

Inhaltsverzeichnis

1	Berechenbarkeitstheorie	2
1.1	Intuitiver Berechenbarkeitsbegriff und Churchsche These	2
1.2	Churchsche These	5
1.3	Turing-Berechenbarkeit	6
1.4	LOOP-, WHILE- und GOTO-Berechenbarkeit	14
1.5	Primitiv rekursive und μ -rekursive Funktionen	21
1.6	Die Ackermann-Funktion	24
1.7	Entscheidbarkeit und Semi-Entscheidbarkeit	29
1.8	Das Halte-Problem und die Reduzierbarkeit	33
1.9	Das Postsche Korrespondenz-Problem	38
1.10	Der Gödelsche Unvollständigkeitssatz	44
2	Komplexitätstheorie	50
2.1	Komplexitätsmaße und Komplexitätsklassen	50
2.2	Die Komplexitätsklassen P und NP	52
2.3	NP -Vollständigkeit	55
2.4	Weitere NP -vollständige Probleme	60
2.4.1	3SAT	60
2.4.2	CLIQUE	62
2.4.3	HAMILTON-KREIS	63

1 Berechenbarkeitstheorie

1.1 Intuitiver Berechenbarkeitsbegriff und Churchsche These

- Es gibt eine intuitive Vorstellung, welche Funktionen (auf den natürlichen Zahlen) berechenbar sind.
- Auf Basis dieser intuitiven Vorstellung können allerdings keine Beweise geführt werden, daß eine bestimmte Funktion nicht berechenbar ist.
- Deshalb ist eine formale mathematische Definition der Berechenbarkeit notwendig.
- Der Nachweis der Nichtberechenbarkeit einer Funktion besteht dann darin nachzuweisen, daß die betrachtete Funktion nicht der Definition entspricht.
- neues Problem: Begründung, daß die formale Definition genau den intuitiven Berechenbarkeitsbegriff erfaßt.
Dazu ist kein formaler Beweis führbar, denn der intuitive Berechenbarkeitsbegriff ist nicht formal erfaßt.

Beispiele für Funktionen, die intuitiv berechenbar sind:

Eine Funktion $f : \mathbb{N}^k \rightarrow \mathbb{N}$ ist berechenbar, falls es einen Algorithmus, also ein mechanisches Rechenverfahren (z.B. MODULA-Programm) gibt, das f berechnet, also bei Eingabe von $(n_1, \dots, n_k) \in \mathbb{N}^k$ nach endlich vielen Schritten mit der Ausgabe $f(n_1, \dots, n_k)$ stoppt.

Eine partielle Funktion $f : \mathbb{N}^k \rightarrow \mathbb{N}$ ist berechenbar, falls der Algorithmus bei Eingabe von $(n_1, \dots, n_k) \notin \text{Def}(f)$ nicht stoppt.

1. Der Algorithmus:

```
INPUT(n);
REPEAT UNTIL FALSE;
```

“berechnet” die total undefinierte Funktion $\Omega : n \rightarrow \text{undef}$.

2. Die Funktion

$$f_\pi(n) = \begin{cases} 1 & \text{falls } n \text{ Anfangsabschnitt der} \\ & \text{Dezimalbruchentwicklung von } \pi \text{ ist} \\ 0 & \text{sonst} \end{cases}$$

ist berechenbar, denn es gibt ein beliebig genaues Näherungsverfahren für die Berechnung von π .

3. Ob die Funktion

$$g(n) = \begin{cases} 1 & \text{falls } n \text{ irgendwo in der Dezimalbruchentwicklung} \\ & \text{von } \pi \text{ vorkommt} \\ 0 & \text{sonst} \end{cases}$$

berechenbar ist, ist bisher unklar.

4. Die Funktion

$$h(n) = \begin{cases} 1 & \text{falls in der Dezimalbruchentwicklung von } \pi \\ & \text{irgendwo mindestens } n\text{-mal hintereinander} \\ & \text{eine 0 vorkommt} \\ 0 & \text{sonst} \end{cases}$$

ist berechenbar!

1.Fall: In π kommen beliebig lange 0-Folgen vor.

$$\Rightarrow h(n) = 1 \text{ für alle } n$$

2.Fall: In π kommen 0-Folgen der Länge n_0 ,
aber nicht der Länge $\geq n_0 + 1$ vor.

$$\Rightarrow h(n) = \begin{cases} 1 & \text{falls } n \leq n_0 \\ 0 & \text{sonst} \end{cases}$$

Es tritt entweder Fall 1 oder Fall 2 ein.

Achtung: Das Verfahren ist nicht konstruktiv;
trotzdem existiert ein Algorithmus.

Beispiel 2 zeigt, daß f_π berechenbar ist.
(Da ein Näherungsverfahren existiert.)
Auch f_e ist berechenbar.

Behauptung.

Es gibt (viele) $r \in \mathbb{R}$, für die f_r nicht berechenbar ist.

Beweis.

Es gibt überabzählbar viele $r \in \mathbb{R}$, aber nur abzählbar viele Rechenverfahren:
Ein Rechenverfahren muß sich durch einen endlichen Text beschreiben lassen
und es gibt höchstens abzählbar viele endliche Texte. ■

Bemerkung

Berechenbarkeit von f_r hat nichts mit rational/irrational zu tun!
Aber es gilt: für alle rationalen Zahlen $q \in \mathbb{Q}$ ist f_q berechenbar.

1.2 Churchsche These

Es gibt verschiedene Vorschläge, den intuitiven Berechenbarkeitsbegriff formal zu erfassen:

- Turing-Maschinen (Turing 1936)
- WHILE-Programme
- GOTO-Programme
- μ -rekursive Funktionen

Interessanter Weise kann bewiesen werden, daß alle diese Definitionen äquivalent sind. Es gibt bisher keinen Vorschlag, der nicht äquivalent wäre.

⇒ **Churchsche These:**

Die durch den formalen Begriff der Turing-Berechenbarkeit (äquivalent: WHILE-Berechenbarkeit, GOTO-Berechenbarkeit, μ -Rekursivität) erfaßte Klasse von Funktionen stimmt genau mit der Klasse der intuitiv berechenbaren Funktionen überein.

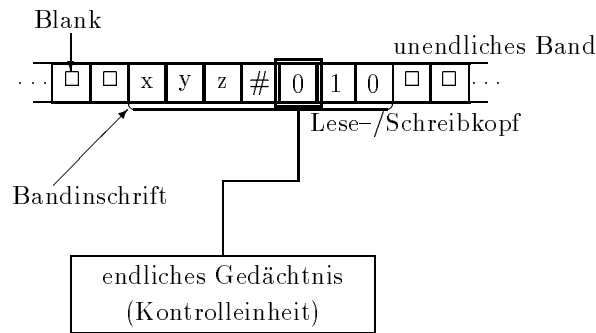
Die Churchsche These ist nicht beweisbar, da der intuitive Begriff der Berechenbarkeit nicht formal faßbar ist.

Sie ist allgemein akzeptiert.

Man kann auf Basis der Churchschen These zeigen, daß bestimmte Funktionen nicht berechenbar sind, indem man lediglich zeigt, daß keine TM zu ihrer Berechnung existieren kann.

1.3 Turing-Berechenbarkeit

Turings Vorschlag war, den Berechenbarkeitsbegriff mit Hilfe einer sehr einfachen Rechenmaschine (heute Turing-Maschine genannt) zu erfassen.



Definition.

Eine (Nichtdeterministische) Turingmaschine (kurz TM) ist ein 7-Tupel

$$M = (Q, \Sigma, \Gamma, \delta, q_0, \square, E)$$

mit

Q endliche Menge von "Zuständen"

Σ endliche Menge von "Eingabesymbolen" (Eingabealphabet)

$\Gamma \supset \Sigma$ endliche Menge von "Bandsymbolen" (Arbeitsalphabet)

$$\left. \begin{array}{l} \delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, N\} \text{ "deterministische"} \\ \delta : Q \times \Gamma \rightarrow 2^{Q \times \Gamma \times \{L, R, N\}} \text{ "nichtdeterministische"} \end{array} \right\} \text{Überföhrungsfunktion}$$

$q_0 \in Q$ "Startzustand"

$\square \in \Gamma - \Sigma$ "Blank" (leeres Bandsymbol)

$E \subseteq Q$ endliche Menge von "Endzuständen"

Informal bedeutet

$$\delta(q, a) = (q', b, m) \text{ bzw. } \delta(q, a) \ni (q', b, m)$$

folgendes:

Befindet sich M in Zustand q und liest Bandsymbol a (unter Lese-/Schreibkopf), so geht M im nächsten Schritt in Zustand q' über, überschreibt a durch b und

führt Kopfbewegung $m \in \{L(inks), R(rechts), N(ichts)\}$ aus.

