Theoretische Informatik II

Einheit 5.3

Funktionale & Logische Programme

1. Der λ-Kalkül
2. Arithmetische Repräsentierbarkeit
3. Die Churchsche These

Grundlage funktionaler Programmiersprachen

- **Einfacher mathematischer Mechanismus**
  - Funktionen werden definiert und angewandt
  - Beschreibung des Funktionsverhaltens wird zum Namen der Funktion
  - Funktionswerte werden ausgerechnet durch Einsetzen von Werten

- **Leicht zu verstehende Basiskonzepte**
  1. Definition einer Funktion:

     $\texttt{f} \mathrel{\hat{=}} \lambda\texttt{x. 2*x+3}$ **λ-Abstraktion**

     Name der Funktion irrelevant für Beschreibung des Verhaltens
  2. Anwendung der Funktion:

     $\texttt{f(4)} \mathrel{\hat{=}} (\lambda\texttt{x. 2*x+3)(4)}$ **Applikation**
  3. Auswertung der Funktion:

     $(\lambda\texttt{x. 2*x+3)(4)} \xrightarrow{\beta} \texttt{2*4+3} \xrightarrow{*} \texttt{11}$ **Reduktion**

λ-Kalkül – Syntax

- **Einfache Programmiersprache: 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
- **Prioritäten und Kurzschreibweisen**
  - Applikation bindet stärker als λ-Abstraktion
  - Applikation ist links-assoziativ: $f\ t_1\ t_2 \mathrel{\hat{=}} (f\ t_1)\ t_2$
  - Notation $f(t_1,..,t_n)$ steht für iterierte Applikation $f\ t_1\ ..\ t_n$
- **Beispiele für λ-Terme**
  - `x` Symbole sind immer Variablen
  - `λf.λx.f(x)`
  - `λf.λg.λx. f g (g x)` Funktionen höherer Ordnung
  - `x(x)` Selbstanwendung

λ-Kalkül – Berechnung durch Auswertung

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

- **Substitution und Reduktion am Beispiel**

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

$$\Downarrow$$

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

Substitution präzisiert

- **Vorkommen von Variablen in λ-Termen**
  - x die Variable x kommt frei vor; $y{\neq}x$ kommt nicht vor
  - $\lambda x.t$: beliebige Vorkommen von x in t werden gebunden
Vorkommen von $y{\neq}x$ in t bleiben unverändert
  - $f\ t$ freie Vorkommen von x in t_i bleiben frei
(t) gebundene Vorkommen von x bleiben gebunden

$$\overbrace{\mathtt{\lambda f.\lambda x.}\underbrace{\mathtt{(\lambda z.\ f\ x\ z)\ x}}_{\mathtt{x}\ \textit{frei}}}^{\mathtt{x}\ \textit{gebunden}}$$

- **Substitution $u[t/x]$ in λ-Termen**

$x[t/x]$	$=t$	$x[t/y]$	$=x$	$(y{\neq}x)$
$\lambda x\,.\,u[t/x]$	$=\lambda x\,.\,u$			
$\lambda x\,.\,u[t/y]$	$=\lfloor\lambda z\,.\,u[z/x]\rfloor[t/y]$ *	$\lambda x\,.\,u[t/y]$	$=\lambda x\,.\lfloor u[t/y]\rfloor$ **	
$f\ u[t/x]$	$=f[t/x]\ u[t/x]$	$(u)[t/x]$	$=\ (u[t/x])$	

*: *$y{\neq}x$, y frei in u, x frei in t, z neue Variable*

**: *$y{\neq}x$, y nicht frei in u oder x nicht frei in t*

Vom λ-Kalkül zu echten Programmen

- **λ-Kalkül ist der Basismechanismus**
  - Die Assemblersprache funktionaler Programme
  - Spezialhardware (Lisp-Maschinen) kann λ-Terme direkt auswerten
- **Programm- und Datenstrukturen werden codiert**
  - Nicht anders als in konventionellen Computern
    - Datenstrukturen werden als als Bitketten codiert
    - Programmstrukturen werden in Sprungbefehle übersetzt
- **Die wichtigsten Strukturen sind leicht codierbar**
  - Boolesche Operationen: T, F, $\mathsf{if}\ \boldsymbol{b}\ \mathsf{then}\ \boldsymbol{s}\ \mathsf{else}\ \boldsymbol{t}$
  - Tupel / Projektionen: $(\boldsymbol{s},\boldsymbol{t})$, $\boldsymbol{pair}.1$, $\boldsymbol{pair}.2$, $\mathsf{let}\ (\boldsymbol{x},\boldsymbol{y}) = \boldsymbol{pair}\ \mathsf{in}\ \boldsymbol{t}$
  - Zahlen und arithmetische Operationen
  - Iteration oder Rekursion von Funktionen

