

Rechnerübung zu Theorembeweiser und ihre Anwendungen

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

Lehrstuhl Programmierparadigmen
IPD Snelting
Universität Karlsruhe (TH)

Teil XI

Strukturierte Beweise mittels Isar: Induktion

Regelinduktion

ein schon bekanntes Beispiel:

```
inductive palin :: "string  $\Rightarrow$  bool"
```

```
where
```

```
  OneElem: "palin [c]"
```

```
  | TwoElem: "palin [c,c]"
```

```
  | HdLastRec: "palin s  $\Longrightarrow$  palin (c#s@[c])"
```

liefert Induktionsregel `palin.induct` automatisch:

```
[[palin x;  $\bigwedge$ c. P [c];  $\bigwedge$ c. P [c, c];
```

```
 $\bigwedge$ s c. [[palin s; P s]  $\Longrightarrow$  P (c # s @ [c])]]
```

```
 $\Longrightarrow$  P x
```

Regelinduktion

Beweis nach bisher bekanntem Muster:

```
lemma "[[palin xs; (length xs) mod 2 = 0]]  $\implies$ 
  take ((length xs) div 2) xs = rev (drop ((length xs) div 2) xs)"
proof(induct rule:palin.induct)
  fix c assume "length [c] mod 2 = 0"
  hence False by simp
  thus "take (length [c] div 2) [c] = rev (drop (length [c] div 2) [c])" by simp
next
  fix c assume "length [c,c] mod 2 = 0"
  thus "take (length [c,c] div 2) [c,c] =
    rev (drop (length [c,c] div 2) [c,c])" by simp
next
  fix s c assume "palin s" and "length (c#s@[c]) mod 2 = 0"
  and IH:"length s mod 2 = 0  $\implies$ 
    take (length s div 2) s = rev (drop (length s div 2) s)"
  from `length (c#s@[c]) mod 2 = 0` have "length s mod 2 = 0" by simp
  from IH[OF this] have "take (length s div 2) s =
    rev (drop (length s div 2) s)" .
  thus "take (length (c#s@[c]) div 2) (c#s@[c]) =
    rev (drop (length (c#s@[c]) div 2) (c#s@[c]))" by simp
qed
```

Regelinduktion

für Isar zu unaussagekräftig!

nur schwer festzustellen, im Beweis welcher Regel man sich befindet

Lösung: `case`!

- anstatt `fix` und `assumes` schreibt man `case` und den entsprechenden Regelnamen, gefolgt von den zu fixenden Variablen, alles in Klammern
Annahmen (was bisher nach `assume` stand) damit bekannt, können sofort in Hochkommata (`` ``) zitiert werden
wenn Annahme benannt werden soll, Schlüsselwort `note`, danach Name, danach `=`, danach in Hochkommata Aussage
Nachteil: Annahmen nicht mehr direkt sichtbar
- Beweisziel jedes aktuellen Regelfalls in Variable `?case` kann als mittels `show ?case` verwendet werden

für Isar zu unaussagekräftig!

nur schwer festzustellen, im Beweis welcher Regel man sich befindet

Lösung: **case**!

- anstatt **fix** und **assumes** schreibt man **case** und den entsprechenden Regelnamen, gefolgt von den zu fixenden Variablen, alles in Klammern
Annahmen (was bisher nach **assume** stand) damit bekannt, können sofort in Hochkommata (`` ``) zitiert werden
wenn Annahme benannt werden soll, Schlüsselwort **note**, danach Name, danach `=`, danach in Hochkommata Aussage
Nachteil: Annahmen nicht mehr direkt sichtbar
- Beweisziel jedes aktuellen Regelfalls in Variable `?case` kann als mittels **show** `?case` verwendet werden

Regelinduktion

gleiches Beispiel in neuer Syntax:

```
lemma "[[palin xs; (length xs) mod 2 = 0]] ==>
  take ((length xs) div 2) xs = rev (drop ((length xs) div 2) xs)"
proof(induct rule:palin.induct)
  case (OneElem c)
  note length = `length [c] mod 2 = 0` — expliziter Name
  from length have False by simp
  thus ?case by simp
next
  case (TwoElem c)
  from `length [c,c] mod 2 = 0` — Aussage zitiert
  show ?case by simp
next
  case (HdLastRec s c)
  note IH = `length s mod 2 = 0 ==>
    take (length s div 2) s = rev (drop (length s div 2) s)`
  from `length (c#s@[c]) mod 2 = 0` have "length s mod 2 = 0" by simp
  from IH[OF this]
  have "take (length s div 2) s = rev (drop (length s div 2) s)" .
  thus ?case by simp
qed
```

Regelinduktion

Zugriff auf alle Prämissen eines **case**:
mittels *prems* oder Name des aktuellen **case**

deshalb Problem, wenn in **case** *bla* entsprechende Regel mit Namen *bla* verwendet werden soll

Lösung: Name der Funktion/induktiven Prädikat vornangesetzt mit `.`

Beispiel:

```
lemma "palin s  $\implies$  palin (rev s)"
```

```
proof(induct rule:palin.induct)
```

```
  case (OneElem c) show ?case by simp(rule palin.OneElem)
next
```

```
  case (TwoElem c) show ?case by simp(rule palin.TwoElem)
next
```

```
  case (HdLastRec c s) thus ?case by simp(rule palin.HdLastRec)
qed
```


