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

Der λ-Kalkül

Grundlage funktionaler Programmiersprachen

- **Einfacher mathematischer Mechanismus**
  - Funktionen werden definiert und angewandt

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

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**

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:

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:

     `f(x) = 2*x+3` $\quad$ `f` $\hat{=}$ `x` $\mapsto$ `2*x+3`

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:

     `f` $\hat{=}$ `\x. 2*x+3` Abstraktion von `x`

     Name der Funktion irrelevant für Beschreibung des Verhaltens

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

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:

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

     Name der Funktion irrelevant für Beschreibung des Verhaltens
  2. Anwendung der Funktion (ohne Auswertung):

     `f(4)`

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:

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

     Name der Funktion irrelevant für Beschreibung des Verhaltens
  2. Anwendung der Funktion (ohne Auswertung):

     $\mathtt{f(4)} \mathrel{\hat{=}} \mathtt{(}\lambda\mathtt{x.\,2*x+3)(4)}$ **Applikation**

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):

     $(\lambda\texttt{x.2*x+3)(4)}$

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:

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

     Name der Funktion irrelevant für Beschreibung des Verhaltens
  2. Anwendung der Funktion (ohne Auswertung):

     $\mathtt{f(4)} \hat{=} (\lambda\mathtt{x.2*x+3)(4)}$ **Applikation**
  3. Auswertung einer Funktionsanwendung (tatsächliches Ausrechnen):

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

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:

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

     Name der Funktion irrelevant für Beschreibung des Verhaltens
  2. Anwendung der Funktion (ohne Auswertung):

     $\texttt{f(4)} \mathrel{\hat{=}} \texttt{(}\lambda\texttt{x.2*x+3)(4)}$ **Applikation**
  3. Auswertung einer Funktionsanwendung (tatsächliches Ausrechnen):

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

λ-KALKÜL – SYNTAX

- **Einfache Programmiersprache: $\boldsymbol{\lambda}$-Terme**
  - Variablen x
  - $\lambda x\,\mathbf{.}\,t$, 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**
  - $\mathbf{(}\,t\,\mathbf{)}$, wobei t λ-Term

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: $f\ \ t_1\ \ t_2 \stackrel{\wedge}{=} (f\ \ t_1)\ \ t_2$
  - Notation $f(t_1,\ldots,t_n)$ steht für iterierte Applikation $f\ \ t_1\ \ ..\ \ t_n$

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: $f\ t_1\ t_2 \mathrel{\hat{=}} (f\ t_1)\ t_2$
  - Notation $f(t_1,..,t_n)$ steht für iterierte Applikation $f\ t_1\ ..\ t_n$
- **Beispiele für λ-Terme**
  - x Symbole sind immer Variablen

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
  - λ`f.`λ`x.f(x)` Anwendung einer Funktion
  - λ`f.`λ`g.`λ`x. f g (g x)` Funktionen höherer Ordnung

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: $f\ t_1\ t_2 \hat{=} (f\ t_1)\ t_2$
  - 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** $(\lambda x\,.\,t)(b) \xrightarrow{\beta} t[b/x]$

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*

λ-Kalkül – Berechnung durch Auswertung

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

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

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

- **Substitution und Reduktion am Beispiel**

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

λ-Kalkül – Berechnung durch Auswertung

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

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

$$\lfloor\lambda \mathtt{f}.\lambda \mathtt{x}.\ \mathtt{n\ f\ (f\ x)}\rfloor[(\lambda \mathtt{f}.\lambda \mathtt{x}.\mathtt{x})/\mathtt{n}]$$

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**

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

$$\longrightarrow \quad \lambda\mathtt{f}.\lambda\mathtt{x}.\ (\lambda\mathtt{f}.\lambda\mathtt{x}.\mathtt{x})\ \mathtt{f\ (f\ x)}$$

λ-Kalkül – Berechnung durch Auswertung

- **Ersetze Funktionsparameter durch -argumente**
  - **Reduktion** $(\lambda x.t)(b) \xrightarrow{\beta} t[b/x]$
  - **Substitution** $t[b/x]$: 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**

$$(\lambda n.\lambda f.\lambda x.\ n\ f\ (f\ x))\ (\lambda f.\lambda x.x)$$

$$\longrightarrow\quad \lambda f.\lambda x.\ (\lambda f.\lambda x.x)\ f\ (f\ x)$$