Darstellung Boolescher Operatoren im λ-Kalkül

Zwei verschiedene Objekte und ein Konditional

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

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

$$
\begin{array}{lcl}
(u,v) & \equiv & \lambda\mathtt{p}.\ \ \mathtt{p}\,u\,v \\
pair.\mathsf{1} & \equiv & pair\ \ (\lambda\mathtt{x}.\lambda\mathtt{y}.\mathtt{x}) \\
pair.\mathsf{2} & \equiv & pair\ \ (\lambda\mathtt{x}.\lambda\mathtt{y}.\mathtt{y}) \\
\mathsf{let}\ \ (x,y)=pair\ \mathsf{in}\ \ t & \equiv & pair\ \ (\lambda x.\lambda y.t)
\end{array}
$$

Analyseoperator ist invers zur Paarbildung

$$
\begin{array}{ll}
 & \mathsf{let}\ \ (x,y)=(u,v)\ \ \mathsf{in}\ \ t \\
\equiv & (u,v)(\lambda x.\lambda y.t) \\
\equiv & (\lambda\mathtt{p}.\ \ \mathtt{p}\,u\,v)(\lambda x.\lambda y.t) \\
\longrightarrow & (\lambda x.\lambda y.t)\ u\,v \\
\longrightarrow & (\lambda y.t[u/x])\ v \\
\longrightarrow & t[u,v/x,y]
\end{array}
$$

Operationen auf natürlichen Zahlen

- **Darstellung von Zahlen durch iterierte Terme**
  - Semantisch: wiederholte Anwendung von Funktionen
  - Repräsentiere die Zahl n durch den Term $\lambda\mathtt{f}.\lambda\mathtt{x}.\underbrace{\mathtt{f}\,(\mathtt{f}\,..(\mathtt{f}\,\mathtt{x})\,..)}_{n\text{-}mal}$
  - Notation: $\overline{\boldsymbol{n}} \;\equiv\; \lambda\mathtt{f}.\lambda\mathtt{x}.\,\mathtt{f}^n\,\mathtt{x}$
  - Bezeichnung: **Church Numerals**
- $\boldsymbol{f}$**:**$\mathbb{N}^{\boldsymbol{n}}\longrightarrow\mathbb{N}$ **λ-berechenbar:**
  - Es gibt einen λ-Term t mit der Eigenschaft

$$\boldsymbol{f(x_1,..,x_n) = m} \;\Leftrightarrow\; \boldsymbol{t}\;\overline{\boldsymbol{x_1}}\,..\,\overline{\boldsymbol{x_n}} = \overline{\boldsymbol{m}}$$

- **Operationen müssen Termvielfachheit berechnen**
  - 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 $\mathsf{s}\ \overline{\boldsymbol{n}}$ ist der Term $\overline{\boldsymbol{n}\mathsf{+1}}$