Regelinduktion

Zugriff auf alle Prämissen eines **case**:
mittels *prems* oder Name des aktuellen **case**

deshalb Problem, wenn in **case** *bla* entsprechende Regel mit Namen *bla* verwendet werden soll

Lösung: Name der Funktion/induktiven Prädikat vornangesetzt mit `.`

Beispiel:

```
lemma "palin s  $\implies$  palin (rev s)"
```

```
proof(induct rule:palin.induct)
```

```
  case (OneElem c) show ?case by simp(rule palin.OneElem)  
next
```

```
  case (TwoElem c) show ?case by simp(rule palin.TwoElem)  
next
```

```
  case (HdLastRec c s) thus ?case by simp(rule palin.HdLastRec)  
qed
```

Induktion über `fun`

noch ein bekanntes Beispiel:

```
fun sep :: "'a  $\Rightarrow$  'a list  $\Rightarrow$  'a list"  
where sep_Rec: "sep a (x#y#zs) = x#a#sep a (y#zs)"  
  | sep_Base: "sep a xs = xs"
```

generiert automatisch Induktionsregel `sep.induct`

```

$$\llbracket \bigwedge a\ x\ y\ zs. ?P\ a\ (y\ \#\ zs) \implies ?P\ a\ (x\ \#\ y\ \#\ zs); \bigwedge a. ?P\ a\ [];$$

$$\llbracket \bigwedge a\ v. ?P\ a\ [v] \rrbracket \implies ?P\ ?a0.0\ ?a1.0$$

```

Vorsicht: zwei Regeln für `sep` angegeben, es gibt aber 3 Induktionsregeln!
voriger Ansatz mit Benennung der Fälle kann nicht funktionieren

Jedoch trotzdem `case` möglich durch Nummerierung der Fälle
angegebene Fälle bekommen nach Reihenfolge Nummer
falls für Induktionsregel ein angegebener Fall n auf mehrere aufgeteilt wird,
erhalten diese Fallnummern " n_1 ", " n_2 " etc.

Befehl `print_cases` innerhalb des Induktionsbeweises: Übersicht der Fälle

Induktion über `fun`

noch ein bekanntes Beispiel:

```
fun sep :: "'a  $\Rightarrow$  'a list  $\Rightarrow$  'a list"  
where sep_Rec: "sep a (x#y#zs) = x#a#sep a (y#zs)"  
  | sep_Base: "sep a xs = xs"
```

generiert automatisch Induktionsregel `sep.induct`

$$\llbracket \bigwedge a\ x\ y\ zs. ?P\ a\ (y\ \#\ zs) \implies ?P\ a\ (x\ \#\ y\ \#\ zs); \bigwedge a. ?P\ a\ []; \bigwedge a\ v. ?P\ a\ [v] \rrbracket \implies ?P\ ?a0.0\ ?a1.0$$

Vorsicht: zwei Regeln für `sep` angegeben, es gibt aber **3** Induktionsregeln!
voriger Ansatz mit Benennung der Fälle kann nicht funktionieren

Jedoch trotzdem `case` möglich durch Nummerierung der Fälle
angegebene Fälle bekommen nach Reihenfolge Nummer
falls für Induktionsregel ein angegebener Fall n auf mehrere aufgeteilt wird,
erhalten diese Fallnummern " n_1 ", " n_2 " etc.

Befehl `print_cases` innerhalb des Induktionsbeweises: Übersicht der Fälle

Induktion über `fun`

noch ein bekanntes Beispiel:

```
fun sep :: "'a  $\Rightarrow$  'a list  $\Rightarrow$  'a list"  
where sep_Rec: "sep a (x#y#zs) = x#a#sep a (y#zs)"  
  | sep_Base: "sep a xs = xs"
```

generiert automatisch Induktionsregel `sep.induct`

$$\llbracket \bigwedge a\ x\ y\ zs. ?P\ a\ (y\ \#\ zs) \implies ?P\ a\ (x\ \#\ y\ \#\ zs); \bigwedge a. ?P\ a\ []; \bigwedge a\ v. ?P\ a\ [v] \rrbracket \implies ?P\ ?a0.0\ ?a1.0$$

Vorsicht: zwei Regeln für `sep` angegeben, es gibt aber **3** Induktionsregeln!
voriger Ansatz mit Benennung der Fälle kann nicht funktionieren

Jedoch trotzdem **case** möglich durch Nummerierung der Fälle
angegebene Fälle bekommen nach Reihenfolge Nummer
falls für Induktionsregel ein angegebener Fall n auf mehrere aufgeteilt wird,
erhalten diese Fallnummern " n_1 ", " n_2 " etc.

Befehl `print_cases` innerhalb des Induktionsbeweises: Übersicht der Fälle

Induktion über `fun`

Beispiel für Beweis mit Fallnummern

`sep_Base` in Induktionsregel auf 2 Fälle aufgeteilt worden

```
lemma "sep a xs = []  $\implies$  xs = []"
```

```
proof(induct a xs rule:sep.induct)
```

```
  case (1 a x y zs)
```

```
  from `sep a (x#y#zs) = []` have False by simp
```

```
  thus ?case by simp
```

```
next
```

```
  case ("2_1" a)
```

```
  show ?case by simp
```

```
next
```

```
  case ("2_2" a v)
```

```
  from `sep a [v] = []` have False by simp
```

```
  thus "[v] = []" by simp
```

```
qed
```