Karlsruher Institut für Technologie

Theorembeweiserpraktikum

Anwendungen in der Sprachtechnologie

LEHRSTUHL PROGRAMMIERPARADIGMEN

Karlsruher Institut für Technologie

Teil IV

Quantoren in Isabelle/HOL

Quantoren in Isabelle/HOL

Die üblichen zwei Quantoren der Logik:

Existenzquantor: $\exists$ (geschrieben `\<exists>`), Syntax: $\exists\, x.\ P\ x$

Allquantor: $\forall$ (geschrieben `\<forall>`), Syntax: $\forall\, x.\ P\ x$

Quantoren in Isabelle/HOL

Die üblichen zwei Quantoren der Logik:

Existenzquantor: $\exists$ (geschrieben `\<exists>`), Syntax: $\exists\, x.\ P\ x$

Allquantor: $\forall$ (geschrieben `\<forall>`), Syntax: $\forall\, x.\ P\ x$

Gültigkeitsbereich der gebundenen Variablen:
bis zum nächsten ; bzw. $\Longrightarrow$

Quantoren in Isabelle/HOL

Die üblichen zwei Quantoren der Logik:

Existenzquantor: $\exists$ (geschrieben `\<exists>`), Syntax: $\exists\, x.\ P\ x$

Allquantor: $\forall$ (geschrieben `\<forall>`), Syntax: $\forall\, x.\ P\ x$

Gültigkeitsbereich der gebundenen Variablen:
bis zum nächsten ; bzw. $\Longrightarrow$

Beispiele

$\forall\, x.\ P\ x \Longrightarrow Q\ x$ x in Konklusion nicht gebunden durch Allquantor

Quantoren in Isabelle/HOL

Die üblichen zwei Quantoren der Logik:

Existenzquantor: $\exists$ (geschrieben `\<exists>`), Syntax: $\exists\, x.\ P\ x$

Allquantor: $\forall$ (geschrieben `\<forall>`), Syntax: $\forall\, x.\ P\ x$

Gültigkeitsbereich der gebundenen Variablen:
bis zum nächsten ; bzw. $\Longrightarrow$

Beispiele

$\forall\, x.\ P\ x \Longrightarrow Q\ x$ x in Konklusion nicht gebunden durch Allquantor

$P\ y \Longrightarrow \exists\, y.\ P\ y$ y in Prämisse nicht gebunden durch Existenzquantor

Quantoren in Isabelle/HOL

Die üblichen zwei Quantoren der Logik:

Existenzquantor: $\exists$ (geschrieben `\<exists>`), Syntax: $\exists x.\ P\ x$

Allquantor: $\forall$ (geschrieben `\<forall>`), Syntax: $\forall x.\ P\ x$

Gültigkeitsbereich der gebundenen Variablen:
bis zum nächsten ; bzw. $\Longrightarrow$

Beispiele

$\forall x.\ P\ x \Longrightarrow Q\ x$ x in Konklusion nicht gebunden durch Allquantor

$P\ y \Longrightarrow \exists y.\ P\ y$ y in Prämisse nicht gebunden durch Existenzquantor

$[\![\forall x.\ P\ x;\ \exists x.\ Q\ x]\!] \Longrightarrow R$

Zwei verschiedene x in den Annahmen
gleichbedeutend mit $[\![\forall y.\ P\ y;\ \exists z.\ Q\ z]\!] \Longrightarrow R$
(*gebundene Namen sind Schall und Rauch*)

Quantoren in Isabelle/HOL

Die üblichen zwei Quantoren der Logik:

Existenzquantor: $\exists$ (geschrieben `\<exists>`), Syntax: $\exists x.\ P\ x$

Allquantor: $\forall$ (geschrieben `\<forall>`), Syntax: $\forall x.\ P\ x$

Gültigkeitsbereich der gebundenen Variablen:
bis zum nächsten ; bzw. $\Longrightarrow$

Beispiele

$\forall x.\ P\ x \Longrightarrow Q\ x$ x in Konklusion nicht gebunden durch Allquantor

$P\ y \Longrightarrow \exists y.\ P\ y$ y in Prämisse nicht gebunden durch Existenzquantor

$[\![\forall x.\ P\ x;\ \exists x.\ Q\ x]\!] \Longrightarrow R$

Zwei verschiedene x in den Annahmen
gleichbedeutend mit $[\![\forall y.\ P\ y;\ \exists z.\ Q\ z]\!] \Longrightarrow R$
(*gebundene Namen sind Schall und Rauch*)

$\forall x.\ P\ x \longrightarrow Q\ x$ *gleiches* x für P und Q

Wie sagt man es Isabelle...?