λ-Kalkül – Berechnung durch Auswertung

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

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

$\longrightarrow$

$$\lambda \mathtt{f}.\lambda \mathtt{x}.\ \lfloor(\lambda \mathtt{x}.\mathtt{x})\rfloor\ [\mathtt{f}/\mathtt{f}]\ (\mathtt{f\ x})$$

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

```
  (λn.λf.λx. n f (f x)) (λf.λx.x)
⟶ λf.λx. (λf.λx.x) f (f x)
⟶ λf.λx. (λx.x) (f x)
```

λ-Kalkül – Berechnung durch Auswertung

- **Ersetze Funktionsparameter durch -argumente**
  - **Reduktion** $(\lambda x.t)(b) \xrightarrow{\beta} t[b/x]$
  - **Substitution** $t[b/x]$: 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**

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

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**

$$
\begin{array}{ll}
 & (\lambda \mathtt{n}.\lambda \mathtt{f}.\lambda \mathtt{x}.\ \mathtt{n}\ \mathtt{f}\ (\mathtt{f}\ \mathtt{x}))\ (\lambda \mathtt{f}.\lambda \mathtt{x}.\mathtt{x}) \\
\longrightarrow & \lambda \mathtt{f}.\lambda \mathtt{x}.\ (\lambda \mathtt{f}.\lambda \mathtt{x}.\mathtt{x})\ \mathtt{f}\ (\mathtt{f}\ \mathtt{x}) \\
\longrightarrow & \\
 & \lambda \mathtt{f}.\lambda \mathtt{x}.\ \lfloor \mathtt{x} \rfloor\ [(\mathtt{f}\ \mathtt{x})/\mathtt{x}]
\end{array}
$$

λ-Kalkül – Berechnung durch Auswertung

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

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

λ-Kalkül – Berechnung durch Auswertung

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

```
      (λn.λf.λx. n f (f x)) (λf.λx.x)
⟶     λf.λx. (λf.λx.x) f (f x)
⟶     λf.λx. (λx.x) (f x)
⟶     λf.λx. (f x)
```

λ-Kalkül – Berechnung durch Auswertung

- **Ersetze Funktionsparameter durch -argumente**
  - **Reduktion** $(\lambda x.t)(b) \xrightarrow{\beta} t[b/x]$
  - **Substitution** $t[b/x]$: 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**

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

$$\Downarrow$$

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

VOM λ-KALKÜL ZU ECHTEN PROGRAMMEN

- **λ-Kalkül ist der Basismechanismus**
  - Die Assemblersprache funktionaler Programme
  - Spezialhardware (Lisp-Maschinen) kann λ-Terme direkt auswerten

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

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

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

Darstellung Boolescher Operatoren im λ-Kalkül

Zwei verschiedene Objekte und ein Test

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

Konditional(-simulation) ist invers zu T und F

if T then s else t

Darstellung Boolescher Operatoren im λ-Kalkül

Zwei verschiedene Objekte und ein Test

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

Konditional(-simulation) ist invers zu T und F

$$\begin{array}{ll} & \text{if } \mathsf{T} \;\text{ then } s \;\text{ else } t \\ \equiv & \mathsf{T}\,s\,t \end{array}$$

