Tutorium 06: λ-Kalkül

Paul Brinkmeier

25. November 2019

Tutorium Programmierparadigmen am KIT

Heutiges Programm

Programm

- Übungsblatt 5
- Church-Zahlen
- Altklausuraufgaben zum λ-Kalkül

Übungsblatt 5

2.1, 2.3 — AST: Datenstruktur

```
module AstType where


data Exp t
  = Var t
  | Const Integer
  | Add (Exp t) (Exp t)
  | Less (Exp t) (Exp t)
  | And (Exp t) (Exp t)
  | Not (Exp t)
  | If (Exp t) (Exp t) (Exp t)
```

- `t` ist Typvariable, um bspw. `Int`s als Namen zuzulassen
- Das kommt bspw. bei Compiler-Optimierungen zum Einsatz

Wiederholung

λ-Terme

Ein Term im λ-Kalkül hat eine der drei folgenden Formen:

Notation	**Besteht aus**	**Bezeichnung**
x	x : Variablenname	Variable
$\lambda p.b$	p : Variablenname b : λ-Term	Abstraktion
$f\ a$	f, a : λ-Terme	Funktionsanwendung

- „λ-Term“: rekursive Datenstruktur

Begriffe im λ-Kalkül

Begriff	**Formel**	**Bedeutung**
α-Äquivalenz	$t_1 \overset{\alpha}{=} t_2$	t_1, t_2 sind gleicher Struktur
η-Äquivalenz	$\lambda x.f\ x \overset{\eta}{=} f$	„Unterversorgung“
Freie Variablen	$fv(\lambda p.b) = b$	Menge der nicht durch λs gebundenen Variablen
Substitution	$(\lambda p.b)\,[b \to c] = \lambda p.c$	Ersetzung nicht-freier Variablen
Redex	$(\lambda p.b)\ t$	„Reducible expression“
β-Reduktion	$(\lambda p.b)\ t \Rightarrow b\,[p \to t]$	„Funktionsanwendung“

Church-Zahlen im λ-Kalkül

Peano-Axiome

$$
\begin{aligned}
c_0 &= ? \\
c_1 &= s(c_0) \\
c_2 &= s(s(c_0)) \\
c_3 &= s(s(s(c_0))) \\
c_8 &= s(s(s(s(s(s(s(s(c_0)))))))))
\end{aligned}
$$

1. Die 0 ist Teil der natürlichen Zahlen
2. Wenn n Teil der natürlichen Zahlen ist,
   ist auch $s(n) = n + 1$ Teil der natürlichen Zahlen

Church-Zahlen

- „Zahlen“ im λ-Kalkül werden durch Funktionen in Normalform dargestellt
- $n\ f\ x = f$ n-mal angewendet auf x
- Bspw. $(3\ g\ y) = g\ (g\ (g\ y)) = g^3\ y$
  Mit $3 = \lambda f.\lambda x.f\ (f\ (f\ x))$
- Schreibt eine λ-Funktion *succ*, die eine Church-Zahl nimmt und zu deren Nachfolger auswertet

Church-Zahlen

- „Zahlen“ im λ-Kalkül werden durch Funktionen in Normalform dargestellt
- $n\ f\ x = f$ n-mal angewendet auf x
- Bspw. $(3\ g\ y) = g\ (g\ (g\ y)) = g^3\ y$
  Mit $3 = \lambda f.\lambda x.f\ (f\ (f\ x))$
- Schreibt eine λ-Funktion *succ*, die eine Church-Zahl nimmt und zu deren Nachfolger auswertet
- Übertragt die Funktion in euren Haskell-Code vom letzten Mal und wertet *succ* c_0 durch wiederholtes Anwenden von `normalBeta` aus
- Vergleicht euer Ergebnis mit dem von Wavelength
  - //pp.ipd.kit.edu/lehre/misc/lambda-ide/Wavelength.html

Klausuraufgabe 1

Klausuraufgabe 2