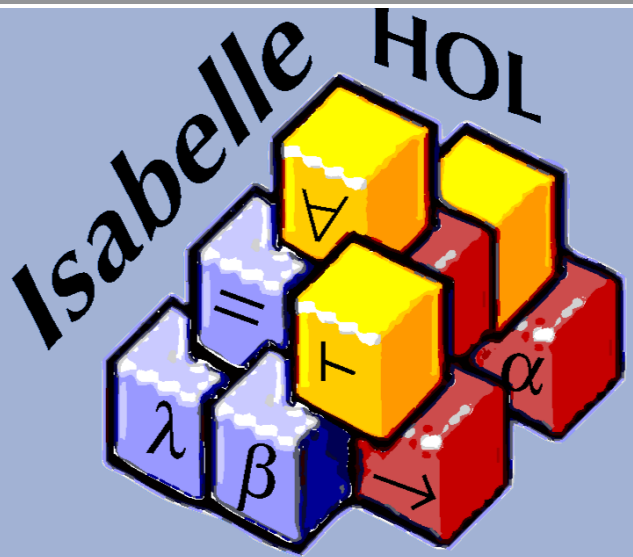
Karlsruher Institut für Technologie

Theorembeweiserpraktikum

Anwendungen in der Sprachtechnologie

LEHRSTUHL PROGRAMMIERPARADIGMEN



Karlsruher Institut für Technologie

Teil XI

Induktive Prädikate und Mengen

Syntax und Beispiel

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

Beispiel 1: Die geraden Zahlen als induktives Prädikat

```
inductive even :: "nat ⇒ bool"
  where "even 0"
  | "even n ⟹ even (n + 2)"
```

Syntax und Beispiel

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

Beispiel 1: Die geraden Zahlen als induktives Prädikat

```
inductive even :: "nat ⇒ bool"
  where "even 0"
  | "even n ⟹ even (n + 2)"
```

Beispiel 2:

Welche Eigenschaft über Strings beschreibt folgendes Prädikat?

```
inductive foo :: "string ⇒ bool"
  where "foo [c]"
  | "foo [c,c]"
  | "foo s ⟹ foo (c#s@[c])"
```

Syntax und Beispiel

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

Beispiel 1: Die geraden Zahlen als induktives Prädikat

```
inductive even :: "nat ⇒ bool"
  where "even 0"
  | "even n ⟹ even (n + 2)"
```

Beispiel 2:

Welche Eigenschaft über Strings beschreibt folgendes Prädikat?

```
inductive foo :: "string ⇒ bool"
  where "foo [c]"
  | "foo [c,c]"
  | "foo s ⟹ foo (c#s@[c])"
```

Der Parameterstring ist ein Palindrom!

Induktive Prädikate

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 (endliche) Anwendung der Basis- und induktiven Regeln konstruiert werden kann

Introduktionsregeln

Jeder Regel kann einzeln ein Name gegeben werden:

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

Introduktionsregeln

Jeder Regel kann einzeln ein Name gegeben werden:

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

Diese Regeln zusammengefasst als `palin.intros`
(allgemein `Prädikatname.intros`)
sieht wie folgt aus:

```
palin [?c]
palin [?c, ?c]
palin ?s ⟹ palin (?c # ?s @ [?c])
```

meist jedoch einzelne Regelnamen verwenden

Eliminationsregel

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

Eliminationsregel

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`**:**

```
⟦palin ?a; ⋀c. ?a = [c] ⟹ ?P; ⋀c. ?a = [c, c] ⟹ ?P;
⋀s c. ⟦?a = c # s @ [c]; palin s⟧ ⟹ ?P⟧ ⟹ ?P
```

