Rechnerübung zu Theorembeweiser und ihre Anwendungen

Prof. Dr.-Ing. Gregor Snelting
Dipl.-Inf. Univ. Daniel Wasserrab

Lehrstuhl Programmierparadigmen
IPD Snelting
Universität Karlsruhe (TH)

Teil IX

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

Aufbau eines einfachen Beispiels

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

allgemeiner Aufbau:

- Lemmadefinition wie gewohnt
- 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**

Aufbau eines einfachen Beispiels

lemma *"A ⟶ A"*
proof(*rule impI*)
  assume *"A"*
  from *‘A‘* **show** *"A"* **by**(*rule a*)
qed

allgemeiner Aufbau:

- Lemmadefinition wie gewohnt
- 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 das ganze Beweisziel
Zwischenaussage nach Schlüsselwort **have**, danach evtl. Name,
danach Aussage, gefolgt von Beweis
dieser Beweis mittels **apply**-Skript oder Isar-**proof**

Prämissen eines Beweises einer Zwischenaussage bzw. des Beweisziels

- aus Annahmen (Aussagen nach **assume**)
- aus bereits bewiesenen Zwischenaussagen (Aussagen nach **have**)

werden vor der Beweisaussage nach Schlüsselwort **from** aufgesammelt
mittels Name oder Aussage in schrägen Hochkommata `` ``
diese beiden Zitationsarten können beliebig gemischt werden

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 das ganze Beweisziel
Zwischenaussage nach Schlüsselwort **have**, danach evtl. Name,
danach Aussage, gefolgt von Beweis
dieser Beweis mittels **apply**-Skript oder Isar-**proof**

Prämissen eines Beweises einer Zwischenaussage bzw. des Beweisziels

- aus Annahmen (Aussagen nach **assume**)
- aus bereits bewiesenen Zwischenaussagen (Aussagen nach **have**)

werden vor der Beweisaussage nach Schlüsselwort **from** aufgesammelt
mittels Name oder Aussage in schrägen Hochkommata ` `
diese beiden Zitationsarten können beliebig gemischt werden

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 -(erule 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

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 -(erule 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 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**
- **from** *a b this* ≡ **with** *a b*
- **from** *this* **show** ≡ **thus**
- **from** *this* **have** ≡ **hence**

Strukturieren des Beweises

aktuell gezeigte 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**
- **from** *a b this* ≡ **with** *a b*
- **from** *this* **show** ≡ **thus**
- **from** *this* **have** ≡ **hence**

proof ohne Regel

wenn nach **proof** keine Methode spezifiziert, wird (falls möglich) sinnvolle erste (Introduktions- oder Eliminations-) Regel angewandt
kann zu unlösbaren subgoals führen, z.B. bei Disjunktion in Konklusion

Beispiel:

```
lemma "⟦A ∨ B; A ⟶ Q⟧ ⟹ (B ⟶ Q) ⟶ Q"
proof                — verwendet implizit (rule impI)
  assume "A ∨ B" and "A ⟶ Q" and "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"
    from `B` `B ⟶ Q` show ?thesis by simp
  qed
qed
```

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

proof ohne Regel

wenn nach **proof** keine Methode spezifiziert, wird (falls möglich) sinnvolle erste (Introduktions- oder Eliminations-) Regel angewandt
kann zu unlösbaren subgoals führen, z.B. bei Disjunktion in Konklusion

Beispiel:

lemma *"⟦A ∨ B; A ⟶ Q⟧ ⟹ (B ⟶ Q) ⟶ Q"*
proof — verwendet implizit *(rule impI)*
  assume *"A ∨ B"* **and** *"A ⟶ Q"* **and** *"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"*
    from *`B`* *`B ⟶ Q`* **show** *?thesis* **by** *simp*
  qed
qed

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

Methoden für proof

proof entweder explizit übergebene Introduktionsregel 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

proof entweder explizit übergebene Introduktionsregel 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 subgoals
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
```

Methoden für proof

Fallunterscheidung und andere Methoden erzeugen mehrere subgoals
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

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

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 **assume**s 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

Beispiel:

```
lemma assumes rev:"rev xs = ys" shows "rev ys = xs"

proof(induct xs)
```

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 subgoals, 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 subgoals beweisbar