Lambda-Kalkül

Prof. Tobias Nipkow

15. November 2004

Inhaltsverzeichnis

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:

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

$(t_1\ t_2)$ 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 β-Reduktion, d.h. der Auswertung einer Applikation $((\lambda x.s)\ t)$ durch Einsetzen des Argumentes t für den formalen Parameter x in s. Beispiele:

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

Notation:

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

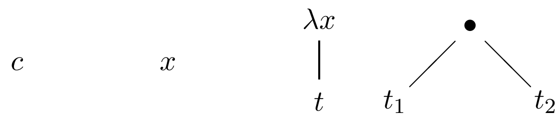
  Beispiel: $\lambda x.x\ x \equiv \lambda x.(x\ x)$
  $\not\equiv (\lambda x.x)\ x$

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

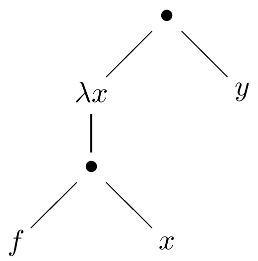
Terme als Bäume:

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

Baum:



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 Currying (Schönfinkeln)

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

Beispiel:

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

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

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

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: $g' \colon \mathbb{N} \to (\mathbb{N} \to \mathbb{N})$

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

Auswertung im Lambda-Kalkül hierzu:

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

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

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

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

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

1.1.3 Statische Bindung und Substitution

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

Beispiel:

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

freie Variable

Die Pfeile zeigen vom Vorkommnis einer Variablen zum bindenden λ.

Menge der freien Variablen eines Terms als Funktion:

$FV\colon \quad \text{Term} \quad \to \quad \text{Menge von Variablen}$

$$\begin{array}{lll} FV(c) & = & \emptyset \\ FV(x) & = & \{x\} \\ FV(s\ t) & = & FV(s) \cup FV(t) \\ FV(\lambda x.t) & = & FV(t) \setminus \{x\} \end{array}$$

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:

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

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:

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

Lemma 1.1.5

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

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

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

1.1.4 α-Konversion

$$s =_\alpha t$$

falls s und t gleich sind modulo Umbenennung gebundener Variablen:

„Gebundene Namen sind Schall und Rauch"

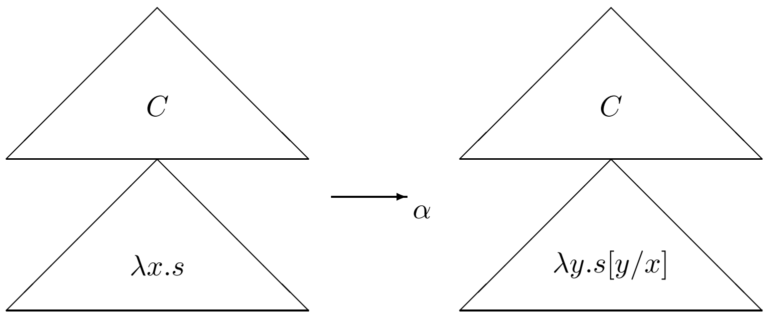
Beispiel:

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

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

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

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. α-äquivalente Terme werden identifiziert, d.h. wir arbeiten mit α-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 $K = \lambda x.\lambda y.x$:

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

   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 β-Reduktion (Kontraktion)

Definition 1.2.1 Ein β-**Redex** (**red**ucible **ex**pression) ist ein Term der Form $(\lambda x.s)t$. Wir definieren β-**Reduktion** durch

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

Ein Term t ist in β-**Normalform** falls er in Normalform bzgl. $\to_\beta$ ist.

$$
\begin{tikzcd}
 & (\lambda x.x\ x)z \arrow[dr, "\beta"] & \\
(\lambda x.x\ x)((\lambda y.y)z) \arrow[ur, "\beta"] \arrow[dr, "\beta"'] & & z\ z \\
 & ((\lambda y.y)z)((\lambda y.y)z) \arrow[ur, "{}^{2}_{\beta}"'] &
\end{tikzcd}
$$

Abbildung 1.1: $\rightarrow_\beta$ kann verzweigen

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

β-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. $s \rightarrow_\beta s' \quad \Rightarrow \quad (s\ t) \rightarrow_\beta (s'\ t)$
3. $s \rightarrow_\beta s' \quad \Rightarrow \quad (t\ s) \rightarrow_\beta (t\ s')$
4. $s \rightarrow_\beta s' \quad \Rightarrow \quad \lambda x.s \rightarrow_\beta \lambda x.s'$

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 $s \rightarrow_\beta s' \quad \Rightarrow \quad s[t/x] \rightarrow_\beta s'[t/x]$

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 $s \xrightarrow{*}_\beta s' \ \wedge\ t \xrightarrow{*}_\beta t' \quad \Rightarrow \quad s[t/x] \xrightarrow{*}_\beta s'[t'/x]$

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

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: $\mathrel{\Vert\!\!\!\rightarrow}$) da $t \mathrel{\Vert\!\!\!\rightarrow} z\ z$.

Problem: $\mathrel{\Vert\!\!\!\rightarrow}$ hat die Diamant-Eigenschaft auch nicht:

$$\begin{tikzcd}
 & ((\lambda x.x)\ c)\ d \arrow[dr, "\Vert" marking] & \\
\underbrace{(\lambda x.\overbrace{(\lambda y.x\ y)\ c})(\overbrace{(\lambda x.x)\ d})} \arrow[ur, "\Vert" marking] \arrow[dr, "\Vert" marking] & & c\ d \\
 & \underbrace{(\lambda y.\underbrace{((\lambda x.x)\ d)}\ y)\ c} &
\end{tikzcd}$$

Es gilt *nicht* $(\lambda y.((\lambda x.x)d)y)c \mathrel{\Vert\!\!\!\rightarrow} c\ d$ da $(\lambda y.((\lambda x.x)d)y)c$ *geschachtelte* Redexe enthält.

Definition 1.2.9 Parallele (und geschachtelte) Reduktion (in Zeichen: $>$)

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:

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

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

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: $u \stackrel{*}{\rightarrow} u', v \stackrel{*}{\rightarrow} v'$
$s = (\lambda x.u)v \rightarrow_\beta^* (\lambda x.u')v \rightarrow_\beta^* (\lambda x.u')v' \rightarrow_\beta u'[v'/x]$
Restliche Fälle: zur Übung □

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

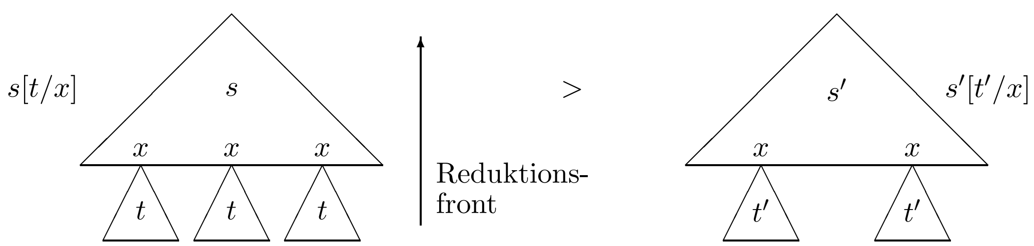
Das nächste Lemma folgt direkt aus der Analyse der anwendbaren Regeln:

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 η-Reduktion

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

Motivation für die η-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 η-Reduktion nicht nur an der Wurzel erlaubt.