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

- **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:**

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

Korrektheit des Programms für die Addition

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

$$
\begin{array}{lll}
\text{add } \overline{m}\ \overline{n} & \equiv & (\lambda\texttt{m}.\lambda\texttt{n}.\lambda\texttt{f}.\lambda\texttt{x}.\ \texttt{m f (n f x))}\ \overline{m}\ \overline{n} \\
& \longrightarrow & (\lambda\texttt{n}.\lambda\texttt{f}.\lambda\texttt{x}.\ \overline{m}\ \texttt{f (n f x))}\ \overline{n} \\
& \longrightarrow & \lambda\texttt{f}.\lambda\texttt{x}.\ \overline{m}\ \texttt{f}\ (\overline{n}\ \texttt{f x)} \\
& \equiv & \lambda\texttt{f}.\lambda\texttt{x}.\ (\lambda\texttt{f}.\lambda\texttt{x}.\ \texttt{f}^m\,\texttt{x)}\ \texttt{f}\ (\overline{n}\ \texttt{f x)} \\
& \longrightarrow & \lambda\texttt{f}.\lambda\texttt{x}.\ (\lambda\texttt{x}.\ \texttt{f}^m\,\texttt{x)}\ (\overline{n}\ \texttt{f x)} \\
& \longrightarrow & \lambda\texttt{f}.\lambda\texttt{x}.\ \texttt{f}^m\,(\overline{n}\ \texttt{f x)} \\
& \equiv & \lambda\texttt{f}.\lambda\texttt{x}.\ \texttt{f}^m\,\texttt{((}\lambda\texttt{f}.\lambda\texttt{x}.\ \texttt{f}^n\,\texttt{x) f x)} \\
& \longrightarrow & \lambda\texttt{f}.\lambda\texttt{x}.\ \texttt{f}^m\,\texttt{((}\lambda\texttt{x}.\ \texttt{f}^n\,\texttt{x) x)} \\
& \longrightarrow & \lambda\texttt{f}.\lambda\texttt{x}.\ \texttt{f}^m\,\texttt{(f}^n\,\texttt{x)} \\
& \longrightarrow & \lambda\texttt{f}.\lambda\texttt{x}.\ \texttt{f}^{m+n}\,\texttt{x} \qquad\qquad \equiv \quad \overline{m{+}n}
\end{array}
$$

Anwendung sogenannter Fixpunktkombinatoren

- **Fixpunktkombinator**
  - λ-Term R mit der Eigenschaft: $R\;t = t\;(R\;t)$ für beliebige Terme t
  - Definiert man eine Funktion f durch $F \equiv R\;(\lambda f.\lambda x.t[f,x])$,
    wobei im Programmkörper t sowohl f als auch x vorkommen können
    dann gilt $\boldsymbol{F\;x} = (\lambda f.\lambda x.t[f,x])\;F\;x \xrightarrow{*} \boldsymbol{t[F,x]}$
  - Kurzschreibweise: $F \equiv$ letrec $f(x) = t$ $\;(\equiv R(\lambda f.\lambda x.t))$

- **Y-Kombinator:** $\mathsf{Y} \equiv \lambda \mathtt{f}.\,(\lambda \mathtt{x}.\,\mathtt{f}\,(\mathtt{x\,x}))\,(\lambda \mathtt{x}.\,\mathtt{f}\,(\mathtt{x\,x}))$
  - Bekanntester Fixpunktkombinator

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

Der λ-Kalkül ist Turing-mächtig

Alle μ-rekursiven Funktionen sind λ-berechenbar

- **Nachfolgerfunktion s:** $\quad \mathsf{s} \equiv \lambda \mathtt{n}.\lambda \mathtt{f}.\lambda \mathtt{x}.\ \mathtt{n\ f\ (f\ x)}$
- **Projektionsfunktionen pr^n_m:** $\quad \mathsf{pr}^n_m \equiv \lambda \mathtt{x}_1..\lambda \mathtt{x}_n.\ \mathtt{x}_m$
- **Konstantenfunktion c^n_m:** $\quad \mathsf{c}^n_m \equiv \lambda \mathtt{x}_1..\lambda \mathtt{x}_n.\ \overline{\mathtt{m}}$
- **Komposition $f \circ (g_1, .., g_n)$:**

  $\circ \equiv \lambda \mathtt{f}.\lambda \mathtt{g}_1..\lambda \mathtt{g}_n.\ \lambda \mathtt{x}.\ \mathtt{f}\ (\mathtt{g}_1 \mathtt{x})..(\mathtt{g}_n \mathtt{x})$
- **Primitive Rekursion $Pr[f, g]$:**

  $\mathsf{PR} \equiv \lambda \mathtt{f}.\lambda \mathtt{g}.$

  $\quad \mathsf{letrec}\ \mathtt{h(x)} = \lambda \mathtt{y}.\mathsf{if\ zero}\ \mathtt{y}\ \mathsf{then}\ \mathtt{f\ x}\ \mathsf{else}\ \mathtt{g\ x\ (p\ y)\ (h\ x\ (p\ y))}$
- **Minimierung $\mu[f]$:**

