
Logik & Semantik

13. Vorlesung

Getypte Logik – Lambda-Kalkül

Getypte Logik

Getypter λ -Kalkül

- Syntax
 - Semantik
 - Reduktions- / Ableitungssystem
-

Die Historie: Russells Paradox

Ausgangspunkt

Cantors Existenzpostulat der Mengentheorie

$$(1) (\forall P) (\exists M) (\forall x) (x \in M \equiv P(x))$$

Russells Paradox

$$(2) (\forall N) (\text{normal}(N) \equiv \neg (N \in N))$$

• Die Anwendung von Cantors Existenzpostulat ergibt:

$$(3) (\exists M) (\forall x) (x \in M \equiv \text{normal}(x))$$

$$(4) (\exists M) (\forall x) (x \in M \equiv \neg (x \in x))$$

$$(5) (\exists M) (M \in M \equiv \neg (M \in M))$$

Widerspruch ! Das Annahmensystem ist also nicht konsistent.

(1) Für alle Eigenschaften gibt es eine Menge mit genau den Objekten, auf die die Eigenschaft zutrifft.

(2) Eine Menge ist genau dann normal, wenn sie sich nicht selbst als Element enthält.

Ein Beispiel für eine normale Menge ist die leere Menge oder die Menge der natürlichen Zahlen.

Die Menge aller Mengen ist nicht normal.

(3) Es gibt eine Menge, die alle normalen Mengen enthält.

- Die Formulierung von Cantors Existenzpostulat ist der heutigen Sprechweise, der formalen Logik und Mengentheorie angepasst.
- „Russells Lösung“ führte zur sogenannten Typen-Theorie, einer spezifischen Axiomatisierung der Mengentheorie.

Die Historie: Russells Paradox

Russells Paradox ohne Mengen

(1) $(\forall P) (\text{normal}(P) \equiv \neg P(P))$

(2) $\text{normal}(\text{normal}) \equiv \neg \text{normal}(\text{normal})$

Widerspruch ! Normalität kann kein Prädikat sein.

Russells Lösung des Problems

- Die syntaktischen Objekte der Logik werden in Typen eingeteilt.
- Es gibt Standard-Objekte (Individuen).
- Es gibt Prädikate über Objekte.
- Es gibt Prädikate über Prädikate über Objekte.
- ...
- Es gibt aber keine Prädikate, die auf allen Ebenen anwendbar sind.

(1) Ein Prädikat / eine Eigenschaft ist genau dann normal, wenn sie nicht auf sich selbst zutrifft.

Ein normales Prädikat ist ‚Primzahl‘ und ‚Baum‘.

Ein nicht normales Prädikat ist ‚Prädikat‘.

Syntaktische Typen in der Logik

In der Aussagenlogik

- Formeln, ein- und zweistellige Junktoren

In der Prädikatenlogik zusätzlich

- Terme (dazu gehören Individuen-Variablen)
- Prädikate/Relationssymbole und Funktionssymbole mit zugeordneter Stelligkeit
- Quantoren

Syntaktische Typdifferenzen spiegeln unterschiedliche Arten der Interpretation wider

- Formeln werden **Wahrheitswerte** zugeordnet,
- Termen **Objekte** der Domäne,
- Prädikaten **Mengen von solchen Objekten**, etc.

- Die Grundidee der Typisierung ist auch in den Standardlogiken (Aussagen- und Prädikatenlogik) vorhanden. In der Typenlogik / getypten Logik nehmen Typen, die hier explizit gemacht sind, eine besondere Rolle ein.

Definition eines logischen Systems: Generelles Schema

Spezifikation

- einer formalen Sprache (zur Repräsentation)
- von Evaluations- / Interpretationsprinzipien
- semantischer Kategorisierungen und Beziehungen
- Ableitungs-, Beweisverfahren

Induktives Schema für Typen (\mathcal{T}_{yp}) – 1

Definition

Die Menge der *Typen* (\mathcal{T}_{yp}) ist die kleinste Menge, so dass

- **Basistypen:** *Bool* und *Ind* sind Typen.
(*Bool* \approx Formeln mit Booleschen Werten,
Ind \approx Terme mit Individuen als Werten).
- **Komplexe Typen:** Wenn *A* und *B* Typen sind,
dann ist $\langle A \rightarrow B \rangle$ ein Typ.

Der komplexe Typ $\langle A \rightarrow B \rangle$ steht für Funktionen, die Objekte vom Typ *A* auf Objekte vom Typ *B* abbilden.

A wird auch als der *Argumenttyp* und *B* als der *Typ des Wertes* bezeichnet.

- Die beiden Basistypen wurden insbesondere von Richard Montague durch *t* und *e* (anstelle von *Bool* und *Ind*) bezeichnet ((*t* \approx truth-value \rightarrow Formeln, *e* \approx entity \rightarrow Terme)). Diese Variante ist auf Montague basierend im Bereich der formalen Semantik natürlicher Sprachen vorherrschend.
- In Montagues Notation wird anstelle des Pfeils ein Komma verwendet.
- R. Montague, ein mathematischer Logiker, war der Begründer eines „Semantik-Programms“ in der Linguistik, Logik und Philosophie, dessen Intention im Titel eines Aufsatzes „English as a Formal Language“ niedergelegt ist.
Entsprechend zur formalen Syntax für natürliche Sprachen (begründet durch Noam Chomsky) soll auch die Semantik natürlicher Sprachen formal, d.h. mengentheoretisch – logisch, expliziert werden.

