Freie Universität Berlin

Der Lambda-Kalkül

Algorithmen und Programmierung I, WS 06/07

Vorlesungsreihe: 22.11.-29.11.06

Syntax

Wir gehen davon aus, dass V eine abzählbare unendliche Menge von Variablen ist.

Die Menge Expr der λ-Ausdrücke wird durch folgende Syntaxregel definiert:

Expr ->	V	*Atome*
|	(Expr Expr)	*Applikationen*
|	(λ x. Expr)	*Abstraktionen*

Konventionen zur Einsparung von Klammern:

- Die Applikation bindet linksassoziativ
- Eine Abstraktion erstreckt sich so weit nach rechts wie möglich

Semantik

λ-Ausdrücke repräsentieren sowohl Funktionen als auch Argumente. Die einzige Auswertungsregel ist die *β–Reduktio*n, die das Einsetzen aktueller Argumente für formale Parameter formalisiert:

$$(\lambda x.\ F)\ A \ \rightarrow_{\beta}\ \$^{x}_{A}\ F, \quad \text{falls } Fr(A) \cap Geb(F) = \emptyset$$

Daneben gibt es die *α–Konversion*, zur Umbenennung gebundener Variablen (formaler Parameter):

$$(\lambda x.\ F) \rightarrow_{\alpha} (\lambda y.\ \$^{x}_{y}\ F)\ , \quad \text{falls y nicht in F vorkommt}$$

Zwei Ausdrücke A und B sind *semantisch gleich* (geschrieben A = B), wenn sie aus demselben Ausdruck durch endlich viele β–Reduktionen und α–Konversionen hervorgehen.

Normalform

Definition: Ein λ-Ausdruck ist in *Normalform*, wenn er keinen ß-Redex enthält.

Satz: Wenn ein λ-Ausdruck eine Normalform besitzt, dann terminiert die sukzessive Reduktion des jeweils am weitesten links stehenden β–Redexes in dieser Normalform.

Normalformen stellen Werte (Daten) dar. Eine Normalform, die keine freien Variablen enthält heißt *Kombinator*.

Natürliche Zahlen und Wahrheitswerte lassen sich durch Normalformen darstellen, d.h. im Lambda-Kalkül kodieren, z.B. nach Church:

$\underline{0} := \lambda f.\ \lambda x.\ x$

$\underline{1} := \lambda f.\ \lambda x.\ f\ x$

$\underline{2} := \lambda f.\ \lambda x.\ f\ (f\ x)$

. . .

$\underline{true} := \lambda x.\ \lambda y.\ x$

$\underline{false} := \lambda x.\ \lambda y.\ y$

Lambda-Kalkül als Programmiersprache

Definition. Eine k-stellige Funktion f über den natürlichen Zahlen heißt λ-definierbar, wenn es einen λ-Ausdruck F gibt, so dass für alle $n_1,\ldots,n_k$ gilt:

$f(n_1,\ldots,n_k) = n$ gdw. $F\ \underline{n_1}\ \ldots\ \underline{n_k} = \underline{n}$.

Satz: Die Nachfolgerfunktion ist λ-definierbar mit

succ := λ n. λ f. λ x. n f (f x)

Beweis: succ n = (λ n. λ f. λ x. n f (f x)) n ->

λ f. λ x. n f (f x)) = λ f. λ x. (λ f. λ x. f x) f (f x)) ->

λ f. λ x. (λ g. λ y. g y) f (f x)) $\to^2$ λ f. λ x. f^{n+1} x = n+1

Satz: Der Test auf Null ist λ-definierbar mit

isZero := λ n. n (λ y. false) true

Beweis: isZero 0 = (λ n. n (λ y. false) true) 0 ->
(λ f. λ x. x) (λ y. false) true -> true

isZero n = (λ n. n (λ y . false) true) (λ f. λ x. f^n x) ->
(λ f. λ x. f^n x) (λ y . false) true -> false , da n ≥ 1

Bedingte Ausdrücke können mit dem λ-Ausdruck

cond := λ b. λ x. λ y. b x y

gebildet werden, wobei folgendes gilt:

cond true E_1 E_2 ->* E_1
cond false E_1 E_2 ->* E_2

Datenstrukturen

Datenstrukturen können durch geeignete Kombinatoren dargestellt werden.

Beispiel Tupel: Mit dem Kombinator

pair := λ x. λ y. λ c. c x y

und den λ-Funktionen

fst := λ p. p true
snd := λ p. p false gilt:

fst (pair A B) = A und
snd (pair A B) = B für alle λ-Ausdrücke A, B

Die Vorgängerfunktion

Satz: Die Vorgängerfunktion v ist λ-definierbar.

Idee: Definiere zunächst v‘ mit v‘ n = pair n-1 n
und v“ mit v“ (pair n-1 n) = pair n n+1 durch

v‘ := λ n. n v“ (pair 0 0)
v“:= λ p. cond (isZero (snd p)) (pair 0 1)
(pair (succ (fst p)) (succ (snd p)))
v := λ n. fst (v‘ n)

Beispiel: v 2 = λ n. fst (v‘ n) 2 -> fst (v‘ 2) ->
fst ((λ n. n v“ (pair 0 0)) 2) -> fst (2 v“ (pair 0 0)) ->
fst (v“ (v“ (pair 0 0))) -> fst (v“ (pair 0 1)) ->
fst (pair 1 2) -> 1

Rekursion

Fixpunktsatz: Zu jedem λ-Ausdruck F gibt es einen Fixpunkt A, so dass F A = A gilt.

Beweis: Man gewinnt den Fixpunkt eines λ-Ausdrucks F durch Anwendung des Y-Operators auf F, wobei

$$Y := \lambda f.\ (\lambda x.\ f\ (x\ x))\ (\lambda x.\ f\ (x\ x))$$

Es gilt: Y F -> (λ x. F (x x)) (λ x. F (x x)) ->
F (λ x. F (x x)) (λ x. F (x x)) <- F (Y F)

Damit ist F (Y F) = (Y F) gezeigt.

Wenn $f\ x = E$ eine rekursive Definition ist, dann gilt:

$f = \lambda\ x.\ E$ und nach einer β–Reduktion rückwärts auch

$f = (\lambda\ f.\ \lambda\ x.\ E)\ f$.

Diese Gleichung lässt sich wie folgt lesen:

f ist Fixpunkt des Ausdrucks $(\lambda\ f.\ \lambda\ x.\ E)$.

Mit dieser Vorüberlegung lässt sich f nun im λ-Kalkül ohne Verwendung der Rekursion durch den Ausdruck

$$f := Y\ (\lambda\ f.\ \lambda\ x.\ E)$$

definieren. Damit ist gezeigt, dass alle berechenbaren Funktionen λ-definierbar sind.