Definition.

Eine Konfiguration einer TM M ist ein Wort $k \in \Gamma^*Q\Gamma^*$

Konfigurationen sind "Momentaufnahmen" von M :

$k = \alpha q \beta$ bedeutet:

$\alpha \beta$ ist der nichtleere (bzw. besuchte) Teil der Bandinschrift.

q ist der Zustand der TM .

Das erste Zeichen von β ist das Zentrum des Lese-/Schreibkopfes.

Startkonfiguration bei Eingabe $w \in \Sigma^* : q_0 w$

$$\begin{array}{c} \square \dots \square w \square \dots \square \\ \uparrow \end{array}$$

w steht auf dem Band

Der Lese-/Schreibkopf steht auf dem ersten Symbol von w

M ist in Zustand q_0

Formale Beschreibung der Arbeit einer TM :

Definition.

Beschreibe die Relation \vdash auf der Menge der Konfigurationen von M :

$$a_1 \dots a_m q b_1 \dots b_n \vdash \begin{cases} a_1 \dots a_m q' c b_2 \dots b_n & \text{falls } (q', c, N) \in \delta(q, b_1) \ m \geq 0, n \geq 1 \\ a_1 \dots a_m c q' b_2 \dots b_n & \text{falls } (q', c, R) \in \delta(q, b_1) \ m \geq 0, n \geq 2 \\ a_1 \dots a_{m-1} q' a_m c b_2 \dots b_n & \text{falls } (q', c, L) \in \delta(q, b_1) \ m \geq 1, n \geq 1 \\ a_1 \dots a_m \square q' c & \text{falls } (q', c, R) \in \delta(q, b_1) \text{ und } n = 1 \\ q' \square c b_2 \dots b_n & \text{falls } (q', c, L) \in \delta(q, b_1) \text{ und } m = 0 \end{cases}$$

\vdash^* bezeichne den transitiven Abschluß von \vdash

Beispiel.

Eine TM , die die Eingabe $w \in \Sigma^*$ als Binärzahl interpretiert und 1 hinzuaddiert:

$$M = (\{q_0, q_1, q_2, q_e\}, \{0, 1\}, \{0, 1, \square\}, \delta, q_0, \square, \{q_e\})$$

mit

$$\delta(q_0, 0) = (q_0, 0, R)$$

$$\delta(q_0, 1) = (q_0, 1, R)$$

$$\delta(q_0, \square) = (q_1, \square, L)$$

$$\delta(q_1, 0) = (q_2, 1, L)$$

$$\begin{aligned}\delta(q_1, 1) &= (q_1, 0, L) \\ \delta(q_1, \square) &= (q_e, 1, N)\end{aligned}$$

$$\begin{aligned}\delta(q_2, 0) &= (q_2, 0, L) \\ \delta(q_2, 1) &= (q_2, 1, L) \\ \delta(q_2, \square) &= (q_e, \square, R)\end{aligned}$$

Beispiel.

$w=101$

$$q_0 \vdash 1q_001 \vdash 10q_01 \vdash 101q_0\square \vdash 10q_1\square \vdash 1q_100\square \vdash q_2110\square \vdash q_2\square110\square \vdash \square q_e110$$

Definition.

Eine Funktion $f : \mathbb{N}^* \rightarrow \mathbb{N}$ heißt *Turing-berechenbar*, falls es eine (deterministische) TM M gibt, so daß für alle $n_1, \dots, n_k, m \in \mathbb{N}$ gilt:

$$f(n_1, \dots, n_k) = m$$

\iff

$$q_0 \text{bin}(n_1) \# \text{bin}(n_2) \# \dots \# \text{bin}(n_k) \vdash^* q_e \text{bin}(m)$$

wobei $q_e \in E$.

($\text{bin}(n)$ bezeichnet die Binärdarstellung von $n \in \mathbb{N}$ ohne führende Nullen)

Definition.

Eine Funktion $f : \Sigma^* \rightarrow \Sigma$ heißt *Turing-berechenbar*, falls es eine (deterministische) TM M gibt, so daß für alle $x, y \in \Sigma^*$ gilt:

$$f(x) = y$$

\iff

$$q_0 x \vdash^* q_e y$$

wobei $q_e \in E$.

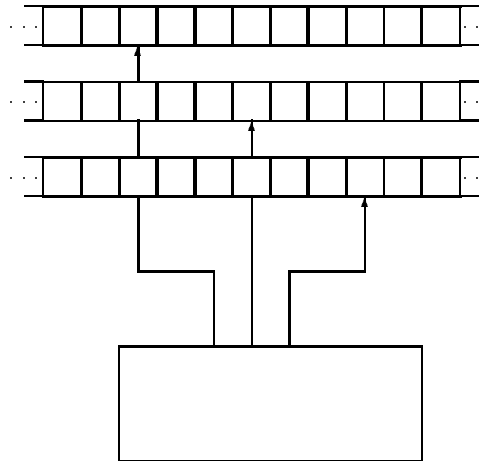
Damit ist ausgedrückt, daß im Falle $f(x) = \text{undef.}$ M in eine unendliche Schleife geht.

Beispiel.

1. $f : n \rightarrow n + 1$ ist Turing-berechenbar.
Das Beispiel überführt $bin(n)$ in $bin(n + 1)$
2. die überall nichtdefinierte Funktion ist Turing-berechenbar;
z.B.: durch eine TM mit

$$\delta(q_0, a) = (q_0, a, R) \text{ für alle } a \in \Gamma$$

3. Typ 0- (semientscheidbare) Sprachen sind berechenbar.

Mehrband-TM:

Die Lese-/Schreibköpfe können sich unabhängig voneinander bewegen.

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

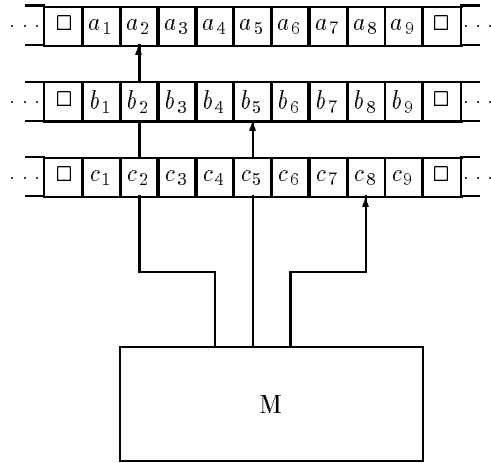
Satz.

Zu jeder Mehrband-TM M gibt es eine 1-Band TM M' die dieselbe Funktion berechnet wie M .

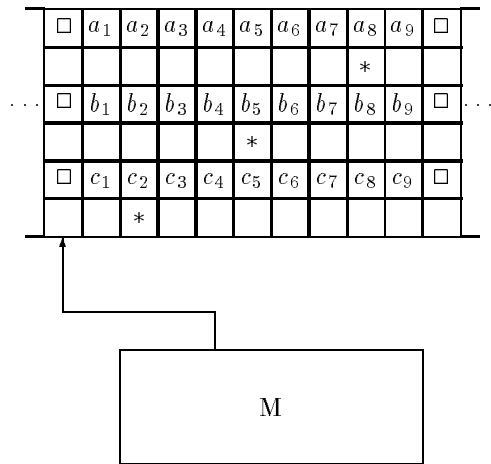
Beweis.

Sei k die Anzahl der Bänder, Γ Arbeitsalphabet von M .

Idee:



wird simuliert durch:



Formal

$$\Gamma' := (\Gamma \cup \{*\})^{2k}$$

M' simuliert M wie folgt:

Gestartet mit Eingabe $x_1, \dots, x_n \in \Gamma^*$ erzeugt M' die Darstellung der Startkonfiguration von M in Spuren-Darstellung.

- M' simuliert jeweils einen Schritt von M durch mehrere Schritte:
 M' positioniert den Lese-/Schreibkopf links von allen *-Markierungen.
 M' geht nach rechts bis alle k Markierungen überschritten sind und
 "weiß" nun worauf die δ -Funktion von M anzuwenden ist.
 M' geht in den entsprechenden Zustand über.
 (denn $|Q'| \geq |Q \times \Gamma^k|$)

Man kann in Zukunft einfach eine Mehrbandmaschine angeben, und wissen, daß diese durch eine 1-Band Maschine simuliert werden kann.

- Sei M eine 1-Band TM .

$$M(i, k) ; i \leq n$$

bezeichne die k -Band TM , die aus M dadurch entsteht, daß alle Aktionen auf Band i ablaufen.

Beispiel.

$M : \delta(q, a) = (q', b, y) , y \in \{L, R, N\}$ entspreche den Übergang in M'

$$\delta'(q, c_1, c_2, a, c_3, c_4) = (q', c_1, c_2, b, c_3, c_4, N, N, y, N, N)$$

falls k unwichtig ist, dann schreibe lediglich $M(i)$ statt $M(i, k)$.