Eliminationsregel

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*:

```
⟦palin ?a; ⋀c. ?a = [c] ⟹ ?P; ⋀c. ?a = [c, c] ⟹ ?P;
⋀s c. ⟦?a = c # s @ [c]; palin s⟧ ⟹ ?P⟧ ⟹ ?P
```

lemma *"palin s ⟹ hd s = last s"*
apply(*erule palin.cases*)

liefert 3 Teilziele:

```
1. ⋀c. s = [c] ⟹ hd s = last s
2. ⋀c. s = [c, c] ⟹ hd s = last s
3. ⋀sa c. ⟦s = c # sa @ [c]; palin sa⟧ ⟹ hd s = last s
```

Induktionsregel

oftmals Fallunterscheidung nicht genug, brauchen Induktionshypothese
für Prädikate in der Prämisse einer Regel
dafür Induktionsregel *Prädikatname.induct*

Beispiel *palin.induct*:

$$\llbracket palin\ ?x;\ \bigwedge c.\ ?P\ [c];\ \bigwedge c.\ ?P\ [c, c];$$
$$\bigwedge s\ c.\ \llbracket palin\ s;\ ?P\ s \rrbracket \Longrightarrow ?P\ (c\ \#\ s\ @\ [c]) \rrbracket \Longrightarrow ?P\ ?x$$

Induktionsregel

oftmals Fallunterscheidung nicht genug, brauchen Induktionshypothese
für Prädikate in der Prämisse einer Regel
dafür Induktionsregel *Prädikatname.induct*

Beispiel *palin.induct*:

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

lemma *"palin s ⟹ hd s = last s"*
apply(*induct rule:palin.induct*)

liefert Teilziele

```
1. ⋀c. hd [c] = last [c]
2. ⋀c. hd [c, c] = last [c, c]
3. ⋀s c. ⟦palin s; hd s = last s⟧
   ⟹ hd (c # s @ [c]) = last (c # s @ [c])
```

Induktiv wechselseitig

Wechselseitigkeit auch bei induktiven Definitionen funktioniert analog zu wechselseitiger Rekursion

Beispiel:

```
inductive even :: "nat ⇒ bool"
      and odd :: "nat ⇒ bool"
  where "even 0"
  | "odd n ⟹ even (Suc n)"
  | "even n ⟹ 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, zweitere liefert *zwei* Regeln für zwei **and**-verbundene Lemmas

Ausblick

statt induktiver Präkate auch induktive Mengen
Schlüsselwort **inductive_set**
Signatur entsprechend `'a set` statt `'a ⇒ bool`

Manchmal fixe Parameter nötig

- bleiben bei Induktionsschritt konstant
- müssen nach **for** mit Namen und Signatur angegeben werden

Beispiel: Reflexive, transitive Hülle

```
inductive rtc :: ('a ⇒ 'a ⇒ bool) ⇒ 'a ⇒ 'a ⇒ bool
  for r :: "'a ⇒ 'a ⇒ bool"
   where refl: "rtc r a a"
   | trans: "⟦ rtc r a b; r b c ⟧ ⟹ rtc r a c"

rtc.induct: ⟦rtc ?r ?x ?y; ⋀a. ?P a a;
    ⋀a b c. ⟦rtc ?r a b; ?P a b; ?r b c⟧ ⟹ ?P a c⟧
   ⟹?P ?x ?y
```

Ausblick

statt induktiver Präkate auch induktive Mengen
Schlüsselwort **inductive_set**
Signatur entsprechend *'a set* statt *'a ⇒ bool*

Manchmal fixe Parameter nötig

- bleiben bei Induktionsschritt konstant
- müssen nach **for** mit Namen und Signatur angegeben werden

Beispiel: Reflexive, transitive Hülle

```
inductive rtc :: ('a ⇒ 'a ⇒ bool) ⇒ 'a ⇒ 'a ⇒ bool

    where refl: "rtc r a a"
    | trans: "⟦ rtc r a b; r b c ⟧ ⟹ rtc r a c"

rtc.induct: ⟦rtc ?r ?x ?y; ⋀r a. ?P r a a;
     ⋀r a b c. ⟦rtc r a b; ?P r a b; r b c⟧ ⟹ ?P r a c⟧
    ⟹?P ?r ?x ?y
```