Vorlesung vom 14.11.05: Logik höherer Stufe I

Fahrplan

- Syntaktische Basis: Der λ-Kalkül
  - Typen und Terme
  - Substitution
  - Reduktion

Wo sind wir?

- Aussagenlogik
- Prädikatenlogik
- *Logik höherer Stufe*
- Isabelle/HOL

Der λ-Kalkül

- In den 30'ern von *Alonzo Church* als theoretisches *Berechhnungsmodell* erfunden.
- In den 60'ern Basis von *Lisp* (*John McCarthy*)
- Mathematische Basis von funktionalen Sprachen (*Haskell* etc.)
- Hier: Grundlage der *Syntax* ("higher-order abstract syntax")
  - *Typisierung*
  - *Gebundene* Variablen

Der einfach getypte λ-Kalkül ($\lambda^{\rightarrow}$)

Typen $\mathcal{T}ype$ gegeben durch

- *Typkonstanten*: $c \in \mathcal{C}_{\mathcal{T}ype}$ (Menge $\mathcal{C}_{\mathcal{T}ype}$ gegeben)
- *Funktionen*: $s, t \in \mathcal{T}ype$ dann $s \Rightarrow t$ in $\mathcal{T}ype$

Terme $\mathcal{T}erm$ gegeben durch

- *Konstanten*: $c \in \mathcal{C}$
- *Variablen*: $v \in \mathcal{V}$
- *Applikation*: $s, t \in \mathcal{T}erm$ dann $st \in \mathcal{T}erm$
- *Abstraktion*: $x \in \mathcal{V}, \tau \in \mathcal{T}ypet \in \mathcal{T}erm$ dann $\lambda\, x^{\tau}.t \in \mathcal{T}erm$
  - Typ τ kann manchmal berechnet werden.
  - Ohne Typen: *ungetypter* λ-Kalkül

Substitution

- $s \left[\begin{smallmatrix} t \\ x \end{smallmatrix}\right]$ ist Ersetzung von x durch t in s
- Definiert durch strukturelle Induktion:

$$
\begin{aligned}
c \left[\begin{smallmatrix} t \\ x \end{smallmatrix}\right] &\stackrel{def}{=} c \\
y \left[\begin{smallmatrix} t \\ x \end{smallmatrix}\right] &\stackrel{def}{=} \begin{cases} t & x = y \\ y & x \neq y \end{cases} \\
(rs) \left[\begin{smallmatrix} t \\ x \end{smallmatrix}\right] &\stackrel{def}{=} r \left[\begin{smallmatrix} t \\ x \end{smallmatrix}\right] s \left[\begin{smallmatrix} t \\ x \end{smallmatrix}\right] \\
(\lambda\, z.s) \left[\begin{smallmatrix} t \\ x \end{smallmatrix}\right] &\stackrel{def}{=} \begin{cases} \lambda\, z.s & x = z \\ \lambda\, z.s \left[\begin{smallmatrix} t \\ x \end{smallmatrix}\right] & x \neq z,\ \ z \notin FV(t) \text{ or } x \notin FV(s) \\ \lambda\, y.s \left[\begin{smallmatrix} y \\ z \end{smallmatrix}\right] \left[\begin{smallmatrix} t \\ x \end{smallmatrix}\right] & x \neq z,\ \ z \in FV(t),\ x \in FV(s), \\ & \qquad\qquad y \notin FV(ts) \end{cases}
\end{aligned}
$$

β-Reduktion

- *β-Reduktion*: $(\lambda\, x.s)t \rightarrow_{\beta} s \left[\begin{smallmatrix} t \\ x \end{smallmatrix}\right]$
- Reduktion im Kontext:

$$\frac{s \rightarrow_{\beta} s'}{st \rightarrow_{\beta} s't} \quad \frac{t \rightarrow_{\beta} t'}{st \rightarrow_{\beta} st'} \quad \frac{t \rightarrow_{\beta} t'}{\lambda\, x.t \rightarrow_{\beta} \lambda\, x.t'}$$

