

Theorembeweiserpraktikum

Anwendungen in der Sprachtechnologie

LEHRSTUHL PROGRAMMIERPARADIGMEN



Teil I

Einleitung

Ziele des Praktikums

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

- Termin: Di, 14.00 – 15.30, Praktikumpool -143, Geb. 50.34
- Unterlagen: auf der Webseite
<http://pp.info.uni-karlsruhe.de/lehre/SS2011/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

Das Praktikum teilt sich in 2 Hälften

■ 1. Hälfte à 7 Veranstaltungen:

- Übungsblätter
- eigenständige Bearbeitung
- Abgabe jeweils folgender Montag, 12.00 Uhr
- per Mail an denis.lohner@kit.edu

■ 2. Hälfte:

- Bearbeitung eines Projekts
- in Zweiergruppen
- Bearbeitungszeitraum: 01.06. – 11.07.10, 12.00 Uhr
- wieder per Mail
- letzter Termin 12.07.10 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
- keine schriftliche Ausarbeitungen

Teil II

Was ist ein Theorembeweiser?

Was ist ein Theorembeweiser?

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

Was ist ein Theorembeweiser?

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

Was ist ein Theorembeweiser?

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

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

Was ist ein Theorembeweiser?

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

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

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.)

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!

Was ist ein Theorembeweiser?

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

Was ist ein Theorembeweiser?

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

automatisch
prozedural
deklarativ

Was ist ein Theorembeweiser?

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 zum Beweisprozess hinzufügen
- nochmals versuchen, Ziel zu zeigen

prozedural

deklarativ

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

Was ist ein Theorembeweiser?

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

Was ist ein Theorembeweiser?

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

Was ist ein Theorembeweiser?

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

- Unifikation und Substitution

Was ist ein Theorembeweiser?

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

- Unifikation und Substitution
- Simplifikation

Was ist ein Theorembeweiser?

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

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

Was ist ein Theorembeweiser?

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?

- 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)

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

Was ist 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)

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

1. Schritt:

Was ist 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)

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

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

Was ist 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)

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

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

Was ist 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)

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

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

Was ist 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)

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

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

Was ist Unifikation?

- 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)

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

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

Was ist Substitution?

Ersetzung von (logisch) äquivalenten Termen.

Regel:

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

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

Was ist 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?

- 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:
$$\frac{P \quad Q}{P \wedge Q}$$

Elimination von Konjunktion:
$$\frac{P \wedge Q \quad \frac{P \quad Q}{R}}{R}$$

mehr in den Übungen!

Was ist Induktion?

Was ist Induktion?

natürliche Induktion:

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

Was ist Induktion?

natürliche Induktion:

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

strukturelle Induktion:

Induktion über rekursive Datentypen

Beispiel: Polymorphe Listen

datatype $'a \text{ list} = [] \mid 'a \# ('a \text{ list})$

($[]$ = leere Liste, $\#$ = Konkatination)

Induktion: zeige $P([])$ und $P(xs) \longrightarrow P(x \# xs)$

Was ist Induktion?

natürliche Induktion:

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

strukturelle Induktion:

Induktion über rekursive Datentypen

Beispiel: Polymorphe Listen

datatype $'a \text{ list} = [] \mid 'a \# ('a \text{ list})$

($[]$ = leere Liste, $\#$ = Konkatenation)

Induktion: zeige $P([])$ und $P(xs) \longrightarrow P(x \# xs)$

wohlgeformte Induktion:

Induktion über Relationen

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

Was ist Induktion?

natürliche Induktion:

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

strukturelle Induktion:

Induktion über rekursive Datentypen

Beispiel: Polymorphe Listen

datatype 'a list = [] | 'a # ('a list)

([] = leere Liste, # = Konkatination)

Induktion: zeige $P([])$ und $P(xs) \longrightarrow P(x \# xs)$

wohlgeformte Induktion:

Induktion über Relationen

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

mehr in den Übungen!

Theorembeweiser sind mächtig, 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

Sind 'Papier und Bleistift' Beweise nicht einfacher?

Sind 'Papier und Bleistift' Beweise nicht einfacher?

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

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

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!

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

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

- Im Allgemeinen: gar nicht!

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 III

Einführung in Isabelle/HOL

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

Land: Großbritannien, Deutschland

Sprache: *SML*

Webseite: `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)

Im Praktikumpool schon vorinstalliert unter
`/home/stud/s_lohner/Isabelle/` und
`/home/stud/s_lohner/i3p/`

Für eigene Installation:

- Auf Seite <http://isabelle.in.tum.de/download.html> gehen
- Isabelle2011-Bundle herunterladen und installieren (ist erklärt)
- I3P von <http://www-pu.informatik.uni-tuebingen.de/i3p> beziehen und entpacken

Starten

ProofGeneral (*emacs-Erweiterung*):

