

# Der untypisierte Lambda-Kalkül als Programmiersprache

Benjamin Peherstorfer

November 29, 2006

betreut von  
Prof. Tobias Nipkow

# Einführung

- ▶ In den 1930er Jahren von Alonzo Church und Stephen Kleene eingeführt
- ▶ Der Lambda-Kalkül ist eine formale Sprache/Berechnungsmodell
- ▶ 3 Bausteine
- ▶ Nur anonyme Funktionen
- ▶ Ebenso mächtig wie Turingmaschine

Church



Kleene



# Definition

## Definition:

$$t ::= x \mid (\lambda x.t) \mid (t_1 t_2)$$

Dabei gilt:

- ▶  $a, b, c, x, y, z$ , Variablen
- ▶  $\lambda x.t$ , Abstraktion entspricht  $f(x) = t$
- ▶  $t_1 t_2$ , Applikation entspricht Funktionsaufruf

## Beispiele

- ▶  $((t_1 t_2) t_3)$
- ▶  $(\lambda x.x)$ , Identität

## Effiziente Klammerung

Um Klammerung zu minimieren, gelten folgende Konventionen:

- ▶ Applikation ist linksassoziativ:  
 $t_1 t_2 t_3 \equiv ((t_1 t_2) t_3)$
- ▶ Abstraktion ist rechtsassoziativ, innere  $\lambda$ 's werden vereint:  
 $\lambda xyz.zyx \equiv \lambda x.(\lambda y.(\lambda z.zyx))$
- ▶ Äußere Klammern entfallen.

### Beispiele

$$\begin{aligned} f (\lambda x.x) g &\equiv ((f (\lambda x.x)) g) \\ \lambda x.(\lambda y.yz)x &\equiv (\lambda x.((\lambda y.(yz))x)) \end{aligned}$$

## Freie und gebundene Variablen

**Definition:** Eine Variable  $x$  wird durch eine Abstraktion  $\lambda x.t$  *gebunden*, wenn sie im Term  $t$  dieser Abstraktion vorkommt. Andernfalls ist sie *frei*.

Die Menge  $FV(t)$  beinhaltet die freien Variablen des Terms  $t$ .

Welche Variablen sind **frei**?

- ▶  $\lambda x.(xy(\lambda z.xz))$   
 $\lambda x.(x**y**(\lambda z.xz))$
- ▶  $(\lambda x.xy)(\lambda z.w(\lambda w.wzyx))$   
 $(\lambda x.x**y**)(\lambda z.**w**(\lambda w.wz**y**x))$
- ▶  $x\lambda z.(x(\lambda w.wzy))$   
**x** $\lambda z.(x(\lambda w.wz**y**))$

## Substitution

- ▶ Ist Grundlage für Auswertung von Termen
- ▶ Schreibweise:  $s[t/x]$ ,  $s[x := t]$  oder  $s[x \leftarrow T]$
- ▶ Ersetze in  $s$  alle freien  $x$  durch den Term  $t$
- ▶ Beachte:
  - ▶ nur freie Variablen ersetzen  
 $(\lambda x.xy)[t/x] \not\rightarrow (\lambda x.ty)$
  - ▶ keine neuen Bindungen  
 $(\lambda x.xy)[x/y] \not\rightarrow (\lambda x.xx)$
- ▶ Durch 6 Regeln definiert. . . .

## Definition der Substitution

Regeln (1)-(6):

$$(1) \quad x[t/x] = t$$

$$(2) \quad a[t/x] = a \quad \text{falls } a \neq x$$

$$(3) \quad (s_1 s_2)[t/x] = (s_1[t/x])(s_2[t/x])$$

$$(4) \quad (\lambda x.s)[t/x] = \lambda x.s$$

$$(5) \quad (\lambda y.s)[t/x] = \lambda y.(s[t/x]) \quad \text{falls } x \neq y \wedge y \notin FV(t)$$

$$(6) \quad (\lambda y.s)[t/x] = \lambda z.(s[z/y][t/x]) \quad \begin{array}{l} \text{falls } x \neq y \wedge \\ z \notin FV(t) \cup FV(s) \end{array}$$

Intuitiv:

Keine Bindungen erzeugen oder zerstören.

