Theoretische Informatik II

Einheit 5.3

Funktionale & Logische Programme

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

DER λ-KALKÜL

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:

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

     Name der Funktion irrelevant für Beschreibung des Verhaltens

  2. Anwendung der Funktion (ohne Auswertung):

     $\texttt{f(4)} \;\hat{=}\; (\lambda\texttt{x.2*x+3)(4)}$ **Applikation**

  3. Auswertung einer Funktionsanwendung (tatsächliches Ausrechnen):

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

λ-Kalkül – Syntax

- **Einfache Programmiersprache: $\boldsymbol{\lambda}$-Terme**
  - Variablen x
  - $\lambda x\,.\,t$, wobei x Variable und t λ-Term **$\boldsymbol{\lambda}$-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: $f\ t_1\ t_2 \hat{=} (f\ t_1)\ t_2$
  - 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)` Anwendung einer Funktion
  - `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

    Sonderfälle: $\lfloor\lambda x\,.\,u\rfloor[b/x] = \lambda x\,.\,u$ *x ist nicht frei in u*

    $\lfloor\lambda x\,.\,u\rfloor[b/y] = \lfloor\lambda z\,.\,u[z/x]\rfloor[b/y]$ *$y{\neq}x$ frei in u, x frei in b, z neu*

- **Substitution und Reduktion am Beispiel**

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

$$\Downarrow$$

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

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**
  - Berechnung auf λ-Ausdrücken muß Effekte auf Struktur simulieren
  - 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, $\mathsf{if}\ b\ \mathsf{then}\ s\ \mathsf{else}\ t$
  - Tupel / Projektionen: (s,t), $pair.1$, $pair.2$, $\mathsf{let}\ (x,y)=pair\ \mathsf{in}\ t$
  - Zahlen und arithmetische Operationen
  - Iteration oder Rekursion von Funktionen

Darstellung Boolescher Operatoren im λ-Kalkül

Zwei verschiedene Objekte und ein Test

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

Konditional(-simulation) ist invers zu T und F

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

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

Paare: Datenkapselung und Komponentenzugriff

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

Analyseoperator ist invers zur Paarbildung

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

Simulation arithmetischer Operationen

- **Darstellung natürlicher Zahlen durch iterierte Terme**
  - Semantisch: wiederholte Anwendung von Funktionen
  - Repräsentiere die Zahl n durch den Term $\lambda\mathtt{f}.\lambda\mathtt{x}.\underbrace{\mathtt{f\,(f..(f\,x)..)}}_{n\text{-}mal}$
  - Notation: $\overline{n} \;\equiv\; \lambda\mathtt{f}.\lambda\mathtt{x}.\,\mathtt{f}^n\,\mathtt{x}$
  - Bezeichnung: **Church Numerals**

- $f{:}\mathbb{N}^n{\longrightarrow}\mathbb{N}$ **λ-berechenbar**:
  - Es gibt einen λ-Term t mit der Eigenschaft

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

- **Operationen müssen Termvielfachheit berechnen**
  - Simulation einer Funktion auf Darstellung von Zahlen muß Darstellung des Funktionsergebnisses liefern
  - z.B. add $\overline{m}$ $\overline{n}$ muß als Wert immer den Term $\overline{m{+}n}$ ergeben

Programmierung im λ-Kalkül

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

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

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

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

- **Wende Fixpunktkombinator auf Funktionskörper an**
  - Fixpunktkombinator: λ-Term R mit Eigenschaft $R\,t = t\,(R\,t)$ für alle t
  - Definiert man 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

$$F\,x = (\lambda f.\lambda x.t[f,x])\;F\;x \xrightarrow{*} t[F,x]$$

  - Kurzschreibweise: $F \equiv$ letrec $f(x)=t$ $(\equiv R(\lambda f.\lambda x.t))$

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

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

Der λ-Kalkül ist Turing-mächtig

Alle μ-rekursiven Funktionen sind λ-berechenbar

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

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 ($\models_T F$), wenn F logisch aus den Axiomen folgt
- **Berechenbarkeitsbegriff: $f{:}\mathbb{N}^k{\longrightarrow}\mathbb{N}$ repräsentierbar in T**
  - Es gibt eine Formel F mit $f(i_1,..,i_k){=}j$ g.d.w. $\models_T F(\overline{i_1},..,\overline{i_k},\overline{j})$, d.h. 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!)

