Korrekte Software: Grundlagen und Methoden
Vorlesung 8 vom 19.05.16: Einführung zu Isabelle

Serge Autexier, Christoph Lüth

Universität Bremen

Sommersemester 2016

Fahrplan

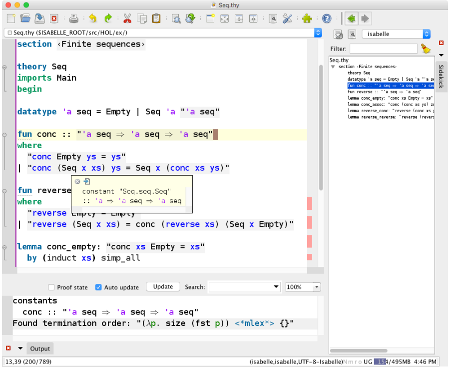
- Einführung
- Die Floyd-Hoare-Logik
- Operationale Semantik
- Denotationale Semantik
- Äquivalenz der Semantiken
- Verifikation: Vorwärts oder Rückwärts?
- Korrektheit des Hoare-Kalküls
- Einführung in Isabelle/HOL
- Weitere Datentypen: Strukturen und Felder
- Funktionen und Prozeduren
- Referenzen und Zeiger
- Frame Conditions & Modification Clauses
- Ausblick und Rückblick

Motivation

- Verwendung des interaktiven Theorembeweisers Isabelle/HOL, um anfallende Beweisverpflichtungen über C0-Software (und kommende Erweiterungen) zu beweisen.

Isabelle/HOL

- Ist ein interaktiver Theorembeweiser
- Website: `isabelle.in.tum.de`
- Basiert auf Logik HOL
- Umfangreiche Automatisierungen für Beweissuche
- High-level Syntax für Modellierung und Beweiskonstruktion
- Gute Editor-Integration (jEdit) ≈ IDE für Isabelle Theorien und Beweise
- Im Reiter "Documentation": Prog-prove, Tutorial



HOL Formeln

- HOL is ein getypte Logik höherer Ordnung (ähnlich zu funktionalen Programmiersprachen)
  - Basistypen: nat, bool, int
  - Typkonstruktoren: list , set
  - Funktionstyp: =>
  - Typvariablen: 'a 'b 'c
- Typdeklarationen:
  - op + :: nat => nat => nat
  - <= :: nat => nat => bool
  - exp2 :: nat => nat

Terme und Formeln

- Terme:
  - Infix Notation a = b, a ~= b, a <= b, a + b usw.
  - Wenn f :: t1 => t2 und t :: t1 dann ist
    f t :: t2
  - Formeln sind Terme vom Typ bool

```
True :: bool, False :: bool
not :: bool => bool                      ~, \<not>
& :: bool => bool => bool                \<and>
| :: bool => bool => bool                \<or>
--> :: bool => bool => bool              \<longrightarrow>
= :: 'a => 'a => bool
ALL x . P                                \<forall> x . P
EX x . P                                 \<exists> x . P
```

Beweiszustände

$$\bigwedge x_1 \ldots x_n.\text{assumptions} \Longrightarrow \text{conclusion}$$

$$\bigwedge x, y, z.\, \overbrace{[\![x \leq y; y \leq z]\!]} \Longrightarrow x \leq z$$

$$\bigwedge x, y, z.\, \overbrace{x \leq y \Longrightarrow y \leq z} \Longrightarrow x \leq z$$

Theorien

Dateiname: T.thy

```
theory T  (* Name muss mit Dateiname uebereinstimmen *)

imports Main (* in Main ist alles drin, was man
                so braucht / erst mal *)
begin
(* ... Definitionen, Theoreme, Beweise *)
end
```

Datentypen

```
datatype 'a list = Nil | Cons 'a "'a list"
```

- Listen von Objekten vom Typ 'a
- Nil hat als Notation auch []
- Cons x xs hat als Notation auch x#xs Erzeugt Induktionsregeln (für Beweise)

$$\frac{\text{P Nil} \qquad \bigwedge x, xs.\text{P xs} \Longrightarrow \text{P (Cons x xs)}}{\text{ALL l . P l}}$$

Konstanten

```
definition eins :: nat where "eins = Suc 0"

definition zweierliste :: "'a => 'a => 'a list" where
"zweierliste x y = x#y#[]"
```

- Erzeugt entsprechende Konstanten, aber keine Simplifikationsregeln

Funktionen

```
fun div2 :: "nat => nat" where
"div2 0 = 0" |
"div2 (Suc 0) = 0" |
"div2 (Suc (Suc n)) = Suc (div2 n)"
```

- Beweis der Terminierung automatisch (falls Fehlschlag, muss man korrigieren oder selber helfen)
- Erzeugt spezielle Induktionsregel

$$\frac{\mathrm{P}\ 0 \qquad \mathrm{P}\ (\mathrm{Suc}\ 0) \qquad \bigwedge n.\mathrm{P}\ \mathrm{n} \Longrightarrow \mathrm{P}\ (\mathrm{Suc}\ (\mathrm{Suc}\ \mathrm{n}))}{\mathrm{ALL}\ \mathrm{n}\ .\ \mathrm{P}\ \mathrm{n}}$$