Syntax des typisierten λ -Kalküls (1)

Definition (Typisierung von Variablen und Konstanten)

Eine **Typen-basierte Sprache** \mathcal{L}_T (getypte Sprache, typisierte Sprache) **zu einer Menge von Typen** $\mathcal{T}yp$ wird festgelegt durch:

Für jeden Typ $\tau \in \mathcal{T}yp$ existiert

- eine abzählbar nicht-endliche Menge $Var_\tau(\mathcal{L}_T)$ von Variablen des Typs τ .
- eine abzählbare, eventuell leere Menge $Kon_\tau(\mathcal{L}_T)$ von Konstanten des Typs τ .

- Die Syntax des λ -Kalküls – und insbesondere die Verwendung des Symbols λ geht auf Alonzo Church zurück.
 - Church, Alonzo (1941). *The calculi of lambda-conversion*. *Ann. of Math. Studies* 6. Princeton NJ.
- Der λ -Kalküls wurde von Church im Rahmen von generellen Untersuchungen zur Berechenbarkeit entwickelt, nicht als ein spezifisches Instrument der Logik, d.h. der Beweistheorie.
- Später – Ende der 50er Jahre – wurde der λ -Kalküls die Grundlage der funktionalen Programmierung: McCarthys Entwicklung der Programmiersprache LISP.

Das syntaktische Inventar der Prädikatenlogik in typenlogischer Notation



	Beispiele	Typ
Formel	$\top \perp$	<i>Bool</i>
Term	$x c$	<i>Ind</i>
einstelliger Junktor	$\neg \square \diamond (\forall x) (\exists x)$	$\langle Bool \rightarrow Bool \rangle$
zweistelliger Junktor	$\wedge \vee \supset \equiv$	$\langle Bool \rightarrow \langle Bool \rightarrow Bool \rangle \rangle$
einst. Funktionssymbol		$\langle Ind \rightarrow Ind \rangle$
einst. Prädikatensymbol		$\langle Ind \rightarrow Bool \rangle$
zweist. Funktionssymbol		$\langle Ind \rightarrow \langle Ind \rightarrow Ind \rangle \rangle$
zweist. Prädikatensymbol		$\langle Ind \rightarrow \langle Ind \rightarrow Bool \rangle \rangle$

Induktives Schema für Typen ($\mathcal{T}yp$) – 2



Definition: Allgemeine Version

Gegeben eine nicht-leere Menge von **Basistypen**: $BasTyp$
Die Menge der (auf den Basistypen aufbauenden) **Typen** ($\mathcal{T}yp$)
ist die kleinste Menge für die gilt:

- $BasTyp \subseteq \mathcal{T}yp$
- Wenn $\sigma \in \mathcal{T}yp$ und $\rho \in \mathcal{T}yp$, dann $\langle \sigma \rightarrow \rho \rangle \in \mathcal{T}yp$
- Typen der Art $\langle \sigma \rightarrow \rho \rangle$ heißen **funktionale Typen**.
- Einen Typ der Art $\langle \sigma_1 \rightarrow \langle \dots \rightarrow \langle \sigma_n \rightarrow \rho \rangle \dots \rangle \rangle$
notieren wir auch als $\langle \sigma_1, \dots, \sigma_n \rightarrow \rho \rangle$

- Während bei der ersten Definition genau zwei Basistypen zugrunde gelegt wurden, wird hier in der allgemeineren Definition festgelegt, dass beliebige Mengen von Basistypen vorhanden sein können.
- Vorsicht: Typen im Sinne der Typenlogik stimmen nicht „eins-zu-eins“ mit dem Typen- bzw. Datentypen-Konzept im Bereich der Syntax und Semantik von Programmiersprachen überein.

Syntax einer typenbasierten Sprache

Definition

Es sei \mathcal{L}_T eine Typen-basierte Sprache zur Typenmenge $\mathcal{T}yp$. Für jeden Typ τ ist die Menge der Ausdrücke des Typs τ ($Exp_\tau(\mathcal{L}_T)$) die kleinste Menge, für die gilt

- $Var_\tau(\mathcal{L}_T) \subseteq Exp_\tau(\mathcal{L}_T)$ und $Kon_\tau(\mathcal{L}_T) \subseteq Exp_\tau(\mathcal{L}_T)$
- Wenn $A \in Exp_{\langle\sigma \rightarrow \tau\rangle}(\mathcal{L}_T)$ und $B \in Exp_\sigma(\mathcal{L}_T)$, dann ist $A(B) \in Exp_\tau(\mathcal{L}_T)$. *Funktionale Applikation*
- Wenn $\tau = \langle\sigma \rightarrow \rho\rangle$, $x \in Var_\sigma(\mathcal{L}_T)$ und $A \in Exp_\rho(\mathcal{L}_T)$, dann ist $(\lambda x A) \in Exp_\tau(\mathcal{L}_T) = Exp_{\langle\sigma \rightarrow \rho\rangle}(\mathcal{L}_T)$ *Funktionale Abstraktion*

Ergänzung des logischen Alphabets

λ : Kein Typ, dafür eine Regel zur Typenzuordnung

Die Ausdrücke aller Typen nennen wir auch λ -Terme.

