Karlsruher Institut für Technologie

Einführung in das λ-Kalkül

Max Wagner

IPD Snelting

Syntax

Abstrakte Syntax:

```
<term> ::= <abs> | <app> | <var>
<abs>  ::= λ <var>. <term>
<app>  ::= <term> <term>
<var>  ::= "a" | "b" | "c" | ...
```

Klammerung ist wichtig!
App implizit linksgeklammert, Abs “greedy”.

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

$$\lambda x.\ x \qquad \lambda s.\ \lambda z.\ s\ z \qquad \lambda s.\ \lambda z.\ s\ (s\ z) \qquad \lambda s.\ \lambda z.\ s\ s\ z$$

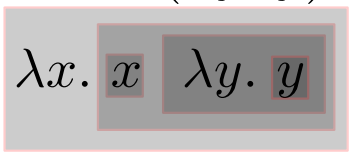
Syntax

Abstrakte Syntax:

```
<term> ::= <abs> | <app> | <var>
<abs>  ::= λ <var>. <term>
<app>  ::= <term> <term>
<var>  ::= "a" | "b" | "c" | ...
```

Klammerung ist wichtig!
App implizit linksgeklammert, Abs “greedy”.

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



$\lambda x.\ x$ $\lambda s.\ \lambda z.\ s\ z$ $\lambda s.\ \lambda z.\ s\ (s\ z)$ $\lambda s.\ \lambda z.\ s\ s\ z$

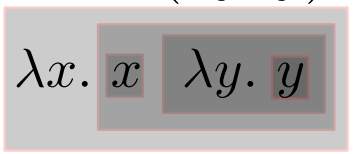
Syntax

Abstrakte Syntax:

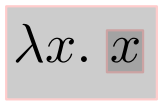
```
<term> ::= <abs> | <app> | <var>
<abs>  ::= λ <var>. <term>
<app>  ::= <term> <term>
<var>  ::= "a" | "b" | "c" | ...
```

Klammerung ist wichtig!
App implizit linksgeklammert, Abs “greedy”.

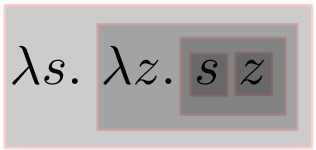
$\lambda x.\ x\ (\lambda y.\ y)$



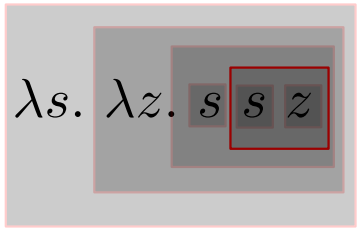
$\lambda x.\ x$



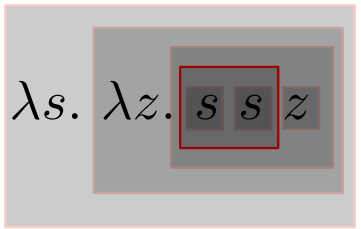
$\lambda s.\ \lambda z.\ s\ z$



$\lambda s.\ \lambda z.\ s\ (s\ z)$

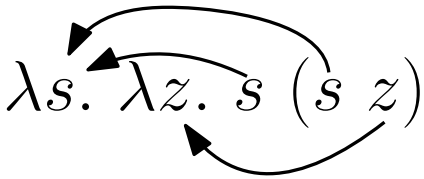


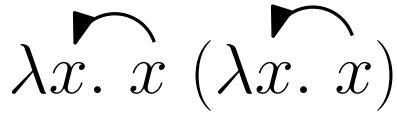
$\lambda s.\ \lambda z.\ s\ s\ z$



Variablenbindung

Unterscheide **freie** und **gebundene** Variablen.





β-Reduktion

Entspricht schrittweiser Funktionsauswertung.
Nur auf sog. **Reducible Expressions** (Redexe) andwendbar.

