Theoretische Informatik II

Einheit 6.4

Andere Berechenbarkeitsmodelle

1. Typ-0 Berechenbarkeit
2. λ-Kalkül
3. Church'sche These

Typ-0 Berechenbarkeit

- $G = (N, T, P, \sigma)$ **Typ-0 Grammatik**
  - N, T: Alphabete (Nonterminalsymbole / Terminalsymbole)
  - P: Menge von Produktionen $\alpha \to \beta$ (α enthält Nonterminalsymbole)
  - σ: Startsymbol
- **Ableitbarkeit** $v \to_G^* w$
  - $v = u_1\alpha_1 z_1 \to u_1\beta_1 z_1 = u_2\alpha_2 z_2 \to u_2\beta_2 z_2 \to \ldots \to u_n\beta_n z_n = w$
- **Erzeugte Sprache** $L(G) = \{w \in T^* \mid \sigma \to_G^* w\}$
- $L \subseteq T^*$ **Typ-0 berechenbar**
  - L wird von einer Typ-0 Grammatik G erzeugt ($L = L(G)$)
  - G "zählt Elemente von L auf"
- $f: X^* \to Y^*$ **Typ-0 berechenbar**
  - $L_f = \{v\#w \mid f(v) = w\}$ Typ-0 berechenbar

L Typ-0 berechenbar $\Leftrightarrow$ L semi-entscheidbar Satz Q

$\Rightarrow$ Simuliere Produktionsregeln der Grammatik G

– Schreibe Wort w auf Band 1 und σ auf Band 2

– In Phase i schreibe alle Worte von $L(G)$ auf Band 2, die in i Schritten ableitbar sind und teste, ob w auf Band 2 vorkommt

– Im Erfolgsfall gebe 1 aus, andernfalls beginne Phase i+1

– Programm terminiert nicht, wenn $w \notin L$ (Ergebnis ist $\perp$)

$\Leftarrow$ Simuliere Abarbeitung der Turingmaschine $\boldsymbol{\tau}$

– Codiere Konfigurationen (s,u,v) als Worte usv

– Simuliere Konfigurationsübergänge als Regeln mit Begrenzern

– Entferne Begrenzer, wenn (Simulation von) τ angehalten hat

Beweis: Typ-0 berechenbar ⇒ semi-entscheidbar

- **Für $G = (N,T,P,\sigma)$ definiere Mengen M_i**
  - $M_0 = \{\sigma\}$, $M_{i+1} = \{u | \exists v \in M_i \; v \to_G u\}$
  - M_i ist die Menge der in i Schritten ableitbaren Worte
- **Beschreibe Mehrband-TM für Test $w \in L(G)$**
  - $L(G) = \bigcup M_i \cap T^*$ also $w \in L(G) \Leftrightarrow \exists i \; w \in M_i$
  - Schreibe w auf Band 1, erzeuge M_i auf Band 2 und vergleiche
  - Trennung zwischen Worten bzw. M_i und M_{i+1} durch Symbole #, $

M_0		M_1								M_2					
σ	$	#	$u_1^1..u_n^1$	#	$u_1^2..u_m^2$	# ... #	$u_1^j..u_k^j$	$	#	$v_1^1..v_l^1$	# ... #	$v_1^j..v_o^j$	$	...	

- **Iterative Erzeugung der M_i durch TM**
  - Codiere Produktionsregeln als Unterprogramme der Turingmaschine
  - Beginne mit $M_0 = \{\sigma\}$
  - Wende Regeln auf Worte von M_i an, schreibe Resultate ans Bandende
  - Beginne Vergleich mit w, wenn M_{i+1} vollständig erzeugt