- Die logischen Operatoren werden in das allgemeine Schema der Bildung von Ausdrücken integriert. Sie spielen also keine syntaktische Sonderrolle mehr.
- Für diesen Formalismus ist zunächst λ der einzige Operator, der eine feste Interpretation erhält.

Beispiel

Es seien x und y Variablen vom Typ τ und
 f eine Variable vom Typ $\langle \tau \rightarrow \rho \rangle$.

Ausdruck	Typ
x	τ
f	$\langle \tau \rightarrow \rho \rangle$
$f(x)$	ρ
$(\lambda x x)$	$\langle \tau \rightarrow \tau \rangle$
$(\lambda x f(x))$	$\langle \tau \rightarrow \rho \rangle$
$(\lambda x x)(y)$	τ
$(\lambda x f(x))(y)$	ρ

Ausdruck	Typ
$(\lambda f f)$	$\langle \langle \tau \rightarrow \rho \rangle \rightarrow \langle \tau \rightarrow \rho \rangle \rangle$
$(\lambda f f(x))$	$\langle \langle \tau \rightarrow \rho \rangle \rightarrow \rho \rangle$
$(\lambda x (\lambda f f(x)))$	$\langle \tau \rightarrow \langle \langle \tau \rightarrow \rho \rangle \rightarrow \rho \rangle \rangle$
$(\lambda f (\lambda x f(x)))$	$\langle \langle \tau \rightarrow \rho \rangle \rightarrow \langle \tau \rightarrow \rho \rangle \rangle$

Beobachtungen



- Die Typen-basierte Definition von Ausdrücken ist sehr einfach und uniform.
- Die Ausdrücke werden aber nicht besonders gut lesbar (wegen der Präfixnotation und der vielen Klammern).
- Wir verwenden für die binären logischen Junktoren weiter Infixnotation.
- Variablenbindung wird durch Typenzuordnung und die obige Definition nicht behandelt.
- Ist A vom Typ $\langle \sigma_1, \dots, \sigma_n \rightarrow \rho \rangle$ und B_1, \dots, B_n haben die Typen $\sigma_1, \dots, \sigma_n$, dann schreiben wir an Stelle von $A(B_1) \dots (B_n)$ auch $A(B_1, \dots, B_n)$

Definition eines logischen Systems: Generelles Schema

Spezifikation

- einer formalen Sprache (zur Repräsentation)
- von Evaluations- / Interpretationsprinzipien
- semantischer Kategorisierungen und Beziehungen
- Ableitungs-, Beweisverfahren

Semantik des typisierten λ -Kalküls (1): Rahmen



Definition

(Rahmen, Domänen für den typisierten λ -Kalkül)

Ein **Rahmen** \mathcal{D} für eine Menge von Basistypen $BasTyp$ (und die aufbauende Typmenge Typ)

ist eine Sammlung von Mengen \mathcal{D}_τ

für die Basistypen τ , genannt **Basis-Domänen**.

Für einen funktionalen Typ $\langle \sigma \rightarrow \rho \rangle$ ist

die durch \mathcal{D} zugeordnete **Domäne** $\mathcal{D}_{\langle \sigma \rightarrow \rho \rangle}$

die Menge aller Funktionen von \mathcal{D}_σ nach \mathcal{D}_ρ

$$\mathcal{D}_{\langle \sigma \rightarrow \rho \rangle} = \{f \mid f: \mathcal{D}_\sigma \rightarrow \mathcal{D}_\rho\} = \mathcal{D}_\rho^{\mathcal{D}_\sigma}$$

- Entsprechend zum Vorgehen in Sortenlogiken wird in typisierten / getypten Logiken eine strukturierte Domäne vorausgesetzt. Wichtig ist dabei, dass
 - die Strukturierung der Typisierung entspricht, d.h. für verschiedene Typen sind verschiedene Subdomänen vorzusehen.
 - die Interpretation respektiert die Typisierung, d.h. die Interpretation interpretiert Ausdrücke ihrem Typ entsprechend in der korrespondierenden Subdomäne.
 - „Typfehler“ entsprechen hierbei Typinkonsistenzen.

Semantik des typisierten λ -Kalküls (2): Modell

Definition (Modell für den typisierten λ -Kalkül)

Ein *Modell* für eine Typen-basierte Sprache \mathcal{L}_T ist ein Paar

$\mathcal{M} = \langle \mathcal{D}, \mathcal{I} \rangle$, wobei

- \mathcal{D} ein Rahmen für die Typen ist und
- $\mathcal{I}: \text{Kon}(\mathcal{L}_T) \rightarrow \mathcal{D}$ eine Abbildung (*Interpretation*) ist, die die Typisierung respektiert, d.h.
wenn $A \in \text{Kon}_\tau(\mathcal{L}_T)$, dann $\mathcal{I}(A) \in \mathcal{D}_\tau$

Semantik des typisierten λ -Kalküls (2): Belegung

Definition (Belegung)

Es sei \mathcal{D} ein Rahmen für $\mathcal{T}yp$. Eine Abbildung, $\mathcal{A}: \mathit{Var}(\mathcal{L}_{\mathcal{T}}) \rightarrow \mathcal{D}$ heißt (typgerechte) **Belegung** der Variablen in \mathcal{D} , wenn $\mathcal{A}(x) \in \mathcal{D}_{\tau}$ für $x \in \mathit{Var}_{\tau}(\mathcal{L}_{\mathcal{T}})$.