Definition (Redex)

Redexe sind Teilausdrücke mit folgender Form:
Applikationen, deren linke Seite eine Abstraktion ist

$$(\lambda V.\ T_1)\ \ T_2$$

Definition (β-Reduktion)

β-Reduktion ersetzt einen Redex mit einer modifizierten Form von T_1, wobei alle in T_1 freien[1] Vorkommnisse von V mit T_2 ersetzt werden.

$$(\lambda V.\ T_1)\ T_2 \quad \overset{\beta}{\Rightarrow} \quad T_1[T_2/V]$$

[1] Achtung: "frei" nicht im Redex, sondern wenn T_1 alleinstehender Ausdruck wäre

Beispiele

$$(\lambda V.\ T_1)\ T_2 \quad \overset{\beta}{\Rightarrow} \quad T_1[T_2/V]$$

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

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

$$z\ (\lambda x.\ (\lambda y.\ y)\ x)$$

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

Beispiele

$$(\lambda V.\ T_1)\ T_2 \quad \overset{\beta}{\Rightarrow} \quad T_1[T_2/V]$$

$$(\lambda x.\ x)\ z \overset{\beta}{\Rightarrow}\ z$$

$$(\lambda x.\ x\ x)\ (\lambda y.\ y) \overset{\beta}{\Rightarrow}\ (\lambda y.\ y)\ (\lambda y.\ y) \overset{\beta}{\Rightarrow}\ \lambda y.\ y$$

$$z\ (\lambda x.\ (\lambda y.\ y)\ x) \overset{\beta}{\Rightarrow}\ z\ (\lambda x.\ x)$$

$$(\lambda y.\ \lambda x.\ y\ x)\ x \overset{\beta}{\Rightarrow}\ \lightning$$

Capture Avoidance und α-Konversion

β-Reduktion nicht korrekt anwendbar, wenn dadurch “aus Versehen” freie Variablen gebunden werden.

Definition (α-Konversion)

Entspricht intuitiver Variablenumbenennung, anwendbar auf Abstraktions-Teilausdrücke.

$$\lambda x.\, T \quad \overset{\alpha}{\Rightarrow} \quad \lambda y.\, T[y/x]$$

Capture Avoidance und α-Konversion

β-Reduktion nicht korrekt anwendbar, wenn dadurch “aus Versehen” freie Variablen gebunden werden.

Definition (α-Konversion)

Entspricht intuitiver Variablenumbenennung, anwendbar auf Abstraktions-Teilausdrücke.

$$\lambda x.\ T \quad \overset{\alpha}{\Rightarrow} \quad \lambda y.\ T[y/x]$$

$$(\lambda y.\ \lambda x.\ y\ x)\ x \ \overset{\alpha}{\Rightarrow}\ (\lambda y.\ \lambda z.\ y\ z)\ x \ \overset{\beta}{\Rightarrow}\ \lambda z.\ x\ z$$

Wird bei der Reduktion von $(\lambda v.\ T_1)\ T_2$ benötigt gdw. in T_1 an einer Stelle ein (freies) v steht, an der der Name einer freien Variable aus T_2 gebunden ist.

Capture Avoidance und α-Konversion

β-Reduktion nicht korrekt anwendbar, wenn dadurch “aus Versehen” freie Variablen gebunden werden.

Definition (α-Konversion)

Entspricht intuitiver Variablenumbenennung, anwendbar auf Abstraktions-Teilausdrücke.

$$\lambda x.\ T \quad \overset{\alpha}{\Rightarrow} \quad \lambda y.\ T[y/x]$$

$$(\lambda y.\ \lambda x.\ y\ x)\ x \ \overset{\alpha}{\Rightarrow}\ (\lambda y.\ \lambda z.\ y\ z)\ x \ \overset{\beta}{\Rightarrow}\ \lambda z.\ x\ z$$

Wird bei der Reduktion von $(\lambda v.\ T_1)\ T_2$ benötigt gdw. in T_1 an einer Stelle ein (freies) v steht, an der der Name einer freien Variable aus T_2 gebunden ist.