- $\rightarrow_{\beta}^{*}$: *Transitiv-Reflexiver* Abschluss von $\rightarrow_{\beta}$
- Exkurs: Der *ungetypte* Lambda-Kalkül als *Berechnungsmodell*
  - *Reduktion* als *Berechnung*, *Fixpunkte*, Church-Numerale

Typisierung

- *Term* t hat *Typ* τ in einem *Kontext* Γ und einer *Signatur* Σ:

$$\Gamma \vdash_{\Sigma} t : \tau$$

- *Kontext*: $x_1 : \tau_1, \ldots, x_n : \tau_n$ ($x_i \in \mathcal{V}$)
  - Typisierung von *Variablen*
- *Signatur*: $c_1 : \tau_1, \ldots, c_m : \tau_m$ ($c_i \in \mathcal{C}$)
  - Typisierung der *Konstanten*
- Vergleiche *Haskell* etc.

Typisierung

- Die Regeln:

$$\frac{c:\tau\in\Sigma}{\Gamma\vdash_\Sigma c:\tau}\,CONST \qquad \frac{x:\tau\in\Gamma}{\Gamma\vdash_\Sigma x:\tau}\,VAR$$

$$\frac{\Gamma\vdash_\Sigma s:\sigma\Rightarrow\tau \quad \Gamma\vdash_\Sigma t:\sigma}{\Gamma\vdash_\Sigma st:\tau}\,APP \qquad \frac{\Gamma,x:\sigma\vdash_\Sigma t:\tau}{\Gamma\vdash_\Sigma \lambda x^\sigma.t:\sigma\Rightarrow\tau}\,ABST$$

- *Lemma:* β-Reduktion von getypten Termen *stark normalisierend*
- *Lemma:* β-Reduktion *bewahrt Typ* (*subject reduction*)

Äquivalenzen

- *Äquivalenz*: symmetrischer Abschluß der Reduktion
- β-*Äquivalenz*:

$$=_\beta \stackrel{def}{=} (\rightarrow_\beta^* \cup {}_\beta^*\!\leftarrow)^*$$

  - $s =_\beta t$ iff. $\exists\, u. s \rightarrow_\beta^* u, t \rightarrow_\beta^* u$ (*Church-Rosser*)
- α-*Äquivalenz*: Name von gebundenen Variablen unerheblich

$$\lambda\, x.t =_\alpha \lambda\, y.t \left[{}^{y}_{x}\right] \quad y \notin FV(t)$$

  - *Implizit* (deBruijn-Indizes)

- η-*Äquivalenz*: "punktfreie" Notation

$$(\lambda\, x.tx) =_\eta t \quad x \notin FV(t)$$

Aussagenlogik in $\lambda^\rightarrow$

- Einbettung der Syntax
- *Basistypen*:
  $\mathcal{C}_{\mathcal{T}ype} = \{o\}$
- *Konstanten*:
  $\Sigma_P = \{false : o, and : o \Rightarrow o \Rightarrow o, impl : o \Rightarrow o \Rightarrow o\}$
- $\phi \in \mathcal{P}rop \longleftrightarrow \vdash_{\Sigma_P} t : o$
- Beispiel: $false \longrightarrow (a \wedge a) \in \mathcal{P}rop \longleftrightarrow (impl\ false)((and\ a)\ a)$

Prädikatenlogik in $\lambda^\rightarrow$

- *Basistypen*:
  $\mathcal{C}_{\mathcal{T}ype} = \{i, o\}$
- *Konstanten*

$$\begin{aligned}\Sigma_T &= \{0 : i, s : i \Rightarrow i, plus : i \Rightarrow i \Rightarrow i\}\\ \Sigma_A &= \{\ eq : i \Rightarrow i \Rightarrow o, false : o, and : o \Rightarrow o \Rightarrow o, \ldots,\\ &\qquad all : (i \Rightarrow o) \Rightarrow o, ex : (i \Rightarrow o) \Rightarrow o\}\\ \Sigma &= \Sigma_T \cup \Sigma_A\end{aligned}$$

- $\phi \in \mathcal{F}orm \longleftrightarrow \vdash_\Sigma t : o$
- Beispiel: $\forall\, x. \exists\, y.(x = y) \longleftrightarrow all\ (\lambda\, x.ex\ (\lambda\, y.(eq\ x)\ y))$