Vorlesung

Theoretische Informatik II

Bernhard Beckert

Institut für Informatik

UNIVERSITÄT KOBLENZ · LANDAU

Wintersemester 2007/2008

Dank

Diese Vorlesungsmaterialien basieren zum Teil auf den Folien zu den Vorlesungen von

Katrin Erk (gehalten an der Universität Koblenz-Landau)

Jürgen Dix (gehalten an der TU Clausthal)

Christoph Kreitz (gehalten an der Universität Potsdam)

Ihnen gilt mein herzlicher Dank.

– Bernhard Beckert, Oktober 2007

Die Idee

Zweck

- Formale Definition von Funktionen
- Untersuchung und Anwendung von Funktionsdefinitionen
- Berechnungsmodell
- Grundlage funktionaler Programmiersprachen
- Einfach und doch mächtig

Grundlegende Eigenschaften

- Funktion identifiziert mit λ-Ausdrücken, die sie beschreiben
- Funktionen angewendet auf Funktionen
- Nichts außer Funktionen (keine anderen Datentypen)

Entwickelt von Alonzo Church und Stephen Kleene in den 1930er Jahren

Syntax

Definition 14.1 (λ-Ausdruck)

Variable Jede Variable

$$x$$

ist ein λ-Ausdruck

Abstraktion Ist x eine Variable und e ein λ-Ausdruck, dann ist

$$\lambda x.e$$

ein λ-Ausdruck

Anwendung Sind e_1, e_2 λ-Ausdrücke, dann ist

$$e_1\ e_2$$

ein λ-Ausdruck

Syntax

Assoziativität und Bindungsstärke

- Anwendungen sind linksassoziativ:

$$e_1\ e_2\ e_3\ =\ (e_1\ e_2)\ e_3$$

- Anwendung bindet stärker als Abstraktion:

$$\lambda x.e_1\ e_2\ =\ \lambda x.(e_1\ e_2)$$

WICHTIG!
Dies muss man wissen, um λ-Ausdrücke verstehen zu können!

Beispiele

Einige wichtige λ-Ausdrücke

Indentitätsfunktion:

$$I = \lambda x.x$$

Selbstanwendung:

$$D = \lambda x.x\ x$$

Auslassen des 2. Argumentes:

$$K = \lambda x.\lambda y.x$$

α-Konversion

Definition 14.2

- Ein Vorkommen einer Variablen x ist gebunden, wenn es im Skopus von λx ist.
- Andernfalls ist es frei.

α-Konversion

λ-Ausdrücke, die gleich sind bis auf Umbenennen gebundener Variablen

- werden identifiziert
- beschreiben dieselbe Funktion

Substitution

Definition 14.3

Das Ergebnis der Substitution von x in e durch e'

in Zeichen: $[e'/x]\,e$

entsteht dadurch, dass

1. gebundene Variablen so umbenannt werden, dass sie nur einmal vorkommen
2. x in e syntaktisch durch e' ersetzt wird

β-Reduktion

β-Reduktion

Der λ-Ausdruck

$$(\lambda x.e)\ e'$$

kann durch β-Reduktion zu dem Ausdruck

$$[e'/x]\,e$$

reduziert werden

In Zeichen: $(\lambda x.e)\ e' \ \rightarrow_\beta \ [e'/x]\,e$

Wenn $e \rightarrow_\beta^* e'$, dann

- werden e, e' identifiziert
- beschreiben dieselbe Funktion

β-Reduktion

Beispiel 14.4

$$(\lambda f.f\ (\lambda x.x))\ (\lambda x.x) \ \rightarrow_\beta^* \ (\lambda y.y)$$

Beispiel 14.5

Der Ausdruck

$$(\lambda x.x\ x)\ (\lambda x.x\ x)$$

kann nicht weiter reduziert werden

Eigenschaften der Reduktion

Eigenschaften der Reduktion mittels α- und β-Reduktion

- β-Reduktion terminiert nicht immer
- β-Reduktion ist nicht deterministisch
- Jedoch: $\rightarrow_\beta^*$ ist konfluent (hat die Church-Rosser-Eigenschaft): Gilt

  $$e \rightarrow_\beta^* e_1 \quad \text{und} \quad e \rightarrow_\beta^* e_2$$

  dann gibt es ein e' mit

  $$e_1 \rightarrow_\beta^* e' \quad \text{und} \quad e_2 \rightarrow_\beta^* e'$$

- Darum: Normalformen sind eindeutig

Mächtigkeit des λ-Kalküls

Ausdrucksstärke

Im λ-Kalkül kann man ausdrücken:

