Churchsche These

Die Klasse der effektiv berechenbaren Funktionen ist genau die Klasse der μ-rekursiven Funktionen. Jede Formalisierung von berechenbaren Funktionen liefert die gleiche Klasse.

Wir werden einige dieser Formalisierungen kurz vorstellen.

6.79 Definition Register-Maschinen (goto-Programme über N)

Goto-Programme über der Variablenmenge $V = \{V_0, \ldots, V_n\}$ sind markierte Befehlsfolgen der Form

$$\begin{aligned} \mathcal{P} :: \quad & 0 : B_0 \\ & 1 : B_1 \\ & \vdots \\ & L : B_L \end{aligned}$$

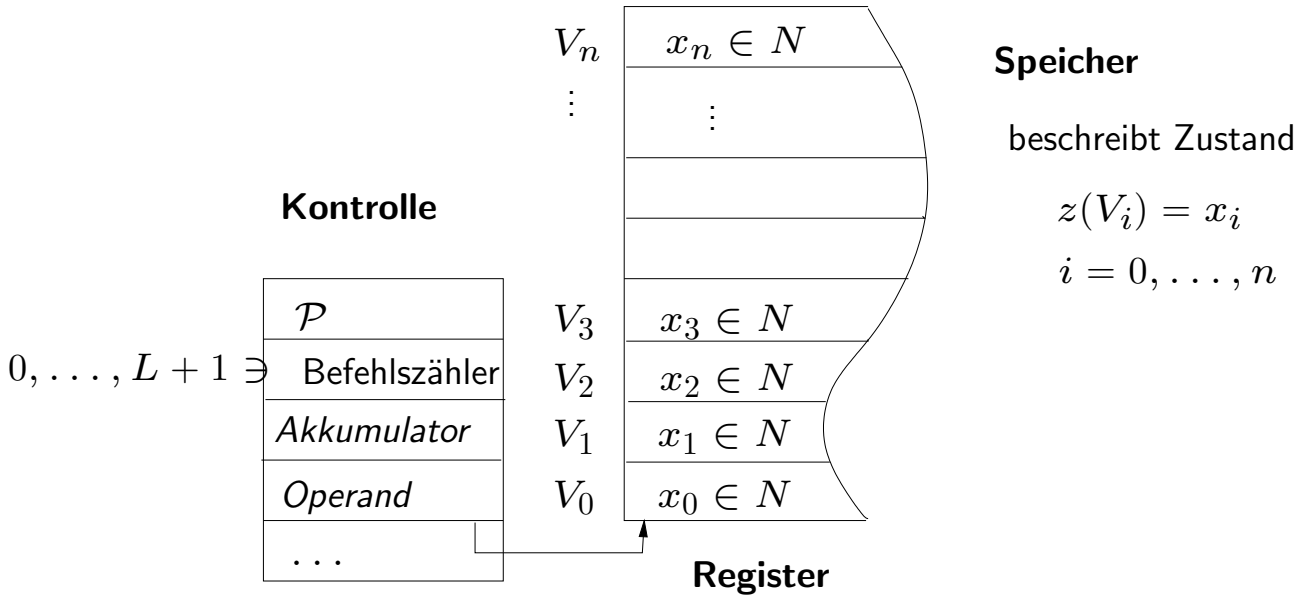
Mit **Befehlen** $B_i, i \in \{0, \ldots, L\}$ einer der Formen

$\bullet V_i := s(V_i)$ $\bullet V_i := p(V_i)$ $\bullet$**if** $V_i = 0$ **then goto** l_1 **else goto** l_2

mit $V_i \in V, l_1, l_2 \in \{0, \ldots, L+1\}$ (**Marken**).

Die intendierte Semantik von $\boldsymbol{s}, \boldsymbol{p}$ ist die Nachfolger- bzw. die Vorgängerfunktion auf $\mathbb{N}$.

Register-Maschinen Semantik



Interpretersemantik:

$I_{\mathcal{P}}(l, z) : \{0, \ldots, L+1\} \times \mathcal{Z} \to \{0, \ldots, L+1\} \times \mathcal{Z}$

Startzustand: $(0, z)$, Eingaben $z(V_i) = x_i \in \mathbb{N}$

$$I_{\mathcal{P}}(l, z) = \begin{cases} (l+1, z(V_i / z(V_i)+1)) & l : V_i := s(V_i) \in \mathcal{P} \\ (l+1, z(V_i / z(V_i) \dot{-} 1)) & l : V_i := p(V_i) \in \mathcal{P} \\ (l_1, z) & l : \underline{\textbf{if}}\ V_i = 0\ \underline{\textbf{then}}\ \textbf{goto}\ l_1\ \underline{\textbf{else}}\ \textbf{goto}\ l_2 \in \mathcal{P} \\ & \qquad \wedge z(V_i) = 0 \\ (l_2, z) & l : \underline{\textbf{if}}\ V_i = 0\ \underline{\textbf{then}}\ \textbf{goto}\ l_1\ \underline{\textbf{else}}\ \textbf{goto}\ l_2 \in \mathcal{P} \\ & \qquad \wedge z(V_i) \neq 0 \\ (l, z) & l = L+1 \text{ oder } l \text{ kein Label in } \mathcal{P}\ \ (\textbf{Stopp}) \end{cases}$$

Register-Maschinen berechenbare Funktionen

Programm $\mathcal{P}$ stoppt aus Startzustand z gdw keine Befehlsausführung mehr möglich.

Ein- Ausgabevereinbarungen für die Berechnung von Funktionen
$f : \mathbb{N}^l \to \mathbb{N}$: $\mathcal{P}$ **berechnet** f gdw

i) Die Rechnung stoppt aus Anfangszustand
$z(V_i) = x_i$, $i = 1, \ldots, l$, $z(V_i) = 0$ sonst gdw
$(x_1, \ldots, x_l) \in dom(f)$.

ii) Gilt $(x_1, \ldots, x_l) \in dom(f)$, $y = f(x_1, \ldots, x_l)$, so stoppt $\mathcal{P}$ in einem Zustand z' mit $z'(V_0) = y$. Also gilt:

$$\exists t \in \mathbb{N} : I^t_{\mathcal{P}}(0, z) = (L + 1, z')$$

6.80 Beispiel Einfache RM bzw. goto-Programme

Sei S festes Register mit Inhalt 0, d. h. $z(V_s) = 0$

a) Register „leeren“
$V \Leftarrow 0 ::$ $0 : V := p(V)$
$1 :$ **if** $V = 0$ **then goto** 2 **else goto** 0

b) $Z \Leftarrow Z + 1$, $Z \Leftarrow Z \dot{-} 1$ sind leicht anzugeben.

Einfache RM bzw. goto-Programme

c) „Inhalt umspeichern“

$Z \Leftarrow Y$ ::	$0 : Z \Leftarrow 0$
copy Y nach Z	$1 :$ **if** $Y = 0$ **then goto** 5 **else goto** 2
Hilfsregister U	$2 : Y := p(Y)$
initialisiert mit 0	$3 : U := s(U)$
unbedingter Sprung:	$4 :$ **if** $V_s = 0$ **then goto** 1 **else goto** 1
goto 1 (Abkürzung)	
	$5 :$ **if** $U = 0$ **then goto** 10 **else goto** 6
	$6 : U := p(U)$
	$7 : Z := s(Z)$
	$8 : Y := s(Y)$
	$9 :$ **goto** 5

6.81 Lemma

- Jede μ-rekursive Funktion ist goto-berechenbar.
- Jede goto-berechenbare Funktion ist μ-rekursiv.

