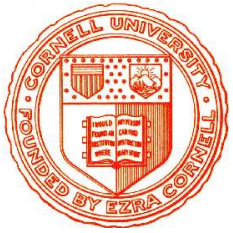
Automatisierte Logik und Programmierung

Einheit 5



Der λ-Kalkül

1. Syntax
2. Operationale Semantik
3. Formales Schließen über Berechnung
4. Ausdruckskraft

Logisches Schliessen über den Wert von Termen

- **Turing-mächtiges Berechenbarkeitsmodell**
  - Eine Logik der Berechnung, präzisiert Termsprache der Prädikatenlogik
  - Grundlage funktionaler Programmiersprachen wie Lisp, ML, Haskell, ...
- **Einfacher mathematischer Mechanismus**
  - Funktionen werden definiert und angewandt
  - Beschreibung des Funktionsverhaltens wird zum Namen der Funktion
  - Funktionswerte werden ausgerechnet durch Einsetzen von Werten
- **Leicht zu verstehende Basiskonzepte**

  1. Definition einer Funktion:

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

     Name der Funktion irrelevant für Beschreibung des Verhaltens

  2. Anwendung der Funktion:

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

  3. Auswertung der Funktion:

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

λ-Kalkül: Unterschiede zur Prädikatenlogik

Mehr Struktur in Beschreibung von Termen

- **Kein Alphabet für Funktionssymbole**
  - λ-Ausdrücke ersetzen Funktionssymbole
  - Interpretation von λ-Ausdrücken festgelegt
- **Regeln zur Auswertung von Termen**
  - Definieren Berechnungsmodell auf beliebigen Ausdrücken
- **Extensionale Gleichheit von Termen**
  - Gleichheit basiert auf Wert von Termen, nicht nur auf Struktur

$$\Downarrow$$

Präzisere Semantik formaler Ausdrücke

λ-Kalkül – Syntax

- **Alphabet für erlaubte Symbole**
  - $\mathcal{V}$: Variablensymbole
- **λ-Terme**
  - Variablen $x \in \mathcal{V}$
  - $\lambda x\,.\,t$, wobei $x \in \mathcal{V}$ und t λ-Term — λ-Abstraktion
  - $f\ t$, 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 $f\,(t_1, \ldots, t_n)$ steht für iterierte Applikation $f\ t_1\ \ldots\ t_n$

Beispiele für λ-Terme

`x`	**Symbole sind immer Variablen**
λ`f.`λ`x.f(x)`	
λ`f.`λ`g.`λ`x. f g (g x)`	**Funktionen höherer Ordnung**
`x(x)`	**Selbstanwendung**
`(`λ`x.x(x)) (`λ`x.x(x))`	**Fixpunkt**

λ-Kalkül – Semantik

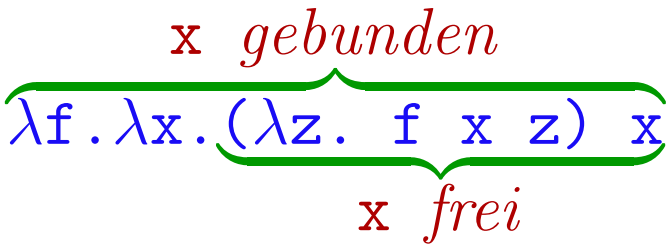
- **Modelltheoretische Semantik zu kompliziert**
  - Abbildung auf Funktionen in einfachem Universum nicht möglich
  - Modellierung benötigt Theorie vollständiger Halbordnungen (CPO's)
- **Operationale Semantik**
  - λ-Terme werden ausgewertet bis ihr Wert bestimmt ist
  - Wert ist ein (irreduzibler) λ-Term
- **Berechnung ist syntaktischer Mechanismus**
  - Auswertung $\hat{=}$ Ersetze Funktionsparameter durch Funktionsargumente
  - **Reduktion**: Substitution λ-gebundener Variablen durch λ-Terme

