Universität Trier

Fachbereich IV - Informatik

Christoph Meinel:

Berechenbarkeit und Komplexität

Vorlesungsskript SS 1994

Inhaltsverzeichnis

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

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

1. Der Algorithmus:

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

   "berechnet" die total undefinierte Funktion $\Omega : n \to undef$.

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!

<u>1.Fall:</u> In π kommen beliebig lange 0–Folgen vor.

$\Rightarrow h(n) = 1$ für alle n

<u>2.Fall:</u> 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 <u>nicht</u> 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.

<u>Beweis.</u>
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 $\mathbb{Q}$ 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.

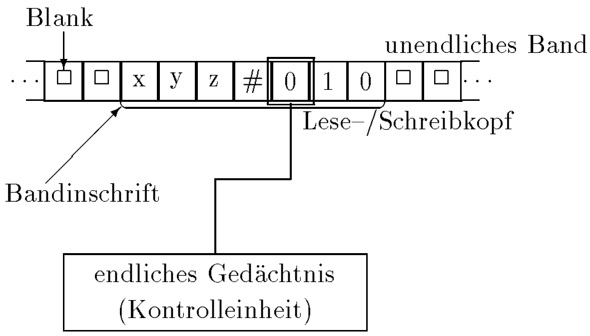
$\Rightarrow$ **Churchsche These:**
Die durch den formalen Begriff der Turing–Berechenbarkeit (äquivalent: WHILE–Berechenbarkeit, GOTO–Berechenbarkeit, μ –Rekursivität) erfaßte Klasse von Funktionen stimmt genau mit der Klasse der intuitiv berechenbaren Funktionen überein.

Die Churchsche These ist nicht beweisbar, da der intuitive Begriff der Berechenbarkeit <u>nicht</u> formal faßbar ist.
Sie ist allgemein akzeptiert.

Man kann auf Basis der Churchschen These zeigen, daß bestimmte Funktionen <u>nicht</u> 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

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

mit

Q endliche Menge von “Zuständen”
Σ endliche Menge von “Eingabesymbolen” (Eingabealphabet)
$\Gamma \supset \Sigma$ endliche Menge von “Bandsymbolen” (Arbeitsalphabet)

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

$q_0 \in Q$ “Startzustand”
$\Box \in \Gamma - \Sigma$ “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 $m \in \{L(inks), R(echts), N(ichts)\}$ aus.

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

Konfigarationen 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 $w \in \Sigma^* : q_0 w$

$$\begin{array}{c}\Box \ldots \Box w \Box \ldots \Box \\ \uparrow\end{array}$$

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:

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

$\vdash^*$ bezeichne den transitiven Abschluß von $\vdash$

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

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

mit

$$\begin{aligned} \delta(q_0, 0) &= (q_0, 0, R) \\ \delta(q_0, 1) &= (q_0, 1, R) \\ \delta(q_0, \Box) &= (q_1, \Box, L) \\ \\ \delta(q_1, 0) &= (q_2, 1, L) \end{aligned}$$

$$\begin{aligned}
\delta(q_1, 1) &= (q_1, 0, L) \\
\delta(q_1, \Box) &= (q_e, 1, N) \\
\\
\delta(q_2, 0) &= (q_2, 0, L) \\
\delta(q_2, 1) &= (q_2, 1, L) \\
\delta(q_2, \Box) &= (q_e, \Box, R)
\end{aligned}$$

Beispiel.

w=101

$$q_0 \vdash 1q_001 \vdash 10q_01 \vdash 101q_0\Box \vdash 10q_1\Box \vdash 1q_100\Box \vdash q_2110\Box \vdash q_2\Box110\Box \vdash \Box q_e110$$

Definition.
Eine Funktion $f : I\!N^* \to I\!N$ heißt *Turing–berechenbar*, falls es eine (deterministische) TM M gibt, so daß für alle $n_1, \ldots, n_k, m \in I\!N$ gilt:

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

$\Longleftrightarrow$

$q_0 bin(n_1)\#bin(n_2)\# \ldots \#bin(n_k) \vdash^* q_e bin(m)$

wobei $q_e \in E$.

($bin(n)$ bezeichnet die Binärdarstellung von $n \in I\!N$ ohne führende Nullen)

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

$$f(x) = y$$

$\Longleftrightarrow$

$q_0 x \vdash^* q_e y$

wobei $q_e \in E$.

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

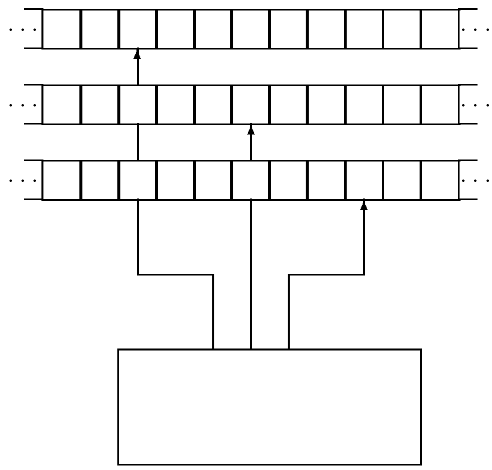
Beispiel.

1. $f : n \to n + 1$ 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:



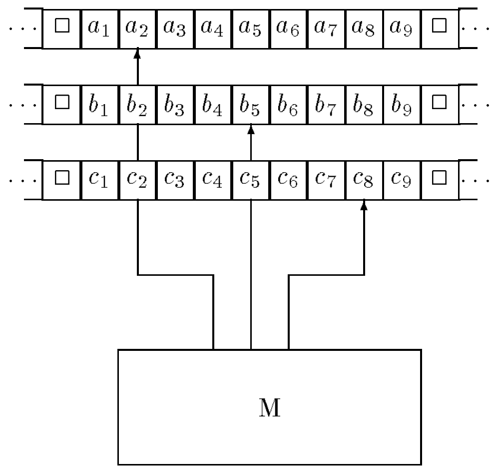
Die Lese-/Schreibköpfe können sich unabhängig voneinander bewegen.

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

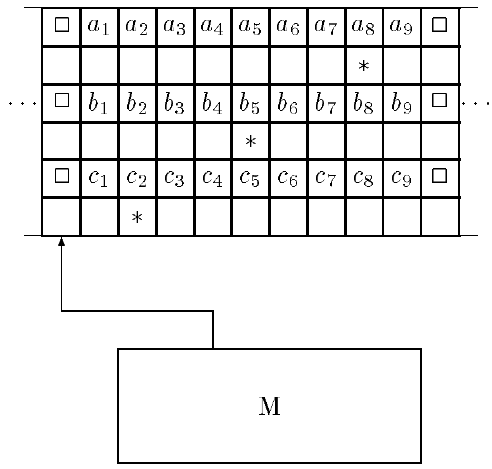
Satz.
Zu jeder Mehrband-TM M gibt es eine 1-Band TM M' die dieselbe Funktion berechnet wie M.

<u>Beweis.</u>
Sei k die Anzahl der Bänder, Γ Arbeitsalphabet von M.

<u>Idee:</u>



wird simuliert durch:



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

M' simuliert M wie folgt:
Gestartet mit Eingabe $x_1, \ldots, x_n \in \Gamma^*$ 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 $M_i = (Q_i, \Sigma, \Gamma_i, \delta_i, q_i, \Box, F_i)$; $i = 1, 2$; $Q_1 \cap Q_2 = \emptyset$

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

  bzw.

  $$M_1 ; M_2$$

  bezeichnet die TM :

  $$M = (Q_1 \cup Q_2, \Sigma, \Gamma_1 \cup \Gamma_2, \delta, q_1, \Box, F_2)$$
  $$\delta = \delta_1 \cup \delta_2 \cup \{(q_e, a, q_2, a, N) : q_e \in F_1,\ a \in \Gamma_1)$$

Beispiel.

- $$\begin{tikzcd} \text{start} \arrow[d] \\ \text{“Band:=Band+1”} \arrow[d] \\ \text{“Band:=Band+1”} \arrow[d] \\ \text{“Band:=Band+1”} \arrow[r] & \text{stop} \end{tikzcd}$$

  bezeichnet die TM , die 3 hinzuaddiert.

- $$\begin{tikzcd} \text{start} \arrow[r] & \text{M} \arrow[r, "q_{e_1}"] \arrow[d, "q_{e_2}"] & M_1 \arrow[r] & \text{stop} \\ & M_2 \arrow[d] & & \\ & \text{stop} & & \end{tikzcd}$$

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

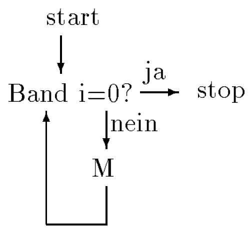
- Betrachte: Die TM “Band=0?”:

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

$$\begin{aligned}
\delta(q_0, a) &= (\text{nein}, a, N) \ \mathit{f\ddot{u}r}\ a \neq \Box \\
\delta(q_0, 0) &= (q1, 0, R) \\
\delta(q_1, a) &= (\text{nein}, a, L) \ \mathit{f\ddot{u}r}\ a \neq \Box \\
\delta(q_1, \Box) &= (\text{ja}, \Box, L)
\end{aligned}$$

Schreibe “Band i=0?” anstelle von “Band=0?”

