Techniken zur Entwicklung Korrekter Software 1

Vorlesung vom 18.12.2007: Grundlagen von Isabelle

Christoph Lüth & Lutz Schröder

WS 07/08

Wo sind wir?

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

Fahrplan

- Grundlagen:
  - der getypte λ-Kalkül
  - Unifikation und Resolution
- Isabelle: Systemarchitektur
- Beweis in Isabelle:
- Theorien

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 polymorph getypte λ-Kalkül

Typen $\mathcal{T}ype$ gegeben durch

- Typkonstanten: $c \in \mathcal{C}_{\mathcal{T}ype}$ (Menge $\mathcal{C}_{\mathcal{T}ype}$ gegeben)
- 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 $st \in \mathcal{T}erm$
- 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.

Substitution

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

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

Reduktion und Äquivalenzen

- β-Reduktion: $(\lambda x.s)t \rightarrow_\beta s\left[\begin{smallmatrix} t \\ x \end{smallmatrix}\right]$
- 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: $\lambda x.t =_\alpha \lambda y.t\left[\begin{smallmatrix} y \\ x \end{smallmatrix}\right]$, $y \notin FV(t)$
  - Name von gebundenen Variablen unerheblich
  - In Isabelle Implizit (deBruijn-Indizes)
- η-Äquivalenz: $\lambda x.tx =_\eta t$, $x \notin FV(t)$
  - "punktfreie" Notation

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.

Einbettungen

- Beispiel: Einbettung der Prädikatenlogik
- 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: $\forall x.\exists y.(x = y) \longleftrightarrow all\ (\lambda x.ex\ (\lambda y.(eq\ x)\ y))$

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:

$$\begin{array}{lcll} \tau(ft, gs) & = & & \bot \\ \tau(ft, fs) & = & & \tau(t, s) \\ \tau(t, x) & = & \left[\begin{smallmatrix} t \\ x \end{smallmatrix}\right] & x \notin FVt \\ \tau(y, s) & = & \left[\begin{smallmatrix} y \\ x \end{smallmatrix}\right] & y \notin FVs \end{array}$$

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

Isabelle Systemarchitektur: LCF-Design

- Isabelle: ca. 150 Kloc SML, ca. 310 Kloc Beweise — Korrektheit?
- Reduktion des Problems:
  - Korrektheit eines logischen Kerns
  - Rest durch Typisierung
- Abstrakter Datentyp `thm`, Inferenz-Regeln als Operationen

```
val assume: cterm -> thm
val implies_intr: cterm -> thm -> thm
```

- Logischer Kern:
  - Typcheck, Signaturen, Unifikation, Meta-Logik: ca. 5500 LOC

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

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

  - 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: vorwärts (`THEN`), oder rückwärts (`rule`)
  - Achtung: Lifting von Meta-Quantoren und Bedingungen

Resolution

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

Vorwärts oder Rückwärts?

- Vorwärtsbeweis:
  - Modifikation von Theoremen mit Attributen:
  - Resolution (`THEN`), Instantiierung (`WHERE`)
- Rückwärtsbeweis:
  - Ausgehend von Beweisziel ψ
  - Beweiszustand ist $[\![\phi_1, \ldots, \phi_n]\!] \Longrightarrow \psi$
  - $\phi_1, \ldots, \phi_n$: subgoals
  - Beweisverfahren: Resolution, Termersetzung, Beweissuche

Isabelle Theorien

- Strukturierung der Entwicklung in Theorien
- Kopf:

```
theory N
imports T1 T2 ... Tn
begin
```

- Typdeklaration: `typedecl t`
- Typsynonym: `types t = S`
- Konstantendeklaration: `consts f :: T`
- Konstantendefinitionen `constdefs f :: T  "f == E"`
  - `E` darf `f` nicht enthalten, $FV(\texttt{E}) \subseteq FV(\texttt{f})$
- Beweise: `lemma` oder `theorem`

Zusammenfassung

- Getypter λ-Kalkül als Grundlage der Metalogik, Modellierung von Logiken durch Einbettung
- Isabelle: Korrektheit durch Systemarchitektur
- Komposition von Regeln durch Unifikation und Resolution
- Beweis: Vorwärtsund Rückwärts
- Nächstes Jahr: Logik höherer Ordnung

Fröhliche Weihnachten!