Lambda-Kalkül

Philipp Meyer

28. April 2009

1 Einleitung

Der λ-Kalkül ist eine von Alonzo Church und Stephen Kleene in den 1930er Jahren eingeführtes Modell zur Beschreibung berechenbarer Funktionen. Es ververwendet nur fundamentale Reduktions- und Konversionsregeln zur Veränderung von textuellen Ausdrücken.

Wie Turing 1937 zeigte, ist der λ-Kalkül gleichmächtig im Sinne der Berechenbarkeit zu der Turing-Maschine. Hierraus folgt entsprechend, dass es im λ-Kalkül unentscheidbar ist, ob zwei Terme äquivalent sind oder ein Term eine Normalform besitzt.

Die typisierten Varianten des λ-Kalküls bilden die Grundlage von funktionalen Programmiersprachen wie ML oder Haskell und können auch als eine Meta-Logik verwendet werden, was zu Theorembeweisern für Logiken höherer Stufe und mächtigen Typsystemen führte.

2 Syntax

2.1 Terme

Definition 1 Die Menge Λ der λ-**Terme** ist induktiv definiert durch:

$$\frac{}{x \in \Lambda} \tag{1}$$

$$\frac{M \in \Lambda}{(\lambda x.M) \in \Lambda} \tag{2}$$

$$\frac{M, N \in \Lambda}{(MN) \in \Lambda} \tag{3}$$

Hierbei ist x eine Variable. Ein durch (2) konstruierter Term wird **Abstraktion** genannt und entspricht einer Funktionsdeklaration. Ein durch (3) konstruierter Term wird **Applikation** genannt und entspricht einer Funktionsanwendung.

Notation

- Äußere Klammern werden weggelassen: $M \equiv (M)$
- Applikation ist linksassoziativ: $MN_1 \dots N_n \equiv M\overrightarrow{N} \equiv (\dots(MN_1)\dots N_n)$
- Der Körper der Abstraktion bindet so weit nach rechts wie möglich:
  $\lambda x.M_1 \dots M_n \equiv \lambda x.\overrightarrow{M} \equiv \lambda x.(M_1 \dots M_n)$
- Bei aufeinanderfolgenden Abstraktionen werden die inneren $\lambda.$ weggelassen:
  $\lambda x_1 \dots x_n.M \equiv \lambda\overrightarrow{x}.M \equiv (\lambda x_1.(\dots(\lambda x_n.M)\dots))$

Definition 2 Die Menge der **Subterme** eines Terms sei durch die Funktion $Sub : \Lambda \mapsto \mathcal{P}(\Lambda)$ definiert:

$$\begin{aligned} Sub(x) &= \{x\} \\ Sub(\lambda x.M) &= Sub(M) \cup \{(\lambda x.M)\} \\ Sub(MN) &= Sub(M) \cup Sub(N) \cup \{(MN)\} \end{aligned}$$

2.2 Bindung und Substitution

Definition 3 Die Menge der **gebundenen Variablen** eines Terms sei durch die Funktion $BV : \Lambda \mapsto \mathcal{P}(Var)$: definiert:

$$\begin{aligned} BV(x) &= \emptyset \\ BV(\lambda x.M) &= BV(M) \cup \{x\} \\ BV(MN) &= BV(M) \cup BV(N) \end{aligned}$$

Definition 4 Die Menge der **freien Variablen** eines Terms sei durch die Funktion $FV : \Lambda \mapsto \mathcal{P}(Var)$: definiert:

$$\begin{aligned} FV(x) &= x \\ FV(\lambda x.M) &= FV(M) \backslash \{x\} \\ FV(MN) &= FV(M) \cup FV(N) \end{aligned}$$

Definition 5 Ein Term T ist **geschlossen**, falls $FV(T) = \emptyset$.