- "WHILE Band i≠0 DO M"



1.4 LOOP-, WHILE- und GOTO–Berechenbarkeit

Die einfache Programmiersprache LOOP:

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

Induktive Definition der Syntax von LOOP:

- Wertzuweisung

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

  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 $(n_1, \ldots, n_k)$ 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 $f : \mathbb{N}^k \rightarrow \mathbb{N}$ heißt *LOOP–berechenbar*, falls es ein LOOP–Programm P gibt, das gestartet mit $n_1, \ldots, n_k$ in den Variablen $x_1, \ldots, x_k$ mit dem Wert $f(n_1, \ldots, n_k)$ 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 DIV z) + (x 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 $n_1, \ldots, n_k$ in $x_1, \ldots, x_k$ (0 sonst) mit dem Wert $f(n_1, \ldots, n_k)$ 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

$M_1 : A_1$; $M_2 : A_2$; ... ; $M_k : A_k$

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

Stopanweisung: HALT

Vereinbarung: Marken von nicht angesprungen Anweisungen werden weggelassen.
Die Semantik ist klar.

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

Satz.
Jedes WHILE–Programm kann durch ein GOTO–Programm simuliert werden.

<u>Beweis.</u>
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.

<u>Beweis.</u>
Gegeben sei ein GOTO–Programm

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

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

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

■

Korollar
Eine Funktion $f : \mathbb{N}^k \to \mathbb{N}$ ist GOTO–berechenbar, falls f WHILE–berechenbar ist.

Satz. *(Kleenesche Normalformsatz für WHILE–Programme)*
Jede WHILE–berechenbare Funktion $f : \mathbb{N}^k \to \mathbb{N}$ 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:

$$\begin{tikzcd} \text{GOTO} \arrow[r, shift left] & \text{WHILE} \arrow[l, shift left] \arrow[r] & \text{TM} \\ & \text{LOOP} \arrow[u] & \end{tikzcd}$$

Satz.
GOTO–Programme können TM simulieren.

Beweis.
Sei $M = (Q, \Sigma, \Gamma, \delta, q_1, \Box, F)$ 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

$$Q = \{q_1, \ldots, q_k\}$$
$$\Gamma = \{a_1, \ldots, a_m\} \ , \ b > \#\Gamma$$

Kodierung der TM –Konfiguration:

$$a_{i_1} \ldots a_{i_p} q_l a_{j_1} \ldots a j_q$$

durch die folgenden Werte von x, y, z:

$$x = (i_1 \ldots i_p)_b$$
$$y = (j_q \ldots j_1)_b$$
$$z = l$$

GOTO–Programmstück $M_2 : P_2$:

M_2: $a := y$ MOD b;
  IF $(z = 1)$ AND $(a = 1)$ THEN GOTO M_{11}
  IF $(z = 1)$ AND $(a = 2)$ THEN GOTO M_{12}
  $\vdots$
  IF $(z = k)$ AND $(a = m)$ THEN GOTO M_{km}
M_{11}: $\otimes$
  GOTO M_2
M_{12}: $\otimes$
  GOTO M_2
$\vdots$
M_{km}: $\otimes$
  GOTO M_2

wobei $\otimes$ bedeutet:
Man nimmt das Programmstück das mit M_{ij} markiert ist

Sei $\delta(q_i, a_j) = (q_i, a_j, L)$

und simuliert den Übergang durch:

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

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.

   $pr_i : \mathbb{N}^k \to \mathbb{N}$
   $pr_i(n_1, \dots, n_k) = n_i$
3. Die Nachfolgerfunktion ist primitiv rekursiv, d.h.

   $s : \mathbb{N}^k \to \mathbb{N}$
   $i \mapsto i + 1$
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:
   Seinen 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:
   $add : \mathbb{N}^2 \to \mathbb{N}$ mit $add(x, y) = x + y$.
   Es gilt:

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

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

2. Multiplikation:
   $mult: \mathbb{N}^2 \to \mathbb{N}$ 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 $f : \mathbb{N}^{k+1} \to \mathbb{N}$
Durch Anwendung des μ–Operators entsteht aus f die Funktion $g : \mathbb{N}^k \to \mathbb{N}$ mit

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

mit min $\emptyset$ = undefiniert.

Durch die Anwendung des μ–Operators können partielle Funktionen entstehen.

Beispiel.
Sei $f(x,y) \equiv 1$
Dann entsteht durch die Anwendung des μ–Operators die vollständig undefinierte Funktion Ω.

Defintion
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.

<u>ohne</u> 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

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

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.

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

z.B.:

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

Betrachte eine ähnlich definierte Funktion a:

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

- 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 $f_P : \mathbb{N} \to \mathbb{N}$ 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 $x_i := x_j \pm c$:

$$\begin{aligned} \Rightarrow f_P(n) &\leq n + n + c = 2n + c \\ \Rightarrow f_P(n) &< 3n + 1 = a(1, n) \leq a(k, n) \text{ für alle } k \geq 1,\ n \geq c \end{aligned}$$

  Wähle k:=c+1.

- Habe P die Form $P_1; P_2$:

  Nach Vorraussetzung gibt es k_1, k_2 mit

$$\begin{aligned} f_{P_1} &< a(k_1, n) \\ f_{P_2} &< a(k_2, n) \text{ für alle} n \geq max\{k_1, k_2\} \end{aligned}$$

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

  Es gilt:

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

  Wähle k :=max$\{k_3, 2\}$

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

  Nach Induktionvorraussetzung gibt es ein k' mit

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

Es gilt:

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

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: $c : \mathbb{N} \to \mathbb{N}$ 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:

Sei k max. mit $2^k < n$
$\Rightarrow \quad x := n - 2^k$
$\quad\quad y := k - x$
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 $(n_1, \ldots, n_k)$ der Inhalt des Stacks,
n_1 das Head–Element.

Kodierung von $(n_1, \ldots, n_k)$ mit Hilfe von c:
$n := c(n1, c(n2, \ldots, c(n_k, 0) \ldots)$

INIT($stack$)	wird simuliert durch	$n := 0$
PUSH($a, stack$)	wird simuliert durch	$n := c(a, n)$
POP($stack$)	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* $\chi_A : \Sigma^* \to \{0,1\}$

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

berechenbar ist.

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

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

berechenbar ist.

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

- Das Semi–Entscheidungsproplem 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,...DO
   IF M1 bei Eingabe x in s Schritten stoppt
     THEN OUTPUT(1) END;
   IF M2 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 $f : I\!N \rightarrow \Sigma^*$ gibt mit