## $\alpha$ -Konversion

**Definition:**  $\lambda x.s \rightarrow_{\alpha} \lambda y.(s[y/x])$  falls  $y \notin FV(s)$ .

- ▶ Erlaubt Umbenennung von gebundenen Variablen wenn ungefährlich.
- ▶ Konvention: Ab jetzt werden gebundene Variablen umbenannt, sollten sie mit freien kollidieren!

Das vereinfacht die Substitution erheblich:

$$(\lambda x.s)[t/y] = \lambda x.(s[t/y])$$

statt Regeln (4)-(6)

## Vereinfachte Substitution

### Beispiel 1

$$\lambda x. xy \rightarrow_{\alpha} \lambda z. zy$$

### Beispiel 2

$$\lambda z. (\lambda x. xy)z \rightarrow_{\alpha} \lambda z. (\lambda z. zy)z \xrightarrow{\alpha^*} \lambda x. (\lambda x. xy)x \not\rightarrow_{\alpha} \not\rightarrow_{\alpha} \lambda y. (\lambda x. xy)y$$

### Beispiel 3

$$(\lambda zy. xx)[y/x] \rightarrow_n \lambda z. ((\lambda y. xx)[y/x]) \rightarrow_{\alpha} (\lambda z. ((\lambda y_1. xx)[y/x])) \rightarrow \lambda zy_1. yy$$

## $\beta$ -Reduktion

**Definition:**  $(\underbrace{\lambda x. s}_\text{Redex})t \rightarrow_\beta s[t/x]$

Lambda-Terme werden durch die  $\beta$ -Reduktion ausgewertet

### Beispiel 1

$$(\lambda x. (x(\lambda x. xx)))z \rightarrow_\beta z(\lambda x. xx)$$

### Beispiel 2

$$(\lambda xy. xyx)y \rightarrow_\alpha (\lambda xy_1. xy_1x)y \rightarrow_\beta \lambda y_1. yy_1y$$

## $\beta$ -Normalform

**Definition:** Der Term  $t_1$  ist in  $\beta$ -Normalform wenn kein Term  $t_2$  mit  $t_1 \rightarrow_{\beta} t_2$  existiert.

- ▶ Nicht jeder Term besitzt eine  $\beta$ -Normalform:

$$(\lambda x.xx)(\lambda x.xx) \rightarrow_{\beta} (\lambda x.xx)(\lambda x.xx) \rightarrow_{\beta} \dots$$

- ▶ Es können mehrere Wege zur  $\beta$ -Normalform führen:

$$(\lambda x.x)((\lambda y.y)x) \rightarrow_{\beta} ((\lambda y.y)x) \rightarrow_{\beta} x$$

$$(\lambda x.x)((\lambda y.y)x) \rightarrow_{\beta} (\lambda x.x)x \rightarrow_{\beta} x$$

- ▶ Die  $\beta$ -Normalform ist eindeutig:

Konsequenz: Man kann  $\beta$ -Reduktion für Berechnungen verwenden. Der Lambda-Kalkül ist als Berechnungsmodell geeignet.

# Wiederholung

- ▶ 3 Arten von Termen
- ▶ Nur anonyme Funktionen
- ▶ Unterscheidung: freie und gebunden Variablen
- ▶ Substitution ist Grundlage für Auswertung
- ▶  $\alpha$ -Konversion: Namen gebundener Variablen bedeutungslos
- ▶  $\beta$ -Reduktion entspricht Auswertung einer Funktion
- ▶  $\beta$ -Normalform entspricht Ergebnis einer Funktion

# Programmieren im Lambda-Kalkül

- ▶ Fallunterscheidung
- ▶ Paare und Listen
- ▶ Zahlen
- ▶ Rekursive Funktionen(!)

