

# Lambda-Kalkül

Prof. Tobias Nipkow

15. November 2004

# Inhaltsverzeichnis

<b>1</b>	<b>Der untypisierte Lambda-Kalkül</b>	<b>3</b>
1.1	Syntax . . . . .	3
1.1.1	Terme . . . . .	3
1.1.2	Currying (Schönfinkeln) . . . . .	4
1.1.3	Statische Bindung und Substitution . . . . .	5
1.1.4	$\alpha$ -Konversion . . . . .	6
1.2	$\beta$ -Reduktion (Kontraktion) . . . . .	7
1.3	$\eta$ -Reduktion . . . . .	11
1.4	$\lambda$ -Kalkül als Gleichungstheorie . . . . .	14
1.4.1	$\beta$ -Konversion . . . . .	14
1.4.2	$\eta$ -Konversion und Extensionalität . . . . .	14
1.5	Reduktionsstrategien . . . . .	15
1.6	Markierte Terme . . . . .	15
1.7	$\lambda$ -Kalkül als „Programmiersprache“ . . . . .	17
1.7.1	Datentypen . . . . .	17
1.7.2	Rekursive Funktionen . . . . .	18
<b>2</b>	<b>Kombinatorische Logik (CL)</b>	<b>21</b>
2.1	Beziehung zwischen $\lambda$ -Kalkül und CL . . . . .	22
2.2	Implementierungsaspekte . . . . .	23
<b>3</b>	<b>Typisierte Lambda-Kalküle</b>	<b>27</b>
3.1	Einfach typisierter $\lambda$ -Kalkül ( $\lambda^{\rightarrow}$ ) . . . . .	28
3.1.1	Typüberprüfung für explizit typisierte Terme . . . . .	28
3.2	Termination von $\rightarrow_{\beta}$ . . . . .	30
3.3	Typinferenz für $\lambda^{\rightarrow}$ . . . . .	31
3.4	let-Polymorphismus . . . . .	32
<b>4</b>	<b>Der Curry-Howard Isomorphismus</b>	<b>35</b>
<b>A</b>	<b>Relationale Grundlagen</b>	<b>39</b>
A.1	Notation . . . . .	39
A.2	Konfluenz . . . . .	39
A.3	Kommutierende Relationen . . . . .	42



# Kapitel 1

## Der untypisierte Lambda-Kalkül

### 1.1 Syntax

#### 1.1.1 Terme

**Definition 1.1.1** Terme des Lambda-Kalküls sind wie folgt definiert:

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

$(t_1 t_2)$  heißt **Applikation** und repräsentiert die Anwendung einer Funktion  $t_1$  auf ein Argument  $t_2$ .

$(\lambda x.t)$  heißt **Abstraktion** und repräsentiert die Funktion mit formalem Parameter  $x$  und Körper  $t$ ;  $x$  ist in  $t$  gebunden.

Konvention:

$x, y, z$	Variablen
$c, d, f, g, h$	Konstanten
$a, b$	Atome = Variablen $\cup$ Konstanten
$r, s, t, u, v, w$	Terme

Ziel: Definition der  $\beta$ -Reduktion, d.h. der Auswertung einer Applikation  $((\lambda x.s) t)$  durch Einsetzen des Argumentes  $t$  für den formalen Parameter  $x$  in  $s$ . Beispiele:

$$\begin{aligned} ((\lambda x.((f x)x))5) & \rightarrow_{\beta} ((f 5)5) \\ ((\lambda x.x)(\lambda x.x)) & \rightarrow_{\beta} (\lambda x.x) \\ (x(\lambda y.y)) & \text{ kann nicht „reduziert“ werden} \end{aligned}$$

Notation:

- Variablen werden nach  $\lambda$  aufgelistet:  $\lambda x_1 \dots x_n.s \equiv \lambda x_1 \dots \lambda x_n.s$
- Applikation assoziiert nach links:  $(t_1 \dots t_n) \equiv (((t_1 t_2)t_3) \dots t_n)$
- $\lambda$  bindet so weit nach rechts wie möglich.

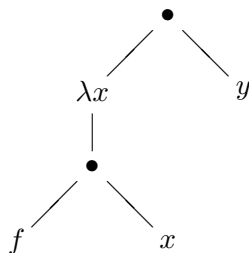
Beispiel:  $\lambda x.x x \equiv \lambda x.(x x)$   
 $\neq (\lambda x.x) x$

- Äußerste Klammern werden weggelassen:  $t_1 \dots t_n \equiv (t_1 \dots t_n)$

Terme als Bäume:

Term:	$c$	$x$	$(\lambda x.t)$	$(t_1 t_2)$
Baum:	$c$	$x$	$\begin{array}{c} \lambda x \\   \\ t \end{array}$	$\begin{array}{c} \bullet \\ / \quad \backslash \\ t_1 \quad t_2 \end{array}$

Beispiel: Baum zum Term  $(\lambda x.f x) y$



**Definition 1.1.2**  $s$  ist **Subterm** von  $t$ , falls der zu  $s$  gehörige Baum ein Unterbaum des zu  $t$  gehörigen Baums ist.

Beispiel:

Ist  $s(t u)$  ein Subterm von  $r s(t u)$ ?

Nein, da  $r s(t u) \equiv (r s)(t u)$

Für alle Terme  $s$  gilt:  $s$  ist Subterm von  $s$ , aber kein *echter Subterm* von  $s$ .

### 1.1.2 Curryng (Schönfinkeln)

*Currying* bedeutet Reduktion mehrstelliger Funktionen auf solche mit nur einem Argument.

Beispiel:

$$f : \begin{cases} \mathbb{N} \rightarrow \mathbb{N} \\ x \mapsto x + x \end{cases}$$

Entsprechung im Lambda-Kalkül:  $f = \lambda x.x + x$

$$g : \begin{cases} \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N} \\ (x, y) \mapsto x + y \end{cases}$$

*falsche* Umsetzung für  $g$ :  $\lambda(x, y).x + y$

Ein solcher „Term“ entspricht nicht der Syntax des Lambda-Kalküls !

stattdessen:  $g \cong g' = \lambda x.\lambda y.x + y$

demnach:  $g': \mathbb{N} \rightarrow (\mathbb{N} \rightarrow \mathbb{N})$

Beispiel zur Auswertung:  $g(5, 3) = 5 + 3$

Auswertung im Lambda-Kalkül hierzu:

$$\begin{aligned} g' \ 5 \ 3 &\equiv ((g' \ 5) \ 3) &\equiv (((\lambda x. \lambda y. x + y) \ 5) \ 3) \\ &\rightarrow_{\beta} ((\lambda y. 5 + y) \ 3) \\ &\rightarrow_{\beta} 5 + 3 \end{aligned}$$

Der Term  $g' \ 5$  ist wohldefiniert (*partielle Applikation*).  
Veranschaulichung: In der Verknüpfungstafel von  $g$

$g$	$b_1$	$b_2$	$\dots$
$a_1$	$\cdot$	$\cdot$	$\dots$
$a_2$	$\cdot$	$\cdot$	$\dots$
$\vdots$	$\vdots$	$\vdots$	$\ddots$

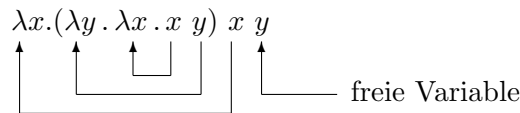
entspricht  $g' \ 5$  der einstelligen Funktion, die durch die Zeile mit  $a_i = 5$  gegeben ist.

Mengentheoretisch:  $(A \times B) \rightarrow C \cong A \rightarrow (B \rightarrow C)$   
( „ $\cong$ “: mengentheoretische Isomorphie )

### 1.1.3 Statische Bindung und Substitution

Eine Variable  $x$  im Term  $s$  wird durch das erste  $\lambda x$  oberhalb von  $x$  (bzgl. des Termbaumes) gebunden.

Beispiel:



Die Pfeile zeigen vom Vorkommen einer Variablen zum bindenden  $\lambda$ .

Menge der freien Variablen eines Terms als Funktion:

$$\begin{aligned} FV: \quad \text{Term} &\rightarrow \text{Menge von Variablen} \\ FV(c) &= \emptyset \\ FV(x) &= \{x\} \\ FV(s \ t) &= FV(s) \cup FV(t) \\ FV(\lambda x. t) &= FV(t) \setminus \{x\} \end{aligned}$$

**Definition 1.1.3** Ein Term  $t$  ist **geschlossen**, falls  $FV(t) = \emptyset$ .

**Definition 1.1.4** Die **Substitution** von  $t$  für  $x$  in  $s$ ,  $s[t/x]$  (sprich: „ $s$  mit  $t$  für  $x$ “), ist rekursiv definiert:

$$\begin{aligned}
x[t/x] &= t \\
a[t/x] &= a && \text{falls } a \neq x \\
(s_1 s_2)[t/x] &= (s_1[t/x]) (s_2[t/x]) \\
(\lambda x.s)[t/x] &= \lambda x.s \\
(\lambda y.s)[t/x] &= \lambda y.(s[t/x]) && \text{falls } x \neq y \wedge y \notin FV(t) \\
(\lambda y.s)[t/x] &= \lambda z.(s[z/y][t/x]) && \text{falls } x \neq y \wedge z \notin FV(t) \cup FV(s)
\end{aligned}$$

Hierbei ist  $a$  ein Atom, d.h. eine Konstante oder eine Variable.

(Man kann zeigen, daß, modulo Umbenennung gebundener Variablen, die vorletzte Vorschrift auf die letzte zurückgeführt werden kann.)

Beispiel:

$$\begin{aligned}
(x (\lambda x.x) (\lambda y.z x)) [y/x] &= (x[y/x]) ( (\lambda x.x)[y/x] ) ( (\lambda y.z x)[y/x] ) = \\
&= y (\lambda x.x) (\lambda y'.z y)
\end{aligned}$$

### Lemma 1.1.5

1.  $s[x/x] = s$
2.  $s[t/x] = s$  falls  $x \notin FV(s)$
3.  $s[y/x][t/y] = s[t/x]$  falls  $y \notin FV(s)$
4.  $s[t/x][u/y] = s[u/y][t[u/y]/x]$  falls  $x \notin FV(u)$
5.  $s[t/x][u/y] = s[u/y][t/x]$  falls  $y \notin FV(t) \wedge x \notin FV(u)$

Bemerkung:

Obige Gleichungen sind nur bis auf Umbenennung gebundener Variablen korrekt.

Beispiel zu 3. mit  $s = \lambda y.y$ :

$$\begin{aligned}
(\lambda y.y)[y/x][c/y] &= \lambda y'.y' \\
\text{aber: } s[c/x] &= (\lambda y.y) = \lambda y.y
\end{aligned}$$

Wir werden in Zukunft Terme wie  $\lambda y.y$  und  $\lambda y'.y'$  identifizieren.

### 1.1.4 $\alpha$ -Konversion

$$s =_{\alpha} t$$

falls  $s$  und  $t$  gleich sind modulo Umbenennung gebundener Variablen:

„Gebundene Namen sind Schall und Rauch“

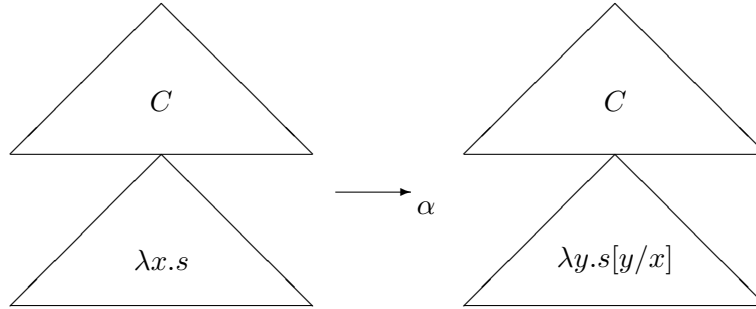
Beispiel:

$$\begin{aligned}
x (\lambda x, y.x y) &=_{\alpha} x (\lambda y, x.y x) =_{\alpha} x (\lambda z, y.z y) \\
&\neq_{\alpha} z (\lambda z, y.z y) \\
&\neq_{\alpha} x (\lambda x, x.x x)
\end{aligned}$$

**Definition 1.1.6** Wir definieren zuerst die 1-Schritt-Umbenennung  $\rightarrow_\alpha$  und dann  $\alpha$ -Äquivalenz  $=_\alpha$  als die transitive und reflexive Hülle von  $\rightarrow_\alpha$ .

$$\begin{aligned} s \rightarrow_\alpha t & :\Leftrightarrow s = C[\lambda x.u] \wedge t = C[\lambda y.(u[y/x])] \wedge y \notin FV(u) \\ s =_\alpha t & :\Leftrightarrow s \rightarrow_\alpha^* t \end{aligned}$$

Hierbei ist  $C[v]$  ein *Kontext* des Terms  $v$ , d.h. ein Term, der  $v$  als Teilterm enthält. Dies ist graphisch in folgendem Bild dargestellt:



**Lemma 1.1.7**  $=_\alpha$  ist symmetrisch.

Konventionen:

1.  $\alpha$ -äquivalente Terme werden identifiziert, d.h. wir arbeiten mit  $\alpha$ -Äquivalenzklassen von Termen. Beispiel:  $\lambda x.x = \lambda y.y$ .
2. Gebundene Variablen werden automatisch so umbenannt, daß sie von allen freien Variablen verschieden sind. Beispiel: Sei  $K = \lambda x.\lambda y.x$ :

$$\begin{aligned} K s & \rightarrow_\beta \lambda y.s && (\text{falls } y \notin FV(s)) \\ K y & \rightarrow_\beta \lambda y'.y && (y \text{ ist frei in } y \text{ und wird daher zu } y' \text{ umbenannt}) \end{aligned}$$

Dies vereinfacht Substitution:

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

Automatisch:

- $x \notin FV(t)$
- $x \neq y$  weil  $x \notin FV(y)$

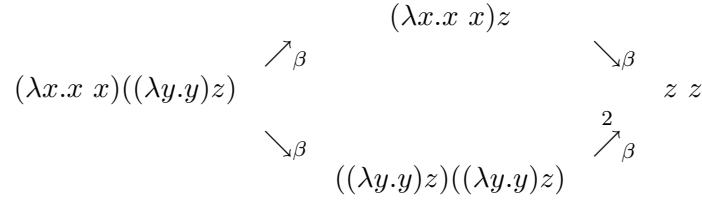
## 1.2 $\beta$ -Reduktion (Kontraktion)

**Definition 1.2.1** Ein  $\beta$ -Redex (**r**educible **e**xpression) ist ein Term der Form  $(\lambda x.s)t$ . Wir definieren  $\beta$ -Reduktion durch

$$C[(\lambda x.s)t] \rightarrow_\beta C[s[t/x]]$$

Ein Term  $t$  ist in  $\beta$ -Normalform falls er in Normalform bzgl.  $\rightarrow_\beta$  ist.