- "Hintereinanderschaltung" von TM :
 Seien $M_i = (Q_i, \Sigma, \Gamma_i, \delta_i, q_i, \square, F_i) ; i = 1, 2 ; Q_1 \cap Q_2 = \emptyset$

$$\text{start} \longrightarrow M_1 \longrightarrow M_2 \longrightarrow \text{stop}$$

bzw.

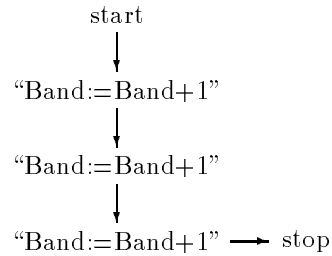
$$M_1; M_2$$

bezeichnet die TM :

$$\begin{aligned}
 M &= (Q_1 \cup Q_2, \Sigma, \Gamma_1 \cup \Gamma_2, \delta, q_1, \square, F_2) \\
 \delta &= \delta_1 \cup \delta_2 \cup \{(q_e, a, q_2, a, N) : q_e \in F_1, a \in \Gamma_1\}
 \end{aligned}$$

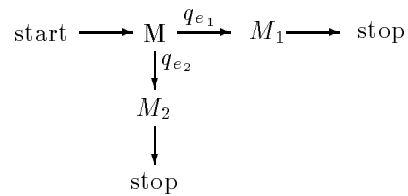
Beispiel.

•



bezeichnet die TM , die 3 hinzuaddiert.

•



bezeichnet die TM , die im Endzustand q_{e_1} M_1 startet und im Endzustand q_{e_2} M_2

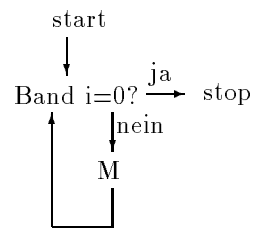
• Betrachte: Die TM "Band=0?" :

$$Q = \{q_0, q_1, \text{ja}, \text{nein}\} \quad q_0 \text{ Startzustand}$$

$$\begin{aligned}
 \delta(q_0, a) &= (\text{nein}, a, N) \text{ f\"ur } a \neq \square \\
 \delta(q_0, 0) &= (q_1, 0, R) \\
 \delta(q_1, a) &= (\text{nein}, a, L) \text{ f\"ur } a \neq \square \\
 \delta(q_1, \square) &= (\text{ja}, \square, L)
 \end{aligned}$$

Schreibe "Band i=0?" anstelle von "Band=0?"

- “WHILE Band $i \neq 0$ DO M”



1.4 LOOP-, WHILE- und GOTO-Berechenbarkeit

Die einfache Programmiersprache LOOP:

- Variablen: x_0, x_1, \dots
- Konstanten: $0, 1, 2, \dots$
- Trennsymbole: $;$, $:=$
- Operationszeichen: $+$, $-$
- Schlüsselwörter: LOOP, DO, END

Induktive Definition der Syntax von LOOP:

- Wertzuweisung

$$\begin{aligned} x_i &:= x_j + c \\ x_i &:= x_j - c, \quad c \text{ Konstante} \end{aligned}$$

ist ein LOOP-Programm

- Sind P_1, P_2 LOOP-Programme, dann auch $P_1; P_2$
- Ist P ein LOOP-Programm, x_i Variable, dann auch

LOOP x_i DO P END;

Semantik der LOOP-Programme:

Soll ein LOOP-Programm eine k -stellige Funktion berechnen, und ist (n_1, \dots, n_k) das Argument, dann gilt:

$$x_i := \begin{cases} n_i & i \leq k \quad (\text{Startsituation}) \\ 0 & \text{sonst} \end{cases}$$

Die Wertzuweisung wird wie üblich interpretiert:

$$x_i := x_j + c .$$

Bei

$$x_i := x_j - c$$

modifizierte Substitution

$$x_i := \begin{cases} x_i - c & c \leq x_i \\ 0 & \text{sonst} \end{cases}$$

$P_1; P_2$ wird so interpretiert, daß zuerst P_1 und dann P_2 ausgeführt wird.

LOOP x_i DO P END wird so interpretiert, daß P sooft ausgeführt wird, wie der Wert der Variable x_i zu Beginn angibt.

Das Resultat ergibt sich als Wert von x_0 .

Definition.

Eine Funktion $f : N^k \rightarrow N$ heißt *LOOP-berechenbar*, falls es ein LOOP-Programm P gibt, das gestartet mit n_1, \dots, n_k in den Variablen x_1, \dots, x_k mit dem Wert $f(n_1, \dots, n_k)$ in x_0 stoppt.

Bemerkung

Alle LOOP-berechenbaren Funktionen sind total definiert.

- IF $x = 0$ THEN A END

kann simuliert werden durch

```

y := 1;
LOOP x DO y := 0 END;
LOOP y DO A END

```

Beispiel.

- Die Addition " $x_0 := x_1 + x_2$ " ist LOOP-berechenbar:

```

x_0 := x_1;
LOOP x_2 DO x_0 := x_0 + 1 END

```

mit Verallgemeinerung auf " $x_0 := x_i + x_j$ "

- Die Multiplikation " $x_0 := x_1 * x_2$ " ist LOOP-berechenbar:

```

x_0 := 0;
LOOP x_2 DO x_0 := x_0 + x_1 END

```

Man kann auch DIV, MOD durch LOOP-Programme beschreiben
 $\rightsquigarrow x := (y \text{ DIV } z) + (x \text{ MOD } 5) * z$

Erweiterung der LOOP-Programme durch die WHILE-Schleife
 Ergebnis: WHILE-Programme:

- Ist P ein WHILE-Programm, x_i Variable, dann ist auch

WHILE $x \neq 0$ DO P END;

ein WHILE-Programm.

Semantik.

P wird solange ausgeführt, wie $x_i \neq 0$ gilt.

Man kommt in WHILE-Programmen ohne LOOP aus:

LOOP x DO P END;

wird simuliert durch

$y := x;$
WHILE $y \neq 0$ DO $y := y - 1; P$ END;

Definition.

Eine Funktion $f : \mathbb{N}^k \rightarrow \mathbb{N}$ heißt *WHILE-berechenbar*, falls es ein WHILE-Programm P gibt, das gestartet mit n_1, \dots, n_k in x_1, \dots, x_k (0 sonst) mit dem Wert $f(n_1, \dots, n_k)$ in x_0 stoppt.

Sonst stoppt P nicht.

Satz.

TM können WHILE-Programme simulieren.

D.h. jede WHILE-berechenbare Funktion ist auch TM-berechenbar.

Beweis.

Man kann die Wertzuweisungen, Sequenzen und WHILE-Schleifen mit einer Mehrband TM simulieren.

(Wobei das i -te Band der i -ten Variablen (binär) entspricht.)

Man kann eine Mehrband TM mit einer 1-Band TM simulieren. ■

Beweis der Umkehrung:

Betrachte GOTO-Programme.

GOTO-Programme bestehen aus Folgen von markierten Anweisungen

$M_1 : A_1 ; M_2 : A_2 ; \dots ; M_k : A_k$

Als Anweisungen sind zugelassen:

Wertzuweisungen: $x_i := x_j \pm c$

unbedingter Sprung: GOTO M_i

bedingter Sprung: IF $x_i = c$ THEN GOTO M_i

Stoppanweisung: HALT

Vereinbarung: Marken von nicht angesprungen Anweisungen werden weggelassen.

Die Semantik ist klar.

Definition.

Eine Funktion $f : \mathbb{N}^k \rightarrow \mathbb{N}$ heißt *GOTO-berechenbar*, falls es ein GOTO-Programm P gibt, das gestartet mit n_1, \dots, n_k in x_1, \dots, x_k (0 sonst) mit dem Wert $f(n_1, \dots, n_k)$ in x_0 stoppt.

Sonst stoppt P nicht.

Satz.

Jedes WHILE-Programm kann durch ein GOTO-Programm simuliert werden.

Beweis.

WHILE $x_i \neq 0$ DO P END

kann simuliert werden durch

M_1 : IF $x_i = 0$ THEN GOTO M_2 ;

P ;

 GOTO M_1 ;

M_2 : ...

■

Korollar

Jede WHILE-berechenbare Funktion ist GOTO-berechenbar.

Satz.

Jedes GOTO-Programm kann durch ein WHILE-Programm (mit nur einer WHILE-Schleife) simuliert werden.

Beweis.