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

"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,...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 $f(I\!N)$=A mittels folgendem Algorithmus:

```
   INPUT(n);
   * Interpretiere n als Kodierung n = c(x,y)
   mit x = c1(n), y = c2(n)*
   x := c1(n);
   y := c2(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 $\overline{A}$ 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

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

- 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 $A' = \{g(0), g(1), \ldots\}$ ■

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 Halteproplem für TM .
  Dazu Kodierung der TM $M = (Q, \Sigma, \Gamma, \delta, q_0, \Box, F)$ als Wort über {0,1}

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

   Sei $Q = \{q_0, \ldots, q_n\}$
   $\Gamma = \{a_0, \ldots a_k\}$

   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

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

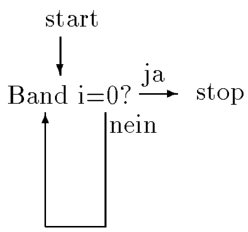
Definiton
Die folgende Sprache

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

heißt *spezielles Halte–Problem.*

Satz.
Das spezielle Halte–Problem ist nicht entscheidbar.

<u>Beweis.</u>
Annahme: K ist entscheidbar.
$\Leftrightarrow \chi_K$ ist berechenbar mittles 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 $f : \Sigma^* \to \Sigma^*$ 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.

<u>Beweis.</u>
(i) Sei $A \leq B$ mittels f

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

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

$\Leftrightarrow \chi_A$ 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.

$f : w\#x \rightarrow$ Code von M.
f ist berechenbar Man kann f zu einer totalen und berechenbaren Funktion erweitern.

Es gilt:

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

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

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

unentscheidbar.

Beweis.
Sei $\Omega \in \mathcal{R}$ eine überall 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 zunächst und verhält sich wie M_w angesetzt auf das leere Band.
> Falls diese Rechnung zu Ende kommt, so verhält sich M danach wie Q angesetzt auf y.

Für 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: $f : w \mapsto$ 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

<u>2. Fall:</u>
Man zeigt analog $H_0 \leq C(\mathcal{S})$ ■

<u>Anwendung:</u>
Betrachte: $\mathcal{S} = \{f \in \mathcal{R} \mid f \text{ ist konstant}\}$
$\Rightarrow_{Satz von Rice} C(\mathcal{S}) = \{w \mid M_w$ berechnet eine konstante Funktion$\}$ ist <u>nicht</u> 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 <u>nicht</u> $\ddot{A} \leq H$

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

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

1.9 Das Postsche Korrespondenz–Problem

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

gegeben:	Eine endliche Folgen $(x_1, y_1), \ldots, (x_k, y_k)$ von Wortpaaren mit $x_i, y_i \in A^+$ (A endliche Alphabet)
gefragt:	Gibt es eine Folge von Indizes $i_1, \ldots, i_n \in \{1, \ldots, k\},\ n \geq 1$, mit $x_{i_1}, \ldots, x_{i_n} = y_{i_1}, \ldots, y_{i_n}$?

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:
Probiere 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 $i_1, \ldots, i_n$ mit $i_1 = 1$

Lemma
MPCP $\leq$ PCP

Beweis.
Sei # ein neues Symbol.
Für $w = a_1 \ldots a_m \in A^+$ sei

$$\bar{w} \quad = \quad \#a_1\#a_2\#\ldots\#a_m\#$$

$$\begin{aligned}\grave{w} &= \#a_1\#a_2\#\ldots\#a_m \\ \acute{w} &= a_1\#a_2\#\ldots\#a_m\#\end{aligned}$$

Sei $K = ((x_1, y_1), \ldots, (x_k, y_k))$ die Eingabe von MPCP.
$\rightarrow f(K) = ((\bar{x_1}, \grave{y_1}), (\acute{x_1}, \grave{y_1}), (\acute{x_2}, \grave{y_2}), \ldots, (\acute{x_k}, \grave{y_k}), (\$, \#\$))$
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 $(i_1, \ldots, i_n)$ mit $i_1 = 1$,
dann ist $(1, i_2 + 1, \ldots, i_n + 1, k + 2)$ eine Lösung für $f(K)$.
($\leftarrow$) Besitzt $f(K)$ eine Lösung $(i_1, \ldots, i_n)$,
so muß gelten:
$i_1 = 1, i_n = k + 2$ mit $i_j \in \{2, \ldots, k + 1\}$ für $2 \leq j \leq n + 1$
$\Rightarrow (1, i_2 - 1, \ldots, i_{n-1} - 1)$ ist Lösung für K.

Zur Unentscheidbarkeit von MPCP:

Lemma
$H \leq MPCP$

Beweis.
Sei $M = (Q, \Sigma, \Gamma, \delta, z_0, \Box, F)$ eine kodierte TM
$w \in \Sigma^+$ die Eingabe.
Suche eine allgemeine Vorschrift, die (M, w) eine Folge $(x_1, y_1), \ldots, (x_n, y_n)$ zuordnet mit
M angesetzt auf w stoppt $\Leftrightarrow (x_1, y_1), \ldots, (x_n, y_n)$ besitzt eine Lösung mit $i_1 = 1$

Konstruktion von MPCP über dem Alphabet $\Gamma \cup Q \cup \{\#\}$.
erstes Wortpaar: $(\#, \#q_0w\#)$
weitere Paare:

1. Kopieregeln:

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

2. Überführungsregeln:

$$\begin{array}{lll}(qa, q'c) & \text{falls} & \delta(q,a) = (q', c, N) \\ (qa, cq') & \text{falls} & \delta(q,a) = (q', c, R)\end{array}$$

$$\begin{array}{rll}
(qa, q'bc) & \text{falls} & \delta(q,a) = (q',c,L) \text{ fr alle } b \in \Gamma \\
(\#qa, \#q'\Box c) & \text{falls} & \delta(q,a) = (q',c,L) \\
(q\#, q'c\#) & \text{falls} & \delta(q,\Box) = (q',c,N) \\
(q\#, cq'\#) & \text{falls} & \delta(q,\Box) = (q',c,R) \\
(bq\#, q'b\#) & \text{falls} & \delta(q,\Box) = (q',c,L) \text{ für alle } b \in \Gamma
\end{array}$$

3. Löschregeln:

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

4. Abschlußregeln:

   $(q_e\#\#, \#)$ für alle $q_e \in F$

($\rightarrow$) Falls M bei w stoppt gibt es eine Folge von
Konfigurationen $(k_0, k_1, \ldots, k_t)$ 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

$$\#k_0\#k_1\ldots\#k_t\#k_t'\#k_t''\ldots\#q_e\#\#$$

$(k_t', k_t'', \ldots$ 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")

<u>Beweis.</u>
Es reicht zu zeigen: PCP $\leq$ 01–PCP.
Sei A das Alphabet von PCP.
<u>Betrachte:</u> $f : A \rightarrow \{0,1\}^*$ mit $f(a_r) = \hat{a_r} = 01^r$.
Setze die Abbildung f auf A^* fort.
Es gilt:
$(x_1, y_1), \ldots, (x_n.y_n)$ besitzt Lösung $\Leftrightarrow$ $(\hat{x_1}, \hat{y_1}), \ldots, (\hat{x_n}.\hat{y_n})$ 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,
$H \leq$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: PCP $\leq$ Schnittproblem