# Fallunterscheidung

## Überlegung:

- ▶ Entscheidung trifft der Wahrheits-Term
- ▶ charakteristische Gleichung:
  - ▶  $\text{true } v \ w = v$
  - ▶  $\text{false } v \ w = w$

## Definition:

- ▶  $\text{true} \equiv \lambda x. \lambda y. x$
- ▶  $\text{false} \equiv \lambda x. \lambda y. y$
- ▶  $\text{test} \equiv \lambda l m n. l m n$

## Beispiele

$$\text{test true } a \ b \rightarrow_{\beta}^* a$$
$$(\lambda x y. x) a \ b \rightarrow_{\beta} a$$

$$\text{test false } a \ b \rightarrow_{\beta}^* b$$
$$(\lambda x y. y) a \ b \rightarrow_{\beta} b$$

# Paare

## charakt. Gleichungen:

- ▶  $\text{fst}(\text{pair } a \ b) = a$
- ▶  $\text{snd}(\text{pair } a \ b) = b$

## Definition:

- ▶  $\text{pair} \equiv \lambda xyz.zxy$
- ▶  $\text{fst} \equiv \lambda p.p \ \text{true}$
- ▶  $\text{snd} \equiv \lambda p.p \ \text{false}$

## Listen:

- ▶ sind verschachtelte Paare
- ▶ Wo ist das Ende einer Liste?
- ▶ Wie erkennt man es?

# Church Numerale

**Definition:**  $n \hat{=} \lambda sz. s^n z$

Bedeutet:

$$0 \equiv \lambda sz. z$$

$$1 \equiv \lambda sz. sz$$

$$2 \equiv \lambda sz. s(sz)$$

$$\vdots \quad \vdots \quad \vdots$$

$$n \equiv \lambda sz. \underbrace{s(s \dots (s z) \dots)}_{n\text{-mal}}$$

Interessante Funktionen:

- ▶ Iteration
- ▶ Nachfolger-Funktion
- ▶ Vorgänger-Funktion
- ▶ IstNull?-Funktion
- ▶ Add, Sub, Mul, Exp

# Iteration

- ▶ Durch Church Numerale sind Iterationen möglich
- ▶  $n f a \equiv (\lambda s z. \underbrace{s(\dots(s z)\dots)}_{n\text{-mal}}) f a = \underbrace{f(f \dots (f(a)\dots))}_{n\text{-mal}}$
- ▶ Anfangszustand ist  $a$
- ▶ Zustandsänderungsfunktion ist  $f$
- ▶ Anzahl der Iterationen durch  $n$  bestimmt

## Beispiel

$\text{add } n m \equiv n \text{ succ } m$

$\text{sub } n m \equiv m \text{ pred } n$

## Interessante Funktionen

- ▶ Die succ-Funktion:  $\text{succ } n = n + 1$ .  
 $\text{succ} \equiv \lambda nsz.s(\text{nsz})$ , stellt  $s$  voran.
- ▶ Die iszero-Funktion:  $\text{iszero } 0 = \text{true}$  und  $\text{iszero } n = \text{false}$ .  
 $\text{iszero} \equiv \lambda n.n(\lambda y.\text{false})\text{true}$ , Schleife
- ▶ Die pred-Funktion:  $\text{pred } n = n - 1$  und  $\text{pred } 0 = 0$ .  
 $\text{pred} \equiv (\lambda n.n(\lambda p.(\text{pair}(p\ \text{false})(\text{succ}(p\ \text{false})))))(\text{pair } 0\ 0))\ \text{true}$ 
  - ▶ Benutze Paare der Form  $(n-1, n)$
  - ▶ Nach  $n$ -maliger Iteration: Vorgänger an erster Stelle

# Rekursive Funktionen

**Definition:** Rekursive Funktionen rufen sich im Funktionsrumpf selbst wieder auf.

## Problemstellung:

- ▶ Nur anonyme Funktionen beschreibbar
- ▶ Konstrukte wie  $f = \lambda xy \dots f \dots$  sind sinnlos
- ▶ Lösung: Darstellung rekursiver Funktionen durch nicht-rekursive.

## Fixpunkte

**Definition:** Term  $t$  ist Fixpunkt von  $s$ , falls  $s t = t$ .

Jeder Term besitzt einen Fixpunkt!