Gegeben sei ein GOTO-Programm

$M_1 : A_1 ; M_2 : A_2 ; \dots ; M_k : A_k$

wird simuliert durch folgendes WHILE-Programm mit nur einer WHILE-Schleife:

count:=1;

WHILE count \neq 0 DO

 IF count = 1 THEN A'_1 END;

 IF count = 2 THEN A'_2 END;

\vdots
 IF $count = k$ THEN A'_k END;
 END

wobei

$$A'_i = \begin{cases} x_j := x_i \pm c; count := count + 1 & \text{falls } A_i = x_j \pm c \\ count := n & \text{falls } A_i = \text{GOTO } M_n \\ \text{IF } x_j = c \text{ THEN } count := n \text{ ELSE} & \text{falls } A_i = \text{THEN GOTO } M_n \\ count := count + 1 \text{ END} & \text{falls } A_i = \text{HALT} \\ count := 0 & \end{cases}$$

■

Korollar

Eine Funktion $f : \mathbb{N}^k \rightarrow \mathbb{N}$ ist GOTO-berechenbar, falls f WHILE-berechenbar ist.

Satz. (Kleenesche Normalformsatz für WHILE-Programme)

Jede WHILE-berechenbare Funktion $f : \mathbb{N}^k \rightarrow \mathbb{N}$ kann durch ein WHILE-Programm mit nur einer WHILE-Schleife berechnet werden.

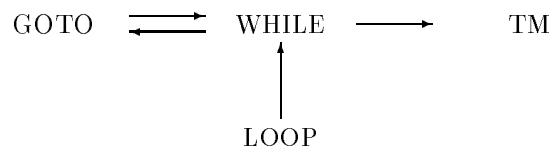
Beweis.

Sei P ein beliebiges WHILE-Programm für f .

P wird simuliert durch das GOTO-Programm P' .

P' wird simuliert durch das WHILE-Programm P'' mit nur einer WHILE-Schleife. ■

Bisher ist gezeigt worden:



Satz.

GOTO-Programme können TM simulieren.

Beweis.

Sei $M = (Q, \Sigma, \Gamma, \delta, q_1, \square, F)$ eine TM die f berechnet.

Das simulierende GOTO-Programm hat folgende Gestalt:

$M_1 : A_1 ; M_2 : A_2 ; M_3 : A_3$

wobei:

- P_1 : transformiert die gegebenen Anfangswerte in Binärdarstellung und erzeugt eine Darstellung der Starkonfiguration von M in die Variablen x, y, z .
- P_2 : simuliert die Rechnung von M Schritt-für-Schritt durch Veränderung der Variablenwerte x, y, z .
- P_3 : erzeugt aus der kodierten Form der Endkonfiguration in x, y, z die eigentliche Ausgabe in der Ausgabevariable x_0 .

Bemerkung

P_1, P_3 hängen nicht von M ab, lediglich von P_2 .

Kodierung:

Sei

$$Q = \{q_1, \dots, q_k\}$$

$$\Gamma = \{a_1, \dots, a_m\}, b > \#\Gamma$$

Kodierung der TM –Konfiguration:

$$a_{i_1} \dots a_{i_p} q_l a_{j_1} \dots a_{j_q}$$

durch die folgenden Werte von x, y, z :

$$x = (i_1 \dots i_p)_b$$

$$y = (j_q \dots j_1)_b$$

$$z = l$$

GOTO-Programmstück $M_2 : P_2$:

```

M2: a := y MOD b;
      IF (z = 1) AND (a = 1) THEN GOTO M11
      IF (z = 1) AND (a = 2) THEN GOTO M12
      ⋮
      IF (z = k) AND (a = m) THEN GOTO Mkm
M11: ⊗
      GOTO M2
M12: ⊗
      GOTO M2
      ⋮
Mkm: ⊗
      GOTO M2

```

wobei \otimes bedeutet:

Man nimmt das Programmstück das mit M_{ij} markiert ist

$$\text{Sei } \delta(q_i, a_j) = (q_i, a_j, L)$$

und simuliert den Übergang durch:

$$\begin{aligned} z &:= i'; \\ y &:= y \text{ DIV } b; \\ y &:= b * y + j'; \\ y &:= b * y + (x \text{ MOD } b); \\ x &:= x \text{ DIV } b; \end{aligned}$$

ist q_i Endzustand, kann man

für \otimes GOTO M_3 setzen.

Der Rest ist klar. ■

Korollar

Eine Funktion ist

TM-berechenbar \Leftrightarrow GOTO-berechenbar \Leftrightarrow WHILE-berechenbar

1.5 Primitiv rekursive und μ -rekursive Funktionen

Ein weiterer, sehr früher Ansatz zur Erfassung des Berechenbarkeitsbegriffs:

Definition.

Die Klasse der primitiv rekursiven Funktionen ist induktiv wie folgt definiert:

1. Alle konstanten Funktionen sind primitiv rekursiv.
2. Projektionsabbildungen sind primitiv rekursiv, d.h.

$$\begin{aligned} pr_i : \mathbb{N}^k &\rightarrow \mathbb{N} \\ pr_i(n_1, \dots, n_k) &= n_i \end{aligned}$$

3. Die Nachfolgerfunktion ist primitiv rekursiv, d.h.

$$\begin{aligned} s : \mathbb{N}^k &\rightarrow \mathbb{N} \\ i &\mapsto i + 1 \end{aligned}$$

4. Jede Funktion, die durch Einsetzung (Komposition) aus primitiv rekursiven Funktionen entsteht ist primitiv rekursiv.
5. Jede Funktion, die durch primitive Rekursion aus primitiv rekursiven Funktionen entsteht, ist primitiv rekursiv.

Schema der primitiven Rekursion:

Seien g, h primitiv rekursive Funktionen

Dann ist

$$\begin{aligned} f(0, \alpha) &= g(\alpha) \\ f(n + 1, \alpha) &= h(n, f(n, \alpha), \alpha) \end{aligned}$$

primitiv rekursiv.

Beispiel.

1. Addition:

$add : \mathbb{N}^2 \rightarrow \mathbb{N}$ mit $add(x, y) = x + y$.

Es gilt:

$$\begin{aligned} add(0, x) &= x && \text{(Projektion)} \\ add(n + 1, x) &= s(pr_1(add(n, x), n)) \end{aligned}$$

(s Nachfolgerfunktion; pr_1 Projektion auf die 1. Komponente)

2. Multiplikation:

$mult : \mathbb{N}^2 \rightarrow \mathbb{N}$ mit $mult(x, y) = xy$.

Es gilt:

$$\begin{aligned} mult(0, x) &= 0 \\ mult(n+1, x) &= add(mult(n, x), x) \end{aligned}$$

- primitiv rekursive Funktionen sind offenbar berechenbar.

Die Basisfunktionen sind berechenbar.

Das Einsetzungsschema ist berechenbar.

Das primitive Rekursionsschema ist berechenbar.

- Primitiv rekursive Funktionen sind total berechenbar.

- Achtung: Es gilt nicht:

primitiv rekursiv = total und berechenbar.

Satz.

Die Klasse der primitiv rekursiven Funktionen stimmt mit der Klasse der LOOP-berechenbaren Funktionen überein.

Beweisidee

Es gelten folgende Korrespondenzen:

Wertzuweisung	Basisfunktion
P; Q	Komposition
LOOP-Schleife	primitive Rekursion

■

Erweiterung der Klasse der primitiv rekursiven Funktionen mit Hilfe des μ -Operators:

Definition.

Sei $f : \mathbb{N}^{k+1} \rightarrow \mathbb{N}$

Durch Anwendung des μ -Operators entsteht aus f die Funktion $g : \mathbb{N}^k \rightarrow \mathbb{N}$

mit

$$g(x_1, \dots, x_k) = \min \{ n : f(n, x_1, \dots, x_k) = 0 \text{ und f\u00fcr alle } m < n \text{ ist } f(m, x_1, \dots, x_k) \text{ def.} \}$$

mit $\min \emptyset = \text{undefiniert}$.

Durch die Anwendung des μ -Operators k\u00f6nnen partielle Funktionen entstehen.

Beispiel.

Sei $f(x, y) \equiv 1$

Dann entsteht durch die Anwendung des μ -Operators die vollständig undefinierte Funktion Ω .

Definition

Die Klasse der μ -rekursiven Funktionen ist die kleinste Klasse von Funktionen, die die Basisfunktionen (konstante Funktionen, Projektionen, Nachfolgerfunktion) enthält und abgeschlossen ist bzgl. der primitiven Rekursion und der Anwendung des μ -Operators.

Satz.

Die Klasse der μ -rekursiven Funktionen stimmt genau mit der Klasse der WHILE- (GOTO-, TM -) berechenbaren Funktionen überein.

ohne Beweis

- Im dem Beweis zum obigen Satz werden WHILE-Schleifen durch Anwendung des μ -Operators simuliert.