- Name: `div2.induct`

Konstanten / Funktionen / Prädikate

```
fun div2 :: "nat => nat" where
"div2 0 = 0" |
"div2 (Suc 0) = 0" |
"div2 (Suc (Suc n)) = Suc (div2 n)"
```

- Beweis der Terminierung automatisch (falls Fehlschlag, muss man korrigieren oder selber helfen)

- Erzeugt spezielle Induktionsregel

$$\frac{\mathrm{P}\ 0 \qquad \mathrm{P}\ (\mathrm{Suc}\ 0) \qquad \bigwedge n. \mathrm{P}\ \mathrm{n} \Longrightarrow \mathrm{P}\ (\mathrm{Suc}\ (\mathrm{Suc}\ \mathrm{n}))}{\mathrm{ALL}\ \mathrm{n}\ .\ \mathrm{P}\ \mathrm{n}}$$

- Name: div2.induct

Theoreme und Beweise

```
lemma rev_app: "rev (app xs ys) = app (rev ys) (rev xs)"
```

- Beweiszustand
- Ein oder mehrere Unterziele
- Beweisskript bearbeitet immer das erste Unterziele
- Anwendung einer Taktik oder Regel mittels apply

Automatisierungen / Beweismethoden

- Es gibt keine vollständige Beweisverfahren für Higher-Order-Logik (HOL), aber Teile lassen sich automatisieren
- Simplifikation: simp
  - Wendet alle verwendbaren Simplifikationsregeln an
    - Datatypdefinitionen, Funktionsdefinitions (auch primrec), keine Konstanten Definitionen
    - Theoreme nur wenn sie mit [simp] gekennzeichnet sind.
  - Keywords:

```
add: <list-of-theorem-names>
del: <list-of-theorem-names>
only: <list-of-theorem-names>
```

- Etwas mehr Automatisierung: auto

Automatisierung

- Arithmetik: arith (eingebaut in simp, auto)

```
lemma "[| ~ (m < n); m < n + (1::nat)|] ==> m = n"

lemma "m ~= (n::nat) ==> (m < n | n < m)"
```

- Noch etwas mehr fastforce (auch Quantoren)

```
lemma "[|\<forall> xs \<in> A .
          \<exists> ys . xs = ys @ ys;
         us \<in> A |]
   ==> \<exists> n . length us = n + n"
```

- Noch etwas mehr: blast
- Sehr viel mehr: sledgehammer

Darüberhinaus. . .

- Fallunterscheidung: case_tac

```
apply (case_tac xs)
```

- Induktion: induct

```
apply (induct xs)
apply (induct xs :rule div2.induct)
```

- Zwischenziele einführen: subgoal_tac

```
lemma "[| A --> B; B --> C|] ==> A --> C"
```

Konstanten / Funktionen / Prädikate

```
fun forever :: "nat => nat" where
"forever 0 = 1" |
"forever (Suc n) = forever (div2 n)"
```

- Beweis der Terminierung automatisch (falls Fehlschlag, muss man korrigieren oder selber helfen)
- Erzeugt spezielle Induktionsregel

$$\frac{\mathrm{P}\ 0 \qquad \bigwedge n.\mathrm{P}\ (\mathrm{div2}\ \ \mathrm{n}) \Longrightarrow \mathrm{P}\ \mathrm{n}}{\mathrm{ALL}\ \mathrm{n}\ .\ \mathrm{P}\ \mathrm{n}}$$

- Name: div2.induct

Einzelne Regeln

- Manchmal helfen die Taktiken nicht, oder machen zu viel, und man muss einzelne Beweisschritte eingeben.
- Basisbeweisschritte sind Kalkülregeln (ähnlich wie Operationale/Axiomatische Semantik)

$$\frac{\Gamma \Longrightarrow ?P \qquad \Gamma \Longrightarrow ?Q}{\Gamma \Longrightarrow ?P \wedge ?Q}\,conjI \qquad\qquad \frac{\Gamma, ?P, ?Q \Longrightarrow G}{\Gamma, ?P \wedge ?Q \Longrightarrow G}\,conjE$$

- rule: match Conclusion und wendet Regel rückwärts an (Einführungsregeln)
- erule: match Conclusion und eine Assumption, wendet Regel an (Eliminationsregeln)
- drule: match eine Assumption, wendet Regel an und löscht verwendete Assumption
- frule: wie drule ohne das Assumption gelöscht wird.

Weitere Einführungsregeln

$$\frac{\Gamma, A \Longrightarrow B}{\Gamma \Longrightarrow A \to B}\, impI \qquad \frac{\bigwedge x.[\Gamma \Longrightarrow (?Px)]}{\Gamma \Longrightarrow \forall x.(?Px)}\, allI$$

Regeln für Gleichheit

$$\frac{\Gamma; s = t \Longrightarrow (P s)}{\Gamma; s = t \Longrightarrow (P t)} subst \qquad \frac{\Gamma; s = t \Longrightarrow (P t)}{\Gamma; s = t \Longrightarrow (P s)} subst$$

- subst, ssubst
- Parameter vorgeben: apply (rule ssubst [where t="(f x)" and s="x"])

Theoreme finden

- Theoreme sind in Lemmata oder Definitionen in importierten Theorien von Main
- Im Reiter “query” im Eingabefeld “find” kann nach Theorem gesucht werden
- Verwende Patterns um nach Struktur zu suchen (Wildcard _)
  - "_ + x = x”
- Weitere Beispiele im Tutorial auf S.34