Unschön. $\Longrightarrow$ De-Bruijn-Indizes

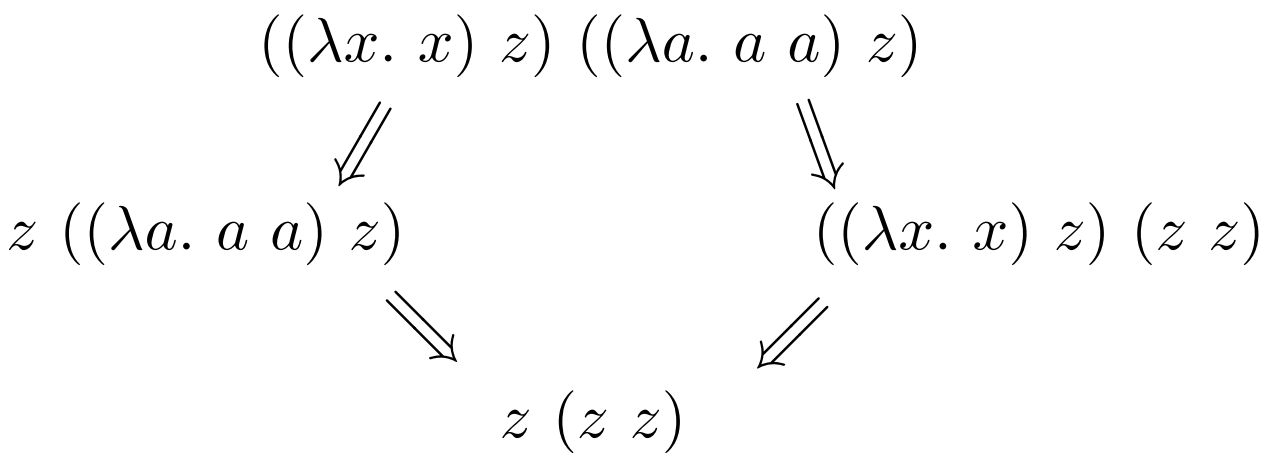
Auswertungsstrategien

Manchmal gibt es mehrere Redexe. Was tun?

$$((\lambda x.\ x)\ z)\ ((\lambda a.\ a\ a)\ z)$$

Auswertungsstrategien

Manchmal gibt es mehrere Redexe. Was tun?



Beispiele für Auswertungsstrategien:

- Normalreihenfolge: zuerst linkesten Redex
- Call-by-Name: zuerst linkesten Redex, **der nicht in einer Abstraktion liegt**
- Call-by-Value: zuerst linkesten Redex, der nicht in einer Abstraktion liegt, **u. dessen Argument vollst. reduziert ist.**

Termination und Divergenz

Reminder: λ-Kalkül ist Turingmächtig.
Das bedeutet aber insbesondere, dass es auch **nicht "terminierende"** Ausdrücke geben muss.

$$(\lambda x.\ x\ x)\ (\lambda x.\ x\ x) \overset{\beta}{\Rightarrow} (\lambda x.\ x\ x)\ (\lambda x.\ x\ x) \overset{\beta}{\Rightarrow} \ldots$$

$$(\lambda x.\ x\ x\ x)\ (\lambda x.\ x\ x\ x) \overset{\beta}{\Rightarrow}\ (\lambda x.\ x\ x\ x)\ (\lambda x.\ x\ x\ x)\ (\lambda x.\ x\ x\ x) \overset{\beta}{\Rightarrow} \ldots$$

Termination und Divergenz

Reminder: λ-Kalkül ist Turingmächtig.
Das bedeutet aber insbesondere, dass es auch **nicht "terminierende"** Ausdrücke geben muss.

$$(\lambda x.\ x\ x)\ (\lambda x.\ x\ x) \overset{\beta}{\Rightarrow} (\lambda x.\ x\ x)\ (\lambda x.\ x\ x) \overset{\beta}{\Rightarrow} \ldots$$

$$(\lambda x.\ x\ x\ x)\ (\lambda x.\ x\ x\ x) \overset{\beta}{\Rightarrow} (\lambda x.\ x\ x\ x)\ (\lambda x.\ x\ x\ x)\ (\lambda x.\ x\ x\ x) \overset{\beta}{\Rightarrow} \ldots$$

Es gibt Ausdrücke, die nur mit bestimmten Auswertungsstrategien terminieren. Beispiel:

$$(\lambda t.\ \lambda f.\ t)\ (\lambda x.\ x)\ T$$

...Mit nicht terminierendem Ausdruck T

How to Turingmächtigkeit

Man nehme...

- Kodierung von Daten
- Kodierung boolscher Werte
- Ein Schema für Fallunterscheidung
- Rekursion

How to Turingmächtigkeit

Man nehme...

- Kodierung von Daten
  Möglich, z.B. Natürliche Zahlen via “Church-Zahlen”
  - 0: $\lambda s.\ \lambda z.\ z$
  - 1: $\lambda s.\ \lambda z.\ s\ z$
  - 4: $\lambda s.\ \lambda z.\ s\ (s\ (s\ (s\ z)))$
- Kodierung boolscher Werte
- Ein Schema für Fallunterscheidung
- Rekursion

How to Turingmächtigkeit

Man nehme...

- Kodierung von Daten
  Möglich, z.B. Natürliche Zahlen via “Church-Zahlen”
  - 0: $\lambda s.\ \lambda z.\ z$
  - 1: $\lambda s.\ \lambda z.\ s\ z$
  - 4: $\lambda s.\ \lambda z.\ s\ (s\ (s\ (s\ z)))$
- Kodierung boolscher Werte
  - True: $\lambda t.\ \lambda f.\ t$
  - False: $\lambda t.\ \lambda f.\ f$
- Ein Schema für Fallunterscheidung

- Rekursion

How to Turingmächtigkeit

Man nehme...

