

Vorlesung

Theoretische Informatik II

Bernhard Beckert

Institut für Informatik



Wintersemester 2007/2008

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*

Teil IV

Der λ -Kalkül

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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!

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!

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!

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!

Einige wichtige λ -Ausdrücke

Identitätsfunktion:

$$I = \lambda x. x$$

Selbstanwendung:

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

Auslassen des 2. Argumentes:

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

Einige wichtige λ -Ausdrücke

Identitätsfunktion:

$$I = \lambda x.x$$

Selbstanwendung:

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

Auslassen des 2. Argumentes:

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

Einige wichtige λ -Ausdrücke

Identitätsfunktion:

$$I = \lambda x.x$$

Selbstanwendung:

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

Auslassen des 2. Argumentes:

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

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

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

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

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

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

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

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

Der λ -Ausdruck

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

kann durch β -Reduktion zu dem Ausdruck

$$[e'/x] e$$

reduziert werden

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

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

- werden e, e' identifiziert
- beschreiben dieselbe Funktion

β -Reduktion

Der λ -Ausdruck

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

kann durch β -Reduktion zu dem Ausdruck

$$[e'/x] e$$

reduziert werden

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

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

- werden e, e' identifiziert
- beschreiben dieselbe Funktion

β -Reduktion

Der λ -Ausdruck

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

kann durch β -Reduktion zu dem Ausdruck

$$[e'/x] e$$

reduziert werden

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

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

- werden e, e' identifiziert
- beschreiben dieselbe Funktion

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

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 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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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