Isabelle-Pfad/bin/isabelle emacs

jEdit: *Isabelle-Pfad*/bin/isabelle jedit

I3P: *I3P-Pfad*/bin/i3p

ProofGeneral

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

ProofGeneral

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

I3P

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

ProofGeneral

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

I3P

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

jEdit

- Experimentelle Beweis-*IDE* als jEdit-Erweiterung
- Formelsyntax mittels Unicode-Zeichen
- Keine Super-/Sub-Scripts ?
- Theoriebrowser

- 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

- 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

- allgemeine Form: $\llbracket P_1; P_2; P_3 \rrbracket \implies Q$
- P_1, P_2, P_3 sind Prämissen der Regel (Annahmen)
- Q die Konklusion (Schlussfolgerung)
- \implies trennt Prämissen und Konklusion
(entspricht “Bruchstrich” der Inferenzregeln)
- Also: “Wenn P_1, P_2 und P_3 , dann Q ”
- Beispiel **Modus Ponens**: $\llbracket P \longrightarrow Q; P \rrbracket \implies Q$

Es gibt folgende logische Operatoren in Isabelle/HOL

<i>Name</i>	<i>Zeichen</i>	<i>Isabelle</i>	<i>Kurz</i>
Negation	\neg	<code>\<not></code>	
Konjunktion	\wedge	<code>\<and></code>	<code>/\</code>
Disjunktion	\vee	<code>\<or></code>	<code>\ </code>
Implikation	\longrightarrow	<code>\<longrightarrow></code>	<code>-- ></code>
Achtung: nicht verwechseln mit \implies			
Gleichheit	$=$		
Ungleichheit	\neq	<code>\<noteq></code>	<code>= /</code>

Ein **Bug** in emacs lässt diesen bei der Eingabe mancher Operatoren abstürzen

\implies Abhilfe: Statt z.B. `\<not>` nur `\not` eingeben

- Jeder Operator besitzt eine Introduktionsregel, wobei der Operator in der Konklusion steht (Standardname ...*I*)

“Was brauche ich, damit die Formel gilt?”

Beispiel: $conjI: \llbracket P; Q \rrbracket \Longrightarrow P \wedge Q$

- Jeder Operator besitzt eine Eliminationsregel, wobei der Operator in der ersten Prämisse steht (Standardname ...*E*)

“Was kann ich aus der Formel folgern?”

Beispiel: $conjE: \llbracket P \wedge Q; \llbracket P; Q \rrbracket \Longrightarrow R \rrbracket \Longrightarrow R$

- Regeln kann man mittels **thm** *lemma-Name* anzeigen lassen

Anwendung von Regeln in Isabelle

- Anwendung von Regeln spaltet das Beweisziel evtl. in Teilziele auf

Anwendung von Regeln in Isabelle

- Anwendung von Regeln spaltet das Beweisziel evtl. in Teilziele auf
- Jedes Teilziel muss erst gezeigt werden, bevor das nächste gezeigt werden kann

- Anwendung von Regeln spaltet das Beweisziel evtl. in Teilziele auf
- Jedes Teilziel muss erst gezeigt werden, bevor das nächste gezeigt werden kann
- Wenn die Konklusion des aktuellen Ziels einer (beliebigen) Prämisse entspricht:

apply assumption

- Anwendung von Regeln spaltet das Beweisziel evtl. in Teilziele auf
- Jedes Teilziel muss erst gezeigt werden, bevor das nächste gezeigt werden kann
- Wenn die Konklusion des aktuellen Ziels einer (beliebigen) Prämisse entspricht:

apply assumption

- Wenn man eine (Introduktions-)Regel auf die **Konklusion** des aktuellen Ziels anwenden möchte:

apply (rule *Regel-Name*)

- Ersetzt Beweis der Konklusion durch (meist mehrere) Beweise der Prämissen der Regel.
- Prämissen des aktuellen Beweisziels bleiben erhalten.

- Wenn man eine (Eliminations-)Regel auf eine **Prämisse** des aktuellen Beweisziels anwenden möchte:

apply (erule *Regel-Name*)

- Eliminiert die passende Prämisse und ersetzt Beweis der Konklusion durch Beweise der weiteren Prämissen der Regel.
- Verbleibende Prämissen bleiben erhalten.
- **Vorsicht:** Die Konklusion des aktuellen Beweisziels muss mit der Konklusion der Regel unifizierbar sein!

- Wenn man eine (Eliminations-)Regel auf eine **Prämisse** des aktuellen Beweisziels anwenden möchte:

apply (*erule Regel-Name*)

- Eliminiert die passende Prämisse und ersetzt Beweis der Konklusion durch Beweise der weiteren Prämissen der Regel.
 - Verbleibende Prämissen bleiben erhalten.
 - **Vorsicht:** Die Konklusion des aktuellen Beweisziels muss mit der Konklusion der Regel unifizierbar sein!
- Wenn Isabelle meldet: `No subgoals!`, Beenden des Beweises mittels **done**

Und jetzt Sie

Viel Spaß beim Ausprobieren!