$$\Downarrow$$

Erweitere Konzept der Substitution

Vorkommen von Variablen in λ-Termen

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



$t[x_1, .., x_n]$: Ausdruck t hat mögliche freie Vorkommen von $x_1, \ldots, x_n$

t **geschlossen**, falls t keine freien Variablen enthält

Substitution $u[t/x]$ in λ-Termen

$$
\begin{array}{lll@{\qquad}lll}
\lfloor x \rfloor[t/x] & = & t & \lfloor x \rfloor[t/y] & = & x \quad (y{\neq}x) \\
\lfloor \lambda x\,.\,u \rfloor[t/x] & = & \lambda x\,.\,u & & & \\
\lfloor \lambda x\,.\,u \rfloor[t/y] & = & \lfloor \lambda z\,.\,u[z/x] \rfloor[t/y] \;{}^{*} & \lfloor \lambda x\,.\,u \rfloor[t/y] & = & \lambda x\,.\,\lfloor u[t/y] \rfloor \;{}^{**} \\
\lfloor f\ u \rfloor\sigma & = & f\sigma\ u\sigma & \lfloor (u) \rfloor\sigma & = & (u\sigma)
\end{array}
$$

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

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

Substitution ausgewertet

$$
\begin{aligned}
& \lfloor \lambda f.\lambda x.\ n\ f\ (f\ x) \rfloor [\lambda f.\lambda x.x / n] \\
= \lambda f.\, & \lfloor \lambda x.\ n\ f\ (f\ x) \rfloor [\lambda f.\lambda x.x / n] \\
= \lambda f.\lambda x.\ & \lfloor n\ f\ (f\ x) \rfloor [\lambda f.\lambda x.x / n] \\
= \lambda f.\lambda x.\ & \lfloor n\ f \rfloor [\lambda f.\lambda x.x / n]\ \lfloor (f\ x) \rfloor [\lambda f.\lambda x.x / n] \\
= \lambda f.\lambda x.\ & \lfloor n \rfloor [\lambda f.\lambda x.x / n]\ \lfloor f \rfloor [\lambda f.\lambda x.x / n] \\
& \quad (\lfloor f\ x \rfloor [\lambda f.\lambda x.x / n]) \\
= \lambda f.\lambda x.\ & (\lambda f.\lambda x.x)\ f \\
& \quad (\lfloor f \rfloor [\lambda f.\lambda x.x / n]\ \lfloor x \rfloor [\lambda f.\lambda x.x / n]) \\
= \lambda f.\lambda x.\ & (\lambda f.\lambda x.x)\ f\ (f\ x)
\end{aligned}
$$

Reduktion – informal

- **α-Konversion:**
  - Umbenennung gebundener Variablen in Termen
  - Ersetze Teilterms der Gestalt $\lambda x\,.\,u$ durch $\lambda z\,.\,u[z/x]$ $\quad (z \in \mathcal{V}$ neu$)$
- **Kongruenz $t \cong u$:**
  - t und u sind **α-konvertibel**
  - u entsteht aus t durch Umbenennung gebundener Variablen
- **Reduktion $(\lambda x\,.\,u)\,s \xrightarrow{\beta} u[s/x]$:**
  - Ersetze Teilterm der Gestalt $\underbrace{(\lambda x\,.\,u)\,s}_{\textbf{Redex}}$ durch $\underbrace{u[s/x]}_{\textbf{Kontraktum}}$
- **Reduzierbarkeit $t \xrightarrow{*} u$:**
  - u entsteht aus t durch endlich viele Reduktionen und α-Konversionen
- **Normalform:**
  - Term u ist nicht mehr reduzierbar
  - **Wert** von t: Normalform u mit $t \xrightarrow{*} u$

Konversion $\cong$ formal

α-Konversion: $\boldsymbol{\lambda x\,.\,u \cong \lambda z\,.\,u[z/x]}$ $\quad (z \in \mathcal{V}$ neu$)$