Abbildung 1.1:  $\rightarrow_{\beta}$  kann verzweigen

Beispiel:  $\lambda x. \underbrace{(\lambda x.x x)(\lambda x.x)} \rightarrow_{\beta} \lambda x. \underbrace{(\lambda x.x)(\lambda x.x)} \rightarrow_{\beta} \lambda x.\lambda x.x$

$\beta$ -Reduktion ist

- nichtdeterministisch aber determiniert. Beispiel: siehe Abb. 1.1.
- nicht-terminierend. Beispiel:  $\Omega := (\lambda x.x x)(\lambda x.x x) \rightarrow_{\beta} \Omega$ .

**Definition 1.2.2** Alternativ zu Definition 1.2.1 kann man  $\rightarrow_{\beta}$  induktiv wie folgt definieren:

1.  $(\lambda x.s)t \rightarrow_{\beta} s[t/x]$
2.  $s \rightarrow_{\beta} s' \Rightarrow (s t) \rightarrow_{\beta} (s' t)$
3.  $s \rightarrow_{\beta} s' \Rightarrow (t s) \rightarrow_{\beta} (t s')$
4.  $s \rightarrow_{\beta} s' \Rightarrow \lambda x.s \rightarrow_{\beta} \lambda x.s'$

Das heißt,  $\rightarrow_{\beta}$  ist die kleinste Relation, die die obigen vier Regeln erfüllt.

**Lemma 1.2.3**  $t \rightarrow_{\beta}^* t' \Rightarrow s[t/x] \rightarrow_{\beta}^* s[t'/x]$

Beweis: mit Induktion über  $s$ :

1.  $s = x$ : klar
2.  $s = y \neq x$ :  $s[t/x] = y \rightarrow_{\beta}^* y = s[t'/x]$
3.  $s = c$ : wie unter 2.
4.  $s = (s_1 s_2)$ :
 
$$\begin{aligned}
 (s_1 s_2)[t/x] &= (s_1[t/x]) (s_2[t/x]) \rightarrow_{\beta}^* (s_1[t'/x]) (s_2[t/x]) \rightarrow_{\beta}^* \\
 &\rightarrow_{\beta}^* (s_1[t'/x]) (s_2[t'/x]) = (s_1 s_2)[t'/x] = s[t'/x]
 \end{aligned}$$

(unter Verwendung der Induktions-Hypothese  $s_i[t/x] \rightarrow_{\beta}^* s_i[t'/x]$ ,  $i = 1, 2$ , sowie der Transitivität von  $\rightarrow_{\beta}^*$ )

5.  $s = \lambda y.r$ :  $s[t/x] = \lambda y.(r[t/x]) \rightarrow_{\beta}^* \lambda y.(r[t'/x]) = (\lambda y.r)[t'/x] = s[t'/x]$   
(unter Verwendung der Induktions-Hypothese  $r[t/x] \rightarrow_{\beta}^* r[t'/x]$ ) □

**Lemma 1.2.4** Die vier Regeln in Definition 1.2.2 gelten auch mit  $\rightarrow_\beta^*$  an Stelle von  $\rightarrow_\beta$ .

**Lemma 1.2.5**  $s \rightarrow_\beta s' \Rightarrow s[t/x] \rightarrow_\beta s'[t/x]$

Beweis: mit Induktion über die Herleitung von  $s \rightarrow_\beta s'$  (Regelinduktion) nach Definition 1.2.2.

1.  $s = (\lambda y.r)u \rightarrow_\beta r[u/y] = s'$ :

$$s[t/x] = (\lambda y.(r[t/x]))(u[t/x]) \rightarrow_\beta (r[t/x])[u[t/x]/y] = (r[u/y])[t/x] = s'[t/x]$$

2.  $s_1 \rightarrow_\beta s'_1$  und  $s = (s_1 s_2) \rightarrow_\beta (s'_1 s_2) = s'$ :

Induktions-Hypothese:  $s_1[t/x] \rightarrow_\beta s'_1[t/x]$

$$\Rightarrow s[t/x] = (s_1[t/x])(s_2[t/x]) \rightarrow_\beta (s'_1[t/x])(s_2[t/x]) = (s'_1 s_2)[t/x] = s'[t/x]$$

3. Analog zu 2.

4. Zur Übung. □

**Korollar 1.2.6**  $s \rightarrow_\beta^n s' \Rightarrow s[t/x] \rightarrow_\beta^n s'[t/x]$  (ebenso mit  $\rightarrow_\beta^*$  statt  $\rightarrow_\beta^n$ )

Beweis: mit Induktion über  $n$  □

**Korollar 1.2.7**  $s \xrightarrow{\beta}^* s' \wedge t \xrightarrow{\beta}^* t' \Rightarrow s[t/x] \xrightarrow{\beta}^* s'[t'/x]$

Beweis:  $s[t/x] \xrightarrow{\beta}^* s'[t/x] \xrightarrow{\beta}^* s'[t'/x]$  □

Gilt auch  $t \rightarrow_\beta t' \Rightarrow s[t/x] \rightarrow_\beta s[t'/x]$ ?

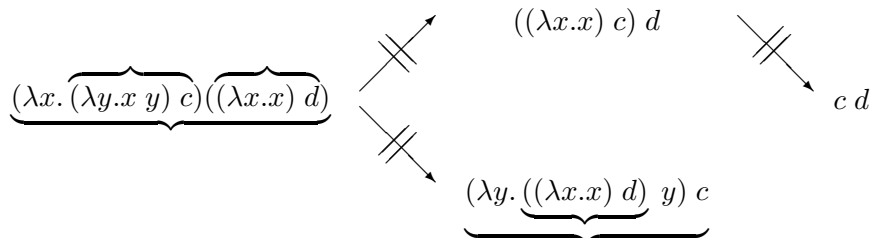
**Übung 1.2.8** Zeige  $s \rightarrow_\beta t \Rightarrow FV(s) \supseteq FV(t)$ . Warum gilt  $FV(s) = FV(t)$  nicht?

### Konfluenz

Wir versuchen Konfluenz über die Diamant-Eigenschaft zu beweisen. Wie man in Abbildung 1.1 sieht, hat aber  $\rightarrow_\beta$  die Diamant-Eigenschaft nicht, da  $t := ((\lambda y.y)z)((\lambda y.y)z)$  nicht in einem Schritt auf  $z z$  reduziert werden kann.

1. Versuch: parallele Reduktion unabhängiger Redexe (in Zeichen:  $\Downarrow$ ) da  $t \Downarrow z z$ .

Problem:  $\Downarrow$  hat die Diamant-Eigenschaft auch nicht:



Es gilt nicht  $(\lambda y.((\lambda x.x)d)y)c \Downarrow c d$  da  $(\lambda y.((\lambda x.x)d)y)c$  geschachtelte Redexe enthält.

**Definition 1.2.9** Parallele (und geschachtelte) Reduktion (in Zeichen:  $\Downarrow$ )

1.  $s > s$
2.  $\lambda x.s > \lambda x.s'$  falls  $s > s'$
3.  $(s t) > (s' t')$  falls  $s > s'$  und  $t > t'$  (parallel)
4.  $(\lambda x.s)t > s'[t'/x]$  falls  $s > s'$  und  $t > t'$  (parallel und geschachtelt)

Beispiel:

$$\underbrace{(\lambda x. \underbrace{((\lambda y.y) x)}_x)}_x \underbrace{((\lambda x.x) z)}_z > z$$

Merke:

$>$  ist echte Teilmenge von  $\rightarrow_{\beta}^*$ : Es gilt  $(\lambda f.f z)(\lambda x.x) \rightarrow_{\beta} (\lambda x.x)z \rightarrow_{\beta} z$  und  $(\lambda f.f z)(\lambda x.x) > (\lambda x.x)z$  aber nicht  $(\lambda f.f z)(\lambda x.x) \rightarrow_{\beta}^* z$ .

**Lemma 1.2.10**  $s \rightarrow_{\beta} t \Rightarrow s > t$

Beweis: mit Induktion über die Herleitung von  $s \rightarrow_{\beta} t$  nach Definition 1.2.2.

1. Fall:  $s = (\lambda x.u) v \rightarrow_{\beta} u[v/x] = t$   
 $\Rightarrow (\lambda x.u) v > u[v/x] = t$ , da  $u > u$  und  $v > v$

Restliche Fälle: zur Übung □

**Lemma 1.2.11**  $s > t \Rightarrow s \rightarrow_{\beta}^* t$

Beweis: mit Induktion über die Herleitung von  $s > t$  nach Definition 1.2.9.

4. Fall:  $s = (\lambda x.u) v > u'[v'/x] = t, u > u', v > v'$   
 Induktions-Hypothesen:  $u \rightarrow_{\beta}^* u', v \rightarrow_{\beta}^* v'$   
 $s = (\lambda x.u)v \rightarrow_{\beta}^* (\lambda x.u')v \rightarrow_{\beta}^* (\lambda x.u')v' \rightarrow_{\beta} u'[v'/x]$

Restliche Fälle: zur Übung □

Damit gilt auch, daß  $\rightarrow_{\beta}^*$  und  $>^*$  identisch sind.

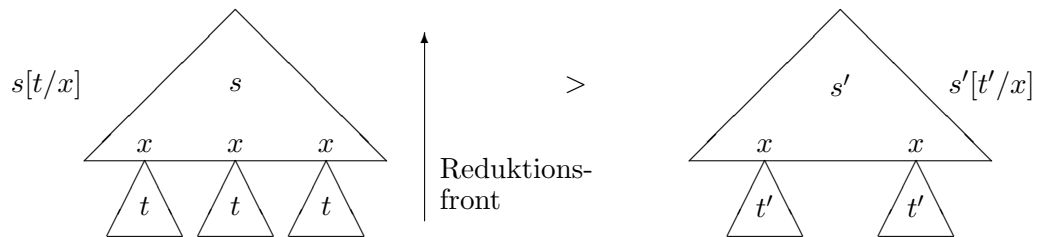
Das nächste Lemma folgt direkt aus der Analyse der anwendbaren Regeln:

**Lemma 1.2.12**  $\lambda x.s > t \Rightarrow \exists s'. t = \lambda x.s' \wedge s > s'$

**Lemma 1.2.13**  $s > s' \wedge t > t' \Rightarrow s[t/x] > s'[t'/x]$

Beweis:

mit Induktion über  $s$ ; im Fall  $s = (s_1 s_2)$  Fallunterscheidung nach angewandter Regel. Details zur Übung. Graphisch läßt sich der Beweis wie folgt veranschaulichen:



**Theorem 1.2.14**  $>$  hat die Diamant-Eigenschaft.

Beweis: wir zeigen  $s > t_1 \wedge s > t_2 \Rightarrow \exists u. t_1 > u \wedge t_2 > u$  mit Induktion über  $s$ .

1.  $s$  ist Atom  $\Rightarrow s = t_1 = t_2 =: u$
2.  $s = \lambda x. s'$   
 $\Rightarrow t_i = \lambda x. t'_i$  und  $s' > t'_i$  (für  $i = 1, 2$ )  
 $\Rightarrow \exists u'. t'_i > u' \quad (i = 1, 2)$  (nach Induktions-Hypothese)  
 $\Rightarrow t_i = \lambda x. t'_i > \lambda x. u' =: u$
3.  $s = (s_1 s_2)$

Fallunterscheidung nach den Regeln. Konvention:  $s_i > s'_i, s''_i$  und  $s'_i, s''_i > u_i$ .

(a) (mit Ind.-Hyp.)

$$\begin{array}{ccc} (s_1 s_2) & >_3 & (s'_1 s'_2) \\ \vee_3 & & \vee_3 \\ (s''_1 s''_2) & >_3 & (u_1 u_2) \end{array}$$

(b) (mit Ind.-Hyp. und Lemma 1.2.13)

$$\begin{array}{ccc} (\lambda x. s_1) s_2 & >_4 & s'_1 [s'_2/x] \\ \vee_4 & & \vee \\ s''_1 [s''_2/x] & > & u_1 [u_2/x] \end{array}$$

(c) (mit Ind.-Hyp. und Lemma 1.2.13)

$$\begin{array}{ccc} (\lambda x. s_1) s_2 & >_3 & (\lambda x. s'_1) s'_2 \\ \vee_4 & & \vee_4 \\ s''_1 [s''_2/x] & > & u_1 [u_2/x] \end{array}$$

Aus den Lemmata 1.2.10 und 1.2.11 und Theorem 1.2.14 folgt mit A.2.5 nun direkt

**Korollar 1.2.15**  $\rightarrow_\beta$  ist konfluent.

### 1.3 $\eta$ -Reduktion

$$\lambda x. (t x) \rightarrow_\eta t \quad \text{falls } x \notin FV(t)$$

Motivation für die  $\eta$ -Reduktion:  $\lambda x. (t x)$  und  $t$  verhalten sich als Funktionen gleich:

$$(\lambda x. (t x)) u \rightarrow_\beta t u$$

falls  $x \notin FV(t)$ .

Natürlich ist  $\eta$ -Reduktion nicht nur an der Wurzel erlaubt.

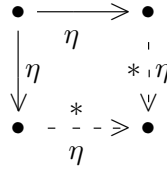
**Definition 1.3.1**  $C[\lambda x. (t x)] \rightarrow_\eta C[t]$  falls  $x \notin FV(t)$ .

**Fakt 1.3.2**  $\rightarrow_\eta$  terminiert.

Es wird die lokale Konfluenz von  $\rightarrow_\eta$  gezeigt; hieraus folgt mit obigem Fakt und Newmanns Lemma die Konfluenz von  $\rightarrow_\eta$ .

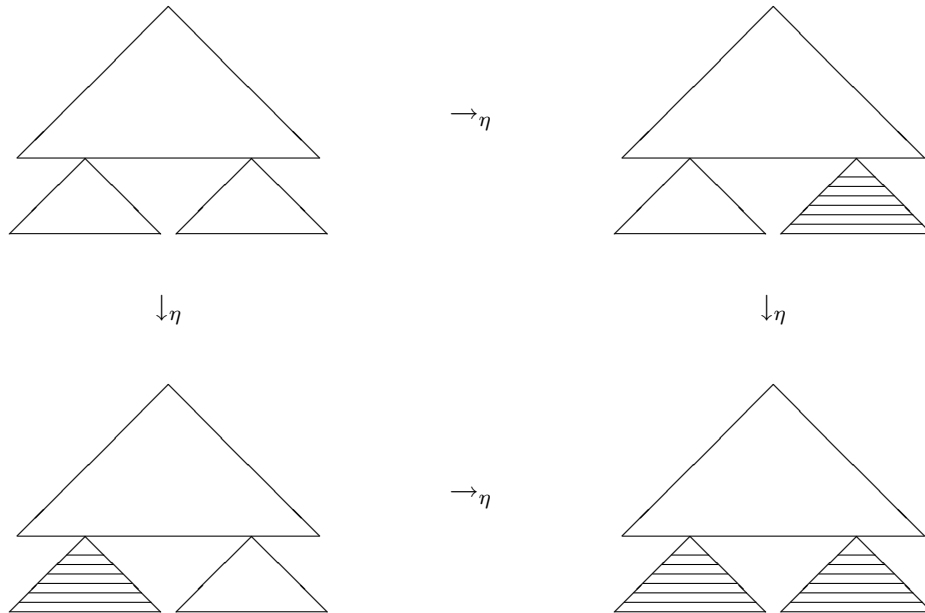
**Fakt 1.3.3**  $s \rightarrow_\eta t \Rightarrow FV(s) = FV(t)$

**Lemma 1.3.4**  $\rightarrow_\eta$  ist lokal konfluent.



Beweis: mit Fallunterscheidung nach der relativen Position der beiden Redexe im Syntaxbaum des Terms

1. Fall: Die Redexe liegen in getrennten Teiltermen.



2. Fall: Die Redexe sind identisch. (klar)

3. Fall: Die Redexe liegen übereinander. Beweis mit Fakt 1.3.3.

$$\begin{array}{ccc}
 \lambda x. s x & \rightarrow_\eta & s \\
 \downarrow \eta & & \downarrow \eta \\
 \lambda x. s' x & \rightarrow_\eta & s'
 \end{array}$$

**Korollar 1.3.5**  $\rightarrow_\eta$  ist konfluent.

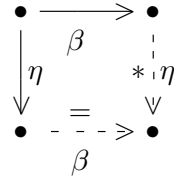
Beweis:  $\rightarrow_\eta$  terminiert und ist lokal konfluent.

