Rechnerübung zu Theorembeweiser und ihre Anwendungen

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

Lehrstuhl Programmierparadigmen
IPD Snelting
Universität Karlsruhe (TH)

Teil V

Induktive Prädikate und Mengen

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 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 ⇒ bool"
  where "even 0"
  | "even n ⟹ even (n + 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**

Die geraden Zahlen als induktives Prädikat:

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

Die geraden Zahlen als induktives Prädikat:

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

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 die einzelnen Regelnamen verwendet

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 die einzelnen Regelnamen verwendet

Eliminationsregel

Prädikat aus Regeln aufgebaut, deswegen "Fallunterscheidung" möglich, aus welcher Regelanwendung entstanden
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 subgoals

```
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
```

Eliminationsregel

Prädikat aus Regeln aufgebaut, deswegen "Fallunterscheidung" möglich, aus welcher Regelanwendung entstanden
entsprechende Regel *Prädikatname.cases*, wie Eliminationsregel verwendet

Beispiel *palin.cases*:
$[\![palin\ ?a;\ \bigwedge c.\ ?a = [c] \Longrightarrow ?P;\ \bigwedge c.\ ?a = [c, c] \Longrightarrow ?P;\ \bigwedge s\ c.\ [\![?a = c\ \#\ s\ @\ [c];\ palin\ s]\!] \Longrightarrow ?P]\!] \Longrightarrow ?P$

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

Eliminationsregel

Prädikat aus Regeln aufgebaut, deswegen "Fallunterscheidung" möglich, aus welcher Regelanwendung entstanden
entsprechende Regel *Prädikatname.cases*, wie Eliminationsregel verwendet

Beispiel *palin.cases*:
$[\![palin\ ?a;\ \bigwedge c.\ ?a = [c] \Longrightarrow ?P;\ \bigwedge c.\ ?a = [c, c] \Longrightarrow ?P;\ \bigwedge s\ c.\ [\![?a = c\ \#\ s\ @\ [c];\ palin\ s]\!] \Longrightarrow ?P]\!] \Longrightarrow ?P$

lemma *"palin s ⟹ hd s = last s"*
apply(*erule palin.cases*)
liefert subgoals
1. $\bigwedge c.\ s = [c] \Longrightarrow hd\ s = last\ s$
2. $\bigwedge c.\ s = [c, c] \Longrightarrow hd\ s = last\ s$
3. $\bigwedge sa\ c.\ [\![s = c\ \#\ sa\ @\ [c];\ palin\ sa]\!] \Longrightarrow hd\ s = last\ s$

Induktionsregel

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

dafür Induktionsregel *Prädikatname.induct*

Beispiel *palin.induct*:

$$[\![\mathit{palin}\ ?x;\ \bigwedge c.\ ?P\ [c];\ \bigwedge c.\ ?P\ [c, c];\ \bigwedge s\ c.\ [\![\mathit{palin}\ s;\ ?P\ s]\!] \Longrightarrow ?P\ (c\ \#\ s\ @\ [c])]\!] \Longrightarrow ?P\ ?x$$

lemma *"palin s ⟹ hd s = last s"*

apply(*induct rule:palin.induct*)

liefert subgoals

1. $\bigwedge c.\ hd\ [c] = last\ [c]$
2. $\bigwedge c.\ hd\ [c, c] = last\ [c, c]$
3. $\bigwedge s\ c.\ [\![palin\ s;\ hd\ s = last\ s]\!] \Longrightarrow hd\ (c\ \#\ s\ @\ [c]) = last\ (c\ \#\ s\ @\ [c])$

Induktionsregel

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

dafür Induktionsregel *Prädikatname.induct*

Beispiel *palin.induct*:

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

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

apply(*induct rule:palin.induct*)

liefert subgoals

1. $\bigwedge c.$ *hd [c] = last [c]*
2. $\bigwedge c.$ *hd [c, c] = last [c, c]*
3. $\bigwedge s\ c.\ [\![palin\ s;\ hd\ s = last\ s]\!] \Longrightarrow$ *hd (c # s @ [c]) = last (c # s @ [c])*

Ausblick

statt induktiver Präkate auch induktive Mengen
Schlüsselwort **inductive_set**
Signatur entsprechend *'a set* statt *'a ⇒ 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 ⇒ 'b ⇒ bool) ⇒ 'a ⇒ bool
  for r :: "'a ⇒ 'b ⇒ bool"
  where DomainP: "r a b ⟹ DomainP r a"
```