Darstellung Boolescher Operatoren im λ-Kalkül

Zwei verschiedene Objekte und ein Test

$$\mathsf{T} \equiv \lambda \mathtt{x}.\lambda \mathtt{y}.\mathtt{x}$$

$$\mathsf{F} \equiv \lambda \mathtt{x}.\lambda \mathtt{y}.\mathtt{y}$$

$$\text{if } b \text{ then } s \text{ else } t \equiv b\,s\,t$$

Konditional(-simulation) ist invers zu T und F

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

Darstellung Boolescher Operatoren im λ-Kalkül

Zwei verschiedene Objekte und ein Test

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

Konditional(-simulation) ist invers zu T und F

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

Darstellung Boolescher Operatoren im λ-Kalkül

Zwei verschiedene Objekte und ein Test

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

Konditional(-simulation) ist invers zu T und F

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

Darstellung Boolescher Operatoren im λ-Kalkül

Zwei verschiedene Objekte und ein Test

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

Konditional(-simulation) ist invers zu T und F

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

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

Paare: Datenkapselung und Komponentenzugriff

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

Paare: Datenkapselung und Komponentenzugriff

$$(u, v) \equiv \lambda\texttt{pair}.\ \texttt{pair}\ u\ v$$

$$pair.1 \equiv pair\ (\lambda\texttt{x}.\lambda\texttt{y}.\texttt{x})$$

$$pair.2 \equiv pair\ (\lambda\texttt{x}.\lambda\texttt{y}.\texttt{y})$$

$$\textsf{let}\ (x, y) = pair\ \textsf{in}\ t \equiv pair\ (\lambda x.\lambda y.t)$$

Analyseoperator ist invers zur Paarbildung

$$\textsf{let}\ (x, y) = (u, v)\ \textsf{in}\ t$$

Paare: Datenkapselung und Komponentenzugriff

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

Analyseoperator ist invers zur Paarbildung

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

Paare: Datenkapselung und Komponentenzugriff

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

Analyseoperator ist invers zur Paarbildung

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

Paare: Datenkapselung und Komponentenzugriff

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

Analyseoperator ist invers zur Paarbildung

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

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

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

Paare: Datenkapselung und Komponentenzugriff

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

Analyseoperator ist invers zur Paarbildung

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

Simulation arithmetischer Operationen

- **Darstellung natürlicher Zahlen durch iterierte Terme**
  - Semantisch: wiederholte Anwendung von Funktionen
  - Repräsentiere die Zahl n durch den Term $\lambda \texttt{f}.\lambda \texttt{x}.\ \underbrace{\texttt{f (f..(f x)..)}}_{n\text{-}\textit{mal}}$

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: $\overline{\boldsymbol{n}} \;\equiv\; \lambda \mathtt{f}.\lambda \mathtt{x}.\,\mathtt{f}^n\,\mathtt{x}$
  - Bezeichnung: **Church Numerals**

Simulation arithmetischer Operationen

- **Darstellung natürlicher Zahlen durch iterierte Terme**
  - Semantisch: wiederholte Anwendung von Funktionen
  - Repräsentiere die Zahl n durch den Term $\lambda\mathtt{f}.\lambda\mathtt{x}.\underbrace{\mathtt{f\,(f..(f\,x)..)}}_{n\textit{-mal}}$
  - 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

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

- **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:** $\texttt{s} \equiv \lambda\texttt{n}.\lambda\texttt{f}.\lambda\texttt{x}.\ \texttt{n f (f x)}$

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 $\overline{n\mathtt{+1}}$

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 $\mathtt{s}\ \overline{\boldsymbol{n}}$ ist der Term $\overline{\boldsymbol{n}\mathtt{+1}}$

$$\mathtt{s}\ \overline{n} \quad\equiv\quad (\lambda\mathtt{n}.\lambda\mathtt{f}.\lambda\mathtt{x}.\ \mathtt{n\ f\ (f\ x)})\ (\lambda\mathtt{f}.\lambda\mathtt{x}.\ \mathtt{f}^{n}\,\mathtt{x})$$

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

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

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

$$
\begin{aligned}
\texttt{s}\ \overline{n} \quad &\equiv \quad (\lambda\texttt{n}.\lambda\texttt{f}.\lambda\texttt{x}.\ \texttt{n f (f x)})\ (\lambda\texttt{f}.\lambda\texttt{x}.\ \texttt{f}^n\texttt{x})\\
&\longrightarrow \lambda\texttt{f}.\lambda\texttt{x}.\ (\lambda\texttt{f}.\lambda\texttt{x}.\ \texttt{f}^n\texttt{x})\ \texttt{f (f x)}\\
&\longrightarrow \lambda\texttt{f}.\lambda\texttt{x}.\ (\lambda\texttt{x}.\ \texttt{f}^n\texttt{x})\ \texttt{(f x)}
\end{aligned}
$$