**Übung:** Definieren Sie  $\rightarrow_\eta$  induktiv und beweisen Sie die lokale Konfluenz von  $\rightarrow_\eta$  mit Hilfe dieser Definition.

Bemerkung:

$\rightarrow_\eta$  hat die Diamant-Eigenschaft nicht, aber man kann den Beweis zu Lemma 1.3.3 leicht modifizieren und zeigen, daß  $\xrightarrow[\eta]{=}$  die Diamant-Eigenschaft hat.

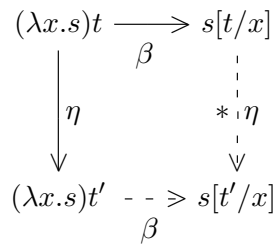
**Lemma 1.3.6**



Beweis: mit Fallunterscheidung nach der relativen Position der Redexe

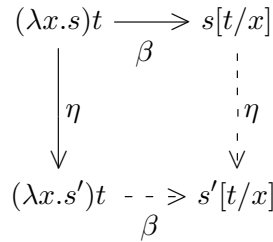
1. in getrennten Teilbäumen: klar
2.  $\eta$ -Redex weit unterhalb des  $\beta$ -Redexes:

(a)  $t \rightarrow_\eta t'$ :

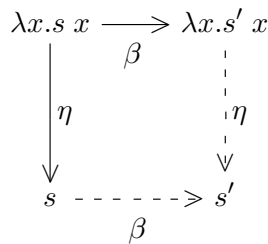


unter Verwendung des Lemmas  $t \rightarrow_\eta t' \Rightarrow s[t/x] \rightarrow_\eta^* s[t'/x]$ .

(b)  $s \rightarrow_\eta s'$ :

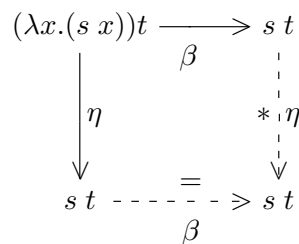


3.  $\beta$ -Redex ( $s \rightarrow_\beta s'$ ) weit unterhalb des  $\eta$ -Redexes:



mit Hilfe von Übung 1.2.8.

4.  $\eta$ -Redex direkt unterhalb des  $\beta$ -Redexes (d.h. überlappend):



5.  $\beta$ -Redex direkt unterhalb des  $\eta$ -Redexes:

$$\begin{array}{ccc}
 \lambda x.((\lambda y.s)x) & \xrightarrow{\beta} & \lambda x.s[x/y] \\
 \downarrow \eta & & \downarrow * \eta \\
 \lambda y.s & \xrightarrow[\beta]{=} & \lambda y.s
 \end{array}$$

weil  $\lambda y.s =_{\alpha} \lambda x.s[x/y]$  da  $x \notin FV(s)$  wegen  $\lambda x.((\lambda y.s)x) \rightarrow_{\eta} \lambda y.s$  □

Mit Lemma A.3.3 folgt, daß  $\xrightarrow{*}_{\beta}$  und  $\xrightarrow{*}_{\eta}$  kommutieren, und da beide konfluent sind, folgt mit dem Lemma von Hindley und Rosen

**Korollar 1.3.7**  $\rightarrow_{\beta\eta}$  ist konfluent.

## 1.4 $\lambda$ -Kalkül als Gleichungstheorie

### 1.4.1 $\beta$ -Konversion

**Definition 1.4.1** [Äquivalenz modulo  $\beta$ -Konversion]

$$s =_{\beta} t :\Leftrightarrow s \leftrightarrow_{\beta}^* t$$

Alternative:

$$(\lambda x.s) t =_{\beta} s[t/x] \quad t =_{\beta} t$$

$$\frac{s =_{\beta} t}{\lambda x.s =_{\beta} \lambda x.t} \quad \frac{s =_{\beta} t}{t =_{\beta} s} \quad \frac{s_1 =_{\beta} t_1 \quad s_2 =_{\beta} t_2}{(s_1 s_2) =_{\beta} (t_1 t_2)} \quad \frac{s =_{\beta} t \quad t =_{\beta} u}{s =_{\beta} u}$$

Da  $\rightarrow_{\beta}$  konfluent ist, kann der Test auf Äquivalenz durch Suche nach einem gemeinsamen Redukt ersetzt werden.

**Theorem 1.4.2**  $s =_{\beta} t$  ist entscheidbar, falls  $s$  und  $t$  eine  $\beta$ -Normalform besitzen, sonst unentscheidbar.

Beweis: Entscheidbarkeit folgt direkt aus Korollar A.2.8, da  $\rightarrow_{\beta}$  konfluent ist. Unentscheidbarkeit folgt daraus, daß  $\lambda$ -Terme Programme sind, und Programmäquivalenz unentscheidbar ist. □

### 1.4.2 $\eta$ -Konversion und Extensionalität

**Extensionalität** besagt, daß zwei Funktionen sind gleich, falls sie auf allen Argumenten gleich sind:

$$\text{ext} : \frac{\forall u. s u = t u}{s = t}$$

**Theorem 1.4.3**  $\beta + \eta$  und  $\beta + \text{ext}$  definieren die gleiche Äquivalenz auf  $\lambda$ -Termen.

Beweis:

$$\eta \Rightarrow \text{ext} : \forall u. s u = t u \Rightarrow s x = t x \text{ wobei } x \notin FV(s, t) \Rightarrow s =_{\eta} \lambda x.(s x) = \lambda x.(t x) = t$$

$$\beta + \text{ext} \Rightarrow \eta : \text{sei } x \notin FV(s) : \forall u. (\lambda x.(s x))u =_{\beta} s u \Rightarrow \lambda x.(s x) = s \quad \square$$

**Definition 1.4.4**

$$s \rightarrow_{\beta\eta} t \quad :\Leftrightarrow \quad s \rightarrow_{\beta} t \vee s \rightarrow_{\eta} t$$

$$s =_{\beta\eta} t \quad :\Leftrightarrow \quad s \leftrightarrow_{\beta\eta}^* t$$

Analog zu  $=_{\beta}$  gilt

**Theorem 1.4.5**  $s =_{\beta\eta} t$  ist entscheidbar, falls  $s$  und  $t$  eine  $\beta\eta$ -Normalform besitzen, und sonst unentscheidbar.

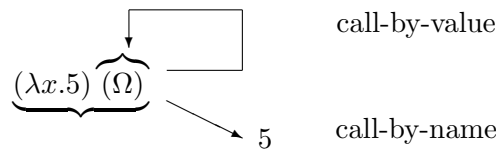
Da  $\rightarrow_{\eta}$  terminiert und konfluent ist gilt natürlich

**Korollar 1.4.6**  $\leftrightarrow_{\eta}^*$  ist entscheidbar.

## 1.5 Reduktionsstrategien

**Theorem 1.5.1** Hat  $t$  eine  $\beta$ -Normalform, dann kann man diese erreichen, indem man immer den jeweils linken  $\beta$ -Redex reduziert.

Beispiel ( $\Omega := (\lambda x.x x)(\lambda x.x x)$ ):



## 1.6 Markierte Terme

Motivation: let-Ausdrücke

$$\text{let } x = s \text{ in } t \rightarrow_{\text{let}} t[s/x]$$

Man kann let als markierten  $\beta$ -Redex auffassen. Beispiel:

$$\begin{array}{ccc} \text{let } x = (\text{let } y = s \text{ in } y + y) \text{ in } x * x & \longrightarrow & \text{let } x = s + s \text{ in } x * x \\ \downarrow & & \vdots \\ (\text{let } y = s \text{ in } y + y) * (\text{let } y = s \text{ in } y + y) & \text{---} & (s + s) * (s + s) \end{array}$$

Menge der markierten Terme:

$$\underline{\mathcal{T}} : \quad t ::= c \mid x \mid (t_1 t_2) \mid \lambda x.t \mid (\underline{\lambda}x.s) t$$

Merke:  $\underline{\lambda}x.s \notin \underline{\mathcal{T}}$  (Warum?)

**Definition 1.6.1**  $\underline{\beta}$ -Reduktion markierter Terme:



$$C[(\underline{\lambda}x.s) t] \rightarrow_{\underline{\beta}} C[s[t/x]]$$

Ziel:  $\rightarrow_{\underline{\beta}}$  terminiert.

Eigenschaft:  $\rightarrow_{\underline{\beta}}$  kann keine neuen markierten Redexe erzeugen, sondern nur existierende Redexe kopieren und modifizieren. Ein Beispiel soll den Unterschied zwischen  $\rightarrow_{\beta}$  und  $\rightarrow_{\underline{\beta}}$  illustrieren:

$$(\lambda x.x x)(\lambda x.x x) \rightarrow_{\beta} \underbrace{(\lambda x.x x)(\lambda x.x x)}_{\text{neuer } \beta\text{-Redex}}$$

aber

$$(\underline{\lambda}x.x x)(\lambda x.x x) \rightarrow_{\underline{\beta}} \underbrace{(\lambda x.x x)(\lambda x.x x)}_{\text{kein } \underline{\beta}\text{-Redex}}$$

Wenn  $s \rightarrow_{\underline{\beta}} t$ , dann stammt jeder  $\underline{\beta}$ -Redex in  $t$  von genau einem  $\underline{\beta}$ -Redex in  $s$  ab.

Im folgenden sei  $s[t_1/x_1, \dots, t_n/x_n]$  die simultane Ersetzung der  $x_i$  durch die  $t_i$  in  $s$ .

### Lemma 1.6.2

1.  $s, t_1, \dots, t_n \in \underline{\mathcal{T}} \Rightarrow s[t_1/x_1, \dots, t_n/x_n] \in \underline{\mathcal{T}}$
2.  $s \in \underline{\mathcal{T}} \wedge s \rightarrow_{\underline{\beta}} t \Rightarrow t \in \underline{\mathcal{T}}$

**Übung 1.6.3** *Beweisen Sie obiges Lemma.*

**Theorem 1.6.4** *Seien  $s, t_1, \dots, t_n \in \underline{\mathcal{T}}$ . Dann terminiert  $s[t_1/x_1, \dots, t_n/x_n]$  bzgl.  $\rightarrow_{\underline{\beta}}$ , falls alle  $t_i$  terminieren.*

Beweis: mit Induktion über  $s$ . Setze  $[\sigma] := [t_1/x_1, \dots, t_n/x_n]$ .

1.  $s$  ist Konstante: klar
2.  $s$  ist Variable:
  - $\forall i. s \neq x_i$ : klar
  - $s = x_i$ : klar, weil  $t_i$  terminiert
3.  $s = (s_1 s_2)$ :  
 $s[\sigma] = (s_1[\sigma])(s_2[\sigma])$  terminiert, weil  $s_i[\sigma]$  terminiert (Ind.-Hyp.), und weil wegen Lemma 1.6.2  $s_1[\sigma] \rightarrow_{\underline{\beta}}^* \underline{\lambda}x.t$  unmöglich ist, da  $s_1[\sigma] \in \underline{\mathcal{T}}$  aber  $\underline{\lambda}x.t \notin \underline{\mathcal{T}}$ .
4.  $s = \lambda x.t$ :  $s[\sigma] = \lambda x.(t[\sigma])$  terminiert, da  $t[\sigma]$  terminiert (Ind.-Hyp.).
5.  $s = (\lambda x.t)u$ :  
 $s[\sigma] = (\lambda x.(t[\sigma]))(u[\sigma])$ , wobei  $t[\sigma]$  und  $u[\sigma]$  terminieren (Ind.-Hyp.). Jede unendliche Reduktion müßte wie folgt aussehen:

$$s[\sigma] \rightarrow_{\underline{\beta}}^* (\underline{\lambda}x.t') u' \rightarrow_{\underline{\beta}} t'[u'/x] \rightarrow_{\underline{\beta}} \dots$$

Aber: Da  $u[\sigma]$  terminiert und  $u[\sigma] \rightarrow_{\underline{\beta}}^* u'$ , muss auch  $u'$  terminieren. Da  $t[\sigma] \rightarrow_{\underline{\beta}}^* t'$  gilt auch

$$\underbrace{t[\sigma, u'/x]}_{\text{terminiert nach Ind.-Hyp., da } \sigma \text{ und } u' \text{ terminieren}} \rightarrow_{\underline{\beta}}^* \underbrace{t'[u'/x]}_{\text{muß also auch terminieren}}$$

$\Rightarrow$  Widerspruch zur Annahme, es gäbe eine unendliche Reduktion. □

**Korollar 1.6.5**  $\rightarrow_{\underline{\beta}}$  terminiert für alle Terme in  $\underline{\mathcal{T}}$ .

Länge der Reduktionssequenz: höchstens exponentiell in der Größe des Eingabeterms.