Definition 1.3.1 $C[\lambda x.(t\ x)] \;\rightarrow_\eta\; C[t] \quad$ 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 $\rightarrow_\eta$.

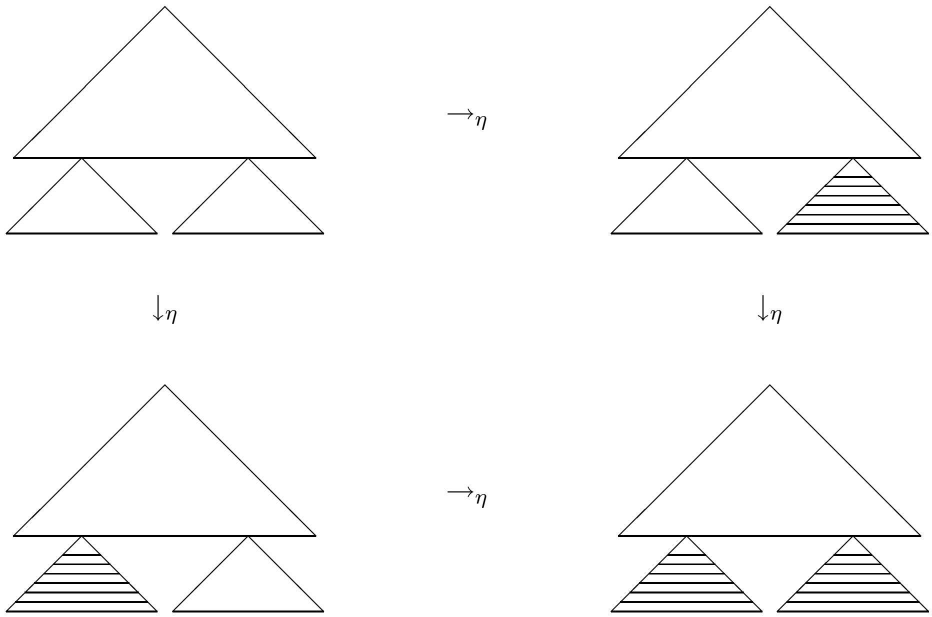
Fakt 1.3.3 $s \rightarrow_\eta t \quad \Rightarrow \quad FV(s) = FV(t)$

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

$$\begin{tikzcd}
\bullet \arrow[r, "\eta"'] \arrow[d, "\eta"] & \bullet \arrow[d, dashed, "*"', "\eta"] \\
\bullet \arrow[r, dashed, "*", "\eta"'] & \bullet
\end{tikzcd}$$

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.

$$\begin{array}{rcl}
\lambda x.s\,x & \rightarrow_\eta & s \\
\downarrow_\eta & & \downarrow_\eta \\
\lambda x.s'\,x & \rightarrow_\eta & s'
\end{array}$$

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 $\overset{=}{\rightarrow}_\eta$ die Diamant-Eigenschaft hat.

Lemma 1.3.6

$$\begin{tikzcd} \bullet \arrow[r, "\beta"'] \arrow[d, "\eta"] & \bullet \arrow[d, dashed, "*"', "\eta"] \\ \bullet \arrow[r, dashed, "=", "\beta"'] & \bullet \end{tikzcd}$$

Beweis: mit Fallunterscheidung nach der relativen Position der Redexe

1. in getrennten Teilbäumen: klar
2. η-Redex weit unterhalb des β-Redexes:
   (a) $t \rightarrow_\eta t'$:

$$\begin{tikzcd} (\lambda x.s)t \arrow[r, "\beta"'] \arrow[d, "\eta"] & s[t/x] \arrow[d, dashed, "*"', "\eta"] \\ (\lambda x.s)t' \arrow[r, dashed, "\beta"'] & s[t'/x] \end{tikzcd}$$

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

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

$$\begin{tikzcd} (\lambda x.s)t \arrow[r, "\beta"'] \arrow[d, "\eta"] & s[t/x] \arrow[d, dashed, "\eta"] \\ (\lambda x.s')t \arrow[r, dashed, "\beta"'] & s'[t/x] \end{tikzcd}$$

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

$$\begin{tikzcd} \lambda x.s\ x \arrow[r, "\beta"'] \arrow[d, "\eta"] & \lambda x.s'\ x \arrow[d, dashed, "\eta"] \\ s \arrow[r, dashed, "\beta"'] & s' \end{tikzcd}$$

   mit Hilfe von Übung 1.2.8.

4. η-Redex direkt unterhalb des β-Redexes (d.h. überlappend):

$$\begin{tikzcd} (\lambda x.(s\ x))t \arrow[r, "\beta"'] \arrow[d, "\eta"] & s\ t \arrow[d, dashed, "*"', "\eta"] \\ s\ t \arrow[r, dashed, "=", "\beta"'] & s\ t \end{tikzcd}$$

5. β-Redex direkt unterhalb des η-Redexes:

$$\begin{tikzcd}
\lambda x.((\lambda y.s)x) \arrow[r, "\beta"'] \arrow[d, "\eta"] & \lambda x.s[x/y] \arrow[d, dashed, "*\ \eta"] \\
\lambda y.s \arrow[r, dashed, "="', "\beta"'] & \lambda y.s
\end{tikzcd}$$

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

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 λ-Kalkül als Gleichungstheorie

1.4.1 β-Konversion

Definition 1.4.1 [Äquivalenz modulo β-Konversion]

$$s =_\beta t \;:\Leftrightarrow\; s \leftrightarrow^*_\beta t$$

Alternative:

$$(\lambda x.s)\ t =_\beta s[t/x] \qquad\qquad t =_\beta t$$

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

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 β-Normalform besitzen, sonst unentscheidbar.*

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

1.4.2 η-Konversion und Extensionalität

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

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

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

Beweis:

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

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

Definition 1.4.4

$$\begin{aligned} s \rightarrow_{\beta\eta} t \quad &:\Leftrightarrow \quad s \rightarrow_\beta t \ \vee \ s \rightarrow_\eta t \\ s =_{\beta\eta} t \quad &:\Leftrightarrow \quad s \leftrightarrow^*_{\beta\eta} t \end{aligned}$$

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.*

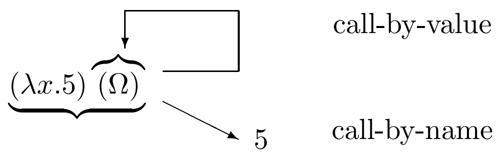
Da $\rightarrow_\eta$ terminiert und konfluent ist gilt natürlich

Korollar 1.4.6 *$\leftrightarrow^*_\eta$ ist entscheidbar.*

1.5 Reduktionsstrategien

Theorem 1.5.1 *Hat t eine β-Normalform, dann kann man diese erreichen, indem man immer den jeweils linkesten β-Redex reduziert.*