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

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

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

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

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

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

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

$$\begin{array}{rcl} \texttt{s}\ \overline{n} & \equiv & (\lambda\texttt{n}.\lambda\texttt{f}.\lambda\texttt{x}.\ \texttt{n f (f x)})\ (\lambda\texttt{f}.\lambda\texttt{x}.\ \texttt{f}^n\texttt{x}) \\ & \longrightarrow & \lambda\texttt{f}.\lambda\texttt{x}.\ (\lambda\texttt{f}.\lambda\texttt{x}.\ \texttt{f}^n\texttt{x})\ \texttt{f (f x)} \\ & \longrightarrow & \lambda\texttt{f}.\lambda\texttt{x}.\ (\lambda\texttt{x}.\ \texttt{f}^n\texttt{x})\ \texttt{(f x)} \\ & \longrightarrow & \lambda\texttt{f}.\lambda\texttt{x}.\ \texttt{f}^n\ \texttt{(f x)} \\ & \longrightarrow & \lambda\texttt{f}.\lambda\texttt{x}.\ \texttt{f}^{n+1}\ \texttt{x} \qquad \equiv \quad \overline{n\texttt{+1}} \end{array}$$

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

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 $\overline{n\mathtt{+1}}$

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

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

Programmierung im λ-Kalkül

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

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

- **Addition:** $\texttt{add} \equiv \lambda\texttt{m}.\lambda\texttt{n}.\lambda\texttt{f}.\lambda\texttt{x}.\ \texttt{m f (n f x)}$
- **Multiplikation:** $\texttt{mul} \equiv \lambda\texttt{m}.\lambda\texttt{n}.\lambda\texttt{f}.\lambda\texttt{x}.\ \texttt{m (n f) x}$
- **Test auf Null:** $\texttt{zero} \equiv \lambda\texttt{n}.\ \texttt{n}\ (\lambda\texttt{n}.\texttt{F})\ \texttt{T}$

Programmierung im λ-Kalkül

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

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

- **Addition:** $\mathsf{add} \equiv \lambda\texttt{m}.\lambda\texttt{n}.\lambda\texttt{f}.\lambda\texttt{x}.\ \texttt{m f (n f x)}$
- **Multiplikation:** $\mathsf{mul} \equiv \lambda\texttt{m}.\lambda\texttt{n}.\lambda\texttt{f}.\lambda\texttt{x}.\ \texttt{m (n f) x}$
- **Test auf Null:** $\mathsf{zero} \equiv \lambda\texttt{n}.\ \texttt{n}\ (\lambda\texttt{n}.\texttt{F})\ \texttt{T}$
- **Vorgängerfunktion:**

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

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

Programmierung rekursiver Funktionen

- **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
    $$\boldsymbol{F\,x} \;=\; (\lambda f.\lambda x.t[f,x])\;F\;\;x \xrightarrow{*} \boldsymbol{t[F,x]}$$
  - Kurzschreibweise: $F \;\equiv\;$ letrec $f(x) = t$ $\;(\equiv R(\lambda f.\lambda x.t))$

Programmierung rekursiver Funktionen

- **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

    $$\boldsymbol{F\,x} = (\lambda f.\lambda x.t[f,x])\;F\;\;x \xrightarrow{*} \boldsymbol{t[F,x]}$$
  - Kurzschreibweise: $F \equiv$ letrec $f(x)=t$ $(\equiv R(\lambda f.\lambda x.t))$
- **Y-Kombinator:** $\mathtt{Y} \equiv \lambda\mathtt{f}.(\lambda\mathtt{x}.\mathtt{f}(\mathtt{x}\,\mathtt{x}))(\lambda\mathtt{x}.\mathtt{f}(\mathtt{x}\,\mathtt{x}))$
  - Bekanntester Fixpunktkombinator