**Theorem 1.6.6**  $\rightarrow_{\underline{\beta}}$  ist konfluent.

Beweis:  $\rightarrow_{\underline{\beta}}$  ist lokal konfluent. (Verwende Termination und Newmanns Lemma.)  $\square$

Zusammenhang zwischen  $\rightarrow_{\underline{\beta}}$  und der parallelen Reduktion  $>$ :

**Theorem 1.6.7** Sei  $|\underline{s}|$  die unmarkierte Version von  $\underline{s} \in \underline{\mathcal{T}}$ . Dann gilt

$$s > t \iff \exists \underline{s} \in \underline{\mathcal{T}}. \underline{s} \rightarrow_{\underline{\beta}}^* t \wedge |\underline{s}| = s$$

## 1.7 $\lambda$ -Kalkül als „Programmiersprache“

### 1.7.1 Datentypen

- bool:

true, false, if mit  $\text{if true } x y \rightarrow_{\beta}^* x$   
und  $\text{if false } x y \rightarrow_{\beta}^* y$

wird realisiert durch

$$\begin{aligned} \text{true} &= \lambda xy.x \\ \text{false} &= \lambda xy.y \\ \text{if} &= \lambda zxy.z x y \end{aligned}$$

- Paare:

fst, snd, pair mit  $\text{fst}(\text{pair } x y) \rightarrow_{\beta}^* x$   
und  $\text{snd}(\text{pair } x y) \rightarrow_{\beta}^* y$

wird realisiert durch

$$\begin{aligned} \text{fst} &= \lambda p.p \text{ true} \\ \text{snd} &= \lambda p.p \text{ false} \\ \text{pair} &= \lambda xy.\lambda z.z x y \end{aligned}$$

Beispiel:

$$\begin{aligned} \text{fst}(\text{pair } x y) &\rightarrow_{\beta} \text{fst}(\lambda z.z x y) \rightarrow_{\beta} (\lambda z.z x y)(\lambda xy.x) \\ &\rightarrow_{\beta} (\lambda x y.x) x y \rightarrow_{\beta} (\lambda y.x) y \rightarrow_{\beta} x \end{aligned}$$

- **nat** (Church-Numerale):

$$\begin{aligned} \underline{0} &= \lambda f. \lambda x. x \\ \underline{1} &= \lambda f. \lambda x. f x \\ \underline{2} &= \lambda f. \lambda x. f(f x) \\ &\vdots \\ \underline{n} &= \lambda f. \lambda x. f^n(x) = \lambda f. \lambda x. \underbrace{f(f(\dots f(x)\dots))}_{n\text{-mal}} \end{aligned}$$

Arithmetik:

$$\begin{aligned} \text{succ} &= \lambda n. \lambda f x. f(n f x) \\ \text{add} &= \lambda m n. \lambda f x. m f(n f x) \\ \text{iszero} &= \lambda n. n(\lambda x. \text{false}) \text{true} \end{aligned}$$

damit:

$$\begin{aligned} \text{add } \underline{n} \ \underline{m} &\xrightarrow{2} \lambda f x. \underline{n} f(\underline{m} f x) \xrightarrow{2} \lambda f x. \underline{n} f(f^m(x)) \\ &\xrightarrow{2} \lambda f x. f^n(f^m(x)) = \lambda f x. f^{n+m}(x) = \underline{n+m} \end{aligned}$$

### Übung 1.7.1

1. Listen im  $\lambda$ -Kalkül: Finde  $\lambda$ -Terme für **nil**, **cons**, **hd**, **tl**, **null** mit

$$\begin{aligned} \text{null nil} &\rightarrow^* \text{true} & \text{hd}(\text{cons } x \ l) &\rightarrow^* x \\ \text{null}(\text{cons } x \ l) &\rightarrow^* \text{false} & \text{tl}(\text{cons } x \ l) &\rightarrow^* l \end{aligned}$$

*Hinweis: Benutze Paare.*

2. Finde **mult** mit  $\text{mult } \underline{m} \ \underline{n} \xrightarrow{*} \underline{m * n}$   
und **expt** mit  $\text{expt } \underline{m} \ \underline{n} \xrightarrow{*} \underline{m^n}$
3. Schwierig: Finde **pred** mit  $\text{pred } \underline{m+1} \xrightarrow{*} \underline{m}$  und  $\text{pred } \underline{0} \xrightarrow{*} \underline{0}$

### 1.7.2 Rekursive Funktionen

Gegeben eine rekursive Funktion  $f(x) = e$  suchen wir eine nicht-rekursive Darstellung  $f = t$ .  
Merke:  $f(x) = e$  ist keine Definition im mathematischen Sinne sondern nur eine (nicht eindeutig) charakterisierende Eigenschaft.

$$\begin{aligned} f(x) &= e \\ \Rightarrow f &= \lambda x. e \\ \Rightarrow f &=_{\beta} (\lambda f. \lambda x. e) f \\ \Rightarrow f &\text{ ist Fixpunkt von } F := \lambda f x. e, \text{ d.h. } f =_{\beta} F f \end{aligned}$$

Sei **fix** ein Fixpunktoperator, d.h.  $\text{fix } t =_{\beta} t(\text{fix } t)$  für alle Terme  $t$ . Dann kann  $f$  nicht-rekursiv definiert werden als

$$f := \text{fix } F$$

Rekursives und nicht-rekursives  $f$  verhalten sich gleich:

1. rekursiv:

$$f s = (\lambda x. e) s \rightarrow_{\beta} e[s/x]$$

2. nicht-rekursiv:

$$f s = \mathbf{fix} F s =_{\beta} F (\mathbf{fix} F) s = F f s \xrightarrow{\beta^2} e[f/f, s/x] = e[s/x]$$

Beispiel:

$$\begin{aligned} \mathbf{add} m n &= \mathbf{if} (\mathbf{zero} m) n (\mathbf{add} (\mathbf{pred} m) (\mathbf{succ} n)) \\ \mathbf{add} &:= \mathbf{fix} \underbrace{(\lambda \mathbf{add}. \lambda m n. \mathbf{if} (\mathbf{zero} m) n (\mathbf{add} (\mathbf{pred} m) (\mathbf{succ} n)))}_{F} \end{aligned}$$

$$\begin{aligned} \mathbf{add} \underline{1} \underline{2} &= \mathbf{fix} F \underline{1} \underline{2} \\ &=_{\beta} F (\mathbf{fix} F) \underline{1} \underline{2} \\ &\xrightarrow{\beta^3} \mathbf{if} (\mathbf{iszero} \underline{1}) \underline{2} (\mathbf{fix} F (\mathbf{pred} \underline{1}) (\mathbf{succ} \underline{2})) \\ &\xrightarrow{\beta^*} \mathbf{fix} F \underline{0} \underline{3} \\ &=_{\beta} F (\mathbf{fix} F) \underline{0} \underline{3} \\ &\xrightarrow{\beta^3} \mathbf{if} (\mathbf{iszero} \underline{0}) \underline{3} (\dots) \\ &\xrightarrow{\beta^*} \underline{3} \end{aligned}$$

Merke: es gilt sogar  $\mathbf{add} \underline{1} \underline{2} \xrightarrow{\beta^*} \underline{3}$ . Warum?

Wir zeigen nun, daß sich  $\mathbf{fix}$ , d.h. der Fixpunktoperator, im reinen  $\lambda$ -Kalkül definieren läßt. Zwei der bekanntesten Lösungen sind:

**Church:**  $V_f := \lambda x. f(x x)$  und  $Y := \lambda f. V_f V_f$

$Y$  heißt „Church’scher Fixpunktoperator“.

$$Y t \rightarrow_{\beta} V_t V_t \rightarrow_{\beta} t(V_t V_t) \leftarrow_{\beta} t((\lambda f. V_f V_f)t) = t(Y t)$$

Also:  $Y t =_{\beta} t(Y t)$

**Turing:**  $A := \lambda x f. f(x x f)$  und  $\Theta := A A \rightarrow_{\beta} \lambda f. f(A A f)$ . Damit gilt

$$\Theta t = A A t \rightarrow_{\beta} (\lambda f. f(A A f))t \rightarrow_{\beta} t(A A t) = t(\Theta t)$$

Also:  $\Theta t \xrightarrow{\beta^*} t(\Theta t)$

### Berechenbare Funktionen auf $\mathbb{N}$ :

**Definition 1.7.2** Eine (evtl. partiell) Funktion  $f : \mathbb{N}^n \rightarrow \mathbb{N}$  ist  **$\lambda$ -definierbar**, wenn es einen geschlossenen reinen  $\lambda$ -Term (ohne Konstanten, ohne freie Variablen!) gibt mit

1.  $t \underline{m_1} \dots \underline{m_n} \xrightarrow{\beta^*} \underline{m}$ , falls  $f(m_1, \dots, m_n) = m$
2.  $t \underline{m_1} \dots \underline{m_n}$  hat keine  $\beta$ -Normalform, falls  $f(m_1, \dots, m_n)$  undefiniert ist.

**Theorem 1.7.3** *Alle Turingmaschinen-berechenbaren (Registermaschinen-berechenbaren, while-berechenbaren,  $\mu$ -rekursiven) Funktionen sind  $\lambda$ -definierbar, und umgekehrt.*



# Kapitel 2

## Kombinatorische Logik (CL)

Schlagwort: „variablenfreies Programmieren“

Terme:

$$X ::= \underbrace{x}_{\text{Variablen}} \mid \underbrace{S \mid K \mid I \mid \dots}_{\text{Konstanten}} \mid X_1 X_2 \mid (X)$$

Applikation assoziiert wie immer nach links:  $X Y Z = (X Y) Z$

**Kombinatoren** sind variablenfreie Terme. (genauer: Sie enthalten nur S und K.)

Rechenregeln für die **schwache Reduktion** (**weak reduction**,  $\rightarrow_w$ ):

$$\begin{aligned} I X &\rightarrow_w X \\ K X Y &\rightarrow_w X \\ S X Y Z &\rightarrow_w (X Z)(Y Z) \\ X \rightarrow_w X' &\Rightarrow X Y \rightarrow_w X' Y \quad \wedge \quad Y X \rightarrow_w Y X' \end{aligned}$$

Beispiele:

1.  $S K X Y \rightarrow_w K Y (X Y) \rightarrow_w Y$
2.  $S K K X \rightarrow_w K X (K X) \rightarrow_w X$

Wir sehen, daß sich  $S K K$  und  $I$  gleich verhalten. Daher ist  $I$  theoretisch entbehrlich, aber praktisch praktisch.

**Theorem 2.0.4**  $\rightarrow_w$  ist konfluent.

Beweismöglichkeiten:

1. parallele Reduktion (einfacher als bei  $\rightarrow_\beta$ )
2. „Jedes orthogonale Termersetzungssystem ist konfluent.“

Das Termersetzungssystem  $\rightarrow_w$  ist nicht terminierend:

**Übung 2.0.5** Finden Sie einen Kombinator  $X$  mit  $X \rightarrow_w^+ X$ .

**Übung 2.0.6** Finden Sie Kombinatoren  $A$ ,  $W$ ,  $B$  mit

$$\begin{aligned} A X &\rightarrow_w^* X X \\ W X Y &\rightarrow_w^* X Y Y \\ B X Y Z &\rightarrow_w^* X (Y Z) \end{aligned}$$

**Theorem 2.0.7** Wenn ein CL-Term eine Normalform besitzt, dann kann man diese finden, indem man immer möglichst weit links reduziert.  $\square$

Beweisidee: orthogonales Termersetzungssystem, und in jeder Regel stehen auf der linken Seite alle Funktionssymbole stets links von den Variablen.  $\square$

## 2.1 Beziehung zwischen $\lambda$ -Kalkül und CL

Übersetzung von  $\lambda$ -Termen in CL-Terme:

$$\begin{aligned} (-)_{\text{CL}} : \quad \lambda\text{-Terme} &\rightarrow \text{CL-Terme} \\ (x)_{\text{CL}} &= x \\ (s t)_{\text{CL}} &= (s)_{\text{CL}} (t)_{\text{CL}} \\ (\lambda x.s)_{\text{CL}} &= \lambda^*x.(s)_{\text{CL}} \end{aligned}$$

Hilfsfunktion  $\lambda^*$ :  $\text{Vars} \times \text{CL-Terme} \rightarrow \text{CL-Terme}$

$$\begin{aligned} \lambda^*x.x &= \mathbf{I} \\ \lambda^*x.X &= \mathbf{K} X && \text{falls } x \notin FV(X) \\ \lambda^*x.(X Y) &= \mathbf{S}(\lambda^*x.X)(\lambda^*x.Y) && \text{falls } x \in FV(X Y) \end{aligned}$$

**Lemma 2.1.1**  $(\lambda^*x.X) Y \rightarrow_w^* X[Y/x]$

Beweis: mit struktureller Induktion über  $X$

- falls  $X \equiv x$ :  $(\lambda^*x.X)Y = \mathbf{I} Y \rightarrow_w Y = X[Y/x]$
- falls  $x$  in  $X$  nicht frei:  $(\lambda^*x.X)Y = \mathbf{K} X Y \rightarrow_w X = X[Y/x]$
- falls  $X \equiv U V$  und  $x \in FV(X)$ :

$$\begin{aligned} (\lambda^*x.(U V))Y &= \mathbf{S}(\lambda^*x.U)(\lambda^*x.V)Y \rightarrow_w ((\lambda^*x.U)Y)((\lambda^*x.V)Y) \\ (\text{Ind.-Hyp.}) \rightarrow_w (U[Y/x])(V[Y/x]) &= X[Y/x] \end{aligned}$$

Übersetzung von CL-Termen in  $\lambda$ -Terme:

$$\begin{aligned} (-)_{\lambda} : \quad \text{CL-Terme} &\rightarrow \lambda\text{-Terme} \\ (x)_{\lambda} &= x \\ (\mathbf{K})_{\lambda} &= \lambda xy.x \\ (\mathbf{S})_{\lambda} &= \lambda xyz.x z (y z) \\ (X Y)_{\lambda} &= (X)_{\lambda} (Y)_{\lambda} \end{aligned}$$

**Theorem 2.1.2**  $((s)_{\text{CL}})_\lambda \xrightarrow{\beta^*} s$

Beweis: mit struktureller Induktion über  $s$ :

1.  $((a)_{\text{CL}})_\lambda = a$
2. Mit Ind.-Hyp.:  $((t u)_{\text{CL}})_\lambda = ((t)_{\text{CL}} (u)_{\text{CL}})_\lambda = ((t)_{\text{CL}})_\lambda ((u)_{\text{CL}})_\lambda \xrightarrow{\beta^*} t u$
3. Mit Lemma 2.1.3 und Ind.-Hyp.:  $((\lambda x.t)_{\text{CL}})_\lambda = (\lambda^* x.(t)_{\text{CL}})_\lambda \xrightarrow{\beta^*} \lambda x.((t)_{\text{CL}})_\lambda \xrightarrow{\beta^*} \lambda x.t \quad \square$

**Lemma 2.1.3**  $(\lambda^* x.P)_\lambda \xrightarrow{\beta^*} \lambda x.(P)_\lambda$

Beweis: zur Übung.

**Korollar 2.1.4**  $S$  und  $K$  reichen aus, um alle  $\lambda$ -Terme darzustellen:  $\forall s \exists X. (X)_\lambda =_\beta s$

Beweis: setze  $X := (s)_{\text{CL}}$

**Übung 2.1.5** Zeigen Sie, daß auch  $B$ ,  $C$ ,  $K$  und  $W$  ausreichen, um alle  $\lambda$ -Terme darzustellen (Hierbei:  $CXYZ \rightarrow_w XZY$ ). Kann man  $K$  auch weglassen?