Wenn \mathcal{A} eine typgerechte Belegung in \mathcal{D} ist, x eine Variable vom Typ τ und $a \in \mathcal{D}_{\tau}$, dann ist $\mathcal{A}[x \leftarrow a]$ die typgerechte Belegung in \mathcal{D} , die x a zuweist und ansonsten mit \mathcal{A} übereinstimmt, also eine bestimmte **x -Variante** von \mathcal{A} .

- $\mathcal{A}[x \leftarrow a]$ ist die Belegung, die x auf $a \in \mathcal{D}$ abbildet und alle $y \neq x$ auf $\mathcal{A}(y)$.

Semantik des typisierten λ -Kalküls (3): Denotation



Definition (Denotation)

Es sei \mathcal{L}_T eine Typen-basierte Sprache, $\mathcal{M} = \langle \mathcal{D}, \mathcal{I} \rangle$ ein Modell für \mathcal{L}_T und $\mathcal{A}: \text{Var}(\mathcal{L}_T) \rightarrow \mathcal{D}$ eine typgerechte Belegung in \mathcal{D} . Die Denotation $F^{\mathcal{I}, \mathcal{A}}$ eines λ -Terms $A \in \text{Exp}(\mathcal{L}_T)$ ist rekursiv definiert durch:

- $x^{\mathcal{I}, \mathcal{A}} = \mathcal{A}(x)$, falls $x \in \text{Var}(\mathcal{L}_T)$
- $c^{\mathcal{I}, \mathcal{A}} = \mathcal{I}(c)$, falls $c \in \text{Kon}(\mathcal{L}_T)$
- $A(B)^{\mathcal{I}, \mathcal{A}} = A^{\mathcal{I}, \mathcal{A}}(B^{\mathcal{I}, \mathcal{A}})$
- Wenn x eine Variable vom Typ τ ist:
 $[(\lambda x A)]^{\mathcal{I}, \mathcal{A}} = f$, so dass für alle $a \in \mathcal{D}_\tau$: $f(a) = A^{\mathcal{I}, \mathcal{A}}[x \leftarrow a]$

- Diese Art der Auswertung von λ -Termen mit Variablenbindung durch λ , ist analog zur Auswertung von durch Quantoren gebundenen Variablen in der Prädikatenlogik.

Typenkorrektheit

Theorem

Es sei \mathcal{L}_T eine Typen-basierte Sprache, τ ein Typ, $\mathcal{M} = \langle \mathcal{D}, \mathcal{I} \rangle$ ein Modell für \mathcal{L}_T und $\mathcal{A}: \text{Var}(\mathcal{L}_T) \rightarrow \mathcal{D}$ eine typgerechte Belegung in \mathcal{D} . Dann gilt:

Für jeden λ -Term $A \in \text{Exp}_\tau(\mathcal{L}_T)$ ist $A^{\mathcal{I}, \mathcal{A}} \in \mathcal{D}_\tau$.

Beweis

Durch Induktion über den Termaufbau.

Beispiel

Es seien x und y Variablen vom Typ Ind und
 P eine Konstante vom Typ $\langle Ind \rightarrow Bool \rangle$.

$\mathcal{M} = \langle \mathcal{D}, \mathcal{I} \rangle$ ein Modell für \mathcal{L}_T mit
 $\mathcal{D}_{Ind} = \{1, 2, 3\}$, $\mathcal{D}_{Bool} = \{t, f\}$,

$\mathcal{I}(P) = f_P$ mit
 $f_P(1) = t$, $f_P(2) = t$, $f_P(3) = f$

$\mathcal{A}: Var(\mathcal{L}_T) \rightarrow \mathcal{D}$
mit $\mathcal{A}(x) = 2$, $\mathcal{A}(y) = 3$

A	$A^{\mathcal{I}, \mathcal{A}}$
P	f_P
x	2
y	3
$P(x)$	t
$P(y)$	f

Beispiel: $(\lambda x P(x))$

Es sei x Variable vom Typ Ind und

P eine Konstante vom Typ $\langle Ind \rightarrow Bool \rangle$.

$\mathcal{M} = \langle \mathcal{D}, \mathcal{I} \rangle$ ein Modell für \mathcal{L}_T mit $\mathcal{D}_{Ind} = \{1, 2, 3\}$, $\mathcal{D}_{Bool} = \{t, f\}$,

$\mathcal{I}(P) = f_P$ mit $f_P(1) = t$, $f_P(2) = t$, $f_P(3) = f$

$\mathcal{A}(x) = 2$

$[(\lambda x A)]^{\mathcal{I}, \mathcal{A}} = f$, so dass für alle $a \in \mathcal{D}_{Ind}$: $f(a) = A^{\mathcal{I}, \mathcal{A}[x \leftarrow a]}$

F	$F^{\mathcal{I}, \mathcal{A}} = A^{\mathcal{I}, \mathcal{A}[x \leftarrow 2]}$	$F^{\mathcal{I}, \mathcal{A}[x \leftarrow 1]}$	$F^{\mathcal{I}, \mathcal{A}[x \leftarrow 3]}$
x	2	1	3
P	f_P	f_P	f_P
$P(x)$	t	t	f
$(\lambda x P(x))$	f_P	f_P	f_P

Beispiel: $(\lambda x x)$

Es sei x Variable vom Typ Ind .