Definition 6 Die **Substitution** von x durch N in M, $M[x := N]$, ist nach dem klassischen Ansatz von Church definiert durch:

$$\begin{aligned} x[x := N] &\equiv N \\ y[x := N] &\equiv y && \text{falls } x \neq y \\ (\lambda x.M)[x := N] &\equiv \lambda x.M \\ (\lambda y.M)[x := N] &\equiv \lambda y.M[x := N] && \text{falls } x \neq y \wedge y \notin FV(N) \\ (\lambda y.M)[x := N] &\equiv \lambda z.(M[y := z])[x := N] && \text{falls } x \neq y \wedge y \in FV(N), \text{ z neue Variable} \\ (M_1 M_2)[x := N] &\equiv (M_1[x := N])(M_2[x := N]) \end{aligned}$$

Definition 7 Die Menge C der λ-**Kontexte** ist definiert durch:

$$\frac{}{x \in C}$$

$$\frac{}{[] \in C}$$

$$\frac{C[] \in C}{(\lambda x.C[]) \in C}$$

$$\frac{C_1[], C_2[] \in C}{(C_1[]C_2[]) \in C}$$

Ein Kontext ist ein λ-Term, der [] als "Löcher" enthalten kann. Ist $C[]$ ein Kontext, so bezeichnet $C[T]$ den Term, der die Löcher durch T ersetzt. Als Beispiel für den Kontext $C[] = ((\lambda x.[]x)M)$ ist $C[\lambda y.y] = (\lambda x.(\lambda y.y)x)M$.

3 Kongruenzregeln

3.1 α-Konversion

Definition 8 Die α-**Konversion** $\rightarrow_\alpha$ und die α-**Äquivalenz** $\equiv_\alpha$ sind definiert als

$$M \rightarrow_\alpha M' :\Leftrightarrow M \equiv C[\lambda x.N] \wedge M' \equiv C[\lambda y.(N[x := y])] \wedge y \notin FV(N)$$
$$M \equiv_\alpha N :\Leftrightarrow M \rightarrow_\alpha^* N$$

$M \equiv_\alpha N$ falls M und N gleich sind modulo Umbenennung gebundener Variablen.

Beispiel

$$\lambda x.x\ y \equiv_\alpha \lambda z.z\ y \not\equiv_\alpha \lambda y.y\ y$$

Konventionen

1. Wir arbeiten mit α-Äquivalenzklassen von Termen. Beispiel $\lambda x.x \equiv \lambda y.y$.
2. Gebundene Variablen werden automatisch umbenannt, so dass sie von allen freien Variablen verschieden sind. Damit wird die Substitution vereinfacht:

   $$(\lambda y.M)[x := N] \equiv \lambda y.M[x := N]$$

   Beispiel:

   $(\lambda y.x\ y)[x := F\ y] \equiv (\lambda y'.F\ y\ y')$ (y ist frei in Fy, das gebundene y wird umbenannt)

3.2 β-Reduktion

Definition 9 Ein β-**Redex** ist ein Term der Form $(\lambda x.M)N$. Wir definieren β-**Reduktion** durch

$$C[(\lambda x.M)N] \rightarrow_\beta C[M[x := N]]$$

Ein Term ist in β-**Normalform** wenn er in Normalform bezüglich $\rightarrow_\beta$ ist.

Beispiel

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

Definition 10 Alternative induktive Definition von $\rightarrow_\beta$:

1. $(\lambda x.M)N \rightarrow_\beta M[x := N]$
2. $M \rightarrow_\beta M' \quad \Rightarrow \quad (MN) \rightarrow_\beta (M'N)$
2. $M \rightarrow_\beta M' \quad \Rightarrow \quad (NM) \rightarrow_\beta (NM')$
2. $M \rightarrow_\beta M' \quad \Rightarrow \quad (\lambda x.M) \rightarrow_\beta (\lambda x.M')$

β-Reduktion ist

- nicht-terminierend. Beispiel: $\Omega := (\lambda x.x\ x)(\lambda x.x\ x) \rightarrow_\beta \Omega$
- nicht-deterministisch aber determiniert. Beispiel: $(\lambda x.x\ x)((\lambda y.y)z)$ kann zu $(\lambda x.x\ x)z$ oder $(((\lambda y.y)z)((\lambda y.y)z)$ reduziert werden, hat als Normalform aber immer $z\ z$.

3.2.1 Konfluenz

Definition 11 Ein Transitionssystem $(D, \rightarrow^*)$ heiß genau dann **konfluent**, wenn

$$\forall t, t_1, t_2 \in D.\ t \rightarrow^* t_1 \wedge t \rightarrow^* t_2 \Rightarrow \exists t' \in D.\ t_1 \rightarrow^* t' \wedge t_2 \rightarrow^* t'$$

In einem konfluenten Transitionssystems besitzt jedes Element höchstens eine Normalform. Ein Term, der auf verschiedenen Wegen ersetzt werden kann, wird nach weiteren Ersetzungen immer zum gleichen Term überführt.