**Theorem 2.1.6**  $((X)_\lambda)_{\text{CL}} =_{\text{w,ext}} X$  wobei  $=_{\text{w}} := \leftrightarrow_w^*$  und

$$(\text{ext}) : \frac{\forall x. X x =_{\text{w,ext}} Y x}{X =_{\text{w,ext}} Y} \quad (\text{Extensionalität})$$

**Theorem 2.1.7**  $X \rightarrow_w Y \Rightarrow (X)_\lambda \rightarrow_{\beta^*} (Y)_\lambda$

Beweis:

$$\begin{array}{ccc} C[K X Y] & \xrightarrow{\quad w \quad} & C[X] \\ \downarrow \lambda & & \downarrow \lambda \\ C_\lambda[(\lambda xy.x) X_\lambda Y_\lambda] & \xrightarrow[\beta^*]{} & C_\lambda[X_\lambda] \end{array}$$

analog für  $S$

$\square$

aber: Im allgemeinen folgt aus  $s \rightarrow_\beta t$  nicht  $(s)_{\text{CL}} \rightarrow_w^* (t)_{\text{CL}}$ .

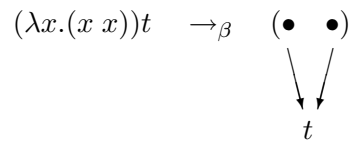
Aufgabe: Man finde ein Gegenbeispiel.

## 2.2 Implementierungsaspekte

Probleme bei der effektiven Implementierung von  $\rightarrow_\beta$ :

- Naive Implementierung durch Kopieren ist sehr ineffizient!
- Kopieren ist manchmal notwendig  
Beispiel: sei  $t := \lambda x.(f x)$ .

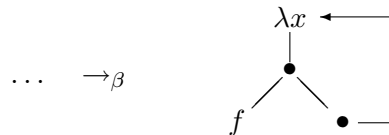




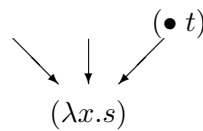
mit Kopie:

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

ohne Kopie entsteht ein zyklischer Term:



im allgemeinen:



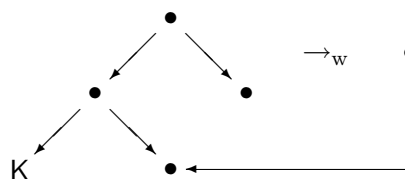
Bei  $\beta$ -Reduktion von  $(\bullet t)$  Kopie von  $s$  nötig!

- $\alpha$ -Konversion nötig

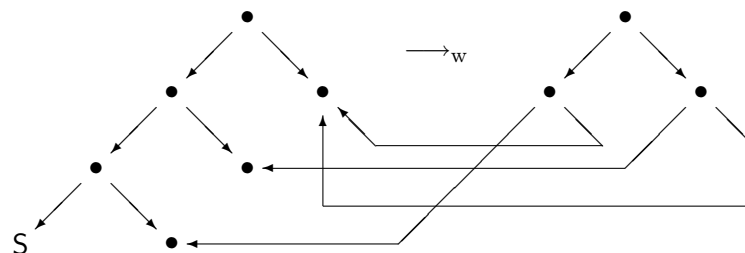
### Graphreduktion

Eine radikale Lösung ist die Übersetzung nach CL, weil  $\rightarrow_w$  auf Graphen ohne Kopieren implementierbar ist:

1.  $(K x) y \rightarrow_w x$ :



2.  $S x y z \rightarrow_w x z (y z)$ :



Ein Problem dabei ist, daß  $(\cdot)_{CL}$  Terme sehr aufblasen kann, was allerdings zum Teil durch Optimierung ausgeglichen werden kann (S und K durch optimierte Kombinatoren ersetzen). Allerdings geht die Struktur der  $\lambda$ -Terme immer verloren.

### De Bruijn Notation

Eine zweite Lösung sind die sogenannten de Bruijn-Indizes:

$$\lambda x. \lambda y. (x z) \cong \lambda \lambda (1 2)$$

Gebundene Variablen werden Indizes, die angeben, wieviele  $\lambda$ s man durchlaufen muß, um an die Bindungsstelle zu kommen. Die Syntax ist daher

$$t ::= i \mid \lambda t \mid (t_1 t_2)$$

Beispiele:

$$\begin{aligned} \lambda x. x &\cong \lambda 0 \\ \lambda x. (y z) &\cong \lambda (1 2) \end{aligned}$$

De Bruijn Terme sind schwer lesbar, da dieselbe gebundene Variable mit verschiedenen Indizes auftauchen kann. Beispiel:

$$\lambda x. x (\lambda y. y x) \cong \lambda (0 (\lambda (0 1)))$$

Aber:  $\alpha$ -äquivalente Terme sind in dieser Notation identisch!

Wir betrachten nun  $\beta$ -Reduktion und Substitution. Beispiele:

$$\begin{aligned} \lambda x. (\lambda y. \lambda z. y) x &\rightarrow_{\beta} \lambda x. \lambda z. x \\ \lambda ((\lambda \lambda 1) 0) &\rightarrow_{\beta} \lambda \lambda 1 \end{aligned}$$

Im allgemeinen:

$$(\lambda s) t \rightarrow_{\beta} s[t/0]$$

wobei  $s[t/i]$  bedeutet: Ersetze  $i$  in  $s$  durch  $t$ , wobei freie Variablen in  $t$  eventuell inkrementiert werden müssen, und dekrementiere alle freien Variablen  $\geq i$  in  $s$  um 1. Formal:

$$\begin{aligned} j[t/i] &= \text{if } i = j \text{ then } t \text{ else if } j > i \text{ then } j - 1 \text{ else } j \\ (s_1 s_2)[t/i] &= (s_1[t/i])(s_2[t/i]) \\ (\lambda s)[t/i] &= \lambda (s[\text{lift}(t, 0)/i + 1]) \end{aligned}$$

wobei  $\text{lift}(t, i)$  bedeutet: inkrementiere in  $t$  alle Variablen  $\geq i$  um 1. Formal:

$$\begin{aligned} \text{lift}(j, i) &= \text{if } j \geq i \text{ then } j + 1 \text{ else } j \\ \text{lift}((s_1 s_2), i) &= (\text{lift}(s_1, i))(\text{lift}(s_2, i)) \\ \text{lift}(\lambda s, i) &= \lambda (\text{lift}(s, i + 1)) \end{aligned}$$

Beispiel:

$$\begin{aligned} (\lambda x y. x) z &\cong (\lambda \lambda 1) 0 \rightarrow_{\beta} (\lambda 1)[0/0] = \lambda (1[\text{lift}(0, 0)/1]) = \\ &= \lambda (1[1/1]) = \lambda 1 \cong \lambda y. z \end{aligned}$$



# Kapitel 3

## Typisierte Lambda-Kalküle

Warum Typen ?

1. um Inkonsistenzen zu vermeiden

Gottlob Frege (Prädikatenlogik,  $\approx 1879$ ):

erlaubt uneingeschränkte Quantifizierung über Prädikate

Russel (1901): Paradox  $\{X \mid X \notin X\}$

Whitehead & Russel: Principia Mathematica (1910–1913)

Typen verbieten  $X \in X$

Church (1930): untypisierter  $\lambda$ -Kalkül als Logik

**True**, **False**,  $\wedge$ , ... sind  $\lambda$ -Terme

$\{x \mid P\} \equiv \lambda x.P$        $x \in M \equiv Mx$

Inkonsistenz:  $R := \lambda x.\text{not}(xx) \Rightarrow RR =_{\beta} \text{not}(RR)$

Church (1940): „Simply Typed  $\lambda$ -Calculus“ erlaubt  $xx$  nicht.

2. um Programmierfehler zu vermeiden.

Klassifikation von Typsystemen:

**monomorph** : Jeder Bezeichner hat genau einen Typ.

**polymorph** : Ein Bezeichner kann mehrere Typen haben.

**statisch** : Typkorrektheit wird zur Übersetzungszeit überprüft.

**dynamisch** : Typkorrektheit wird zur Laufzeit überprüft.

	statisch	dynamisch
monomorph	Pascal	
polymorph	ML, Haskell (C++,) Java	Lisp, Smalltalk

3. um Spezifikationen durch Typen auszudrücken

Methode: abhängige Typen

Beispiel:  $\text{mod}: \text{nat} \times m:\text{nat} \rightarrow \{k \mid 0 \leq k < m\}$

Resultattyp hängt vom Eingabewert ab („Typtheorie“)

### 3.1 Einfach typisierter $\lambda$ -Kalkül ( $\lambda^{\rightarrow}$ )

Kern jeder (funktionalen) Programmiersprache

Typen:

$$\tau ::= \underbrace{\text{bool} \mid \text{nat} \mid \text{int} \mid \dots}_{\text{Basistypen}} \mid \tau_1 \rightarrow \tau_2$$

Konvention:  $\rightarrow$  assoziiert nach rechts:

$$\tau_1 \rightarrow \tau_2 \rightarrow \tau_3 \equiv \tau_1 \rightarrow (\tau_2 \rightarrow \tau_3)$$

Terme:

1. implizit typisiert: Terme wie im reinen untypisierten  $\lambda$ -Kalkül, aber jede Variable hat (implizit) einen eindeutigen Typ.
2. explizit typisierte Terme:  $t ::= x \mid (t_1 t_2) \mid \lambda x : \tau. t$

In beiden Fällen handelt es sich um sogenannte „rohe“ typisierte Terme, die nicht notwendigerweise typkorrekt sind, z.B.  $\lambda x : \text{int}.(x x)$ .

#### 3.1.1 Typüberprüfung für explizit typisierte Terme

Ziel ist die Herleitung von Aussagen der Form  $\Gamma \vdash t : \tau$ , d.h. im Kontext  $\Gamma$  hat  $t$  den Typ  $\tau$ . Hierbei ist  $\Gamma$  eine endliche Funktion von Variablen auf Typen. Schreibweise:  $[x_1 : \tau_1, \dots, x_n : \tau_n]$ . Die Notation  $\Gamma[x : \tau]$  bedeutet das Überschreiben von  $\Gamma$  mit der Abbildung  $x \mapsto \tau$ . Formal:

$$(\Gamma[x : \tau])(y) = \begin{cases} \tau & \text{falls } x = y \\ \Gamma(y) & \text{sonst} \end{cases}$$

Regeln:

$$\frac{\Gamma(x) \text{ ist definiert}}{\Gamma \vdash x : \Gamma(x)} \text{ (Var)} \qquad \frac{\Gamma \vdash t_1 : \tau_1 \rightarrow \tau_2 \quad \Gamma \vdash t_2 : \tau_1}{\Gamma \vdash (t_1 t_2) : \tau_2} \text{ (App)} \qquad \frac{\Gamma[x : \tau] \vdash t : \tau'}{\Gamma \vdash \lambda x : \tau. t : \tau \rightarrow \tau'} \text{ (Abs)}$$

Regeln sind Algorithmus zur Typüberprüfung: Regeln werden wie in Prolog rückwärts angewandt (terminiert und ist deterministisch).

Beispiele:

- Eine einfache Herleitung:

$$\frac{\Gamma[x : \tau] \vdash x : \tau}{\Gamma \vdash \lambda x : \tau. x : \tau \rightarrow \tau}$$

- Nicht jeder Term hat einen Typ: Es gibt keinen Kontext  $\Gamma$  und Typen  $\tau$  und  $\tau'$  so daß  $\Gamma \vdash \lambda x : \tau.(x x) : \tau'$ , denn:

$$\frac{\frac{\tau = \tau_2 \rightarrow \tau_1}{\Gamma[x : \tau] \vdash x : \tau_2 \rightarrow \tau_1} \quad \frac{\tau = \tau_2}{\Gamma[x : \tau] \vdash x : \tau_2}}{\Gamma[x : \tau] \vdash (x x) : \tau_1} \quad \tau' = \tau \rightarrow \tau_1}{\Gamma \vdash \lambda x : \tau.(x x) : \tau'}$$

$\Rightarrow$  Widerspruch:  $\neg \exists \tau_1, \tau_2 : \tau_2 \rightarrow \tau_1 = \tau_2$

**Definition 3.1.1**  $t$  ist **typkorrekt** (bzgl.  $\Gamma$ ), falls es  $\tau$  gibt mit  $\Gamma \vdash t : \tau$

**Lemma 3.1.2** Der Typ eines typkorrekten Termes ist eindeutig bestimmt (bzgl. eines festen Kontextes  $\Gamma$ ).

Wir haben es hier also mit einem monomorphen Typsystem zu tun.

**Theorem 3.1.3** Jeder Teilterm eines typkorrekten Termes ist typkorrekt.

Beweis: mit Induktion über die Terme

**Theorem 3.1.4 (Subject Reduction)**  $\Gamma \vdash t : \tau \wedge t \rightarrow_{\beta} t' \Rightarrow \Gamma \vdash t' : \tau$  („keine Typfehler zur Laufzeit“)

Dies gilt nicht für  $\beta$ -Expansion:

$$[x : \text{int}, y : \tau] \vdash y : \tau$$

und

$$y : \tau \leftarrow_{\beta} (\lambda z : \text{bool}.y) x$$

aber:  $(\lambda z : \text{bool}.y) x$  ist nicht typkorrekt!

**Theorem 3.1.5**  $\rightarrow_{\beta}$  ( $\rightarrow_{\eta}, \rightarrow_{\beta\eta}$ ) auf typkorrekten Termen ist konfluent.

Dies gilt nicht für alle rohen Terme:

$$\lambda x : \text{int}.(\lambda y : \text{bool}.y) x \begin{array}{l} \nearrow_{\beta} \lambda x : \text{int}.x \\ \searrow_{\eta} \lambda y : \text{bool}.y \end{array}$$

**Theorem 3.1.6**  $\rightarrow_{\beta}$  terminiert auf typkorrekten Termen.

Der Beweis wird in Abschnitt 3.2 besprochen.

Intuition: Selbstapplikation und damit Rekursion sind ausgeschlossen.

Dies hat folgende positiv Konsequenz:

**Korollar 3.1.7**  $=_{\beta}$  ist für typkorrekte Terme entscheidbar.

Allerdings gibt es typkorrekte Terme  $s$ , so daß die kürzeste Reduktion von  $s$  in eine Normalform die Länge

$$\underbrace{2^{2^{2^{\dots^2}}}}_{\text{Größe von } s}$$

hat. Diese pathologischen Beispiele sind allerdings in der Praxis sehr selten.

Die negative Konsequenz aus Theorem 3.1.6 ist

**Korollar 3.1.8** Nicht alle berechenbaren Funktionen sind als typkorrekte  $\lambda^{\rightarrow}$ -Terme darstellbar. (sogar ziemlich wenige: Polynome + Fallunterscheidung)

Frage: warum sind typisierte funktionale Sprachen trotzdem Turing-mächtig?

**Theorem 3.1.9** Jede berechenbare Funktion ist als geschlossener typkorrekter  $\lambda^{\rightarrow}$ -Term darstellbar, der als einzige Konstanten Fixpunktoperatoren  $Y_{\tau} : (\tau \rightarrow \tau) \rightarrow \tau$  enthält, für die die Reduktionsregel  $Y_{\tau} t \rightarrow t(Y_{\tau} t)$  gilt.