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



1.6 Markierte Terme

Motivation: `let`-Ausdrücke

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

Man kann `let` als markierten β-Redex auffassen. Beispiel:

$$\begin{tikzcd} \texttt{let}\ x = (\texttt{let}\ y = s\ \texttt{in}\ y + y)\ \texttt{in}\ x * x \arrow[r] \arrow[d] & \texttt{let}\ x = s + s\ \texttt{in}\ x * x \arrow[d, dashed] \\ (\texttt{let}\ y = s\ \texttt{in}\ y + y) * (\texttt{let}\ y = s\ \texttt{in}\ y + y) \arrow[r, dashed] & (s + s) * (s + s) \end{tikzcd}$$

Menge der markierten Terme:

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

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

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

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

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

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

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

aber

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

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

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

Lemma 1.6.2

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

Übung 1.6.3 *Beweisen Sie obiges Lemma.*

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

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

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 $s_1[\sigma] \to^*_{\underline{\beta}} \underline{\lambda}x.t$ 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. $s = (\underline{\lambda}x.t)u$:
   $s[\sigma] = (\underline{\lambda}x.(t[\sigma]))(u[\sigma])$, wobei $t[\sigma]$ und $u[\sigma]$ terminieren (Ind.-Hyp.). Jede unendliche Reduktion müßte wie folgt aussehen:
   $$s[\sigma] \quad \to^*_{\underline{\beta}} \quad (\underline{\lambda}x.t')\ u' \quad \to_{\underline{\beta}} \quad t'[u'/x] \quad \to_{\underline{\beta}} \quad \ldots$$
   Aber: Da $u[\sigma]$ terminiert und $u[\sigma] \to^*_{\underline{\beta}} u'$, muss auch u' terminieren. Da $t[\sigma] \to^*_{\underline{\beta}} t'$ gilt auch
   $$\underbrace{t[\sigma, u'/x]}_{\substack{\text{terminiert nach Ind.-Hyp.,}\\ \text{da } \sigma \text{ und } u' \text{ terminieren}}} \qquad \to^*_{\underline{\beta}} \qquad \underbrace{t'[u'/x]}_{\substack{\text{muß also auch}\\ \text{terminieren}}}$$
   $\Rightarrow$ Widerspruch zur Annahme, es gäbe eine unendliche Reduktion. $\square$

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.) □

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*

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

1.7 λ-Kalkül als „Programmiersprache“

1.7.1 Datentypen

- `bool`:

  $$\begin{array}{lll} \texttt{true}, \texttt{false}, \texttt{if} & \text{mit} & \texttt{if}\ \texttt{true}\ x\ y \rightarrow_{\beta}^{*} x \\ & \text{und} & \texttt{if}\ \texttt{false}\ x\ y \rightarrow_{\beta}^{*} y \end{array}$$

  wird realisiert durch

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

- Paare:

  $$\begin{array}{lll} \texttt{fst}, \texttt{snd}, \texttt{pair} & \text{mit} & \texttt{fts}(\texttt{pair}\ x\ y) \rightarrow_{\beta}^{*} x \\ & \text{und} & \texttt{snd}(\texttt{pair}\ x\ y) \rightarrow_{\beta}^{*} y \end{array}$$

  wird realisiert durch

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

  Beispiel:

  $$\begin{array}{lllllll} \texttt{fst}(\texttt{pair}\ x\ y) & \rightarrow_{\beta} & \texttt{fst}(\lambda z.z\ x\ y) & \rightarrow_{\beta} & (\lambda z.z\ x\ y)(\lambda xy.x) & & \\ & \rightarrow_{\beta} & (\lambda x\ y.x)\ x\ y & \rightarrow_{\beta} & (\lambda y.x)\ y & \rightarrow_{\beta} & x \end{array}$$

- `nat` (Church-Numerale):

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

Arithmetik:

$$\begin{array}{rcl}
\texttt{succ} &=& \lambda n.\lambda f\,x.f(n\ f\ x) \\
\texttt{add} &=& \lambda m\,n.\lambda f\,x.m\ f(n\ f\ x) \\
\texttt{iszero} &=& \lambda n.n(\lambda x.\texttt{false})\ \texttt{true}
\end{array}$$

damit:

$$\begin{array}{llllll}
\texttt{add}\ \underline{n}\ \underline{m} & \to^2 & \lambda f\,x.\underline{n}\ f(\underline{m}\ f\ x) & \to^2 & \lambda f\,x.\underline{n}\ f(f^m(x)) & \\
& \to^2 & \lambda f\,x.f^n(f^m(x)) & = & \lambda f\,x.f^{n+m}(x) & = \ \underline{n+m}
\end{array}$$

Übung 1.7.1

1. *Listen im λ-Kalkül: Finde λ-Terme für* `nil`, `cons`, `hd`, `tl`, `null` *mit*

$$\begin{array}{ll}
\texttt{null nil} \to^* \texttt{true} & \texttt{hd}(\texttt{cons}\ x\ l) \to^* x \\
\texttt{null}(\texttt{cons}\ x\ l) \to^* \texttt{false} & \texttt{tl}(\texttt{cons}\ x\ l) \to^* l
\end{array}$$

   Hinweis: Benutze Paare.

2. *Finde* `mult` *mit* $\texttt{mult}\ \underline{m}\ \underline{n} \xrightarrow{*} \underline{m*n}$
   und `expt` *mit* $\texttt{expt}\ \underline{m}\ \underline{n} \xrightarrow{*} \underline{m^n}$

3. *Schwierig: Finde* `pred` *mit* $\texttt{pred}\ \underline{m+1} \xrightarrow{*} \underline{m}$ *und* $\texttt{pred}\ \underline{0} \xrightarrow{*} \underline{0}$

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.

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

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

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

Rekursives und nicht-rekursives f verhalten sich gleich:

1. rekursiv:
$$f\ s = (\lambda x.e)\ s \to_\beta e[s/x]$$

2. nicht-rekursiv:
$$f\ s\ =\ \texttt{fix}\ F\ s\ =_\beta\ F\ (\texttt{fix}\ F)\ s\ =\ F\ f\ s\ \to_\beta^2\ e[f/f, s/x]\ =\ e[s/x]$$

Beispiel:
$$\texttt{add}\ m\ n = \texttt{if}\ (\texttt{zero}\ m)\ n\ (\texttt{add}\ (\texttt{pred}\ m)\ (\texttt{succ}\ n))$$
$$\texttt{add} := \texttt{fix}\ \underbrace{(\lambda\texttt{add}.\lambda m\,n.\texttt{if}\ (\texttt{zero}\ m)\ n\ (\texttt{add}\ (\texttt{pred}\ m)(\texttt{succ}\ n)))}_{F}$$

$$\begin{array}{rcl}
\texttt{add}\ \underline{1}\ \underline{2} & = & \texttt{fix}\ F\ \underline{1}\ \underline{2} \\
 & =_\beta & F\ (\texttt{fix}\ F)\ \underline{1}\ \underline{2} \\
 & \to_\beta^3 & \texttt{if}\ (\texttt{iszero}\ \underline{1})\ \underline{2}\ (\texttt{fix}\ F\ (\texttt{pred}\ \underline{1})\ (\texttt{succ}\ \underline{2})) \\
 & \to_\beta^* & \texttt{fix}\ F\ \underline{0}\ \underline{3} \\
 & =_\beta & F\ (\texttt{fix}\ F)\ \underline{0}\ \underline{3} \\
 & \to_\beta^3 & \texttt{if}\ (\texttt{iszero}\ \underline{0})\ \underline{3}\ (...) \\
 & \to_\beta^* & \underline{3}
\end{array}$$