Beweisidee:

- Zeige die Grundfunktionen sind goto-berechenbar.

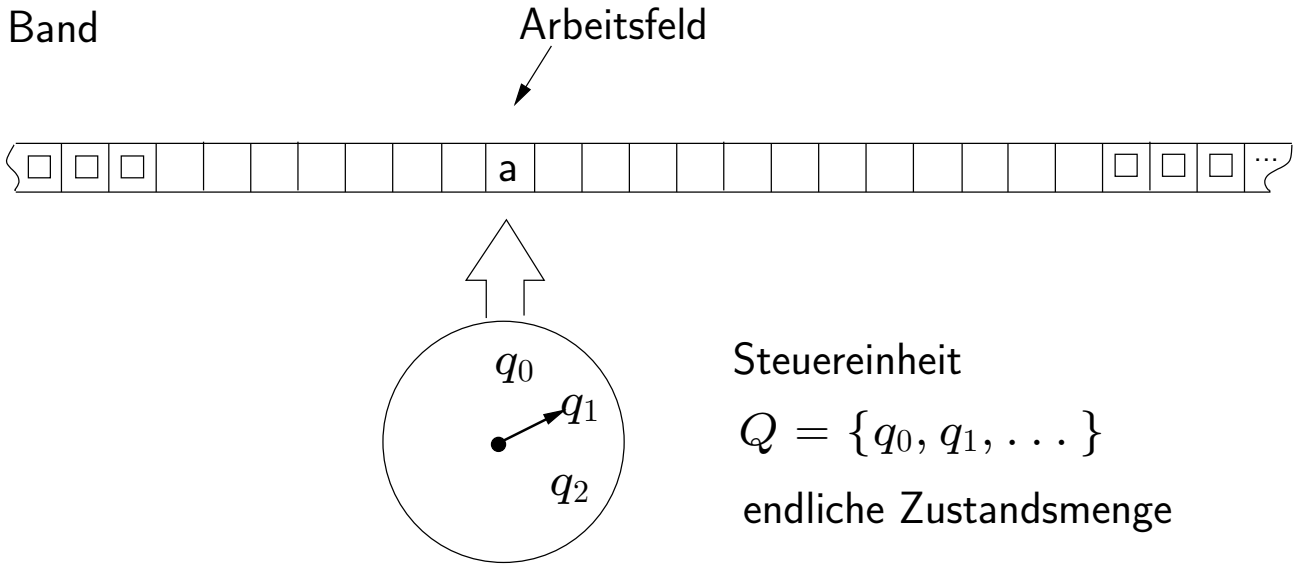
$f = go(h_1, \ldots, h_m), \quad f = R(g, h), \quad f = \mu.g$

Lassen sich durch Goto-Programme berechnen, falls $g, h_1, \ldots, h_m, h$ goto-berechenbar.

- Zeige die Funktion $I_{\mathcal{P}}$ lässt sich durch eine primitiv rekursive Funktion simulieren. Dann Iteration und Minimierung.

Turingmaschinen (nach A. Turing)

$\Sigma = \{b_1, \ldots, b_r\}$ Alphabet, Leersymbol $\square \notin \Sigma$, $a \in \Sigma$



Zu jedem Zeitpunkt sind nur endlich viele Felder nicht mit $\square$ belegt. Es gibt somit stets zusammenhängenden Block endlicher Länge, der das A-Feld enthält und außerhalb davon nur Leerzeichen vorkommen.

Erlaubte Operationen:

In Abhängigkeit vom Zustand und Inhalt des A-Felds schreibe Zeichen ins A-Feld, bewege Lese-Schreibkopf um ein Feld nach links (L), rechts (R) oder bleibe darauf (S), ändere Zustand.

Beschreibung durch „**Übergangsfunktion**“
$\delta : Q \times \Gamma \longrightarrow Q \times \Gamma \times \{L, R, S\}$, wobei Q endliche Zustandsmenge und Γ Bandalphabet sind.

Turingmaschinen (Forts.)

6.82 Definition

Eine Turingmaschine T ist ein 6 -Tupel $T = (Q, \Sigma, \Gamma, \delta, q_0, F)$ mit folgenden Bestandteilen:

- Q ist endliche **Zustandsmenge**.
- Σ Eingabealphabet mit $\square \notin \Sigma$. **Eingabezeichen.**
- Γ Bandalphabet mit $\Sigma \subseteq \Gamma$ und $\square \in \Gamma$. **Bandzeichen**.
- q_0 ist der **Startzustand**.
- $F \subseteq Q$ Menge der **Endzustände**.
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, S\}$ (oft als total verlangt) genügt $dom(\delta) = (Q \backslash F) \times \Gamma$. **Übergangsfunktion**.
  Wird oft als Tafel oder Tabelle angegeben.

Ein **Bandzustand** von T ist ein Tripel (q, x, β) mit $q \in Q$ (aktueller Zustand), $x \in \mathbb{Z}$ (aktuelle Kopfposition), $\beta : \mathbb{Z} \to \Gamma$ totale Funktion (aktueller Bandinhalt) mit $\beta(y) = \square$ für alle bis auf endlich viele $y \in \mathbb{Z}$.

Turingmaschinen (Forts.)

T überführt den Bandzustand (q, x, β) in den Bandzustand (q', x', β') (**Folgezustand**), falls

- $\delta(q, \beta(x)) = (q', \beta'(x), M)$
- $\beta'(y) = \beta(y)$ für alle $y \neq x$ Folgezustand
- $x' = \begin{cases} x - 1 & \text{falls } M = L \\ x + 1 & \text{falls } M = R \\ x & \text{falls } M = S \end{cases}$

Eine **Rechnung** von T ist eine endliche Folge von Bandzuständen $(z_0, \ldots, z_n)$, so dass T für alle $0 \leq i < n$ den Zustand z_i in z_{i+1} überführt.

Eine Rechnung heißt haltend, falls $z_n = (q, x, \beta) \wedge q \in F$.

6.83 Beispiel

$\Sigma = \{1, 2\}$, $\Gamma = \Sigma \cup \{\Box\}$, $Q = \{q_0, q_1, q_2\}$, $F = \{q_2\}$

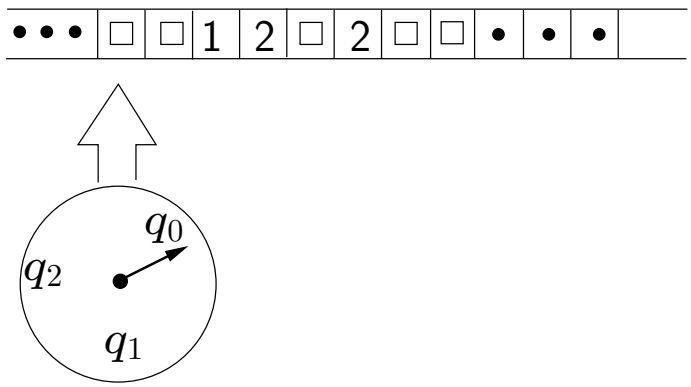
δ	$\Box$	1	2
q_0	$(q_0, \Box, R)$	$(q_1, \Box, R)$	$(q_1, \Box, R)$
q_1	$(q_2, \Box, S)$	$(q_1, \Box, R)$	$(q_1, \Box, R)$
q_2	$(q_2, \Box, S)$	$(q_2, 1, S)$	$(q_2, 2, S)$

Andere Beschreibungen von δ möglich: z.B.