Anmerkungen zum Beweis

- **TM berechnet partiell-charakteristische Funktion $\psi_{L(G)}$**
  - Erfolgreicher Vergleich bedeutet $w \in M_i \subseteq L(G)$ (Ergebnis 1)
  - $w \notin L(G)$ bedeutet $w \notin M_i$ für alle i: keine Terminierung (Ergebnis $\perp$)
- **Technische Details aufwendig**
  - Codierung der Regeln als Programme
  - Suchmechanismen: Anfang von M_i ($), nächstes Wort in M_i (#)
  - Erzeugung aller ableitbaren Worte durch Bestimmung aller anwendbaren Regeln (ggf. Anwendbarkeit einer Regel auf mehrere Teilworte)
  - Vergleich von w mit Worten auf Band 2, korrekte Ausgabe erzeugen
- **Optimierungen möglich**
  - Erzeuge M_{i+1} auf drittem Band
  - Überschreibe M_i, wenn M_{i+1} vollständig erzeugt

⋮

Projekt: Details heraussuchen, kurze Ausarbeitung schreiben

Beweis: semi-entscheidbar $\Rightarrow$ Typ-0 berechenbar

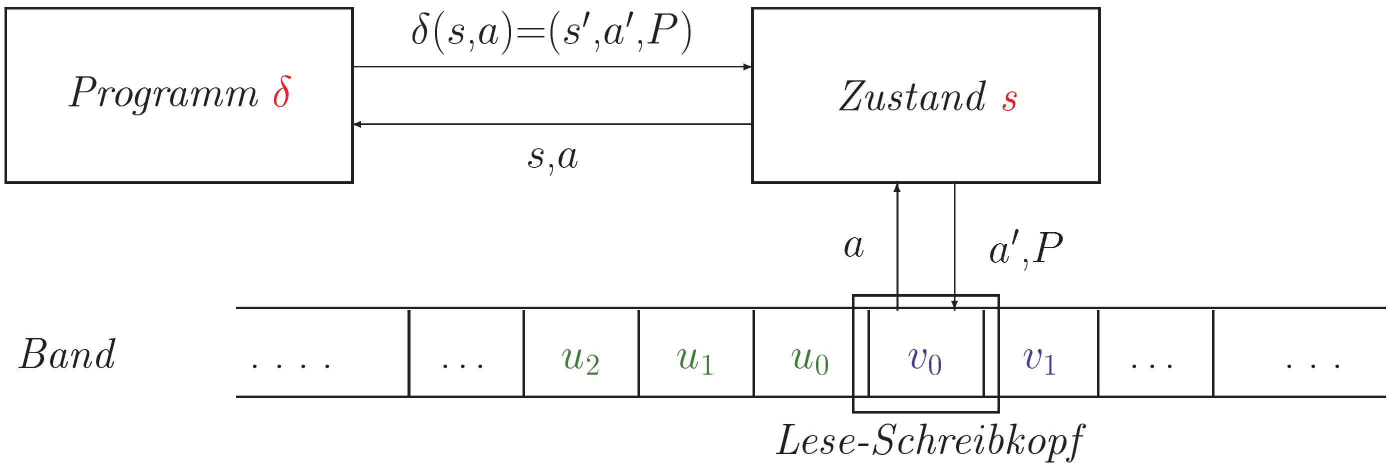
- **Zeige: Jede Funktion h_τ ist Typ-0 berechenbar**
  - Allgemeinere Aussage: jede Turingmaschine kann simuliert werden
  - Spezieller Beweis ist dann: L semi-entscheidbar

$$\begin{aligned} &\Rightarrow \psi_L \text{ Turing-berechenbar} \\ &\Rightarrow L_{\psi_L} = \{w\#1 | w \in L\} \text{ Typ-0 berechenbar} \\ &\Rightarrow L = \{w | w \in L\} \text{ Typ-0 berechenbar} \end{aligned}$$

- **Idee: Generiere alle Konfigurationen von τ**
  - Erzeuge alle möglichen Eingabeworte und Anfangskonfigurationen
  - Codiere Konfigurationsübergänge von τ als Regeln
  - Simuliere Ausgabe durch Löschen von Nonterminalsymbolen