- Datentypen wie Integer, Boolean, Listen, Bäume, ...
- Verzweigung
- Rekursion

Turing-Mächtigkeit

Der λ-Kalkül kann Turing-Maschinen simulieren

Unentscheidbarkeit

Es ist unentscheidbar, ob für beliebig gegebene e, e' gilt, dass

$$e \ \rightarrow_\beta^* \ e'$$

Darstellung von Aussagenlogik im λ-Kalkül

Wahrheitswerte

$$\begin{aligned} \mathit{true} &=_{\text{def}} \quad \lambda x.\lambda y.\, x \\ \mathit{false} &=_{\text{def}} \quad \lambda x.\lambda y.\, y \end{aligned}$$

if-then-else

$$\mathit{if}\ C\ \mathit{then}\ U\ \mathit{else}\ V \ =_{\text{def}} \quad ???\, C\ U\ V$$

Darstellung von Aussagenlogik im λ-Kalkül

Mit true, false, if-then-else alles darstellbar

$$\begin{aligned} \neg x &\equiv \mathit{if}\ x\ \mathit{then}\ \mathit{false}\ \mathit{else}\ \mathit{true} \\ x \wedge y &\equiv \mathit{if}\ x\ \mathit{then}\ y\ \mathit{else}\ \mathit{false} \\ x \vee y &\equiv \mathit{if}\ x\ \mathit{then}\ \mathit{true}\ \mathit{else}\ y \end{aligned}$$

Damit

$$\begin{aligned} \neg x &\equiv x\ \mathit{false}\ \mathit{true} \\ x \wedge y &\equiv x\ y\ \mathit{false} \\ x \vee y &\equiv x\ \mathit{true}\ y \end{aligned}$$

Darstellung Paaren im λ-Kalkül

Paare

$$\begin{aligned} \mathit{mkpair}(x,y) &=_{\text{def}} \quad \lambda b.\, b\ x\ y \\ \mathit{fst}(p) &=_{\text{def}} \quad p\ \mathit{true} \\ \mathit{snd}(p) &=_{\text{def}} \quad p\ \mathit{false} \end{aligned}$$

Ähnlich für

- Tupel
- Listen
- etc.

Darstellung natürlicher Zahlen im λ-Kalkül

Natürliche Zahlen

$$\begin{aligned} 0 &=_{\text{def}} \quad \lambda f.\lambda x.\, x \\ 1 &=_{\text{def}} \quad \lambda f.\lambda x.\, f\ x \\ 2 &=_{\text{def}} \quad \lambda f.\lambda x.\, f\,(f\ x) \\ 3 &=_{\text{def}} \quad \lambda f.\lambda x.\, f(\,f\,(f\ x)) \\ &\vdots \\ n &=_{\text{def}} \quad \lambda f.\lambda x.\, \underbrace{f(\ldots(f}_{n\ \text{mal}}\ x)\ldots) \end{aligned}$$

Darstellung natürlicher Zahlen im λ-Kalkül

Operationen auf natürlichen Zahlen

$$
\begin{aligned}
succ(n) &=_{\text{def}} \quad \lambda g.\lambda y.\, n\ g\ (g\ y) \\
+(n,m) &=_{\text{def}} \quad n\ succ\ m \\
*(n,m) &=_{\text{def}} \quad n\ (m\ succ)\ 0 \\
iszero(n) &=_{\text{def}} \quad n\ (\lambda b.false)\ true
\end{aligned}
$$

Rekursion im λ-Kalkül

Der Y-Operator

$$Y =_{\text{def}} \quad \lambda F.\,(\lambda y.\,F(y\ y))\ (\lambda x.\,F(xx))$$

Y ist Fixpunkt-Operator

$$f(Y\,f) = Y\,f$$

für alle f

Rekursion im λ-Kalkül

Beispiel

$$fak\ n =_{\text{def}} \quad (Y\ F)\ n$$

mit

$$F =_{\text{def}} \quad \lambda f.\lambda n.\ if\ iszero(n)\ then\ 1\ else\ n*(f(n-1))$$

Allgemein

Der Y-Operator erlaubt es, beliebige μ-rekursive Funktionen im λ-Kalkül zu definieren.