- **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

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

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

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

Der λ-Kalkül ist Turing-mächtig

Alle μ-rekursiven Funktionen sind λ-berechenbar

- **Nachfolgerfunktion s:** $\quad$ `s` $\equiv$ λ`n.`λ`f.`λ`x. n f (f x)`

DER λ-KALKÜL IST TURING-MÄCHTIG

Alle μ-rekursiven Funktionen sind λ-berechenbar

- **Nachfolgerfunktion s:** $\quad$ `s` $\equiv$ λ`n.`λ`f.`λ`x. n f (f x)`
- **Projektionsfunktionen pr_m^n:** $\quad$ $\mathtt{pr}_m^n \equiv \lambda \mathtt{x}_1 \mathtt{..} \lambda \mathtt{x}_n\mathtt{.}\ \mathtt{x}_m$

Der λ-Kalkül ist Turing-mächtig

Alle μ-rekursiven Funktionen sind λ-berechenbar

- **Nachfolgerfunktion s:** $\quad \texttt{s} \equiv \lambda\texttt{n}.\lambda\texttt{f}.\lambda\texttt{x}.\ \texttt{n f (f x)}$
- **Projektionsfunktionen pr_m^n:** $\quad \texttt{pr}_m^n \equiv \lambda\texttt{x}_1..\lambda\texttt{x}_n.\ \texttt{x}_m$
- **Konstantenfunktion c_m^n:** $\quad \texttt{c}_m^n \equiv \lambda\texttt{x}_1..\lambda\texttt{x}_n.\ \overline{\texttt{m}}$

Alle μ-rekursiven Funktionen sind λ-berechenbar

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

Alle μ-rekursiven Funktionen sind λ-berechenbar

- **Nachfolgerfunktion s:** $\quad$ `s` $\equiv$ `λn.λf.λx. n f (f x)`
- **Projektionsfunktionen pr_m^n:** $\quad$ $\mathtt{pr}_m^n \equiv \lambda \mathtt{x}_1..\lambda \mathtt{x}_n.\ \mathtt{x}_m$
- **Konstantenfunktion c_m^n:** $\quad$ $\mathtt{c}_m^n \equiv \lambda \mathtt{x}_1..\lambda \mathtt{x}_n.\ \overline{\mathtt{m}}$
- **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 $Pr[f, g]$:**

  PR $\equiv$ `λf.λg.` letrec `h(x)=λy.`if zero `y` then `f x` else `g x (p y) (h x (p y))`

Der λ-Kalkül ist Turing-mächtig

Alle μ-rekursiven Funktionen sind λ-berechenbar

- **Nachfolgerfunktion s:** $\quad$ s $\equiv$ λn.λf.λx. n f (f x)

- **Projektionsfunktionen pr_m^n:** $\quad$ $\mathrm{pr}_m^n \equiv \lambda x_1..\lambda x_n.\, x_m$

- **Konstantenfunktion c_m^n:** $\quad$ $\mathrm{c}_m^n \equiv \lambda x_1..\lambda x_n.\, \overline{m}$

- **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 $\equiv$ λf.λg. letrec h(x) = λy.if zero y then f x else g x (p y) (h x (p y))

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

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

Rechtfertigung logischer Programmiersprachen

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

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** $\boldsymbol{T}$ gegeben durch formale Sprache und Axiome
  - Formel F ist gültig in T ($\models_{\boldsymbol{T}} \boldsymbol{F}$), wenn F logisch aus den Axiomen folgt

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 $\boldsymbol{T}$** gegeben durch formale Sprache und Axiome
  - Formel F ist gültig in T ($\models_{\boldsymbol{T}} \boldsymbol{F}$), wenn F logisch aus den Axiomen folgt
- **Berechenbarkeitsbegriff: $\boldsymbol{f}{:}\mathbb{N}^{\boldsymbol{k}}{\longrightarrow}\mathbb{N}$ repräsentierbar in $\boldsymbol{T}$**
  - Es gibt eine Formel F mit $\boldsymbol{f(i_1,..,i_k){=}j}$ 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 $*$

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$_7$: $\forall x, y. x * \mathsf{s}(y) = (x * y) + x$