Argumentation mit Quantoren erfordert Aussagen über *beliebige* Werte
Nur: Wie weiß Isabelle, dass ein Wert *beliebig* ist?

Wie sagt man es Isabelle...?

Argumentation mit Quantoren erfordert Aussagen über *beliebige* Werte
Nur: Wie weiß Isabelle, dass ein Wert *beliebig* ist?

Lösung: Meta-Logik

Syntax: $\bigwedge x.\ [\![\ldots]\!] \Longrightarrow \ldots$

$\bigwedge$ heisst Meta-Allquantor, Variablen dahinter Parameter

Gültigkeitsbereich der Parameter: ganzes Teilziel

Beispiel: $\bigwedge x\ y.\ [\![\forall y.\ P\ y \longrightarrow Q\ z\ y;\ Q\ x\ y]\!] \Longrightarrow \exists x.\ Q\ x\ y$

entspricht $\bigwedge x\ y.\ [\![\forall y_1.\ P\ y_1 \longrightarrow Q\ z\ y_1;\ Q\ x\ y]\!] \Longrightarrow \exists x_1.\ Q\ x_1\ y$

Wie sagt man es Isabelle...?

Argumentation mit Quantoren erfordert Aussagen über *beliebige* Werte
Nur: Wie weiß Isabelle, dass ein Wert *beliebig* ist?

Lösung: Meta-Logik

Syntax: $\bigwedge x.\ [\![\ldots]\!] \Longrightarrow \ldots$

$\bigwedge$ heisst Meta-Allquantor, Variablen dahinter Parameter

Gültigkeitsbereich der Parameter: ganzes Teilziel

Beispiel: $\bigwedge x\ y.\ [\![\forall y.\ P\ y \longrightarrow Q\ z\ y;\ Q\ x\ y]\!] \Longrightarrow \exists x.\ Q\ x\ y$
entspricht $\bigwedge x\ y.\ [\![\forall y_1.\ P\ y_1 \longrightarrow Q\ z\ y_1;\ Q\ x\ y]\!] \Longrightarrow \exists x_1.\ Q\ x_1\ y$

Auch $\Longrightarrow$ ist Teil der Meta-Logik, entspricht Meta-Implikation
Trennt Annahmen und Konklusion

Wie sagt man es Isabelle...?

Argumentation mit Quantoren erfordert Aussagen über *beliebige* Werte
Nur: Wie weiß Isabelle, dass ein Wert *beliebig* ist?

Lösung: Meta-Logik

Syntax: $\bigwedge x.\ [\![\ldots]\!] \Longrightarrow \ldots$

$\bigwedge$ heisst Meta-Allquantor, Variablen dahinter Parameter

Gültigkeitsbereich der Parameter: ganzes Teilziel

Beispiel: $\bigwedge x\ y.\ [\![\forall y.\ P\ y \longrightarrow Q\ z\ y;\ Q\ x\ y]\!] \Longrightarrow \exists x.\ Q\ x\ y$

entspricht $\bigwedge x\ y.\ [\![\forall y_1.\ P\ y_1 \longrightarrow Q\ z\ y_1;\ Q\ x\ y]\!] \Longrightarrow \exists x_1.\ Q\ x_1\ y$

Auch $\Longrightarrow$ ist Teil der Meta-Logik, entspricht Meta-Implikation
Trennt Annahmen und Konklusion

$\forall$ und $\longrightarrow$ entsprechen nicht $\bigwedge$ und $\Longrightarrow$, die ersten beiden nur in HOL!

Regeln

Jeder Quantor hat Introduktions- und Eliminationsregel:

Regeln

Jeder Quantor hat Introduktions- und Eliminationsregel:

- *allI:* $(\bigwedge x.\ P\ x) \Longrightarrow \forall x.\ P\ x$
  Eine Aussage gilt für beliebige x (Meta-Ebene),
  also gilt sie auch für alle (HOL-Ebene)

Regeln

Jeder Quantor hat Introduktions- und Eliminationsregel:

- *allI*: $(\bigwedge x.\ P\ x) \Longrightarrow \forall x.\ P\ x$
  Eine Aussage gilt für beliebige x (Meta-Ebene),
  also gilt sie auch für alle (HOL-Ebene)

- *allE*: $[\![\forall x.\ P\ x;\ P\ ?x \Longrightarrow R]\!] \Longrightarrow R$
  Eine Aussage gilt für alle x, also folgt die Konklusion auch,
  wenn diese Aussage für irgendeine (selbst wählbare) Variable x gilt
  Vorsicht: x nach Anwendung der Regel beliebige Variable ($?x$)!
  Möglichst gleich spezifizieren duch *erule_tac*

