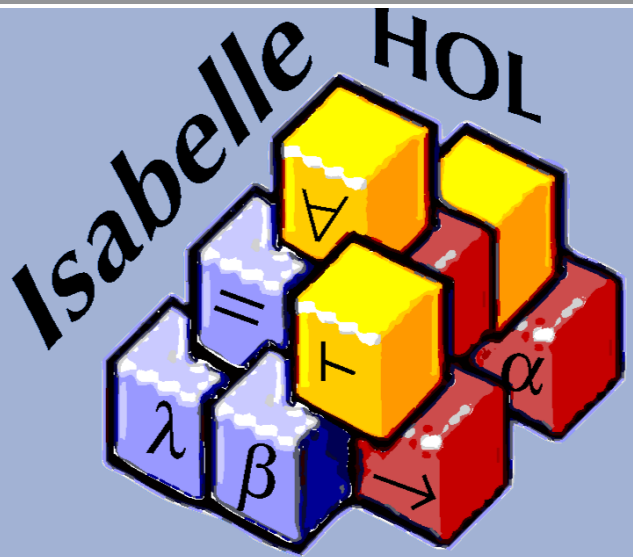
Karlsruher Institut für Technologie

Theorembeweiserpraktikum

Anwendungen in der Sprachtechnologie

LEHRSTUHL PROGRAMMIERPARADIGMEN



Karlsruher Institut für Technologie

Teil XII

Strukturierte Beweise mittels Isar

Was ist Isar?

apply-Skripten für externen Leser völlig unverständlich

Was ist Isar?

apply-Skripten für externen Leser völlig unverständlich

Lösung: Isar!

Was ist Isar?

apply-Skripten für externen Leser völlig unverständlich

Lösung: Isar!

- Vater des Gedanken: Mizar
- bessere Strukturierung durch Herleitung von “Zwischenaussagen”
- aussagekräftigere Schlüsselwörter (ähnlich mathematischer Notation)
- (gut geschriebener) Beweis verständlich ohne aktuellen Beweiszustand
- einfachere Anpassung bei Änderungen

Ein einfaches Beispiel

```
lemma imp_refl: shows "A ⟶ A"
proof (rule impI)
  assume "A"
  from `A` show "A" by (rule a)
qed
```

Ein einfaches Beispiel

```
lemma imp_refl: shows "A ⟶ A"
proof (rule impI)
  assume "A"
  from `A` show "A" by (rule a)
qed
```

allgemeiner Aufbau:

- Lemmadefinition wie gewohnt, oder besser mit **assumes** und **shows**
- dann **proof**, gefolgt von Beweismethode
- Prämissen des aktuellen subgoals mit **assume** dargestellt (können auch Namen erhalten)
- Ziel des aktuellen subgoals mittels **show** ausgedrückt, danach folgt Beweis der Aussage
- **qed** als “schliessende Klammer” für jedes **proof**

Wie beweise ich in Isar?

Ziel in Isar:

für jeden Beweis nur die dafür nötigen Prämissen verwenden

Dafür: Zwischenaussagen beweisen

- Bilden Prämissen für nächste Zwischenaussage oder Beweisziel
- Benötigte Prämissen/Zwischenaussagen mit **from** aufsammeln
- Schlüsselwort **have** leitet Beweis einer Zwischenaussage ein
- Beweis der Zwischenaussage mit Isar-**proof** oder **apply**-Skript

Wie beweise ich in Isar?

Karlsruher Institut für Technologie

Ziel in Isar:

für jeden Beweis nur die dafür nötigen Prämissen verwenden

Dafür: Zwischenaussagen beweisen

- Bilden Prämissen für nächste Zwischenaussage oder Beweisziel
- Benötigte Prämissen/Zwischenaussagen mit **from** aufsammeln
- Schlüsselwort **have** leitet Beweis einer Zwischenaussage ein
- Beweis der Zwischenaussage mit Isar-**proof** oder **apply**-Skript

Syntax

from *`prem1` `prem2` named_prem [...]* **have** *new_name: "P"*

Wie beweise ich in Isar?

