Karlsruher Institut für Technologie

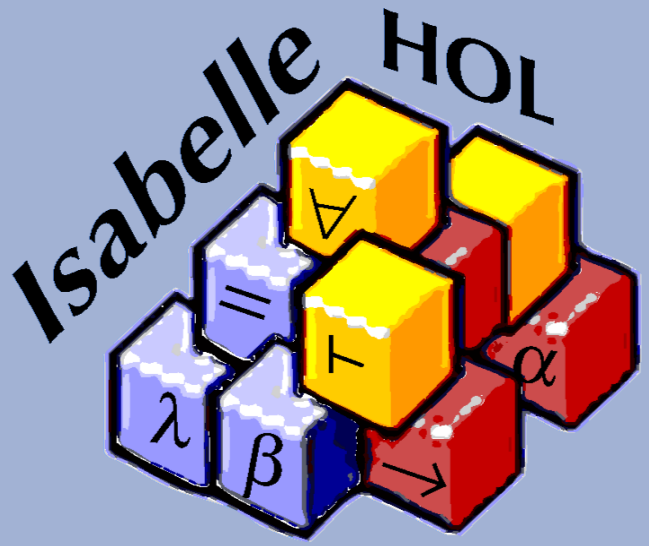
Theorembeweiserpraktikum

Anwendungen in der Sprachtechnologie

http://pp.info.uni-karlsruhe.de/lehre/SS2010/tba/

Daniel Wasserrab

IPD Snelting, Lehrstuhl Programmierparadigmen



Ziele des Praktikums

- Kennenlernen der Arbeit mit Theorembeweisern
- Erlernen des konkreten Beweisassistenten Isabelle/HOL
- Eigenständige Verifikation eines Projekts aus der Sprachtechnologie

Organisatorisches

- Termin: Di, 14.00-15.30, Praktikumspool -143, Geb. 50.34
- Unterlagen: auf der Webseite
  `http://pp.info.uni-karlsruhe.de/lehre/SS2010/tba/`
- Diplom-Studenten:
  - Veranstaltung des Hauptstudium
  - Teil der Vertiefungsfächer

    VF 1 Theoretische Grundlagen
    VF 6 Softwaretechnik und Übersetzerbau
  - Veranstaltung ist prüfbar
- Master-Studenten:
  - Veranstaltung Teil der Module

    [IN4INSPT] Sprachtechnologien
    [IN4INFM] Formale Methoden

Organisatorisches

Karlsruher Institut für Technologie

Das Praktikum teilt sich in 2 Hälften:

- 1. Hälfte à 7 Veranstaltungen:
  - Übungsblätter
  - eigenständige Bearbeitung
  - Anwesenheitspflicht!
  - Abgabe jeweils folgender Montag, 12.00 Uhr
  - per Mail an daniel.wasserrab@kit.edu
- 2. Hälfte:
  - Bearbeitung eines Projekts
  - in Zweiergruppen
  - Bearbeitungszeitraum: 01.06.-12.07.10, 12.00 Uhr
  - wieder per Mail
  - letzter Termin 13.07.10 Projektbesprechung
  - Dienstagstermine Zeit für Fragen, Problembesprechung, etc.

Was wird erwartet?

- Bearbeitung und Abgabe aller Übungsblätter (einzeln)
- Bearbeitung und Abgabe des Projekts als Zweiergruppe
- Anwesenheit an allen Übungsterminen, bei Projektvorstellung und -abschlussbesprechung
- keinerlei schriftliche Ausarbeitungen

Teil I

Was ist ein Theorembeweiser?

Erstmal ganz abstrakt...

Karlsruher Institut für Technologie

Ein Theorembeweiser beweist Aussagen über formale Strukturen durch Anwendung von Regeln.

Etwas genauer...

Ein Theorembeweiser beweist Aussagen über *formale Strukturen*
durch Anwendung von Regeln.

Etwas genauer...

Ein Theorembeweiser beweist Aussagen über *formale Strukturen* durch Anwendung von Regeln.

- Typen und Datentypen (natürliche Zahlen, Listen, Paare, . . .)

Etwas genauer...

Ein Theorembeweiser beweist Aussagen über *formale Strukturen* durch Anwendung von Regeln.

- Typen und Datentypen (natürliche Zahlen, Listen, Paare, . . .)
- Mengen, Relationen, Funktionen

Etwas genauer...

Ein Theorembeweiser beweist Aussagen über *formale Strukturen* durch Anwendung von Regeln.

