Formale Methoden der Softwaretechnik 1
Vorlesung vom 16.11.09: Grundlage von Isabelle

Christoph Lüth, Lutz Schröder

Universität Bremen

Wintersemester 2009/10

Fahrplan

- Teil I: Grundlagen der Formalen Logik
- Teil II: Arbeiten mit Isabelle
  - Grundlagen von Isabelle
  - Logik höherer Ordnung
  - Isabelle/HOL
  - Beweise über funktionale Programme in Isabelle/HOL
  - Beweisen mit Isabelle: Simplifikation, Taktiken
- Teil III: Modellierung imperative Programme

Fahrplan

- Grundlagen:
  - der getypte λ-Kalkül
  - Unifikation und Resolution
- Beweisen in Isabelle

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
  - Warum?

Der polymorph getypte λ-Kalkül

- Typen $\mathcal{T}ype$ gegeben durch
  - Typkonstanten: $c \in \mathcal{C}_{\mathcal{T}ype}$
  - Typvariablen: $\alpha \in \mathcal{V}_{\mathcal{T}ype}$ (Menge $\mathcal{V}_{\mathcal{T}ype}$ 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 $s\ t \in \mathcal{T}erm$
  - Abstraktion: $x \in \mathcal{V}, \tau \in \mathcal{T}ype, t \in \mathcal{T}erm$ dann $\lambda x^{\tau}.t \in \mathcal{T}erm$
    - Typ τ kann manchmal berechnet werden.
- Signatur enthält $\mathcal{C}_{\mathcal{T}ype}$, $\mathcal{C}$

Typisierung

- Signatur: $\Sigma = \langle \mathcal{C}_{\mathcal{T}ype}, \mathcal{C}, ar \rangle, ar : \mathcal{C} \to \mathcal{T}ype$
  - Typisierung der Konstanten
  - Notation: $\Sigma = \{c_1 : \tau_1, \ldots, c_n : \tau_n\}$ für $ar(c_i) = \tau_i$
- 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
- Vergleiche Haskell etc.

Typisierung

- Die Regeln:

$$\frac{c : \tau \in \Sigma}{\Gamma \vdash_\Sigma c : \tau}\ \mathit{CONST}$$

$$\frac{x : \tau \in \Gamma}{\Gamma \vdash_\Sigma x : \tau}\ \mathit{VAR}$$

$$\frac{\Gamma \vdash_\Sigma s : \sigma \Rightarrow \tau \quad \Gamma \vdash_\Sigma t : \sigma}{\Gamma \vdash_\Sigma s\ t : \tau}\ \mathit{APP}$$

$$\frac{\Gamma, x : \sigma \vdash_\Sigma t : \tau}{\Gamma \vdash_\Sigma \lambda x^\sigma . t : \sigma \Rightarrow \tau}\ \mathit{ABST}$$

Einbettungen

- Beispiel: Einbettung der Prädikatenlogik mit PA
- Basistypen:

$$\mathcal{C}_{\mathcal{T}ype} = \{i, o\}$$

- Konstanten:

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

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

Substitution

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

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

Reduktion und Äquivalenzen

- β-Reduktion: $(\lambda x.s)t \to_\beta s\left[\begin{smallmatrix} t \\ x \end{smallmatrix}\right]$
- β-Äquivalenz: $=_\beta \stackrel{def}{=} (\to_\beta^* \cup {}_\beta^* \leftarrow)^*$
  - $s =_\beta t$ iff. $\exists u.s \to_\beta^* u, t \to_\beta^* u$ (Church-Rosser)
- α-Äquivalenz: $\lambda x.t =_\alpha \lambda y.t\left[\begin{smallmatrix} y \\ x \end{smallmatrix}\right]$, $y \notin FV(t)$
  - Name von gebundenen Variablen unerheblich
  - In Isabelle Implizit (deBruijn-Indizes)
- η-Äquivalenz: $\lambda x.tx =_\eta t$, $x \notin FV(t)$
  - "punktfreie" Notation