Beispiel:

```
lemma "A ∧ B ⟶ B ∧ A"
proof (rule impI)
  assume ab: "A ∧ B"
  from ab have a: "A" by -(erule conjE)
  from `A ∧ B` have "B" by (rule conjE)
  from `B` a show "B ∧ A" by (rule conjI)
qed
```

Wie beweise ich in Isar?

Beispiel:

```
lemma "A ∧ B ⟶ B ∧ A"
proof (rule impI)
  assume ab: "A ∧ B"
  from ab have a: "A" by -(erule conjE)
  from `A ∧ B` have "B" by (rule conjE)
  from `B` a show "B ∧ A" by (rule conjI)
qed
```

hier werden Namen und direkte Zitationen fröhlich gemischt,
in sauberem Beweis besser

- direkt zitieren bei kurzen Aussagen,
- Namen für längere Aussagen

direkt zitieren ist lesbarer, Namen anpassbarer bei Änderungen

Strukturieren des Beweises

aktuell gezeigte oder angenommene Aussage mittels *this* verfügbar:

lemma *"A ⟶ A"*
proof *(rule impI)*
  assume *"A"*
  from *this* **show** *"A"* **by** *assumption*
qed

also Beweis möglichst so strukturieren, dass jede Aussage auf der vorherigen aufbaut

Strukturieren des Beweises

aktuell gezeigte oder angenommene Aussage mittels *this* verfügbar:

lemma *"A ⟶ A"*
proof *(rule impI)*
  assume *"A"*
  from *this* **show** *"A"* **by** *assumption*
qed

also Beweis möglichst so strukturieren, dass jede Aussage auf der vorherigen aufbaut

Syntaktische Abkürzungen:

- **from** *this* ≡ **then** (nur diese eine Annahme!)
- **from** *a b this* ≡ **with** *a b* (mehrere Annahmen)
- **from** *this* **show** ≡ **then show** ≡ **thus**
- **from** *this* **have** ≡ **then have** ≡ **hence**

proof ohne Regel

wenn nach **proof** keine Methode spezifiziert ist, wird (falls möglich) sinnvolle erste (Introduktions- oder Eliminations-) Regel angewandt

$\Rightarrow$ Kann zu unbeweisbaren Zielen führen (z.B. Disjunktion in Konklusion)

proof ohne Regel

wenn nach **proof** keine Methode spezifiziert ist, wird (falls möglich) sinnvolle erste (Introduktions- oder Eliminations-) Regel angewandt

$\Rightarrow$ Kann zu unbeweisbaren Zielen führen (z.B. Disjunktion in Konklusion)

```
lemma assumes "A ∨ B" and "A ⟶ Q"
  shows "(B ⟶ Q) ⟶ Q"
proof              — verwendet implizit (rule impI)
  assume "B ⟶ Q"
  from `A ∨ B` show "Q"
  proof            — verwendet implizit (erule disjE)
    assume "A"
    from `A` `A ⟶ Q` show ?thesis by simp
  next
    assume "B"
    with `B ⟶ Q` show ?thesis by (rule mp)
  qed
qed
```

mit `?thesis` wird aktuelles Ziel abgekürzt, hier `Q`

Methoden für proof

proof entweder explizit übergebene Introduktions- oder Eliminationsregel oder implizit automatisch gewählte (wenn Beweismethode weggelassen)

Methoden für proof

proof entweder explizit übergebene Introduktions- oder Eliminationsregel oder implizit automatisch gewählte (wenn Beweismethode weggelassen)

weitere Möglichkeiten:

- keine Beweismethode passend oder Automatismus nicht gewollt:
  proof `-`: Beweisziel unverändert
- Fallunterscheidung durch **proof** *`(cases P)`*
- strukturelle Induktion über Datentyp *`D`* durch **proof** *`(induct D)`*
- Regelinduktion (für **fun** und **inductive**) analog
  proof *`(induct rule:bla.induct)`*

Bitte kein **proof** *`simp`* oder **proof** *`auto`*!