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

Wir zeigen nun, daß sich `fix`, d.h. der Fixpunktoperator, im reinen λ-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“.

$$Y\ t\ \to_\beta\ V_t\ V_t\ \to_\beta\ t(V_t\ V_t)\ \leftarrow_\beta\ t((\lambda f.V_f\ V_f)t)\ =\ t(Y\ t)$$

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

Turing: $A := \lambda x\,f.f(x\ x\ f)$ und $\Theta := A\ A \to_\beta \lambda f.f(A\ A\ f)$. Damit gilt

$$\Theta\ t\ =\ A\ A\ t \to_\beta (\lambda f.f(A\ A\ f))t\ \to_\beta\ t(A\ A\ t)\ =\ t(\Theta\ t)$$

Also: $\Theta\ t \to_\beta^* t(\Theta\ t)$

Berechenbare Funktionen auf $\mathbb{N}$:

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

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

Theorem 1.7.3 *Alle Turingmaschinen-berechenbaren (Registermaschinen-berechenbaren,* `while`*-berechenbaren, μ-rekursiven) Funktionen sind λ-definierbar, und umgekehrt.*

Kapitel 2

Kombinatorische Logik (CL)

Schlagwort: „variablenfreies Programmieren“

Terme:

$$X \quad ::= \quad \underbrace{x}_{\text{Variablen}} \quad | \quad \underbrace{\mathsf{S} \;|\; \mathsf{K} \;|\; \mathsf{I} \;|\; \ldots}_{\text{Konstanten}} \quad | \quad X_1\, X_2 \quad | \quad (X)$$

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

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

Rechenregeln für die **schwache Reduktion** (**weak reduction**, $\to_{\mathrm{w}}$):

$$\begin{array}{rcl}
\mathsf{I}\,X & \to_{\mathrm{w}} & X \\
\mathsf{K}\,X\,Y & \to_{\mathrm{w}} & X \\
\mathsf{S}\,X\,Y\,Z & \to_{\mathrm{w}} & (X\ Z)(Y\ Z) \\
X \to_{\mathrm{w}} X' & \Rightarrow & X\ Y \to_{\mathrm{w}} X'\ Y \quad \wedge \quad Y\ X \to_{\mathrm{w}} Y\ X'
\end{array}$$

Beispiele:

1. $\mathsf{S}\,\mathsf{K}\,X\,Y \quad \to_{\mathrm{w}} \quad \mathsf{K}\,Y\,(X\,Y) \quad \to_{\mathrm{w}} \quad Y$
2. $\mathsf{S}\,\mathsf{K}\,\mathsf{K}\,X \quad \to_{\mathrm{w}} \quad \mathsf{K}\,X\,(\mathsf{K}\,X) \quad \to_{\mathrm{w}} \quad X$

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

Theorem 2.0.4 $\to_{\mathrm{w}}$ *ist konfluent.*

Beweismöglichkeiten:

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

Das Termersetzungssystem $\to_{\mathrm{w}}$ ist nicht terminierend:

Übung 2.0.5 *Finden Sie einen Kombinator X mit $X \to_{\mathrm{w}}^{+} X$.*

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

$$\begin{array}{rcl} \mathsf{A}\,X & \to_{\mathrm{w}}^{*} & X\,X \\ \mathsf{W}\,X\,Y & \to_{\mathrm{w}}^{*} & X\,Y\,Y \\ \mathsf{B}\,X\,Y\,Z & \to_{\mathrm{w}}^{*} & X\,(Y\,Z) \end{array}$$

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

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

2.1 Beziehung zwischen λ-Kalkül und CL

Übersetzung von λ-Termen in CL-Terme:

$$(\_)_{\mathrm{CL}} : \quad \lambda\text{-Terme} \quad \to \quad \text{CL-Terme}$$

$$\begin{array}{rcl} (x)_{\mathrm{CL}} & = & x \\ (s\ t)_{\mathrm{CL}} & = & (s)_{\mathrm{CL}}\ (t)_{\mathrm{CL}} \\ (\lambda x.s)_{\mathrm{CL}} & = & \lambda^{*}x.(s)_{\mathrm{CL}} \end{array}$$

Hilfsfunktion λ^*: Vars $\times$ CL-Terme $\to$ CL-Terme

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

Lemma 2.1.1 $(\lambda^{*}x.X)\ Y \ \to_{\mathrm{w}}^{*}\ X[Y/x]$

Beweis: mit struktureller Induktion über X

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

$$\begin{array}{rcccc} (\lambda^{*}x.(U\ V))Y & = & \mathsf{S}(\lambda^{*}x.U)(\lambda^{*}x.V)Y & \to_{\mathrm{w}} & ((\lambda^{*}x.U)Y)((\lambda^{*}x.V)Y) \\ \text{(Ind.-Hyp.)} & \to_{\mathrm{w}} & (U[Y/x])(V[Y/x]) & = & X[Y/x] \end{array}$$

Übersetzung von CL-Termen in λ-Terme:

$$(\_)_{\lambda} : \quad \text{CL-Terme} \quad \to \quad \lambda\text{-Terme}$$

$$\begin{array}{rcl} (x)_{\lambda} & = & x \\ (\mathsf{K})_{\lambda} & = & \lambda xy.x \\ (\mathsf{S})_{\lambda} & = & \lambda xyz.x\ z\ (y\ z) \\ (X\ Y)_{\lambda} & = & (X)_{\lambda}\ (Y)_{\lambda} \end{array}$$

Theorem 2.1.2 $((s)_{\mathrm{CL}})_\lambda \quad \to_\beta^* \quad s$

Beweis: mit struktureller Induktion über s:

1. $((a)_{\mathrm{CL}})_\lambda = a$
2. Mit Ind.-Hyp.: $((t\ u)_{\mathrm{CL}})_\lambda = ((t)_{\mathrm{CL}}\ (u)_{\mathrm{CL}})_\lambda = ((t)_{\mathrm{CL}})_\lambda((u)_{\mathrm{CL}})_\lambda \stackrel{*}{\to}_\beta t\ u$
3. Mit Lemma 2.1.3 und Ind.-Hyp.: $((\lambda x.t)_{\mathrm{CL}})_\lambda = (\lambda^* x.(t)_{\mathrm{CL}})_\lambda \stackrel{*}{\to}_\beta \lambda x.((t)_{\mathrm{CL}})_\lambda \stackrel{*}{\to}_\beta \lambda x.t$ □

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

Beweis: zur Übung.

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

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

Übung 2.1.5 *Zeigen Sie, daß auch* B, C, K *und* W *ausreichen, um alle λ-Terme darzustellen (Hierbei:* $\mathsf{C}\, X\, Y\, Z \ \to_{\mathrm{w}} \ X\, Z\, Y$*). Kann man* K *auch weglassen?*

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

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

Theorem 2.1.7 $\quad X \to_{\mathrm{w}} Y \quad \Rightarrow \quad (X)_\lambda \to_\beta^* (Y)_\lambda$

Beweis:

$$\begin{tikzcd} C[\mathsf{K}\ X\ Y] \arrow[r, "\mathrm{w}"'] \arrow[d, mapsto, "\lambda"] & C[X] \arrow[d, mapsto, "\lambda"] \\ C_\lambda[(\lambda xy.x)\ X_\lambda\ Y_\lambda] \arrow[r, "*", "\beta"'] & C_\lambda[X_\lambda] \end{tikzcd}$$

analog für S □

aber: Im allgemeinen folgt aus $s \to_\beta t$ *nicht* $(s)_{\mathrm{CL}} \to_{\mathrm{w}}^* (t)_{\mathrm{CL}}$.