Jedem PCP $K = ((x_1, y_1), \ldots, (x_k, y_k))$ 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 $G_i = (\{S_i\}, A \cup \{a_1, \ldots, a_k\}, P_i, S_i)$ $i = 1, 2$
mit

$$\begin{aligned} P_1 &= \{S_1 \to a_1 x_1 | \ldots | a_k x_k\} \cup \{S_1 \to a_1 S_1 x_1 | \ldots | a_k S_1 x_k\} \\ P_2 &= \{S_2 \to a_1 y_1 | \ldots | a_k y_k\} \cup \{S_2 \to a_1 S_2 y_1 | \ldots | a_k S_2 y_k\} \end{aligned}$$

Es gilt:
K besitzt die Lösung $i_1, \ldots, i_n$

$\Leftrightarrow$
$a_{i_n} \dots a_{i_2} a_{i_1} x_{i_1} x_{i_2} \dots x_{i_n} =$
$a_{i_n} \dots a_{i_2} a_{i_1} y_{i_1} y_{i_2} \dots y_{i_n} \in L(G_1) \cap L(G_2)$ ■

Folgerung
Das Schnittproblem für deterministisch kontextfreie Sprachen ist unentscheidbar.

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

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
$\Leftrightarrow L(G_1) \cap L(G_2) = \emptyset$
$\Leftrightarrow L(G_1) \subseteq \overline{L(G_2)}$
$\Leftrightarrow L(G_1) \subseteq L(G_2')$ G_2' Grammatik des Komplements von G_2
$\Leftrightarrow L(G_1) \cup L(G_2) = L(G_2')$
$\Leftrightarrow L(g_3) = L(G_2)$ G_3 Vereinigungsgrammatik von G_1 und G_2
$\Leftrightarrow (G_3, G_2') \in$ Äquivalenzproblem ■

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

1.10 Der Gödelsche Unvollständigkeitssatz

- Jedes Beweissystem für die wahren arithmetischen Formeln ist notwendigerweise unvollständig.
- Es gibt immer Formeln, die wahr sind, was aber nicht bewiesen werden können.

Definition von arithmetischen Formeln:

Definition.
(Arithmetische) Terme sind induktiv wie folgt definiert:

1. Jedes $n \in I\!N$ und jede Variable x_i, $i \in I\!N$ ist ein Term.
2. Sind t_1, t_2 Terme, so sind auch $(t_1 + t_2)$ und $(t_1 * t_2)$ Terme.

(Arithmetische) Formeln sind wie folgt definiert:

1. Sind t_1, t_2 Terme, dann ist $t_1 = t_2$ eine Formel.
2. Sind F, G Formeln, dann auch $\neg F$, $F \vee G$ und $F \wedge G$.
3. Ist x eine Variable und F eine Formel, dann sind $\exists x F$ und $\forall x F$ Formeln.

Definition.
Eine Variable heißt *gebunden*, wenn sie in Wirkungsbereich eines Quantoren steht.
Sonst heißt eine Variable *frei*.

Beispiel
$\exists x \exists y \exists z (((x * y) = z) \vee ((1 + x) = (x * y)))$

Festlegung der semantischen Interpretation über $I\!N$:

Sei $\phi : V \to I\!N$ "Belegung"
Erweiterung von ϕ auf Terme:

$$\begin{aligned} \phi(n) &= n \\ \phi(t_1 + t_2) &= \phi(t_1) + \phi(t_2) \\ \phi(t_1 * t_2) &= \phi(t_1) + \phi(t_2) \end{aligned}$$

Beispiel

$$\begin{aligned} \phi \quad : \quad & x \to 10 \\ & y \to 11 \end{aligned}$$

$\phi((x + (5 * y))) = 10 + 5 * 11 = 65$

$F(x/n)$ bezeichne die Formel, die aus F entsteht, indem alle freien Vorkommen von x durch die Konstante n ersetzt werden.

Definition.
Eine arithmetische Formel "F ist wahr":
$(t_1 = t_2)$ ist wahr, falls $\phi(t_1) = \phi(t_2)$ für alle Belegungen ϕ.
$\neg F$ ist wahr, falls F nicht wahr ist.
$(F \wedge G)$ ist wahr, falls F wahr und G wahr ist.
$(F \vee G)$ ist wahr, falls F wahr oder G wahr ist.
$\exists x F$ ist wahr, falls es ein $n \in \mathbb{N}$ gibt, so daß $F(x/n)$ wahr ist.
$\forall x F$ ist wahr, falls für alle $n \in \mathbb{N}$ gilt, daß $F(x/n)$ wahr ist.

Beispiel
$\forall x \exists y((x + y) = (y * x))$ ist (über $\mathbb{N}$) nicht wahr,
da z.B. für $x = 7$ kein y existiert mit $x + y = x * y$.

$\forall x \exists y((x + (x * y)) = (x * (1 + y)))$ ist wahr,
da gilt: $x + x * y = x * 1 + x * y$

Definiton
Eine Funktion $f : \mathbb{N}^k \rightarrow \mathbb{N}$ ist *arithmetisch repräsentierbar*, falls es eine arithmetische Formel $F(x_1, \ldots, x_k, y)$ gibt, so daß für alle $n_1, \ldots, n_k, m \in \mathbb{N}$ gilt:
$f(n_1, \ldots, n_k) = m \Leftrightarrow F(x_1/n_1, \ldots, x_k/n_k, y/m)$ ist wahr.

Definition.
Es sind arithmetisch repräsentierbar:

Addition	mittels	$y = x_1 + x_2$
Multiplikation	mittels	$y = x_1 * x_2$
DIV	mittels	$\exists r((r < x_2) \wedge (x_1 = y * x_2 + r))$
MOD	mittels	$\exists k((y < x_2) \wedge (x_1 = k * x_2 + y))$
$<$	mittels	$\exists z(a + z + 1 = b)$

Vereinbarung.

- Weglassen von Klammern nach Möglichkeit.
- $\forall x < k(\ldots)$ bedeutet
  $\forall x(\neg(x < k) \vee (\ldots))$.
- $\exists x < k(\ldots)$ bedeutet
  $\exists x((x < k) \wedge (\ldots))$.

Satz.
Jede WHILE–berechenbare Funktion ist arithmetisch repräsentierbar.

Benötigen zum Beweis folgenden Hilfssatz:

Hilfssatz.
Für jedes Tupel $(n_0, \ldots, n_k)$ gibt es a und b, so daß für alle $i = 0, 1, \ldots, k$ gilt:

$$n_i = a \text{ MOD } (1 + (i+1) * b).$$

Beweis.
Sei $s := max(k, n_0, \ldots, n_k)$ und
$b := s!$
Behauptung
$b_i = 1 + (i+1)b$ sind paarweise teilerfremd.
Ann.: es gibt eine Primzahl p mit $p|b_i$ und $p|b_j$ $(i < j)$
$\Rightarrow p|(b_j - b_i) = (j-i)b$
wegen $(j-i) \leq k \leq s$ gilt: $(j-i)|b$
$\Rightarrow p|b$
$\Rightarrow_{p|b_i=1+(i+1)b)} p|1$ (Widerspruch!)

Behauptung
Für $a \neq a'$, $0 \leq a < a' < b_0 * b_1 * \ldots * b_k$
sind auch die Lösungen der Restprobleme

$$\begin{aligned} n_i &= aMODb_i \ (i = 0, \ldots, k) \\ n'_i &= a'MODb_i \ (i = 0, \ldots, k) \end{aligned}$$

verschieden.
Ann.: $(n_0, \ldots, n_k) = (n'_0, \ldots, n'_k)$
$\Rightarrow b_i|a' - a$ für alle $0 \leq i \leq k$
$\Rightarrow_{b_i paarweise teilerfremd} b_0 * \ldots * b_k|a' - a$
Widerspruch! zu $a' - a < b_0 * \ldots * b_k$

Wähle also b so wie oben und a, so daß $(n_0, \ldots, n_k)$ Lösung des Restsystems ist.■

Beweis des Satzes
Zu zeigen:
Für jedes WHILE–Programm P mit den Variablen $x_0, \ldots, x_k$ gibt es eine Formel F_p mit den freien Variablen $x_0, \ldots, x_k$ und $y_0, \ldots, y_k$, so daß für alle $m_i, n_i \in I\!N$ gilt:

$F_p(m_0, \ldots, m_k, n_0, \ldots, n_k)$ ist wahr.
$\Leftrightarrow$ P stoppt, gestartet mit $m_0, \ldots m_k$ mit den Werten $n_0, \ldots, n_k$.

Berechnung:
Ein WHILE–Programm P mit den Programmvarablen $x_0, \ldots, x_k$ und eine n–stellige Funktion f mit $n < k$.

Behauptung:
f ist arithmetisch repräsentierbar durch:

$$F(x_1, \ldots, x_n, y) = \exists w_1 \ldots \exists w_k F_p(0, x_1, \ldots, x_n, \underbrace{0, \ldots, 0}_{k-n}, y, w_1, \ldots, w_k)$$

<u>Beweis durch Induktion über den Aufbau von P:</u>

1. P habe die Form $P = \ x_i := x_j + c$
   $\Rightarrow F_P = (y_i = x_j + c) \wedge \bigwedge_{k \neq i}(y_k = x_k$