  $\mathsf{Mu} \equiv \lambda \mathtt{f}.\lambda \mathtt{x}.\ (\mathsf{letrec}\ \mathtt{min(y)} = \mathsf{if\ zero}(\mathtt{f\ x\ y})\ \mathsf{then}\ \mathtt{y}\ \mathsf{else}\ \mathtt{min}\ (\mathtt{s\ y}))\ \overline{0}$

Arithmetische Repräsentierbarkeit

Rechtfertigung logischer Programmiersprachen

- **Spezifikation von Funktionen in logischem Kalkül**
  - Formeln repräsentieren Ein-/Ausgabeverhalten von Funktionen
  - Repräsentation muß eindeutig sein (nur eine Ausgabe pro Eingabe)
  - Eindeutigkeit muß ausschließlich aus logischen Axiomen beweisbar sein
- **Zentraler Begriff: Gültigkeit in einer Theorie**
  - Logische **Theorie T** gegeben durch formale Sprache und Axiome
  - Formel F ist **gültig in T**, wenn sie logisch aus den Axiomen folgt
  - Kurzschreibweise $\models_T F$
- **Berechenbarkeit: $f : \mathbb{N}^k \rightarrow \mathbb{N}$ repräsentierbar in T**
  - $f(i_1,..,i_k)=j$ g.d.w. $\models_T F(\overline{i_1},..,\overline{i_k},\overline{j})$ für eine Formel F
    In der Theorie T ist beweisbar, ob f einen bestimmten Wert annimmt
  - $\overline{n}$ ist ein Term der formalen Sprache, der die Zahl n codiert

Die arithmetische Theorie $\mathcal{Q}$

- **Formale Sprache**
  - Sprache der Prädikatenlogik (mit Gleichheit)
  - Konstantensymbol $\overline{0}$
  - Einstelliges Funktionssymbol s
  - Zweistellige Funktionssymbole $+$ und $*$
- **Semantik: Logik + 7 Axiome** (ohne Induktion!)

  Q1: $\forall x, y.\ \mathsf{s}(x) = \mathsf{s}(y) \Rightarrow x = y$

  Q2: $\forall x.\ \mathsf{s}(x) \neq \overline{0}$

  Q3: $\forall x.\ x \neq \overline{0} \Rightarrow \exists y. x = \mathsf{s}(y)$

  Q4: $\forall x.\ x + \overline{0} = x$

  Q5: $\forall x, y.\ x + \mathsf{s}(y) = \mathsf{s}(x + y)$

  Q6: $\forall x.\ x * \overline{0} = \overline{0}$

  Q7: $\forall x, y. x * \mathsf{s}(y) = (x * y) + x$
- **Axiome gelten auch für Nichtstandardzahlen**
  - Es sind auch andere Interpretationen der Symbole s, $+$, $*$ möglich

    Definiere Operationen s, $+$, $*$ auf $\mathbb{N} \cup \{\infty, \infty'\}$

    Kommutativität, Assoziativität müssen auf $\mathbb{N} \cup \{\infty, \infty'\}$ nicht gelten

  Dennoch kann man alle berechenbaren Funktionen in $\mathcal{Q}$ repräsentieren

Repräsentierbarkeit in $\mathcal{Q}$

- **$f : \mathbb{N}^k \longrightarrow \mathbb{N}$ repräsentierbar in $\mathcal{Q}$**
  – Es gibt eine k+1-stellige Formel F, so daß für alle $i_1, .., i_k, j \in \mathbb{N}$ gilt:

  $$f(i_1, .., i_k) = j \text{ genau dann, wenn } \models_{\mathcal{Q}} F(\overline{i_1}, .., \overline{i_k}, \overline{j}) \qquad (\overline{n} \equiv \underbrace{\mathsf{s}(\ldots(\mathsf{s}(\overline{0}))\ldots)}_{n-mal})$$

- **Beispiele repräsentierbarer Funktionen**

  Addition: Bestimme 3-stellige Formel `ADD` mit $i+j=k$ gdw. $\models_{\mathcal{Q}} \mathtt{ADD}(\overline{i},\overline{j},\overline{k})$

  Einfach, da $+$ Teil der Sprache ist: $\mathtt{ADD}(x_1, x_2, y) \equiv y\mathtt{=}x_1\mathtt{+}x_2$