\Rightarrow Kleene'sche Normalformtheorem für WHILE-Programme

Satz. (Kleene)

Für jede n -stellige μ -rekursive Funktion f gibt es zwei $n + 1$ stellige, primitiv rekursive Funktionen p und q , so daß sich f darstellen läßt als

$$f(x_1, \dots, x_n) = p(x_1, \dots, x_n, \mu q(x_1, \dots, x_n)).$$

1.6 Die Ackermann-Funktion

Ackermann gab 1925 eine Funktion an, die intuitiv berechenbar (also WHILE-berechenbar), jedoch nicht primitiv rekursiv (also LOOP-berechenbar) ist.

$$\begin{aligned} \text{ack}(0, y) &= y + 1 \\ \text{ack}(x, 0) &= \text{ack}(x - 1, 1) \\ \text{ack}(x, y) &= \text{ack}(x - 1, \text{ack}(x, y - 1)) \end{aligned}$$

z.B.:

$$\text{ack}(x, y) = \underbrace{\text{ack}(x - 1, \text{ack}(x - 1, \dots \text{ack}(x - 1, 1) \dots))}_{y\text{-mal}}$$

Betrachte eine ähnlich definierte Funktion a :

$$\begin{aligned} a(0, y) &= a(x, 0) = 1 \\ a(1, y) &= 3y + 1 \\ a(x, y) &= \underbrace{a(x - 1, a(x - 1, \dots a(x - 1, y) \dots))}_{y\text{-mal}} \end{aligned}$$

- a ist total definiert (vollständige Induktion)

MODULA-Prozedur für die Berechnung von a :

```

PROCEDURE a(x, y: CARDINAL): CARDINAL;
VAR i, s : CARDINAL;
BEGIN
  IF (x = 0) OR (y = 0) THEN RETURN 1
  ELSIF x = 1 THEN RETURN 3 * y + 1
  ELSE s := y
      FOR i := 1 TO y DO
        s = a(x - 1, s)
      END;
  RETURN s
END;
END a;

```

a ist nicht LOOP-berechenbar, denn

Sei P ein LOOP-Programm.
 Ordne P die Funktion $f_P : \mathbb{N} \rightarrow \mathbb{N}$ zu:
 Seien x_0, \dots, x_n alle in P vorkommenden Variable.

n_i sei der Startwert von x_i .

n'_i sei der Endwert von x_i nach Ablauf von P .

$$f_P(n) := \max\{\sum_{i=0}^k n'_i \mid \sum_{i=0}^k n_i \leq n\}$$

(größtmögliche Summe aller Variablenendwerte, falls P mit dem Startwert der Summe $\leq n$ gestartet wird.)

Lemma

Für jedes LOOP-Programm P gibt es eine Konstante k mit $f_P(n) < a(k, n)$ für alle $n \geq k$.

Beweis.

Induktion über den Aufbau von P

- Habe P die Form $x_i := x_j \pm c$:

$$\Rightarrow f_P(n) \leq n + n + c = 2n + c$$

$$\Rightarrow f_P(n) < 3n + 1 = a(1, n) \leq a(k, n) \text{ für alle } k \geq 1, n \geq c$$

Wähle $k := c + 1$.

- Habe P die Form $P_1; P_2$:

Nach Voraussetzung gibt es k_1, k_2 mit

$$f_{P_1} < a(k_1, n)$$

$$f_{P_2} < a(k_2, n) \text{ für allen } n \geq \max\{k_1, k_2\}$$

Setze $k_3 := \max\{k_1, k_2\}$

Es gilt:

$$\begin{aligned} f_P(n) &\leq f_{P_2}(f_{P_1}(n)) \\ &\leq f_{P_2}(a(k_1, n)) \\ &< a(k_2, a(k_1, n)) \\ &\leq a(k_3, a(k_3, n)) \\ &\leq \underbrace{a(k_3, a(k_3, \dots a(k_3, n) \dots))}_{n\text{-mal}} \text{ falls } n \geq 2 \\ &= a(k_3 + 1, n) \end{aligned}$$

Wähle $k := \max\{k_3, 2\}$

- Habe P die Form LOOP x_i DO P' END:

Nach Induktionvoraussetzung gibt es ein k' mit

$f'_P < a(k', n)$ für alle $n \geq k'$

Es gilt:

$$\begin{aligned}
 f_P(n) &= \underbrace{f'_P(f'_P(\dots f'_P(n) \dots))}_{n_i\text{-mal}} \\
 &< \underbrace{a(k', a((k', \dots a(k', n) \dots))}_{n_i\text{-mal}} \\
 &\leq \underbrace{a(k', a(k', \dots a(k', n) \dots))}_{n\text{-mal}} \\
 &= a(k' + 1, n)
 \end{aligned}$$

wähle $k := k' + 1$

■

Satz.

Die Ackermann-Funktion a ist nicht LOOP-berechenbar.

Beweis.

Annahme: a ist LOOP-berechenbar

$\Rightarrow g(n) := a(n, n)$ ist LOOP-berechenbar

Sei P ein LOOP-Programm für g

$\Rightarrow g(n) \leq f_P(n)$

wähle k in P mit: $f_P(n) < a(k, n)$

Für $n = k$ gilt:

$g(k) \leq f_P(k) < a(k, k) = g(k)$ (Widerspruch) ■

- a ist auch formal berechenbar.

WHILE-Programm für a :

Zunächst Angabe eines Programmes zur Berechnung von a , daß mit den Stackoperationen PUSH und POP operiert:

```

INPUT(x, y);
INIT(stack);
PUSH(x, stack);
PUSH(y, stack);
WHILE size(stack) ≠ 1 DO
  y := POP(stack);
  x := POP(stack);
  IF (x = 0) OR (y = 0) THEN PUSH(1, stack);

```

```

    ELSIF  $x = 1$  THEN PUSH( $3 * y + 1, stack$ );
    ELSE LOOP  $y$  DO PUSH( $k - 1, stack$ ) END;
    PUSH( $y, stack$ )
END;
 $result := POP(stack)$ ;
OUTPUT( $result$ );

```

Man kann die einzelnen Stackoperationen durch WHILE-Programme simulieren:

Betrachte: $c : \mathbb{N} \rightarrow \mathbb{N}$ mit $c(x, y) := 2^{x+y} + x$
 c ist WHILE berechenbar und injektiv;

Beispiel: Werte für c :

	0	1	2	3	4
0	1	3	6	11	20
1	2	5	10	19	36
2	4	9	18	35	68
3	8	17	34	67	132
4	16	33	66	131	260

Man benutzt c zur Kodierung von Paaren.

Man kann aus $c(x, y) = n$ das Paar (x, y) zurückgewinnen:

$$\begin{aligned}
 &\text{Sei } k \text{ max. mit } 2^k < n \\
 &\Rightarrow x := n - 2^k \\
 &\quad y := k - x
 \end{aligned}$$

Definiere:

$$c_1(n) := \begin{cases} x & \text{falls } n \in c(\mathbb{N}^2) \\ 0 & \text{sonst} \end{cases} ; c_2(n) := \begin{cases} y & \text{falls } n \in c(\mathbb{N}^2) \\ 0 & \text{sonst} \end{cases}$$

c_1, c_2 sind WHILE- (sogar LOOP-) berechenbar.

Nun zur Simulation des Stacks durch ein WHILE-Programm:

Sei (n_1, \dots, n_k) der Inhalt des Stacks,

n_1 das Head-Element.

Kodierung von (n_1, \dots, n_k) mit Hilfe von c :

$$n := c(n_1, c(n_2, \dots, c(n_k, 0) \dots))$$

INIT(<i>stack</i>)	wird simuliert durch	$n := 0$
PUSH($a, stack$)	wird simuliert durch	$n := c(a, n)$
POP(<i>stack</i>)	wird simuliert durch	$result := c_1(n)$
		$n := c_2(n)$
		RETURN $result$
size($stack \neq 1$)	wird simuliert durch	$c_2(n) \neq 0$

Lemma

Die Ackermannfunktion ist WHILE-berechenbar.

1.7 Entscheidbarkeit und Semi-Entscheidbarkeit

- Der Berechenbarkeitsbegriff betrifft Funktionen.
- Einführung eines entsprechenden Begriffs für Sprachen.

Definition.

$A \subseteq \Sigma^*$ heißt *entscheidbar*, falls die *charakteristische Funktion* $\chi_A : \Sigma^* \rightarrow \{0, 1\}$

$$\chi_A(w) = \begin{cases} 1 & \text{falls } w \in A \\ 0 & \text{sonst} \end{cases}$$

berechenbar ist.

$A \subseteq \Sigma^*$ heißt *semi-entscheidbar*, falls die charakteristische Funktion $\chi'_A : \Sigma^* \rightarrow \{0, 1\}$

$$\chi'_A(w) = \begin{cases} 1 & \text{falls } w \in A \\ \text{undefiniert} & \text{sonst} \end{cases}$$

berechenbar ist.