2. P habe die Form $P = \ x_i := x_j - c$
   $\Rightarrow F_P = [(x_j < c) \vee (x_j = y_i + c)] \wedge [(x_j \geq c) \vee (y_i = 0)] \wedge \bigwedge_{k \neq i}(y_k = x_k)$
3. P habe die Form $P = \ Q, R$
   $\Rightarrow$ Nach Induktionsvorraussetzung gibt es Formeln F_Q und F_R.
   $\Rightarrow F_P = \exists z_o \ldots \exists z_k (F_Q(x_0, \ldots, x_k, z_0, \ldots, z_k) \wedge F_R(z_0, \ldots, z_k, y_0, \ldots, y_k))$
4. P habe die Form $P =$ WHILE $x_i \neq 0$ DO Q END
   $\Rightarrow$ Nach Induktionvorraussetzung gibt es eine Formel F_Q

$$\begin{aligned}
F + p \quad = \quad & \exists a_0 \exists b_0 \ldots \exists a_k \exists b_k \exists t (1)\\
& [G(a_0, b_0, 0, x_0) \wedge \ldots \wedge G(a_k, b_k, 0, x_k) \wedge (2)\\
& G(a_0, b_0, t, x_0) \wedge \ldots \wedge G(a_k, b_k, t, x_k) \wedge (3)\\
& \forall j < t \exists w ((G(a_i, b_i, j, w) \wedge \neg(w = 0)) \wedge (4)\\
& G(a_i, b_i, t, 0) \wedge (5)\\
& \forall j < t \exists w_0 \ldots \exists w_k \exists w'_0 \ldots \exists w'_k (6)\\
& [F_Q(w_0, \ldots, w_k, w'_0, \ldots, w'_k) \wedge\\
& G(a_0, b_0, j, w_0) \wedge \ldots \wedge G(a_k, b_k, j, w_k) \wedge\\
& G(a_0, b_0, j+1, w'_0) \wedge \ldots \wedge G(a_k, b_k, j+1, w'_k)]]
\end{aligned}$$

Erläuterung:

Zeile 1:	Existenz von Zahlenfolgen gemäß Hilfssatz. Die Zahlenfolgen sollen die Werte der Programmvariablen x_n im Verlauf von t Durchläufen der WHILE–Schleife repräsentieren. t ist die Gesamtzahl der Schleifendurchläufe bis x_i den Wert 0 erreicht.
Zeile 2:	Die Startwerte der Programmvariablen sind $x_0, \ldots, x_k$.
Zeile 3:	Die Endwerte der Programmvariablen sind $y_0, \ldots, y_k$.
Zeiel 4:	in jedem Durchlauf hat die Programmvariable x_i einen Wert $\neq 0$.
Zeile 5:	Der Wert 0 wird im t–ten Schleifendurchlauf erreicht.
Zeile 6:	Während aller Schleifendurchläufe wird gewährleistet, daß die Variablenwerte vor und nach Ausführung von Q gemäß der Formel F_Q miteinander verknüpft sind.

■

Satz.
Die Mege der arithmetischen Formeln ist nicht rekursiv aufzählbar.

Beweis.
Für jede Formel gilt:
entweder F ist wahr, oder $\neg F$ ist wahr.

$WA = \{F \mid F$ ist wahre arithmetische Formel $\}$

Annahme: WA ist aufzählbar.
$\Rightarrow$ WA ist entscheidbar:
Bei Eingabe F, zähle WA auf:
$WA := \{F_0, F_1, F_2, \ldots\}$
bis für ein i $F = F_i$ oder $F = \neg F_i$
Zeige:
WA ist nicht entscheidbar.
Sei A eine rekursiv aufzählbare ($\Rightarrow$ semientscheidbare), nicht entscheidbare Sprache (z.B. PCP).

$$\Rightarrow \chi_A(x) = \begin{cases} 1 & x \in A \\ \text{undef.} & x \notin A \end{cases}$$

ist WHILE–berechenbar.
$\Rightarrow_{letzter Satz}$ Sei $F_A(x,y)$ arithmetische Repräsentation von χ_A

Es gilt:

$$\begin{aligned} x \in A &\Leftrightarrow \chi_A(x) = 1 \\ &\Leftrightarrow F_A(x,1) \text{ ist wahr} \\ &\Leftrightarrow F_A(x,1) \in WA \end{aligned}$$

d.h. $A \leq WA$ mittels $x \mapsto F(x,1)$
$\Rightarrow WA$ ist nicht entscheidbar, also auch nicht rekursiv aufzählbar. ■

Bemerkung
WA heißt "Arithmetik" oder "elementare Zahlentheorie"
Bezeichnung: $Th(\mathbb{N})$ bzw. $Th(\mathbb{N}, *, +)$

Alter Wunsch:
Axiomatisierung ("Kalkülisieren") von WA
Mit dem letzten Satz kann man zeigen, daß das nicht vollständig geht:

Definition.
Ein *Beweissystem* für eine Menge $A \subseteq \Gamma^*$ ist ein Paar (B, F) mit den folgenden beiden Eigenschaften:

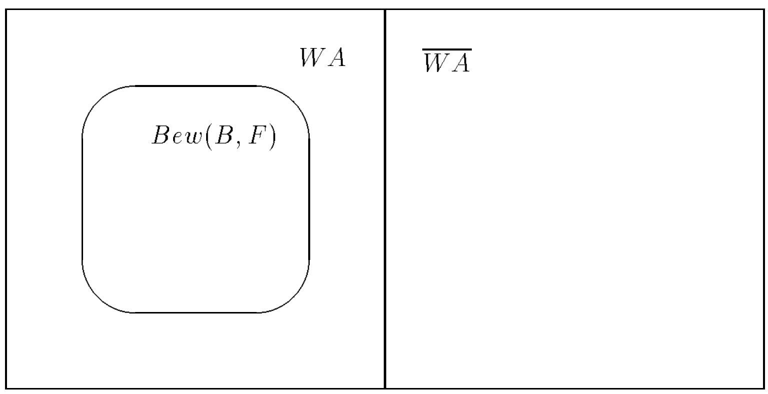
1. $B \subseteq \Gamma^*$ ist entscheidbar.
2. $F : B \to A$ ist total und berechenbar.

Bezeichnung
$Bew(B, F) = \{y \in \Gamma^* \mid \text{es gibt ein } b \in B \text{ mit } F(b) = y\}$
"alles mit B beweisbare"

Ein Beweissystem (B, F) für A heißt *vollständig*, falls
$A \subseteq Bew(B, F)$ (also $A = Bew(B, F)$)
(d.h. F ist surjektiv)

Gödelscher Unvollständigkeitssatz
Jedes Beweissystem für WA ist notwendigerweise unvollständig (d.h. es bleiben stets Formeln, die nicht beweisbar sind).

Beweis.
Annahme: $WA = Bew(B, F)$ für ein Beweissystem (B, F)
$\Rightarrow WA$ ist rekursiv aufzählbar:
man durchläuft alle $b \in B$ und gibt $F(b)$ aus. (Widerspruch!) ■



2 Komplexitätstheorie

Die Komplexitätstheorie untersucht den Ressourcenbedarf (Zeitbedarf, Speicherplatz, Programmlänge,...) von Algorithmen.

Die Komplexitätstheorie ist interessiert an:

oberen Schranken:	Garantie, daß das untersuchte Problem im Rahmen der vorgegebenen Ressourcen gelöst werden kann. Ein solcher Algorithmus heißt obere Schranke.
unteren Schranken:	Aussagen über den Mindestbedarf an Ressourcen für die Lösung eines Problems.

Ein Algorithmus ist (asymptotisch) *optimal* für ein Problem, wenn er bei der Lösung des Problems mit den durch die untere Schranke beschriebenen Ressourcen auskommt.

2.1 Komplexitätsmaße und Komplexitätsklassen

Beschreibe Problem als formale Sprache "Entscheidungsproblem":

$$A \subseteq \Sigma^*$$
$$x \in A \Leftrightarrow x \text{"löst" Problem}$$

Da Algorithmen untersucht werden, kann man sich auf die Klasse der <u>entscheidbaren</u> Sprachen beschränken.

Sei M eine Mehrband–TM, die bei allen Eingaben $x \in \Sigma^*$ hält.

$time_M : \Sigma^* \to I\!N$ "Zeitkomplexität von M".
$time_M(x) =$ Anzahl der Rechenschritte von M bei Eingabe von x.
$space_M : \Sigma^* \to I\!N$ "Speicherkomplexität von M".
$time_M(x) =$ Anzahl der bei Eingabe von x von M benutzten Felder der Bänder.

Sei $A \subseteq \Sigma^*$ entscheidbar.
$time_A(x) = min\{time_M(x) : M$ ist Mehrband–TM für $A\}$ ist Zeitkomplexität von A
$space_A(x) = min\{space_M(x) : M$ ist Mehrband–TM für $A\}$ ist Speicherkomplexität von A

Sei $f : I\!N \to I\!N$
$TIME(f(n)) = \{A \subseteq \Sigma^* : time_A(x) \leq f(|x|)\}$
$SPACE(f(n)) = \{A \subseteq \Sigma^* : space_A(x) \leq f(|x|)\}$

Satz.
Die Komplexitätsklasse $TIME(f(n))$, wobei $f(n)$ nach oben durch LOOP–Programme beschränkt werden kann, ist enthalten in der Klasse der primitiv rekursiven Sprachen (bzw. der LOOP–berechenbaren Sprachen).

Beweis.
Sei M TM mit $time_M(x) \leq f(|x|)$.
Simuliere M durch ein GOTO–Programm , wobei jeder TM–Schritt durch eine endliche Zahl von Wertzuweisungen bzw. GOTOs simuliert wird. Forme das GOTO–Programm in ein äquivalentes WHILE–Programm mit nur einer WHILE–Schleife um.
Die Anzahl der Durchläufe der WHILE–Schleife ist durch die Zahl $f(n)$ beschränkt.
Ersetze "WHILE count$\neq$ 0 DO" durch "$y := f(n)$;LOOP y DO".