- Typen und Datentypen (natürliche Zahlen, Listen, Paare, . . .)
- Mengen, Relationen, Funktionen
- funktionale Programmierung ermöglicht selbstdefinierte Strukturen (durch Rekursion, Fallunterscheidung etc.)

Etwas genauer...

Ein Theorembeweiser beweist Aussagen über *formale Strukturen* durch Anwendung von Regeln.

- Typen und Datentypen (natürliche Zahlen, Listen, Paare, . . .)
- Mengen, Relationen, Funktionen
- funktionale Programmierung ermöglicht selbstdefinierte Strukturen (durch Rekursion, Fallunterscheidung etc.)
- definiert im jeweiligen System!

Etwas genauer...

Ein Theorembeweiser *beweist Aussagen* über formale Strukturen durch Anwendung von Regeln.

automatisch:

prozedural:

deklarativ:

Etwas genauer...

Ein Theorembeweiser *beweist Aussagen* über formale Strukturen durch Anwendung von Regeln.

automatisch:

- Theorembeweiser versucht Ziel eigenständig zu lösen
- bei Nichtgelingen Meldung, woran gescheitert und Abbruch
- Hilfslemmas zeigen und Beweisprozess hinzufügen
- nochmals versuchen, Ziel zu zeigen

prozedural:

deklarativ:

Etwas genauer...

Ein Theorembeweiser *beweist Aussagen* über formale Strukturen durch Anwendung von Regeln.

automatisch:

prozedural:

- Taktiken für bestimmte automatisierte Prozesse
- können durch vorher gezeigte Hilfslemmas erweitert werden
- Beweisprozess wird nicht abgebrochen falls erfolglos, sondern an den Benutzer übergeben
- mittels Beweisskripten 'Dirigieren' der Schlussfolgerung

deklarativ:

Etwas genauer...

Ein Theorembeweiser *beweist Aussagen* über formale Strukturen durch Anwendung von Regeln.

automatisch:

prozedural:

deklarativ:

- Benutzer schreibt kompletten Beweis
- System prüft den Beweis
- bricht ab, falls Schlussfolgerung nicht korrekt

Etwas genauer...

Ein Theorembeweiser beweist Aussagen über formale Strukturen durch *Anwendung von Regeln*.

Etwas genauer...

Ein Theorembeweiser beweist Aussagen über formale Strukturen durch *Anwendung von Regeln*.

- Unifikation und Substitution

Etwas genauer...

Ein Theorembeweiser beweist Aussagen über formale Strukturen durch *Anwendung von Regeln*.

- Unifikation und Substitution
- Simplifikation

Etwas genauer...

Ein Theorembeweiser beweist Aussagen über formale Strukturen durch *Anwendung von Regeln*.

- Unifikation und Substitution
- Simplifikation
- (natürliche) Deduktion inkl. Quantoren

Etwas genauer...

Ein Theorembeweiser beweist Aussagen über formale Strukturen durch *Anwendung von Regeln*.

- Unifikation und Substitution
- Simplifikation
- (natürliche) Deduktion inkl. Quantoren
- Induktion (natürlich, wohlgeformt, strukturell, Regel-Ind., . . .)

Was ist Unifikation?

Unifikation:

- Verfahren, um zwei Terme identisch zu machen
- eventuell durch Ersetzen der schematischen Variablen durch Terme
- Anderes Beispiel: Pattern Matching

Beispiel: Unifikation der Terme ($?x, ?y, ?z$ Variablen, a Konstante)

$f(?x, g(?x, ?x))$ und $f(h(?y), g(?z, h(a)))$

Was ist Unifikation?

Karlsruher Institut für Technologie

Unifikation:

- Verfahren, um zwei Terme identisch zu machen
- eventuell durch Ersetzen der schematischen Variablen durch Terme
- Anderes Beispiel: Pattern Matching

Beispiel: Unifikation der Terme ($?x, ?y, ?z$ Variablen, a Konstante)

$f(?x, g(?x, ?x))$ und $f(h(?y), g(?z, h(a)))$

1. Schritt:

Was ist Unifikation?

Unifikation:

- Verfahren, um zwei Terme identisch zu machen
- eventuell durch Ersetzen der schematischen Variablen durch Terme
- Anderes Beispiel: Pattern Matching

Beispiel: Unifikation der Terme ($?x, ?y, ?z$ Variablen, a Konstante)

$$f(h(?y), g(h(?y), h(?y))) \qquad \text{und } f(h(?y), g(?z, h(a)))$$

1. Schritt: $?x = h(?y)$

Was ist Unifikation?