μ-Konversion: $f\ t \cong f\ u$, falls $t \cong u$

ν-Konversion: $f\ t \cong g\ t$, falls $f \cong g$

ξ-Konversion: $\lambda x\,.\,t \cong \lambda x\,.\,u$, falls $t \cong u$

ρ -Konversion: $t \cong t$

σ-Konversion: $t \cong u$, falls $u \cong t$

τ-Konversion: $t \cong u$, falls $t \cong s$ und $s \cong u$

Reduktion $\longrightarrow$ formal

β-Reduktion: $(\lambda x\,.\,u)\,s \longrightarrow u[s/x]$

μ–Reduktion: $f\ t \longrightarrow f\ u$, falls $t \longrightarrow u$

ν–Reduktion: $f\ t \longrightarrow g\ t$, falls $f \longrightarrow g$

ξ–Reduktion: $\lambda x\,.\,t \longrightarrow \lambda x.u$, falls $t \longrightarrow u$

τ–Reduktion: $t \longrightarrow u$, falls $t \longrightarrow s$ und $s \cong u$
oder $t \cong s$ und $s \longrightarrow u$

Reduzierbarkeit: $t \xrightarrow{*} u$, falls $t \xrightarrow{n} u$ für ein $n \in \mathbb{N}$

$t \xrightarrow{0} u$, falls $t \cong u$

$t \xrightarrow{n+1} u$, falls $t \longrightarrow s$ und $s \xrightarrow{n} u$

Reduktion am Beispiel

$$
\begin{aligned}
& (\lambda n.\lambda f.\lambda x.\ n\ f\ (f\ x))\ (\lambda f.\lambda x.x) \\
\longrightarrow\quad & \lambda f.\lambda x.\ (\lambda f.\lambda x.x)\ f\ (f\ x) \\
\longrightarrow\quad & \lambda f.\lambda x.(\lambda x.x)\ (f\ x) \\
\longrightarrow\quad & \lambda f.\lambda x.\ f\ x
\end{aligned}
$$

$$\Downarrow$$

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

Reduktion ist nicht deterministisch

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

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

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

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

$$\longrightarrow \quad \lambda x.\ \lambda y.\ y$$

Alternative Reduktionsreihenfolge

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

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

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

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

$$\longrightarrow \quad \lambda x.\ \lambda y.\ y$$

Semantik von λ-Termen

Interpretiere λ-Terme durch ihren Wert

- **u in Normalform**
  - u hat keine Redizes als Teilterme
- **u Normalform von t**
  - u in Normalform und $t \xrightarrow{*} u$
- **t normalisierbar**
  - es gibt eine Normalform u von t
- **$t = u$ (Extensionale Gleichheit)**
  - es gibt v mit $t \xrightarrow{*} v$ und $u \xrightarrow{*} v$
  - t und u sind semantisch gleich / konvertierbar

Hat jeder λ-Term einen eindeutigen Wert?

Eigenschaften des Berechnungskalküls

- **Hat jeder λ-Term eine Normalform?**

  Nein: $(\lambda x.\, x\ x)\ (\lambda x.\, x\ x)$ ist ein λ-Term ohne Normalform

- **Führt jede Reduktionsfolge zu einer Normalform?**

  Nein: Reduktionsfolgen normalisierbarer Terme müssen nicht terminieren

  $$(\lambda x.\lambda y.\, y)\ ((\lambda x.\, x\,x\,x)\ (\lambda x.\, x\,x\,x))\ (\lambda x.\, x)$$
  $$\longrightarrow (\lambda y.\, y)\ (\lambda x.\, x)$$
  $$\longrightarrow \lambda x.\, x$$