Das Ergebnis ist ein LOOP–Programm, das M simuliert. ∎

Korollar
$TIME(n^k)$ $(k \in I\!N), TIME(2^n), TIME(2^{2^{\cdot^{\cdot^{\cdot^{2}}}}})$ enthalten nur primitiv rekursive Mengen.

Komplexitätsmaße können auch mit Hilfe von WHILE–Programmen eingeführt werden.
Vorsicht
$x_i := x_j$ muß so groß angesetzt werden, wie die Anzahl der Bits, die bei dieser Aktion übertragen werden (also etwa $\log x_j$).
("logarithmisches Kostenmaß")
Dieses Kostenmaß ist zwar genau, aber schwierig zu analysieren.
Setzt man dagegen die Kosten für elementare Anweisungen 1, spricht man vom "uniformen Kostenmaß".

Beispiel
INPUT (n);
$x := 2$;
LOOP n DO $x := x * x$ END;
OUTPUT(x);

Der Algorithmus berechnet 2^{2^n}.
Kosten bei: "uniformem Kostenmaß": $O(n)$
"logarithmischem Kostenmaß": $O(2^n)$

Üblich
Die Komplexität eines Algorithmus wird unter dem uniformen Kostenmaß angegeben.

2.2 Die Komplexitätsklassen P und NP

Es ist sinnvoll, anstelle von Funktionen Funktionsklassen zu betrachten:

Beispiel
Problem ist mit Einband–TM M lösbar $\Leftrightarrow$
Problem ist mit Mehrband–TM N lösbar,
aber
$time_M = (time_N)^2$

Es werden deshalb Klassen betrachtet, die gegenüber solchen Modifikationen abgeschlossen sind.

Definition.
Ein *Polynom* $p : \mathbb{N} \to \mathbb{N}$ ist eine Funktion der Form
$p(n) = a_k n^k + \ldots + a_1 n + a_0 \quad a_i \in \mathbb{N}, k \in \mathbb{N}$

Komplexitätsklasse P

$$\begin{aligned} P &= \bigcup_{P\ Polynom} TIME(p(n)) \\ &= \{A : \exists \text{ TM } M \text{ mit } L(M) = A \text{ und } time_M(x) \leq p(|x|) \text{ für ein Polynom } p\} \\ \Rightarrow P &= \bigcup_{k \geq 1} TIME(O(n^k)) \end{aligned}$$

Bermerkung
Auch Algorithmen der Komplexität $n \log n$ sind polynomial, da
$n \log n = O(n^2)$.

$n^{\log n}, 2^n$ sind nicht polynomial ("exponentiell").

Allgemein akzeptiert

$\boxed{P = \text{Klasse der effizienten Algorithmen(d.h. der praktisch realisierbaren Algorithmen)}}$.
Algorithmen mit exponentieller Laufzeit sind nicht effizient.

- P könnte auch als WHILE–Programm mit logarithmischen Kostenmaß definiert werden.
- $P \subseteq TIME(2^n) \subseteq \{\text{Primitiv rekursive Sprache}\}$
- P enthält alle Probleme, für die sich in polynomieller Zeit ein Beweis finden läßt.

Betrachte daneben die Klasse NP der Probleme, für die sich in polynomieller Zeit ein vorgegebener Beweis überprüfen läßt:

Definition.
Eine *nichtdeterministische TM* ist diejenige TM, bei der die Übergangsfunktion eine Übergangsrelation ist (eine Konfiguration hat i.a. mehrere mögliche Nachfolgekonfigurationen). Eine *akzeptierende Berechnung* besteht aus einer zulässigen Folge von Konfigurationen, die mit der Startkonfiguaration beginnt und in einer akzeptierenden Konfiguration endet.

Definition.
Sei M eine nichtdeterministische Mehrband–TM.

$$time_M(x) = \begin{cases} \min\{\text{Länge einer akzeptierenden Berechnung von } M \text{ auf } x\}, & x \in L(M) \\ 0, & x \notin L(M) \end{cases}$$

$f : \mathbb{N} \to \mathbb{N}$

$$NTIME(f(n)) = \{A : \exists \text{ NTM } M \text{ mit } L(M) = A \text{ und } time_M(x) \leq f(|x|) \forall x \in \Sigma^*\}$$

$$\begin{aligned} NP &:= \bigcup_{p \ Polynom} NTIME\ p(n) \\ &= \bigcup_{k \geq 1} TIME(O(n^k)) \end{aligned}$$

Offenbar gilt:

$$P \subseteq NP$$

Die Umkehrung ist unklar: "$P - NP$–Problem"

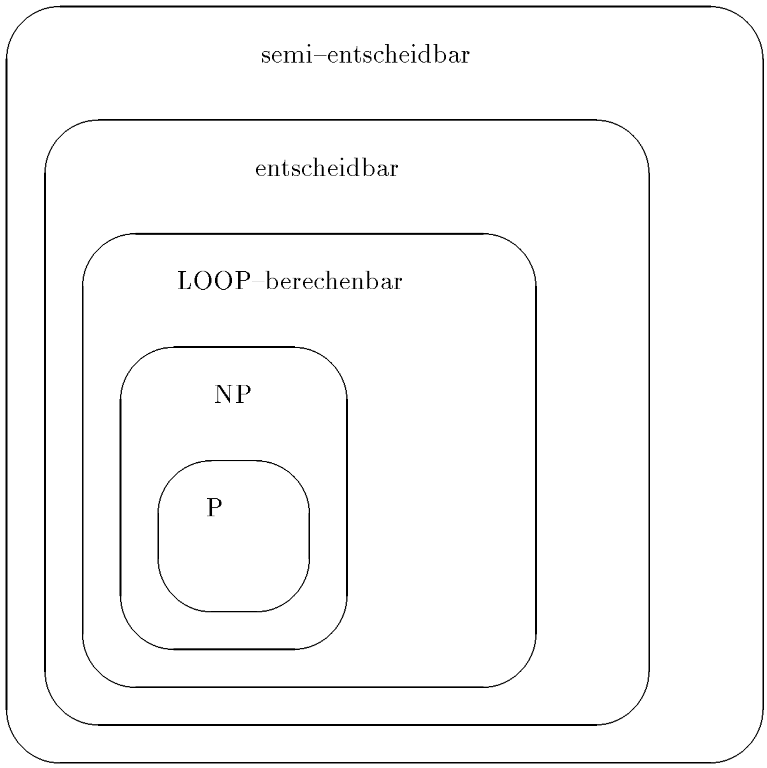
Das $P - NP$–Problem ist wegen seiner Bedeutung für die Findung effizienter Algorithmen von großer Bedeutung (von vielen als das bedeutendste Problem der Theoretischen Informatik angesehen).

Allgemeine Annahme

$$P \neq NP$$

Bei der Untersuchung des $P - NP$–Problems wurde die Theorie der NP–Vollständigkeit entwickelt. (Cook '71, Karp '72)

Es gilt:



2.3 NP–Vollständigkeit

Definition.
Seien $A, B \subseteq \Sigma^*$.
A heißt *polynomiell auf B reduzierbar* $(A \leq_p B)$, falls es eine totale und mit polynomialer Komplexität berechenbare Funktion $f : \Sigma^* \to \Sigma^*$ gibt, so daß für alle $x \in \Sigma^*$ gilt:

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

Lemma
$\leq_p$ ist transitiv.

Beweis.
Übungsaufgabe.

Lemma
i) $A \leq_p B, B \in P \Rightarrow A \in P$
ii) $A \leq_p B, B \in NP \Rightarrow A \in NP$

Beweis.
i) Sei $A \leq_p B$ mittels f.
TM M_f berechne f in Polynomialzeit p.
Sei $B \in P$ mittels TM M in Rechenzeit q.
$\Rightarrow (M_f; M)$ berechnet A in Rechenzeit ∎
$p(|x|) + q(|f(x)|) \leq p(|x|) + q(p(|x|))$ ist Polynom.

ii) analog

Definition.
Eine Sprache A heißt *NP–hart*, falls für alle Sprachen $L \in NP$ gilt: $L \leq_p A$.
Eine Sprache A heißt *NP–vollständig*, falls A NP–hart ist und $A \in NP$ gilt.

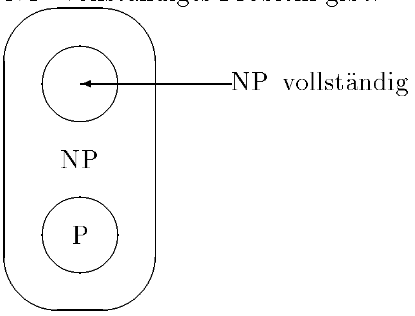
Bemerkung
NP-vollständige Sprachen sind "schwere" Sprachen in NP.

Satz.
Sei A NP–vollständig.
Dann gilt:

$$A \in P \Leftrightarrow P = NP$$

Beweis.
$(\rightarrow)$: Sei $A \in P, L \in NP$ beliebig.
da A NP–hart gilt: $L \leq_p A$
$\Rightarrow_{Lemma} L \in P$ ∎
$(\leftarrow)$: $P = NP \Rightarrow A \in P$

$\Rightarrow$ Zum Nachweis von $P = NP$ genügt die Angabe eines polynomialen Algorithmus für ein NP–vollständiges Problem:
Die Annahme $P \neq NP$ bedeutet, daß es keinen effizienten Algorithmus für ein NP–vollständiges Problem gibt.



Definition.
Das folgende Problem heißt "Erfüllbarkeitsproblem der Aussagenlogik" SAT.

Gegeben:	Eine Formel F der Aussagenlogik mit n Variablen, $n \in \mathbb{N}$
Gefragt:	Ist F erfüllbar?
	(d.h. $\exists$ Belegung $a \in \{0,1\}^n$ mit $F(a) = 1$?)

Formal
$SAT = \{code(F) \in \Sigma^* : F$ ist erfüllbare Formel der Aussagenlogik$\}$

Theorem von Cook
Das Erfüllbarkeitsproblem der Aussagenlogik SAT ist NP–vollständig.

Beweis.
1.) Zu zeigen: $SAT \in NP$.
Angabe einer polynomial zeitbeschränkten NTM für SAT:
M stellt in einem Durchlauf über der Eingabe fest, welche Variablen in F vorkommen.
Seien dies $x_1, \ldots, x_k$.
M rät die Werte $a_1, \ldots, a_k \in \{0,1\}$ für $x_1, \ldots, x_k$ und setzt diese in F ein.
(es existieren 2^k mögliche unabhängige Berechnungen – für jede Belegung eine)
Für jede Belegung rechnet M jeweils deterministisch den Wert von F aus und akzeptiert, falls dieser 1 ist.
$F \in SAT \Leftrightarrow M$ akzeptiert F.
Wegen $k \leq |F|$ ist M polynomialzeit beschränkt.
$\Rightarrow SAT \in NP$.

2.) Zu zeigen: SAT ist NP–hart.