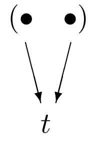
Aufgabe: Man finde ein Gegenbeispiel.

2.2 Implementierungsaspekte

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

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

$(\lambda x.(x\ x))t \quad \to_\beta$

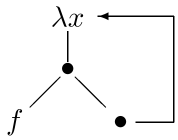


mit Kopie:

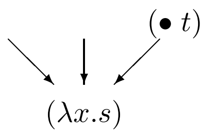
$$\ldots \quad \to_\beta \quad f(\lambda x.f\ x)$$

ohne Kopie ensteht ein zyklischer Term:

$\ldots \quad \to_\beta$



im allgemeinen:



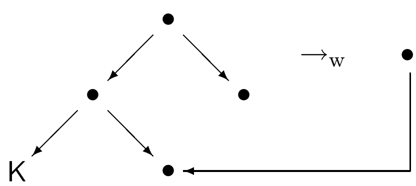
Bei β-Reduktion von $(\bullet\ t)$ Kopie von s nötig!

- α-Konversion nötig

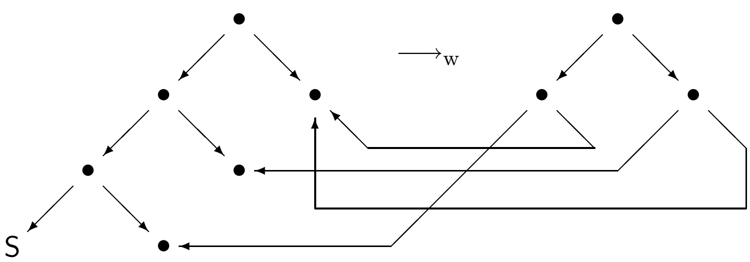
Graphreduktion

Eine radikale Lösung ist die Übersetzung nach CL, weil $\to_{\mathrm{w}}$ auf Graphen ohne Kopieren implementierbar ist:

1. $(\mathsf{K}\ x)\ y \quad \to_{\mathrm{w}} \quad x$:



2. $\mathsf{S}\ x\ y\ z \quad \to_{\mathrm{w}} \quad x\ z\ (y\ z)$:



Ein Problem dabei ist, daß $(\cdot)_{\mathrm{CL}}$ 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 λ-Terme immer verloren.

De Bruijn Notation

Eine zweite Lösung sind die sogenannten de Bruijn-Indizes:

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

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

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

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:

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

Aber: α-äquivalente Terme sind in dieser Notation identisch!

Wir betrachten nun β-Reduktion und Substitution. Beispiele:

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

Im allgemeinen:

$$(\lambda s)t \quad \to_\beta \quad s[t/0]$$

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:

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

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

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

Beispiel:

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

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 λ-Kalkül als Logik
   `True`, `False`, $\wedge$, ... sind λ-Terme
   $\{x \mid P\} \equiv \lambda x.P \qquad x \in M \equiv Mx$
   Inkonsistenz: $R := \lambda x.\, \texttt{not}\, (x\ x) \quad \Rightarrow \quad R\ R =_\beta \texttt{not}\, (R\ R)$

   Church (1940): „Simply Typed λ-Calculus“ erlaubt $x\ x$ 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: mod: nat $\times$ m:nat $\rightarrow \{k \mid 0 \leq k < m\}$

   Resultattyp hängt vom Eingabewert ab („Typtheorie“)

3.1 Einfach typisierter λ-Kalkül ($\lambda^{\to}$)

Kern jeder (funktionalen) Programmiersprache

Typen:

$$\tau \quad ::= \quad \underbrace{\texttt{bool} \mid \texttt{nat} \mid \texttt{int} \mid \ldots}_{\text{Basistypen}} \mid \tau_1 \to \tau_2$$

Konvention: $\to$ assoziiert nach rechts:

$$\tau_1 \to \tau_2 \to \tau_3 \quad \equiv \quad \tau_1 \to (\tau_2 \to \tau_3)$$

Terme:

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

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

3.1.1 Typüberprüfung für explizit typisierte Terme

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

$$\frac{\Gamma(x) \text{ ist definiert}}{\Gamma \vdash x : \Gamma(x)} \ (\text{Var})$$

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

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

- Eine einfache Herleitung:

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

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

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

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

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

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

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 β-Expansion:

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

und

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

aber: $(\lambda z : \mathtt{bool}.y)\; x$ 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:

$$\lambda x : \mathtt{int}.(\lambda y : \mathtt{bool}.y)\; x \quad \begin{array}{l} \xrightarrow{\beta} \;\; \lambda x : \mathtt{int}.x \\[1em] \xrightarrow{\eta} \;\; \lambda y : \mathtt{bool}.y \end{array}$$

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

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

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_τ

3.2 Termination von $\to_\beta$

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 λ-Term. Wir sagen, dass t (bzgl. $\to_\beta$) **divergiert**, g.d.w. es eine unendliche Reduktionsfolge $t \to_\beta t_1 \to_\beta t_2 \to_\beta \cdots$ gibt. Wir sagen, dass t (bzgl. $\to_\beta$) **terminiert**, g.d.w. t nicht divergiert; dann schreiben wir $t\Downarrow$.

Wir definieren zuerst eine Teilmenge T der untypisierten λ-Terme:

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

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

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

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

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

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

$$(\lambda x.r)\, s\, s_1 \ldots s_n \to_\beta^* (\lambda x.r')\, s'\, s'_1 \ldots s'_n \to_\beta r'[s'/x]\, s'_1 \ldots s'_n \to_\beta \cdots$$

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

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 Γ und wir schreiben einfach $t : \tau$.

Wir nennen einen Typ τ **applikativ** g.d.w. für alle t, r und σ gilt

$$\frac{t : \tau \to \sigma \quad r : \tau \quad t \in T \quad r \in T}{t\, r \in T}$$

Wir nennen τ **substitutiv** g.d.w. für alle s, r und σ 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 τ substitutiv. Wir zeigen mit Induktion über die Herleitung von $t \in T$, dass τ applikativ ist.

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

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

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

Lemma 3.2.4 *Sei $\tau = \tau_1 \to \cdots \to \tau_k \to \tau'$, wobei τ' kein Funktionstyp ist. Falls alle τ_i applikativ sind, so ist τ 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 τ_i applikativ sind, gilt $s[r/x] = r(s_1[r/x]) \dots (s_n[r/x]) \in T$.

(λ) 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 (λ) folgt.