- **Abzuleitende Worte in simulierender Grammatik**
  - Worte der Form $w\ \#\ @\ u\ s\ v\ \$$
  - w Eingabewort, (s,u,v) Konfiguration von τ bei Verarbeitung von w
  - @, \$ Trennsymbole

Rückblick: Konfigurationen in Wort-Darstellung



- **Konfigurationstripel (s,u,v)**
  - s aktueller Zustand, u, v String links/rechts vom Kopf
- **Konfigurationsübergänge**
  - $\delta(s,a) = (s',a',l)$ liefert $\hat{\delta}(s, u_0u, av) = (s', u, u_0a'v)$
  - $\delta(s,a) = (s',a',r)$ liefert $\hat{\delta}(s, u, av) = (s', a'u, v)$

Regeln der Grammatik

- **Erzeugung von Anfangskonfigurationen**
  - Regeln zur Erzeugung von Worten der Form $w\#@s_0w\$$ für $w \in X^*$
- **Simulation der Konfigurationsübergänge**
  - Regeln der Form $s\,a\,v_1 \mapsto a'\,s'\,v_1$ für $v_1 \in \Gamma$ und $\delta(s,a) = (s',a',r)$
  - Regeln der Form $s\,a\,\$ \mapsto a'\,s'\,b\,\$$ für $\delta(s,a) = (s',a',r)$
  - Regeln der Form $u_0\,s\,a \mapsto s'\,u_0\,a'$ für $u_0 \in \Gamma$ und $\delta(s,a) = (s',a',l)$
  - Regeln der Form $@\,s\,a \mapsto @\,s'\,b\,a'$ für $\delta(s,a) = (s',a',l)$
  - Regeln der Form $s\,a \mapsto q_f\,a'$ für $\delta(s,a) = (s',a',h)$
- **Schlußregeln (Ausgabe)**
  - Regeln zum Löschen der Symbole @, \$ im Kontext q_f
  - Regeln müssen q_f nach links und dann nach rechts schieben
- **Grammatik erzeugt die Sprache** $\{w\#v | h_\tau(w) = v\}$
  - Details z.B. in Erk-Priese, Seite 199–201

Grundlage funktionaler Programmiersprachen

- **Einfacher mathematischer Mechanismus**
  - Funktionen werden definiert und angewandt
  - Die Beschreibung des Funktionsverhaltens ist der Name der Funktion
  - Funktionswerte werden ausgerechnet durch Einsetzen von Werten

- **Leicht zu verstehen**
  - Definition einer Funktion: `f` $\hat{=}$ λ`x. 2*x+3` — λ-Notation
  - Auswertung der Funktion: $(\lambda$`x. 2*x+3)(4)` $\xrightarrow{\beta}$ 11 — Applikation

  Name der Funktion ist irrelevant — + Reduktion

λ-Kalkül – Syntax und Semantik

- **λ-Terme**
  - Variablen x
  - $\lambda x\,.t$, wobei x Variable und t λ-Term — λ-Abstraktion
    Vorkommen von x in t werden gebunden
  - $f\ t$, wobei t und f λ-Terme — Applikation
  - (t), wobei t λ-Term

- **Konventionen**
  - Applikation bindet stärker als λ-Abstraktion
  - Applikation ist links-assoziativ: $f\ t_1\ t_2 \hat{=} (f\ t_1)\ t_2$
  - Notation $f(t_1, \ldots, t_n)$ entspricht iterierter Applikation $f\ t_1\ \ldots t_n$

- **Auswertung von λ-Termen**
  - Ersetze Funktionsparameter durch Funktionsargumente
  - Reduktion $(\lambda x\,.t)(b) \xrightarrow{\beta} t[b/x]$
  - Substitution $t[b/x]$: ersetze freie Vorkommen von x in t durch b