Unifikation:

- Verfahren, um zwei Terme identisch zu machen
- eventuell durch Ersetzen der schematischen Variablen durch Terme
- Anderes Beispiel: Pattern Matching

Beispiel: Unifikation der Terme ($?x, ?y, ?z$ Variablen, a Konstante)

$$f(h(?y), g(h(?y), h(?y))) \qquad \text{und } f(h(?y), g(?z, h(a)))$$

1. Schritt: $?x = h(?y)$
2. Schritt:

Was ist Unifikation?

Unifikation:

- Verfahren, um zwei Terme identisch zu machen
- eventuell durch Ersetzen der schematischen Variablen durch Terme
- Anderes Beispiel: Pattern Matching

Beispiel: Unifikation der Terme ($?x, ?y, ?z$ Variablen, a Konstante)

$$f(h(?y), g(h(?y), h(?y))) \qquad \text{und } f(h(?y), g(h(?y), h(a)))$$

1. Schritt: $?x = h(?y)$
2. Schritt: $?z = h(?y)$

Was ist Unifikation?

Unifikation:

- Verfahren, um zwei Terme identisch zu machen
- eventuell durch Ersetzen der schematischen Variablen durch Terme
- Anderes Beispiel: Pattern Matching

Beispiel: Unifikation der Terme ($?x, ?y, ?z$ Variablen, a Konstante)

$$f(h(?y), g(h(?y), h(?y))) \qquad \text{und } f(h(?y), g(h(?y), h(a)))$$

1. Schritt: $?x = h(?y)$
2. Schritt: $?z = h(?y)$
3. Schritt:

Was ist Unifikation?

Unifikation:

- Verfahren, um zwei Terme identisch zu machen
- eventuell durch Ersetzen der schematischen Variablen durch Terme
- Anderes Beispiel: Pattern Matching

Beispiel: Unifikation der Terme ($?x, ?y, ?z$ Variablen, a Konstante)

$$f(h(a), g(h(a), h(a))) \qquad \text{und } f(h(a), g(h(a), h(a)))$$

1. Schritt: $?x = h(a)$
2. Schritt: $?z = h(a)$
3. Schritt: $?y = a$

Was ist Substitution?

Substitution:
Können einen Term durch einen anderen ersetzen, wenn beide gleich sind

Regel:

$$\frac{s = t \qquad P[s/x]}{P[t/x]}$$

$P[t/x]$: ersetze x in P durch t

Was ist Deduktion?

Deduktion:

- meist Inferenzregeln (aus Prämissen folgt Konklusion)
- für jedes logische Symbol zwei Regeln:

  Introduktion: wie erhalte ich diese Formel?

  Elimination: was kann ich aus dieser Formel folgern?

Was ist Deduktion?

Deduktion:

- meist Inferenzregeln (aus Prämissen folgt Konklusion)
- für jedes logische Symbol zwei Regeln:

  Introduktion: wie erhalte ich diese Formel?

  Elimination: was kann ich aus dieser Formel folgern?

Beispiel:

Introduktion von Konjunktion: $\dfrac{P \qquad Q}{P \wedge Q}$

Elimination von Konjunktion: $\dfrac{P \wedge Q \qquad \dfrac{P \qquad Q}{R}}{R}$

mehr in den Übungen!

Was ist Induktion?

Induktion:

Was ist Induktion?

Induktion:

natürliche Induktion: zeige $P(0)$ und $P(n) \longrightarrow P(n+1)$

Was ist Induktion?

Induktion:

natürliche Induktion: zeige $P(0)$ und $P(n) \longrightarrow P(n+1)$

strukturelle Induktion: Induktion über rekursive Datentypen

Beispiel: Listen von natürlichen Zahlen $nat\ list = [] \mid nat\#nat\ list$
([] =leere Liste, # = Konkatenation)
Induktion: zeige $P([])$ und $P(ns) \longrightarrow P(n\#ns)$

Was ist Induktion?

Induktion:

natürliche Induktion: zeige $P(0)$ und $P(n) \longrightarrow P(n+1)$

strukturelle Induktion: Induktion über rekursive Datentypen

Beispiel: Listen von natürlichen Zahlen $nat\ list = [] \mid nat\#nat\ list$
([] =leere Liste, # = Konkatenation)
Induktion: zeige $P([])$ und $P(ns) \longrightarrow P(n\#ns)$

wohlgeformte Induktion: Induktion über Relationen

Beispiel: $<$ (auch: *starke Induktion*) $(\forall k < n.\ P(k)) \longrightarrow P(n)$

Was ist Induktion?