  Q$_1$: $\forall x,y.\ \mathsf{s}(x) = \mathsf{s}(y) \Rightarrow x = y$

  Q$_2$: $\forall x.\ \mathsf{s}(x) \neq \overline{0}$

  Q$_3$: $\forall x.\ x \neq \overline{0} \Rightarrow \exists y. x = \mathsf{s}(y)$

  Q$_4$: $\forall x.\ x + \overline{0} = x$

  Q$_5$: $\forall x,y.\ x + \mathsf{s}(y) = \mathsf{s}(x + y)$

  Q$_6$: $\forall x.\ x * \overline{0} = \overline{0}$

  Q$_7$: $\forall x,y. x * \mathsf{s}(y) = (x * y) + x$
- **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:$\mathbb{N}^k \longrightarrow \mathbb{N}$ arithmetisch repräsentierbar**
  – f ist repräsentierbar in $\mathcal{Q}$, wobei $n \in \mathbb{N}$ codiert als $\overline{n} \equiv \underbrace{\mathtt{s}(\ldots(\mathtt{s}(\overline{0}))\ldots)}_{n-mal}$
- **Beispiele arithmetisch repräsentierbarer Funktionen**

Addition: suche 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: $\mathtt{ADD}(x_1,x_2,y) \equiv y{=}x_1{+}x_2$

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

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

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

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

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

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 $\mathcal{Q}$ $\mapsto$ Anhang

Weitere Modelle für Berechenbarkeit

- **Abakus**
  - Erweiterung des mechanischen Abakus: beliebig viele Stangen / 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 / mini-JAVA**
  - 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 ansieht, aber nicht mit einer Turingmaschine berechnen kann

- **These von Alonzo 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, die es ermöglicht, in Beweisen intuitiv formulierte Programme anstelle von konkreten Turingmaschinen zu verwenden

Berechenbarkeitsmodelle im Rückblick

- **Es gibt viele äquivalente Modelle**
  - Maschinenbasierte Modelle: Turingmaschinen, Registermaschinen, ...
  - Programmiersprachenbasierte 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, $\ldots$)
  - 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 $\overline{n} \equiv \underbrace{\mathsf{s}(\ldots(\mathsf{s}(\overline{0}))\ldots)}_{n-mal}$**
  - Codierung der Zahl n als Term der formalen Sprache von $\mathcal{Q}$
- **Definiere: $f : \mathbb{N}^k \rightarrow \mathbb{N}$ 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:

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

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

  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. $\models_{\mathcal{Q}}$ ADD$(\bar{i},\bar{j},\bar{k})$
  - Einfach, da Addition vordefiniert: ADD$(x_1,x_2,y)\quad\equiv\quad y$=$x_1$+$x_2$
- **Korrektheit der Repräsentation**
  - Zeige: für alle $i,j,k\in\mathbb{N}$ mit $i+j=k$ gilt $\models_{\mathcal{Q}}\ \bar{k}$=$\bar{i}$+$\bar{j}$ ($\hat{=}$ ADD$(\bar{i},\bar{j},\bar{k})$)
    und für alle $i,j,k\in\mathbb{N}$ mit $i+j\neq k$ gilt $\models_{\mathcal{Q}}\ \bar{k}\neq\bar{i}$+$\bar{j}$