Beweis:

1. Datentypen durch typkorrekte  $\lambda^{\rightarrow}$ -Terme darstellbar
2. Rekursion mit  $Y_{\tau}$

### 3.2 Termination von $\rightarrow_\beta$

Der Beweis in diesem Abschnitt orientiert sich sehr stark an dem kombinatorischen Beweis von Loader [Loa98]. Ein allgemeinerer Beweis, der auf Tate zurückgeht, findet sich ebenfalls bei Loader und in der Standard-Literatur, z.B. [HS86, GLT90, Han04].

Der Einfachheit halber arbeiten wir mit implizit typisierten oder sogar ganz untypisierten Termen.

**Definition 3.2.1** Sei  $t$  ein beliebiger  $\lambda$ -Term. Wir sagen, dass  $t$  (bzgl.  $\rightarrow_\beta$ ) **divergiert**, g.d.w. es eine unendliche Reduktionsfolge  $t \rightarrow_\beta t_1 \rightarrow_\beta t_2 \rightarrow_\beta \dots$  gibt. Wir sagen, dass  $t$  (bzgl.  $\rightarrow_\beta$ ) **terminiert**, g.d.w.  $t$  nicht divergiert; dann schreiben wir  $t \Downarrow$ .

Wir definieren zuerst eine Teilmenge  $T$  der untypisierten  $\lambda$ -Terme:

$$\frac{r_1, \dots, r_n \in T}{x r_1 \dots r_n \in T} (Var) \quad \frac{r \in T}{\lambda x.r \in T} (\lambda) \quad \frac{r[s/x] s_1 \dots s_n \in T \quad s \in T}{(\lambda x.r) s s_1 \dots s_n \in T} (\beta)$$

**Lemma 3.2.2**  $t \in T \Rightarrow t \Downarrow$

**Beweis** mit Induktion über die Ableitung von  $t \in T$  (“Regelinduktion”).

(*Var*) Aus  $r_1 \Downarrow, \dots, r_n \Downarrow$  folgt direkt  $(x r_1 \dots r_n) \Downarrow$  da  $x$  eine Variable ist.

( $\lambda$ ) Aus  $r \Downarrow$  folgt direkt  $(\lambda x.r) \Downarrow$ .

( $\beta$ ) Aus der I.H.  $(r[s/x] s_1 \dots s_n) \Downarrow$  folgt, dass  $r \Downarrow$  und  $s_i \Downarrow, i = 1, \dots, n$ . Falls  $(\lambda x.r) s s_1 \dots s_n$  divergierte, so müsste die unendliche Reduktionsfolge von der folgenden Form sein:

$$(\lambda x.r) s s_1 \dots s_n \rightarrow_\beta^* (\lambda x.r') s' s'_1 \dots s'_n \rightarrow_\beta r'[s'/x] s'_1 \dots s'_n \rightarrow_\beta \dots$$

da  $r, s$  (nach I.H.) und alle  $s_i$  terminieren. Da aber ebenfalls  $r[s/x] s_1 \dots s_n \rightarrow_\beta^* r'[s'/x] s'_1 \dots s'_n$  gilt, widerspricht dies der Termination von  $r[s/x] s_1 \dots s_n$ , und somit kann  $(\lambda x.r) s s_1 \dots s_n$  nicht divergieren.  $\square$

Man kann auch die Umkehrung zeigen. Damit enthält  $T$  genau die terminierenden Terme.

Wir wollen nun zeigen, dass  $T$  unter Applikation und Substitution von typkorrekten Termen abgeschlossen ist. Dies geschieht mit Induktion über die Typen. Da wir mit impliziert typisierten Termen arbeiten, entfällt der Kontext  $\Gamma$  und wir schreiben einfach  $t : \tau$ .

Wir nennen einen Typ  $\tau$  **applikativ** g.d.w. für alle  $t, r$  und  $\sigma$  gilt

$$\frac{t : \tau \rightarrow \sigma \quad r : \tau \quad t \in T \quad r \in T}{tr \in T}$$

Wir nennen  $\tau$  **substitutiv** g.d.w. für alle  $s, r$  und  $\sigma$  gilt

$$\frac{s : \sigma \quad r : \tau \quad x : \tau \quad s \in T \quad r \in T}{s[r/x] \in T}$$

**Lemma 3.2.3** *Jeder substitutive Typ ist applikativ.*

**Beweis** Sei  $\tau$  substitutiv. Wir zeigen mit Induktion über die Herleitung von  $t \in T$ , dass  $\tau$  applikativ ist.

(*Var*) Falls  $t = x r_1 \dots r_n$  und alle  $r_i \in T$ , dann folgt mit (*Var*) ebenfalls  $tr = x r_1 \dots r_n r \in T$  da  $r \in T$  nach Voraussetzung.

( $\lambda$ ) Falls  $t = \lambda x.s$  und  $s \in T$ , dann gilt  $s[r/x] \in T$ , da  $\tau$  substitutiv ist, und damit folgt mit ( $\beta$ ), dass  $tr = (\lambda x.s)r \in T$  da  $r \in T$  nach Voraussetzung.

( $\beta$ ) Falls  $t = (\lambda x.r) s s_1 \dots s_n$  und  $r[s/x] s_1 \dots s_n \in T$  und  $s \in T$ , dann gilt nach I.H.  $r[s/x] s_1 \dots s_n r \in T$ . Da  $s \in T$ , folgt mit ( $\beta$ ), dass  $tr = (\lambda x.r) s s_1 \dots s_n r \in T$ .  $\square$

**Lemma 3.2.4** Sei  $\tau = \tau_1 \rightarrow \dots \rightarrow \tau_k \rightarrow \tau'$ , wobei  $\tau'$  kein Funktionstyp ist. Falls alle  $\tau_i$  applikativ sind, so ist  $\tau$  substitutiv.

**Beweis** mit Induktion über die Herleitung von  $s \in T$ .

(Var) Falls  $s = y s_1 \dots s_n$  und alle  $s_i \in T$ , so gilt  $s_i[r/x] \in T$  nach I.H.,  $i = 1, \dots, n$ . Falls  $x \neq y$ , so gilt mit (Var)  $s[r/x] = y(s_1[r/x]) \dots (s_n[r/x]) \in T$ . Falls  $x = y$ , so gilt  $y : \tau$  und somit  $s_i : \tau_i$  und auch  $s_i[r/x] : \tau_i$ ,  $i = 1, \dots, n$ . Da alle  $\tau_i$  applikativ sind, gilt  $s[r/x] = r(s_1[r/x]) \dots (s_n[r/x]) \in T$ .

( $\lambda$ ) Falls  $s = \lambda y. u$  mit  $u \in T$ , dann gilt mit I.H.  $u[r/x] \in T$ , woraus  $s[r/x] = \lambda y. (u[r/x]) \in T$  mit ( $\lambda$ ) folgt.

( $\beta$ ) Falls  $s = (\lambda y. u) s_0 s_1 \dots s_n$  mit  $u[s_0/y] s_1 \dots s_n \in T$  und  $s_0 \in T$ , dann folgt  $s[r/x] = (\lambda y. (u[r/x]))(s_0[r/x]) \dots (s_n[r/x]) \in T$  mit ( $\beta$ ) da  $u[r/x][s_0[r/x]/y](s_1[r/x]) \dots (s_n[r/x]) = (u[s_0/y] s_1 \dots s_n)[r/x] \in T$  und  $s_0[r/x] \in T$  mit I.H.  $\square$

**Übung 3.2.5** Zeige, dass für typkorrekte  $s$  und  $t$  gilt:  $s \in T$  und  $t \in T$  impliziert  $st \in T$ .

**Theorem 3.2.6** Ist  $t$  typkorrekt, so gilt  $t \in T$ .

**Beweis** mit Induktion über die Herleitung des Typs von  $t$ . Ist  $t$  eine Variable, so gilt  $t \in T$  mit (Var). Falls  $t = \lambda x. r$ , so folgt  $t \in T$  mit ( $\lambda$ ) aus der I.H.  $r \in T$ . Falls  $t = r s$ , so folgt  $t \in T$  mit Übung 3.2.5 aus den I.H.  $r \in T$  und  $s \in T$ .  $\square$

Theorem 3.1.6 ist nun ein Korollar aus Theorem 3.2.6 und Lemma 3.2.2.

### 3.3 Typinferenz für $\lambda \rightarrow$

Typen:  $\tau ::= \text{bool} \mid \text{int} \mid \dots$  Basistypen  
 $\mid \alpha \mid \beta \mid \gamma \mid \dots$  Typvariablen  
 $\mid \tau_1 \rightarrow \tau_2$

Terme: untypisierte  $\lambda$ -Terme

Typinferenzregeln:

$$\Gamma \vdash x : \Gamma(x) \quad \frac{\Gamma \vdash t_1 : \tau_1 \rightarrow \tau_2 \quad \Gamma \vdash t_2 : \tau_1}{\Gamma \vdash (t_1 t_2) : \tau_2} \quad \frac{\Gamma[x : \tau_1] \vdash t : \tau_2}{\Gamma \vdash (\lambda x. t) : \tau_1 \rightarrow \tau_2}$$

Terme können verschiedene Typen haben (Polymorphie):

$$\lambda x. x : \alpha \rightarrow \alpha$$

$$\lambda x. x : \text{int} \rightarrow \text{int}$$

**Definition 3.3.1**  $\tau_1 \lesssim \tau_2 \Leftrightarrow \exists$  Substitution  $\Theta$  (von Typen für Typvariable) mit  $\tau_1 = \Theta(\tau_2)$  („ $\tau_2$  ist allgemeiner oder äquivalent  $\tau_1$ .“)

Beispiel:  $\text{int} \rightarrow \text{int} \lesssim \alpha \rightarrow \alpha \lesssim \beta \rightarrow \beta \lesssim \alpha \rightarrow \alpha$

**Theorem 3.3.2**  $\Gamma \vdash t : \tau \Rightarrow \exists \sigma. \Gamma \vdash t : \sigma \wedge \forall \tau'. \Gamma \vdash t : \tau' \Rightarrow \tau' \lesssim \sigma$   
 „Jeder typkorrekte Term hat einen allgemeinsten Typ.“

**Beweis:** Betrachte Regeln als Prolog-Programm. Prolog berechnet die allgemeinste Lösung. Die Regeln sind deterministisch, d.h. es gibt maximal eine Lösung.

Beispiel:



$$\begin{array}{l}
\Gamma \vdash \lambda x. \lambda y. (y x) : A \\
\text{falls } [x : B] \vdash \lambda y. (y x) : C \text{ und } A = B \rightarrow C \\
\text{falls } [x : B, y : D] \vdash (y x) : E \text{ und } C = D \rightarrow E \\
\text{falls } [x : B, y : D] \vdash y : F \rightarrow E \quad \text{und} \quad [x : B, y : D] \vdash x : F \\
\text{falls } D = F \rightarrow E \quad \text{und} \quad B = F
\end{array}$$

Also:  $A = B \rightarrow C = F \rightarrow (D \rightarrow E) = F \rightarrow ((F \rightarrow E) \rightarrow E)$

### 3.4 let-Polymorphismus

Terme:  $t ::= x \mid (t_1 t_2) \mid \lambda x. t \mid \text{let } x = t_1 \text{ in } t_2$

Semantik:  $\text{let } x = t_1 \text{ in } t_2 \equiv t_2[t_1/x]$

(wohldefiniert wegen Termination und Konfluenz von  $\rightarrow_\beta$ )

Beispiel:

$$\text{let } \underbrace{f = \lambda x. x}_{f : \forall \alpha. \underbrace{\alpha \rightarrow \alpha}_\tau} \text{ in pair } \underbrace{(f 0)}_{f : \tau[\text{int}/\alpha]} \underbrace{(f \text{true})}_{f : \tau[\text{bool}/\alpha]}$$

Allquantifizierte Typvariablen dürfen beliebig ersetzt werden.

$(\lambda f. \text{pair } (f 0) (f \text{true})) (\lambda x. x)$  ist semantisch äquivalent zu obigem **let**-Term, aber nicht typkorrekt, weil  $\lambda$ -gebundene Variablen keine allquantifizierten Typen haben.

Typen:  $\tau ::= \text{bool} \mid \dots \mid \alpha \mid \dots \mid \tau_1 \rightarrow \tau_2$

Typschemata:  $\sigma ::= \forall \alpha. \sigma \mid \tau$

Beispiele für Typschemata:

$\alpha, \text{int}, \forall \alpha. \alpha \rightarrow \alpha, \forall \alpha, \beta. \alpha \rightarrow \beta$

aber nicht  $(\forall \alpha. \alpha \rightarrow \alpha) \rightarrow \text{bool}$  (Der Allquantor tritt nicht ganz außen auf!)

Typinferenzregeln ( $\Gamma = [x_1 : \sigma_1, \dots, x_n : \sigma_n]$ ):

$$\begin{array}{c}
\frac{}{\Gamma \vdash x : \Gamma(x)} \text{ (Var)} \\
\frac{\Gamma \vdash t_1 : \tau_2 \rightarrow \tau \quad \Gamma \vdash t_2 : \tau_2}{\Gamma \vdash (t_1 t_2) : \tau} \text{ (App)} \\
\frac{\Gamma[x : \tau_1] \vdash t : \tau_2}{\Gamma \vdash (\lambda x. t) : \tau_1 \rightarrow \tau_2} \text{ (Abs)} \\
\frac{\Gamma \vdash t_1 : \sigma_1 \quad \Gamma[x : \sigma_1] \vdash t_2 : \sigma_2}{\Gamma \vdash \text{let } x = t_1 \text{ in } t_2 : \sigma_2} \text{ (Let)}
\end{array}$$

Quantorenregeln:

$$\begin{array}{c}
\frac{\Gamma \vdash t : \forall \alpha. \sigma}{\Gamma \vdash t : \sigma[\tau/\alpha]} \text{ (\forall Elim)} \\
\frac{\Gamma \vdash t : \sigma}{\Gamma \vdash t : \forall \alpha. \sigma} \text{ (\forall Intro)} \quad \text{falls } \alpha \notin FV(\Gamma)
\end{array}$$

wobei  $FV([x_1 : \sigma_1, \dots, x_n : \sigma_n]) = \bigcup_{i=1}^n FV(\sigma_i)$

Warum wird bei ( $\forall$ Intro) die Bedingung  $\alpha \notin FV(\Gamma)$  benötigt ?

Logik:  $x = 0 \vdash x = 0 \not\Rightarrow x = 0 \vdash \forall x. x = 0$

ML:  $\lambda x. \text{let } y = x \text{ in } y + (y 1)$  sollte nicht typkorrekt sein.