Fünftupel $\{q\ b\ q'\ b'\ M : q \in Q, b \in \Gamma\}$

Beispiele von Turingmaschinen

Beispiel Rechnung:



Anfangszustand $z_0 = (q_0, 0, \beta)$ wobei $\beta(2) = 1$
$\beta(3) = 2$
$\beta(5) = 2$
sonst $\square$

$z_1 = (q_0, 1, \beta)$
$z_2 = (q_0, 2, \beta)$
$z_3 = (q_1, 3, \beta_1)$ mit $\beta_1(3) = 2 = \beta_3(5)$ sonst $\square$
$z_4 = (q_1, 4, \beta_2)$ mit $\beta_3(5) = 0$ sonst $\square$
$z_5 = (q_2, 4, \beta_2)$ $\ni q_2$ haltend oder „Haltezustand“
$z_6 = (q_2, 4, \beta_2)$ „Endzustand“

Wirkung: TM sucht rechts vom A-Feld $w \in \Sigma^*$ als Block und löscht es. Bleibt auf Leerzeichen hinter w stehen, falls $w \in \Sigma^+$ existiert. Stoppt nicht, falls auf AFeld und rechts davon lauter $\square$-Zeichen sind.

Turing-berechenbare Funktionen

Unäre Codierung von Zahlen $n \longrightarrow \underbrace{|||\ldots|}_{n}$

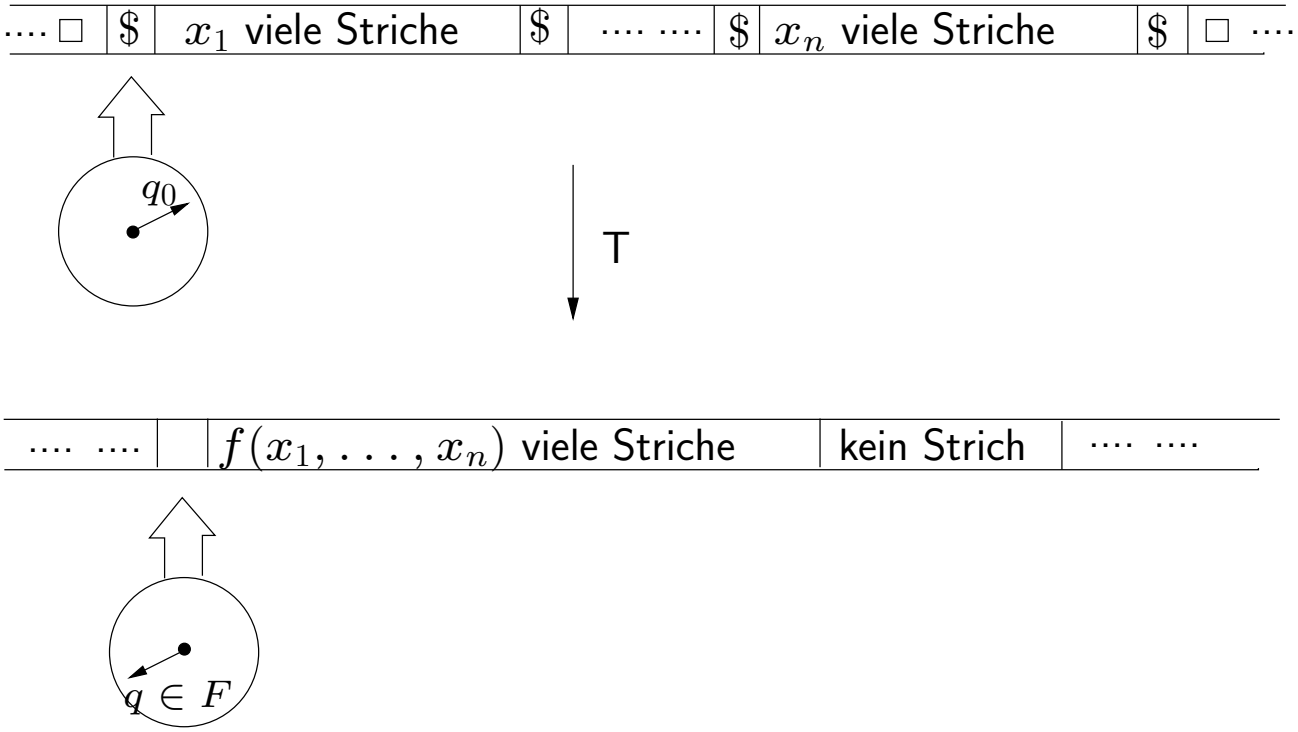
6.84 Definition

Eine Funktion $f : \mathbb{N}^n \longrightarrow \mathbb{N}$ heißt **Turing-berechenbar**, falls es eine TM T mit Eingabealphabet $\{|, \$\}$ gibt, so dass der Bandzustand $(q_0, 0, \beta)$ mit

- $\beta(i) = \square$ für $i < 0$ und $i > x_1 + x_2 + \cdots + x_n + n$
- $\beta(0) = \beta(x_1 + 1) = \beta(x_1 + x_2 + 2) = \cdots = \beta(x_1 + \cdots + x_n + n) = \$$
- $\beta(i) = |$ für alle anderen i

genau dann zu einer haltenden Rechnung ergänzt werden kann, wenn $f(x_1, \ldots, x_n) \downarrow$ und, ist in diesem Fall (q, i, β') der Zustand, in dem die Rechnung hält, dann ist die Anzahl der Striche $|$, die in $\beta'(i + 1), \beta'(i + 2), \ldots$ unmittelbar aufeinanderfolgen, gleich $f(x_1, \ldots, x_n)$.

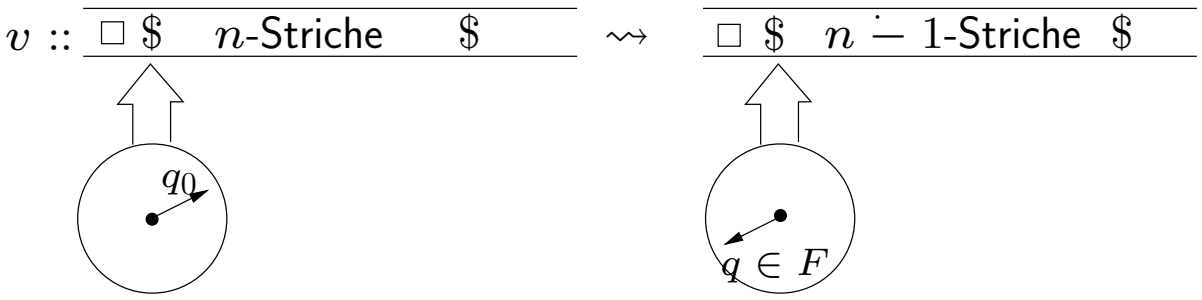
Turing-berechenbare Funktionen (Forts.)