  Multiplikation $\mathtt{MUL}(x_1, x_2, y) \equiv y\mathtt{=}x_1\mathtt{*}x_2$

  Vergleich $\leq$ *(Hilfsprädikat für Funktionsbeschreibungen)* $\mathtt{LE}(x, y) \equiv \exists\mathtt{z}.\, x\mathtt{+z=}y$

  $<$ $\mathtt{LT}(x, y) \equiv \mathtt{LE}(\mathsf{s}(x), y)$

  Subtraktion $\mathtt{SUB}(x_1, x_2, y) \equiv x_1\mathtt{=}x_2\mathtt{+}y \;\vee\; (\mathtt{LE}(x_1, x_2) \wedge y\mathtt{=}\overline{0})$

  Division $\mathtt{DIV}(x_1, x_2, y) \equiv \exists\mathtt{z}.\, \mathtt{LT}(\mathtt{z}, x_2) \wedge x_2\mathtt{*}y\mathtt{+z} = x_1$

  Divisionsrest/Modulo $\mathtt{MOD}(x_1, x_2, y) \equiv \mathtt{LT}(y, x_2) \;\wedge\; \exists\mathtt{z}.\, x_2\mathtt{*z+}y = x_1$

- **Repräsentierbarkeit in $\mathcal{Q}$ ist Turing-mächtig**
  – Alle μ-rekursiven Funktionen sind repräsentierbar in $\mathcal{Q}$

Weitere Modelle für Berechenbarkeit

- **Abakus**
  - Erweiterung des mechanischen Abakus: beliebig viele Stangen und Kugeln
  - Zwei Operationen: Kugel hinzunehmen / Kugel wegnehmen
- **Registermaschinen**
  - Direkter Zugriff auf endliche Zahl von Registern
  - Register enthalten (unbegrenzte) natürliche Zahlen
  - Befehle entsprechen elementarem Assembler
- **Mini-PASCAL**
  - Basisversion einer imperativen höheren Programmiersprache
  - Arithmetische Operationen, Fallunterscheidung, Schleifen
  - Operationale Semantik erklärt Bedeutung der Befehle
- **Markov-Algorithmen**
  - Wie Typ-0 Grammatiken, aber mit fester Strategie für Regelanwendung
  - Verarbeitet Eingabeworte, statt mit einem Startsymbol zu beginnen

Alle Modelle sind ebenfalls Turing-mächtig

Die Churchsche These

- **Alle Berechenbarkeitsmodelle sind äquivalent**
  - Keines kann mehr berechnen als Turingmaschinen
  - Es ist keine Funktion bekannt, die man intuitiv als berechenbar ansehen würde, aber nicht mit einer Turingmaschine berechnen kann
- **These von A. Church: Die Klasse der Turing-berechenbaren Funktionen ist identisch mit der Klasse der intuitiv berechenbaren Funktionen**
  - **Unbeweisbar**e, aber wahrscheinlich richtige Behauptung
  - **Arbeitshypothese** für theoretische Argumente
    erlaubt, intuitiv formulierte Programme in Beweisen zu verwenden

Berechenbarkeitsmodelle im Rückblick

- **Es gibt viele äquivalente Modelle**
  - Maschinen-basierte Modelle: Turingmaschinen, Registermaschinen, ...
  - Programmiersprachen-basierte Modelle: Mini-PASCAL, Mini-Java, ...
  - Abstrakte mathematische Beschreibung: rekursive Funktionen
  - Funktionale Programmierung: λ-Kalkül
  - Logische Programmierung: Arithmetische Repräsentierbarkeit

- **Alle Berechenbarkeitsmodelle sind i.w. äquivalent**
  - These: Alle berechenbaren Funktionen sind Turing-berechenbar (oder rekursiv, λ-berechenbar, arithmetisch repräsentierbar, ...)
  - Die Theorie des Berechenbaren hängt nicht vom konkreten Modell ab, sondern basiert auf allgemeinen Eigenschaften, die alle Modelle (implizit) gemeinsam haben

ANHANG

Repräsentierbarkeit in $\mathcal{Q}$

- **Definiere Numeral $\overline{n}$ $\equiv$ $\underbrace{\mathsf{s}(\ldots(\mathsf{s}(\overline{0}))\ldots)}_{n\text{–}mal}$**
  - Codierung der Zahl n als Term der formalen Sprache von $\mathcal{Q}$
- **Definiere: $f : \mathbb{N}^k \to \mathbb{N}$ repräsentierbar in $\mathcal{Q}$**