(β) 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 (β) 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. □

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

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 (λ) 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 $s \in T$. □

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

3.3 Typinferenz für $\lambda^{\to}$

Typen:	τ	::=	`bool` $\mid$ `int` $\mid \dots$	Basistypen
			$\mid \alpha \mid \beta \mid \gamma \mid \dots$	Typvariablen
			$\mid \tau_1 \to \tau_2$	

Terme: untypisierte λ-Terme

Typinferenzregeln:

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

Terme können verschiedene Typen haben (Polymorphie):

$$\lambda x.x : \quad \alpha \to \alpha$$
$$\lambda x.x : \quad \texttt{int} \to \texttt{int}$$

Definition 3.3.1 $\tau_1 \lesssim \tau_2 \;:\Leftrightarrow\; \exists$ Substitution Θ (von Typen für Typvariable) mit $\tau_1 = \Theta(\tau_2)$ („τ_2 ist allgemeiner oder äquivalent τ_1.“)

Beispiel: $\quad \texttt{int} \to \texttt{int} \;\lesssim\; \alpha \to \alpha \;\lesssim\; \beta \to \beta \;\lesssim\; \alpha \to \alpha$

Theorem 3.3.2 $\Gamma \vdash t : \tau \quad \Rightarrow \quad \exists \sigma. \Gamma \vdash t : \sigma \;\wedge\; \forall \tau'. \Gamma \vdash t : \tau' \;\Rightarrow\; \tau' \lesssim \sigma$
„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:

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

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

3.4 `let`-Polymorphismus

Terme: $t \quad ::= \quad x \mid (t_1\ t_2) \mid \lambda x.t \mid \texttt{let}\ x = t_1\ \texttt{in}\ t_2$
Semantik: $\texttt{let}\ x = t_1\ \texttt{in}\ \ t_2 \quad \equiv \quad t_2[t_1/x]$
(wohldefiniert wegen Termination und Konfluenz von $\rightarrow_{\underline{\beta}}$)

Beispiel:

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

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 λ-gebundene Variablen keine allquantifizierten Typen haben.

Typen: $\tau \quad ::= \quad \texttt{bool} \mid \ldots \mid \alpha \mid ... \mid \tau_1 \rightarrow \tau_2$
Typschemata: $\sigma \quad ::= \quad \forall\alpha.\sigma \mid \tau$

Beispiele für Typschemata:
α, $\texttt{int}$, $\forall\alpha.\alpha \rightarrow \alpha$, $\forall\alpha,\beta.\alpha \rightarrow \beta$
aber nicht $(\forall\alpha.\alpha \rightarrow \alpha) \rightarrow \texttt{bool}$ (Der Allquantor tritt nicht ganz außen auf!)

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

$$\frac{}{\Gamma \vdash x : \Gamma(x)}\ (\text{Var})$$

$$\frac{\Gamma \vdash t_1 : \tau_2 \rightarrow \tau \qquad \Gamma \vdash t_2 : \tau_2}{\Gamma \vdash (t_1\ t_2) : \tau}\ (\text{App})$$

$$\frac{\Gamma[x : \tau_1] \vdash t : \tau_2}{\Gamma \vdash (\lambda x.t) : \tau_1 \rightarrow \tau_2}\ (\text{Abs})$$

$$\frac{\Gamma \vdash t_1 : \sigma_1 \qquad \Gamma[x : \sigma_1] \vdash t_2 : \sigma_2}{\Gamma \vdash \texttt{let}\ \ x = t_1\ \ \texttt{in}\ \ t_2 : \sigma_2}\ (\text{Let})$$

Quantorenregeln:

$$\frac{\Gamma \vdash t : \forall\alpha.\sigma}{\Gamma \vdash t : \sigma[\tau/\alpha]}\ (\forall\text{Elim})$$

$$\frac{\Gamma \vdash t : \sigma}{\Gamma \vdash t : \forall\alpha.\sigma}\ (\forall\text{Intro}) \qquad \text{falls } \alpha \notin FV(\Gamma)$$

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

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

Logik: $x = 0 \vdash x = 0 \quad \not\Rightarrow \quad x = 0 \vdash \forall x.x = 0$
ML: $\lambda x.\texttt{let}\ \ y = x\ \ \texttt{in}\ \ y + (y\ 1)$ sollte nicht typkorrekt sein.

Herleitung hierfür, unter Verletzung der Bedingung:

$$\dfrac{\dfrac{\dfrac{[x:\alpha]\vdash x:\alpha}{[x:\alpha]\vdash x:\forall\alpha.\alpha}\,(\forall\text{Intro}) \qquad [y:\forall\alpha.\alpha]\vdash y+(y\;1):\texttt{int}}{[x:\alpha]\vdash \texttt{let}\;\;y=x\;\;\texttt{in}\;\;y+(y\;1):\texttt{int}}}{\lambda y.\texttt{let}\;\;y=x\;\;\texttt{in}\;\;y+(y\;1):\alpha\to\texttt{int}}$$

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

Lösung: Integriere (∀Elim) mit (Var) und (∀Intro) mit (Let):

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

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

(Var) und (Let) werden durch (Var') und (Let') ersetzt, (App) und (Abs) bleiben unverändert, und (∀Intro) und (∀Elim) verschwinden: Das resultierende System hat vier syntaxgesteuerte Regeln.
Bemerkung: Typschemata kommen nur noch in Γ vor.

Beispiel:

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

(wobei $\Gamma'=\Gamma[x:A,z:C]$ und $\Gamma''=\Gamma[x:A,y:\forall C.C\to A*C]$)

$\Rightarrow\;\; B=A*(A*\texttt{int})$

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

Komplexität der Typinferenz:

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

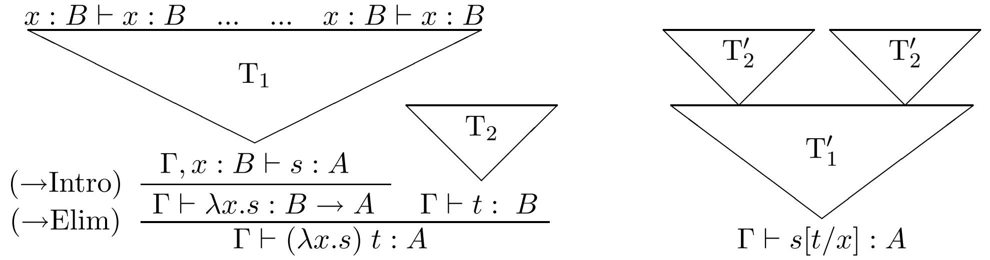
Beispiel:

$$\begin{array}{l}
\texttt{let}\; x_0=\lambda y.\lambda z.z\;y\;y\\
\texttt{in}\;\texttt{let}\; x_1=\lambda y.x_0\;(x_0\;y)\\
\quad\texttt{in}\;\dots\\
\qquad\ddots\\
\qquad\quad\texttt{let}\; x_{n+1}=\lambda y.x_n\;(x_n\;y)\\
\qquad\quad\texttt{in}\; x_{n+1}\;(\lambda z.z)
\end{array}$$