  Eine "innermost" Strategie führt bei diesem Term nicht zur Normalform

  $$(\lambda x.\lambda y.\, y)\ ((\lambda x.\, x\,x\,x)\ (\lambda x.\, x\,x\,x))\ (\lambda x.\, x)$$
  $$\overset{*}{\longrightarrow} (\lambda x.\lambda y.\, y)((\lambda x.\, x\,x\,x)\ (\lambda x.\, x\,x\,x)\ (\lambda x.\, x\,x\,x))\ (\lambda x.\, x)$$
  $$\overset{*}{\longrightarrow} (\lambda x.\lambda y.\, y)((\lambda x.\, x\,x\,x)(\lambda x.\, x\,x\,x)(\lambda x.\, x\,x\,x)(\lambda x.\, x\,x\,x))(\lambda x.\, x)$$
  $$\vdots \qquad\qquad \vdots$$

Eigenschaften des Berechnungskalküls (II)

- **Kann man eine Normalform immer finden?**
  - **Ja**: Reduktion des äußersten Redex ("leftmost reduction") führt zur Normalform, wenn es eine gibt — Beweis: Curry & Feys 1958, p142
- **Ist die Normalform eines λ-Terms eindeutig?**
  - **Ja**: Reduktion ist konfluent — Beweis: Barendregt 1981 §3.2

    Gilt $t \xrightarrow{*} u$ und $t \xrightarrow{*} v$, so gibt es s mit $u \xrightarrow{*} s$ und $v \xrightarrow{*} s$.

    (**Church-Rosser Theorem**)

  $\mapsto$ Alle Normalformen eines λ-Terms sind kongruent

  $\mapsto$ Semantisch gleiche Terme haben dieselben Normalformen (oder keine)

  $\mapsto$ Normalformen, die nicht kongruent sind, sind nicht semantisch gleich
- **Der λ-Kalkül ist eine Logik-freie Theorie**
  - Die Gleichheit von Termen ist der Hauptaspekt
  - Konnektive / Quantoren stehen außerhalb des reinen Kalküls

Ergänze prädikatenlogischen Kalkül um Regeln zur Auswertung von Termen

applyEq

$$\frac{\Gamma \vdash f\,t = g\,u}{\Gamma \vdash f = g \qquad \Gamma \vdash t = u}$$

lambdaEq *

$$\frac{\Gamma \vdash \lambda x\,.\,t = \lambda y\,.\,u}{\Gamma,\ x'{:}\mathtt{U} \vdash t[x'/x] = u[x'/y]}$$

reduction

$$\frac{\Gamma \vdash (\lambda x\,.\,u)\,s = t}{\Gamma \vdash u[s/x] = t}$$

**: Die Umbenennung $[x'/x]$ kann entfallen, wenn x nicht frei in Γ vorkommt*

Sequenzenbeweis für eine Reduktion

```
⊢ (λf.λx.f x (f f)) (λx.λy.y) = λx.λy.y                    BY reduction
\
 ⊢ λx.(λx.λy.y) x ((λx.λy.y)(λx.λy.y)) = λx.λy.y            BY lambdaEq
 \
   x:U ⊢ (λx.λy.y) x ((λx.λy.y)(λx.λy.y)) = λy.y            BY reduction
   \
    x:U ⊢ (λy.y)((λx.λy.y)(λx.λy.y)) = λy.y                 BY reduction
    \
     x:U ⊢ (λx.λy.y)(λx.λy.y) = λy.y                        BY reduction
     \
      x:U ⊢ λy.y = λy.y                                   BY reflexivity
```