Herleitung hierfür, unter Verletzung der Bedingung:

$$\frac{\frac{[x : \alpha] \vdash x : \alpha}{[x : \alpha] \vdash x : \forall \alpha. \alpha} (\forall \text{Intro}) \quad [y : \forall \alpha. \alpha] \vdash y + (y \ 1) : \text{int}}{[x : \alpha] \vdash \text{let } y = x \text{ in } y + (y \ 1) : \text{int}}}{\lambda y. \text{let } y = x \text{ in } y + (y \ 1) : \alpha \rightarrow \text{int}}$$

Problem: Die Regeln liefern keinen Algorithmus, da Quantorenregeln nicht syntaxgesteuert, d.h. (fast) immer anwendbar sind.

Lösung: Integriere ( $\forall \text{Elim}$ ) mit ( $\text{Var}$ ) und ( $\forall \text{Intro}$ ) mit ( $\text{Let}$ ):

$$\frac{\Gamma(x) = \forall \alpha_1, \dots, \alpha_n. \tau}{\Gamma \vdash x : \tau[\tau_1/\alpha_1, \dots, \tau_n/\alpha_n]} (\text{Var}')$$

$$\frac{\Gamma \vdash t_1 : \tau \quad \Gamma[x : \forall \alpha_1, \dots, \alpha_n. \tau] \vdash t_2 : \tau_2}{\Gamma \vdash \text{let } x = t_1 \text{ in } t_2 : \tau_2} (\text{Let}') \quad \{\alpha_1, \dots, \alpha_n\} = FV(\tau) \setminus FV(\Gamma)$$

( $\text{Var}$ ) und ( $\text{Let}$ ) werden durch ( $\text{Var}'$ ) und ( $\text{Let}'$ ) ersetzt, ( $\text{App}$ ) und ( $\text{Abs}$ ) bleiben unverändert, und ( $\forall \text{Intro}$ ) und ( $\forall \text{Elim}$ ) verschwinden: Das resultierende System hat vier syntaxgesteuerte Regeln.

Bemerkung: Typschemata kommen nur noch in  $\Gamma$  vor.

Beispiel:

$$\frac{\frac{\frac{D = F * E}{\Gamma' \vdash p : F \rightarrow (E \rightarrow D)} \quad \frac{F = A}{\Gamma' \vdash x : F}}{\Gamma' \vdash p x : E \rightarrow D} \quad \frac{C = E}{\Gamma' \vdash z : E}}{\Gamma' \vdash (p x) z : D} \quad \frac{B = A * G}{\Gamma'' \vdash y : G \rightarrow B} \quad \frac{\frac{G = A * \text{int}}{\Gamma'' \vdash y : H \rightarrow G} \quad \frac{H = \text{int}}{\Gamma'' \vdash 1 : H}}{\Gamma'' \vdash y \ 1 : G}}{\Gamma'' \vdash y (y \ 1) : B}}{\Gamma[x : A] \vdash \lambda z. p x z : C \rightarrow D} \quad \frac{\Gamma[x : A] \vdash \text{let } y = \lambda z. p x z \text{ in } y (y \ 1) : B}{\Gamma = [1 : \text{int}, p : \forall \alpha, \beta. \alpha \rightarrow \beta \rightarrow (\alpha * \beta)] \vdash \lambda x. \text{let } y = \lambda z. p x z \text{ in } y (y \ 1) : A \rightarrow B}$$

(wobei  $\Gamma' = \Gamma[x : A, z : C]$  und  $\Gamma'' = \Gamma[x : A, y : \forall C. C \rightarrow A * C]$ )  
 $\Rightarrow B = A * (A * \text{int})$

Beweis der Äquivalenz der beiden Systeme: Jeder Ableitungsbaum mit expliziten Quantorenregeln läßt sich so transformieren, daß ( $\forall \text{Elim}$ ) nur unterhalb der ( $\text{Var}$ )-Regel und ( $\forall \text{Intro}$ ) nur in der linken Prämisse der ( $\text{let}$ )-Regel vorkommt.

Komplexität der Typinferenz:

- ohne **let**: linear
- mit **let**: DEXPTIME-vollständig (Typen *können* exponentiell mit der Größe der Terme wachsen.)

Beispiel:

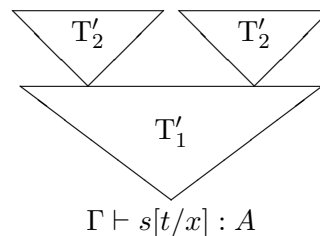
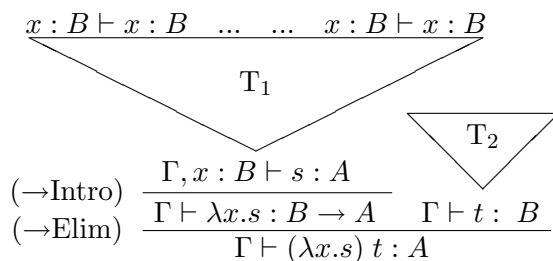
$$\begin{aligned} & \text{let } x_0 = \lambda y. \lambda z. z \ y \ y \\ & \text{in let } x_1 = \lambda y. x_0 (x_0 \ y) \\ & \quad \text{in } \dots \\ & \quad \quad \ddots \\ & \quad \quad \quad \text{let } x_{n+1} = \lambda y. x_n (x_n \ y) \\ & \quad \quad \quad \text{in } x_{n+1} (\lambda z. z) \end{aligned}$$



# Kapitel 4

## Der Curry-Howard Isomorphismus

typisierter $\lambda$ -Kalkül ( $\lambda^{\rightarrow}$ )	konstruktive Logik (minimale Aussagenlogik)
Typen: $\tau ::= \alpha   \beta   \gamma   \dots   \tau \rightarrow \tau$	Formeln: $A ::= \underbrace{P   Q   R   \dots}_{\text{Aussagenvariable}}   A \rightarrow A$
$\Gamma \vdash t : \tau$	$\Gamma \vdash A$ ( $\Gamma$ : Menge von Formeln)
$\frac{\Gamma \vdash t_1 : \tau_2 \rightarrow \tau_1 \quad \Gamma \vdash t_2 : \tau_2}{\Gamma \vdash (t_1 t_2) : \tau_1}$ (App)	$\frac{\Gamma \vdash A \rightarrow B \quad \Gamma \vdash A}{\Gamma \vdash B}$ ( $\rightarrow$ Elim)
$\frac{\Gamma[x : \tau_1] \vdash t : \tau_2}{\Gamma \vdash \lambda x.t : \tau_1 \rightarrow \tau_2}$ (Abs)	$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \rightarrow B}$ ( $\rightarrow$ Intro)
$\Gamma \vdash x : \Gamma(x)$ falls $\Gamma(x)$ definiert	$\Gamma \vdash A$ falls $A \in \Gamma$
typkorrekte $\lambda$ -Terme	Beweise
Beispiel: $\frac{[x : \alpha] \vdash x : \alpha}{\vdash \lambda x.x : \alpha \rightarrow \alpha}$	$\frac{A \vdash A}{\vdash A \rightarrow A}$
Der $\lambda$ -Term kodiert das Skelett des Beweises.	Diese Herleitung wird in kompakter Weise durch $\lambda x.x$ repräsentiert und kann durch Typinferenz rekonstruiert werden.



Beweisreduktion = Lemma-Elimination

Korrektheit folgt aus Subject Reduction: Typen sind invariant unter  $\beta$ -Reduktion

Beispiel:

$$\underbrace{\underbrace{((A \rightarrow A) \rightarrow B \rightarrow C)}_{a'}}_x \rightarrow \underbrace{((A \rightarrow A) \rightarrow B)}_y \rightarrow C =: \phi$$

2 Beweise:

$$\begin{aligned} & \lambda x. \lambda y. (\lambda a'. x a' (y a')) (\lambda a. a) : \phi && \text{Beweis mit Lemma } A \rightarrow A \\ \rightarrow & \lambda x. \lambda y. x (\lambda a. a) (y (\lambda a. a)) : \phi && \text{Beweis in Normalform} \end{aligned}$$

**Definition 4.0.1** Ein Beweis ist in **Normalform**, wenn der zugehörige  $\lambda$ -Term in  $\beta$ -Normalform ist.

Ein Beweis ist genau dann in Normalform, wenn kein Teilbeweis der Form

$$(\text{Elim}) \frac{(\text{Intro}) \frac{\dots}{\dots} \dots}{\dots}$$

auftritt. Damit erhält man unmittelbar

**Lemma 4.0.2** *Ein Beweis in Normalform, der mit  $(\rightarrow\text{Elim})$  endet, muß folgende Form haben:*

$$(\rightarrow\text{Elim}) \frac{\frac{\Gamma \vdash A_n \rightarrow A}{\Gamma \vdash A} \left. \begin{array}{l} T \\ T_1 \end{array} \right\} (*)}{\Gamma \vdash A}$$

wobei der Teilbaum  $T$  folgende Gestalt hat:

$$\begin{array}{c} (\rightarrow\text{Elim}) \frac{\frac{\text{Annahme-Regel}}{\Gamma \vdash A_1 \rightarrow \dots \rightarrow A_n \rightarrow A} \quad \frac{\Gamma \vdash A_1}{\Gamma \vdash A_1}}{\Gamma \vdash A_n \rightarrow A} \quad \cdot \\ \cdot \\ \cdot \\ (\rightarrow\text{Elim}) \frac{\frac{\Gamma \vdash A_{n-1} \rightarrow (A_n \rightarrow A)}{\Gamma \vdash A_n \rightarrow A} \quad \frac{\Gamma \vdash A_{n-1}}{\Gamma \vdash A_{n-1}}}{\Gamma \vdash A_n \rightarrow A} \end{array}$$

**Theorem 4.0.3** *In einem Beweis in Normalform von  $\Gamma \vdash A$  kommen nur Subformeln von  $\Gamma$  und  $A$  vor. ("Subformel" ist reflexiv.)*

Beweis: mit Induktion über die Herleitung von  $\Gamma \vdash A$

1.  $\Gamma \vdash A$  mit  $A \in \Gamma$ : klar
- 2.

$$\begin{array}{c}
 \triangle \\
 \Gamma \\
 (\rightarrow\text{Intro}) \frac{\Gamma, A_1 \vdash A_2}{\Gamma \vdash A_1 \rightarrow A_2}
 \end{array}$$

Induktions-Hypothese: in  $T$  nur Subformeln von  $\Gamma$ ,  $A_1$  und  $A_2$

Daraus folgt die Behauptung unmittelbar.

3. siehe (\*)

wegen Annahme-Regel:  $A_1 \rightarrow A_2 \rightarrow \dots \rightarrow A_n \rightarrow A \in \Gamma$

Induktions-Hypothese 1: in  $T_1$  nur Subformeln von  $\Gamma$ ,  $A_n \Rightarrow$  in  $T_1$  nur Subformeln von  $\Gamma$

Induktions-Hypothese 2: in  $T$  nur Subformeln von  $\Gamma$ ,  $A_n \rightarrow A \Rightarrow$  in  $T$  nur Subformeln von  $\Gamma$

deshalb im ganzen Baum nur Subformeln von  $\Gamma$  □

**Theorem 4.0.4**  $\Gamma \vdash A$  ist entscheidbar.

Beweis: durch Algorithmus:

Endliche Suche nach Beweisbaum in Normalform (existiert immer, da  $\rightarrow_\beta$  für typkorrekte Terme terminiert) durch Aufbau von den Wurzeln zu den Blättern; solange der Baum unvollständig ist, wähle unbewiesenes Blatt:

Falls  $\Gamma \vdash A$  mit  $A \in \Gamma$ , dann Beweis durch Annahme-Regel;

sonst: Zyklustest: Kam das Blatt auf dem Pfad zur Wurzel schon mal vor?

Wenn ja: Abbruch dieser Alternative (Backtracking)

sonst: Benutze  $(\rightarrow\text{Intro})$  (Prämisse ist eindeutig bestimmt) oder

$$\frac{\Gamma \vdash A_n \rightarrow A \quad \Gamma \vdash A_n}{\Gamma \vdash A} (\rightarrow \text{Elim})$$

so dass  $A_1 \rightarrow \dots \rightarrow A_n \rightarrow A \in \Gamma$  (endliche Auswahl).

Dieser Algorithmus terminiert, da die Wurzel nur endlich viele Subformeln hat und oberhalb der Wurzel nur diese Subformeln vorkommen (per Konstruktion), d.h. es gibt nur endlich viele  $\Gamma' \vdash A'$ , die oberhalb der Wurzel erscheinen können (Kontext ist Menge, d.h. keine Duplikate), und da Zyklen erkannt werden. □

Beispiel:

$$\frac{\frac{\Gamma \vdash P \rightarrow Q \rightarrow R \quad \Gamma \vdash P}{\Gamma \vdash Q \rightarrow R} (\rightarrow \text{Elim}) \quad \frac{\Gamma \vdash P \rightarrow Q \quad \Gamma \vdash P}{\Gamma \vdash Q} (\rightarrow \text{Elim})}{\frac{\Gamma := P \rightarrow Q \rightarrow R, P \rightarrow Q, P \vdash R}{\vdash (P \rightarrow Q \rightarrow R) \rightarrow (P \rightarrow Q) \rightarrow P \rightarrow R} \text{ 3mal } (\rightarrow \text{I})}$$

**Peirce-Formel:**  $((P \rightarrow Q) \rightarrow P) \rightarrow P$  (2-wertig wahr)

$$\frac{\frac{\Gamma \vdash A_n \rightarrow P \quad \Gamma \vdash A_n}{\Gamma := (P \rightarrow Q) \rightarrow P \vdash P} (\rightarrow \text{Elim})}{\vdash ((P \rightarrow Q) \rightarrow P) \rightarrow P} (\rightarrow \text{Intro})$$

mit  $A_1 \rightarrow \dots \rightarrow A_n \rightarrow P \in \Gamma \Rightarrow n = 1$  und  $A_n = P \rightarrow Q$ . Betrachte  $\Gamma \vdash P \rightarrow Q$ . Die Herleitung kann nicht mit (Elim) erfolgt sein, denn  $\Gamma$  enthält keine Formel der Gestalt

$\dots \rightarrow (P \rightarrow Q)$ . Daher:

$$\frac{\frac{\Gamma, P \vdash B_n \rightarrow Q \quad \Gamma, P \vdash B_n}{\Gamma, P \vdash Q} \text{ (Elim)}}{\Gamma \vdash P \rightarrow Q} \text{ (Intro)}$$

mit  $B_1 \rightarrow \dots \rightarrow B_n \rightarrow Q \in \Gamma, P$  — eine solche Formel findet sich in  $\Gamma$  und  $P$  aber nicht. Die Peirce-Formel ist hiermit nicht beweisbar.

Da Peirce-Formel in der zweiwertigen Aussagenlogik gilt, ist die konstruktive Logik unvollständig bzgl. des zweiwertigen Modellbegriffs. Es gibt einen der konstruktiven Logik angepaßten Modellbegriff, der zu einem vollständigen Beweissystem führt.