Kapitel 4

Der Curry-Howard Isomorphismus

typisierter λ-Kalkül ($\lambda^{\rightarrow}$)	konstruktive Logik (minimale Aussagenlogik)
Typen: $\tau ::= \alpha \mid \beta \mid \gamma \mid \ldots \mid \tau \rightarrow \tau$	Formeln: $A ::= \underbrace{P \mid Q \mid R \mid \ldots}_{\text{Aussagenvariable}} \mid A \rightarrow A$
$\Gamma \vdash t : \tau$	$\Gamma \vdash A$ (Γ: Menge von Formeln)
$\dfrac{\Gamma \vdash t_1 : \tau_2 \rightarrow \tau_1 \qquad \Gamma \vdash t_2 : \tau_2}{\Gamma \vdash (t_1\ t_2) : \tau_1}$ (App)	$\dfrac{\Gamma \vdash A \rightarrow B \qquad \Gamma \vdash A}{\Gamma \vdash B}$ ($\rightarrow$ Elim)
$\dfrac{\Gamma[x : \tau_1] \vdash t : \tau_2}{\Gamma \vdash \lambda x.t : \tau_1 \rightarrow \tau_2}$ (Abs)	$\dfrac{\Gamma, A \vdash B}{\Gamma \vdash A \rightarrow B}$ ($\rightarrow$ Intro)
$\Gamma \vdash x : \Gamma(x)$ falls $\Gamma(x)$ definiert	$\Gamma \vdash A$ falls $A \in \Gamma$
typkorrekte λ-Terme	Beweise
Beispiel: $\dfrac{[x : \alpha] \vdash x : \alpha}{\vdash \lambda x.x : \alpha \rightarrow \alpha}$	$\dfrac{A \vdash A}{\vdash A \rightarrow A}$
Der λ-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 β-Reduktion

Beispiel:

$$\underbrace{(\underbrace{(A \to A)}_{a'} \to B \to C)}_{x} \to \underbrace{((A \to A) \to B)}_{y} \to C \ =: \ \phi$$

2 Beweise:

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

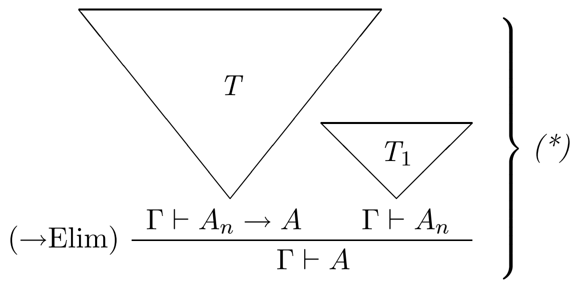
Definition 4.0.1 Ein Beweis ist in **Normalform**, wenn der zugehörige λ-Term in β-Normalform ist.

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

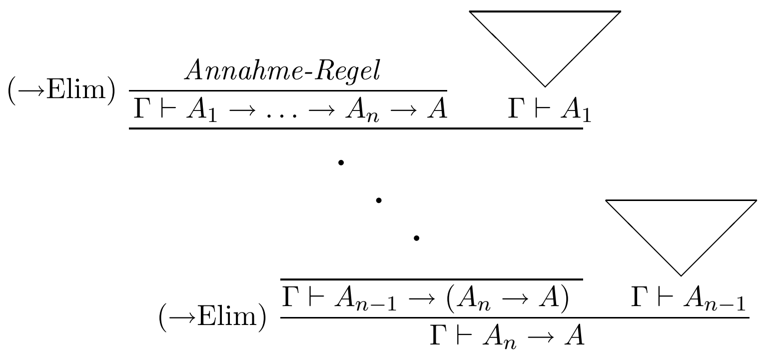
$$(\text{Elim})\dfrac{(\text{Intro})\dfrac{\cdots}{\cdots} \qquad \cdots}{\cdots}$$

auftritt. Damit erhält man unmittelbar

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



wobei der Teilbaum T folgende Gestalt hat:

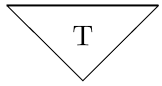


Theorem 4.0.3 *In einem Beweis in Normalform von $\Gamma \vdash A$ kommen nur Subformeln von Γ und A vor. ("Subformel" ist reflexiv.)*

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

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

2.



$$(\to\text{Intro})\ \dfrac{\Gamma, A_1 \vdash A_2}{\Gamma \vdash A_1 \to A_2}$$

Induktions-Hypothese: in T nur Subformeln von Γ, A_1 und A_2

Daraus folgt die Behauptung unmittelbar.

3. siehe (*)

   wegen Annahme-Regel: $A_1 \to A_2 \to \ldots \to A_n \to A \in \Gamma$

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

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

   deshalb im ganzen Baum nur Subformeln von Γ □

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

Beweis: durch Algorithmus:

Endliche Suche nach Beweisbaum in Normalform (existiert immer, da $\to_\beta$ 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 ($\to$Intro) (Prämisse ist eindeutig bestimmt) oder

$$\dfrac{\Gamma \vdash A_n \to A \qquad \Gamma \vdash A_n}{\Gamma \vdash A}\ (\to \text{Elim})$$

so dass $A_1 \to \cdots A_n \to A \in \Gamma$ (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:

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

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

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

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

$\cdots \to (P \to Q)$. Daher:

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

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

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

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

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:

   $$f : \mathbb{N}_{\geq 8} \to \mathbb{N} \times \mathbb{N}$$
   $$\begin{aligned} f(8) &= (1, 1) \\ f(k+1) &= \texttt{let}\ (m, n) = f(k) \\ &\qquad \texttt{in if}\ n \neq 0\ \texttt{then}\ (m + 2, n - 1)\ \texttt{else}\ (m - 3, n + 2) \end{aligned}$$

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 τ ?	Ist t ein korrekter Beweis der Formel τ ?
$\exists \tau. t : \tau$	Typinferenz	Was beweist der Beweis t ?
$\exists t. t : \tau$	Programmsynthese	Beweissuche

Beweissuche in $\lambda^{\to}$ 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

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

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 $x \stackrel{*}{\rightarrow} y_1 \ \wedge x \stackrel{*}{\rightarrow} y_2 \ \Rightarrow\ \exists z.\ y_1 \stackrel{*}{\rightarrow} z \ \wedge\ y_2 \stackrel{*}{\rightarrow} z$.

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

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

$$
\begin{tikzcd}
x \arrow[r, "*"] \arrow[d, "*"'] & y_1 \arrow[d, dashed, "*"'] \\
y_2 \arrow[r, dashed, "*"] & z
\end{tikzcd}
$$

Konfluenz

$$
\begin{tikzcd}
x \arrow[r] \arrow[d] & y_1 \arrow[d, dashed, "*"'] \\
y_2 \arrow[r, dashed, "*"] & z
\end{tikzcd}
$$

lokale Konfluenz

$$
\begin{tikzcd}
x \arrow[r] \arrow[d] & y_1 \arrow[d, dashed] \\
y_2 \arrow[r, dashed] & z
\end{tikzcd}
$$

Diamant-Eigenschaft

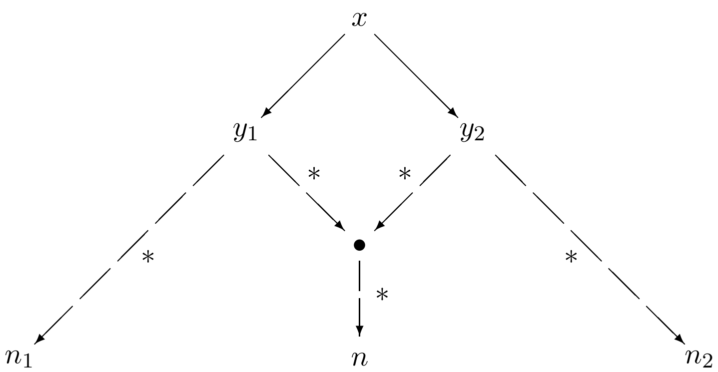
Abbildung A.1: Skizze zu Definition A.2.1

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.*

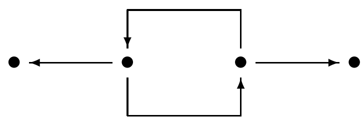
Beweis: indirekt

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“.