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

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 $\mathcal{Q}$

- $\boldsymbol{f}$**:**$\mathbb{N}^{\boldsymbol{k}} \longrightarrow \mathbb{N}$ **arithmetisch repräsentierbar**
  - f ist repräsentierbar in $\mathcal{Q}$, wobei $n \in \mathbb{N}$ codiert als $\overline{n} \equiv \underbrace{\mathsf{s}(\ldots(\mathsf{s}(\overline{0}))\ldots)}_{n-mal}$

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 $\overline{n} \equiv \underbrace{\mathsf{s}(\ldots(\mathsf{s}(\overline{0}))\ldots)}_{n-mal}$
- **Beispiele arithmetisch repräsentierbarer Funktionen**

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

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

Repräsentierbarkeit in $\mathcal{Q}$

- $\boldsymbol{f}$**:**$\mathbb{N}^{\boldsymbol{k}} \longrightarrow \mathbb{N}$ **arithmetisch repräsentierbar**
  - f ist repräsentierbar in $\mathcal{Q}$, wobei $n \in \mathbb{N}$ codiert als $\overline{n} \equiv \underbrace{\mathsf{s}(\ldots(\mathsf{s}(\overline{0}))\ldots)}_{n-mal}$
- **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: $\mathsf{ADD}(x_1, x_2, y) \equiv y\mathtt{=}x_1\mathtt{+}x_2$

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

Repräsentierbarkeit in $\mathcal{Q}$

- $\boldsymbol{f}$**:**$\mathbb{N}^{\boldsymbol{k}}\longrightarrow\mathbb{N}$ **arithmetisch repräsentierbar**
  - f ist repräsentierbar in $\mathcal{Q}$, wobei $n\in\mathbb{N}$ codiert als $\overline{n} \equiv \underbrace{\mathsf{s}(\ldots(\mathsf{s}(\overline{0}))\ldots)}_{n-mal}$
- **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: $\mathsf{ADD}(x_1,x_2,y) \equiv y\mathtt{=}x_1\mathtt{+}x_2$

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

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

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

REPRÄSENTIERBARKEIT IN $\mathcal{Q}$

- $\boldsymbol{f}:\mathbb{N}^{\boldsymbol{k}}\longrightarrow\mathbb{N}$ **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: $\mathtt{ADD}(x_1,x_2,y) \equiv y\mathtt{=}x_1\mathtt{+}x_2$

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

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

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

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

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

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

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

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

Subtraktion $\mathtt{SUB}(x_1, x_2, y) \equiv x_1\mathtt{=}x_2\mathtt{+}y \ \lor\ (\mathtt{LE}(x_1, x_2) \land y\mathtt{=}\overline{0})$

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

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

Repräsentierbarkeit in $\mathcal{Q}$

- $\boldsymbol{f}$**:**$\mathbb{N}^{\boldsymbol{k}}\longrightarrow\mathbb{N}$ **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: $\mathtt{ADD}(x_1,x_2,y) \;\equiv\; y\mathtt{=}x_1\mathtt{+}x_2$

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

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

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

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

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

- **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

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

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

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

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

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

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**

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

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, $\ldots$)

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 $\overline{n}$** $\equiv \underbrace{\mathsf{s}(\ldots(\mathsf{s}(\bar{\mathbf{0}}))\ldots)}_{n-mal}$
  - Codierung der Zahl n als Term der formalen Sprache von $\mathcal{Q}$

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: $f : \mathbb{N}^k \longrightarrow \mathbb{N}$ 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:

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

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

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:** $f: \mathbb{N}^k \to \mathbb{N}$ **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

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. $\models_{\mathcal{Q}} \mathtt{ADD}(\bar{i},\bar{j},\bar{k})$

Beispiele repräsentierbarer Funktionen

- **Addition**
  - Bestimme eine 3-stellige Formel ADD mit $i+j=k$ gdw. $\models_{\mathcal{Q}} \mathsf{ADD}(\bar{i},\bar{j},\bar{k})$
  - Einfach, da Addition vordefiniert: $\mathsf{ADD}(x_1,x_2,y) \quad\equiv\quad y\texttt{=}x_1\texttt{+}x_2$

Beispiele repräsentierbarer Funktionen

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

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

Beispiele repräsentierbarer Funktionen

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

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

Sei i beliebig, aber fest.