Substitution und Reduktion am Beispiel

$$
\begin{array}{rl}
& \texttt{(}\lambda\texttt{n.}\lambda\texttt{f.}\lambda\texttt{x. n f (f x)) (}\lambda\texttt{f.}\lambda\texttt{x.x)} \\
\longrightarrow & \lambda\texttt{f.}\lambda\texttt{x. (}\lambda\texttt{f.}\lambda\texttt{x.x) f (f x)} \\
\longrightarrow & \lambda\texttt{f.}\lambda\texttt{x.(}\lambda\texttt{x.x) (f x)} \\
\longrightarrow & \lambda\texttt{f.}\lambda\texttt{x. f x}
\end{array}
$$

$$\Downarrow$$

$$\texttt{(}\lambda\texttt{n.}\lambda\texttt{f.}\lambda\texttt{x. n f (f x)) (}\lambda\texttt{f.}\lambda\texttt{x.x)} \xrightarrow{3} \lambda\texttt{f.}\lambda\texttt{x. f x}$$

Darstellung Boolescher Operatoren im λ-Kalkül

$$\begin{aligned}
\mathsf{T} &\equiv \lambda \mathtt{x}.\lambda \mathtt{y}.\mathtt{x}\\
\mathsf{F} &\equiv \lambda \mathtt{x}.\lambda \mathtt{y}.\mathtt{y}\\
\mathsf{if}\ b\ \mathsf{then}\ s\ \mathsf{else}\ t &\equiv b\,s\,t
\end{aligned}$$

Konditional ist invers zu T und F

$$\begin{aligned}
& \mathsf{if}\ \mathsf{T}\ \mathsf{then}\ s\ \mathsf{else}\ t\\
\equiv\ & \mathsf{T}\,s\,t\\
\equiv\ & (\lambda \mathtt{x}.\lambda \mathtt{y}.\mathtt{x})\,s\,t\\
\longrightarrow\ & (\lambda \mathtt{y}.s)\,t\\
\longrightarrow\ & s
\end{aligned}$$

$$\begin{aligned}
& \mathsf{if}\ \mathsf{F}\ \mathsf{then}\ s\ \mathsf{else}\ t\\
\equiv\ & \mathsf{F}\,s\,t\\
\equiv\ & (\lambda \mathtt{x}.\lambda \mathtt{y}.\mathtt{y})\,s\,t\\
\longrightarrow\ & (\lambda \mathtt{y}.\mathtt{y})\,t\\
\longrightarrow\ & t
\end{aligned}$$

Bildung und Analyse von Paaren

$$
\begin{aligned}
\langle s, t\rangle &\equiv \lambda \mathtt{p}.\ \mathtt{p}\, s\, t\\
pair\mathbf{.1} &\equiv pair\ (\lambda \mathtt{x}.\lambda \mathtt{y}.\mathtt{x})\\
pair\mathbf{.2} &\equiv pair\ (\lambda \mathtt{x}.\lambda \mathtt{y}.\mathtt{y})\\
\mathbf{let}\ \langle x, y\rangle = pair\ \mathbf{in}\ t &\equiv pair\ (\lambda x.\lambda y.t)
\end{aligned}
$$

Analyseoperator ist invers zur Paarbildung

$$
\begin{aligned}
&\mathbf{let}\ \langle x, y\rangle = \langle u, v\rangle\ \mathbf{in}\ t\\
\equiv\ &\langle u, v\rangle(\lambda x.\lambda y.t)\\
\equiv\ &(\lambda \mathtt{p}.\ \mathtt{p}\, u\, v)(\lambda x.\lambda y.t)\\
\longrightarrow\ &(\lambda x.\lambda y.t)\, u\, v\\
\longrightarrow\ &(\lambda y.t[u/x])\, u\, v\\
\longrightarrow\ &t[u, v/x, y])
\end{aligned}
$$