Wir versuchen Konfluenz von $\rightarrow_\beta$ über die Konfluenz von $>$ zu beweisen:

Definition 12 Parallele und geschachtelte Reduktion $>$

1.	$M > M$	
2.	$\lambda x.M > \lambda x.M'$	falls $M > M'$
3.	$(M\ N) > (M'N')$	falls $M > M'$ und $N > N'$ (parallel)
4.	$(\lambda x.M)N > M'[x := N']$	falls $M > M'$ und $N > N'$ (parallel und geschachtelt)

Lemma 1 $M \rightarrow_\beta M' \Rightarrow M > M'$

Lemma 2 $M > M' \Rightarrow M \rightarrow_\beta^* M'$

Korollar 1 $>^* = \rightarrow_\beta^*$

Lemma 3 $\lambda x.M > N \Rightarrow N \equiv x.M' \wedge M > M'$

Lemma 4 *$>$ ist konfluent*

Beweis Durch Induktion über Definition von $>$ wird gezeigt: $\forall M, M_1, M_2.M > M_1 \wedge M > M_2 \Rightarrow \exists M_3.M_1 > M_3 \wedge M_2 > M_3$

- $M_1 \equiv M$: Wähle $M_3 = M_2$
- $M \equiv \lambda x.$

$$\begin{aligned}
&\Rightarrow M_2 \equiv \lambda x.P'' \wedge P > P'' && \text{(Nach Lemma 3)}\\
&\Rightarrow \exists P'''.P' > P'' \wedge P > P''' && \text{(Induktionshypothese)}\\
&\Rightarrow M_3 = \lambda x.P'''
\end{aligned}$$

- $M \equiv P\ Q \wedge M_1 \equiv P'\ Q' \wedge P > P' \wedge Q > Q'$: Hier gibt es zwei Fälle zu unterscheiden:
  1. $M_2 \equiv P''\ Q'' \wedge P > P'' \wedge Q > Q''$

     $$\begin{array}{ll} \Rightarrow \exists P'''.P' > P''' \wedge P'' > P''' & \text{(Induktionshypothese)} \\ \wedge \quad \exists Q'''.Q' > Q''' \wedge Q'' > Q''' & \text{(Induktionshypothese)} \\ \Rightarrow M_3 = P'''\ Q''' & \end{array}$$

  2. $M_2 \equiv P_1''[x := Q''] \wedge P \equiv \lambda x.P_1 \wedge P_1 > P_1'' \wedge Q > Q''$

     $$\begin{array}{ll} \Rightarrow P' \equiv \lambda x.P_1' \wedge P_1 > P_1' & \text{(Nach Lemma 3)} \\ \Rightarrow \exists P_1'''.P_1' > P_1''' \wedge P_1'' > P_1''' & \text{(Induktionshypothese)} \\ \wedge \quad \exists Q'''.Q' > Q'' \wedge Q'' > Q''' & \text{(Induktionshypothese)} \\ \Rightarrow M_3 = P_1'''\ [x := Q'''] & \end{array}$$

- $M \equiv (\lambda x.P)Q \wedge M_1 \equiv P'[x := Q'] \wedge P > P' \wedge Q > Q'$: Analog zu oben mit Fällen für $M_2 = P''[x := Q'']$ und $M_2 = (\lambda x.P'')Q''$.

Korollar 2 *$\rightarrow_\beta$ ist konfluent.*

4 Berechnung im λ-Kalkül

4.1 Boolsche Werte

true, false und if können im λ-Kalkül wie folgt definiert werden:

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

Beispiel $\texttt{if true}\ M\ N \rightarrow_\beta^* M$ und $\texttt{if false}\ M\ N \rightarrow_\beta^* N$.