  Beispiel:
  apply *(erule_tac x="z"* **in** *allE)*

Regeln

- *exI*: $P\ ?x \Longrightarrow \exists x.\ P\ x$
  Eine Aussage gilt für eine Variable x, also gibt es ein x, wofür sie gilt
  Vorsicht: x nach Anwendung der Regel beliebige Variable (*?x*)!
  Möglichst gleich spezifizieren duch *rule_tac*

  Beispiel:
  apply *(rule_tac x="z"* **in** *exI)*

Regeln

- exI: $P\ ?x \Longrightarrow \exists x.\ P\ x$
  Eine Aussage gilt für eine Variable x, also gibt es ein x, wofür sie gilt
  Vorsicht: x nach Anwendung der Regel beliebige Variable ($?x$)!
  Möglichst gleich spezifizieren duch $rule\_tac$

  Beispiel:
  apply $(rule\_tac\ x="z"$ **in** $exI)$

- exE: $[\![\exists x.\ P\ x;\ \bigwedge x.\ P\ x \Longrightarrow Q]\!] \Longrightarrow Q$
  Eine Aussage gilt für ein x, also folgt die Konklusion auch,
  wenn diese Aussage für eine beliebige (vorgegebene!) Variable gilt.

Variablen festlegen bei Regelanwendung

Den Regeln `allE` und `exI` gemeinsam:
Nach Anwendung der entsprechenden Methode (also `erule` bzw. `rule`) unspezifizierte Variable (`?x`) im Teilziel
⇒ Meist nicht gewollt, da schlecht Aussagen darüber möglich

Besser: entsprechende Variable gleich festlegen

Variablen festlegen bei Regelanwendung

Den Regeln `allE` und `exI` gemeinsam:
Nach Anwendung der entsprechenden Methode (also `erule` bzw. `rule`) unspezifizierte Variable (`?x`) im Teilziel
⇒ Meist nicht gewollt, da schlecht Aussagen darüber möglich

Besser: entsprechende Variable gleich festlegen

Methode: `rule_tac v1 = t1` **and** ... **and** `vk = tk` **in** `R`

`?v1`, ..., `?vk` freie Variable in der anzuwendenden Regel `R`
(nicht im aktuellen Teilziel!)
analog: `erule_tac`

Variablen festlegen bei Regelanwendung

Den Regeln *allE* und *exI* gemeinsam:
Nach Anwendung der entsprechenden Methode (also *erule* bzw. *rule*) unspezifizierte Variable (*?x*) im Teilziel
⇒ Meist nicht gewollt, da schlecht Aussagen darüber möglich

Besser: entsprechende Variable gleich festlegen

Methode: *rule_tac v1 = t1* **and** ... **and** *vk = tk* **in** *R*

?v1, ..., *?vk* freie Variable in der anzuwendenden Regel *R*
(nicht im aktuellen Teilziel!)
analog: *erule_tac*

also möglichst immer **apply** *(rule_tac x=gewollte Variable* **in** *exI)*
bzw. **apply** *(erule_tac x=gewollte Variable* **in** *allE)*

Teil V

Fallunterscheidung und Definition

Fallunterscheidung

In (klassischen) Beweisen Fallunterscheidung wichtiges Hilfsmittel

Fallunterscheidung

In (klassischen) Beweisen Fallunterscheidung wichtiges Hilfsmittel

In Isabelle: einfach durch die Methode `cases P`
(`P` beliebiges boolesches Prädikat)

Teilt aktuelles Teilziel in 2 neue Teilziele auf:

- erstes mit `P` zusätzlich in den Annahmen
- zweites mit `¬ P` in den Annahmen

Beispiel

apply `(cases "x > y")`

generiert 2 Teilziele mit den (zusätzlichen) Annahmen

- $x > y$
- $\neg\ x > y$

Fallunterscheidung: Beispiel

aktuelles Teilziel: $[\![B;\ C]\!] \Longrightarrow A \wedge B \vee \neg A \wedge C$
so nicht (einfach) lösbar!