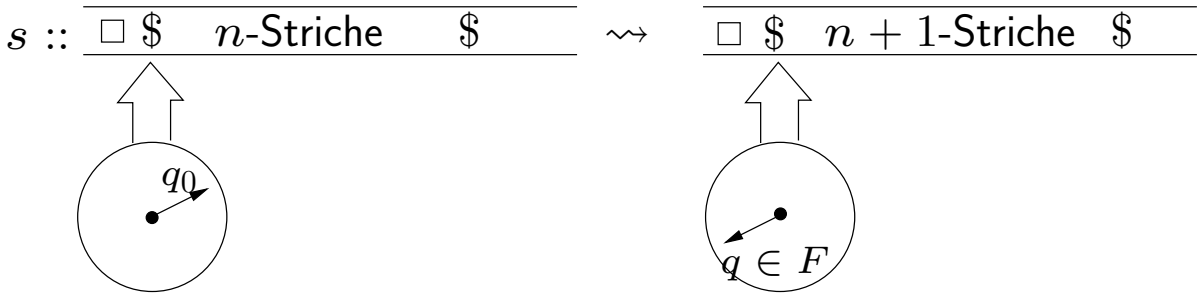
Beispiele

6.85 Beispiel

1. Vorgänger und Nachfolger: $v(x) = n - 1, \;\; s(n) = n + 1$



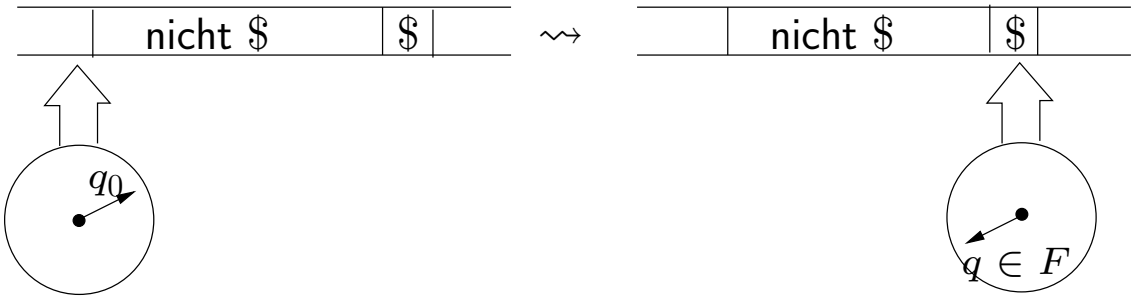
$$
\begin{array}{lll}
\delta(q_0, \$) & = (q_1, \square, R) & \Sigma = \{|, \$\} \\
\delta(q_1, |) & = (q_3, \$, S) & \Gamma = \{|, \$, \square\} \\
\delta(q_1, \$) & = (q_2, \$, L) & Q = \{q_0, q_1, q_2, q_3\} \\
\delta(q_2, \square) & = (q_3, \$, S) & F = \{q_3\}
\end{array}
$$



$$
\begin{array}{lll}
\delta(q_0, \$) & = (q_0, |, L) & \Sigma = \{|, \$\} \\
\delta(q_0, \square) & = (q_1, \$, S) & \Gamma = \{|, \$, \square\} \\
 & & Q = \{q_0, q_1\} \\
 & & F = \{q_1\}
\end{array}
$$

Beispiele (2)

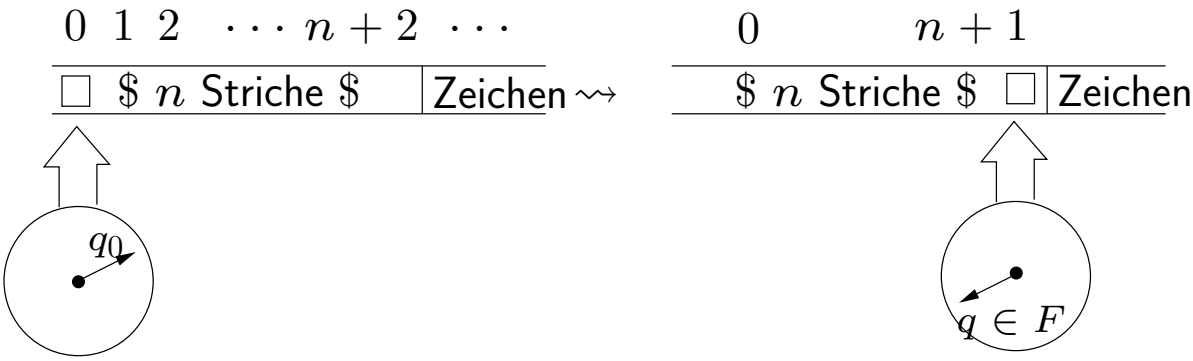
2. Suche rechts vom A-Feld erstes Vorkommen von \$, bleibe dort stehen.



$$\begin{aligned}
\delta(q_0, b) &= (q_1, b, R) \qquad b \in \Gamma \\
\delta(q_1, b) &= (q_1, b, R) \qquad b \in \Gamma \backslash \$ \\
\delta(q_1, \$) &= (q_2, \$, S)
\end{aligned}$$

SL\$. Analog SR\$

3. Verschiebe Block \$$n$-Striche\$ um ein Feld nach links



Beispiele (2) (Forts.)

$\Sigma = \{|, \$\}$
$\Gamma = \Sigma \cup \{\Box\}$
$Q = \{q_0, \ldots, q_5\}$
$F = \{q_5\}$

$q_0\ b\ \$\ R\ q_1$
$q_1\ b\ b\ R\ q_2$
$q_2\ |\ |\ L\ q_3$
$q_2\ \$\ \Box\ L\ q_4$
$q_3\ b\ |\ R\ q_1$
$q_4\ b\ \$\ R\ q_5$
$q_5\ b\ b\ S\ q_5$

$b \in \Gamma$

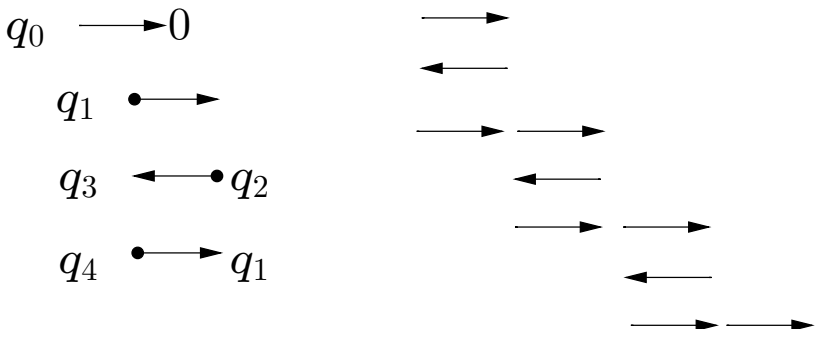
VL. Analog VR (verschiebe nach rechts).

$\delta(q_2, \$) = (q_4, \Box, L)$

q_3 merkt sich |

q_4 merkt sich \$

Strategie:



Simulation von RM-Programme durch TM

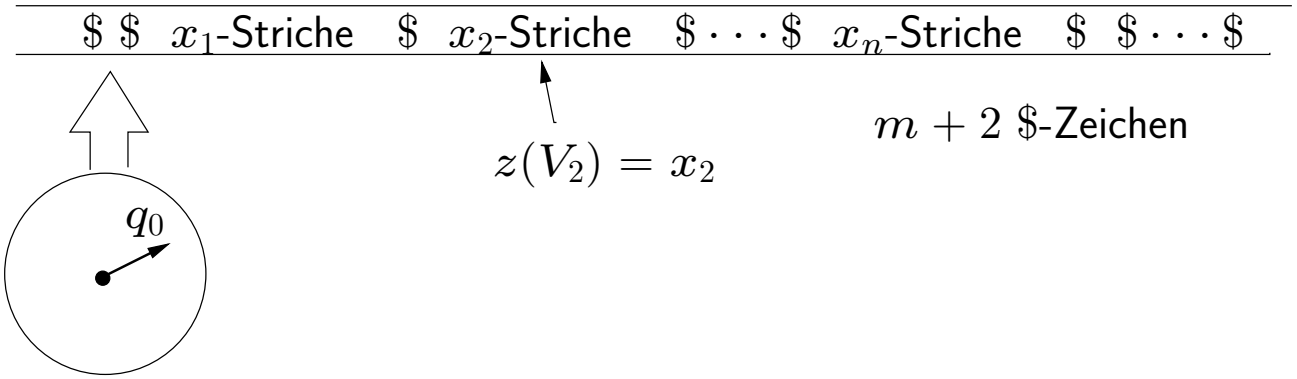
6.86 Lemma

• Jede RM (goto)-berechenbare Funktion lässt sich durch eine TM berechnen. Also ist jede μ-rekursiv Funktion TM-berechenbar.