  Sei i beliebig, aber fest. Wir führen den Beweis durch Induktion über j:
  - Für $j=0$ folgt $i=k$, also $\bar{i}$=$\bar{k}$ und über Axiom Q_4: $\models_{\mathcal{Q}}\ \bar{i}$+$\bar{0}$ = $\bar{i}$
  - Es gelte $\models_{\mathcal{Q}}\ \bar{n}$=$\bar{i}$+$\bar{j}$ für alle n und $j=m\in\mathbb{N}$ mit $i+j=n$
  - Es sei $j=m+1$, also $\bar{j}$=s$(\bar{m})$ und es gelte $i+j=k$.
    Dann gilt $k=i+m+1=n+1$ und $\bar{k}$=s$(\bar{n})$.
    Mit Axiom Q_5 folgt $\models_{\mathcal{Q}}\ \bar{i}$+$\bar{j}$ = $\bar{i}$+s$(\bar{m})$ = s$(\bar{i}$+$\bar{m})$ = s$(\bar{n})$ = $\bar{k}$

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

Beispiele repräsentierbarer Funktionen (II)

- **Multiplikation** $\mathtt{MUL}(x_1, x_2, y) \equiv y\mathtt{=}x_1\mathtt{*}x_2$
- **Vergleich $\leq$** (Prädikat) $\mathtt{LE}(x, y) \equiv \exists\mathtt{z}.x\mathtt{+z=}y$

  $<$ $\mathtt{LT}(x, y) \equiv \mathtt{LE}(\mathtt{s}(x), y)$
- **Subtraktion** $\mathtt{SUB}(x_1, x_2, y) \equiv x_1\mathtt{=}x_2\mathtt{+}y \lor (\mathtt{LE}(x_1, x_2) \land y\mathtt{=}\overline{\mathtt{0}})$
- **Division** $\mathtt{DIV}(x_1, x_2, y) \equiv \exists\mathtt{z}.\mathtt{LT(z,}x_2\mathtt{)} \land x_2\mathtt{*}y\mathtt{+z=}x_1$
- **Divisionsrest/Modulo**

  $\mathtt{MOD}(x_1, x_2, y) \equiv \mathtt{LT(}y\mathtt{,}x_2\mathtt{)} \land \exists\mathtt{z}.x_2\mathtt{*z+}y\mathtt{=}x_1$
- **Teilbarkeit** (Prädikat) $\mathtt{DIVIDES}(x_1, x_2) \equiv \exists\mathtt{z}.x_1\mathtt{*z=}x_2$
- **Primzahleigenschaft** (Prädikat)

  $\mathtt{PRIME}(x) \equiv \forall\mathtt{y}.(\mathtt{LT(y,}x\mathtt{)} \land \mathtt{LT(}\overline{\mathtt{1}}\mathtt{,y))} \Rightarrow \neg\mathtt{DIVIDES(y,}x\mathtt{)}$

Mehr Beispiele in den Übungen

REPRÄSENTIERBARKEIT IN $\mathcal{Q}$ IST TURING-MÄCHTIG (I)

Definiere Min-rekursive Funktionen

Defi nition rekursiver Funktionen ohne primitive Rekursion

- **$\mathcal{R}_{min}$: 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

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

- **$\mathcal{R} \subseteq \mathcal{R}_{min}$:** 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:** $\quad \mathtt{S}(x,y) \equiv y\mathtt{=s(}x\mathtt{)}$
- **Projektionsfunktionen pr_m^n:** $\quad \mathtt{PR}_m^n(x_1,..,x_n,y) \equiv y\mathtt{=}x_m$
- **Konstantenfunktion c_m^n:** $\quad \mathtt{C}_m^n(x_1,..,x_n,y) \equiv y\mathtt{=}\overline{m}$
- **Addition add:** $\quad \mathtt{ADD}(x_1,x_2,y) \equiv y\mathtt{=}x_1\mathtt{+}x_2$
- **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 $\mu[f]$:**

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

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