Fallunterscheidung: Beispiel

aktuelles Teilziel: $[\![B;\ C]\!] \Longrightarrow A \wedge B \vee \neg A \wedge C$
so nicht (einfach) lösbar!

jedoch nach **apply** $(cases\ A)$ neue Teilziele

Fallunterscheidung: Beispiel

aktuelles Teilziel: $[\![B;\ C]\!] \Longrightarrow A \wedge B \vee \neg A \wedge C$
so nicht (einfach) lösbar!

jedoch nach **apply** *(cases A)* neue Teilziele

1. $[\![B;\ C;\ A]\!] \Longrightarrow A \wedge B \vee \neg A \wedge C$
2. $[\![B;\ C;\ \neg A]\!] \Longrightarrow A \wedge B \vee \neg A \wedge C$

Fallunterscheidung: Beispiel

aktuelles Teilziel: $[\![B;\ C]\!] \Longrightarrow A \wedge B \vee \neg A \wedge C$
so nicht (einfach) lösbar!

jedoch nach **apply** `(cases A)` neue Teilziele

1. $[\![B;\ C;\ A]\!] \Longrightarrow A \wedge B \vee \neg A \wedge C$
2. $[\![B;\ C;\ \neg A]\!] \Longrightarrow A \wedge B \vee \neg A \wedge C$

jetzt einfach mit Introduktionsregeln für $\wedge$ und $\vee$ zu lösen

definition

Ermöglicht direkte Definition von nichtrekursiven Funktionen
verbindet Deklaration mit Definition der Funktion

definition

Ermöglicht direkte Definition von nichtrekursiven Funktionen
verbindet Deklaration mit Definition der Funktion

Beispiel: Funktion `nand` (= "not and")

definition $nand$ :: "$bool \Rightarrow bool \Rightarrow bool$"
where "$nand\ A\ B \equiv \neg\ (A \wedge B)$"

definition

Ermöglicht direkte Definition von nichtrekursiven Funktionen
verbindet Deklaration mit Definition der Funktion

Beispiel: Funktion `nand` **(= "not and")**

definition *nand* :: "*bool* $\Rightarrow$ *bool* $\Rightarrow$ *bool*"
where "*nand A B* $\equiv$ $\neg$ (*A* $\wedge$ *B*)"

automatisch generierte Simplifikationsregel: *nand_def*

Allgemein: *Funktionsname_def*
$\Rightarrow$ kann dem Simplifier übergeben werden (siehe später)

definition – Syntaxdefinition

`nand` ist binärer Operator

definition – Syntaxdefinition

`nand` ist binärer Operator
$\Longrightarrow$ Infixoperator bietet sich an

definition – Syntaxdefinition

`nand` ist binärer Operator
$\Longrightarrow$ Infixoperator bietet sich an

Syntaxdefinition (Infix-Notation)

Schreibe (**infixl** *"Operatorsymbol" n*) an die Deklarationszeile, wobei

- **infixl** für linksgebundenen Infixoperator steht,
  infixr für rechtsgebundene
- *Operatorsymbol* ein beliebig wählbares Symbol für den Operator ist,
- n eine Zahl ist, welche die Präzedenz dieses Operators angibt

definition – Syntaxdefinition

`nand` ist binärer Operator
$\Longrightarrow$ Infixoperator bietet sich an

Syntaxdefinition (Infix-Notation)

Schreibe (**infixl** *"Operatorsymbol" n*) an die Deklarationszeile, wobei

- **infixl** für linksgebundenen Infixoperator steht,
  infixr für rechtsgebundene
- *Operatorsymbol* ein beliebig wählbares Symbol für den Operator ist,
- *n* eine Zahl ist, welche die Präzedenz dieses Operators angibt

Beispiel: Operator `nand`

definition *nand* :: *"bool* $\Rightarrow$ *bool* $\Rightarrow$ *bool"* **(infixl** *"$\bowtie$" 36)*
where *"nand A B* $\equiv$ $\neg$ *(A* $\wedge$ *B)"*

Jetzt: $A \bowtie B \bowtie C$ gleichbedeutend mit *nand (nand A B) C*

Teil VI

Simplifikation

Simplifikation

- Simplifikationsregeln: Gleichungen
- entsprechende Taktik: `simp`
  - besitzt Pool an Termersetzungsregeln
  - prüft für jede solche Regel, ob Term mit linker Seite einer Gleichung unifizierbar
  - falls ja, ersetzen mit entsprechend unifizierter rechten Seite
- genauer: Termersetzung (weil Ausdruck rechts in der Gleichung nicht notwendigerweise einfacher)

Simplifikation

- Simplifikationsregeln: Gleichungen
- entsprechende Taktik: `simp`
  - besitzt Pool an Termersetzungsregeln
  - prüft für jede solche Regel, ob Term mit linker Seite einer Gleichung unifizierbar
  - falls ja, ersetzen mit entsprechend unifizierter rechten Seite
- genauer: Termersetzung (weil Ausdruck rechts in der Gleichung nicht notwendigerweise einfacher)