λ-berechenbare Funktionen auf $\mathbb{N}$

- **Darstellung von Zahlen durch iterierte Terme**
  - Semantisch: wiederholte Anwendung von Funktionen
  - Repräsentiere die Zahl n durch den Term $\underbrace{\lambda\mathtt{f}.\lambda\mathtt{x}.\,\mathtt{f}\,(\mathtt{f}\,..(\mathtt{f}\,\mathtt{t})\,..)}_{n\text{-}mal}$
  - Notation: $\overline{\boldsymbol{n}} \quad \equiv \quad \lambda\mathtt{f}.\lambda\mathtt{x}.\,\mathtt{f}^n\,\mathtt{x}$
  - Bezeichnung: **Church Numerals**
- **f:$\mathbb{N}^n \to \mathbb{N}$ λ-berechenbar:**
  - Es gibt einen λ-Term t mit $f(x_1, ..., x_n) = m \quad \Leftrightarrow \quad t\ \overline{x_1}\ \ldots\overline{x_n} = \overline{m}$
- **Operationen müssen Termvielfachheit verändern**
  - z.B. **add** $\overline{\boldsymbol{m}}$ $\overline{\boldsymbol{n}}$ muß als Wert immer den Term $\overline{\boldsymbol{m+n}}$ ergeben

Programmierung im λ-Kalkül

- **Nachfolgerfunktion:** $\mathsf{s} \equiv \lambda\mathtt{n}.\lambda\mathtt{f}.\lambda\mathtt{x}.\ \mathtt{n\ f\ (f\ x)}$
  – Zeige: Der Wert von $\mathsf{s}\ \overline{n}$ ist der Term $\overline{n{+}1}$

$$
\begin{aligned}
\mathsf{s}\ \overline{n} \quad &\equiv \quad (\lambda\mathtt{n}.\lambda\mathtt{f}.\lambda\mathtt{x}.\ \mathtt{n\ f\ (f\ x)})\ (\lambda\mathtt{f}.\lambda\mathtt{x}.\ \mathtt{f}^n\,\mathtt{x})\\
&\longrightarrow \lambda\mathtt{f}.\lambda\mathtt{x}.\ (\lambda\mathtt{f}.\lambda\mathtt{x}.\ \mathtt{f}^n\,\mathtt{x})\ \mathtt{f\ (f\ x)}\\
&\longrightarrow \lambda\mathtt{f}.\lambda\mathtt{x}.\ (\lambda\mathtt{x}.\ \mathtt{f}^n\,\mathtt{x})\ \mathtt{(f\ x)}\\
&\longrightarrow \lambda\mathtt{f}.\lambda\mathtt{x}.\ \mathtt{f}^n\ \mathtt{(f\ x)}\\
&\longrightarrow \lambda\mathtt{f}.\lambda\mathtt{x}.\ \mathtt{f}^{n+1}\ \mathtt{x} \qquad\qquad \equiv \quad \overline{n{+}1}
\end{aligned}
$$

- **Addition:** $\mathsf{add} \equiv \lambda\mathtt{m}.\lambda\mathtt{n}.\lambda\mathtt{f}.\lambda\mathtt{x}.\ \mathtt{m\ f\ (n\ f\ x)}$

- **Multiplikation:** $\mathsf{mul} \equiv \lambda\mathtt{m}.\lambda\mathtt{n}.\lambda\mathtt{f}.\lambda\mathtt{x}.\ \mathtt{m\ (n\ f)\ x}$

- **Test auf Null:** $\mathsf{zero} \equiv \lambda\mathtt{n}.\ \mathtt{n}\ (\lambda\mathtt{n}.\mathsf{F})\ \mathsf{T}$

- **Vorgängerfunktion:**

$$\mathsf{p} \equiv \lambda\mathtt{n}.\ (\mathtt{n}\ (\lambda\mathtt{fx}.\ \langle\mathsf{s}, \mathbf{let}\ \langle\mathtt{f},\mathtt{x}\rangle = \mathtt{fx}\ \mathbf{in}\ \mathtt{f\ x}\rangle)\ \langle\lambda\mathtt{z}.\overline{0}, \overline{0}\rangle).2$$