4.2 Paare

Paare werden 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 $\texttt{fst}(\texttt{pair}\ x\ y) \rightarrow_\beta \texttt{fst}(\lambda z.z\ x\ y) \rightarrow_\beta (\lambda z.z\ x\ y)(\lambda xy.x) \rightarrow_\beta (\lambda xy.x)\ x\ y \rightarrow_\beta x$

4.3 Arithmetik

Eine Möglichkeit, natürliche Zahlen darzustellen sind die Church-Numerale:

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

Damit können arithmetische Funktionen definiert werden:

$$\begin{aligned}
\texttt{succ} &= \lambda n.\lambda f x.f(n\ f\ x) \\
\texttt{add} &= \lambda mn.\lambda f x.m\ f(n\ f\ x) \\
\texttt{mult} &= \lambda mn.\lambda f.m\ (n\ f) \\
\texttt{iszero} &= \lambda n.n(\lambda x.\texttt{false})\texttt{true} \\
\texttt{pred} &= \lambda n.\lambda f x.n(\lambda gh.h(g\ f))(\lambda u.x)(\lambda u.u)
\end{aligned}$$

Beispiel

$$\texttt{add}\ \underline{m}\ \underline{n} \to \lambda f x.\underline{m}\ f(\underline{n}\ f\ x) \to \lambda f x.\underline{m}\ f\ (f^n(x)) \to \lambda f x.f^m(f^n(x)) \quad = \lambda f x.f^{m+n}(x) = \underline{m+n}$$

4.4 Rekursion

Rekursion ist im λ-Kalkül nicht ohne weiteres möglich, da bei einer Funktionsdefinition der Funktionsname nicht verwendet werden kann. Möchte man eine rekursive Funktion $f(x) = e$ darstellen, so benötigt man eine nicht-rekursive Darstellung $F = \lambda f.\lambda x.e$. Die Funktion, die an F übergeben wird, muss bestimmte Eigenschaften haben, und zwar muss f zu $F(f)$ expandieren. Damit ist f ein Fixpunkt von F. So eine Funktion kann mit einem Fixpunktoperator implementiert werden. Eine bekannte Lösung ist der Church'sche Fixpunktoperator:

$$Y = \lambda f.V_f V_f \quad \text{mit} \quad V_f = \lambda x.f(x\ x)$$

Damit lässt sich f als $Y\ F$ darstellen, denn

$$Y\ F \to_\beta (\lambda x.F(x\ x))(\lambda x.F(x\ x)) \to_\beta F((\lambda x.F(x\ x))(\lambda x.F(x\ x))) = F(YF)$$

Beispiel Die Fakultätsfunktion lässt sich im λ-Kalkül folgendermaßen schreiben:

$$\texttt{fac} = Y\,\underbrace{(\lambda fac.\lambda n.\texttt{if}(\texttt{iszero}\ n)\ \underline{1}\ (\texttt{mult}\ n\ (fac(\texttt{pred}\ n))))}_{F}$$

$$\begin{aligned}
\texttt{fac}\ \underline{2} &= Y\ F\ \underline{2} \\
&\rightarrow_\beta^* F\ (Y\ F)\ \underline{2} \\
&\rightarrow_\beta^* \texttt{if}(\texttt{iszero}\ \underline{2})\ \underline{1}\ (\texttt{mult}\ \underline{2}\ (Y\ F\ (\texttt{pred}\ \underline{2}))) \\
&\rightarrow_\beta^* \texttt{mult}\ \underline{2}\ (Y\ F\ \underline{1}) \\
&\rightarrow_\beta^* \texttt{mult}\ \underline{2}\ (\texttt{mult}\ \underline{1}(Y\ F\ \underline{0})) \\
&\rightarrow_\beta^* \texttt{mult}\ \underline{2}\ (\texttt{mult}\ \underline{1}(F(Y\ F)\underline{0})) \\
&\rightarrow_\beta^* \texttt{mult}\ \underline{2}\ (\texttt{mult}\ \underline{1}(\texttt{if}(\texttt{iszero}\ \underline{0})\ \underline{1}\ \ldots) \\
&\rightarrow_\beta^* \texttt{mult}\ \underline{2}\ (\texttt{mult}\ \underline{1}\ \underline{1}) \\
&\rightarrow_\beta^* \underline{2}
\end{aligned}$$

Literatur

[1] Barendregt, Hendrik P.: *The Lambda Calculus. Its Syntax and Semantics*. Amsterdam u.a. : Elsevier, 1984

[2] Hankin, Chris: *An introduction to Lambda calculi for computer scientists*. London : King's College Publ., 2004

[3] Nipkow, Tobias: *Lambda-Kalkül*. `http://www4.in.tum.de/lehre/vorlesungen/logik/WS0607/lambda.pdf`. Version: 15. November 2004