Beispiel:

aktuelles Subgoal: $C \Longrightarrow P\ (\mathit{if\ False\ then\ A\ else\ B} \longrightarrow D)$

`simp` wendet folgende Termersetzungsregel an:

$\mathit{HOL.if\_False}:\ (\mathit{if\ False\ then\ ?x\ else\ ?y}) = \mathit{?y}$

Simplifikation

- Simplifikationsregeln: Gleichungen
- entsprechende Taktik: `simp`
  - besitzt Pool an Termersetzungsregeln
  - prüft für jede solche Regel, ob Term mit linker Seite einer Gleichung unifizierbar
  - falls ja, ersetzen mit entsprechend unifizierter rechten Seite
- genauer: Termersetzung (weil Ausdruck rechts in der Gleichung nicht notwendigerweise einfacher)

Beispiel:

aktuelles Subgoal: $C \Longrightarrow P\ (\mathit{if}\ \mathit{False}\ \mathit{then}\ A\ \mathit{else}\ B \longrightarrow D)$

`simp` wendet folgende Termersetzungsregel an:

$\mathit{HOL.if\_False}$: $(\mathit{if}\ \mathit{False}\ \mathit{then}\ ?x\ \mathit{else}\ ?y) = ?y$

Resultat: $C \Longrightarrow P\ (B \longrightarrow D)$

Simplifikation

Auch bedingte Ersetzungsregeln sind möglich, also in der Form

$[\![\ldots]\!] \Longrightarrow \ \ldots \ = \ \ldots$

Dazu: Prämissen der Regel aus aktuellen Annahmen *via Simplifikation* herleitbar

Simplifikation

Auch bedingte Ersetzungsregeln sind möglich, also in der Form

$[\![\ldots]\!] \Longrightarrow \ \ldots \ = \ldots$

Dazu: Prämissen der Regel aus aktuellen Annahmen *via Simplifikation* herleitbar

Simplifier modifizieren:

- selbstgeschriebene Simplifikationslemmas zu Taktik hinzufügen:
  apply `(simp add:` Regel_1 Regel_2 `...)`
- nur bestimmte Ersetzungsregeln verwenden:
  apply `(simp only:` Regel_1 Regel_2 `...)`
- Ersetzungsregeln aus dem Standardpool von `simp` entfernen:
  apply `(simp del:` Regel_1 Regel_2 `...)`

Simplifikation

Auch möglich: Ersetzungsregeln in den Standardpool von `simp` einfügen
Zwei Varianten:

- Zusatz *[simp]* hinter Lemmanamen
  Beispiel: **lemma** *bla [simp]: "A = True* $\Longrightarrow$ *A* $\wedge$ *B = B"*
- mittels **declare** *[simp]*
  Beispiel: **declare** *[simp]: foo bar*
  Analog: mittels **declare** *[simp del]* Ersetzungsregeln
  aus Standardpool entfernen

Simplifikation

Auch möglich: Ersetzungsregeln in den Standardpool von `simp` einfügen
Zwei Varianten:

- Zusatz *[simp]* hinter Lemmanamen
  Beispiel: **lemma** *bla [simp]: "A = True* $\Longrightarrow$ *A* $\wedge$ *B = B"*
- mittels **declare** *[simp]*
  Beispiel: **declare** *[simp]: foo bar*
  Analog: mittels **declare** *[simp del]* Ersetzungsregeln
  aus Standardpool entfernen

Vorsicht!

- Nur Regeln zu Standardpool hinzufügen, dessen rechte Seite einfacher als linke Seite!
- Sicherstellen, dass `simp` durch neue Regeln nicht in Endlosschleifen hängenbleibt!

Einschub: Arbeiten mit apply-Skripten

- nach **apply** mehrere Regelanwendungen bzw. Taktiken hintereinander
  Beispiel: **apply** *(simp, rule foo, auto)*

- reguläre Ausdrücke:
  - *+* hinter Regel bzw. Taktik wendet diese ein- oder mehrfach an
    Bsp: **apply** *(assumption)+*
  - *?* analog für null- oder einfache Anwendung
    Bsp: **apply** *(assumption)?*
  - *|* zwischen zwei Regeln bzw. Taktiken wendet die erste an,
    bei Nichtgelingen zweite; Bsp: **apply** *(assumption|arith)*

- **by** ersetzt letzte **apply**-Regelanwendung bzw. Taktik und **done**
  alle *assumption* Anwendungen nach **apply** automatisch angewandt
  Bsp: statt **apply** *(rule foo, assumption, assumption)* **done**
  nur **by** *(rule foo)*