Auswertung der Additionsfunktion

- **Zeige:** $\mathbf{add}\ \overline{m}\ \overline{n}$ **reduziert zu** $\overline{m{+}n}$

$$
\begin{array}{lcl}
\mathbf{add}\ \overline{m}\ \overline{n} & \equiv & (\lambda\mathtt{m}.\lambda\mathtt{n}.\lambda\mathtt{f}.\lambda\mathtt{x}.\ \mathtt{m\ f\ (n\ f\ x)})\ \overline{m}\ \overline{n} \\
& \longrightarrow & (\lambda\mathtt{n}.\lambda\mathtt{f}.\lambda\mathtt{x}.\ \overline{m}\ \mathtt{f\ (n\ f\ x)})\ \overline{n} \\
& \longrightarrow & \lambda\mathtt{f}.\lambda\mathtt{x}.\ \overline{m}\ \mathtt{f}\ (\overline{n}\ \mathtt{f\ x}) \\
& \equiv & \lambda\mathtt{f}.\lambda\mathtt{x}.\ (\lambda\mathtt{f}.\lambda\mathtt{x}.\ \mathtt{f}^{m}\,\mathtt{x})\ \mathtt{f}\ (\overline{n}\ \mathtt{f\ x}) \\
& \longrightarrow & \lambda\mathtt{f}.\lambda\mathtt{x}.\ (\lambda\mathtt{x}.\ \mathtt{f}^{m}\,\mathtt{x})\ (\overline{n}\ \mathtt{f\ x}) \\
& \longrightarrow & \lambda\mathtt{f}.\lambda\mathtt{x}.\ \mathtt{f}^{m}\,(\overline{n}\ \mathtt{f\ x}) \\
& \equiv & \lambda\mathtt{f}.\lambda\mathtt{x}.\ \mathtt{f}^{m}\,((\lambda\mathtt{f}.\lambda\mathtt{x}.\ \mathtt{f}^{n}\,\mathtt{x})\ \mathtt{f\ x}) \\
& \longrightarrow & \lambda\mathtt{f}.\lambda\mathtt{x}.\ \mathtt{f}^{m}\,((\lambda\mathtt{x}.\ \mathtt{f}^{n}\,\mathtt{x})\ \mathtt{x}) \\
& \longrightarrow & \lambda\mathtt{f}.\lambda\mathtt{x}.\ \mathtt{f}^{m}\,(\mathtt{f}^{n}\,\mathtt{x}) \\
& \longrightarrow & \lambda\mathtt{f}.\lambda\mathtt{x}.\ \mathtt{f}^{m+n}\,\mathtt{x} \qquad \equiv \qquad \overline{m{+}n}
\end{array}
$$

Rekursion im λ-Kalkül

Y-Kombinator: $\mathbf{Y} \equiv \lambda\mathtt{f}.\,(\lambda\mathtt{x}.\,\mathtt{f}\,(\mathtt{x}\,\mathtt{x}))\,(\lambda\mathtt{x}.\,\mathtt{f}\,(\mathtt{x}\,\mathtt{x}))$

- **Y ist Fixpunktkombinator**
  - $\mathbf{Y}\ t = t\ (\mathbf{Y}\ t)$ für beliebige Terme t

$$
\begin{array}{lcl}
\mathbf{Y}\ t & \equiv & \lambda\mathtt{f}.\,(\lambda\mathtt{x}.\,\mathtt{f}\ (\mathtt{x}\ \mathtt{x}))\,(\lambda\mathtt{x}.\,\mathtt{f}\ (\mathtt{x}\ \mathtt{x}))\ t \\
 & \longrightarrow & (\lambda\mathtt{x}.\,t\ (\mathtt{x}\ \mathtt{x}))\ (\lambda\mathtt{x}.\,t\ (\mathtt{x}\ \mathtt{x})) \\
 & \longrightarrow & t\ (\ (\lambda\mathtt{x}.\,t\ (\mathtt{x}\ \mathtt{x}))\ (\lambda\mathtt{x}.\,t\ (\mathtt{x}\ \mathtt{x}))\ ) \\
t\ (\mathbf{Y}\ t) & \equiv & t\ (\lambda\mathtt{f}.\,(\lambda\mathtt{x}.\,\mathtt{f}\ (\mathtt{x}\ \mathtt{x}))\,(\lambda\mathtt{x}.\,\mathtt{f}\ (\mathtt{x}\ \mathtt{x}))\ t) \\
 & \longrightarrow & t\ (\ (\lambda\mathtt{x}.\,t\ (\mathtt{x}\ \mathtt{x}))\ (\lambda\mathtt{x}.\,t\ (\mathtt{x}\ \mathtt{x}))\ )
\end{array}
$$

