

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.ipd.kit.edu/lehre/SS2013/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
 - [IN4INPRAK2] Informatik-Praktikum 2

Das Praktikum teilt sich in 2 Hälften

■ 1. Hälfte à 7 Veranstaltungen:

- Übungsblätter
- eigenständige Bearbeitung
- Abgabe jeweils folgender Montag, 12.00 Uhr
- Über <https://praktomat.info.uni-karlsruhe.de/>, SCC-Login

■ 2. Hälfte:

- Bearbeitung eines Projekts
- in Zweiergruppen
- Bearbeitungszeitraum: 4.06. – 8.7.2013, 12.00 Uhr
- wieder über den Praktomaten
- 9.7.2013: Informationen zur Projektpräsentation
- letzter Termin 16.7.2013, **16.00 Uhr**, Projektpräsentation im Oberseminar
- An den Dienstagstermine Zeit für Fragen, Problembesprechung, etc (bitte vorher Bescheid geben)

Was wird erwartet?

- Bearbeitung und Abgabe aller Übungsblätter (einzeln)
- Bearbeitung und Abgabe des Projekts als Zweiergruppe
- Anwesenheit an allen Übungsterminen, bei Projektvorstellung und -präsentation
- kurze Abschlusspräsentation im Oberseminar des Lehrstuhls
- 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

Was ist ein Theorembeweiser?

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
- nicht korrekte Schlussfolgerungen werden aufgezeigt

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

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

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

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)
- lassen sich in zwei Klassen aufteilen:

Introduktion: wie erhalte ich diese Formel?

Elimination: was kann ich aus dieser Formel folgern?

Was ist Deduktion?

- meist Inferenzregeln (aus Prämissen folgt Konklusion)
- lassen sich in zwei Klassen aufteilen:

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 $P(n) \rightarrow P(n+1)$

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 $'a\ list = [] \mid 'a \# ('a\ list)$

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

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 $'a\ list = [] \mid 'a \# ('a\ list)$

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

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

wohlgeformte Induktion:

Induktion über Relationen

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

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
`/opt/Isabelle2013/bin/isabelle`

Für eigene Installation:

- Auf Seite <http://isabelle.in.tum.de/download.html> gehen
- Isabelle2013-Bundle herunterladen und installieren (ist erklärt)

Starten:

- `Isabelle-Pfad/bin/isabelle jedit`

- Isabelle-Dateien haben die Endung `.thy`
- Eine Datei beginnt mit:
theory *Dateiname* **imports** *Basisdateiname* (Standard: **Main**) **begin**
- Dann folgt die Formalisierung, die automatisch im Hintergrund geprüft wird
- Wichtig sind dabei die Informationen im Fenster „Output“
- Das Ende der Datei wird mit **end** markiert

- 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>Anzeige</i>	<i>Sourcecode</i>	<i>JEdit-Kürzel</i>
Negation	\neg	<code>\<not></code>	<code>~</code>
Konjunktion	\wedge	<code>\<and></code>	<code>/\</code>
Disjunktion	\vee	<code>\<or></code>	<code>\/</code>
Implikation	\longrightarrow	<code>\<longrightarrow></code>	<code>-- ></code>
Gleichheit	$=$		
Ungleichheit	\neq	<code>\<noteq></code>	<code>~=</code>

Die JEdit-Kürzel werden mit einem TAB abgeschlossen.

Achtung: \longrightarrow und \implies sind verschieden

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

“Was brauche ich, damit die Formel gilt?”

Beispiel: $\text{conj}I: \llbracket P; Q \rrbracket \Longrightarrow P \wedge Q$

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

“Was kann ich aus der Formel folgern?”

Beispiel: $\text{conj}E: \llbracket P \wedge Q; \llbracket P; Q \rrbracket \Longrightarrow R \rrbracket \Longrightarrow R$

- Regeln kann man mittels **thm** $\langle \text{lemma-Name} \rangle$ anzeigen lassen

- 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

Beispiel:

lemma *imp_uncurry*: " $(P \longrightarrow (Q \longrightarrow R)) \longrightarrow P \wedge Q \longrightarrow R$ "

Dem Lemma folgt dann der Beweis...

- Ein Beweis beginnt mit **proof** (*rule* \langle Regel \rangle) und endet mit **qed**
- Dazwischen werden Zwischenschritte angegeben, in drei Varianten:
 - **assume** " \langle Aussage \rangle " führt eine Aussage ein, die angenommen wird (also nicht bewiesen werden muss).
 - **have** " \langle Aussage \rangle " \langle Beweis \rangle führt eine bewiesene Hilfsaussage ein.
 - **show** " \langle Aussage \rangle " \langle Beweis \rangle beweist eine Aussage, die einen Fall des Beweises abschließt.
- Mehrere Fälle werden durch **next** getrennt

proof unifiziert die Konklusion der Regel mit der zu zeigenden Aussage. Die Prämissen der Regel sind die zu zeigenden Teilziele.

Zur Erinnerung:

$$\text{conjI: } \llbracket P; Q \rrbracket \implies P \wedge Q$$

Beispiel:

```
lemma "foo  $\wedge$  bar"  
proof (rule conjI)  
  show "foo"  $\langle$ proof $\rangle$   
next  
  show "bar"  $\langle$ proof $\rangle$   
qed
```

- Aussagen, die der **proof** von **have** und **show** direkt verwenden soll, werden mit **from** aufgezählt. Sie werden benannt durch
 - Die Aussage in Backticks (``⟨Aussage⟩``),
 - oder Namen, die der Aussage optional mit Doppelpunkt vorangestellt wurden oder
 - mit *this*, was stets die letzte gemachte Aussage ist.
- Diese werden, in der angegebenen Reihenfolge, mit den (Konklusionen der) Annahmen der Regel unifiziert
- Bei „**proof**–“ wird *keine* Regel angewandt.
- Bei **qed** dürfen nur noch Fälle offen sein, deren Ziele bereits unter den aufgesammelten Annahmen sind.

Beispiel:

from ``A ∧ B``

have "A"

proof (*rule conjE*)

assume "A"

from *this*

show "A".

qed

Zur Erinnerung:

conjE: $\llbracket P \wedge Q ; \llbracket P ; Q \rrbracket \implies R \rrbracket \implies R$

- **then** \equiv **from** *this*
- **with** *a b* \equiv **from** *a b this* (Reihenfolge beachten!)
- **hence** \equiv **then have**
- **thus** \equiv **then show**
- **by** (*rule* \langle Regel \rangle) \equiv **proof** (*rule* \langle Regel \rangle) **qed**
- **proof** \equiv **proof** (*rule*) (passende Regel wird automatisch gewählt)
- **..** \equiv **by** (*rule*)
- **.** \equiv **by-** \equiv **proof-** **qed**

- **then** \equiv **from** *this*
- **with** *a b* \equiv **from** *a b this* (Reihenfolge beachten!)
- **hence** \equiv **then have**
- **thus** \equiv **then show**
- **by** (*rule* \langle Regel \rangle) \equiv **proof** (*rule* \langle Regel \rangle) **qed**
- **proof** \equiv **proof** (*rule*) (passende Regel wird automatisch gewählt)
- **..** \equiv **by** (*rule*)
- **.** \equiv **by-** \equiv **proof-** **qed**

Und Abkürzungen anderer Art sind die Beweise

- **oops**: Bricht den aktuellen Beweis ab.
- **sorry**: Beweist alles (und sollte in fertigen Theorien nicht stehen).

Und jetzt Sie

Viel Spaß beim Ausprobieren!