



tut07

PROGRAMMIERUNG

ÜBUNG 6: λ -KALKÜL (TEIL 2)

Eric Kunze

`eric.kunze@mailbox.tu-dresden.de`

INHALT

1. Funktionale Programmierung
 - 1.1 Einführung in Haskell: Listen
 - 1.2 Algebraische Datentypen
 - 1.3 Funktionen höherer Ordnung
 - 1.4 Typpolymorphie & Unifikation
 - 1.5 Beweis von Programmeigenschaften
 - 1.6 λ - **Kalkül**
2. Logikprogrammierung
3. Implementierung einer imperativen Programmiersprache
4. Verifikation von Programmeigenschaften
5. H_0 – ein einfacher Kern von Haskell

Der λ -Kalkül

DER λ -KALKÜL

Atome. x, y 

Abstraktion. $(\lambda x. t)$ $(f(x) = t, \text{ anonyme Funktion})$

Applikation. $(t_1 \ t_2)$ 
Einsetzung *pl2?*

Verabredungen:

- ▶ Applikation ist linksassoziativ:
 $((t_1 \ t_2) \ t_3) = t_1 \ t_2 \ t_3$ für alle $t_1, t_2, t_3 \in \lambda(\Sigma)$
- ▶ mehrfache Abstraktion:
 $(\lambda x_1. (\lambda x_2. (\lambda x_3. t))) = (\lambda x_1 x_2 x_3. t)$ für alle $t \in \lambda(\Sigma)$
- ▶ Applikation vor Abstraktion:

$$\begin{aligned} (\lambda x. xy) &= (\lambda x. (xy)) \\ &\neq ((\lambda x. x)y) \end{aligned}$$

α -KONVERSION & β -REDUKTION

β -Reduktion

Seien $s, t \in \lambda(\Sigma)$ gültige λ -Terme und es gilt

$$GV(t) \cap FV(s) = \emptyset.$$

$$(\lambda x. t) s \rightarrow_{\beta} t[x/s]$$

\uparrow \uparrow
 GV FV

$$\begin{aligned} f(x) &= 3x \\ f(2) &= 3 \cdot 2 \end{aligned}$$

α -Konversion

Sei $t \in \lambda(\Sigma)$ und $z \notin GV(t) \cup FV(t)$.

$$(\lambda x. t) \rightarrow_{\alpha} \lambda z. t[x/z]$$

3

CHURCH-NUMERALS

Dadurch, dass wir im Folgenden keine Symbole mehr zulassen (d.h. $\Sigma = \emptyset$), benötigen wir eine alternative Charakterisierung dieser. Zuerst beschäftigen uns die natürlichen Zahlen.

Darstellung der natürlichen Zahlen: Church-Numerals

$$\langle 0 \rangle = (\lambda xy. y)$$

$$\langle 1 \rangle = (\lambda xy. xy)$$


$$\langle 2 \rangle = (\lambda xy. x(xy))$$

\vdots


$$\langle n \rangle = (\lambda xy. \underbrace{x(x \dots (xy) \dots)}_n)$$

4

FIXPUNKTKOMBINATOR UND REKURSION

- Ein $t \in \Sigma(\lambda)$ heißt **geschlossener Term**, falls $FV(t) = \emptyset$.
Ein geschlossener Term heißt auch **Kombinator**.
- **Fixpunktkombinator.**
 $\langle Y \rangle = (\lambda z. (\lambda u. z(uu)) (\lambda u. z(uu))) \in \lambda(\emptyset)$ 
- Der Fixpunktkombinator ermöglicht Rekursion.
- weitere definierte λ -Terme (siehe Skript S. 198f.):

$$\begin{aligned} \langle true \rangle &= (\lambda xy. x) & \langle false \rangle &= (\lambda xy. y) \\ \langle succ \rangle &= (\lambda z. (\lambda xy. x(zxy))) & \langle pred \rangle \langle 0 \rangle &\Rightarrow^* \langle 0 \rangle \\ \langle succ \rangle \langle n \rangle &\Rightarrow^* \langle n+1 \rangle & \langle pred \rangle \langle n \rangle &\Rightarrow^* \langle n-1 \rangle \end{aligned}$$



$$\langle ite \rangle s s_1 s_2 \Rightarrow^* \begin{cases} s_1 & \text{wenn } s \Rightarrow^* \langle true \rangle \\ s_2 & \text{wenn } s \Rightarrow^* \langle false \rangle \end{cases}$$