Beispiele repräsentierbarer Funktionen

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

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

Sei i beliebig, aber fest. Wir führen den Beweis durch Induktion über j:

Beispiele repräsentierbarer Funktionen

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

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

  Sei i beliebig, aber fest. Wir führen den Beweis durch Induktion über j:

  - Für $j = 0$ folgt $i=k$, also $\bar{i}\mathsf{=}\overline{k}$ und über Axiom Q_4: $\models_{\mathcal{Q}} \;\bar{i}\mathsf{+}\bar{0}\mathsf{=}\bar{i}$

Beispiele repräsentierbarer Funktionen

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

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

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

Beispiele repräsentierbarer Funktionen

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

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

Sei i beliebig, aber fest. Wir führen den Beweis durch Induktion über j:

- Für $j=0$ folgt $i=k$, also $\overline{i}\mathsf{=}\overline{k}$ und über Axiom Q$_4$: $\models_{\mathcal{Q}}\ \overline{i}\mathsf{+}\overline{0}\mathsf{=}\overline{i}$
- Es gelte $\models_{\mathcal{Q}}\ \overline{n}\mathsf{=}\overline{i}\mathsf{+}\overline{j}$ 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 $\overline{k}\mathsf{=s}(\overline{n})$.

Beispiele repräsentierbarer Funktionen

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

    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 $\bar{j}{=}\mathsf{s}(\bar{m})$ und es gelte $i{+}j{=}k$.

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

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

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: $\mathsf{ADD}(x_1, x_2, y) \quad\equiv\quad y\mathtt{=}x_1\mathtt{+}x_2$
- **Korrektheit der Repräsentation**
  - Zeige: für alle $i, j, k \in \mathbb{N}$ mit $i{+}j{=}k$ gilt $\models_{\mathcal{Q}}\ \overline{k}\mathtt{=}\overline{i}\mathtt{+}\overline{j}$ $\quad(\hat{=}\ \mathtt{ADD}(\overline{i},\overline{j},\overline{k}))$

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

  Sei i beliebig, aber fest. Wir führen den Beweis durch Induktion über j:
  - Für $j = 0$ folgt $i{=}k$, also $\overline{i}\mathtt{=}\overline{k}$ und über Axiom Q$_4$: $\models_{\mathcal{Q}}\ \overline{i}\mathtt{+}\overline{0}\mathtt{=}\overline{i}$
  - Es gelte $\models_{\mathcal{Q}}\ \overline{n}\mathtt{=}\overline{i}\mathtt{+}\overline{j}$ für alle n und $j{=}m \in \mathbb{N}$ mit $i{+}j{=}n$
  - Es sei $j{=}m{+}1$, also $\overline{j}\mathtt{=}\mathtt{s}(\overline{m})$ und es gelte $i{+}j{=}k$.

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

    Mit Axiom Q$_5$ folgt $\models_{\mathcal{Q}}\ \overline{i}\mathtt{+}\overline{j}\ \mathtt{=}\ \overline{i}\mathtt{+}\mathtt{s}(\overline{m})\ \mathtt{=}\ \mathtt{s}(\overline{i}\mathtt{+}\overline{m})\ \mathtt{=}\ \mathtt{s}(\overline{n})\ \mathtt{=}\ \overline{k}$

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

Beispiele repräsentierbarer Funktionen (II)

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

Beispiele repräsentierbarer Funktionen (II)

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

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

Beispiele repräsentierbarer Funktionen (II)

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

  $< \quad \mathtt{LT}(x, y) \equiv \mathtt{LE}(\mathtt{s}(x), y)$
- **Subtraktion** $\quad \mathtt{SUB}(x_1, x_2, y) \equiv x_1\mathtt{=}x_2\mathtt{+}y \vee (\mathtt{LE}(x_1, x_2) \wedge y\mathtt{=}\overline{\mathtt{0}})$

Beispiele repräsentierbarer Funktionen (II)