$\mathcal{M} = \langle \mathcal{D}, \mathcal{I} \rangle$ ein Modell für \mathcal{L}_T mit $\mathcal{D}_{Ind} = \{1, 2, 3\}$

$\mathcal{A}(x) = 2$

$[(\lambda x A)]^{\mathcal{I}, \mathcal{A}} = f$, so dass für alle $a \in \mathcal{D}_{Ind}$: $f(a) = A^{\mathcal{I}, \mathcal{A}[x \leftarrow a]}$

F	$F^{\mathcal{I}, \mathcal{A}} = F^{\mathcal{I}, \mathcal{A}[x \leftarrow 2]}$	$F^{\mathcal{I}, \mathcal{A}[x \leftarrow 1]}$	$F^{\mathcal{I}, \mathcal{A}[x \leftarrow 3]}$
x	2	1	3
$(\lambda x x)$	id	id	id

id: die ‚Identische Funktion‘, die jeden Wert auf sich selbst abbildet.

Also **id**(1) = 1, **id**(2) = 2, **id**(3) = 3

Beispiel

Es seien x und y Variablen vom Typ Ind und
 P eine Konstante vom Typ $\langle Ind \rightarrow Bool \rangle$.

$\mathcal{M} = \langle \mathcal{D}, \mathcal{I} \rangle$ ein Modell für \mathcal{L}_T mit
 $\mathcal{D}_{Ind} = \{1, 2, 3\}$, $\mathcal{D}_{Bool} = \{t, f\}$,

$\mathcal{I}(P) = f_P$ mit
 $f_P(1) = t$, $f_P(2) = t$, $f_P(3) = f$

$\mathcal{A}(x) = 2$, $\mathcal{A}(y) = 3$

$[(\lambda x A)]^{\mathcal{I}, \mathcal{A}} = f$, so dass
 für alle $a \in \mathcal{D}_{Ind}$: $f(a) = A^{\mathcal{I}, \mathcal{A}}[x \leftarrow a]$

A	$A^{\mathcal{I}, \mathcal{A}}$
$(\lambda x P(x))$	f_P
$(\lambda x P(x))(y)$	f
$(\lambda x P(x))(x)$	t
$(\lambda x x)$	id
$(\lambda x x)(y)$	3
$(\lambda x x)(x)$	2

- id : die ‚Identische Funktion‘, die jeden Wert auf sich selbst abbildet.
 Also $id(1) = 1$, $id(2) = 2$, $id(3) = 3$

Definition eines logischen Systems: Generelles Schema

Spezifikation

- einer formalen Sprache (zur Repräsentation)
- von Evaluations- / Interpretationsprinzipien
- **semantischer Kategorisierungen und Beziehungen**
- Ableitungs-, Beweisverfahren

Äquivalenz von λ -Termen

Definition

Es sei \mathcal{L}_T eine Typen-basierte Sprache.

Zwei λ -Terme $A, B \in \text{Exp}(\mathcal{L}_T)$ sind genau dann (*logisch*) *äquivalent*, wenn für jedes Modell $\mathcal{M} = \langle \mathcal{D}, \mathcal{I} \rangle$ und jede typgerechte Belegung \mathcal{A} in \mathcal{D} gilt:

$$A^{\mathcal{I}, \mathcal{A}} = B^{\mathcal{I}, \mathcal{A}}$$

Lemma

Wenn zwei λ -Terme A, B einer Typen-basierten Sprache \mathcal{L}_T *äquivalent* sind, dann haben sie denselben Typ.

- Diese Äquivalenzbegriff entspricht dem der Standardlogiken.
- Ausdrücke verschiedener Typen können in einzelnen Modellen durchaus denselben Wert erhalten, denn es wurde nicht erzwungen, dass die Basis-Domänen disjunkt sind.

Beispiele

Es seien x und y Variablen vom Typ τ und f eine Konstante oder Variable vom Typ $\langle \tau \rightarrow \rho \rangle$.

Dann sind äquivalent:

- $(\lambda x f(x))$ und f
- $(\lambda x f(x))$ und $(\lambda y f(y))$
- $(\lambda x x)(y)$ und y
- $(\lambda x f(x))(y)$ und $f(y)$

Freie und gebundene Variablen

Definition

Es sei \mathcal{L}_T eine Typen-basierte Sprache. Die Funktion

Free: $\text{Exp}(\mathcal{L}_T) \rightarrow 2^{\text{Var}(\mathcal{L}_T)}$, die jedem λ -Term die Menge der **freien Variablen** zuordnet, ist rekursiv definiert durch:

- **Free**(x) = $\{x\}$, falls $x \in \text{Var}(\mathcal{L}_T)$
- **Free**(c) = \emptyset , falls $c \in \text{Kon}(\mathcal{L}_T)$
- **Free**($A(B)$) = **Free**(A) \cup **Free**(B)
- **Free**($(\lambda x A)$) = **Free**(A) $- \{x\}$

Diese Festlegung zeigt, dass λ der einzige Variablenbindende Operator (in \mathcal{L}_T) ist.