- Man kann an Stelle von $A \subseteq \Sigma^*$ auch $A \subseteq \mathbb{N}$ betrachten.
- Das Entscheidungsproblem für A ist die Frage nach einem stoppenden Algorithmus mit



- Das Semi-Entscheidungsproblem für A ist die Frage nach einem Algorithmus mit



o

Hat der Algorithmus noch nicht gestoppt, dann ist unklar ob $w \in A$ oder nicht.

Beispiel.

Das Entscheidungsproblem für die Prädikatenlogik (“Theorembeweiser”).

Satz.

A ist entscheidbar \Leftrightarrow sowohl A als auch \overline{A} sind semi-entscheidbar.

Beweis.

(\rightarrow): klar.

(\leftarrow): Sei M_1 ein Semi-Entscheidungsverfahren für A .
Sei M_2 ein Semi-Entscheidungsverfahren für \overline{A} .

Erhalte ein Entscheidungsverfahren für A :

INPUT(x);

FOR $s := 1, 2, 3, \dots$ DO

 IF M_1 bei Eingabe x in s Schritten stoppt

 THEN OUTPUT(1) END;

 IF M_2 bei Eingabe x in s Schritten stoppt

 THEN OUTPUT(0) END;

END

Definition.

$A \subseteq \Sigma^*$ heißt *rekursiv aufzählbar*, falls $A = \emptyset$ oder es eine totale und berechenbare Funktion $f : \mathbb{N} \rightarrow \Sigma^*$ gibt mit

$$A = \{f(0), f(1), f(2), \dots\}$$

“ f zählt A auf.” (evtl. mit $f(i) = f(j)$ für $i \neq j$!)

Satz.

Eine Sprache ist rekursiv aufzählbar, genau dann wenn sie semi-entscheidbar ist.

Beweis.

(\rightarrow): Sei A rekursiv aufzählbar mittels der Funktion f .

Erhalte ein Semi-Entscheidungsverfahren für A :

INPUT(x);

FOR $n := 0, 1, 2, 3, \dots$ DO

 IF $f(n) = x$ THEN OUTPUT(1) END;

END

(\leftarrow): Sei $A \neq \emptyset$ semi-entscheidbar mittels Algorithmus M .

Sei $a \in A$ fixiert

Definiere eine totale und berechenbare Funktion f

mit $f(\mathbb{N}) = A$ mittels folgendem Algorithmus:

INPUT(n);

* Interpretiere n als Kodierung $n = c(x, y)$

mit $x = c_1(n)$, $y = c_2(n)$ *

$x := c_1(n)$;

$y := c_2(n)$;

IF M angesetzt auf x in y Schritten stoppt

THEN OUTPUT(x) ELSE OUTPUT(a) END;

Der Algorithmus stoppt stets und gibt nur Elemente aus A aus.

$\Rightarrow f$ ist total und berechenbar,
 $f(\mathbb{N}) \subseteq A$

Noch zu zeigen: $f(\mathbb{N}) = A$, denn

Sei $b \in A$ beliebig.
 $\Rightarrow M$ stoppt bei Eingabe b in s Schritten.

Betrachte: $n = c(b, s)$
 $\Rightarrow f(n) = b$ nach Konstruktion des Algorithmus ■

Insgesamt erhält man:

Satz.

Eine Sprache A ist entscheidbar, genau dann wenn A und \bar{A} rekursiv aufzählbar sind.

Zusammenfassung:

Bisher ist die Äquivalenz der folgenden Aussagen gezeigt worden:

- A ist rekursiv aufzählbar.
- $\Leftrightarrow A$ ist semi-entscheidbar.
- $\Leftrightarrow A$ ist vom Typ 0 (als formale Sprache).
- $\Leftrightarrow A = L(M)$ für eine TM M .
- $\Leftrightarrow \chi'_A$ ist berechenbar.
- $\Leftrightarrow A$ ist Definitionsbereich einer berechenbaren Funktion.
- $\Leftrightarrow A$ ist Wertebereich einer berechenbaren Funktion.

Abschließende Bemerkung zum Zusammenhang

Abzählbarkeit — rekursive Aufzählbarkeit

Definition.

A heißt *abzählbar*, falls $A = \emptyset$ oder es gibt eine totale Funktion f gibt mit

$$A = \{f(0), f(1), f(2), \dots\}$$

- A ist rekursiv aufzählbar, falls A durch eine totale rekursive Funktion abzählbar ist.

Unterschied:

Sei A abzählbar, $A' \subseteq A \Rightarrow A'$ ist abzählbar

Beweis.

Sei A abzählbar mittels f , $a \in A'$ fixiert.

Betrachte:

$$g(n) = \begin{cases} f(n) & \text{falls } f(n) \in A' \\ a & \text{sonst} \end{cases}$$

$g(n)$ zählt A' ab, da $A' = \{g(0), g(1), \dots\}$ ■

Sei A rekursiv abzählbar.

Es gibt Teilmengen $A'' \subseteq A$, die nicht rekursiv abzählbar sind.

Beweis.

später.

1.8 Das Halte-Problem und die Reduzierbarkeit

- Kennenlernen unentscheidbarer Probleme.
Besonders berühmt: Das Halteproblem für TM .
Dazu Kodierung der TM $M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$ als Wort über $\{0,1\}$

1. Kodierung von M als Wort über $\{0,1,\#\}$:

$$\text{Sei } \begin{aligned} Q &= \{q_0, \dots, q_n\} \\ \Gamma &= \{a_0, \dots, a_k\} \end{aligned}$$

Schreibe $\delta(q_i, a_j) = (q_{i'}, a_{j'}, y)$
als

$$w_{i,j,i',j',y} = \#\#bin(i)\#bin(j)\#bin(i')\#bin(j')\#bin(m) \quad \text{mit } m = \begin{cases} 0 & y = L \\ 1 & y = R \\ 2 & y = N \end{cases}$$

Kodierung von M durch Konkatenation aller Worte $w_{i,j,i',j',y}$, die zu δ gehören.

2. Kodierung von M durch ein Wort über $\{0,1\}$:
Kodierung mit Hilfe von

$$0 \mapsto 00$$

$$1 \mapsto 01$$

$$\# \mapsto 11$$

$w_{i,j,i',j',y}$ durch ein Wort über $\{0,1\}$

Sei M_0 eine fixierte TM

$$w \in \{0,1\}^* \mapsto M_w = \begin{cases} M & \text{falls } w \text{ Codewort von } M \text{ ist} \\ 0 & \text{sonst} \end{cases}$$

Definiton

Die folgende Sprache

$$K = \{w \in \{0,1\}^* \mid M_w \text{ angesetzt auf } w \text{ hält}\}$$

heißt *spezielles Halte-Problem*.

Satz.

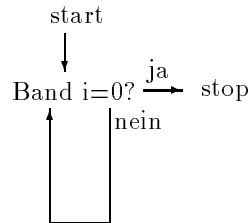
Das spezielle Halte-Problem ist nicht entscheidbar.

Beweis.

Annahme: K ist entscheidbar.

$\Leftrightarrow \chi_K$ ist berechenbar mittels TM M .

Betrachte: TM M'



M' stoppt, falls M 0 ausgibt.

Gibt M 1 aus, geht M' in eine Endlos-Schleife.

Sei $w' \in \{0, 1\}$ mit $M_{w'} = M$

Es gilt:

M' angesetzt auf w' hält.

$\Leftrightarrow M$ angesetzt auf w' gibt 0 aus.

$\Leftrightarrow \chi_K(w') = 0$ (Def. von M)

$\Leftrightarrow w' \in K$

$\Leftrightarrow M_{w'} = M'$ hält angesetzt auf w' nicht. (Widerspruch) ■

Das Reduktionskonzept ermöglicht eine "leichte" Übertragung dieses Resultats auf weitere Probleme:

Definition.

Seien $A, B \subseteq \Sigma^*$

A heißt *auf B reduzierbar* ($A \leq B$), falls es eine totale und berechenbare Funktion $f : \Sigma^* \rightarrow \Sigma^*$ gibt mit

$$x \in A \Leftrightarrow f(x) \in B$$

für alle $x \in \Sigma^*$.

Lemma

(i) Gilt $A \leq B$ und ist B entscheidbar, so ist auch A entscheidbar.

(ii) Gilt $A \leq B$ und ist B semientscheidbar, so ist auch A semientscheidbar.

Beweis.