Sequenzenbeweis für eine semantische Gleichheit

$$\mathbf{Y} \equiv \lambda f.\,(\lambda x.\,f\ (x\ x))\,(\lambda x.\,f\ (x\ x))$$

```
⊢ ∀t:U. Y t = t (Y t)                                                    BY all_i
\
 t:U ⊢ Y t = t (Y t)                                                     BY reduction
 \
  t:U ⊢ (λx.t (x x))(λx.t (x x)) = t (Y t)                               BY reduction
  \
   t:U ⊢ t (λx.t (x x))(λx.t (x x)) = t (Y t)                            BY symmetry
   \
    t:U ⊢ t (Y t) = t (λx.t (x x))(λx.t (x x))                           BY applyEq
    |\
    | t:U ⊢ t = t                                                       BY reflexivity
     \
      t:U ⊢ Y t = (λx.t (x x))(λx.t (x x))                              BY reduction
      \
       t:U ⊢ (λx.t (x x))(λx.t (x x)) = (λx.t (x x))(λx.t (x x))
                                                                        BY reflexivity
```

Eigenschaften des Beweiskalküls

- **Korrektheit**
  - Wenn $\vdash u{=}v$ im erweiterten Sequenzenkalkül bewiesen werden kann, dann sind u und v semantisch gleich ($u = v$)
  - *Die neuen Regeln sind korrekt in Bezug auf semantische Gleichheit*
- **Vollständigkeit**
  - Gilt $u = v$, so ist $\vdash u{=}v$ beweisbar
  - β-Reduktion wird von `reduction` abgedeckt
  - α- und ξ-Konversionen und -Reduktionen werden von `lambdaEq` erfaßt
  - μ- und ν-Konversionen und -Reduktionen werden von `applyEq` erfaßt
  - Die üblichen Gleichheitsregeln decken den Rest ab

Vom λ-Kalkül zu echten Programmen

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

Der λ-Kalkül kann alle berechenbaren Funktionen repräsentieren

Darstellung Boolescher Operatoren im λ-Kalkül

Zwei verschiedene Objekte und ein Konditional

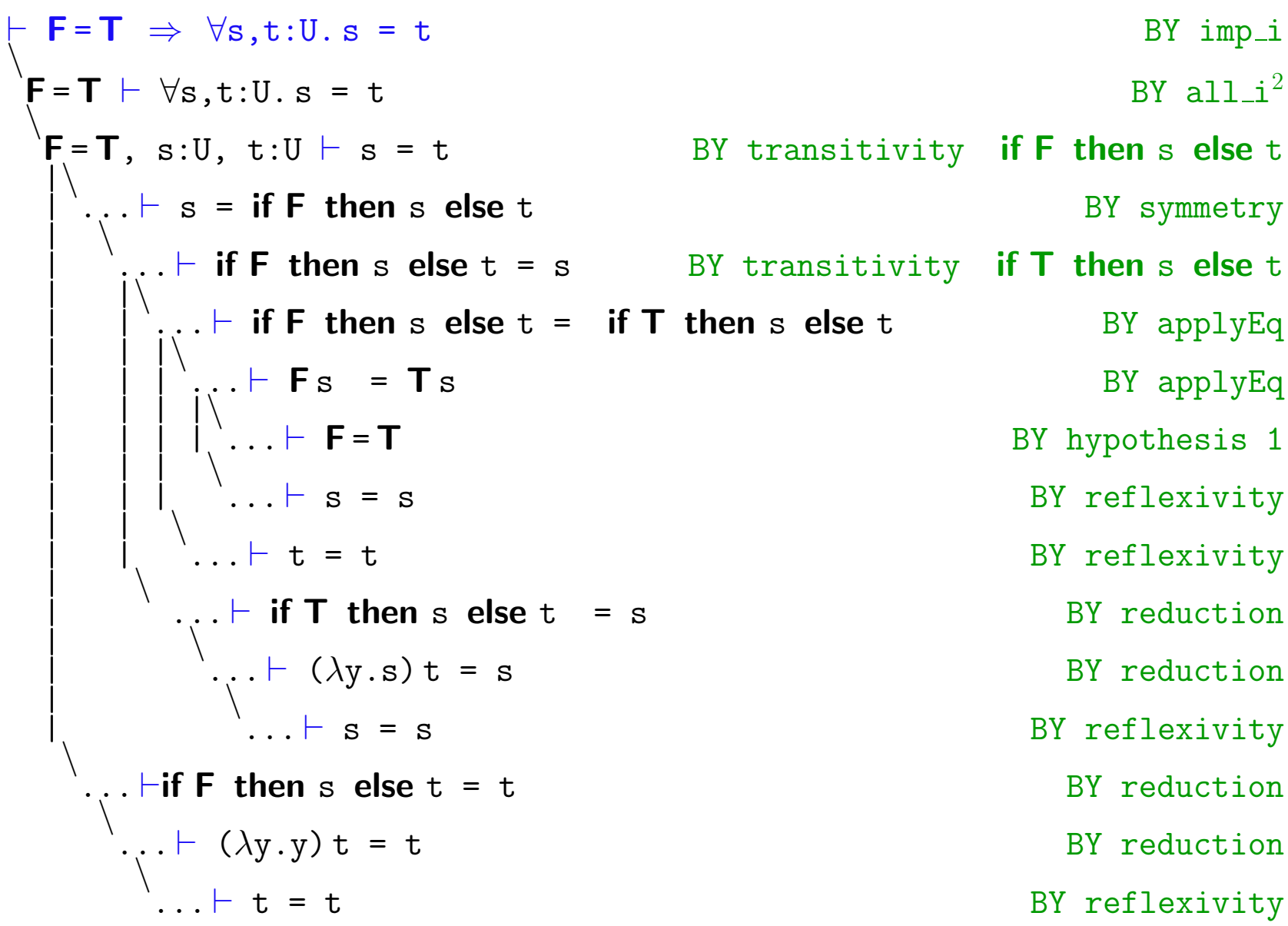
$$\begin{aligned} \mathsf{T} &\equiv \lambda \mathtt{x}\,.\,\lambda \mathtt{y}\,.\,\mathtt{x} \\ \mathsf{F} &\equiv \lambda \mathtt{x}\,.\,\lambda \mathtt{y}\,.\,\mathtt{y} \\ \textsf{if } b \textsf{ then } s \textsf{ else } t &\equiv b\,s\,t \end{aligned}$$