- Die hier verwendete Sprechweise stimmt mit der in der Prädikatenlogik überein.

Substitution

Definition

Es sei \mathcal{L}_T eine Typen-basierte Sprache zur Typenmenge \mathcal{T}_{yp} , $\tau \in \mathcal{T}_{yp}$, $A \in \mathcal{Exp}(\mathcal{L}_T)$, $x \in \mathcal{Var}_\tau(\mathcal{L}_T)$ und $B \in \mathcal{Exp}_\tau(\mathcal{L}_T)$. Die **Substitution** $A\{x / B\}$ der freien Vorkommen von x in A durch den Term B ist rekursiv definiert durch:

- $x\{x / B\} = B$
- $y\{x / B\} = y$, falls $y \in \mathcal{Var}(\mathcal{L}_T)$ und $y \neq x$
- $c\{x / B\} = c$, falls $c \in \mathcal{Kon}(\mathcal{L}_T)$
- $[A(C)]\{x / B\} = A\{x / B\}(C\{x / B\})$
- $[(\lambda x A)]\{x / B\} = (\lambda x A)$
- $[(\lambda y A)]\{x / B\} = (\lambda y A\{x / B\})$, falls $y \neq x$

Freiheit in bezug auf Substitutionen

Definition

Der λ -Term B ist *frei für* x in A – $\mathbf{FreeFor}(B, x, A)$ – gdw gilt:

- $A \in \mathcal{V}ar(\mathcal{L}_\top)$
- $A \in \mathcal{K}on(\mathcal{L}_\top)$
- $A = C(D)$ und $\mathbf{FreeFor}(B, x, C)$ und $\mathbf{FreeFor}(B, x, D)$
- $A = (\lambda x C)$
- $A = (\lambda y C)$ und $x \neq y$, $\mathbf{FreeFor}(B, x, C)$ und entweder $x \notin \mathbf{Free}(C)$ oder $y \notin \mathbf{Free}(B)$

-
- Die vierte Bedingung schließt aus, dass eine freie Variable von B gebunden wird, wenn x in A durch B substituiert wird.

- Entsprechende Bedingungen an die Substitution existieren auch in der Prädikatenlogik.

Generelle Äquivalenzen im λ -Kalkül

Äquivalenzen für λ -Terme

- Gebundene Umbenennung
 - Wenn $y \notin \mathbf{Free}(A)$ und y frei für x in A , dann ist $(\lambda x A)$ äquivalent mit $(\lambda y A\{x / y\})$
- Applikation nach Abstraktion
 - Wenn B frei für x in A , dann ist $(\lambda x A)(B)$ äquivalent mit $A\{x / B\}$
- Abstraktion nach Applikation
 - Wenn $x \notin \mathbf{Free}(A)$, dann ist $(\lambda x A(x))$ äquivalent mit A

Definition eines logischen Systems: Generelles Schema

Spezifikation

- einer formalen Sprache (zur Repräsentation)
- von Evaluations- / Interpretationsprinzipien
- semantischer Kategorisierungen und Beziehungen
- **Ableitungs-, Beweisverfahren**

λ -Kalkül als Axiomatisches System

Axiome sind die Instanzen der folgenden Schemata

- α -Conversion / α -Reduktion [Gebundene Umbenennung]
Wenn $y \notin \mathbf{Free}(A)$ und y frei für x in A
 $\vdash_{\lambda} (\lambda x A) \Rightarrow (\lambda y A\{x / y\})$
- β -Contraction / β -Reduktion [Applikation nach Abstraktion]
Wenn B frei für x in A
 $\vdash_{\lambda} (\lambda x A)(B) \Rightarrow A\{x / B\}$
- η -Contraction / η -Reduktion [Abstraktion nach Applikation]
Wenn $x \notin \mathbf{Free}(A)$
 $\vdash_{\lambda} (\lambda x A(x)) \Rightarrow A$

- Die hier dargestellten Axiome und Ersetzungsregeln gehen auf
 - Curry, H. B. & Feys, R. (1958). *Combinatory Logic, Vol I*. Amsterdam: North Holland.

zurück.

- Das Zeichen \Rightarrow steht nicht für die logische Implikation sondern steht für die Möglichkeit einer Umformung von Ausdrücken.
- Vergleiche auch Beweise im Sequenzen-Ansatz von Gentzen zu der Beweisstruktur im λ -Kalkül.

Ableitungsregeln im λ -Kalkül

Ableitungsregeln sind die Instanzen der folgenden Schemata

- | | |
|--|-------------------------|
| • $\vdash_{\lambda} A \Rightarrow A$ | Reflexivität |
| • $A \Rightarrow B \vdash_{\lambda} A(C) \Rightarrow B(C)$ | Kongruenz (Applikation) |
| • $B \Rightarrow C \vdash_{\lambda} A(B) \Rightarrow A(C)$ | Kongruenz (Applikation) |
| • $A \Rightarrow B \vdash_{\lambda} (\lambda x A) \Rightarrow (\lambda x B)$ | Kongruenz (Abstraktion) |
| • $A \Rightarrow B, B \Rightarrow C \vdash_{\lambda} A \Rightarrow C$ | Transitivität |
| • $A \Rightarrow C, B \Rightarrow C \vdash_{\lambda} A \Leftrightarrow B$ | Ableitungs-Äquivalenz |

- Diese Ableitungsregeln erlauben es, „Beweise“ innerhalb des λ -Kalküls zu führen.