• Jede Turing-berechenbare Funktion ist μ-rekursiv.

Beweisidee:

Simuliere Berechnung des goto-Programms über $V_0, \ldots, V_m$, $f : \mathbb{N} \longrightarrow \mathbb{N}$. Speichere Zustand $z : V \longrightarrow \mathbb{N}$ als



Ein-Befehl wird durch mehrere TM-Schritte simuliert.
Die Zustände entsprechen Marken im Goto-Programm.
$V_i := s(V_i)$

- Verschiebe die Blöcke vor V_i jeweils um ein Feld nach links wie oben.
- Wende s TM an.
- SL$ i-mal.

Simulation von TM durch μ rekursive Funktionen

Simulation der Überführungsfunktion einer Turing-Maschine durch eine primitiv-rekursive Funktion $i_T : \mathbb{N} \longrightarrow \mathbb{N}$, die auf geeignet codierten Bandzuständen arbeitet (q, x, β).

Dann wie üblich.

Wir haben somit weitere Charakterisierungen der μ-rekursiven Funktionen, die die Churchsche These untermauern.

Man kann für $\mathcal{P}(\mathbb{N})$ (primitiv-rekursiven Funktionen) ebenfalls eine Charakterisierung mit Hilfe einfacher Programmiersprachen finden.

z. B. For-Programme über $\mathbb{N}$

Anweisungsfolgen:

Anweisung: Zuweisung, Test oder For-Schleife der Form:

$$\underline{\textbf{for}}\ I = 0\ \underline{\textbf{to}}\ J\ \underline{\textbf{do}}\ \alpha\ \underline{\textbf{end}};$$

$I, J \in V$ α For-Programm über V, das keine Zuweisung der Form $I := t$ oder $J := t$ mit Term t enthält (Schleife wird genau $z(J)$ mal ausgeführt, dabei wird stets α ausgeführt und I um 1 erhöht).

6.6 Berechenbarkeit auf Zeichenreihen Wortfunktionen

Wortfunktionen: $f : (\Sigma^*)^n \to \Sigma^*$

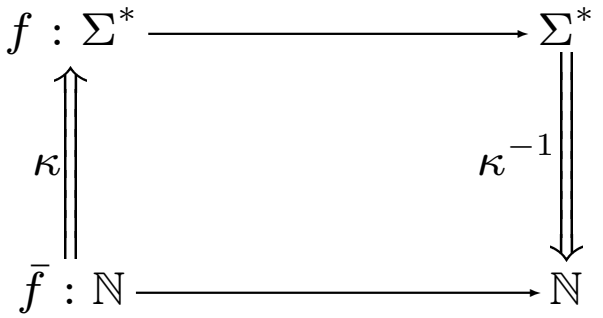
Wortrelationen $R \subseteq (\Sigma^*)^n$ **Sprachen** $R \subseteq \Sigma^*$

Bisher: Funktionen, Relationen auf $\mathbb{N}$: μ-rekursive Funktionen.

Turing-Maschinen und While-Programme sind für beliebige Alphabete bzw. beliebige Strukturen definiert.

Verallgemeinerung der Ergebnisse der Rekursionstheorie, insbesondere über Entscheidbarkeit und Nichtentscheidbarkeit auf Wortfunktionen und Relationen.

1. Möglichkeit: Codierung von Σ^* in $\mathbb{N}$. Einfache effektive Codierungen: z. B. Folgencodierungsfunktion oder Interpretation als Zahl (binäre-, dezimale-Darstellung).



$$\bar{f}(n) = \kappa^{-1}(f(\kappa(n)))$$

Berechenbarkeit auf Zeichenreihen (2)

Definition $f \in \mathcal{R}_p(\Sigma)$ gdw $\bar{f} \in \mathcal{R}_p(\mathbb{N})$.

Zeige: Unabhängig von der gewählten effektiven Codierung.

2. Möglichkeit: $\Sigma = \{a_1, \ldots, a_n\}$ $(n \geq 1)$

While-Programme:

Betrachte die Algebra

$String = (\Sigma^*, \varepsilon, succ_{a_1}, \ldots, succ_{a_n}, pred)$ mit

- $succ_a(u) = au \quad (a \in \Sigma)$
- $pred(au) = u \quad pred(\varepsilon) = \varepsilon$

Ordnungen auf Σ^*: $\leq_{llex}$ Länge-Lexikographisch,

d. h. $u \leq_{llex} v$ gdw $|u| < |v|$ oder
$|u| = |v| \wedge u \leq_{lex} v$

Wobei $\leq_{lex}$ lex. Ordnung, die von lin. Ordnung auf Σ induziert wird (z. B. $a_1 < a_2 < a_3 \cdots < a_n$).

Beachte: $|\cdot| : \Sigma^* \to \Sigma^* \quad |u| = a_1^{|u|}$ und $\chi_{\leq llex}$ sind while-berechenbar.

3. Möglichkeit: **μ-rekursive Funktionen über Σ^***: $a \in \Sigma$

- $f^{(n)}_{NULL}(\vec{w}) = \varepsilon$, $f^{(n)}_{SUCC_a}(\vec{w}) = aw_1$ $(\vec{w} = (w_1, \ldots, w_n))$
- $f^n_{PROJ(i)}$ wie bisher.

Berechenbarkeit auf Zeichenreihen (3)

Komposition: $f \circ (g_1, \ldots, g_n)$ wie bisher.

Primitive Rekursion: $f = R_\Sigma(g, h_1, \ldots, h_n)$, falls

- $f(\vec{u}, \varepsilon) = g(\vec{u}, \varepsilon)$
- $f(\vec{u}, a_i v) = h_i(\vec{u}, f(\vec{u}, v), v)$

Minimierung: $f(\vec{u}) = \mu_{llex} v \cdot g(\vec{u}, v) = \varepsilon$

$f(\vec{u}) = w$ $llex$-minimal mit $g(\vec{u}, w) = \varepsilon$, sofern ein solches existiert.