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

  $<$ $\quad \mathtt{LT}(x, y) \equiv \mathtt{LE}(\mathtt{s}(x), y)$
- **Subtraktion** $\quad \mathtt{SUB}(x_1, x_2, y) \equiv x_1 \mathtt{=} x_2 \mathtt{+} y \vee (\mathtt{LE}(x_1, x_2) \wedge y \mathtt{=} \overline{\mathtt{0}})$
- **Division** $\quad \mathtt{DIV}(x_1, x_2, y) \equiv \exists \mathtt{z.\,LT(z,} x_2\mathtt{)} \wedge x_2 \mathtt{*} y \mathtt{+z=} x_1$

Beispiele repräsentierbarer Funktionen (II)

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

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

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

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** $\texttt{DIV}(x_1, x_2, y) \equiv \exists\texttt{z.LT(z,}x_2\texttt{)} \wedge x_2\texttt{*}y\texttt{+z=}x_1$
- **Divisionsrest/Modulo**

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

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

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

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

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

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

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

Mehr Beispiele in den Übungen

Definiere Min-rekursive Funktionen

Definition rekursiver Funktionen ohne primitive Rekursion

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

Definiere Min-rekursive Funktionen

Definition 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

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 f¨ur Vergleiche mit anderen Modellen

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 f¨ur Vergleiche mit anderen Modellen

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

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

Definiere Min-rekursive Funktionen

Definition 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

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

Alle min-rekursiven Funktionen sind repräsentierbar

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

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

Alle min-rekursiven Funktionen sind repräsentierbar

- **Nachfolgerfunktion s:** $\mathsf{S}(x,y) \equiv y\mathtt{=s(}x\mathtt{)}$
- **Projektionsfunktionen pr_m^n:** $\mathsf{PR}_m^n(x_1,..,x_n,y) \equiv y\mathtt{=}x_m$

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

Alle min-rekursiven Funktionen sind repräsentierbar

- **Nachfolgerfunktion s:** $\quad \mathsf{S}(x,y) \;\equiv\; y\mathsf{=s(}x\mathsf{)}$
- **Projektionsfunktionen pr_m^n:** $\quad \mathsf{PR}_m^n(x_1,..,x_n,y) \;\equiv\; y\mathsf{=}x_m$
- **Konstantenfunktion c_m^n:** $\quad \mathsf{C}_m^n(x_1,..,x_n,y) \;\equiv\; y\mathsf{=}\overline{m}$

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

Alle min-rekursiven Funktionen sind repräsentierbar

- **Nachfolgerfunktion s:** $\mathsf{S}(x,y) \equiv y\mathsf{=s(}x\mathsf{)}$
- **Projektionsfunktionen pr_m^n:** $\mathsf{PR}_m^n(x_1,..,x_n,y) \equiv y\mathsf{=}x_m$
- **Konstantenfunktion c_m^n:** $\mathsf{C}_m^n(x_1,..,x_n,y) \equiv y\mathsf{=}\overline{m}$
- **Addition add:** $\mathsf{ADD}(x_1,x_2,y) \equiv y\mathsf{=}x_1\mathsf{+}x_2$

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

Alle min-rekursiven Funktionen sind repräsentierbar

- **Nachfolgerfunktion s:** $\quad \mathsf{S}(x,y) \;\equiv\; y\texttt{=}\mathsf{s}(x)$
- **Projektionsfunktionen pr_m^n:** $\quad \mathsf{PR}_m^n(x_1,..,x_n,y) \;\equiv\; y\texttt{=}x_m$
- **Konstantenfunktion c_m^n:** $\quad \mathsf{C}_m^n(x_1,..,x_n,y) \;\equiv\; y\texttt{=}\overline{m}$
- **Addition add:** $\quad \mathsf{ADD}(x_1,x_2,y) \;\equiv\; y\texttt{=}x_1\texttt{+}x_2$
- **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 $f, g_1,..,g_n$

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^n_m:** $\quad \mathtt{PR}^n_m(x_1,..,x_n,y) \;\equiv\; y\mathtt{=}x_m$

- **Konstantenfunktion c^n_m:** $\quad \mathtt{C}^n_m(x_1,..,x_n,y) \;\equiv\; y\mathtt{=}\overline{m}$

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

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

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