Ableitung im λ -Kalkül

Definition (Beweis im λ -Kalkül)

Ein *Beweis von* $A \Rightarrow B$ ist eine Sequenz $\vdash_{\lambda} A_1 \Rightarrow B_1, \dots, \vdash_{\lambda} A_n \Rightarrow B_n$, mit $A_n = A$ und $B_n = B$ und jedes $A_i \Rightarrow B_i$ erfüllt eine der folgenden Bedingungen:

1. $\vdash_{\lambda} A_i \Rightarrow B_i$ ist Axiom
2. $\Phi \vdash_{\lambda} A_i \Rightarrow B_i$ ist Instanz einer Ableitungsregel mit $\Phi \subseteq \{A_1 \Rightarrow B_1, \dots, A_{i-1} \Rightarrow B_{i-1}\}$

Ein *Beweis von* $A \Leftrightarrow B$ ist eine Sequenz $\vdash_{\lambda} A_1 \Rightarrow B_1, \dots, \vdash_{\lambda} A_n \Rightarrow B_n$, $\vdash_{\lambda} A \Leftrightarrow B$, jedes $A_i \Rightarrow B_i$ erfüllt eine der Bedingungen 1–2 und $\Phi \vdash_{\lambda} A \Leftrightarrow B$ ist Instanz einer Ableitungsregel mit $\Phi \subseteq \{A_1 \Rightarrow B_1, \dots, A_n \Rightarrow B_n\}$



Ableitung mit Voraussetzungen im λ -Kalkül

Definition (Beweis im λ -Kalkül)

Ein **Beweis von $A \Rightarrow B$ aus Ψ** ist eine Sequenz $\Psi \vdash_{\lambda} A_1 \Rightarrow B_1, \dots, \Psi \vdash_{\lambda} A_n \Rightarrow B_n$, mit $A_n = A$ und $B_n = B$ und jedes $A_i \Rightarrow B_i$ erfüllt eine der folgenden Bedingungen:

1. $\vdash_{\lambda} A_i \Rightarrow B_i$ ist Axiom
2. $A_i \Rightarrow B_i \in \Psi$
3. $\Phi \vdash_{\lambda} A_i \Rightarrow B_i$ ist Instanz einer Ableitungsregel mit $\Phi \subseteq \{A_1 \Rightarrow B_1, \dots, A_{i-1} \Rightarrow B_{i-1}\}$

Ein **Beweis von $A \Leftrightarrow B$ aus Ψ** ist eine Sequenz $\Psi \vdash_{\lambda} A_1 \Rightarrow B_1, \dots, \Psi \vdash_{\lambda} A_n \Rightarrow B_n$, $\Psi \vdash_{\lambda} A \Leftrightarrow B$, jedes $A_i \Rightarrow B_i$ erfüllt eine der Bedingungen 1–3 und

4. $\Phi \vdash_{\lambda} A \Leftrightarrow B$ ist Instanz einer Ableitungsregel mit $\Phi \subseteq \{A_1 \Rightarrow B_1, \dots, A_n \Rightarrow B_n\}$

- Im allgemeinen Fall ist Ψ z.B. dafür da, Abkürzungen von λ -Termen aufzunehmen. Z.B. könnte es sinnvoll sein, eine Abkürzung **perm** für eine Funktion einzuführen, die die Argumentreihenfolge einer zweistelligen Relation vertauscht. (Im typisierten Kalkül muss allerdings für jede mögliche Typenkombination eine eigene Abkürzung eingeführt werden.)
- $\text{perm}_{\langle \sigma \rightarrow \langle \tau \rightarrow \rho \rangle \rangle} \Rightarrow (\lambda f (\lambda x (\lambda y f(x)(y))))$
- wobei **x** Variable vom Typ τ , **y** Variable vom Typ σ und **f** Variable vom Typ $\langle \sigma \rightarrow \langle \tau \rightarrow \rho \rangle \rangle$ ist.

Korrektheit des λ -Kalküls

Korrektheitstheorem

Wenn $\vdash_{\lambda} A \Rightarrow B$, dann sind A und B äquivalent, d.h.

$$A^{\mathcal{I}, \mathcal{A}} = B^{\mathcal{I}, \mathcal{A}}$$

für jedes Modell $\mathcal{M} = \langle \mathcal{D}, \mathcal{I} \rangle$ und jede Belegung \mathcal{A} .

Beweisstruktur

Für jedes Axiom der Form $\vdash_{\lambda} A \Rightarrow B$ ist zu zeigen, dass A und B äquivalent sind.

Für jede Ableitungsregel ist zu zeigen: Wenn die Behauptung für die linke Seite der Regel gilt, dann auch für die rechte Seite.

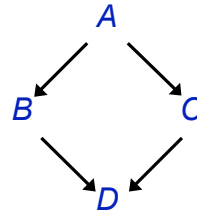
+ Induktion über die Länge des Beweises

- Das Korrektheitstheorem rechtfertigt, den λ -Kalkül als einen logischen Kalkül (Beweissystem) aufzufassen.