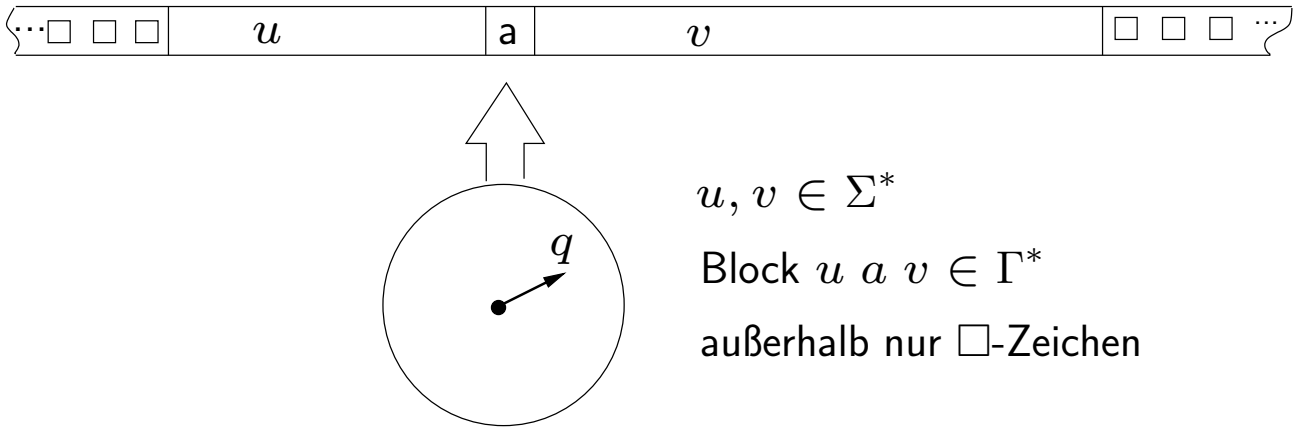
4. Möglichkeit: **RM (Goto-Programme)**: $z(V_i) \in \Sigma^*$

Befehle:

- $X := s(a, X)$ Wirkung wie $succ_a$ in Σ^*
- $X := p(X)$ Wirkung wie pred in Σ^*
- **if** $X = \varepsilon$ **then goto** l_1 **else goto** l_2
- Test (Anfangsbuchstabe ist $a \in \Sigma$):
  if $AB(X) = a$ **then goto** l_1 **else goto** l_2

Berechenbarkeit auf Zeichenreihen (4)

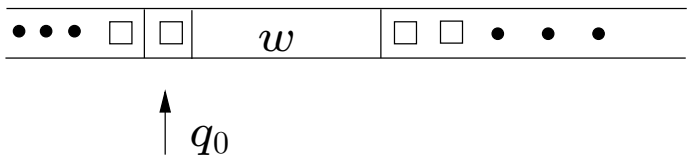
5. Möglichkeit:**Turing-Maschinen**:
$T = (Q, \Sigma, \Gamma \supseteq \Sigma \cup \{\square\}, \delta, q_0, F \subseteq \Gamma)$



Konfiguration: $u\ q\ a\ v \in \Gamma^* \cdot Q \cdot \Gamma^+ \quad (\Gamma^+ = \Gamma^* \backslash \{\varepsilon\})$

Zwei Konfigurationen heißen **äquivalent**, falls sie sich nur durch Blöcke von □-Zeichen davor und danach unterscheiden.

Anfangskonfigurationen: $q_0\ \square\ w \qquad w \in \Sigma^*$



Berechenbarkeit auf Zeichenreihen (5)

Folgekonfigurationen

k' ist Folgekonfiguration von k : $k \underset{T}{\vdash} k'$, falls gilt

k	$\delta(q,a)$	k'	
$u\ q\ a\ v$	(q', a', S)	$u\ q'\ a'\ v$	
$u\ q\ a\ v$	(q', a', R)	$u\ q'\ a'\ v$	$v \in \Gamma^+$
$u\ q\ a$	(q', a', R)	$u\ q'\ a'\ \square$	
$u\ b\ q\ a\ v$	(q', a', L)	$u\ q'\ b\ a'\ v$	$b \in \Gamma$
$q\ a\ v$	(q', a', L)	$q'\ \square\ a'\ v$	

Eine **Rechnung** einer TM T ist Folge von Konfigurationen $(k_0, \ldots, k_n)$ mit $k_i \underset{T}{\vdash} k_{i+1}$. Sie ist haltend, falls k_n eine Endkonfiguration ist, d. h. $k_n = u\ q\ v$ mit $q \in F$. Schreibe $k_0 \overset{*}{\vdash}_T k_n$

Eine **Berechnung** einer TM T ist eine Rechnung wobei k_0 eine Anfangskonfiguration $(k_0 = q_0\ \square\ w)w \in \Sigma^*$ ist.

TM-berechenbare Funktionen: $f : (\Sigma^*)^n \to \Sigma^*$ ist TM-berechenbar, falls es eine TM T gibt die f berechnet, d. h.

a) T stoppt für Anfangskonfiguration $k_0 = q_0 \square x_1 \square x_2 \square \cdots \square x_n \square$ $\text{gdw } (x_1, \ldots, x_n) \in dom(f)$
b) Gilt $(x_1, \ldots, x_n) \in dom(f)$ und $y = f(x_1, \ldots, x_n)$, so hat T beim Stopp die Konfiguration $\square^i\ q \square x_1 \square \cdots \square x_n \square y \square^j$, für geeignete $i, j \in \mathbb{N}$.

Berechenbarkeit auf Zeichenreihen (6)

6.87 Satz

$f : (\Sigma^*)^n \to \Sigma^*$, dann sind äquivalent

- $f \in \mathcal{R}_p(\Sigma)$, d. h. f ist μ-rekursiv.
- f ist while-programmierbar über String.
- f ist RM-(goto)-berechenbar.
- f ist TM-berechenbar.

Existenz universeller Funktionen, universeller Programme und universeller Maschinen wie bisher.

- Relationen: Entscheidbarkeit, rek-Aufzählbarkeit
  $R \subseteq (\Sigma^*)^n$
- R entscheidbar $\mathrm{gdw}\ \chi_R \in \mathcal{R}_p(\Sigma), \chi_R(\vec{w}) = \begin{cases} \varepsilon & w \notin R \\ a_1 & w \in R \end{cases}$
- R rekursiv-aufzählbar $\mathrm{gdw}\ R = dom(f)$, $f \in \mathcal{R}_p(\Sigma)$.
- Halteproblem: $K_0 = \{(T, w) \mid T$ mit Anfangskonfiguration $q_0 \,\square\, w$ hält, d. h. Berechnung mit Endkonfiguration$\}$

Ist nicht entscheidbar.

Bisherige Ergebnisse lassen sich übertragen.

Insbesondere: Reduzierbarkeit $\leq_m$.

Turing-Maschinen als Akzeptoren von Sprachen und als entscheidende Automaten

6.88 Definition Akzeptierende und erkennende TM

Sei $T = (Q, \Sigma, \Gamma \supseteq \Sigma \cup \{\Box\}, \delta, q_0, F)$

- T **akzeptiert** die Sprache $L \subseteq \Sigma^*$ gdw für
  $w \in \Sigma^* : q_0 \Box w \overset{*}{\vdash}_T u\ q\ v$ mit $q \in F$ gdw $w \in L$,
  d. h. es gibt haltende Berechnung aus $q_0 \Box w$ gdw $w \in L$,
  $L = L(T)$.
- T **entscheidet** die Sprache $L \subseteq \Sigma^*$ gdw für jede Eingabe
  $w \in \Sigma^*$ hält T: $q_0 \Box w \overset{*}{\vdash}_T u\ q\ v$ mit $q \in F$ und
  $w \in L$, so $q = q_y$ $\quad$ $w \notin L$, so $q = q_n$
  wobei $q_y, q_n \in F$ spezielle „Ja“-, „Nein“- Zustände sind.

6.89 Lemma

- $L \subseteq \Sigma^*$ ist entscheidbar gdw es gibt eine TM T, die L entscheidet.
- $L \subseteq \Sigma^*$ ist rekursiv aufzählbar gdw es gibt eine TM T, die L akzeptiert, d. h. $L = L(T)$.

Beachte: Andere Konventionen sind möglich. Andere TM: Mehrband TM, δ unvollständig, Band einseitig unendlich, mehrspurig, nicht deterministisch.