Methoden für proof

Fallunterscheidung und andere Methoden erzeugen mehrere Teilziele.
Wenn erstes bewiesen, Übergang zum nächsten mittels **next** (statt **qed**)

Methoden für proof

Fallunterscheidung und andere Methoden erzeugen mehrere Teilziele.
Wenn erstes bewiesen, Übergang zum nächsten mittels **next** (statt **qed**)

```
lemma "(A ⟶ B) ∧ (¬ A ⟶ B) ⟶ B"
proof
  assume "(A ⟶ B) ∧ (¬ A ⟶ B)"
  hence "A ⟶ B" and "¬ A ⟶ B" by -(erule conjE|assumption)+
  show "B"
  proof (cases A)
    assume "A"
    with `A ⟶ B` show ?thesis by (rule mp)
  next
    assume "¬ A"
    with `¬ A ⟶ B` show ?thesis by (rule mp)
  qed
qed
```

Lemma mit Prämissen und Konklusion

Lemma besteht aus Prämissen und Konklusion, dann

- Liste der Prämissen nach Schlüsselwort **assumes**, evtl. mit Namen und mit **and** getrennt
- Konklusion nach **shows**
- Lemmaname kann entweder wie bisher vor allen **assumes** angegeben werden oder direkt vor **shows**

Prämissen können dann im Beweisskript zitiert werden

Lemma mit Prämissen und Konklusion

Lemma besteht aus Prämissen und Konklusion, dann

- Liste der Prämissen nach Schlüsselwort **assumes**, evtl. mit Namen und mit **and** getrennt
- Konklusion nach **shows**
- Lemmaname kann entweder wie bisher vor allen **assumes** angegeben werden oder direkt vor **shows**

Prämissen können dann im Beweisskript zitiert werden

so wird aus **lemma** `bla: "⟦P; Q⟧ ⟹ R"`
lemma assumes `"P"` **and** `"Q"` **shows** `bla:"R"`

Lemma mit Prämissen und Konklusion

Vorsicht! Induktionsregeln o.ä. benötigen eine oder mehrere Prämissen, um die Regel anwenden zu können
in dieser Schreibform: angeben, welche Prämissen für Regel nötig sind
Syntax: vor **proof** hinter Schlüsselwort **using** Namen der benötigten Prämissen

Lemma mit Prämissen und Konklusion

Vorsicht! Induktionsregeln o.ä. benötigen eine oder mehrere Prämissen, um die Regel anwenden zu können
in dieser Schreibform: angeben, welche Prämissen für Regel nötig sind
Syntax: vor **proof** hinter Schlüsselwort **using** Namen der benötigten Prämissen

Beispiel:
lemma assumes *rev:"rev xs = ys"* **shows** *"rev ys = xs"*

proof*(induct xs)*

liefert unbeweisbare Teilziele, da nur über *xs* in Konklusion induziert wird

Lemma mit Prämissen und Konklusion

Vorsicht! Induktionsregeln o.ä. benötigen eine oder mehrere Prämissen, um die Regel anwenden zu können
in dieser Schreibform: angeben, welche Prämissen für Regel nötig sind
Syntax: vor **proof** hinter Schlüsselwort **using** Namen der benötigten Prämissen

Beispiel:

```
lemma assumes rev:"rev xs = ys" shows "rev ys = xs"
using rev
proof(induct xs)
```

Induktion über `xs` in Prämisse und Konklusion, damit Teilziele beweisbar

Allquantoren in zu zeigenden Aussagen

Wie gehe ich mit Parametern um?
Beispiel:
lemma assumes `"¬(∃x. ¬P x)"` **shows** `"∀x. P x"`
proof

nach Anwendung von **proof** (implizit `allI`) subgoal: `⋀x. P x`
doch **show** `"⋀x. P x"` funktioniert nicht!

