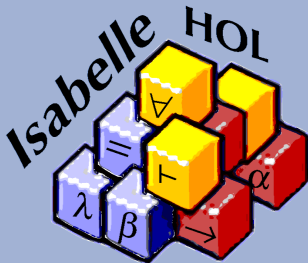


# Theorembeweiserpraktikum

## Anwendungen in der Sprachtechnologie

<http://pp.info.uni-karlsruhe.de/lehre/SS2010/tba/>  
Daniel Wasserrab

IPD Snelting, Lehrstuhl Programmierparadigmen



# Teil X

## Induktive Prädikate und Mengen

Induktive Prädikate nicht rekursiv über Datentypen definiert, sondern über ein **Regelwerk**, bestehend aus

- einer oder mehreren Basisregeln und
- einer oder mehreren induktiven Regeln, wobei Prädikat in Prämissen mit “kleineren” Parametern vorkommt (evtl. auch mehrfach)

Das Prädikat gilt für bestimmte Parameter, wenn es durch Anwendung der Basis- und induktiven Regeln konstruiert werden kann

# Syntax und Beispiel

Schlüsselwort: **inductive**, Syntax wie **primrec** oder **fun**

Die geraden Zahlen als induktives Prädikat:

**inductive** *even* :: "*nat*  $\Rightarrow$  *bool*"

**where** "*even* 0"

| "*even*  $n \Rightarrow$  *even* ( $n + 2$ )"

# Syntax und Beispiel

Schlüsselwort: **inductive**, Syntax wie **primrec** oder **fun**

Die geraden Zahlen als induktives Prädikat:

```
inductive even :: "nat  $\Rightarrow$  bool"  
  where "even 0"  
  | "even n  $\Rightarrow$  even (n + 2)"
```

Welche Eigenschaft über Strings beschreibt folgendes Prädikat?

```
inductive foo :: "string  $\Rightarrow$  bool"  
  where "foo [c]"  
  | "foo [c,c]"  
  | "foo s  $\Rightarrow$  foo (c#s@[c])"
```

# Syntax und Beispiel

Schlüsselwort: **inductive**, Syntax wie **primrec** oder **fun**

Die geraden Zahlen als induktives Prädikat:

```
inductive even :: "nat  $\Rightarrow$  bool"  
  where "even 0"  
        | "even n  $\Rightarrow$  even (n + 2)"
```

Welche Eigenschaft über Strings beschreibt folgendes Prädikat?

```
inductive foo :: "string  $\Rightarrow$  bool"  
  where "foo [c]"  
        | "foo [c,c]"  
        | "foo s  $\Rightarrow$  foo (c#s@[c])"
```

Der Parameterstring ist ein Palindrom!

# Introduktionsregeln

jeder Regel kann einzeln ein Name gegeben werden:

```
inductive palin :: "string  $\Rightarrow$  bool"  
  where OneElem: "palin [c]"  
        | TwoElem: "palin [c,c]"  
        | HdLastRec: "palin s  $\implies$  palin (c#s@[c])"
```

jeder Regel kann einzeln ein Name gegeben werden:

```
inductive palin :: "string  $\Rightarrow$  bool"  
  where OneElem: "palin [c]"  
        | TwoElem: "palin [c,c]"  
        | HdLastRec: "palin s  $\Rightarrow$  palin (c#s@[c])"
```

Diese Regeln zusammengefasst als *palin.intros*  
(allgemein *Prädikatname.intros*)

sieht wie folgt aus:

```
palin [?c]  
palin [?c, ?c]  
palin ?s  $\Rightarrow$  palin (?c # ?s @ [?c])
```

meist jedoch die einzelnen Regelnamen verwendet



# Eliminationsregel

Prädikat aus Regeln aufgebaut, deswegen “Fallunterscheidung” möglich,  
aus welcher Regelanwendung entstanden  
Argumentation über Regelaufbau: *Regelinversion*  
entsprechende Regel *Prädikatname.cases*, wie Eliminationsregel verwendet

Prädikat aus Regeln aufgebaut, deswegen “Fallunterscheidung” möglich,  
aus welcher Regelanwendung entstanden