  Es gibt eine $k+1$-stellige Formel F, so daß für alle $(i_1, .., i_k) \in \mathbb{N}^k$, $j \in \mathbb{N}$ gilt:

  $$f(i_1, .., i_k) = j \quad \text{impliziert} \quad \models_{\mathcal{Q}} F(\overline{i_1}, .., \overline{i_k}, \overline{j})$$

  $$f(i_1, .., i_k) \neq j \quad \text{impliziert} \quad \models_{\mathcal{Q}} \neg F(\overline{i_1}, .., \overline{i_k}, \overline{j})$$

  In $\mathcal{Q}$ ist beweisbar, ob f bei Eingabe $i_1, .., i_k$ einen Wert j annimmt
- **Konstruktion repräsentierbarer Funktionen**
  - Angabe einer Formel F, die das Ein-/Ausgabeverhalten von f beschreibt
  - Nachweis, daß F (nur) für gültige Ein-/Ausgabepaare in $\mathcal{Q}$ beweisbar ist

Beispiele repräsentierbarer Funktionen

- **Addition**
  - Bestimme eine 3-stellige Formel ADD mit $i{+}j{=}k$ gdw. $\models_{\mathcal{Q}}$ $\mathtt{ADD}(\overline{i},\overline{j},\overline{k})$
  - Einfach, da Addition vordefiniert: $\mathtt{ADD}(x_1, x_2, y) \;\equiv\; y\texttt{=}x_1\texttt{+}x_2$
- **Korrektheit der Repräsentation**
  - Zeige: für alle $i, j, k \in \mathbb{N}$ mit $i{+}j{=}k$ gilt $\models_{\mathcal{Q}}$ $\overline{k}\texttt{=}\overline{i}\texttt{+}\overline{j}$ $\quad(\hat{=}\ \mathtt{ADD}(\overline{i},\overline{j},\overline{k}))$

    und für alle $i, j, k \in \mathbb{N}$ mit $i{+}j{\neq}k$ gilt $\models_{\mathcal{Q}}$ $\overline{k}{\neq}\overline{i}\texttt{+}\overline{j}$

  Sei i beliebig, aber fest. Wir führen den Beweis durch Induktion über j:
  - Für $j = 0$ folgt $i{=}k$, also $\overline{i}\texttt{=}\overline{k}$ und über Axiom Q_4: $\models_{\mathcal{Q}}$ $\overline{i}\texttt{+}\overline{0}\,\texttt{=}\,\overline{i}$
  - Es gelte $\models_{\mathcal{Q}}$ $\overline{n}\texttt{=}\overline{i}\texttt{+}\overline{j}$ für alle n und $j{=}m \in \mathbb{N}$ mit $i{+}j{=}n$
  - Es sei $j{=}m{+}1$, also $\overline{j}{=}\mathsf{s}(\overline{m})$ und es gelte $i{+}j{=}k$.

    Dann gilt $k = i{+}m{+}1 = n{+}1$ und $\overline{k}{=}\mathsf{s}(\overline{n})$.

    Mit Axiom Q_5 folgt $\models_{\mathcal{Q}}$ $\overline{i}\texttt{+}\overline{j} \;\texttt{=}\; \overline{i}\texttt{+}\mathtt{s}(\overline{m}) \;\texttt{=}\; \mathtt{s}(\overline{i}\texttt{+}\overline{m}) \;\texttt{=}\; \mathtt{s}(\overline{n}) \;\texttt{=}\; \overline{k}$

  Der Beweis für $i{+}j{\neq}k$ impliziert $\models_{\mathcal{Q}}$ $\overline{k}{\neq}\overline{i}\texttt{+}\overline{j}$ ist analog

Beispiele repräsentierbarer Funktionen (II)