Allquantoren in zu zeigenden Aussagen

Wie gehe ich mit Parametern um?
Beispiel:
lemma assumes `"¬(∃x. ¬P x)"` **shows** `"∀x. P x"`
proof

nach Anwendung von **proof** (implizit `allI`) subgoal: `⋀x. P x`
doch **show** `"⋀x. P x"` funktioniert nicht!

Parameter müssen zu Beginn des Beweises “gefixt” werden:
`⋀a b c. P` wird zu **fix** `a b c` und **show** `P`

Allquantoren in zu zeigenden Aussagen

Wie gehe ich mit Parametern um?
Beispiel:
lemma assumes *"¬(∃x. ¬P x)"* **shows** *"∀x. P x"*
proof

nach Anwendung von **proof** (implizit *allI*) subgoal: *⋀x. P x*
doch **show** *"⋀x. P x"* funktioniert nicht!

Parameter müssen zu Beginn des Beweises “gefixt” werden:
⋀a b c. P wird zu **fix** *a b c* und **show** *P*
Unser Beispiel:
lemma assumes *"¬(∃x. ¬P x)"* **shows** *"∀x. P x"*
proof
  fix *x*
  show *"P x"* ...
qed

Repräsentanten

Um aus Existenzquantoren Repräsentanten zu erhalten, **obtain**
Syntax: **obtain** *Repräsentant* **where** *Aussage Beweis*
Beispiel:

```
lemma "∃x. P x ⟹ ¬ (∀x. ¬ P x)"
proof -
  assume ex: "∃x. P x"
  from ex obtain z where "P z" by blast
  ...
```

Repräsentanten

Um aus Existenzquantoren Repräsentanten zu erhalten, **obtain**
Syntax: **obtain** *Repräsentant* **where** *Aussage Beweis*
Beispiel:

```
lemma "∃x. P x ⟹ ¬ (∀x. ¬ P x)"
proof -
  assume ex: "∃x. P x"
  from ex obtain z where "P z" by blast
  ...
```

- Repräsentantenname kann beliebig gewählt sein
  (nicht vorhandene Variable!)
- Aussage nach **where** kann beliebigen Namen erhalten
- auch aus Allquantor **obtain** möglich (da Typen in Isabelle immer nichtleer)

obtain-Beweise

Was genau passiert bei den **obtain**-Beweisen?
Isabelle baut aus Zeile **obtain** *x* **where** *"P x"*
Beweisziel $(\bigwedge x.\ P\ x \Longrightarrow thesis) \Longrightarrow thesis$ ($thesis$ beliebige Aussage)

obtain-Beweise

Was genau passiert bei den **obtain**-Beweisen?
Isabelle baut aus Zeile **obtain** *x* **where** *"P x"*
Beweisziel *(⋀x. P x ⟹ thesis) ⟹ thesis* (*thesis* beliebige Aussage)
Also (mittels geeigneter Annahmen) linke Seite der
"P x ⟹ thesis"–Metaimplikation zeigen für beliebige *x*

obtain-Beweise

Was genau passiert bei den **obtain**-Beweisen?
Isabelle baut aus Zeile **obtain** *x* **where** *"P x"*
Beweisziel *(⋀x. P x ⟹ thesis) ⟹ thesis* (*thesis* beliebige Aussage)
Also (mittels geeigneter Annahmen) linke Seite der
"P x ⟹ thesis"–Metaimplikation zeigen für beliebige *x*

Trotzdem Keine Panik!

- Wenn nach **where** exakt Aussage einer Existenzaussage, Beweis einfach mittel *blast*
- auch sonst meist Beweis mittels *blast*, *auto* oder *fastsimp*
- manchmal evtl. noch Fallunterscheidung nötig (z.B. Variable ist Tupel, Bestimmen der Tupeleinträge)