Sei $L \in NP$ beliebig.
M NTM für L der Rechenzeit p.
OBdA gelte: $\delta(z_e, a) \ni (z_e, a, N)$ (d.h. erreichte Endzustände werden nicht mehr

verlassen).
Sei $x = x_1, \ldots, x_n \in \Sigma^*$ die Eingabe von M.
Konstruktion einer Fromel F mit

$$x \in L \Leftrightarrow F \text{ ist erfüllbar.}$$

Sei $\Gamma = \{a_1, \ldots, a_l\}$ das Anfangsalphabet
$Z = \{z_1, \ldots, z_k\}$ die Zustandsmenge von M
F enthält folgende Variable:

Variable	Indizes	intendierte Bedeutung
$zust_{t,z}$	$t = 0, \ldots, p(n)$ $z \in Z$	$zust_{t,z} = 1 \Leftrightarrow$ nach t Schritten befindet sich M im Zustand z
$pos_{t,i}$	$t = 0, \ldots, p(n)$ $i = -p(n), \ldots, p(n)$	$pos_{i,j} = 1 \Leftrightarrow$ M's Schreib–Lesekopf befindet sich nach t Schritten auf Position i
$band_{t,i,a}$	$t = 0, \ldots, p(n)$ $i = -p(n), \ldots, p(n)$ $a \in \Gamma$	$band_{t,i,a} = 1 \Leftrightarrow$ nach t Schritten befindet sich auf Bandposition i das Zeichen a

F besteht aus mehreren Bauteilen:

$G(x_1, \ldots, x_m) = 1 \Leftrightarrow$ für geanu ein i ist $x_i = 1$.

Behauptung
G existiert und es gilt $size(G) = O(m^2)$.

Beweis.

$$G = (\bigvee_{i=1}^{k} x_i) \wedge (\bigwedge_{j=1}^{m-1} \bigwedge_{l=j+1}^{m} (\neg(x_j \wedge x_l))$$

Die erste Teilformel wird genau dann wahr, wenn mindestens eine Variable wahr ist.
Die zweite Teilformel wird genau dann wahr, wenn höchstens eine Variable wahr wird.

$$F = R \wedge A \wedge \ddot{U}_1 \wedge \ddot{U}_2 \wedge E \text{ , wobei}$$

R	für Randbedingung
A	für Anfangsbedingung
$\ddot{U}_1, U_2$	für Übergangsbedingung und
E	für Endbedingung steht.

R drückt aus:

- Zu jedem Zeitpunkt t ergibt sich für genau ein z $zust_{t,z} = 1$.
- Zu jedem Zeitpunkt t gibt es genau eine Bandposition i mit $pos_{t,i} = 1$.
- Zu jedem Zeitpunkt t und jeder Bandposition i gibt es genau ein a mit $band_{t,i,a} = 1$

$$R = \bigwedge_t [G(zust_{t,z_1}, \ldots, zust_{t,z_k}) \wedge G(pos_{t,-p(n)}, \ldots, pos_{t,p(n)}) \wedge \bigwedge_i G(band_{t,i,a_1}, \ldots, band_{t,i,a_l})]$$

A beschreibt den Status der Variablen für den Fall $t = 0$:

$$A = zust_{0,z_0} \wedge pos_{0,1} \wedge \bigwedge_{j=1}^{n} band_{0,j,x_j} \wedge \bigwedge_{j=-p(n)}^{0} band_{0,j,\square} \wedge \bigwedge_{j=n+1}^{p(n)} band_{0,j,\square}$$

$\ddot{U}_1$ beschreibt den Übergang von Zeitpunkt t nach $t+1$ an der Kopfposition ($y \in \{-1, 0, +1\}$):

$\ddot{U}_1 = \bigwedge_{t,z,i,a}[(zust_{t,z} \wedge pos_{t,i} \wedge band_{t,i,a}) \rightarrow \bigvee_{z',a',y \text{ mit } \delta(z,a) \ni (z',a',y)} (zust_{t+1,z'} \wedge pos_{t+1,i+y} \wedge band_{t+1,i,a'}]$

$\ddot{U}_2$ besagt, daß auf den übrigen Bandfeldern nichts passiert:

$$\ddot{U}_2 = \bigwedge_{t,i,a} ((\neg pos_{t,i} \wedge band_{t,i,a}) \rightarrow band_{t+1,i,a})$$

E überprüft, ob der Endzustand erreicht ist (wird auf jeden Fall im Zeitpunkt $p(n)$ erreicht):

$$E = \bigvee_{z \in E} zust_{p(n),z}$$

($\rightarrow$): Sei $x \in L$
$\Rightarrow \exists$ nichtdeterministische Rechnung der Länge $p(n)$, die in einen Endzustand führt.
$\Rightarrow$ Alle Teilformeln von F erhalten den Wert 1.
$\Rightarrow F(x)$ erhält den Wert 1.
$\Rightarrow F(x)$ ist erfüllbar.

($\leftarrow$) : Sei $F(x)$ erfüllbar.
$\Rightarrow \exists$ Belegung, die F und alle Teilformeln den Wert 1 annehmen läßt.

Insbesondere ist R erfüllt:
$\Rightarrow zust_{t,z}, pos_{t,i}, band_{t,i,a}$ können $\forall t$ als Konfiguration von M interpretiert werden.
Insbesondere ist A erfüllt:

für $t = 0$ kann aus den Variablenwerten die Startkonfiguration von M abgelesen werden.

Insbesondere sind $\ddot{U}_1, \ddot{U}_2$ erfüllt:
$\Rightarrow$ zwischten t und $t+1$ ist die Nachfolgekonfigurationsbedingung erfüllt.
$\Rightarrow \forall t = 0, 1, 2, \ldots$ ist eine mögliche nichtdeterministische Rechnung beschrieben.

Insbesondere ist E erfüllt:
$\Rightarrow$ Rechnung erreicht Endzustand.

Insgesamt gilt also :

$$x \in L(M)$$

Noch zu zeigen: F ist in polynomieller Zeit berechenbar:
Offenbar ist der Aufwand zur Erzeugung von F linear in der Länge von F.

$$\begin{aligned}
\text{Wegen } |R| &= O(n^2) \\
|A| &= O(n) \\
|\ddot{U}_1| &= O(n^2) \\
|\ddot{U}_2| &= O(n^2) \\
|E| &= O(1) \\
\\
\text{gilt } |F| &= O(n^2)
\end{aligned}$$

■

NP–Berechnungen: "guess and check"
Alle deterministischen Algorithmen zur Berechnung von SAT haben Komplexität $2^{O(n)}$.
(Z.B. systematisches Durchprobieren aller Eingabeformeln).
Da SAT NP–hart ist folgt:

$$NP \subseteq \bigcup_{p \text{ Polynom}} TIME(2^{p(n)})$$

2.4 Weitere NP–vollständige Probleme

2.4.1 $3SAT$

Definition. ($3SAT$)
Gegeben: Boolesche Formel F in KNF mit höchstens 3 Literalen pro Klausel.
Gefragt: Ist F erfüllbar?

Satz.
$3SAT$ ist NP–vollständig.

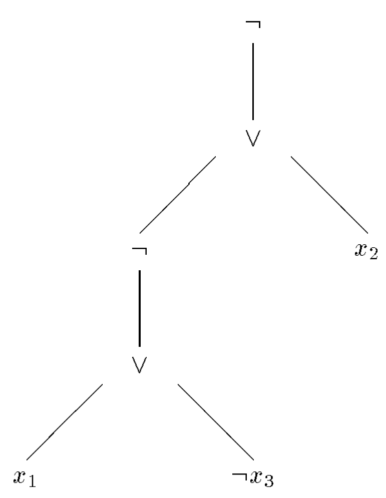
Beweis. 1.) $3SAT \in NP$, klar mit "guess and check"– Argument.
2.) $3SAT$ NP–hart.
Es reicht zu zeigen : $SAT \leq_p 3SAT$.
Angabe eines polynomiellen Verfahrens, das eine beliebige Formel F in eine Formel F' in KNF mit höchstens 3 Literalen pro Klausel umformt mit:

$$F \text{ erfüllbar} \Leftrightarrow F' \text{ erfüllbar.}$$

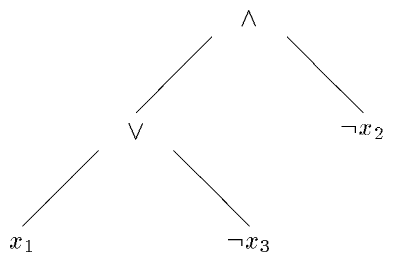
(Dabei genügt "Erfüllbarkeitsäquivalenz". "Äquivalenz ist nicht notwendig.)

Allgemeine Verfahren zur Überführung von F in äquivalente KNF benötigt i.a. exponentielle Zeit und garantiert nicht, daß alle Klauseln höchstens 3 Literale enthalten.

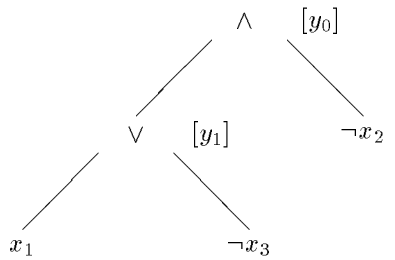
Erläuterung des Verfahrens am Beispiel:



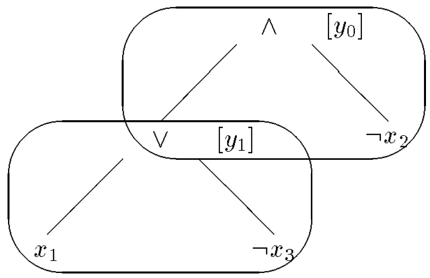
1. Schritt
Mit den DeMorgan'schen Regeln werden alle Negationszeichen zu den Variablen gebracht.



2. Schritt
Jedem inneren Knoten wird eine Variable $\{y_0, y_1, \ldots\}$ zugeordnet, wobei der Baumwurzel y_0 zugeordnet wird.



3. Schritt
Jedem inneren Knoten wird eine Teilformel der Form $(v \leftrightarrow (y \circ z))$, $\circ \in \{\wedge, \vee\}$ zugeordnet. Man erhält eine neue Formel F_1, indem man alle Teilformeln durch $\wedge$ verknüpft und für die Wurzel y_0 die Teilformel $[y_0]$ hinzunimmt.



Es gilt:

$$F \text{ erfüllbar } \Leftrightarrow F_1 \text{ erfüllbar}$$

$(\rightarrow)$: Eine erfüllende Belegung von F liefert eine erfüllende Belegung von F_1.
$(\leftarrow)$: Die Belegung der x–Variablen einer erfüllenden Belegung von F_1 liefert eine erfüllende belegung für F.