Induktion:

natürliche Induktion: zeige $P(0)$ und $P(n) \longrightarrow P(n+1)$

strukturelle Induktion: Induktion über rekursive Datentypen

Beispiel: Listen von natürlichen Zahlen $nat\ list = [] \mid nat\#nat\ list$
([] =leere Liste, # = Konkatenation)
Induktion: zeige $P([])$ und $P(ns) \longrightarrow P(n\#ns)$

wohlgeformte Induktion: Induktion über Relationen

Beispiel: $<$ (auch: *starke Induktion*) $(\forall k < n.\ P(k)) \longrightarrow P(n)$

mehr in den Übungen!

Fazit

Theorembeweiser sind mächtiges Tool, aber kein ’goldener Hammer’!

- Kann Sicherheit bzgl. Aussagen beträchtlich erhöhen
- aber ’schnell mal etwas formalisieren und beweisen’ unmöglich
- meistens werden Aussagen über Kernprobleme formalisiert und bewiesen

Fragen

Sind 'Papier und Bleistift' Beweise nicht einfacher?

Fragen

Sind 'Papier und Bleistift' Beweise nicht einfacher?

- Formalisierung in Theorembeweiser braucht viel Formalisierungsarbeit, auch für scheinbar 'triviale' Dinge

Fragen

Sind ’Papier und Bleistift’ Beweise nicht einfacher?

- Formalisierung in Theorembeweiser braucht viel Formalisierungsarbeit, auch für scheinbar ’triviale’ Dinge
- doch Beweise von Hand enthalten oftmals Fehler, vor allem für komplexe Strukturen

Fragen

Sind 'Papier und Bleistift' Beweise nicht einfacher?

- Formalisierung in Theorembeweiser braucht viel Formalisierungsarbeit, auch für scheinbar 'triviale' Dinge
- doch Beweise von Hand enthalten oftmals Fehler, vor allem für komplexe Strukturen
- Viel Aufwand, dafür garantierte Korrektheit!

Fragen

Wie kann ich sicher sein, dass meine Abstraktion das gewählte Problem beschreibt?

Fragen

Wie kann ich sicher sein, dass meine Abstraktion das gewählte Problem beschreibt?

- Im Allgemeinen: gar nicht!

Fragen

Wie kann ich sicher sein, dass meine Abstraktion das gewählte Problem beschreibt?

- Im Allgemeinen: gar nicht!
- Je genauer am konkreten Problem, desto größer die Sicherheit, aber ’Formalisierungslücke’ bleibt

Teil II

Einführung in Isabelle

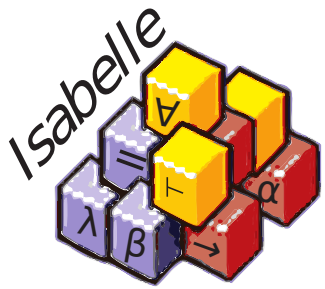
Isabelle

Autoren: Larry Paulson, Tobias Nipkow,
Markus (Makarius) Wenzel

Land: Großbritannien, Deutschland

Sprache: *SML*

Website: `isabelle.in.tum.de`



prozeduraler/deklarativer Beweiser
generisch, d.h. instantiierbar z.B. mit Typ- (HOL)
oder Mengentheorie (Zermelo-Fraenkel) (als *Objektlogiken*)

Beweiserstellung

prozedural mittels unstrukturierten Taktikskripten (“*apply-Skripten*”)

deklarativ mittels strukturierten Isar Beweisskripten
(nahe an üblicher mathematischer Notation)

Installation

Im Praktikumspool schon vorinstalliert
für eigene Installation:

- Auf Seite `http://isabelle.in.tum.de/download.html` gehen
- Isabelle, ProofGeneral, PolyML und HOL herunterladen und installieren (ist erklärt)

Starten des ProofGeneral in `emacs`: *`Isabelle-Pfad`*`/isabelle emacs`
Im Poolraum: `/usr/local/Isabelle/bin/isabelle emacs`

Die Option `-p xemacs` nach `isabelle emacs` startet ProofGeneral in `xemacs`
leider im Poolraum nicht installiert

GUIs

ProofGeneral

- Isabelle-GUI, Plugin für (X)Emacs
- Formelsyntax mittels Unicode-Zeichen
- bietet Isabelle-spezifische Menüs
- bietet Buttons zum Steuern des Beweisprozesses
  (Retract, Undo, Next, Use, Goto, Stop)

Variante: I3P

- Forschungsprototyp, ziemlich neu, wenig Erfahrungen
- verfügbar unter
  `http://www-pu.informatik.uni-tuebingen.de/i3p`
- Emacs-Kürzel (statt Ctrl-C Ctrl-Shift)
- Formalsyntax mittels XSymbols, doch ohne Super-/Sub-Scripts
- Theoriebrowser für Lemmas, Definitionen, etc.