Church-Rosser-Eigenschaft

Church-Rosser-Theorem

Wenn $\vdash_{\lambda} A \Rightarrow B$ und $\vdash_{\lambda} A \Rightarrow C$,
dann gibt es ein D , so dass
 $\vdash_{\lambda} B \Rightarrow D$ und $\vdash_{\lambda} C \Rightarrow D$



Beweisstruktur

Zeige: Wenn B und C in einem Schritt aus A abgeleitet werden,
dann gibt es Ableitungssequenzen, die zu einem einheitlichen D
führen.

Verallgemeinerung durch Induktion über die Länge der Ableitung,
die zu B bzw. C führen.

- Das Church-Rosser-Theorem zeigt, dass die Reihenfolge von Reduktionsschritten „unerheblich“ ist: Durch weitere Reduktionsschritte ist es stets möglich, zu äquivalenten Formeln zu gelangen.
- Für den Beweis s. z.B. Barendregt, H.P. (1981). The Lambda Calculus. North-Holland: Amsterdam.

Alphabetische Varianten – Normalform

Definition

Zwei λ -Terme A , B einer Typen-basierten Sprache \mathcal{L}_T sind genau dann **alphabetische Varianten** voneinander, wenn $\vdash_\lambda A \Rightarrow B$ nachweisbar ist, wobei nur die α -Reduktion (gebundene Umbenennung) und die Ableitungsregeln verwendet wurden.

Definition

Ein λ -Term A einer Typen-basierten Sprache \mathcal{L}_T ist genau dann in **$\beta\eta$ -Normalform**, wenn es keinen Teilausdruck B von A gibt, der eine alphabetische Variante C hat, auf die die β -Reduktion oder die η -Reduktion angewendet werden kann.

Starke Normalisierung

Theorem

In einer Typen-basierten Sprache \mathcal{L}_T gibt es keine unendlich lange Sequenz von λ -Termen A_1, \dots, A_n, \dots so dass für alle $i > 0$ gilt:
 $\vdash_\lambda A_i \Rightarrow A_{i+1}$ und A_i und A_{i+1} sind keine alphabetischen Varianten voneinander.

... also

- Die Reduktionsrelation \Rightarrow ist eine Wohlordnung.
- Jeder λ -Term einer Typen-basierten Sprache \mathcal{L}_T kann in endlich vielen Schritten in eine äquivalente $\beta\eta$ -Normalform überführt werden.

- Im nicht-typisierten λ -Kalkül gilt die starke Normalisierung nicht. Damit gilt sie auch nicht für Lisp oder für ML. Evaluationsstrategien sind bei diesen Sprachen für die Termination entscheidend
- „Die Reduktionsrelation \Rightarrow ist eine Wohlordnung.“ genauer gesagt: ‚Alphabetische Variante‘ ist eine Äquivalenzrelation und $\vdash_\lambda . \Rightarrow .$ spezifiziert eine Wohlordnung für die damit gebildeten Äquivalenzklassen.

Vollständigkeit – Entscheidbarkeit

Lemma

Zwei λ -Terme A, B einer Typen-basierten Sprache \mathcal{L}_T in $\beta\eta$ -Normalform sind genau dann äquivalent, wenn sie alphabetische Varianten voneinander sind.

Theorem (Vollständigkeit)

Zwei λ -Terme A, B einer Typen-basierten Sprache \mathcal{L}_T sind genau dann logisch äquivalent, wenn $\vdash_\lambda A \Leftrightarrow B$.

Theorem (Entscheidbarkeit)

Es gibt einen Algorithmus, der immer terminiert und für zwei λ -Terme einer Typen-basierten Sprache \mathcal{L}_T feststellen kann, ob sie logisch äquivalent sind.



Reduktionen im λ -Kalkül – Funktionale Programmierung

Eine LISP Funktionsdefinition

`(DEFUN SQUARE(X) (TIMES X X))`

Hierdurch wird `SQUARE` die Abkürzung für einen λ -Term:

`SQUARE` \Rightarrow `(LAMBDA(X) (TIMES X X))`

Auswertung für den Eingabewert 2:

- `(SQUARE 2)`

Einsetzung der Definition

- `((LAMBDA(X) (TIMES X X)) 2)`

β -Reduktion

- `(TIMES 2 2)`

Eingebaut Reduktion: `(TIMES 2 2)` \Rightarrow 4

- 4

- Die Auswertung innerhalb von LISP folgt – weitgehend – dem Prinzip der funktionalen Applikation.

Reduktionsstrategien

(`SQUARE (SQUARE 2)`)

Call by value

das Argument wird zuerst reduziert: (`SQUARE 4`)

Dann wird die Funktion angewendet: 16

Call by name

zuerst wird substituiert

(`TIMES (SQUARE 2) (SQUARE 2)`)

(`TIMES (TIMES 2 2) (TIMES 2 2)`)

Dann wird die Funktion angewendet

(`TIMES 4 4`)

16

Zusammenfassung

Logische Systeme

- Einheitliche Anforderungen an Spezifikation
 - Syntax
 - Semantik
- Großes Spektrum
 - Unterschiedliche Historien, Anwendungsfelder
 - Ausdrucksmächtigkeit
 - Verarbeitbarkeit (Entscheidbarkeit, Komplexität)
- Gemeinsamkeiten
 - teilweise erst nachträglich erkannt