**Definition:** Ein Fixpunktoperator  $Y$  konstruiert für jeden beliebigen Term  $s$  einen Fixpunkt:  $Y s = s (Y s)$ .

A. Church und A. Turing haben jeweils einen Fixpunktoperator im reinen Lambda-Kalkül definiert.

# Churchscher Fixpunktoperator

**Definition** Es sei  $V_f \equiv \lambda x.f(xx)$  und  $Y \equiv \lambda f.V_f V_f$ , dann ist  $Y$  ein Fixpunktoperator.

## Beweis

$$\begin{aligned}
 Y t &\equiv (\lambda f.((\lambda x.f(xx))(\lambda x.f(xx))))t \\
 &\rightarrow_{\beta} (\lambda x.t(xx))(\lambda x.t(xx)) \\
 &\rightarrow_{\beta} t(\lambda x.t(xx))(\lambda x.t(xx)) \\
 &\leftarrow_{\beta} t((\lambda t.((\lambda x.t(xx))(\lambda x.t(xx))))t) \\
 &\equiv t(Y t)
 \end{aligned}$$

Grundlage ist Selbstapplikation!

## Darstellung rek. Funktionen

### Lösung:

- ▶ Geg.: rek. Funktion  $f = \lambda x.s$  und Term  $\text{fix}$  mit  $\text{fix } t = t (\text{fix } t)$
- ▶ Funktion  $f$  soll sich an der Stelle  $f$  in  $s$  selbst substituieren
- ▶ Man bindet das freie  $f$ :  $F \equiv \lambda f x.s$
- ▶ Ziel:  $\lambda f x.s = (\lambda f x.s)(\lambda f x.s) = (\lambda f x.s)((\lambda f x.s)(\lambda f x.s))$  oder anders  $F = F F = F (F F) = \dots$
- ▶ Term  $\text{fix}$ :  $\text{fix } F = F(\text{fix } F) = F(F(\text{fix } F)) = \dots$
- ▶ Der Term  $\text{fix } F$  entspricht somit der rekursiven Funktion

## Fibonacci-Folge

```
fib = λn.  
  (test (iszero n) 0 (  
    test (iszero pred n) 1 (  
      add (fib pred n)  
      (fib (pred (pred n))))  
    )  
  )
```

```
int fib(int n) {  
  if(n == 0)  
    return 0;  
  else {  
    if((n-1) == 0)  
      return 1;  
    else  
      return fib(n-1)  
      + fib(n-2);  
  }  
}
```

## Beispiel

- ▶ In nicht-rekursive Darstellung bringen:  
$$\text{fib}' = \lambda fn. (\text{test} (\text{iszero } n) 0 (\text{test} (\text{iszero } \text{pred } n) 1 (\text{add } (f \text{ pred } n) (f (\text{pred} (\text{pred } n))))))$$
- ▶ Es gilt  $\text{fib} = \text{fix fib}'$  und damit ...

## Beispiel

$$\begin{aligned}
 & \text{fib } 2 = \\
 = & \text{fix fib}' \ 2 \\
 \rightarrow_{\beta}^* & \text{fib}' (\text{fix fib}') \ 2 \\
 \rightarrow_{\beta}^* & (\text{test}(\text{iszero } 2)0(\text{test}(\text{iszero } 1)1(\text{add}((\text{fix fib}')1)(\text{fix fib}')0))) \\
 \rightarrow_{\beta}^* & (\text{add}(\text{fib}'(\text{fix fib}')1)(\text{fib}'(\text{fix fib}')0)) \\
 \rightarrow_{\beta}^* & (\text{add} (\text{test}(\text{iszero } 1)0(\text{test}(\text{iszero } 0)1(\text{add}((\text{fix fib}')0)((\text{fix fib}')0)))) \\
 & ((\text{test}(\text{iszero } 0)0(\text{test}(\text{iszero } 0)1(\text{add}((\text{fix fib}')0)((\text{fix fib}')0)))))) \\
 \rightarrow_{\beta}^* & (\text{add } 1 \ 0) \\
 \rightarrow_{\beta}^* & 1
 \end{aligned}$$