(i) Sei $A \leq B$ mittels f

Sei χ_B berechenbar
 $\Leftrightarrow \chi_B \circ f$ ist berechenbar
 Es gilt:

$$\chi_A(x) = \begin{cases} 1 & x \in A \\ 0 & x \notin A \end{cases} = \begin{cases} 1 & f(x) \in B \\ 0 & f(x) \notin B \end{cases} = \chi_B(f(x))$$

$\Leftrightarrow \chi_A$ ist berechenbar und A ist entscheidbar

(ii) ersetze in (i) χ durch χ' und 0 durch undefiniert. ■

Korollar

$A \leq B$ und A ist nicht entscheidbar.
 $\Rightarrow B$ ist nicht entscheidbar.

Beweis.

Kontraposition von (i) ■

Definition.

Die Sprache

$$H = \{w\#x \mid M_w \text{ angesetzt auf } x \text{ hält}\}$$

heißt (allgemeines) *Halte-Problem*.

Satz.

Das Halte-Problem ist nicht entscheidbar.

Beweis.

Es reicht zu zeigen: $K \leq H$

wähle $f(w) = w\#w$

$\Rightarrow w \in K \Leftrightarrow f(w) \in H$ ■

Definiton

Die Sprache

$$H_0 = \{w \mid M_w \text{ angesetzt auf leeren Band hält}\}$$

heißt *Halte-Problem auf leeren Band*.

Satz.

Das Halte-Problem auf dem leeren Band H_0 ist nicht entscheidbar.

Beweis.

Es reicht zu zeigen: $H \leq H_0$.

Ordne $w\#x$ folgende TM zu:

gestartet mit leeren Band schreibt M x auf das Band, arbeitet dann wie M_w (angesetzt auf x).

Die Arbeitsweise von M gestartet mit nicht leeren Band ist unerheblich.

$f : w\#x \rightarrow \text{Code von } M$.

f ist berechenbar. Man kann f zu einer totalen und berechenbaren Funktion erweitern.

Es gilt:

$$\begin{aligned} w\#x \in H &\Leftrightarrow M_w \text{ angesetzt auf } x \text{ h\u00e4lt} \\ &\Leftrightarrow M \text{ angesetzt auf leerem Band h\u00e4lt} \\ &\Leftrightarrow f(w\#x) \in H_0 \end{aligned}$$

Satz von Rice.

Sei \mathcal{R} die Klasse aller TM-berechenbaren Funktionen

Sei $\mathcal{S} \subset \mathcal{R}$, $\mathcal{S} \neq \emptyset$,

Dann ist die Sprache

$$C(\mathcal{S}) := \{w \mid \text{die von } M_w \text{ berechnete Funktion liegt in } \mathcal{S}\}$$

unentscheidbar.

Beweis.

Sei $\Omega \in \mathcal{R}$ eine \u00fcberall undefinierte Funktion.

1. Fall: $\Omega \in \mathcal{S}$

Wegen $\mathcal{S} \neq \mathcal{R}$ gibt es eine Funktion $q \in \mathcal{R} - \mathcal{S}$.

Sei Q eine TM, die q berechnet.

ordne $w \in \{0, 1\}$ die TM M zu mit:

Angesetzt auf die Eingabe y ignoriert M diese zun\u00e4chst und verh\u00e4lt sich wie M_w angesetzt auf das leere Band.

Falls diese Rechnung zu Ende kommt, so verh\u00e4lt sich M danach wie Q angesetzt auf y .

F\u00fcr die von M berechnete Funktion g gilt:

$$g = \begin{cases} \Omega & \text{falls } M_w \text{ auf dem leeren Band nicht stoppt} \\ q & \text{sonst} \end{cases}$$

Betrachte: $f : w \mapsto \text{Code von } M$
 f ist total und berechenbar.

Es gilt:

$w \in H_0 \Rightarrow M_w$ stoppt angesetzt auf dem leeren Band.
 $\Rightarrow M$ berechnet q .
 \Rightarrow die von $M_{f(w)}$ berechnete Funktion liegt nicht in \mathcal{S} .
 $\Rightarrow f(w) \notin C(\mathcal{S})$

$w \notin H_0 \Rightarrow M_w$ stoppt angesetzt auf dem leeren Band nicht.
 $\Rightarrow M$ berechnet Ω .
 \Rightarrow die von $M_{f(w)}$ berechnete Funktion liegt in \mathcal{S} .
 $\Rightarrow f(w) \in C(\mathcal{S})$

d.h.: f vermittelt eine Reduktion :

$$\overline{H_0} \leq C(\mathcal{S})$$

wegen H_0 unentscheidbar

$\Rightarrow \overline{H_0}$ unentscheidbar

$\Rightarrow C(\mathcal{S})$ unentscheidbar

2. Fall:

Man zeigt analog $H_0 \leq C(\mathcal{S})$ ■

Anwendung:

Betrachte: $\mathcal{S} = \{f \in \mathcal{R} \mid f \text{ ist konstant}\}$

$\Rightarrow_{\text{Satz von Rice}} C(\mathcal{S}) = \{w \mid M_w \text{ berechnet eine konstante Funktion}\}$ ist nicht entscheidbar.

Es gibt verschiedene Klassen der "Unlösbarkeit":

Betrachte: Das Äquivalenzproblem für TM :

$$\ddot{A} = \{u \# w \mid M_w \text{ berechnet dieselbe Funktion wie } M_u\}$$

Es gilt:

$H \leq \ddot{A}$ aber nicht $\ddot{A} \leq H$

Man kann unendlich lange Folgen von Problemen A_1, A_2, \dots konstruieren mit

$$A_i \leq A_{i+1} \text{ aber nicht } A_{i+1} \leq A_i$$

1.9 Das Postsche Korrespondenz-Problem

Definition.

Das nachfolgend beschriebene Problem heißt *Postsches Korrespondenz-Problem (PCP)*.

gegeben: Eine endliche Folgen $(x_1, y_1), \dots, (x_k, y_k)$ von Wortpaaren mit $x_i, y_i \in A^+$ (A endliche Alphabet)

gefragt: Gibt es eine Folge von Indizes $i_1, \dots, i_n \in \{1, \dots, k\}$, $n \geq 1$, mit $x_{i_1}, \dots, x_{i_n} = y_{i_1}, \dots, y_{i_n}$?

Beispiel.

Das Postsche Korrespondenz-Problem

$$K = ((1, 101), (10, 00), (011, 11)),$$

also

$$\begin{array}{lll} x_1 = 1 & x_2 = 10 & x_3 = 011 \\ y_1 = 101 & y_2 = 00 & y_3 = 11 \end{array}$$

besitzt die Lösung $(1,3,2,3)$, denn es gilt:

$$x_1 x_3 x_2 x_3 = 101110011 = y_1 y_3 y_2 y_3$$

PCP ist schwer.

z.B.: $K = ((001, 0), (01, 011), (01, 101), (10, 001))$

besitzt eine Lösung, aber die kürzeste besteht aus 66 Indizes!

PCP ist semi-entscheidbar:

Probieren alle möglichen Indexfolgen mit zunehmender Länge durch.

Satz.

Das Postsche Korrespondenz-Problem ist nicht entscheidbar.

Beweis.

Betrachte modifiziertes PCP:

MPCP

gegeben: wie bei PCP

gefragt: gibt es eine Lösung i_1, \dots, i_n mit $i_1 = 1$

Lemma

$\text{MPCP} \leq \text{PCP}$

Beweis.

Sei $\#$ ein neues Symbol.

Für $w = a_1 \dots a_m \in A^+$ sei

$$\bar{w} = \#a_1\#a_2\#\dots\#a_m\#$$

$$\begin{aligned}\dot{w} &= \#a_1\#a_2\#\dots\#a_m \\ \acute{w} &= a_1\#a_2\#\dots\#a_m\#\end{aligned}$$

Sei $K = ((x_1, y_1), \dots, (x_k, y_k))$ die Eingabe von MPCP.
 $\rightarrow f(K) = ((\bar{x}_1, \dot{y}_1), (x'_1, \dot{y}_1), (x'_2, \dot{y}_2), \dots, (x'_k, \dot{y}_k), (\$, \#\$))$
 f ist berechenbar.
 f vermittelt eine Reduktion von MPCP auf PCP.

Behauptung.

K besitzt eine Lösung mit $i_1 = 1$
 $\Leftrightarrow f(K)$ besitzt irgend eine Lösung.

Beweis.

- (\rightarrow) Besitzt K eine Lösung (i_1, \dots, i_n) mit $i_1 = 1$,
dann ist $(1, i_2 + 1, \dots, i_n + 1, k + 2)$ eine Lösung für $f(K)$.
(\leftarrow) Besitzt $f(K)$ eine Lösung (i_1, \dots, i_n) ,
so muß gelten:
 $i_1 = 1, i_n = k + 2$ mit $i_j \in \{2, \dots, k + 1\}$ für $2 \leq j \leq n + 1$
 $\Rightarrow (1, i_2 - 1, \dots, i_{n-1} - 1)$ ist Lösung für K .