- **Rekursion darstellbar als**

$$\textbf{letrec}\ f(x) = t \ \equiv\ \mathbf{Y}(\lambda f.\lambda x.t)$$

Ausdruckskraft des λ-Kalküls

Alle μ-rekursiven Funktione sind λ-berechenbar

- **Nachfolgerfunktion s:** $\quad \mathbf{s} \equiv \lambda n.\lambda f.\lambda x.\ n\ f\ (f\ x)$
- **Projektionsfunktionen pr_m^n** $\quad \mathbf{pr}_m^n \equiv \lambda x_1..\lambda x_n.\ x_m$
- **Konstantenfunktion c_m^n:** $\quad \mathbf{c}_m^n \equiv \lambda x_1..\lambda x_n.\ \overline{m}$
- **Komposition $f \circ (g_1 ... g_n)$:**
  - $\circ \equiv \lambda f.\lambda g_1..\lambda g_n.\ \lambda x.\ f\ (g_1 x)..(g_n x)$
- **Primitive Rekursion $Pr[f,g]$:**
  - **PR** $\equiv \lambda f.\lambda g.$
    letrec $h(x) = \lambda y.$ **if zero** y **then** $f\ x$ **else** $g\ x\ (\mathbf{p}\ y)\ (h\ x\ (\mathbf{p}\ y))$
- **Minimierung $\mu[f]$:**
  - **Mu** $\equiv \lambda f.\lambda x.$
    (**letrec** $min(y) =$ **if zero** $(f\ x\ y)$ **then** y **else** $min\ (\mathbf{s}\ y))\ \overline{0}$

Weitere Modelle

- **Nichtdeterministische Turingmaschine**
  – Arbeitsweise wie gewöhnliche Turingmaschine
  – Zustandsüberführungsfunktion erlaubt alternative Resultate

- **Abakus**
  – Erweiterung des mechanischen Abakus: beliebig viele Stangen und Kugeln
  – Zwei Operationen: Kugel hinzunehmen / Kugel wegnehmen

- **Markov-Algorithmen**
  – Wie Typ-0 Grammatiken, aber mit fester Strategie für Regelanwendung
  – Verarbeitet Eingabeworte, statt mit einem Startsymbol zu beginnen

- **Arithmetische Repräsentierbarkeit**
  – Spezifikation von Funktionen in arithmetisch-logischem Kalkül
  – f ist repräsentierbar, wenn das Ein-/Ausgabeverhalten von f eindeutig durch eine Formel spezifiziert werden kann
  – Eindeutigkeit muß ausschließlich aus logischen Axiomen beweisbar sein

Die Church'sche These

- **Alle Berechenbarkeitsmodelle sind äquivalent**
  - Keines kann mehr berechnen als Turingmaschinen
  - Es ist keine intuitiv berechenbare Funktion bekannt, die nicht von Turingmaschinen berechnet werden kann

- **Church'sche These:**
  Die Klasse der Turing-berechenbaren Funktionen stimmt mit der Klasse der intuitiv berechenbaren Funktionen überein
  - Unbeweisbare, aber wahrscheinlich richtige Behauptung
  - Arbeitshypothese für theoretische Argumente
    - man darf in Beweisen "intuitive" Programme angeben