- **Multiplikation** $\texttt{MUL}(x_1, x_2, y) \;\equiv\; y\texttt{=}x_1\texttt{*}x_2$
- **Vergleich $\leq$** (Prädikat) $\texttt{LE}(x, y) \;\equiv\; \exists\texttt{z}.x\texttt{+z=}y$
  $<$ $\texttt{LT}(x, y) \;\equiv\; \texttt{LE}(\texttt{s}(x), y)$
- **Subtraktion** $\texttt{SUB}(x_1, x_2, y) \;\equiv\; x_1\texttt{=}x_2\texttt{+}y \;\vee\; (\texttt{LE}(x_1, x_2) \wedge y\texttt{=}\overline{\texttt{0}})$
- **Division** $\texttt{DIV}(x_1, x_2, y) \;\equiv\; \exists\texttt{z}.\,\texttt{LT}(\texttt{z},x_2) \wedge x_2\texttt{*}y\texttt{+z}=x_1$
- **Divisionsrest/Modulo**
  $\texttt{MOD}(x_1, x_2, y) \;\equiv\; \texttt{LT}(y,x_2) \;\wedge\; \exists\texttt{z}.\,x_2\texttt{*z+}y=x_1$
- **Teilbarkeit** (Prädikat) $\texttt{DIVIDES}(x_1, x_2) \;\equiv\; \exists\texttt{z}.\,x_1\texttt{*z=}x_2$
- **Primzahleigenschaft** (Prädikat)
  $\texttt{PRIME}(x) \;\equiv\; \forall\texttt{y}.\,(\texttt{LT}(\texttt{y},x) \wedge \texttt{LT}(\overline{\texttt{1}},\texttt{y})) \;\Rightarrow\; \neg\texttt{DIVIDES}(\texttt{y},x)$

Mehr Beispiele in den Übungen

Repräsentierbarkeit in $\mathcal{Q}$ ist Turing-mächtig (I)

Definiere Min-rekursive Funktionen

Definition rekursiver Funktionen ohne primitive Rekursion

- $\boldsymbol{\mathcal{R}_{min}}$: **Menge der min-rekursiven Funktionen**
  – Addition, Nachfolger, Projektions- oder Konstantenfunktion sowie
  – Alle Funktionen, die aus min-rekursiven Funktionen durch Komposition oder Minimierung entstehen

  Wichtiger Sonderfall für Vergleiche mit anderen Modellen

- $\boldsymbol{\mathcal{R}_{min} \subseteq \mathcal{R}}$: min-rekursive Funktionen sind μ-rekursiv
  – Offensichtlich, da Additition μ-rekursiv ist

- $\boldsymbol{\mathcal{R} \subseteq \mathcal{R}_{min}}$: μ-rekursive Funktionen sind min-rekursiv
  – Beschreibe Abarbeitung des Stacks einer primitiven Rekursion
  – Suche nach erstem erzeugten Stack der Länge 1 (Details aufwendig)

Repräsentierbarkeit in $\mathcal{Q}$ ist Turing-mächtig (II)

Alle min-rekursiven Funktionen sind repräsentierbar

- **Nachfolgerfunktion s:** $\qquad \mathtt{S}(x,y) \;\equiv\; y\mathtt{=s(}x\mathtt{)}$

- **Projektionsfunktionen pr_m^n:** $\qquad \mathtt{PR}_m^n(x_1,..,x_n,y) \;\equiv\; y\mathtt{=}x_m$

- **Konstantenfunktion c_m^n:** $\qquad \mathtt{C}_m^n(x_1,..,x_n,y) \;\equiv\; y\mathtt{=}\overline{m}$

- **Addition add:** $\qquad \mathtt{ADD}(x_1,x_2,y) \;\equiv\; y\mathtt{=}x_1\mathtt{+}x_2$

- **Komposition $f \circ (g_1,..,g_n)$:**

$$H(\vec{x},z) \;\equiv\; \exists y_1,..,y_n.(G_1(\vec{x},y_1) \wedge .. \wedge G_n(\vec{x},y_n) \wedge F(y_1,..,y_n,z))$$

  H repräsentiert $f \circ (g_1,..,g_n)$, wenn $F, G_1, ..G_n$ Repräsentationen von $f, g_1, .., g_n$

- **Minimierung $\mu[f]$:**

$$H(\vec{x},y) \;\equiv\; \forall \mathtt{w}.\,\mathtt{LE(w,}y\mathtt{)} \Rightarrow \mathtt{[}F\mathtt{(}\vec{x}\mathtt{,}y\mathtt{,}\overline{\mathtt{0}}\mathtt{)} \Leftrightarrow \mathtt{w=}y\mathtt{]}$$

  H repräsentiert $\mu[f]$, wenn F Repräsentation von f