Konditional ist invers zu T und F

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

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

"Beweis" für $\mathsf{F} \neq \mathsf{T}$



Bildung und Analyse von Paaren

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

Analyseoperator ist invers zur Paarbildung

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

Operationen auf natürlichen Zahlen

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

$$\boldsymbol{f}(\boldsymbol{x}_1,..,\boldsymbol{x}_n) = \boldsymbol{m} \;\Leftrightarrow\; \boldsymbol{t}\;\overline{\boldsymbol{x}_1}\,..\overline{\boldsymbol{x}_n} = \overline{\boldsymbol{m}}$$

- **Operationen müssen Termvielfachheit berechnen**
  - z.B. muß **add** $\overline{\boldsymbol{m}}\;\overline{\boldsymbol{n}}$ als Wert immer den Term $\overline{\boldsymbol{m{+}n}}$ ergeben

Programmierung im λ-Kalkül

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

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

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

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

- **Einfache Rekursion:** $\mathsf{PRs}[b,h] \equiv \lambda n.\ n\ h\ b$

Auswertung der Additionsfunktion

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

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

Rekursionsgleichungen der Einfachen Rekursion

Sei $f \equiv \mathbf{PRs}[b,h] \equiv \lambda \mathtt{n}.\ \mathtt{n}\ h\ b$

$\vdash f\ \overline{0} = b$ BY reduction

$\vdash \overline{0}\ h\ b = b$ BY reduction

$\vdash (\lambda \mathtt{x}.\mathtt{x})\ b = b$ BY reduction

$\vdash b = b$ BY reflexivity

$\vdash f\ \overline{n+1} = h\ (f\ \overline{n})$ BY reduction

$\vdash \overline{n+1}\ h\ b = h\ (f\ \overline{n})$ BY reduction^2

$\vdash h^{n+1}\ b = h\ (f\ \overline{n})$ BY symmetry

$\vdash h\ (f\ \overline{n}) = h\ (h^n\ b)$ BY applyEq

$\vdash f\ \overline{n} = h^n\ b$ BY reduction

$\vdash \overline{n}\ h\ b = h^n\ b$ BY reduction^2

$\vdash h^n\ b = h^n\ b$ BY reflexivity

Darstellung von Listen

- **Erweiterung des Konzepts natürlicher Zahlen**
  - Wiederholte Anwendung einer Funktion auf Listenelemente
  - Repräsentiere $a_1..a_n$ durch den Term $\lambda\mathtt{f}.\lambda\mathtt{x}.\,\mathtt{f}\,a_1\ (\mathtt{f}\,..(\mathtt{f}\,a_n\ \mathtt{x})..)$

- **Grundkonzepte: Leere Liste, Anhängen, Rekursion**

$$
\begin{array}{lcl}
[] & \equiv & \lambda\mathtt{f}.\lambda\mathtt{x}.\mathtt{x} \\
t::list & \equiv & \lambda\mathtt{f}.\lambda\mathtt{x}.\,\mathtt{f}\ t\ (list\ \mathtt{f}\ \mathtt{x}) \\
\mathsf{list\_ind}[b,h] & \equiv & \lambda\mathtt{list}.\ \mathtt{list}\ h\ b
\end{array}
$$

Rekursionsgleichungen der Listeninduktion

Sei $f \equiv$ **list_ind**$[b,h] \equiv \lambda\texttt{list.}\ \texttt{list}\ h\ b$

Basisfall:

$$
\begin{aligned}
f\ [] &\equiv (\lambda\texttt{list.}\ \texttt{list}\ h\ b)\ [] \\
&\longrightarrow [] \ h\ b \\
&\equiv (\lambda\texttt{f.}\lambda\texttt{x.x})\ h\ b \\
&\longrightarrow (\lambda\texttt{x.x})\ b \\
&\longrightarrow b
\end{aligned}
$$

Rekursionsfall:

$$
\begin{aligned}
f\ (t::list) &\equiv (\lambda\texttt{list.}\ \texttt{list}\ h\ b)\ t::list \\
&\longrightarrow t::list\ h\ b \\
&\equiv (\lambda\texttt{f.}\lambda\texttt{x.f}\ t\ (list\ \texttt{f}\ \texttt{x}))\ h\ b \\
&\longrightarrow (\lambda\texttt{x.}\ h\ t\ (list\ h\ \texttt{x}))\ b \\
&\longrightarrow h\ t\ (list\ h\ b)
\end{aligned}
$$

$$
\begin{aligned}
h\ t\ (f\ list) &\equiv h\ t\ ((\lambda\texttt{list.}\ \texttt{list}\ h\ b)\ list) \\
&\longrightarrow h\ t\ (list\ h\ b)
\end{aligned}
$$

Programmierung rekursiver Funktionen

- **Fixpunktkombinator (Rekursor)**
  - λ-Term R mit der Eigenschaft: $R\,t = t\,(R\,t)$ für beliebige Terme t
- **Liefert Programmschema für rekursive Funktionen**
  - Definiere Funktion F durch $F \equiv R\ (\lambda f\,.\,\lambda x\,.\,t[f,x])$,
    wobei im Programmkörper t sowohl f als auch x vorkommen können
    Dann gilt $\boldsymbol{F\,x} = (\lambda f\,.\,\lambda x\,.\,t[f,x])\ F\ x \overset{*}{\longrightarrow} \boldsymbol{t[F,x]}$
- **Y-Kombinator:** $\mathsf{Y} \equiv \lambda \mathtt{f}\,.\,(\lambda \mathtt{x}\,.\,\mathtt{f}\,(\mathtt{x}\,\mathtt{x}))\,(\lambda \mathtt{x}\,.\,\mathtt{f}\,(\mathtt{x}\,\mathtt{x}))$
  - **Y** ist Fixpunktkombinator

