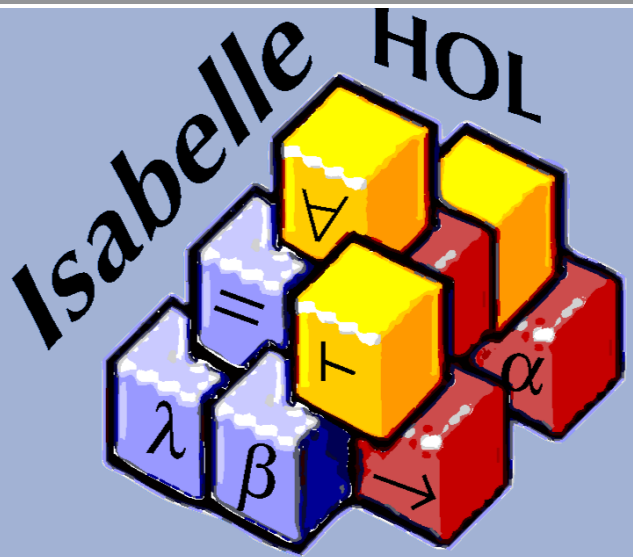
Karlsruher Institut für Technologie

Theorembeweiserpraktikum

Anwendungen in der Sprachtechnologie

LEHRSTUHL PROGRAMMIERPARADIGMEN



Karlsruher Institut für Technologie

Teil XIII

Mehr Strukturierte Beweise mittels Isar: Induktion

Regelinduktion

ein schon bekanntes Beispiel:

inductive *palin :: "string ⇒ bool"*
where

```
  OneElem: "palin [c]"
  | TwoElem: "palin [c,c]"
  | HdLastRec: "palin s ⟹ palin (c#s@[c])"
```

liefert Induktionsregel *palin.induct* automatisch:

```
⟦palin x; ⋀c. P [c]; ⋀c. P [c, c];
⋀s c. ⟦palin s; P s⟧ ⟹ P (c # s @ [c])⟧
⟹ P x
```

Regelinduktion

Beweis nach bisher bekanntem Muster:

```
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)
  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 ⟹
    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

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 benamt 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 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 Name des aktuellen **case**
Die *n*te Prämisse bekommt man mittels **case**-Name(*n*)

Regelinduktion

Zugriff auf alle Prämissen eines **case** mittels Name des aktuellen **case**
Die *n*te Prämisse bekommt man mittels **case**-Name(*n*)

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 ⟹ 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 ⇒ 'a list ⇒ '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`

```
⟦⋀a x y zs. ?P a (y # zs) ⟹ ?P a (x # y # zs); ⋀a. ?P a []; ⋀a v. ?P a [v]⟧ ⟹ ?P ?a0.0 ?a1.0
```

Induktion über fun

noch ein bekanntes Beispiel:

```
fun sep :: "'a ⇒ 'a list ⇒ '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*

```
⟦⋀a x y zs. ?P a (y # zs) ⟹ ?P a (x # y # zs); ⋀a. ?P a []; ⋀a v. ?P a [v]⟧ ⟹ ?P ?a0.0 ?a1.0
```

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

Induktion über fun

noch ein bekanntes Beispiel:

```
fun sep :: "'a ⇒ 'a list ⇒ '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`

```
⟦⋀a x y zs. ?P a (y # zs) ⟹ ?P a (x # y # zs); ⋀a. ?P a []; ⋀a v. ?P a [v]⟧ ⟹ ?P ?a0.0 ?a1.0
```

Vorsicht: zwei Regeln für `sep` angegeben, es gibt aber 3 Induktionsregeln!
voriger Ansatz mit Benamung 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 = [] ⟹ 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
```

Karlsruher Institut für Technologie

Teil XIV

Und noch mehr Struktur

qed mit Taktik

- In Induktions- oder Fallunterscheidungsbeweisen oft die einzelnen Fälle unterschiedlich schwer
- Während manche komplett strukturierten Beweis benötigen, reicht für andere einfache Taktik
- Man kann alle Fälle, die mit der gleichen Taktik einfach zu beweisen sind, zusammenziehen
- einfach entsprechende Fälle im **proof**-Body komplett weglassen, dafür nach **qed** Taktik angeben

qed mit Taktik

Karlsruher Institut für Technologie

Beispiel von vorher:

```
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
```

qed mit Taktik

Beispiel von vorher:

```
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 (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(simp)
```

qed mit Taktik

Karlsruher Institut für Technologie

Beispiel von vorher:

```
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 (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(simp_all)
```

Vorsicht: Bei mehreren weggelassenen Fällen muss auch die Taktik mehrere Fälle abdecken!
also z.B. *simp_all* statt *simp*, *auto* statt *fastsimp*, etc. (oder mit *+* arbeiten)

Aufsammeln von Zwischenzielen

- Isabelle bietet Möglichkeit, bewiesene Zwischenziele aufzusammeln
  Vorsicht! nur **proof**-lokal!
- Neues Schlüsselwort **moreover**
- **moreover** nach einer Aussage + Beweis sammelt diese mit auf
- Aussagen in Variable *calculation* aufgesammelt
  also bei **moreover**: *calculation* := *calculation* + *this*
- Danach kann beliebig weiterbewiesen werden, d.h. auch Aussagen, die nicht mit aufgesammelt werden sollen
- um für Beweis alle aufgesammelten Fakten (und gerade bewiesenes Fakt!) zu verwenden, neues Schlüsselwort **ultimately**
  entspricht also **with** *calculation*
- Vorsicht! Nach **ultimately** dürfen keine weiteren Fakten zitiert werden
  Falls weitere Fakten benötigt werden, mittels **using** nach(!) **have** bzw. **show**, aber vor Beweis

Aufsammeln von Zwischenzielen

Einfaches Beispiel:

```
lemma assumes "A ∧ B" and "B ⟶ C" and "D" shows "C ∧ A ∧ D"
proof -
  from `A ∧ B` have "A" by -(erule conjE)
  moreover
  from `A ∧ B` have "B" by -(erule conjE)
  with `B ⟶ C` have "C" by(rule mp)
  ultimately show ?thesis using `D` by -(rule conjI,assumption)+
qed
```