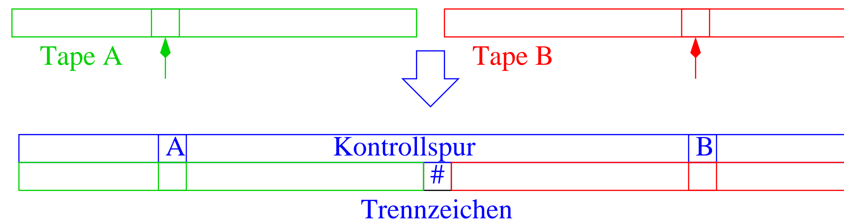
Eigentlich wissen wir wie das geht.

Problem: Bandalphabet unabhängig von M aber $> \{0,1\}$

Codiere Bandalphabet durch konst. viele $\{0,1\}$.

Problem: Eingabe muss auch codiert werden.

Das erledigt eine vorgeschaltete CodierungsTM.



Halteproblem

$H := \{w_i v : M_i$ angesetzt auf v hält$\}$

Satz: H ist nicht entscheidbar.

Beweis: angenommen H sei entscheidbar.

Wir konstruieren eine TM, die $\bar{L}_d$ akzeptiert.

$w_i \in \bar{L}_d$?

$\Leftrightarrow M_i$ akzeptiert w_i.

$\Leftrightarrow w_i w_i \in H \wedge M_i$ akzeptiert w_i.

Dies könnten wir mit Hilfe einer TM für H und einer universellen TM leicht ausrechnen.

Widerspruch.

Das beschränkte Halteproblem

Satz:

$\{w_i v \# w_j : M_i$ angesetzt auf v hält nach höchstens j Schritten$\}$

ist entscheidbar.

Beweisskizze:

Lasse universelle TM U angesetzt auf $w_i v$

für j simulierte Schritte laufen.

Mehr unentscheidbare Probleme

Gegeben Turingmaschinen T, T'

- $L(T) = \emptyset$? — Leerheit
- $|L(T)| = \infty$? — Unendlichkeit
- $L(T) = \Sigma^*$? — Vollständigkeit
- $L(T) = L(T')$? — Äquivalenz

Unentscheidbarkeit von Leerheit

Angenommen M akzeptiert $\{i : L(M_i) = \emptyset\}$

Wir zeigen, dass $\bar{L}_d$ dann entscheidbar wäre.

$w_i \in \bar{L}_d = \{w_i : M_i$ akzeptiert $w_i\}$?

Konstruiere eine Turingmaschine $T(i)$:

> erase input
> run M_i on w_i
> **if** state$(M_i) \neq 2$ **then** endless loop

Nun ist $L(T(i)) \neq \emptyset$ gdw $w_i \in \bar{L}_d$.

Also wäre $\bar{L}_d$ entscheidbar.

Widerspruch

Unentscheidbarkeit von Vollständigkeit

$L(T) = \Sigma^*$?

Gleicher Beweis wie bei Leerheit! Da $T(i)$ seine Eingabe ignoriert!

Metaprogrammierung

Der Beweis von Leerheit nimmt ein Programm und transformiert es in ein anderes.

Wichtige Programmiertechnik.

Postsches Korrespondenzproblem (PKP)

Geg: endliche Folge von Wortpaaren:

$$K = (x_1, y_1) \cdots (x_n, y_n) \in (\Sigma^+ \times \Sigma^+)^*$$

Frage:

$$\exists i_1, \ldots, i_k \in \{1, \ldots, n\} : x_{i_1} \ldots x_{i_k} = y_{i_1} \ldots y_{i_k}$$

?

Beispiel

- $K = ((1,111),(10111,10),(10,0))$ hat die Lösung $(2,1,1,3)$, denn es gilt:

  $$x_2 x_1 x_1 x_3 = 101111110 = 101111110 = y_2 y_1 y_1 y_3$$

- $K = ((10,101),(011,11),(101,011))$ hat keine Lösung: $(133\cdots)$
- $K = ((001,0),(01,011),(01,101),(10,001))$ hat eine Lösung der Länge 66: 243444212434344344214421341134442121113434121442141134113113121 4113

PCP ist semientscheidbar

Algorithmus:

Procedure PCP$((x_1,y_1)\cdots(x_n,y_n))$
  for $k := 1$ **to** ∞ **do**
    foreach $i_1\cdots i_k \in \{1..n\}^k$ **do**
      if $x_{i_1}\cdots x_{i_k} = y_{i_1}\cdots y_{i_k}$ **then**
        output $i_1\cdots i_k$
        return

PCP ist unentscheidbar

Beweis siehe Schöning.

Idee: angenommen lösbar $\longrightarrow$ Halteproblem lösbar

$x_{i_1}\ldots x_{i_k} = y_{i_1}\ldots y_{i_k} = (s)w\#\cdots\#u(f)v$

beschreibt akzeptierende Folge von TM-Konfigurationen

Hilberts 10. Problem — Diophantische Gleichungen

Gegeben:

multivariates Polynom p

mit ganzzahligen Koeffizienten.

Frage [Hilbert 1900]:

$$\exists x_1,\ldots,x_n \in \mathbb{Z} : p(x_1,\ldots,x_n) = 0?$$

[Matiyasevich 1970]: Das Problem ist unentscheidbar.

Abgeschlossenheit entscheidbarer Sprachen

abgeschlossen unter

$\cup$

$\cap$

$\bar{\cdot}$

Abgeschlossenheit semientscheidbarer Sprachen

abgeschlossen unter

$\cup$

$\cap$

nicht abgeschlossen unter

$\bar{\cdot}$

Abgeschlossenheit semientscheidbarer Sprachen unter Vereinigung

Seien M_1 und M_2 Akzeptoren für L_1 bzw. L_2

Akzeptor für $L_1 \cup L_2$:

for $j := 1$ **to** ∞ **do**

  if M_1 accepts w after j steps **then** accept

  if M_2 accepts w after j steps **then** accept

Nichtabgeschlossenheit semientscheidbarer Sprachen unter Komplementbildung

Annahme: Abgeschlossenheit gilt doch.

Sei M Akzeptor für L_d, $\bar{M}$ Akzeptor für $\bar{L}_d$

Function isInLd(w)

  for $j := 1$ **to** ∞ **do**

    if M accepts w after j steps **then return** true

    if $\bar{M}$ accepts w after j steps **then return** false

Eine von beiden hält.

$\longrightarrow L_d$ entscheidbar.

Widerspruch

Anwendung der nebenläufigen Ausführung

Mehrere Algorithmen $A_1, \ldots, A_k$, die ein schwieriges Problem lösen (langsam, schnell, nie).

Führe alle Algoirthmen (pseudo)gleichzeitig aus.

− Wenn alle gleich schnell haben wir Faktor k Rechenaufwand verschwendet

+ Wenn eins nie fertig wird haben wir unendlich viel gewonnen

+ Bei sehr unterschiedlicher Ausführungszeit können wir im Mittel gewinnen.

+ Wir können parallele Prozessoren verwenden.

+ Oft können wir einen Teil der redundanten Arbeit einsparen

Nebenläufige Ausführung

Anwendungen: Theorembeweiser, Programm/Hardware-Verifizierer, schwierige Planungs- und Optimierungsprobleme

Beispiel: Erfüllbarkeit aussagenlogischer Formeln in Zeit $O\left((4/3)^{\text{\#Variablen}}\right)$.

[U. Schöningh, A Probabilistic Algorithm for k-SAT and Constraint Satisfaction Problems, FOCS, 1999]