- Kodierung von Daten
  Möglich, z.B. Natürliche Zahlen via “Church-Zahlen”
  0: $\lambda s.\ \lambda z.\ z$
  1: $\lambda s.\ \lambda z.\ s\ z$
  4: $\lambda s.\ \lambda z.\ s\ (s\ (s\ (s\ z)))$
- Kodierung boolscher Werte
  True: $\lambda t.\ \lambda f.\ t$
  False: $\lambda t.\ \lambda f.\ f$
- Ein Schema für Fallunterscheidung
  Boolsche Werte sind Fallunterscheidungs-Funktionen!
  if A then B else C : $A\ B\ C$
- Rekursion

How to Turingmächtigkeit

Man nehme...

- Kodierung von Daten
  Möglich, z.B. Natürliche Zahlen via “Church-Zahlen”
  0: $\lambda s.\ \lambda z.\ z$
  1: $\lambda s.\ \lambda z.\ s\ z$
  4: $\lambda s.\ \lambda z.\ s\ (s\ (s\ (s\ z)))$
- Kodierung boolscher Werte
  True: $\lambda t.\ \lambda f.\ t$
  False: $\lambda t.\ \lambda f.\ f$
- Ein Schema für Fallunterscheidung
  Boolsche Werte sind Fallunterscheidungs-Funktionen!
  if A then B else C : $A\ B\ C$
- Rekursion
  ???

Rekursion

Problem: brauche Namensbindung für selbstreferenziellen Aufruf.

$\mathbf{f} = \lambda n.\ f\ n$

Rekursion

Problem: brauche Namensbindung für selbstreferenziellen Aufruf.

$\lambda f.\ \lambda n.\ f\ n$

Rekursion

Problem: brauche Namensbindung für selbstreferenziellen Aufruf.

$\lambda f.\ \lambda n.\ f\ n$

$(\lambda f.\ \lambda n.\ f\ n)\ (\lambda f.\ \lambda n.\ f\ n) \overset{\beta}{\Rightarrow} \lambda n.\ (\lambda f.\ \lambda n.\ f\ n)\ n$

Rekursion

Problem: brauche Namensbindung für selbstreferenziellen Aufruf.

$\lambda f.\ \lambda n.\ f\ n$

$(\lambda f.\ \lambda n.\ f\ n)\ (\lambda f.\ \lambda n.\ f\ n) \overset{\beta}{\Rightarrow} \lambda n.\ (\lambda f.\ \lambda n.\ f\ n)\ n$

$(\lambda f.\ \lambda n.\ f\ n)\ ((\lambda f.\ \lambda n.\ f\ n)\ (\lambda f.\ \lambda n.\ f\ n)) \overset{\beta}{\Rightarrow} \lambda n.\ (\lambda n.\ (\lambda f.\ \lambda n.\ f\ n)\ n)\ n$

Rekursion

Problem: brauche Namensbindung für selbstreferenziellen Aufruf.

$\lambda f.\ \lambda n.\ f\ n$

$(\lambda f.\ \lambda n.\ f\ n)\ (\lambda f.\ \lambda n.\ f\ n) \overset{\beta}{\Rightarrow} \lambda n.\ (\lambda f.\ \lambda n.\ f\ n)\ n$

$(\lambda f.\ \lambda n.\ f\ n)\ ((\lambda f.\ \lambda n.\ f\ n)\ (\lambda f.\ \lambda n.\ f\ n)) \overset{\beta}{\Rightarrow} \lambda n.\ (\lambda n.\ (\lambda f.\ \lambda n.\ f\ n)\ n)\ n$

Trick: Finde besonderen λ-Ausdruck, der solche rekursiven Terme “richtig bedient”.

Rekursion

Problem: brauche Namensbindung für selbstreferenziellen Aufruf.

$\lambda f.\ \lambda n.\ f\ n$

$(\lambda f.\ \lambda n.\ f\ n)\ (\lambda f.\ \lambda n.\ f\ n) \overset{\beta}{\Rightarrow} \lambda n.\ (\lambda f.\ \lambda n.\ f\ n)\ n$

$(\lambda f.\ \lambda n.\ f\ n)\ ((\lambda f.\ \lambda n.\ f\ n)\ (\lambda f.\ \lambda n.\ f\ n)) \overset{\beta}{\Rightarrow} \lambda n.\ (\lambda n.\ (\lambda f.\ \lambda n.\ f\ n)\ n)\ n$

Trick: Finde besonderen λ-Ausdruck, der solche rekursiven Terme "richtig bedient".