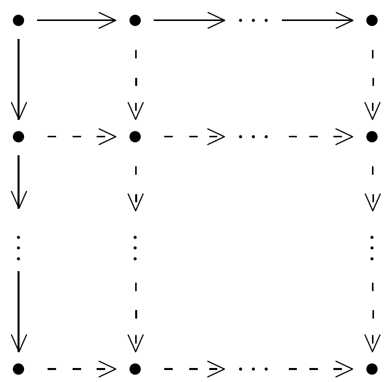
1. $n \neq n_1$: y_1 hat zwei verschiedene Normalformen.
2. $n \neq n_2$: y_2 hat zwei verschiedene Normalformen. □

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 $\rightarrow \subseteq > \subseteq \xrightarrow{*}$. Dann ist $\rightarrow$ konfluent, falls $>$ die Diamant-Eigenschaft hat.*

Beweis:

1. Da * monoton und idempotent ist, folgt aus $\to \subseteq > \subseteq \stackrel{*}{\to}$ direkt $\stackrel{*}{\to} \subseteq >^* \subseteq (\stackrel{*}{\to})^* = \stackrel{*}{\to}$ und damit $\stackrel{*}{\to} = >^*$.

2. $$\begin{array}{ll} & > \text{ hat die Diamant-Eigenschaft} \\ \Rightarrow & > \text{ ist konfluent (Lemma A.2.4)} \\ \Leftrightarrow & >^* \text{ hat die Diamant-Eigenschaft} \\ \Leftrightarrow & \stackrel{*}{\to} \text{ hat die Diamant-Eigenschaft} \\ \Leftrightarrow & \to \text{ ist konfluent.} \end{array}$$ □

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

$$a \stackrel{*}{\leftrightarrow} b \;\Leftrightarrow\; \exists c.\; a \stackrel{*}{\to} c \stackrel{*}{\leftarrow} b$$

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

Beweis:

„$\Leftarrow$“: klar

„$\Rightarrow$“:

1. $a \stackrel{*}{\to} c \stackrel{*}{\leftarrow} b \;\Rightarrow\; a \stackrel{*}{\leftrightarrow} b$

2. $a \leftrightarrow b$:

$$\begin{tikzcd}
 & & \bullet \arrow[r, "*"] \arrow[d, "*"'] & b \arrow[d, dashed, "*"'] \\
 & \bullet \arrow[r, "*"] \arrow[d, "*"'] & \bullet \arrow[r, dashed, "*"] \arrow[d, dashed, "*"'] & \bullet \arrow[d, dashed, "*"'] \\
\bullet \arrow[r, "*"] \arrow[d, "*"'] & \bullet \arrow[r, dashed, "*"] \arrow[d, dashed, "*"'] & \bullet \arrow[r, dashed, "*"] \arrow[d, dashed, "*"'] & \bullet \arrow[d, dashed, "*"'] \\
a \arrow[r, dashed, "*"] & \bullet \arrow[r, dashed, "*"] & \bullet \arrow[r, dashed, "*"] & c
\end{tikzcd}$$

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

$$a \stackrel{*}{\leftrightarrow} b \quad \Leftrightarrow \quad a{\downarrow} = b{\downarrow}$$

Beweis:

$\Leftarrow$: klar

$\Rightarrow$:

$$\begin{tikzcd}
a \arrow[rr, leftrightarrow, "*"] \arrow[dd, "*"'] \arrow[dr, "*"] & \text{[CR]} & b \arrow[dd, "*"] \arrow[dl, "*"'] \\
\text{[K]} & c \arrow[dl] \arrow[dr] & \text{[K]} \\
a{\downarrow} & \text{[K]} \;=\; & b{\downarrow}
\end{tikzcd}$$

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

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:

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

$$\begin{tikzcd}
\bullet \arrow[r, "1"'] \arrow[d, "2"] & \bullet \arrow[d, dashed, "2"] \\
\bullet \arrow[r, dashed, "1"'] & \bullet
\end{tikzcd}$$

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:

$$\begin{tikzcd}
\bullet \arrow[r, "*", "1"'] \arrow[d, "*"', "2"] \arrow[dr, phantom, "{[Km]}"] & \bullet \arrow[r, "*", "2"'] \arrow[d, dashed, "*"', "2"] \arrow[dr, phantom, "{[Kf]}"] & \bullet \arrow[r, "*", "1"'] \arrow[d, dashed, "*"', "2"] \arrow[dr, phantom, "{[Km]}"] & \bullet \arrow[d, dashed, "*"', "2"] \\
\bullet \arrow[r, dashed, "*", "1"'] \arrow[d, "*"', "1"] \arrow[dr, phantom, "{[Kf]}"] & \bullet \arrow[r, dashed, "*", "2"'] \arrow[d, dashed, "*"', "1"] \arrow[dr, phantom, "{[Km]}"] & \bullet \arrow[r, dashed, "*", "1"'] \arrow[d, dashed, "*"', "1"] \arrow[dr, phantom, "{[Kf]}"] & \bullet \arrow[d, dashed, "*"', "1"] \\
\bullet \arrow[r, dashed, "*", "1"'] & \bullet \arrow[r, dashed, "*", "2"'] & \bullet \arrow[r, dashed, "*", "1"'] & \bullet
\end{tikzcd}$$

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

Lemma A.3.3

$$\begin{tikzcd}
\bullet \arrow[r, "1"'] \arrow[d, "2"] & \bullet \arrow[d, dashed, "*"', "2"] \\
\bullet \arrow[r, dashed, "=", "1"'] & \bullet
\end{tikzcd} \qquad \Rightarrow \qquad \xrightarrow{*}_1 \textit{ und } \xrightarrow{*}_2 \textit{ kommutieren.}$$

Beweis:

$$\begin{tikzcd}
s \arrow[r, "1"'] \arrow[d, "2"] & \bullet \arrow[r, "1"'] \arrow[d, "*"', "2"] & \cdots \arrow[r, "1"'] & t \arrow[ddd, "*"', "2"] \\
\bullet \arrow[r, "=", "1"'] \arrow[d, "2"] & \bullet \arrow[d, "*"', "2"] & & \\
\bullet \arrow[r, "=", "1"'] \arrow[d, "2"] & \bullet \arrow[d, "*"', "2"] & & \\
u \arrow[r, "=", "1"'] & \bullet \arrow[r, "=", "1"'] & \cdots \arrow[r, "=", "1"'] & \bullet
\end{tikzcd}$$

Formal: Induktion erst über die Länge von $s \rightarrow_1^* t$ und dann über die Länge von $s \rightarrow_2^* u$. □

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 λ-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.