Unifikation

- Für $s, t \in \mathcal{T}erm$ oder $s, t \in \mathcal{T}ype$ ist Substitution σ Unifikator wenn $s\sigma = t\sigma$
  - Allgemeinster Unifikator τ: alle anderen Unifikatoren sind Instanzen von τ
  - Allgemeinster Unifikator eindeutig, wenn er existiert
- Unifikationsalgorithmus:

$$\begin{array}{rcll} \tau(f\ t, g\ s) & = & \mathit{false} & \\ \tau(f\ t, f\ s) & = & \tau(t, s) & \\ \tau(t, x) & = & \left[\begin{smallmatrix} t \\ x \end{smallmatrix}\right] & x \notin \mathit{FV}t \\ \tau(y, s) & = & \left[\begin{smallmatrix} y \\ x \end{smallmatrix}\right] & y \notin \mathit{FV}s \end{array}$$

- Matching: einseitige Unifikation (σ mit $s\sigma = t$)

Grundlagen von Isabelle

- Grundlage: getypter Λ-Kalkül
- Unifikation und Matching (`apply (rule r)`)
- Äquivalenzen:
  - β-Reduktion automatisch
  - α-Äquivalenz eingebaut (deBruijn-Indexing)
  - η-Äquivalenz automatisch
- Variablen, Formeln, Resolution

Variablen

- Meta-Variablen: können unifiziert und beliebig instantiiert werden
- freie Variablen (fixed): beliebig aber fest
- Gebundene Variablen: Name beliebig (α-Äquivalenz)

Variablen

- Meta-Variablen: können unifiziert und beliebig instantiiert werden
- freie Variablen (fixed): beliebig aber fest
- Gebundene Variablen: Name beliebig (α-Äquivalenz)
- Meta-Quantoren: Isabelles Eigenvariablen

$$\frac{\bigwedge x.P(x)}{\forall x.P(x)}\ \mathit{alll} \qquad \texttt{!!x. P x ==> ALL x. P x}$$

  - Beliebig instantiierbar
  - Gültigkeit auf diese (Teil)-Formel begrenzt

Formeln

- Formeln in Isabelle:

$$\frac{\phi_1, \ldots, \phi_n}{\psi} \qquad [\![\phi_1, \ldots, \phi_n]\!] \Longrightarrow \psi$$

  - $\phi_1, \ldots, \phi_n$ Formeln, ψ atomar
  - Theoreme: ableitbare Formeln
  - Ableitung von Formeln: Resolution, Instantiierung, Gleichheit
- Randbemerkung:
  - $\Longrightarrow$, $\bigwedge$, $\equiv$ formen Meta-Logik
  - Einbettung anderer Logiken als HOL möglich — generischer Theorembeweiser

Resolution

- Einfache Resolution (`rule`)
  - Achtung: Lifting von Meta-Quantoren und Bedingungen

Resolution

- Einfache Resolution (`rule`)
  - Achtung: Lifting von Meta-Quantoren und Bedingungen
  - Randbemerkung: Unifikation höherer Stufe unentscheidbar
- Eliminationsresolution (`erule`)
- Destruktionsresolution (`drule`)

Rückwärtsbeweis

- Ausgehend von Beweisziel ψ
- Beweiszustand ist $[\![\phi_1, \ldots, \phi_n]\!] \Longrightarrow \psi$
- $\phi_1, \ldots, \phi_n$: subgoals
- Beweisverfahren: Resolution, Termersetzung, Beweissuche

Zusammenfassung

- Getypter λ-Kalkül als Grundlage der Metalogik, Modellierung von Logiken durch Einbettung
- Isabelle: Korrektheit durch Systemarchitektur
- Beweis: Rückwärts, Resolution
  - Anmerkung: auch Vorwärtsbeweis möglich
- Donnerstag: Logik höherer Ordnung in Isabelle