$\mathbf{Y} = \lambda f.\ (\lambda x.\ f\ (x\ x))\ (\lambda x.\ f\ (x\ x))$

Formal: Y ist Fixpunktkombinator, denn $\forall f\colon\ \ Y\ f \overset{\beta}{\Rightarrow}{}^{*}\ \ f\ (Y\ f)$
Schema: für rekursive Funktion f mit Rumpf T benutze $Y\ (\lambda f.\ T)$ als Funktionsdefinition.

Beispiel: Teilbarkeit von Zahlen überprüfen

Gegeben:

- natürliche Zahlen, minus, isZero
- boolsche Werte ctrue, cfalse

```
def less(a, b):
    if a - b == 0:
        return not b - a == 0
    return False
```

Beispiel: Teilbarkeit von Zahlen überprüfen

Gegeben:

- natürliche Zahlen, minus, isZero
- boolsche Werte ctrue, cfalse

```
def less(a, b):
    if isZero(minus(a, b)):
        return not isZero(minus(b, a))
    return cfalse
```

Beispiel: Teilbarkeit von Zahlen überprüfen

Gegeben:

- natürliche Zahlen, minus, isZero
- boolsche Werte ctrue, cfalse

$$
\begin{aligned}
&\mathrm{less} = \lambda \mathrm{a}.\ \lambda \mathrm{b}. \\
&\quad (\mathrm{isZero}(\mathrm{minus}(\mathrm{a},\ \mathrm{b}))) \\
&\qquad (\mathrm{not}\ \mathrm{isZero}(\mathrm{minus}(\mathrm{b},\ \mathrm{a}))) \\
&\quad (\mathrm{cfalse})
\end{aligned}
$$

Beispiel: Teilbarkeit von Zahlen überprüfen

Gegeben:

- natürliche Zahlen, minus, isZero
- boolsche Werte ctrue, cfalse

```
less = λa. λb.
   (isZero (minus a b))
       (isZero (minus b a) cfalse ctrue)
   cfalse
```

Beispiel: Teilbarkeit von Zahlen überprüfen

Gegeben:

- natürliche Zahlen, minus, isZero
- boolsche Werte ctrue, cfalse

```
def divides(a, b):
    if b == 0:
        return True
    if b < a:
        return False
    return divides(a, b - a)
```

Beispiel: Teilbarkeit von Zahlen überprüfen

Gegeben:

- natürliche Zahlen, minus, isZero
- boolsche Werte ctrue, cfalse

```
def divides(a, b):
    if isZero(b):
        return ctrue
    if less(b, a):
        return cfalse
    return divides(a, minus(b, a))
```

Beispiel: Teilbarkeit von Zahlen überprüfen

Gegeben:

- natürliche Zahlen, minus, isZero
- boolsche Werte ctrue, cfalse

```
def divides(a, b):
    isZero b
        ctrue
    (
    (less b a)
        cfalse
    divides(a, minus(b, a))
    )
```

Beispiel: Teilbarkeit von Zahlen überprüfen

Gegeben:

- natürliche Zahlen, minus, isZero
- boolsche Werte ctrue, cfalse

$$
\begin{aligned}
&\text{divides} = \lambda a.\ \lambda b. \\
&\quad \text{isZero b} \\
&\qquad \text{ctrue} \\
&\quad ((\text{less b a}) \\
&\qquad \text{cfalse} \\
&\quad (\text{divides a (minus b a)})
\end{aligned}
$$

Beispiel: Teilbarkeit von Zahlen überprüfen

Gegeben:

- natürliche Zahlen, minus, isZero
- boolsche Werte ctrue, cfalse

$$
\begin{aligned}
&\text{divides} = \text{Y}\ (\lambda\text{divides}.\ \lambda\text{a}.\ \lambda\text{b}.\\
&\quad \text{isZero b}\\
&\qquad \text{ctrue}\\
&\quad ((\text{less b a})\\
&\qquad \text{cfalse}\\
&\quad (\text{divides a (minus b a)}))
\end{aligned}
$$

10 Minuten Rechnen später...

```
divides c7 c21
=> ctrue
                                    3254 Steps
```

```
divides c6 c21
=> cfalse
                                    3528 Steps
```