Zur Unentscheidbarkeit von MPCP:

Lemma

$H \leq MPCP$

Beweis.

Sei $M = (Q, \Sigma, \Gamma, \delta, z_0, \square, F)$ eine kodierte TM

$w \in \Sigma^+$ die Eingabe.

Suche eine allgemeine Vorschrift, die (M, w) eine Folge $(x_1, y_1), \dots, (x_n, y_n)$ zuordnet mit

M angesetzt auf w stoppt $\Leftrightarrow (x_1, y_1), \dots, (x_n, y_n)$ besitzt eine Lösung mit $i_1 = 1$

Konstruktion von MPCP über dem Alphabet $\Gamma \cup Q \cup \{\#\}$.

erstes Wortpaar: $(\#, \#q_0w\#)$

weitere Paare:

1. Kopiereregeln:

$$(a, a) \text{ für alle } a \in \Gamma \cup \{\#\}$$

2. Überführungsregeln:

$$(qa, q'c) \text{ falls } \delta(q, a) = (q', c, N)$$

$$(qa, cq') \text{ falls } \delta(q, a) = (q', c, R)$$

- $(qa, q'bc)$ falls $\delta(q, a) = (q', c, L)$ für alle $b \in \Gamma$
- $(\#qa, \#q'\square c)$ falls $\delta(q, a) = (q', c, L)$
- $(q\#, q'c\#)$ falls $\delta(q, \square) = (q', c, N)$
- $(q\#, cq'\#)$ falls $\delta(q, \square) = (q', c, R)$
- $(bq\#, q'b\#)$ falls $\delta(q, \square) = (q', c, L)$ für alle $b \in \Gamma$

3. Löseregeln:

$$(aq_e, q_e) \text{ und } (q_e a, q_e) \text{ für alle } a \in \Gamma, q_e \in F$$

4. Abschlußregeln:

$$(q_e \#\#\#, \#) \text{ für alle } q_e \in F$$

- (\rightarrow) Falls M bei w stoppt gibt es eine Folge von Konfigurationen (k_0, k_1, \dots, k_t) mit $k_0 = q_0 w$, k_t ist Endkonfiguration also $k_t = uq_e v$ mit $u, v \in A^*$, $q_e \in F$) und $k_i \vdash k_{i+1}$
 \Leftrightarrow Die obige Eingabe für das MPCP besitzt ein Lösungswort

$$\#k_0\#k_1 \dots \#k_t\#k'_t\#k''_t \dots \#q_e\#\#$$

(k'_t, k''_t, \dots entstehen aus $k_t = uq_e v$ durch Löschung von Nachbarsymbolen aus q_e)

- (\leftarrow) Besitzt der obige MPCP eine Lösung mit $i_1 = 1$, dann läßt sich eine stoppende Rechnung von M bei w ablesen. ■

Folgerung

PCP bleibt unentscheidbar, falls für das Alphabet A gilt: $A = \{0, 1\}$. ("01-PCP")

Beweis.

Es reicht zu zeigen: $\text{PCP} \leq \text{01-PCP}$.

Sei A das Alphabet von PCP.

Betrachte: $f : A \rightarrow \{0, 1\}^*$ mit $f(a_r) = \hat{a}_r = 01^r$.

Setze die Abbildung f auf A^* fort.

Es gilt:

$(x_1, y_1), \dots, (x_n, y_n)$ besitzt Lösung $\Leftrightarrow (\hat{x}_1, \hat{y}_1), \dots, (\hat{x}_n, \hat{y}_n)$ besitzt Lösung. ■

Bemerkung

Sei PCP_k die Variante des PCPs, deren Eingabe aus genau k Wortpaaren besteht.

Es gilt:

PCP_k ist unentscheidbar für $k \geq 9$.

PCP_k ist entscheidbar für $k \leq 2$.

offen sonst.

Folgerung

Das Halte-Problem H für TM ist semi-entscheidbar.

Beweis.

PCP ist semi-entscheidbar,

$H \leq \text{PCP}$.

\Rightarrow Beh. ■

Universelle TM: Nachobiger Folgerung gibt es eine TM U , die sich bei Eingabe von $w\#x$ so verhält wie M_w bei Eingabe von x . (Zunächst nur im Bezug auf Halten, bzw. Nicht-Halten)

TM U verhält sich wie ein TM-Interpreter

Der erste Teil programmiert die TM.

Der zweite Teil ist die eigentliche Eingabe.

Man kann mit Hilfe des PCP-Resultats die Unentscheidbarkeit von Problemen der Theorie der formalen Sprachen nachweisen:

Satz.

Das Schnittproblem für kontextfreie Sprachen (G_1, G_2 kontextfreie Sprachen.

Gilt: $L(G_1) \cap L(G_2) = \emptyset$?) ist unentscheidbar.

Beweis.

Zu zeigen: $\text{PCP} \leq \text{Schnittproblem}$

Jedem PCP $K = ((x_1, y_1), \dots, (x_k, y_k))$ müssen effektiv zwei Grammatiken G_1, G_2 zugeordnet werden,

so daß gilt:

K besitzt eine Lösung

\Leftrightarrow es gibt ein $w \in L(G_1) \cap L(G_2)$

Idee: Erzeugung von Folgen von x_i -Wörtern mit G_1 .

Erzeugung von Folgen von y_i -Wörtern mit G_2 .

Die Indizes der x_i müssen mit denen der y_i in allen potentiellen Lösungen übereinstimmen.

Sei $G_i = (\{S_i\}, A \cup \{a_1, \dots, a_k\}, P_i, S_i)$ $i = 1, 2$

mit

$$P_1 = \{S_1 \rightarrow a_1 x_1 | \dots | a_k x_k\} \cup \{S_1 \rightarrow a_1 S_1 x_1 | \dots | a_k S_1 x_k\}$$

$$P_2 = \{S_2 \rightarrow a_1 y_1 | \dots | a_k y_k\} \cup \{S_2 \rightarrow a_1 S_2 y_1 | \dots | a_k S_2 y_k\}$$

Es gilt:

K besitzt die Lösung i_1, \dots, i_n

\Leftrightarrow

$$a_{i_n} \dots a_{i_2} a_{i_1} x_{i_1} x_{i_2} \dots x_{i_n} =$$

$$a_{i_n} \dots a_{i_2} a_{i_1} y_{i_1} y_{i_2} \dots y_{i_n} \in L(G_1) \cap L(G_2) \quad \blacksquare$$

Folgerung

Das Schnittproblem für deterministisch kontextfreie Sprachen ist unentscheidbar.

Beweis.

G_1, G_2 aus dem Beweis des letzten Satz es sind deterministisch. \blacksquare

Satz.

Das Äquivalenzproblem für kontextfreie Sprachen ist unentscheidbar.

Beweis.

Reduzierung des Schnittproblems für kontextfreie Sprachen auf das Äquivalenzproblem für Kontextfreie Sprachen.

Nutzung von

- deterministisch kontextfreie Sprachen sind effektiv unter Komplementbildung abgeschlossen.
- deterministisch kontextfreie Sprachen sind effektiv unter Vereinigung abgeschlossen.

Es gilt: (G_1, G_2) Schnittproblem

$$\Leftrightarrow L(G_1) \cap L(G_2) = \emptyset$$

$$\Leftrightarrow L(G_1) \subseteq L(G_2)$$

$$\Leftrightarrow L(G_1) \subseteq L(G'_2) \quad G'_2 \quad \text{Grammatik des Komplements von } G_2$$

$$\Leftrightarrow L(G_1) \cup L(G_2) = L(G'_2)$$

$$\Leftrightarrow L(G_3) = L(G_2) \quad G_3 \quad \text{Vereinigungsgrammatik von } G_1 \text{ und } G_2$$

$$\Leftrightarrow (G_3, G'_2) \in \text{Äquivalenzproblem} \quad \blacksquare$$

noch offen: Das Äquivalenzproblem für deterministisch kontextfreie Grammatiken.

Folgerung

Das Äquivalenzproblem für folgende Probleme ist unentscheidbar:

- nichtdeterministische Kellerautomaten
- BNF
- EBNF
- Syntaxdiagramme

- LBA
- kontextsensitive Grammatiken
- TM

Beweis.

Man kann kontextfreie Grammatiken effektiv in die genannten Probleme übersetzen. ■

Satz.

Das Leerheitsproblem für kontextsensitive Sprachen ist unentscheidbar.

Beweis.

Das Schnittproblem für kontextfreie Sprachen ist entscheidbar.

$\Leftrightarrow (G_1, G_2) \mapsto G_3$ ist berechenbar, wobei

$$L(G_3) = L(G_1) \cap L(G_2)$$

Diese Abbildung vermittelt die Reduktion des Schnittproblems für kontextfreie Sprachen auf das Leerheitsproblem für kontextsensitive Sprachen. ■