Beispiele

Turing-Programme

- Turing Befehl hat die Form

  $B \equiv Op$ $\quad Op \in \Gamma \dot{\cup} \{R, L, \text{stopp}\}$

  $B \equiv q$ $\quad q \in Q$ unbedingter Sprung

  $B \equiv a, q$ $\quad a \in \Gamma, q \in Q$ bedingter Sprung nach q, falls a in A-Feld
- Turing Programm ist endliche Folge markierter Befehle

  $Q = \{q_0, q_1, \ldots, q_n\}$, $q_i \neq q_j$ für $i \neq j$

  TP:: $q_0 : B_0$ $\quad B_i$ Turing-Befehl

  $q_1 : B_1$

  $\vdots$

  $q_n : B_n$
- Semantik eines T-Programms durch Angabe der TM

  $T = (Q', \Sigma, \Gamma, \delta, q_0, F)$, $a \in \Gamma$, $Q' = Q \cup \{q_{n+1}\}$

  $$\begin{aligned} \delta(q_i, a) &= (q_{i+1}, a', S) && B_i \equiv a' \in \Gamma \\ &= (q_{i+1}, a, M) && B_i \equiv M \in \{L, R\} \\ &= (q_{i+1}, a, S) && B_i \equiv a', q \;\; a \neq a' \\ &= (q, a, S) && B_i \equiv a, q \\ &= (q_{n+1}, a, S) && B_i \equiv \text{ stopp oder } i = n+1 \end{aligned}$$

  $F = \{q_{n+1}\}$
- Eigenschaft:Jede TM kann durch ein äquivalentes T-Programm beschrieben werden.

Beispiele

Suche Links von AF das erste Vorkommen von $\square$ ... Rechts ...

$SL\ \square:$	L	$SR\ \square:$	R
	$\square, Fin$		$\square, Fin$
	$SL\ \square$		$SR\ \square$
$Fin:$	Stopp	$Fin:$	Stopp

TM, die die Menge der Palindrome über $\{a, b\}^*$ entscheidet
$L = \{w \in \{a, b\}^* : w = w^{mi}\}$

q_0:	R	q_b:	$\square$
	$\square : q_y$		$SR\ \square$
	$a : q_a$		L
	$b : q_b$		$\square : q_y$
			$b : q$
			$a : q_n$
q_a:	$\square$	q:	$\square$
	$SR\ \square$		$SL\ \square$
	L		q_0
	$\square : q_y$		
	$a : q$		
	$b : q_n$		

Diese Turing Programm hält für jede Eingabe $w \in \Sigma^*$ und entscheidet die Menge der Palindrome.

Simulation von TM-Berechnungen durch Wortersetzungssystemen (Σ, Π)

Π ist Menge von Produktionen $l ::= r$, mit $l \in \Delta^+$, $r \in \Delta^*$

Kalkül: $\dfrac{u\ l\ v}{u\ r\ v}$ für $l ::= r \in \Pi$, $u, v \in \Delta^*$.

Sei

$T = (Q, \Sigma, \Gamma, \delta, q_0, F)$ und $\Delta = Q \dot\cup \Gamma \dot\cup \{\#\}$

Produktionen Π_T:

Für jedes $\delta(q, a) = (q', a', S)$: $\quad q\ a ::= q'\ a' \in \Pi_T$.

Für jedes $\delta(q, a) = (q', a', R)$ und $b \in \Gamma$:

$$q\ a\ b ::= a'\ q'\ b \in \Pi_T$$
$$q\ a\ \# ::= a'\ q'\ \Box\ \# \in \Pi_T$$

Für jedes $\delta(q, a) = (q', a', L)$ und $b \in \Gamma$

$$b\ q\ a ::= q'\ b\ a' \in \Pi_T$$
$$\#q\ a ::= \#q'\ \Box\ a' \in \Pi_T$$

Offenbar gilt:

$k = u\ q\ v \underset{T}{\vdash} u'\ q'\ v' = k'$, d. h. k' ist Folgekonfiguration von k

$\text{gdw} \qquad \#u\ q\ v\# \overset{1}{\underset{\Pi_T}{\vdash}} \#\ u'\ q'\ v'\ \#$,

d. h. Rechnungen der TM T können in Π_T simuliert werden.

$\#q_0\ \Box\ w\ \# \underset{\Pi_T}{\vdash} \#\ u\ q\ v\ \# \quad \text{gdw} \quad q_0\ \Box\ w \overset{*}{\underset{T}{\vdash}} u\ q\ v$

für $w \in \Sigma^*$, $u, v \in \Gamma^*$, $q \in Q$.

Das Ableitbarkeitsproblem

6.90 Definition Sei (Σ, Π) ein Wortersetzungssystem.

Das **Ableitbarkeitsproblem** $Abl \subset \Sigma^* \times \Sigma^*$ für (Σ, Π) ist gegeben durch

$$Abl\ x\ y \qquad \text{gdw} \qquad x \underset{\Pi}{\vdash} y$$

(für $x, y \in \Sigma^*$) d.h. „y lässt sich aus x mit Hilfe der Produktionen aus Π ableiten".

6.91 Satz Unentscheidbarkeit des Ableitbarkeitsproblems

Das Ableitbarkeitsproblem für beliebige Wortersetzungssysteme ist nicht entscheidbar.

Beweis:

Reduziere das Halteproblem für TM auf das Ableitbarkeitsproblem. Die Konstruktion TM $T \longrightarrow$ simulierendes Wortersetzungssystem Π_T ist effektiv. Für $q \in F$ füge noch die Produktionen
$a\ q ::= q$, $q\ a ::= q$, für $a \in \Delta \backslash \{\#\}$ und $\#\ q\ \# ::= q$ hinzu.

Dann gilt: T hält mit Eingabe w

$\text{gdw}\ \exists u, v \in \Gamma^*, q \in F$ mit $q_0\ \Box\ w \overset{*}{\underset{T}{\vdash}} u\ q\ v$

$\text{gdw}\ \exists u, v \in \Gamma^*, q \in F$ mit $\#\ q_0\ \Box\ w\ \# \underset{\Pi_T}{\vdash} \#\ u\ q\ v\ \#$

$\text{gdw}\ \exists q \in F$ mit $\#\ q_0\ \Box\ w\ \# \underset{\Pi_T}{\vdash} q.$

Also ist das Halteproblem auf das Ableitbarkeitsproblem reduzierbar.

Speziellere Ergebnisse (z.B. spezielles Wort ableitbar) sind möglich.

Das Postsche Korrespondenzproblem (PCP)

6.92 Definition

Das Postsche Korrespondenzproblem (**PCP**) besteht aus allen Listen von Wortpaaren

$$\mathcal{L} = (x_1 \sim y_1, \ldots, x_k \sim y_k) \qquad k \geq 1$$

mit nichtleeren Wörtern $x_i, y_i \in \Sigma^* (1 \leq i \leq k)$ zu denen es eine Indexfolge $i_1, \ldots, i_n \in \{1, \ldots, k\}$ mit $n \geq 1$ gibt, so dass

$(*)$ $\quad x_{i_1} \cdots x_{i_n} = y_{i_1} \ldots y_{i_n}$ gilt.

Schreibe: PCP($\mathcal{L}$). Die Folge $(i_1, \ldots, i_n)$ ist Lösung, falls $(*)$ gilt.

Einschränkungen: z. B. $i_1 = 1$ spezielle PCP (**SPCP**).

Beachte Parameter: $\Sigma, k, x_i, y_i \in \Sigma^+, 1 \leq i \leq k$

Lösung: Liste natürlicher Zahlen aus $\{1, \ldots, k\}$.

Beachte: Zu gegebener Liste $i_1, \ldots, i_n$ ist es einfach zu überprüfen, ob sie eine Lösung ist.