**Übung 4.0.5** *Beweise*  $\vdash (((p \rightarrow q) \rightarrow p) \rightarrow p) \rightarrow q$

Beispiele zum Unterschied „konstruktiv“ — „nicht konstruktiv“:

1.  $\forall k \geq 8. \exists m, n. k = 3m + 5n$

Beweis: mit Induktion über  $k$ :

Basis:  $k = 8 \Rightarrow (m, n) = (1, 1)$

Schritt: Es gelte  $k = 3m + 5n$  (Induktions-Hypothese)

Fallunterscheidung:

1.  $n \neq 0 \Rightarrow k + 1 = (m + 2) * 3 + (n - 1) * 5$

2.  $n = 0 \Rightarrow m \geq 3 \Rightarrow k + 1 = (m - 3) * 3 + (n + 2) * 5$  □

zugehöriger Algorithmus:

$$f : \mathbb{N}_{\geq 8} \rightarrow \mathbb{N} \times \mathbb{N}$$

$$f(8) = (1, 1)$$

$$f(k + 1) = \mathbf{let} (m, n) = f(k)$$

$$\mathbf{in} \mathbf{if} n \neq 0 \mathbf{then} (m + 2, n - 1) \mathbf{else} (m - 3, n + 2)$$

2.  $\exists$  irrationale  $a, b$ .  $a^b$  ist rational

Fallunterscheidung:

1.  $\sqrt{2}^{\sqrt{2}}$  rational  $\Rightarrow a = b = \sqrt{2}$

2.  $\sqrt{2}^{\sqrt{2}}$  irrational  $\Rightarrow a = \sqrt{2}^{\sqrt{2}}, b = \sqrt{2} \Rightarrow a^b = \sqrt{2}^2 = 2$

Klassifikation:

Frage	Typen	Formeln
$t : \tau ?$ ( $t$ explizit typisiert)	Hat $t$ den Typ $\tau$ ?	Ist $t$ ein korrekter Beweis der Formel $\tau$ ?
$\exists \tau. t : \tau$	Typinferenz	Was beweist der Beweis $t$ ?
$\exists t. t : \tau$	Programmsynthese	Beweissuche

Beweissuche in  $\lambda^{\rightarrow}$  ist PSPACE-vollständig !

# Anhang A

## Relationale Grundlagen

### A.1 Notation

Im Folgenden ist  $\rightarrow \subseteq A \times A$  eine beliebige binäre Relation auf einer Menge  $A$ . Statt  $(a, b) \in \rightarrow$  schreiben wir  $a \rightarrow b$ .

#### Definition A.1.1

$$\begin{aligned}
 x \xrightarrow{=} y & :\Leftrightarrow x \rightarrow y \vee x = y && \text{(reflexive Hülle)} \\
 x \leftrightarrow y & :\Leftrightarrow x \rightarrow y \vee y \rightarrow x && \text{(symmetrische Hülle)} \\
 x \xrightarrow{n} y & :\Leftrightarrow \exists x_1, \dots, x_n. x = x_1 \rightarrow x_2 \rightarrow \dots \rightarrow x_n = y \\
 x \xrightarrow{+} y & :\Leftrightarrow \exists n > 0. x \xrightarrow{n} y && \text{(transitive Hülle)} \\
 x \xrightarrow{*} y & :\Leftrightarrow \exists n \geq 0. x \xrightarrow{n} y && \text{(reflexive und transitive Hülle)} \\
 x \leftrightarrow^* y & :\Leftrightarrow x (\leftrightarrow)^* y && \text{(reflexive, transitive und symmetrische Hülle)}
 \end{aligned}$$

**Definition A.1.2** Ein Element  $a$  ist in **Normalform bzgl.**  $\rightarrow$  falls es kein  $b$  mit  $a \rightarrow b$  gibt.

### A.2 Konfluenz

**Definition A.2.1** Eine Relation  $\rightarrow$

ist **konfluent**, falls  $x \xrightarrow{*} y_1 \wedge x \xrightarrow{*} y_2 \Rightarrow \exists z. y_1 \xrightarrow{*} z \wedge y_2 \xrightarrow{*} z$ .

ist **lokal konfluent**, falls  $x \rightarrow y_1 \wedge x \rightarrow y_2 \Rightarrow \exists z. y_1 \xrightarrow{*} z \wedge y_2 \xrightarrow{*} z$ .

hat die **Diamant-Eigenschaft**, falls  $x \rightarrow y_1 \wedge x \rightarrow y_2 \Rightarrow \exists z. y_1 \rightarrow z \wedge y_2 \rightarrow z$ .

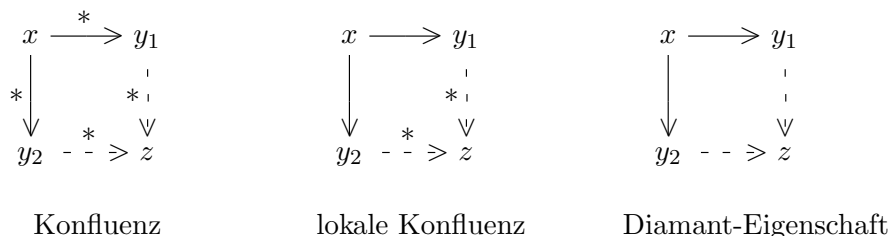


Abbildung A.1: Skizze zu Definition A.2.1

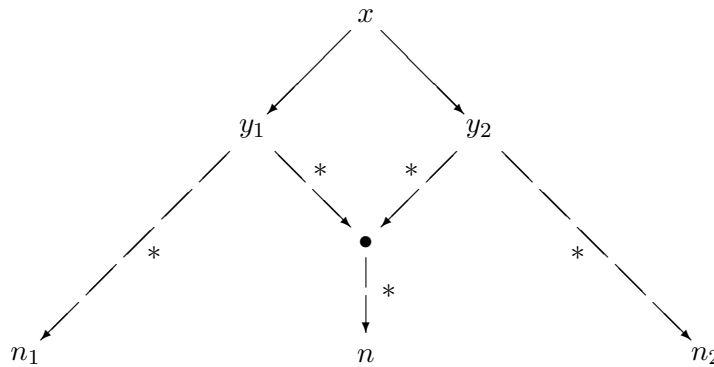
**Fakt A.2.2** Ist  $\rightarrow$  konfluent, dann hat jedes Element höchstens eine Normalform.



**Lemma A.2.3 (Newmann's Lemma)** *Ist  $\rightarrow$  lokal konfluent und terminiert, so ist  $\rightarrow$  auch konfluent.*

Beweis: indirekt

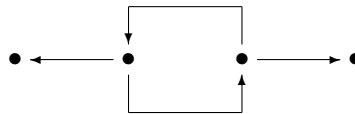
Annahme:  $\rightarrow$  ist nicht konfluent, d.h. es gibt ein  $x$  mit zwei verschiedenen Normalformen  $n_1$  und  $n_2$ . Wir zeigen: Hat  $x$  zwei verschiedene Normalformen, so hat  $x$  einen direkten Nachfolger mit zwei verschiedenen Normalformen. Dies ist ein Widerspruch zu „ $\rightarrow$  terminiert“.



1.  $n \neq n_1$ :  $y_1$  hat zwei verschiedene Normalformen.
2.  $n \neq n_2$ :  $y_2$  hat zwei verschiedene Normalformen.

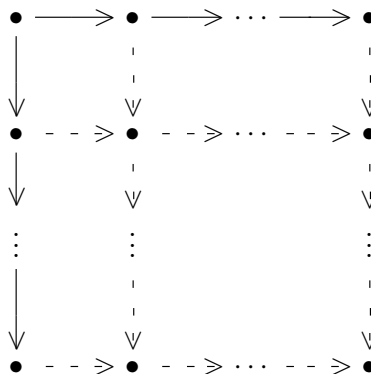
□

Beispiel für eine lokal konfluente, aber nicht konfluente Relation:



**Lemma A.2.4** *Hat  $\rightarrow$  die Diamant-Eigenschaft, so ist  $\rightarrow$  konfluent.*

Beweis: siehe folgende Skizze:



□

**Lemma A.2.5** *Seien  $\rightarrow$  und  $>$  binäre Relationen mit  $\rightarrow \subseteq > \subseteq \overset{*}{\rightarrow}$ . Dann ist  $\rightarrow$  konfluent, falls  $>$  die Diamant-Eigenschaft hat.*

Beweis:

1. Da  $*$  monoton und idempotent ist, folgt aus  $\rightarrow \subseteq > \subseteq \overset{*}{\rightarrow}$  direkt  $\overset{*}{\rightarrow} \subseteq >^* \subseteq (\overset{*}{\rightarrow})^* = \overset{*}{\rightarrow}$  und damit  $\overset{*}{\rightarrow} = >^*$ .
2.  $>$  hat die Diamant-Eigenschaft  
 $\Rightarrow >$  ist konfluent (Lemma A.2.4)  
 $\Leftrightarrow >^*$  hat die Diamant-Eigenschaft  
 $\Leftrightarrow \overset{*}{\rightarrow}$  hat die Diamant-Eigenschaft  
 $\Leftrightarrow \rightarrow$  ist konfluent. □

**Definition A.2.6** Eine Relation  $\rightarrow \subseteq A \times A$  hat die **Church-Rosser Eigenschaft** falls

$$a \overset{*}{\leftrightarrow} b \Leftrightarrow \exists c. a \overset{*}{\rightarrow} c \overset{*}{\leftarrow} b$$

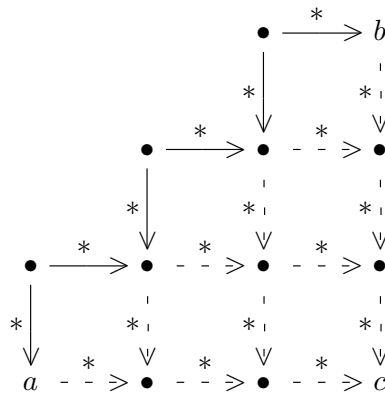
**Theorem A.2.7** Eine Relation  $\rightarrow$  ist konfluent gdw sie die Church-Rosser Eigenschaft hat.

Beweis:

„ $\Leftarrow$ “: klar

„ $\Rightarrow$ “:

1.  $a \overset{*}{\rightarrow} c \overset{*}{\leftarrow} b \Rightarrow a \overset{*}{\leftrightarrow} b$
2.  $a \leftrightarrow b$ :



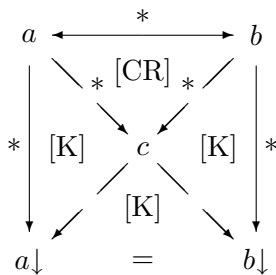
**Korollar A.2.8** Ist  $\rightarrow$  konfluent und haben  $a$  und  $b$  die Normalform  $a\downarrow$  bzw.  $b\downarrow$ , dann gilt:

$$a \overset{*}{\leftrightarrow} b \Leftrightarrow a\downarrow = b\downarrow$$

Beweis:

$\Leftarrow$  : klar

$\Rightarrow$  :

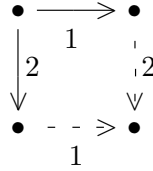


[K]: Konfluenz von  $\rightarrow$   
 [CR]: Die Church-Rosser Eigenschaft von  $\rightarrow$

### A.3 Kommutierende Relationen

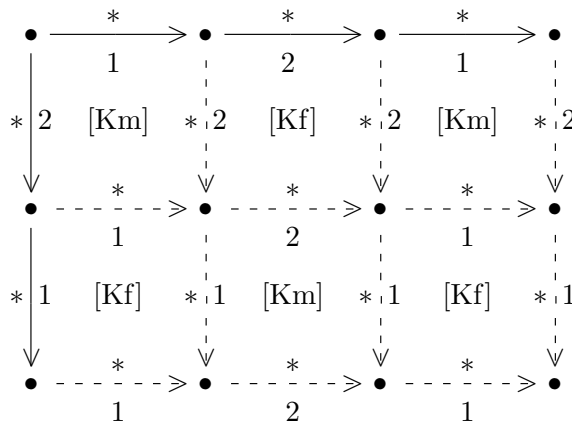
**Definition A.3.1** Seien  $\rightarrow_1$  und  $\rightarrow_2$  beliebige Relationen.  $\rightarrow_1$  und  $\rightarrow_2$  **kommutieren**, falls für alle  $s, t_1, t_2$  gilt:

$$(s \rightarrow_1 t_1 \wedge s \rightarrow_2 t_2) \Rightarrow \exists u. (t_1 \rightarrow_2 u \wedge t_2 \rightarrow_1 u)$$



**Lemma A.3.2 (Hindley/Rosen)** Falls  $\rightarrow_1$  und  $\rightarrow_2$  konfluent sind und  $\xrightarrow{*}_1$  und  $\xrightarrow{*}_2$  kommutieren, dann ist  $\rightarrow_{12} := \rightarrow_1 \cup \rightarrow_2$  konfluent.

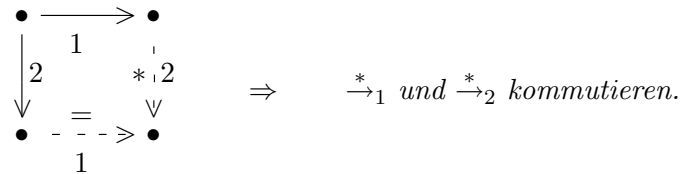
Beweis:



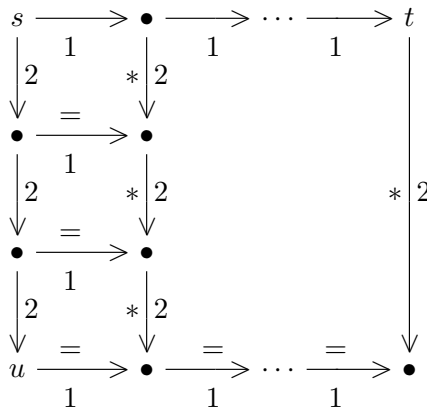
[Kf]:  $\rightarrow_1$  bzw.  $\rightarrow_2$  ist konfluent.  
 [Km]:  $\rightarrow_1$  und  $\rightarrow_2$  kommutieren.

□

**Lemma A.3.3**



Beweis:



Formal: Induktion erst über die Länge von  $s \xrightarrow{*}_1 t$  und dann über die Länge von  $s \xrightarrow{*}_2 u$ . □

# Literaturverzeichnis

- [Bar84] Hendrik Pieter Barendregt. *The Lambda Calculus, its Syntax and Semantics*. North-Holland, 2nd edition, 1984.
- [GLT90] Jean-Yves Girard, Yves Lafont, and Paul Taylor. *Proofs and Types*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1990.
- [Han04] Chris Hankin. *An Introduction to Lambda Calculi for Computer Scientists*. King's College Publications, 2004.
- [HS86] J. Roger Hindley and Jonathan P. Seldin. *Introduction to Combinators and  $\lambda$ -Calculus*. Cambridge University Press, 1986.
- [Loa98] Ralph Loader. Notes on simply typed lambda calculus. Technical Report ECS-LFCS-98-381, Department of Computer Science, University of Edinburgh, 1998.