Argumentation über Regelaufbau: *Regelinversion*

entsprechende Regel *Prädikatname.cases*, wie Eliminationsregel verwendet

Beispiel *palin.cases*:

$$\llbracket \text{palin } ?a; \bigwedge c. ?a = [c] \implies ?P; \bigwedge c. ?a = [c, c] \implies ?P; \\ \bigwedge s c. \llbracket ?a = c \# s @ [c]; \text{palin } s \rrbracket \implies ?P \rrbracket \implies ?P$$

Prädikat aus Regeln aufgebaut, deswegen “Fallunterscheidung” möglich,  
aus welcher Regelanwendung entstanden

Argumentation über Regelaufbau: *Regelinversion*

entsprechende Regel *Prädikatname.cases*, wie Eliminationsregel verwendet

Beispiel *palin.cases*:

$$\llbracket \text{palin } ?a; \bigwedge c. ?a = [c] \implies ?P; \bigwedge c. ?a = [c, c] \implies ?P; \\ \bigwedge s c. \llbracket ?a = c \# s @ [c]; \text{palin } s \rrbracket \implies ?P \rrbracket \implies ?P$$

**lemma** "*palin s  $\implies$  hd s = last s*"

**apply**(*erule palin.cases*)

liefert subgoals

1.  $\bigwedge c. s = [c] \implies \text{hd } s = \text{last } s$
2.  $\bigwedge c. s = [c, c] \implies \text{hd } s = \text{last } s$
3.  $\bigwedge sa c. \llbracket s = c \# sa @ [c]; \text{palin } sa \rrbracket \implies \text{hd } s = \text{last } s$

oftmals Fallunterscheidung nicht genug, brauchen Induktionshypothese für Prädikate in der Prämisse einer Regel

dafür Induktionsregel *Prädikatname.induct*

Beispiel *palin.induct*:

$\llbracket \text{palin } ?x; \wedge c. ?P [c]; \wedge c. ?P [c, c];$

$\wedge s c. \llbracket \text{palin } s; ?P s \rrbracket \implies ?P (c \# s @ [c]) \rrbracket \implies ?P ?x$

oftmals Fallunterscheidung nicht genug, brauchen Induktionshypothese für Prädikate in der Prämisse einer Regel

dafür Induktionsregel *Prädikatname.induct*

Beispiel *palin.induct*:

$$\llbracket \text{palin } ?x; \wedge c. ?P [c]; \wedge c. ?P [c, c]; \wedge s c. \llbracket \text{palin } s; ?P s \rrbracket \implies ?P (c \# s @ [c]) \rrbracket \implies ?P ?x$$

**lemma** "*palin s  $\implies$  hd s = last s*"

**apply** (*induct rule:palin.induct*)

liefert subgoals

1.  $\wedge c. \text{hd } [c] = \text{last } [c]$
2.  $\wedge c. \text{hd } [c, c] = \text{last } [c, c]$
3.  $\wedge s c. \llbracket \text{palin } s; \text{hd } s = \text{last } s \rrbracket \implies \text{hd } (c \# s @ [c]) = \text{last } (c \# s @ [c])$

Wechselseitigkeit auch bei induktiven Definitionen funktioniert analog zu wechselseitiger Rekursion

Beispiel:

```
inductive even :: "nat  $\Rightarrow$  bool"  
  and odd :: "nat  $\Rightarrow$  bool"  
  where "even 0"  
  | "odd n  $\implies$  even (Suc n)"  
  | "even n  $\implies$  odd (Suc n)"
```

generiert Regeln *eval.cases*, *odd.cases*, *even\_odd.induct* and *even\_odd.inducts*

erstere Induktionsregel benötigt  $\wedge$ -Verknüpfung von *even* und *odd* in Konklusion, zweite liefert *zwei* Regeln für zwei **and**-verbundene Lemmas

statt induktiver Prädikate auch **induktive Mengen**

Schlüsselwort **inductive\_set**

Signatur entsprechend  $'a \text{ set}$  statt  $'a \Rightarrow \text{bool}$

Manchmal fixe Parameter nötig, beeinflussen induktive Definition nicht  
müssen nach **for** mit Namen und Signatur angegeben werden

Beispiel: Domain eines Prädikats

```
inductive DomainP :: ('a  $\Rightarrow$  'b  $\Rightarrow$  bool)  $\Rightarrow$  'a  $\Rightarrow$  bool
```

```
  for r :: "'a  $\Rightarrow$  'b  $\Rightarrow$  bool"
```

```
  where DomainP: "r a b  $\implies$  DomainP r a"
```