Beispiel

6.93 Beispiel $\Sigma = \{0, 1\}$

- $\mathcal{L}_1 = (0 \sim 0\,0\,0,\ 0\,1\,0\,0 \sim 0\,1,\ 0\,0\,1 \sim 1) \quad k = 3$
  Lösungen können nur mit $i_1 = 1$ oder $i_1 = 2$ beginnen.
  1. Lösung: $i_1 = 1, i_2 = 3 : \underline{0}\ \ \underline{0\,0\,1} = \underline{0\,0\,0}\ \ \underline{1}$.
  2. Lösung: $i_1 = 2$, $i_2 = 1$, $i_3 = 1$, $i_4 = 3$.

  $x_2x_1x_1x_3$ $\quad \underline{0\,1\,0\,0}\,\underline{0}\,0\,0\,0\,1$
  $y_2y_1y_1y_3$ $\quad \underline{0\,1}\,\underline{0\,0\,0}\,0\,0\,0\,1$

  d. h. PCP($\mathcal{L}_1$), SPCP($\mathcal{L}_1$)
- $\mathcal{L}_2 = (\underline{1} \sim \underline{1}\,1\,1, \underline{1\,0}\,1\,1\,1 \sim \underline{1\,0},\ 1\,0 \sim 0) \quad k = 3$
  Lösung: $2, 1, 1, 3$ (muss mit 1 oder 2 beginnen).

  $x_2x_1x_1x_3$ $\quad \underline{1\,0\,1\,1\,1}\,\underline{1}\,\underline{1}\,\underline{1\,0} \quad \underline{1}\,\underline{1}\cdots$ keine
  $y_2y_1y_1y_3$ $\quad \underline{1\,0}\,\underline{1\,1\,1}\,\underline{1\,1\,1}\,\underline{0} \quad \underline{1\,1\,1}\,\underline{1\,1\,1}\cdots$ Lösung

  d. h. PCP($\mathcal{L}_2$), $\neg$ SPCP($\mathcal{L}_2$)
- $\mathcal{L}_3 = (\underline{0\,1} \sim \underline{0\,1}\,0, 1\,\underline{0\,0} \sim \underline{0\,0}, 0\,1\,0 \sim 1\,0\,0)$
  Behauptung: $\neg$ PCP($\mathcal{L}_3$):: Lösung muss mit 1 beginnen und mit 2 enden. $t \in \{1, 2, 3\}^+$ $t = 1t'2$.

  $\underline{0\,1}\,\underline{0\,1\,0}\,\underline{0\,1\,0}\cdots$ $\quad t'$ muss mit 3 beginnen
  $\underline{0\,1\,0}\,\underline{1\,0\,0}\,\underline{1\,0\,0}\cdots$

  Keine Fortsetzung möglich, da kein y mit 1 endet
  $\cdots\underline{1\,0\,0}$
  $\cdots\underline{0\,0}$

Beispiel (Forts.)

- $\mathcal{L}_4 = (0\,0\,1 \sim 0, 0\,1 \sim 0\,1\,1, 0\,1 \sim 1\,0\,1, 1\,0 \sim 0\,0\,1)$
  Es gilt PCP($\mathcal{L}_4$) aber $\neg$ SPCP($\mathcal{L}_4$).
  Lösung etwas länger: mit 2 beginnen, mit 3 enden, 1 muss verwendet werden.

$\underline{0\,1}\;\underline{1\,0}\,0\,1\,0\,1\,1\,0$ keine Forts.

$\underline{0\,1\,1}\;\underline{0\,0\,1}\,0\,1\,1\,0\,1\,1\,0\,0\,1$

$\underline{0\,1}\;\underline{1\,0}\;\underline{0\,1}\,1\,0\,1\,0\,0\,1\,0\,0\,1\,0\,1\,1\,0\,0\,1\,1\,0\cdots$

$\underline{0\,1\,1}\;\underline{0\,0\,1}\;\underline{1\,0\,1}\,0\,0\,1\,0\,0\,1\,0\,1\,1\,0\,\underline{0}\;\underline{1}\;\underline{1}\,0\,0\,1\,1\,0\,1\,0\,0\,1$

Unentscheidbarkeit von PCP

6.94 Satz Das PCP ist unentscheidbar.

Beweisidee: Reduziere Ableitbarkeitsproblem für WES auf PCP.

- Sei $\gamma = (\Sigma, \Pi)$ ein Wortersetzungssystem ohne ε-Regeln.
  $Abl_\gamma(u, v)$ gdw $u \underset{\Pi}{\vdash} v$
- Konstruktion: Aus $\gamma, u, v \rightsquigarrow \mathcal{L}_{u,v}$ mit $u \underset{\Pi}{\vdash} v$ gdw PCP($\mathcal{L}_{u,v}$).
  Sei $\gamma = (\Sigma, \Pi)$, $\Pi = \{u_i ::= v_i \mid i = 1, \ldots, n\}$,
  $\Sigma = \{a_1, \ldots, a_r\}$. $\hat{\Sigma} = \{\hat{a}_1, \ldots, \hat{a}_r\}$ Kopie von Σ und
  $\Gamma = \Gamma \dot\cup \hat{\Sigma} \dot\cup \{+, \hat{+}, [,]\}$.
- $u, v \in \Sigma^*$. Definiere $\mathcal{L}_{u,v}$ über Γ durch

$$
\begin{array}{lcccccccccccccccccccc}
X_i ::& [u & + , & &] , & & + , & \hat{+} , & a_1 , & \ldots , & a_r , & \hat{a}_1 , & \ldots , & \hat{a}_r , & v_1 , & \ldots , & v_n , & \hat{v}_1 , & \ldots , & \hat{v}_n \\
& & \wr & & \wr & & \wr & \wr & \wr & & \wr & \wr & & \wr & \wr & & \wr & \wr & & \wr \\
Y_i ::& [& & , \hat{+} v] , & & \hat{+} , & + , & \hat{a}_1 , & \ldots , & \hat{a}_r , & a_1 , & \ldots , & a_r , & \hat{u}_1 , & \ldots , & \hat{u}_n , & u_1 , & \ldots , & u_n \\
\text{NR}: & 1 & & 2 & & 3 & 4 & 5 & \ldots & l & m & & & & & & & & & n
\end{array}
$$

$l = 4 + r$, $m = 4 + r + 1$,$n = 4 + 2r + 2n$

Behauptung: PCP($\mathcal{L}_{u,v}$) gdw $u \underset{\Pi}{\vdash} v$ (o.b.d.A. $u \neq v$).

„$\Leftarrow$“ $u \underset{\Pi}{\vdash} v$ etwa $u = w_0 \overset{(1)}{\underset{\Pi}{\vdash}} w_1 \overset{(1)}{\underset{\Pi}{\vdash}} \cdots \overset{(1)}{\underset{\Pi}{\vdash}} w_k = v \quad k > 0$

Es gibt $t_j \in J^*$, $j = 1, \ldots, k$ mit

j-gerade
$$X(t_j) = [w_0 + \hat{w}_1 \hat{+} w_2 + \cdots + \hat{w}_{j-1} \hat{+} w_j$$
$$Y(t_j) = [w_0 + \hat{w}_1 \hat{+} w_2 + \cdots + \hat{w}_{j-1}$$

j-ungerade
$$X(t_j) = [w_0 + \hat{w}_1 \hat{+} w_2 + \cdots \hat{+} w_{j-1} + \hat{w}_j$$
$$Y(t_j) = [w_0 + \hat{w}_1 \hat{+} w_2 + \cdots \hat{+} w_{j-1}$$