4. Schritt
Umformung jeder Teilformel in KNF.
Es enstehen Klauseln mit höchstens 3 Literalen.
Das Verfahren ist polynomial, da jede Teilformel in konstanter Länge umgeformt werden kann.

Beispiel $[a \leftrightarrow (b \vee c)] \mapsto (a \vee \neg b) \wedge (\neg a \vee b \vee c) \wedge (a \vee \neg c)$
$[a \leftrightarrow (b \wedge c)] \mapsto (\neg a \vee b) \wedge (\neg a \vee c) \wedge (a \vee \neg b \vee \neg c)$

$\Rightarrow F_1 = y_0 \wedge (\neg y_0 \vee y_1) \wedge (\neg y_0 \vee \neg x_2) \wedge (y_0 \vee \neg y_1 \vee x_2) \wedge (y_1 \vee \neg x_1) \wedge (\neg y_1 \vee x_1 \vee \neg x_3) \wedge (y_1 \vee x_3)$
mit
F_1 ist erfüllbarkeitsäquivalent mit F.

Umformung von F in F_1 ist in polynomieller Zeit möglich. ■

Bemerkung
Für ein analoges Problem gilt:
$2SAT \in P$, da es nur polynomiell viele verschiedene Klauseln mit höchstens 2 Literalen über $\{x_1, .., x_n\}$ gibt.

2.4.2 CLIQUE

Definition. (CLIQUE)
Gegeben: Ein ungerichteter Graph $G = (V, E)$, $k \in \mathbb{N}$.
Gefragt: Besitzt G eine "Clique" der Größe k?

Wobei Clique ein vollständiger Teilgraph $G' = (V', E')$ ist, mit

$$(u, v) \in E \quad \forall u, v \in V', \ u \neq v$$

Satz.
CLIQUE ist NP–vollständig.

Beweis.
1.) CLIQUE$\in NP$ mit "guess and check".
2.) CLIQUE ist NP–hart.
Sei F Formel in KNF mit (genau) 3 Literalen pro Klausel.
$F = (z_{1,1} \vee z_{1,2} \vee z_{1,3}) \wedge \ldots \wedge (z_{m,1} \vee z_{m,2} \vee z_{m,3})$ mit $z_{i,j} \in \{x_1, x_2, \ldots\} \wedge \{\neg x_1, \neg x_2, \ldots\}$

Ordne F Graph $G = (V, E)$ und eine Zahl k zu gemäß:

$$\begin{aligned} V &= \{(1,1),(1,2),\ldots,(m,1),(m,2),(m,3)\} \\ E &= \{\{(i,j),(p,q) : i \neq p \text{ und } z_{i,j} \neq \neg z_{p,q}\} \\ k &= m \end{aligned}$$

Es gilt: F ist erfüllbar durch Belegung B.
$\Leftrightarrow$ Jede Klausel hat ein Literal, das unter der Belegung B den Wert 1 annimmt, z.B.: $z_{1,j_1}, z_{2,j_2}, \ldots, z_{m,j_m}$.
$\Leftrightarrow$ Es gibt Literale $z_{1,j_1}, z_{2,j_2}, \ldots, z_{m,j_m}$, die paarweise nicht komplementär sind.
$\Leftrightarrow$ Es gibt Knoten $(1, j_1), (2, j_2), \ldots, (m, j_m)$, die paarweise verbunden sind.
$\Leftrightarrow$ G hat CLIQUE der Größe k. ■

2.4.3 HAMILTON–KREIS

Definition. (GERICHTETER HAMILTON–KREIS)
Gegeben: Ein gerichteter Graph $G = (V, E)$.
Gefragt: Besitzt G einen Hamilton–Kreis?

Wobei ein Hamilton–Kreis eine Permutation der Knotenindizes $(v_{\pi(1)}, \ldots, v_{\pi(n)})$, so daß $(v_{\pi(i)}, v_{\pi(i+1)}) \in E \ \forall i = 1, \ldots, n-1$ und $(v_\pi(n), v_{\pi(1)}) \in E$.

Definition. (UNGERICHTETER HAMILTON–KREIS)
Gegeben: Ein ungerichteter Graph $G = (V, E)$.
Gefragt: Besitzt G einen Hamilton–Kreis?

Satz.
GERICHTETER HAMILTON–KREIS ist NP–vollständig.

<u>Beweis.</u>
GERICHTETER HAMILTON–KREIS $\in NP$ "guess and check".

Noch zu zeigen: GERICHTETER HAMILTON–KREIS NP–hart.
Es wird gezeigt: 3SAT$\leq_p$GERICHTETER HAMILTON–KREIS.

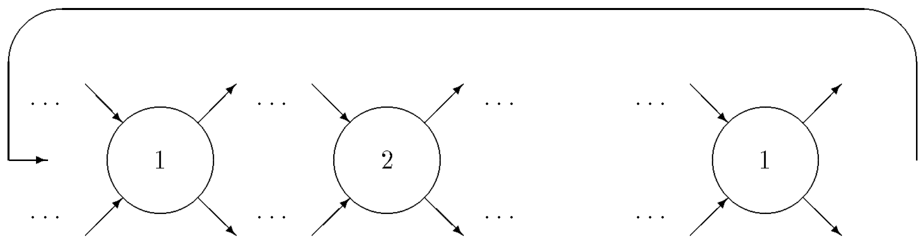
Sei F Formel in KNF mit genau 3 Literalen pro Klausel.

$$\Rightarrow F = (z_{1,1} \vee z_{1,2} \vee z_{1,3}) \wedge \ldots \wedge (z_{m,1} \vee z_{m,2} \vee z_{m,3})$$

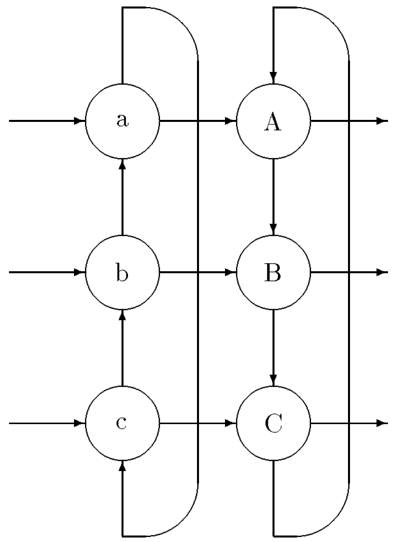
mit

$$z_{i,j} \in \{x_1, \ldots, x_n\} \cup \{\neg x_1, \ldots, \neg x_n\}$$

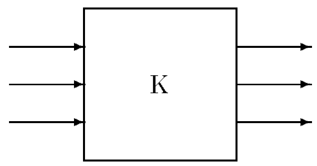
Konstruktion eines gerichteten Graphen:



Vom gleichen Knoten i gehen jeweils 2 Kanten aus.
Die beiden Kanten führen durch folgenden "Klauselgraph" K, von dem m Kopien bereitstehen.



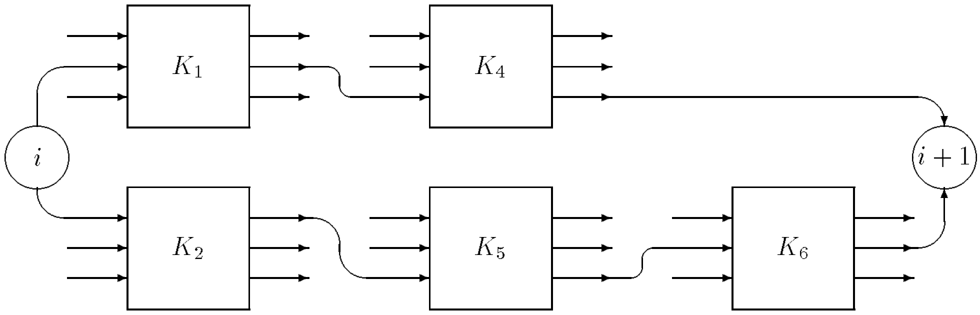
Symbol:



Beispiel.

Literal x_i	kommt in Klausel 1 an Position 2
	und in Klausel 4 an Position 3 vor.
Literal $\neg x_i$	kommt in Klausel 2 an Position1
	in Klausel 5 an Posiyion 3
	und in Klausel 6 an Position 2 vor

Verbindungen von i nach $i+1$:



Beobachtung
Enthält der konstruierte Graph G einen Hamilton–Kreis, so verläßt dieser jedes K wie folgt:
Kommt der Hamilton–Pfad bei $a(b,c)$ an, so verläßt er K in $A(B,C)$.

Beweis.
Annahme: Hamilton–Pfad erreicht K in a und verläßt K nicht in A:
Mögliche Fälle:
$a-A-B$ Sackgasse in b.
$a-A-B-C$ Sackgasse in b.
$a-c-b-B$ A und C sind nicht mehr erreichbar.
$a-c-b-B-C$ A nicht mehr erreichbar.
$a-c-C-A-B$ Sackgasse bei b.

Analog für b und c.

$\Rightarrow$ Ein Hamilton–Pfad kann durch K nur folgende Wege nehmen:
$a-A$
$a-c-C-A$
$a-c-b-B-C-A$

Behauptung.
F erfüllbar $\Leftrightarrow$ G hat Hamilton–Kreis.

Beweis.
($\rightarrow$) : Habe F erfüllende Belegung:
Gilt $x_i = 1$, dann folge dem oberen Pfad von i an.
Gilt $x_i = 0$, dann folge dem unteren Pfad.

Die entsprechenden "Klauselgraphen" werden durchlaufen.

So wird erreicht, daß bei Rückkehr nach 1 alle Konten von G durchlaufen wurden.

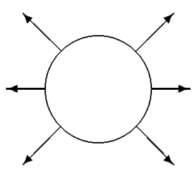
($\leftarrow$) : G besitzt Hamilton–Kreis.
Definiere Variablenbelegung für F:
$x_i = 1$, falls Hamilton–Kreis Knoten i nach "oben" verläßt.
$x_i = 0$, falls Hamilton–Kreis Konten i nach "unten" verläßt.

Die Belegung erfüllt F,da jeder Klauselgraph durchlaufen, entsprechende Klausel also erfüllt wird. ∎

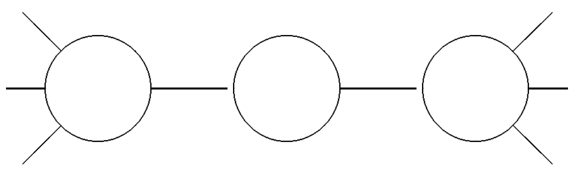
Satz.
UNGERICHTETER HAMILTON–KREIS ist NP–vollständig.

Bewies
Es gilt:
GERICHTETER HAMILTON–KREIS $\leq_p$ UNGERICHTETER HAMILTON–KREIS.

Ersetze in gerichtetem Graphen Knoten der Form



durch ungerichteten Teilgraphen



und erzwinge damit, daß jeder Hamilton–Kreis G_v in Pfeilrichtung durchlaufen wird. ■

Bemerkung
Hamilton–Kreis ("Jeder Knoten wird genau einmal durchlaufen") ist NP–vollständig. Eulerkreis("Jede Kante wird genau einmal durchlaufen") ist in P ($\rightarrow$ Königsberger Brückenproblem).

Definition. (TRAVELING SALESPERSON Problem)
Gegeben: $n \times n$ Matrix $(M_{i,j})$ von "Entfernungen" zwischen n Städten.
Gefragt: $\exists$ Permutation ("Rundreise") mit

$$\sum_{i=1}^{n-1} M_{\pi(i),\pi(i+1)} + M_{\pi(n),\pi(1)} \leq k?$$

Satz.
Das TRAVELING SALESPERSON Problem ist NP–vollständig.

Beweis.
TRAVELING SALESPERSON Problem$\in NP$ "guess and check".

Noch zu zeigen:
UNGERICHTETER HAMILTON–KREIS $\leq_p$ TRAVELING SALESPERSON Problem:

$$G = (\{1, \ldots, n\}, E) \mapsto \left\{ \begin{array}{ll} \text{Matrix} M_{i,j} = & \left\{ \begin{array}{ll} 1 & \{i,j\} \in E \\ 2 & \{i,j\} \notin E \end{array} \right. \\ \text{Rundreiselänge:} & n \end{array} \right.$$

■