

Lambda-Kalkül



tut06_han...

PROGRAMMIERUNG

ÜBUNG 6: λ -KALKÜL (TEIL 1)

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

λ -KALKÜL

- ▶ weitere funktionale Programmiersprache
- ▶ Programme = λ -Terme ←
- ▶ Vorstellung: *anonyme* Funktionen

2

SYNTAX

Sei X eine Menge mit Variablen, Σ eine Menge mit Symbolen. Die gültigen λ -Terme sind *induktiv* definiert:

1. **Atome** (Variablen oder Symbole) sind gültige λ -Terme.
2. **Abstraktion:** Ist t ein gültiger λ -Term und $x \in X$ eine Variable, dann ist auch $(\lambda x.t)$ ein gültiger λ -Term.
3. **Applikation:** Sind t_1 und t_2 gültige λ -Terme, dann ist auch $(t_1 t_2)$ ein gültiger λ -Term.

Variable
↓
 $(\lambda x.x)(\lambda x.x * x)$
↑
Term

 $\lambda x. \underbrace{x * x}_{t}$

3

BEISPIELE (INFORMELL)

Vorstellung: Jeder λ -Term beschreibt eine anonyme Funktion.

- Abstraktion gibt das Argument an:

$$\lambda x.t \leftrightarrow f(x) = t$$

- Applikation beschreibt Funktionsanwendung (Einsetzen):

$$((\lambda x.t) 2) \leftrightarrow f(2)$$

$f(x) = t$

Beispiel:

$$\begin{aligned} \text{quadriere} &= \lambda x.x * x \rightarrow f(x) = x * x \\ ((\lambda x.x * x) 2) &= 2 * 2 = 4 \rightarrow f(2) = 2 * 2 = 4 \end{aligned}$$

4

VERABREDUNGEN

- Applikation ist linksassoziativ:

$$f : [\text{Int} \rightarrow (\text{Int}) \rightarrow \text{Int}]$$

$$f(t_1 t_2) t_3 = t_1 t_2 t_3$$

- mehrfache Abstraktion:

$$f(x_1)(x_2)(x_3) \quad (\lambda x_1.(\lambda x_2.(\lambda x_3.t))) = \lambda x_1 x_2 x_3. t \quad f(x_1, x_2, x_3)$$

- Applikation vor Abstraktion:

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

5

GEBUNDENE UND FREIE VORKOMMEN

Mengen $\text{FV}(t)$ und $\text{GV}(t)$ geben frei bzw. gebunden vorkommende Variablen von t an — induktive Definition

- Home**
- einzelne **Variablen** sind immer frei:
 $x \in X \Rightarrow \text{FV}(x) = \{x\}, \text{GV}(x) = \emptyset$
 - **Symbole** sind weder frei noch gebunden
 - } ► **Applikation:** Sei $t = (t_1 t_2)$. Dann
 $\Rightarrow \text{FV}(t) = \text{FV}(t_1) \cup \text{FV}(t_2), \text{GV}(t) = \text{GV}(t_1) \cup \text{GV}(t_2)$
 - } ► **Abstraktion:** $t = \lambda x.t'$
 $\Rightarrow \text{FV}(t) = \text{FV}(t') \setminus \{x\}, \text{GV}(t) = \text{GV}(t') \cup \{x\}$

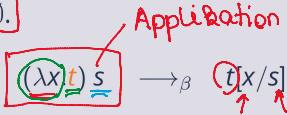
6

β – REDUKTION

β -Reduktion

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

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



- Bedeutung von $t[x/s]$: Ersetze jedes *freie* Vorkommen von x in t durch s .
- Erinnerung: Vorstellung der Applikation als „Einsetzen“ in Funktionen
- beachte: Abstraktion λx entfällt

Bsp.: Seien die Symbole gegeben durch $\Sigma = \{3, a\}$.

$$(\lambda x. +x3) (\lambda z.a) \xrightarrow{\beta} +(\lambda z.a)3$$

$\cancel{GV=\emptyset} \quad \cancel{FV=\emptyset}$

$$\begin{aligned} f(x) &= \underbrace{x^2}_t + 2 \Rightarrow f(2) = 2^2 + 2 \\ &= 6 \\ (\lambda x. (x^2 + 2))(2) & \quad s \end{aligned}$$

7

α – KONVERSION

- Was machen wir, wenn Voraussetzung $FV(t) \cap GV(t) = \emptyset$ für β -Reduktion nicht erfüllt ist?
- einfacher Ausweg: entsprechende Variablen umbenennen, sodass Bedingung erfüllt ist

α -Konversion

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

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

8

Bsp.: Seien die Symbole gegeben durch $\Sigma = \{3, a\}$.

$$(\lambda x. (\lambda y. +xy)) (\underbrace{y}_{FV=\{y\}}) \xrightarrow{\alpha} (\lambda x. (\lambda z. +xz)) (\underbrace{y}_{FV=\{y\}}) \xrightarrow{\beta} \dots$$

$\cancel{GV=\{y\}} \cap \cancel{GV=\{y\}} = \emptyset \neq \emptyset$
 \rightarrow keine β -Red.