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

- **Rekursionsschema:** $\mathbf{letrec}\ f(x) = t \equiv \mathsf{Y}(\lambda f\,.\,\lambda x\,.\,t)$
  - Beschreibt Definition einer rekursiven Funktion f mit Argument x

Ausdruckskraft des λ-Kalküls

Der λ-Kalkül erfaßt alle berechenbaren Funktionen

- **λ-berechenbare Funktionen sind berechenbar**
  - Offensichtlich, da nur syntaktische Ersetzung
- **Alle berechenbaren Funktionen sind λ-berechenbar**
  - Beweis durch Vergleich mit anderem Berechenbarkeitsmodell
    - Turingmaschinen
    - Registermaschine
    - PASCAL-reduziert
    - μ-rekursive Funktionen
    - Logische Repräsentierbarkeit
    - Markov-Algorithmen

      $\vdots$

μ-rekursive Funktionen

1. **Nachfolgerfunktion s** mit $s(n) = n + 1$ für alle $n \in \mathbb{N}$
2. **Projektionsfunktionen pr^n_m** mit $pr^n_m(x_1, ..., x_n) = x_m$
3. **Konstantenfunktionen c^n_m** mit $c^n_m(x_1, ..., x_n) = m$
4. **Komposition $f \circ (g_1 ... g_n)$** der Funktionen $f, g_1 ... g_n$:
   $h = f \circ (g_1 ... g_n)$, wenn $h(\vec{x}) = f(\, g_1(\vec{x}), ..., g_n(\vec{x})\,)$
5. **primitive Rekursion $Pr[f, g]$** zweier Funktionen f und g:
   $h = Pr[f, g]$, wenn $h(\vec{x}, 0) = f(\vec{x})$, $h(\vec{x}, y + 1) = g(\vec{x}, y, h(\vec{x}, y))$
6. **Minimierung $\mu[f]$** einer Funktion f:

$$h = \mu[f], \text{ wenn } h(\vec{x}) = \begin{cases} min\{y \mid f(\vec{x}, y) = 0\} & \text{falls dies existiert und} \\ & \text{alle } f(\vec{x}, i), i<y \text{ definiert} \\ \text{undefiniert} & \text{sonst} \end{cases}$$

Alle berechenbaren Funktionen sind μ-rekursiv

Alle μ-rekursiven Funktione sind λ-berechenbar

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

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

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

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

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

Konsequenzen der grossen Ausdruckskraft

Alle nichttrivialen extensionalen Eigenschaften von λ-Termen sind unentscheidbar (Satz von Rice)

- **Terminiert die Berechnung einer Funktion f bei Eingabe x?**
- **Ist die durch einen λ-Term f beschriebene Funktion total?**
- **Gehört ein Wert y zum Wertebereich einer Funktion f?**
- **Berechnet die Funktion f bei Eingabe von x den Wert y?**
- **Verhalten sich zwei λ-berechenbare Funktionen gleich?**
  (Dies ist nicht einmal beweisbar)

λ-Kalkül – Grenzen

- **Vielseitige, ausdrucksstarke Sprache**
  - Klares Berechenbarkeitsmodell
  - Immer noch zu wenig Vorgaben für formales Schließen
  - Mengentheoretische Semantik extrem kompliziert (domain theory)
- **Kein Schließen über Datentypen möglich**
  - Wann gehört ein Objekt zu einem bestimmten Datentyp?
  - Welche Struktur hat der Datentyp eines Objekts?
  - Welche Eigenschaften hat ein Programm?
- **Erweiterung von Semantik / Inferenzsystem nötig**
  - Beschränke Freiheit der λ-Terme durch Präzisierung ihrer Eigenschaften
  - Wähle Datentypen als Beschreibung von Programmeigenschaften

Typentheorie