Dateien in Isabelle

- Isabelle-Dateien haben die Endung .thy
- eine Datei beginnt mit:
  theory *Dateiname* **imports** *Basisdateiname* (Standard: **Main**) **begin**
- durch die Buttons leitet man den Beweisprozess,
  blau unterlegter Text wurde bearbeitet
- Dateiende wird durch **end** ausgedrückt

Aussagen

- In Isabelle werden zu zeigende Aussagen mit dem Schlüsselwort **lemma** eingeleitet (auch möglich: **corollary** und **theorem**)
- danach folgt optional ein Name, beendet durch :
- danach folgt die zu zeigende Aussage in Anführungszeichen
- darauf werden mittels **apply** Regeln angewandt

Regeln

- allgemeine Form: $[\![P1;\ P2;\ P3]\!] \Longrightarrow Q$
- $P1$, $P2$, $P3$ sind Prämissen der Regel (Annahmen)
- Q die Konklusion (Schlussfolgerung)
- $\Longrightarrow$ trennt Prämissen und Konklusion
  (entspricht “Bruchstrich” der Inferenzregeln)
- Also: “Wenn $P1$, $P2$ und $P3$, dann Q”
- Beispiel Modus Ponens: $[\![P \longrightarrow Q;\ P]\!] \Longrightarrow Q$

Operatoren

Es gibt folgende logische Operatoren in Isabelle/HOL:

- Negation ¬ (geschrieben \<not>)
- Konjunktion ∧ (geschrieben \<and> oder kurz: /\)
- Disjunktion ∨ (geschrieben \<or> oder kurz: \/)
- Implikation ⟶, nicht verwechseln mit ⟹!
  (geschrieben \<longrightarrow> oder kurz: −− >)
- Gleichheit =
- Ungleichheit ≠ (geschrieben \<noteq> oder kurz: = /)

Introduktion und Elimination

- Jeder Operator besitzt eine Introduktionsregel, wobei der Operator in der Konklusion steht (Standardname ... I)
  "Was brauche ich, damit die Formel gilt?"
  Beispiel: $conjI$: $[\![P;\ Q]\!] \Longrightarrow P \wedge Q$
- Jeder Operator besitzt eine Eliminationsregel, wobei der Operator in der ersten Prämisse steht (Standardname ... E)
  "Was kann ich aus der Formel folgern?"
  Beispiel: $conjE$: $[\![P \wedge Q;\ [\![P;\ Q]\!] \Longrightarrow R]\!] \Longrightarrow R$
- Regeln kann ich mir mittels **thm** *lemma-Name* anzeigen lassen

Anwendung von Regeln in Isabelle

Karlsruher Institut für Technologie

- Anwendung von Regeln spaltet das Beweisziel evtl. in subgoals auf
- Jedes subgoal muss erst gezeigt werden, bevor das nächste gezeigt werden kann
- Wenn die Konklusion einer Prämisse entspricht:
  apply assumption
- Wenn man eine (Introduktions-)Regel auf eine Konklusion anwenden möchte: **apply**(rule *Regel-Name*)
  ersetzt Beweis der Konklusion der Regel durch (meist mehrere) Beweise der Prämissen der Regel
  vorher vorhandene Prämissen werden wieder Prämissen der Beweise

Anwendung von Regeln in Isabelle

- Wenn man eine (Eliminations-)Regel auf eine Prämisse anwenden möchte: **apply**(erule *Regel-Name*)
  Vorsicht: Das aktuell zu zeigende subgoal muss mit der Konklusion der Regel unifizierbar sein!
  eliminiert die passende Prämisse und ersetzt Beweis der Konklusion der Regel durch Beweise der weiteren Prämissen der Regel
  andere vorhandene Prämissen bleiben wieder erhalten
- Wenn Isabelle meldet: `No subgoals!`, Beenden des Beweises mittels **done**

Und jetzt Sie!

Karlsruher Institut für Technologie

Viel Spass beim Ausprobieren!